

# DHSI

DIGITAL HUMANITIES SUMMER INSTITUTE

---

# NLP Coding Libraries and Network Analysis for Text Corpora. With a Bonus Track: #GraphPoem Live Interactive Coding

**Chris Tanasescu**

---

This package is intended for the personal, educational use of DHSI attendees. Portions appear here with consideration of fair use and fair dealing guidelines.

© DHSI 2023



Social Sciences and Humanities  
Research Council of Canada

Conseil de recherches en  
sciences humaines du Canada

# Welcome to DHSI 2023!

Thank you for joining the DHSI community!

In this coursepack, you will find essential workshop materials prefaced by some useful general information about DHSI 2023.

Given our community's focus on things computational, it will be a surprise to no one that we might expect additional information and materials online for some of the workshops—which will be made available to you where applicable—or that the most current version of all DHSI-related information may be found on our website at [dhsi.org](http://dhsi.org). Do check in there first if you need any information that's not in this coursepack.

Please also note that materials in DHSI's online workshop folders could be updated at any point. We recommend checking back on any DHSI online workshop folder(s) that have been shared with you in case additional materials are added as DHSI approaches and takes place.

And please don't hesitate to be in touch with us at [institut@uvic.ca](mailto:institut@uvic.ca) or via Twitter at [@AlyssaA\\_DHSI](https://twitter.com/AlyssaA_DHSI) or [@DHInstitute](https://twitter.com/DHInstitute) if we can be of any help.

We hope you enjoy your time with us!



## Statement of Ethics & Inclusion

---

Please review the DHSI Statement of Ethics & Inclusion available here:

<https://dhsi.org/statement-of-ethics-inclusion/>

DHSI is dedicated to offering a safe, respectful, friendly, and collegial environment for the benefit of everyone who attends and for the advancement of the interests that bring us together. There is no place at DHSI for harassment or intimidation of any kind.

By registering for DHSI, you have agreed to comply with these commitments.

## Virtual Sessions

---

Your registration in DHSI 2023 also includes access to the virtual [institute lecture](#) sessions. Access details for these talks will be shared as DHSI approaches.

Due to the high volume of attendees, please ensure your DHSI registration name or DHSI preferred name and your Zoom name match so that we know to let you into the virtual sessions.

## DHSI Materials

---

DHSI materials (ex. videos, documents, etc.) are intended for registrant use only. By registering, you have agreed that you will not circulate any DHSI content. If someone asks you for the materials, please invite them to complete the registration form to request access or contact us at [institut@uvic.ca](mailto:institut@uvic.ca).

## Auditor and participant registration

---

If you registered to **audit** any workshops, note that auditor involvement is intended to be fully self-directed without active participation in the workshop. The auditor option offers more flexibility regarding pace and time with the workshop content. Your registration as an auditor will include access to some asynchronous workshop materials only and does not include access to live workshop sessions and/or individual/group instruction or consultation. Please direct any questions about DHSI workshop auditing to [institut@uvic.ca](mailto:institut@uvic.ca).

If you registered as a **participant** in any workshops, your registration includes access to asynchronous content + active participation in live workshop session(s). The workshop instructor(s) will contact you about the date(s), time(s), and platform(s) of the live workshop session(s).

If you are unsure whether you registered as an auditor or participant, please check your registration confirmation email. Further questions can be directed to [institut@uvic.ca](mailto:institut@uvic.ca).

## Schedule

---

The at-a-glance schedule of DHSI 2023 courses, workshops, institute lectures and aligned conferences & events can be found here: <https://dhsi.org/timetable/>

All times are listed in North American **Pacific Time Zone**.

For those who registered as participants in any workshops, live sessions for online workshops are not currently listed on the above-referenced schedule. **Instructors will be in touch with registered participants directly about the exact date(s) and time(s) of their live workshop session(s).**

## Acknowledgements

---

We would like to thank our partners and sponsors (including the Social Sciences and Humanities Research Council), workshop instructors, aligned conference & event organizers, institute lecturers, local facilitators, and beyond for making this possible.

## Further information

---

General DHSI 2023 information: <https://dhsi.org/program/>

Full course listings (in-person): <https://dhsi.org/on-campus-courses/>

Full workshop listings (online): <https://dhsi.org/online-workshops/>

Aligned conferences & events (in-person): <https://dhsi.org/on-campus-aligned-conferences-events/>

Aligned conferences & events (online): <https://dhsi.org/online-aligned-conferences-events/>

Institute lectures: <https://dhsi.org/institute-lectures/>

Frequently asked questions: <https://dhsi.org/faq/>

Any questions not addressed in the above pages? Please email us at [institut@uvic.ca](mailto:institut@uvic.ca)!

# NLP Coding Libraries and Network Analysis for Corpora. With a Bonus Track: #GraphPoem Live Interactive Coding

DHSI 2022

Instructor: Chris Tanasescu (Margento)

The 3-day workshop offers a quick and effective intro to natural language processing (NLP) and textual corpus network visualization and analysis.

We will be doing coding in Python and learning how to use (and compare) certain relevant libraries such as Scikit-learn, SpaCy, Gensim, FastText, and NetworkX. We will apply those packages in computationally analyzing texts and textual corpora, representing the corpora as networks, and thus finding out unexpected if not amazing things about the texts they contain.

The knowledge and skills acquired—alongside our in-class applications—will be useful in education and research in NLP, automated text and corpus analysis, network science and graph theory applications, computational literary analysis and criticism, computational linguistics, and vector space (and topic) modeling for the humanities.

On the fourth day, everybody will have the opportunity to participate in the #GraphPoem event that will involve some of the Python scripts developed during the workshop. We will run those and other scripts live (on JupyterHub) on ready-made and individually/collaboratively assembled and expanded corpora, thus feeding into a hypermedia performance involving a Twitter bot and a cross-artform livestream.

## Coursepak

### 1. What is NLP?

<https://www.gyansetu.in/what-is-natural-language-processing/>

### 2. Scikit-learn, the best machine learning library?

Buitinck, L., et al. 2013. "API design for machine learning software: experiences from the Scikit-learn project"

<https://arxiv.org/pdf/1309.0238.pdf>

### 3. FastText, the Facebook-trained word embeddings modeled on "subword" data (character n-grams)

Bojanowski, P., et al. 2017. "Enriching Word Vectors with Subword Information"

<https://arxiv.org/pdf/1607.04606.pdf>

Joulin, A., et al. 2016. "Bag of Tricks for Efficient Text Classification"

<https://arxiv.org/pdf/1607.01759.pdf>

### 4. What is SpaCy and what do we need it for?

<https://spacy.io/usage/spacy-101>

### 5. Gensim, "topic modeling for humans"...

Gensim, "generate similar"?

<https://radimrehurek.com/gensim/intro.html>

Radim Řehůřek talk: "Word2vec & friends" (7.1.2015)

<https://www.youtube.com/watch?v=wTp3P2UnTfQ>

## 6. NetworkX

The Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks,  
<https://networkx.org/documentation/stable/index.html>

Hagberg, A.A., et al. 2008. "Exploring Network Structure, Dynamics, and Function using NetworkX"  
[http://conference.scipy.org/proceedings/SciPy2008/paper\\_2/full\\_text.pdf](http://conference.scipy.org/proceedings/SciPy2008/paper_2/full_text.pdf)

## **Additional Readings**

### 1. Fundamentals

1.0. Sebastian, F. 2002. "Machine Learning in Automatic Text Categorization." <https://dl.acm.org/doi/10.1145/505282.505283>

1.1. Jurafsky, D. & Martin, J.H. 2021. "Text Processing." Slides presentation format.  
[https://web.stanford.edu/~jurafsky/slp3/slides/2\\_TextProc\\_Mar\\_25\\_2021.pdf](https://web.stanford.edu/~jurafsky/slp3/slides/2_TextProc_Mar_25_2021.pdf)

1.2. Jurafsky, D. & Martin, J.H. 2020. "Vector Semantic and Embeddings."  
<https://web.stanford.edu/~jurafsky/slp3/6.pdf>

### 2. Applications (selected)

2.0. Greene, E., et al. 2010. "Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation."  
<https://www.aclweb.org/anthology/D10-1051.pdf>

2.1. Ganguly, D., et al. 2014. "Automatic Prediction of Text Aesthetics and Interestingness." <https://www.aclweb.org/anthology/C14-1086.pdf>

2.2. Lou, A., et al. 2015. "Multilabel Subject-based Classification of Poetry." <https://bit.ly/3dmdL93>

2.3. Tanasescu, C., et al. 2018. "Metaphor Detection by Deep Learning and the Place of Poetic Metaphor in Digital Humanities."  
<https://aaai.org/ocs/index.php/FLAIRS/FLAIRS18/paper/view/17704/16866>

2.4. Sondheim, A., et al. 2019. "Our Shared World of Language: Reflections on 'US' Poets Foreign Poets." Blog post,  
<https://www.asymptotejournal.com/blog/2019/05/30/our-shared-world-of-language-reflections-on-us-poets-foreign-poets/>

2.5. Chatsiou, K. & Jankin Mikhaylov, S. 2020. "Deep Learning for Political Science." <https://arxiv.org/pdf/2005.06540.pdf>

### 3. New Directions

3.0. Jurafsky, D. & Martin, J.H. 2020. "Neural Networks and Neural Language Models." <https://web.stanford.edu/~jurafsky/slp3/7.pdf>

3.1. Linzen, T. 2020. "How Can We Accelerate Progress Towards Human-like

Linguistic Generalization?" <https://arxiv.org/pdf/2005.00955.pdf>

3.2. Ribeiro, M.T., et al. 2020. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." <https://arxiv.org/pdf/2005.04118.pdf>

3.3. Kurfalı M. & Östling, R. 2021. "Let's be explicit about that: Distant supervision for implicit discourse relation classification via connective prediction." <https://aclanthology.org/2021.unimplicit-1.1/>.

3.4. Wang, S., et al. 2021. "Phrase-BERT: Improved Phrase Embeddings from BERT with an Application to Corpus Exploration." <https://paperswithcode.com/paper/phrase-bert-improved-phrase-embeddings-from>

3.5. Gweon, H. & Schonlau, M. 2022. "Automated classification for open-ended questions with BERT." <https://arxiv.org/abs/2209.06178>



# Machine Learning in Automated Text Categorization

FABRIZIO SEBASTIANI

*Consiglio Nazionale delle Ricerche, Italy*

The automated categorization (or classification) of texts into predefined categories has witnessed a booming interest in the last 10 years, due to the increased availability of documents in digital form and the ensuing need to organize them. In the research community the dominant approach to this problem is based on machine learning techniques: a general inductive process automatically builds a classifier by learning, from a set of preclassified documents, the characteristics of the categories. The advantages of this approach over the knowledge engineering approach (consisting in the manual definition of a classifier by domain experts) are a very good effectiveness, considerable savings in terms of expert labor power, and straightforward portability to different domains. This survey discusses the main approaches to text categorization that fall within the machine learning paradigm. We will discuss in detail issues pertaining to three different problems, namely, document representation, classifier construction, and classifier evaluation.

Categories and Subject Descriptors: H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*; I.2.6 [**Artificial Intelligence**]: Learning—*Induction*

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Machine learning, text categorization, text classification

## 1. INTRODUCTION

In the last 10 years content-based document management tasks (collectively known as *information retrieval*—IR) have gained a prominent status in the information systems field, due to the increased availability of documents in digital form and the ensuing need to access them in flexible ways. *Text categorization* (TC—*a.k.a. text classification, or topic spotting*), the activity of labeling natural language

texts with thematic categories from a predefined set, is one such task. TC dates back to the early '60s, but only in the early '90s did it become a major subfield of the information systems discipline, thanks to increased applicative interest and to the availability of more powerful hardware. TC is now being applied in many contexts, ranging from document indexing based on a controlled vocabulary, to document filtering, automated metadata generation, word sense disambiguation, population of

---

Author's address: Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Via G. Moruzzi 1, 56124 Pisa, Italy; e-mail: [fabrizio@iei.pi.cnr.it](mailto:fabrizio@iei.pi.cnr.it).

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

©2002 ACM 0360-0300/02/0300-0001 \$5.00

hierarchical catalogues of Web resources, and in general any application requiring document organization or selective and adaptive document dispatching.

Until the late '80s the most popular approach to TC, at least in the “operational” (i.e., real-world applications) community, was a *knowledge engineering* (KE) one, consisting in manually defining a set of rules encoding expert knowledge on how to classify documents under the given categories. In the '90s this approach has increasingly lost popularity (especially in the research community) in favor of the *machine learning* (ML) paradigm, according to which a general inductive process automatically builds an automatic text classifier by learning, from a set of preclassified documents, the characteristics of the categories of interest. The advantages of this approach are an accuracy comparable to that achieved by human experts, and a considerable savings in terms of expert labor power, since no intervention from either knowledge engineers or domain experts is needed for the construction of the classifier or for its porting to a different set of categories. It is the ML approach to TC that this paper concentrates on.

Current-day TC is thus a discipline at the crossroads of ML and IR, and as such it shares a number of characteristics with other tasks such as *information / knowledge extraction from texts* and *text mining* [Knight 1999; Pazienza 1997]. There is still considerable debate on where the exact border between these disciplines lies, and the terminology is still evolving. “Text mining” is increasingly being used to denote all the tasks that, by analyzing large quantities of text and detecting usage patterns, try to extract probably useful (although only probably correct) information. According to this view, TC is an instance of text mining. TC enjoys quite a rich literature now, but this is still fairly scattered.<sup>1</sup> Although two international journals have devoted special issues to

this topic [Joachims and Sebastiani 2002; Lewis and Hayes 1994], there are no systematic treatments of the subject: there are neither textbooks nor journals entirely devoted to TC yet, and Manning and Schütze [1999, Chapter 16] is the only chapter-length treatment of the subject. As a note, we should warn the reader that the term “automatic text classification” has sometimes been used in the literature to mean things quite different from the ones discussed here. Aside from (i) the automatic assignment of documents to a predefined set of categories, which is the main topic of this paper, the term has also been used to mean (ii) the automatic identification of such a set of categories (e.g., Borko and Bernick [1963]), or (iii) the automatic identification of such a set of categories *and* the grouping of documents under them (e.g., Merkl [1998]), a task usually called *text clustering*, or (iv) any activity of placing text items into groups, a task that has thus both TC and text clustering as particular instances [Manning and Schütze 1999].

This paper is organized as follows. In Section 2 we formally define TC and its various subcases, and in Section 3 we review its most important applications. Section 4 describes the main ideas underlying the ML approach to classification. Our discussion of *text* classification starts in Section 5 by introducing *text indexing*, that is, the transformation of textual documents into a form that can be interpreted by a classifier-building algorithm and by the classifier eventually built by it. Section 6 tackles the inductive construction of a text classifier from a “training” set of preclassified documents. Section 7 discusses the evaluation of text classifiers. Section 8 concludes, discussing open issues and possible avenues of further research for TC.

## 2. TEXT CATEGORIZATION

### 2.1. A Definition of Text Categorization

Text categorization is the task of assigning a Boolean value to each pair  $\langle d_j, c_i \rangle \in \mathcal{D} \times \mathcal{C}$ , where  $\mathcal{D}$  is a domain of documents and

<sup>1</sup> A fully searchable bibliography on TC created and maintained by this author is available at <http://liinwww.ira.uka.de/bibliography/Ai/automated.text.categorization.html>.

$\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  is a set of predefined *categories*. A value of  $T$  assigned to  $\langle d_j, c_i \rangle$  indicates a decision to file  $d_j$  under  $c_i$ , while a value of  $F$  indicates a decision not to file  $d_j$  under  $c_i$ . More formally, the task is to approximate the unknown *target function*  $\check{\Phi} : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$  (that describes how documents ought to be classified) by means of a function  $\Phi : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$  called the *classifier* (aka *rule*, or *hypothesis*, or *model*) such that  $\check{\Phi}$  and  $\Phi$  “coincide as much as possible.” How to precisely define and measure this coincidence (called *effectiveness*) will be discussed in Section 7.1. From now on we will assume that:

- The categories are just symbolic labels, and no additional knowledge (of a procedural or declarative nature) of their meaning is available.
- No *exogenous* knowledge (i.e., data provided for classification purposes by an external source) is available; therefore, classification must be accomplished on the basis of *endogenous* knowledge only (i.e., knowledge extracted from the documents). In particular, this means that *metadata* such as, for example, publication date, document type, publication source, etc., is not assumed to be available.

The TC methods we will discuss are thus completely general, and do not depend on the availability of special-purpose resources that might be unavailable or costly to develop. Of course, these assumptions need not be verified in operational settings, where it is legitimate to use any source of information that might be available or deemed worth developing [Díaz Esteban et al. 1998; Junker and Abecker 1997]. Relying only on endogenous knowledge means classifying a document based solely on its semantics, and given that the semantics of a document is a *subjective* notion, it follows that the membership of a document in a category (pretty much as the relevance of a document to an information need in IR [Saracevic 1975]) cannot be decided deterministically. This is exemplified by the

phenomenon of *inter-indexer inconsistency* [Cleverdon 1984]: when two human experts decide whether to classify document  $d_j$  under category  $c_i$ , they may disagree, and this in fact happens with relatively high frequency. A news article on Clinton attending Dizzy Gillespie’s funeral could be filed under Politics, or under Jazz, or under both, or even under neither, depending on the subjective judgment of the expert.

## 2.2. Single-Label Versus Multilabel Text Categorization

Different constraints may be enforced on the TC task, depending on the application. For instance we might need that, for a given integer  $k$ , exactly  $k$  (or  $\leq k$ , or  $\geq k$ ) elements of  $\mathcal{C}$  be assigned to each  $d_j \in \mathcal{D}$ . The case in which exactly one category must be assigned to each  $d_j \in \mathcal{D}$  is often called the *single-label* (a.k.a. *nonoverlapping categories*) case, while the case in which any number of categories from 0 to  $|\mathcal{C}|$  may be assigned to the same  $d_j \in \mathcal{D}$  is dubbed the *multilabel* (aka *overlapping categories*) case. A special case of single-label TC is *binary* TC, in which each  $d_j \in \mathcal{D}$  must be assigned either to category  $c_i$  or to its complement  $\bar{c}_i$ .

From a theoretical point of view, the binary case (hence, the single-label case, too) is more general than the multilabel, since an algorithm for binary classification can also be used for multilabel classification: one needs only transform the problem of multilabel classification under  $\{c_1, \dots, c_{|\mathcal{C}|}\}$  into  $|\mathcal{C}|$  independent problems of binary classification under  $\{c_i, \bar{c}_i\}$ , for  $i = 1, \dots, |\mathcal{C}|$ . However, this requires that categories be stochastically independent of each other, that is, for any  $c', c''$ , the value of  $\check{\Phi}(d_j, c')$  does not depend on the value of  $\check{\Phi}(d_j, c'')$  and vice versa; this is usually assumed to be the case (applications in which this is not the case are discussed in Section 3.5). The converse is not true: an algorithm for multilabel classification cannot be used for either binary or single-label classification. In fact, given a document  $d_j$  to classify, (i) the classifier might attribute  $k > 1$  categories to  $d_j$ , and it might not be obvious how to

choose a “most appropriate” category from them; or (ii) the classifier might attribute to  $d_j$  no category at all, and it might not be obvious how to choose a “least inappropriate” category from  $\mathcal{C}$ .

In the rest of the paper, unless explicitly mentioned, we will deal with the binary case. There are various reasons for this:

- The binary case is important in itself because important TC applications, including filtering (see Section 3.3), consist of binary classification problems (e.g., deciding whether  $d_j$  is about Jazz or not). In TC, most binary classification problems feature unevenly populated categories (e.g., much fewer documents are about Jazz than are not) and unevenly characterized categories (e.g., what is about Jazz can be characterized much better than what is not).
- Solving the binary case also means solving the multilabel case, which is also representative of important TC applications, including automated indexing for Boolean systems (see Section 3.1).
- Most of the TC literature is couched in terms of the binary case.
- Most techniques for binary classification are just special cases of existing techniques for the single-label case, and are simpler to illustrate than these latter.

This ultimately means that we will view classification under  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  as consisting of  $|\mathcal{C}|$  independent problems of classifying the documents in  $\mathcal{D}$  under a given category  $c_i$ , for  $i = 1, \dots, |\mathcal{C}|$ . A *classifier for  $c_i$*  is then a function  $\Phi_i : \mathcal{D} \rightarrow \{T, F\}$  that approximates an unknown target function  $\Phi_i : \mathcal{D} \rightarrow \{T, F\}$ .

### 2.3. Category-Pivoted Versus Document-Pivoted Text Categorization

There are two different ways of using a text classifier. Given  $d_j \in \mathcal{D}$ , we might want to find all the  $c_i \in \mathcal{C}$  under which it should be filed (*document-pivoted categorization*—DPC); alternatively, given  $c_i \in \mathcal{C}$ , we might want to find all the  $d_j \in \mathcal{D}$  that should be filed under it (*category-pivoted*

*categorization*—CPC). This distinction is more pragmatic than conceptual, but is important since the sets  $\mathcal{C}$  and  $\mathcal{D}$  might not be available in their entirety right from the start. It is also relevant to the choice of the classifier-building method, as some of these methods (see Section 6.9) allow the construction of classifiers with a definite slant toward one or the other style.

DPC is thus suitable when documents become available at different moments in time, e.g., in filtering e-mail. CPC is instead suitable when (i) a new category  $c_{|\mathcal{C}|+1}$  may be added to an existing set  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  after a number of documents have already been classified under  $\mathcal{C}$ , and (ii) these documents need to be re-considered for classification under  $c_{|\mathcal{C}|+1}$  (e.g., Larkey [1999]). DPC is used more often than CPC, as the former situation is more common than the latter.

Although some specific techniques apply to one style and not to the other (e.g., the proportional thresholding method discussed in Section 6.1 applies only to CPC), this is more the exception than the rule: most of the techniques we will discuss allow the construction of classifiers capable of working in either mode.

### 2.4. “Hard” Categorization Versus Ranking Categorization

While a complete automation of the TC task requires a  $T$  or  $F$  decision for each pair  $\langle d_j, c_i \rangle$ , a partial automation of this process might have different requirements.

For instance, given  $d_j \in \mathcal{D}$  a system might simply *rank* the categories in  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  according to their estimated appropriateness to  $d_j$ , without taking any “hard” decision on any of them. Such a ranked list would be of great help to a human expert in charge of taking the final categorization decision, since she could thus restrict the choice to the category (or categories) at the top of the list, rather than having to examine the entire set. Alternatively, given  $c_i \in \mathcal{C}$  a system might simply rank the documents in  $\mathcal{D}$  according to their estimated appropriateness to  $c_i$ ; symmetrically, for

classification under  $c_i$  a human expert would just examine the top-ranked documents instead of the entire document set. These two modalities are sometimes called *category-ranking TC* and *document-ranking TC* [Yang 1999], respectively, and are the obvious counterparts of DPC and CPC.

Semiautomated, “interactive” classification systems [Larkey and Croft 1996] are useful especially in critical applications in which the effectiveness of a fully automated system may be expected to be significantly lower than that of a human expert. This may be the case when the quality of the training data (see Section 4) is low, or when the training documents cannot be trusted to be a representative sample of the unseen documents that are to come, so that the results of a completely automatic classifier could not be trusted completely.

In the rest of the paper, unless explicitly mentioned, we will deal with “hard” classification; however, many of the algorithms we will discuss naturally lend themselves to ranking TC too (more details on this in Section 6.1).

### 3. APPLICATIONS OF TEXT CATEGORIZATION

TC goes back to Maron’s [1961] seminal work on probabilistic text classification. Since then, it has been used for a number of different applications, of which we here briefly review the most important ones. Note that the borders between the different classes of applications listed here are fuzzy and somehow artificial, and some of these may be considered special cases of others. Other applications we do not explicitly discuss are speech categorization by means of a combination of speech recognition and TC [Myers et al. 2000; Schapire and Singer 2000], multimedia document categorization through the analysis of textual captions [Sable and Hatzivassiloglou 2000], author identification for literary texts of unknown or disputed authorship [Forsyth 1999], language identification for texts of unknown language [Cavnar and Trenkle 1994],

automated identification of text genre [Kessler et al. 1997], and automated essay grading [Larkey 1998].

#### 3.1. Automatic Indexing for Boolean Information Retrieval Systems

The application that has spawned most of the early research in the field [Borko and Bernick 1963; Field 1975; Gray and Harley 1971; Heaps 1973; Maron 1961] is that of automatic document indexing for IR systems relying on a controlled dictionary, the most prominent example of which is Boolean systems. In these latter each document is assigned one or more key words or key phrases describing its content, where these key words and key phrases belong to a finite set called *controlled dictionary*, often consisting of a thematic hierarchical thesaurus (e.g., the NASA thesaurus for the aerospace discipline, or the MESH thesaurus for medicine). Usually, this assignment is done by trained human indexers, and is thus a costly activity.

If the entries in the controlled vocabulary are viewed as categories, text indexing is an instance of TC, and may thus be addressed by the automatic techniques described in this paper. Recalling Section 2.2, note that this application may typically require that  $k_1 \leq x \leq k_2$  key words are assigned to each document, for given  $k_1, k_2$ . Document-pivoted TC is probably the best option, so that new documents may be classified as they become available. Various text classifiers explicitly conceived for document indexing have been described in the literature; see, for example, Fuhr and Knorz [1984], Robertson and Harding [1984], and Tzeras and Hartmann [1993].

Automatic indexing with controlled dictionaries is closely related to *automated metadata generation*. In digital libraries, one is usually interested in tagging documents by metadata that describes them under a variety of aspects (e.g., creation date, document type or format, availability, etc.). Some of this metadata is *thematic*, that is, its role is to describe the semantics of the document by means of

bibliographic codes, key words or key phrases. The generation of this metadata may thus be viewed as a problem of document indexing with controlled dictionary, and thus tackled by means of TC techniques.

### 3.2. Document Organization

Indexing with a controlled vocabulary is an instance of the general problem of document base organization. In general, many other issues pertaining to document organization and filing, be it for purposes of personal organization or structuring of a corporate document base, may be addressed by TC techniques. For instance, at the offices of a newspaper incoming “classified” ads must be, prior to publication, categorized under categories such as Personals, Cars for Sale, Real Estate, etc. Newspapers dealing with a high volume of classified ads would benefit from an automatic system that chooses the most suitable category for a given ad. Other possible applications are the organization of patents into categories for making their search easier [Larkey 1999], the automatic filing of newspaper articles under the appropriate sections (e.g., Politics, Home News, Lifestyles, etc.), or the automatic grouping of conference papers into sessions.

### 3.3. Text Filtering

*Text filtering* is the activity of classifying a stream of incoming documents dispatched in an asynchronous way by an information producer to an information consumer [Belkin and Croft 1992]. A typical case is a newsfeed, where the producer is a news agency and the consumer is a newspaper [Hayes et al. 1990]. In this case, the filtering system should block the delivery of the documents the consumer is likely not interested in (e.g., all news not concerning sports, in the case of a sports newspaper). Filtering can be seen as a case of single-label TC, that is, the classification of incoming documents into two disjoint categories, the relevant and the irrelevant. Additionally,

a filtering system may also further classify the documents deemed relevant to the consumer into thematic categories; in the example above, all articles about sports should be further classified according to which sport they deal with, so as to allow journalists specialized in individual sports to access only documents of prospective interest for them. Similarly, an e-mail filter might be trained to discard “junk” mail [Androutsopoulos et al. 2000; Drucker et al. 1999] and further classify nonjunk mail into topical categories of interest to the user.

A filtering system may be installed at the producer end, in which case it must route the documents to the interested consumers only, or at the consumer end, in which case it must block the delivery of documents deemed uninteresting to the consumer. In the former case, the system builds and updates a “profile” for each consumer [Liddy et al. 1994], while in the latter case (which is the more common, and to which we will refer in the rest of this section) a single profile is needed.

A profile may be initially specified by the user, thereby resembling a standing IR query, and is updated by the system by using feedback information provided (either implicitly or explicitly) by the user on the relevance or nonrelevance of the delivered messages. In the TREC community [Lewis 1995c], this is called *adaptive filtering*, while the case in which no user-specified profile is available is called either *routing* or *batch filtering*, depending on whether documents have to be ranked in decreasing order of estimated relevance or just accepted/rejected. Batch filtering thus coincides with single-label TC under  $|C|=2$  categories; since this latter is a completely general TC task, some authors [Hull 1994; Hull et al. 1996; Schapire et al. 1998; Schütze et al. 1995], somewhat confusingly, use the term “filtering” in place of the more appropriate term “categorization.”

In information science, document filtering has a tradition dating back to the '60s, when, addressed by systems of various degrees of automation and dealing with the multiconsumer case discussed

above, it was called *selective dissemination of information* or *current awareness* (see Korfhage [1997, Chapter 6]). The explosion in the availability of digital information has boosted the importance of such systems, which are nowadays being used in contexts such as the creation of personalized Web newspapers, junk e-mail blocking, and Usenet news selection.

Information filtering by ML techniques is widely discussed in the literature: see Amati and Crestani [1999], Iyer et al. [2000], Kim et al. [2000], Tauritz et al. [2000], and Yu and Lam [1998].

### 3.4. Word Sense Disambiguation

*Word sense disambiguation* (WSD) is the activity of finding, given the occurrence in a text of an ambiguous (i.e., polysemous or homonymous) word, the sense of this particular word occurrence. For instance, bank may have (at least) two different senses in English, as in the Bank of England (a financial institution) or the bank of river Thames (a hydraulic engineering artifact). It is thus a WSD task to decide which of the above senses the occurrence of bank in Last week I borrowed some money from the bank has. WSD is very important for many applications, including natural language processing, and indexing documents by word senses rather than by words for IR purposes. WSD may be seen as a TC task (see Gale et al. [1993]; Escudero et al. [2000]) once we view word occurrence contexts as documents and word senses as categories. Quite obviously, this is a single-label TC case, and one in which document-pivoted TC is usually the right choice.

WSD is just an example of the more general issue of resolving natural language ambiguities, one of the most important problems in computational linguistics. Other examples, which may all be tackled by means of TC techniques along the lines discussed for WSD, are *context-sensitive spelling correction*, *prepositional phrase attachment*, *part of speech tagging*, and *word choice selection* in machine translation; see Roth [1998] for an introduction.

### 3.5. Hierarchical Categorization of Web Pages

TC has recently aroused a lot of interest also for its possible application to automatically classifying Web pages, or sites, under the hierarchical catalogues hosted by popular Internet portals. When Web documents are catalogued in this way, rather than issuing a query to a general-purpose Web search engine a searcher may find it easier to first navigate in the hierarchy of categories and then restrict her search to a particular category of interest.

Classifying Web pages automatically has obvious advantages, since the manual categorization of a large enough subset of the Web is infeasible. Unlike in the previous applications, it is typically the case that each category must be populated by a set of  $k_1 \leq x \leq k_2$  documents. CPC should be chosen so as to allow new categories to be added and obsolete ones to be deleted.

With respect to previously discussed TC applications, automatic Web page categorization has two essential peculiarities:

- (1) *The hypertextual nature of the documents*: Links are a rich source of information, as they may be understood as stating the relevance of the linked page to the linking page. Techniques exploiting this intuition in a TC context have been presented by Attardi et al. [1998], Chakrabarti et al. [1998b], Fürnkranz [1999], Gövert et al. [1999], and Oh et al. [2000] and experimentally compared by Yang et al. [2002].
- (2) *The hierarchical structure of the category set*: This may be used, for example, by decomposing the classification problem into a number of smaller classification problems, each corresponding to a branching decision at an internal node. Techniques exploiting this intuition in a TC context have been presented by Dumais and Chen [2000], Chakrabarti et al. [1998a], Koller and Sahami [1997], McCallum et al. [1998], Ruiz and Srinivasan [1999], and Weigend et al. [1999].

<b>if</b>	<i>((wheat &amp; farm)</i>	<b>or</b>
	<i>(wheat &amp; commodity)</i>	<b>or</b>
	<i>(bushels &amp; export)</i>	<b>or</b>
	<i>(wheat &amp; tonnes)</i>	<b>or</b>
	<i>(wheat &amp; winter &amp; ¬ soft))</i>	<b>then</b> WHEAT <b>else</b> ¬ WHEAT

**Fig. 1.** Rule-based classifier for the WHEAT category; key words are indicated in *italic*, categories are indicated in SMALL CAPS (from Apté et al. [1994]).

#### 4. THE MACHINE LEARNING APPROACH TO TEXT CATEGORIZATION

In the '80s, the most popular approach (at least in operational settings) for the creation of automatic document classifiers consisted in manually building, by means of *knowledge engineering* (KE) techniques, an expert system capable of taking TC decisions. Such an expert system would typically consist of a set of manually defined logical rules, one per category, of type

**if** *⟨DNF formula⟩* **then** *⟨category⟩*.

A DNF (“disjunctive normal form”) formula is a disjunction of conjunctive clauses; the document is classified under *⟨category⟩* iff it satisfies the formula, that is, iff it satisfies at least one of the clauses. The most famous example of this approach is the CONSTRUE system [Hayes et al. 1990], built by Carnegie Group for the Reuters news agency. A sample rule of the type used in CONSTRUE is illustrated in Figure 1.

The drawback of this approach is the *knowledge acquisition bottleneck* well known from the expert systems literature. That is, the rules must be manually defined by a knowledge engineer with the aid of a domain expert (in this case, an expert in the membership of documents in the chosen set of categories): if the set of categories is updated, then these two professionals must intervene again, and if the classifier is ported to a completely different domain (i.e., set of categories), a different domain expert needs to intervene and the work has to be repeated from scratch.

On the other hand, it was originally suggested that this approach can give very good effectiveness results: Hayes et al. [1990] reported a .90 “breakeven” result (see Section 7) on a subset of the Reuters test collection, a figure that outperforms

even the best classifiers built in the late '90s by state-of-the-art ML techniques. However, no other classifier has been tested on the same dataset as CONSTRUE, and it is not clear whether this was a randomly chosen or a favorable subset of the entire Reuters collection. As argued by Yang [1999], the results above do not allow us to state that these effectiveness results may be obtained in general.

Since the early '90s, the ML approach to TC has gained popularity and has eventually become the dominant one, at least in the research community (see Mitchell [1996] for a comprehensive introduction to ML). In this approach, a general inductive process (also called the *learner*) automatically builds a classifier for a category  $c_i$  by observing the characteristics of a set of documents manually classified under  $c_i$  or  $\bar{c}_i$  by a domain expert; from these characteristics, the inductive process gleans the characteristics that a new unseen document should have in order to be classified under  $c_i$ . In ML terminology, the classification problem is an activity of *supervised* learning, since the learning process is “supervised” by the knowledge of the categories and of the training instances that belong to them.<sup>2</sup>

The advantages of the ML approach over the KE approach are evident. The engineering effort goes toward the construction not of a classifier, but of an automatic builder of classifiers (the *learner*). This means that if a learner is (as it often is) available off-the-shelf, all that is needed is the inductive, *automatic* construction of a classifier from a set of manually classified documents. The same happens if a

<sup>2</sup> Within the area of content-based document management tasks, an example of an *unsupervised* learning activity is *document clustering* (see Section 1).



classifier already exists and the original set of categories is updated, or if the classifier is ported to a completely different domain.

In the ML approach, the preclassified documents are then the key resource. In the most favorable case, they are already available; this typically happens for organizations which have previously carried out the same categorization activity manually and decide to automate the process. The less favorable case is when no manually classified documents are available; this typically happens for organizations that start a categorization activity and opt for an automated modality straightaway. The ML approach is more convenient than the KE approach also in this latter case. In fact, it is easier to manually classify a set of documents than to build and tune a set of rules, since it is easier to characterize a concept extensionally (i.e., to select instances of it) than intensionally (i.e., to describe the concept in words, or to describe a procedure for recognizing its instances).

Classifiers built by means of ML techniques nowadays achieve impressive levels of effectiveness (see Section 7), making automatic classification a *qualitatively* (and not only economically) viable alternative to manual classification.

#### 4.1. Training Set, Test Set, and Validation Set

The ML approach relies on the availability of an *initial corpus*  $\Omega = \{d_1, \dots, d_{|\Omega|}\} \subset \mathcal{D}$  of documents preclassified under  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$ . That is, the values of the total function  $\Phi : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$  are known for every pair  $\langle d_j, c_i \rangle \in \Omega \times \mathcal{C}$ . A document  $d_j$  is a *positive example* of  $c_i$  if  $\Phi(d_j, c_i) = T$ , a *negative example* of  $c_i$  if  $\Phi(d_j, c_i) = F$ .

In research settings (and in most operational settings too), once a classifier  $\Phi$  has been built it is desirable to evaluate its effectiveness. In this case, prior to classifier construction the initial corpus is split in two sets, not necessarily of equal size:

—a *training(-and-validation) set*  $TV = \{d_1, \dots, d_{|TV|}\}$ . The classifier  $\Phi$  for categories  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  is inductively

built by observing the characteristics of these documents;

—a *test set*  $Te = \{d_{|TV|+1}, \dots, d_{|\Omega|}\}$ , used for testing the effectiveness of the classifiers. Each  $d_j \in Te$  is fed to the classifier, and the classifier decisions  $\Phi(d_j, c_i)$  are compared with the expert decisions  $\check{\Phi}(d_j, c_i)$ . A measure of classification effectiveness is based on how often the  $\Phi(d_j, c_i)$  values match the  $\check{\Phi}(d_j, c_i)$  values.

The documents in  $Te$  cannot participate in any way in the inductive construction of the classifiers; if this condition were not satisfied, the experimental results obtained would likely be unrealistically good, and the evaluation would thus have no scientific character [Mitchell 1996, page 129]. In an operational setting, after evaluation has been performed one would typically retrain the classifier on the entire initial corpus, in order to boost effectiveness. In this case, the results of the previous evaluation would be a pessimistic estimate of the real performance, since the final classifier has been trained on more data than the classifier evaluated.

This is called the *train-and-test* approach. An alternative is the *k-fold cross-validation* approach (see Mitchell [1996], page 146), in which  $k$  different classifiers  $\Phi_1, \dots, \Phi_k$  are built by partitioning the initial corpus into  $k$  disjoint sets  $Te_1, \dots, Te_k$  and then iteratively applying the train-and-test approach on pairs  $\langle TV_i = \Omega - Te_i, Te_i \rangle$ . The final effectiveness figure is obtained by individually computing the effectiveness of  $\Phi_1, \dots, \Phi_k$ , and then averaging the individual results in some way.

In both approaches, it is often the case that the internal parameters of the classifiers must be tuned by testing which values of the parameters yield the best effectiveness. In order to make this optimization possible, in the train-and-test approach the set  $\{d_1, \dots, d_{|TV|}\}$  is further split into a *training set*  $Tr = \{d_1, \dots, d_{|Tr|}\}$ , from which the classifier is built, and a *validation set*  $Va = \{d_{|Tr|+1}, \dots, d_{|TV|}\}$  (sometimes called a *hold-out set*), on which the repeated tests of the classifier aimed

at parameter optimization are performed; the obvious variant may be used in the  $k$ -fold cross-validation case. Note that, for the same reason why we do not test a classifier on the documents it has been trained on, we do not test it on the documents it has been optimized on: test set and validation set must be kept separate.<sup>3</sup>

Given a corpus  $\Omega$ , one may define the *generality*  $g_{\Omega}(c_i)$  of a category  $c_i$  as the percentage of documents that belong to  $c_i$ , that is:

$$g_{\Omega}(c_i) = \frac{|\{d_j \in \Omega \mid \check{\Phi}(d_j, c_i) = T\}|}{|\Omega|}.$$

The *training set generality*  $g_{T_r}(c_i)$ , *validation set generality*  $g_{V_a}(c_i)$ , and *test set generality*  $g_{T_e}(c_i)$  of  $c_i$  may be defined in the obvious way.

#### 4.2. Information Retrieval Techniques and Text Categorization

Text categorization heavily relies on the basic machinery of IR. The reason is that TC is a content-based document management task, and as such it shares many characteristics with other IR tasks such as text search.

IR techniques are used in three phases of the text classifier life cycle:

- (1) IR-style *indexing* is always performed on the documents of the initial corpus and on those to be classified during the operational phase;
- (2) IR-style techniques (such as document-request matching, query reformulation, ...) are often used in the *inductive construction* of the classifiers;
- (3) IR-style *evaluation* of the effectiveness of the classifiers is performed.

The various approaches to classification differ mostly for how they tackle (2), although in a few cases nonstandard

<sup>3</sup> From now on, we will take the freedom to use the expression “test document” to denote any document not in the training set and validation set. This includes thus any document submitted to the classifier in the operational phase.

approaches to (1) and (3) are also used. Indexing, induction, and evaluation are the themes of Sections 5, 6 and 7, respectively.

## 5. DOCUMENT INDEXING AND DIMENSIONALITY REDUCTION

### 5.1. Document Indexing

Texts cannot be directly interpreted by a classifier or by a classifier-building algorithm. Because of this, an *indexing* procedure that maps a text  $d_j$  into a compact representation of its content needs to be uniformly applied to training, validation, and test documents. The choice of a representation for text depends on what one regards as the meaningful units of text (the problem of *lexical semantics*) and the meaningful natural language rules for the combination of these units (the problem of *compositional semantics*). Similarly to what happens in IR, in TC this latter problem is usually disregarded,<sup>4</sup> and a text  $d_j$  is usually represented as a vector of term *weights*  $\vec{d}_j = (w_{1j}, \dots, w_{|T|j})$ , where  $T$  is the set of *terms* (sometimes called *features*) that occur at least once in at least one document of  $Tr$ , and  $0 \leq w_{kj} \leq 1$  represents, loosely speaking, how much term  $t_k$  contributes to the semantics of document  $d_j$ . Differences among approaches are accounted for by

- (1) different ways to understand what a term is;
- (2) different ways to compute term weights.

A typical choice for (1) is to identify terms with words. This is often called either the *set of words* or the *bag of words* approach to document representation, depending on whether weights are binary or not.

In a number of experiments [Apté et al. 1994; Dumais et al. 1998; Lewis 1992a], it has been found that representations more sophisticated than this do not yield significantly better effectiveness, thereby confirming similar results from IR

<sup>4</sup> An exception to this is represented by learning approaches based on *hidden Markov models* [Denoyer et al. 2001; Frasconi et al. 2002].

[Salton and Buckley 1988]. In particular, some authors have used *phrases*, rather than individual words, as indexing terms [Fuhr et al. 1991; Schütze et al. 1995; Tzeras and Hartmann 1993], but the experimental results found to date have not been uniformly encouraging, irrespectively of whether the notion of “phrase” is motivated

- syntactically*, that is, the phrase is such according to a grammar of the language (see Lewis [1992a]); or
- statistically*, that is, the phrase is not grammatically such, but is composed of a set/sequence of words whose patterns of contiguous occurrence in the collection are statistically significant (see Caropreso et al. [2001]).

Lewis [1992a] argued that the likely reason for the discouraging results is that, although indexing languages based on phrases have superior semantic qualities, they have inferior statistical qualities with respect to word-only indexing languages: a phrase-only indexing language has “more terms, more synonymous or nearly synonymous terms, lower consistency of assignment (since synonymous terms are not assigned to the same documents), and lower document frequency for terms” [Lewis 1992a, page 40]. Although his remarks are about syntactically motivated phrases, they also apply to statistically motivated ones, although perhaps to a smaller degree. A combination of the two approaches is probably the best way to go: Tzeras and Hartmann [1993] obtained significant improvements by using noun phrases obtained through a combination of syntactic and statistical criteria, where a “crude” syntactic method was complemented by a statistical filter (only those syntactic phrases that occurred at least three times in the positive examples of a category  $c_i$  were retained). It is likely that the final word on the usefulness of phrase indexing in TC has still to be told, and investigations in this direction are still being actively pursued [Caropreso et al. 2001; Mladenić and Grobelnik 1998].

As for issue (2), weights usually range between 0 and 1 (an exception is

Lewis et al. [1996]), and for ease of exposition we will assume they always do. As a special case, binary weights may be used (1 denoting presence and 0 absence of the term in the document); whether binary or nonbinary weights are used depends on the classifier learning algorithm used. In the case of nonbinary indexing, for determining the weight  $w_{kj}$  of term  $t_k$  in document  $d_j$  any IR-style indexing technique that represents a document as a vector of weighted terms may be used. Most of the times, the standard *tfidf* function is used (see Salton and Buckley [1988]), defined as

$$tfidf(t_k, d_j) = \#(t_k, d_j) \cdot \log \frac{|Tr|}{\#_{Tr}(t_k)}, \quad (1)$$

where  $\#(t_k, d_j)$  denotes the number of times  $t_k$  occurs in  $d_j$ , and  $\#_{Tr}(t_k)$  denotes the *document frequency* of term  $t_k$ , that is, the number of documents in  $Tr$  in which  $t_k$  occurs. This function embodies the intuitions that (i) the more often a term occurs in a document, the more it is representative of its content, and (ii) the more documents a term occurs in, the less discriminating it is.<sup>5</sup> Note that this formula (as most other indexing formulae) weights the importance of a term to a document in terms of occurrence considerations only, thereby deeming of null importance the order in which the terms occur in the document and the syntactic role they play. In other words, the semantics of a document is reduced to the collective lexical semantics of the terms that occur in it, thereby disregarding the issue of compositional semantics (an exception are the representation techniques used for FOIL [Cohen 1995a] and SLEEPING EXPERTS [Cohen and Singer 1999]).

In order for the weights to fall in the  $[0,1]$  interval and for the documents to be represented by vectors of equal length, the weights resulting from *tfidf* are often

<sup>5</sup> There exist many variants of *tfidf*, that differ from each other in terms of logarithms, normalization or other correction factors. Formula 1 is just one of the possible instances of this class; see Salton and Buckley [1988] and Singhal et al. [1996] for variations on this theme.

normalized by *cosine normalization*, given by

$$w_{kj} = \frac{tfidf(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} (tfidf(t_s, d_j))^2}}. \quad (2)$$

Although normalized *tfidf* is the most popular one, other indexing functions have also been used, including probabilistic techniques [Gövert et al. 1999] or techniques for indexing structured documents [Larkey and Croft 1996]. Functions different from *tfidf* are especially needed when *Tr* is not available in its entirety from the start and  $\#_{Tr}(t_k)$  cannot thus be computed, as in adaptive filtering; in this case, approximations of *tfidf* are usually employed [Dagan et al. 1997, Section 4.3].

Before indexing, the removal of *function words* (i.e., topic-neutral words such as articles, prepositions, conjunctions, etc.) is almost always performed (exceptions include Lewis et al. [1996], Nigam et al. [2000], and Riloff [1995]).<sup>6</sup> Concerning *stemming* (i.e., grouping words that share the same morphological root), its suitability to TC is controversial. Although, similarly to unsupervised term clustering (see Section 5.5.1) of which it is an instance, stemming has sometimes been reported to hurt effectiveness (e.g., Baker and McCallum [1998]), the recent tendency is to adopt it, as it reduces both the dimensionality of the term space (see Section 5.3) and the stochastic dependence between terms (see Section 6.2).

Depending on the application, either the full text of the document or selected parts of it are indexed. While the former option is the rule, exceptions exist. For instance, in a patent categorization application Larkey [1999] indexed only the title, the abstract, the first 20 lines of the summary, and the section containing

the claims of novelty of the described invention. This approach was made possible by the fact that documents describing patents are structured. Similarly, when a document title is available, one can pay extra importance to the words it contains [Apté et al. 1994; Cohen and Singer 1999; Weiss et al. 1999]. When documents are flat, identifying the most relevant part of a document is instead a nonobvious task.

## 5.2. The Darmstadt Indexing Approach

The AIR/X system [Fuhr et al. 1991] occupies a special place in the literature on indexing for TC. This system is the final result of the AIR project, one of the most important efforts in the history of TC: spanning a duration of more than 10 years [Knorz 1982; Tzeras and Hartmann 1993], it has produced a system operatively employed since 1985 in the classification of corpora of scientific literature of  $O(10^5)$  documents and  $O(10^4)$  categories, and has had important theoretical spin-offs in the field of probabilistic indexing [Fuhr 1989; Fuhr and Buckley 1991].<sup>7</sup>

The approach to indexing taken in AIR/X is known as the *Darmstadt Indexing Approach* (DIA) [Fuhr 1985]. Here, “indexing” is used in the sense of Section 3.1, that is, as using terms from a controlled vocabulary, and is thus a synonym of TC (the DIA was later extended to indexing with free terms [Fuhr and Buckley 1991]). The idea that underlies the DIA is the use of a much wider set of “features” than described in Section 5.1. All other approaches mentioned in this paper view *terms* as the dimensions of the learning space, where terms may be single words, stems, phrases, or (see Sections 5.5.1 and 5.5.2) combinations of any of these. In contrast, the DIA considers *properties* (of terms, documents,

<sup>6</sup> One application of TC in which it would be inappropriate to remove function words is author identification for documents of disputed paternity. In fact, as noted in Manning and Schütze [1999], page 589, “it is often the ‘little’ words that give an author away (for example, the relative frequencies of words like because or though).”

<sup>7</sup> The AIR/X system, its applications (including the AIR/PHYS system [Biebricher et al. 1988], an application of AIR/X to indexing physics literature), and its experiments have also been richly documented in a series of papers and doctoral theses written in German. The interested reader may consult Fuhr et al. [1991] for a detailed bibliography.

categories, or pairwise relationships among these) as basic dimensions of the learning space. Examples of these are

- properties of a term  $t_k$* : e.g. the *idf* of  $t_k$ ;
- properties of the relationship between a term  $t_k$  and a document  $d_j$* : for example, the *tf* of  $t_k$  in  $d_j$ ; or the location (e.g., in the title, or in the abstract) of  $t_k$  within  $d_j$ ;
- properties of a document  $d_j$* : for example, the length of  $d_j$ ;
- properties of a category  $c_i$* : for example, the training set generality of  $c_i$ .

For each possible document-category pair, the values of these features are collected in a so-called *relevance description* vector  $rd(d_j, c_i)$ . The size of this vector is determined by the number of properties considered, and is thus independent of specific terms, categories, or documents (for multivalued features, appropriate aggregation functions are applied in order to yield a single value to be included in  $rd(d_j, c_i)$ ); in this way an abstraction from specific terms, categories, or documents is achieved.

The main advantage of this approach is the possibility to consider additional features that can hardly be accounted for in the usual term-based approaches, for example, the location of a term within a document, or the certainty with which a phrase was identified in a document. The term-category relationship is described by estimates, derived from the training set, of the probability  $P(c_i | t_k)$  that a document belongs to category  $c_i$ , given that it contains term  $t_k$  (the *DIA association factor*).<sup>8</sup> Relevance description vectors  $rd(d_j, c_i)$  are then the final representations that are used for the classification of document  $d_j$  under category  $c_i$ .

The essential ideas of the DIA—transforming the classification space by means of abstraction and using a more detailed text representation than the standard bag-of-words approach—have not

been taken up by other researchers so far. For new TC applications dealing with structured documents or categorization of Web pages, these ideas may become of increasing importance.

### 5.3. Dimensionality Reduction

Unlike in text retrieval, in TC the high dimensionality of the term space (i.e., the large value of  $|T|$ ) may be problematic. In fact, while typical algorithms used in text retrieval (such as cosine matching) can scale to high values of  $|T|$ , the same does not hold of many sophisticated learning algorithms used for classifier induction (e.g., the LLSF algorithm of Yang and Chute [1994]). Because of this, before classifier induction one often applies a pass of *dimensionality reduction* (DR), whose effect is to reduce the size of the vector space from  $|T|$  to  $|T'| \ll |T|$ ; the set  $T'$  is called the *reduced term set*.

DR is also beneficial since it tends to reduce *overfitting*, that is, the phenomenon by which a classifier is tuned also to the *contingent* characteristics of the training data rather than just the *constitutive* characteristics of the categories. Classifiers that overfit the training data are good at reclassifying the data they have been trained on, but much worse at classifying previously unseen data. Experiments have shown that, in order to avoid overfitting a number of training examples roughly proportional to the number of terms used is needed; Fuhr and Buckley [1991, page 235] have suggested that 50–100 training examples per term may be needed in TC tasks. This means that, if DR is performed, overfitting may be avoided even if a smaller amount of training examples is used. However, in removing terms the risk is to remove potentially useful information on the meaning of the documents. It is then clear that, in order to obtain optimal (cost-)effectiveness, the reduction process must be performed with care. Various DR methods have been proposed, either from the information theory or from the linear algebra literature, and their relative merits have been tested by experimentally evaluating the variation

<sup>8</sup> Association factors are called *adhesion coefficients* in many early papers on TC; see Field [1975]; Robertson and Harding [1984].

in effectiveness that a given classifier undergoes after application of the function to the term space.

There are two distinct ways of viewing DR, depending on whether the task is performed locally (i.e., for each individual category) or globally:

- local DR*: for each category  $c_i$ , a set  $\mathcal{T}'_i$  of terms, with  $|\mathcal{T}'_i| \ll |\mathcal{T}|$ , is chosen for classification under  $c_i$  (see Apté et al. [1994]; Lewis and Ringuette [1994]; Li and Jain [1998]; Ng et al. [1997]; Sable and Hatzivassiloglou [2000]; Schütze et al. [1995], Wiener et al. [1995]). This means that different subsets of  $\vec{d}_j$  are used when working with the different categories. Typical values are  $10 \leq |\mathcal{T}'_i| \leq 50$ .
- global DR*: a set  $\mathcal{T}'$  of terms, with  $|\mathcal{T}'| \ll |\mathcal{T}|$ , is chosen for the classification under all categories  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  (see Caropreso et al. [2001]; Mladenić [1998]; Yang [1999]; Yang and Pedersen [1997]).

This distinction usually does not impact on the choice of DR technique, since most such techniques can be used (and have been used) for local and global DR alike (*supervised* DR techniques—see Section 5.5.1—are exceptions to this rule). In the rest of this section, we will assume that the global approach is used, although everything we will say also applies to the local approach.

A second, orthogonal distinction may be drawn in terms of the nature of the resulting terms:

- DR by term selection*:  $\mathcal{T}'$  is a subset of  $\mathcal{T}$ ;
- DR by term extraction*: the terms in  $\mathcal{T}'$  are not of the same type of the terms in  $\mathcal{T}$  (e.g., if the terms in  $\mathcal{T}$  are words, the terms in  $\mathcal{T}'$  may not be words at all), but are obtained by combinations or transformations of the original ones.

Unlike in the previous distinction, these two ways of doing DR are tackled by very different techniques; we will address them separately in the next two sections.

#### 5.4. Dimensionality Reduction by Term Selection

Given a predetermined integer  $r$ , techniques for term selection (also called *term space reduction*—TSR) attempt to select, from the original set  $\mathcal{T}$ , the set  $\mathcal{T}'$  of terms (with  $|\mathcal{T}'| \ll |\mathcal{T}|$ ) that, when used for document indexing, yields the highest effectiveness. Yang and Pedersen [1997] have shown that TSR may even result in a moderate ( $\leq 5\%$ ) increase in effectiveness, depending on the classifier, on the *aggressivity*  $\frac{|\mathcal{T}'|}{|\mathcal{T}|}$  of the reduction, and on the TSR technique used.

Moulinier et al. [1996] have used a so-called *wrapper* approach, that is, one in which  $\mathcal{T}'$  is identified by means of the same learning method that will be used for building the classifier [John et al. 1994]. Starting from an initial term set, a new term set is generated by either adding or removing a term. When a new term set is generated, a classifier based on it is built and then tested on a validation set. The term set that results in the best effectiveness is chosen. This approach has the advantage of being tuned to the learning algorithm being used; moreover, if local DR is performed, different numbers of terms for different categories may be chosen, depending on whether a category is or is not easily separable from the others. However, the sheer size of the space of different term sets makes its cost-prohibitive for standard TC applications.

A computationally easier alternative is the *filtering* approach [John et al. 1994], that is, keeping the  $|\mathcal{T}'| \ll |\mathcal{T}|$  terms that receive the highest score according to a function that measures the “importance” of the term for the TC task. We will explore this solution in the rest of this section.

**5.4.1. Document Frequency.** A simple and effective global TSR function is the *document frequency*  $\#_{\mathcal{T}}(t_k)$  of a term  $t_k$ , that is, only the terms that occur in the highest number of documents are retained. In a series of experiments Yang and Pedersen [1997] have shown that with  $\#_{\mathcal{T}}(t_k)$  it is possible to reduce the dimensionality by a factor of 10 with no loss in effectiveness (a

reduction by a factor of 100 bringing about just a small loss).

This seems to indicate that the terms occurring most frequently in the collection are the most valuable for TC. As such, this would seem to contradict a well-known “law” of IR, according to which the terms with low-to-medium document frequency are the most informative ones [Salton and Buckley 1988]. But these two results do not contradict each other, since it is well known (see Salton et al. [1975]) that the large majority of the words occurring in a corpus have a *very* low document frequency; this means that by reducing the term set by a factor of 10 using document frequency, only such words are removed, while the words from low-to-medium to high document frequency are preserved. Of course, stop words need to be removed in advance, lest only topic-neutral words are retained [Mladenić 1998].

Finally, note that a slightly more empirical form of TSR by document frequency is adopted by many authors, who remove all terms occurring in at most  $x$  training documents (popular values for  $x$  range from 1 to 3), either as the only form of DR [Maron 1961; Ittner et al. 1995] or before applying another more sophisticated form [Dumais et al. 1998; Li and Jain 1998]. A variant of this policy is removing all terms that occur at most  $x$  times in the training set (e.g., Dagan et al. [1997]; Joachims [1997]), with popular values for  $x$  ranging from 1 (e.g., Baker and McCallum [1998]) to 5 (e.g., Apté et al. [1994]; Cohen [1995a]).

**5.4.2. Other Information-Theoretic Term Selection Functions.** Other more sophisticated information-theoretic functions have been used in the literature, among them the *DIA association factor* [Fuhr et al. 1991], *chi-square* [Caropreso et al. 2001; Galavotti et al. 2000; Schütze et al. 1995; Sebastiani et al. 2000; Yang and Pedersen 1997; Yang and Liu 1999], *NGL coefficient* [Ng et al. 1997; Ruiz and Srinivasan 1999], *information gain* [Caropreso et al. 2001; Larkey 1998; Lewis 1992a; Lewis and Ringuette 1994; Mladenić 1998; Moulinier and Ganascia

1996; Yang and Pedersen 1997, Yang and Liu 1999], *mutual information* [Dumais et al. 1998; Lam et al. 1997; Larkey and Croft 1996; Lewis and Ringuette 1994; Li and Jain 1998; Moulinier et al. 1996; Ruiz and Srinivasan 1999; Taira and Haruno 1999; Yang and Pedersen 1997], *odds ratio* [Caropreso et al. 2001; Mladenić 1998; Ruiz and Srinivasan 1999], *relevancy score* [Wiener et al. 1995], and *GSS coefficient* [Galavotti et al. 2000]. The mathematical definitions of these measures are summarized for convenience in Table I.<sup>9</sup> Here, probabilities are interpreted on an event space of documents (e.g.,  $P(t_k, c_i)$  denotes the probability that, for a random document  $x$ , term  $t_k$  does not occur in  $x$  and  $x$  belongs to category  $c_i$ ), and are estimated by counting occurrences in the training set. All functions are specified “locally” to a specific category  $c_i$ ; in order to assess the value of a term  $t_k$  in a “global,” category-independent sense, either the sum  $f_{sum}(t_k) = \sum_{i=1}^{|C|} f(t_k, c_i)$ , or the weighted sum  $f_{wsum}(t_k) = \sum_{i=1}^{|C|} P(c_i) f(t_k, c_i)$ , or the maximum  $f_{max}(t_k) = \max_{i=1}^{|C|} f(t_k, c_i)$  of their category-specific values  $f(t_k, c_i)$  are usually computed.

These functions try to capture the intuition that the best terms for  $c_i$  are the ones distributed most differently in the sets of positive and negative examples of  $c_i$ . However, interpretations of this principle vary across different functions. For instance, in the experimental sciences  $\chi^2$  is used to measure how the results of an observation differ (i.e., are independent) from the results expected according to an initial hypothesis (lower values indicate lower dependence). In DR we measure how independent  $t_k$  and  $c_i$  are. The terms  $t_k$

<sup>9</sup> For better uniformity Table I views all the TSR functions of this section in terms of subjective probability. In some cases such as  $\chi^2(t_k, c_i)$  this is slightly artificial, since this function is not usually viewed in probabilistic terms. The formulae refer to the “local” (i.e., category-specific) forms of the functions, which again is slightly artificial in some cases. Note that the NGL and GSS coefficients are here named after their authors, since they had originally been given names that might generate some confusion if used here.

**Table I.** Main Functions Used for Term Space Reduction Purposes. Information Gain Is Also Known as *Expected Mutual Information*, and Is Used Under This Name by Lewis [1992a, page 44] and Larkey [1998]. In the  $RS(t_k, c_i)$  Formula,  $d$  Is a Constant Damping Factor.

Function	Denoted by	Mathematical form
DIA association factor	$z(t_k, c_i)$	$P(c_i   t_k)$
Information gain	$IG(t_k, c_i)$	$\sum_{c \in \{c_i, \bar{c}_i\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t, c) \cdot \log \frac{P(t, c)}{P(t) \cdot P(c)}$
Mutual information	$MI(t_k, c_i)$	$\log \frac{P(t_k, c_i)}{P(t_k) \cdot P(c_i)}$
Chi-square	$\chi^2(t_k, c_i)$	$\frac{ Tr  \cdot [P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i) \cdot P(\bar{t}_k, c_i)]^2}{P(t_k) \cdot P(\bar{t}_k) \cdot P(c_i) \cdot P(\bar{c}_i)}$
NGL coefficient	$NGL(t_k, c_i)$	$\frac{\sqrt{ Tr  \cdot [P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i) \cdot P(\bar{t}_k, c_i)]}}{\sqrt{P(t_k) \cdot P(\bar{t}_k) \cdot P(c_i) \cdot P(\bar{c}_i)}}$
Relevancy score	$RS(t_k, c_i)$	$\log \frac{P(t_k   c_i) + d}{P(\bar{t}_k   \bar{c}_i) + d}$
Odds ratio	$OR(t_k, c_i)$	$\frac{P(t_k   c_i) \cdot (1 - P(t_k   \bar{c}_i))}{(1 - P(t_k   c_i)) \cdot P(t_k   \bar{c}_i)}$
GSS coefficient	$GSS(t_k, c_i)$	$P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i) \cdot P(\bar{t}_k, c_i)$

with the lowest value for  $\chi^2(t_k, c_i)$  are thus the most independent from  $c_i$ ; since we are interested in the terms which are not, we select the terms for which  $\chi^2(t_k, c_i)$  is highest.

While each TSR function has its own rationale, the ultimate word on its value is the effectiveness it brings about. Various experimental comparisons of TSR functions have thus been carried out [Caropreso et al. 2001; Galavotti et al. 2000; Mladenić 1998; Yang and Pedersen 1997]. In these experiments most functions listed in Table I (with the possible exception of  $MI$ ) have improved on the results of document frequency. For instance, Yang and Pedersen [1997] have shown that, with various classifiers and various initial corpora, sophisticated techniques such as  $IG_{sum}(t_k, c_i)$  or  $\chi_{max}^2(t_k, c_i)$  can reduce the dimensionality of the term space by a factor of 100 with no loss (or even with a small increase) of effectiveness. Collectively, the experiments reported in the above-mentioned papers seem to indicate that  $\{OR_{sum}, NGL_{sum}, GSS_{max}\} > \{\chi_{max}^2, IG_{sum}\} > \{\chi_{avg}^2\} \gg \{MI_{max}, MI_{wsum}\}$ , where “>” means “performs better than.”

However, it should be noted that these results are just indicative, and that more general statements on the relative merits of these functions could be made only as a result of comparative experiments performed in thoroughly controlled conditions and on a variety of different situations (e.g., different classifiers, different initial corpora, ...).

### 5.5. Dimensionality Reduction by Term Extraction

Given a predetermined  $|T'| \ll |T|$ , *term extraction* attempts to generate, from the original set  $T$ , a set  $T'$  of “synthetic” terms that maximize effectiveness. The rationale for using synthetic (rather than naturally occurring) terms is that, due to the pervasive problems of polysemy, homonymy, and synonymy, the original terms may not be optimal dimensions for document content representation. Methods for term extraction try to solve these problems by creating artificial terms that do not suffer from them. Any term extraction method consists in (i) a method for extracting the new terms from the



old ones, and (ii) a method for converting the original document representations into new representations based on the newly synthesized dimensions. Two term extraction methods have been experimented with in TC, namely term clustering and latent semantic indexing.

*5.5.1. Term Clustering.* *Term clustering* tries to group words with a high degree of pairwise semantic relatedness, so that the groups (or their centroids, or a representative of them) may be used instead of the terms as dimensions of the vector space. Term clustering is different from term selection, since the former tends to address terms *synonymous* (or near-synonymous) with other terms, while the latter targets *noninformative* terms.<sup>10</sup>

Lewis [1992a] was the first to investigate the use of term clustering in TC. The method he employed, called *reciprocal nearest neighbor clustering*, consists in creating clusters of two terms that are one the most similar to the other according to some measure of similarity. His results were inferior to those obtained by single-word indexing, possibly due to a disappointing performance by the clustering method: as Lewis [1992a, page 48] said, “The relationships captured in the clusters are mostly accidental, rather than the systematic relationships that were hoped for.”

Li and Jain [1998] viewed semantic relatedness between words in terms of their co-occurrence and co-absence within training documents. By using this technique in the context of a hierarchical clustering algorithm, they witnessed only a marginal effectiveness improvement; however, the small size of their experiment (see Section 6.11) hardly allows any definitive conclusion to be reached.

Both Lewis [1992a] and Li and Jain [1998] are examples of *unsupervised* clustering, since clustering is not affected by the category labels attached to the docu-

ments. Baker and McCallum [1998] provided instead an example of *supervised* clustering, as the *distributional clustering* method they employed clusters together those terms that tend to indicate the presence of the same category, or group of categories. Their experiments, carried out in the context of a Naïve Bayes classifier (see Section 6.2), showed only a 2% effectiveness loss with an aggressivity of 1,000, and even showed some effectiveness improvement with less aggressive levels of reduction. Later experiments by Slonim and Tishby [2001] have confirmed the potential of supervised clustering methods for term extraction.

*5.5.2. Latent Semantic Indexing.* *Latent semantic indexing* (LSI—[Deerwester et al. 1990]) is a DR technique developed in IR in order to address the problems deriving from the use of synonymous, near-synonymous, and polysemous words as dimensions of document representations. This technique compresses document vectors into vectors of a lower-dimensional space whose dimensions are obtained as combinations of the original dimensions by looking at their patterns of co-occurrence. In practice, LSI infers the dependence among the original terms from a corpus and “wires” this dependence into the newly obtained, independent dimensions. The function mapping original vectors into new vectors is obtained by applying a singular value decomposition to the matrix formed by the original document vectors. In TC this technique is applied by deriving the mapping function from the training set and then applying it to training and test documents alike.

One characteristic of LSI is that the newly obtained dimensions are not, unlike in term selection and term clustering, intuitively interpretable. However, they work well in bringing out the “latent” semantic structure of the vocabulary used in the corpus. For instance, Schütze et al. [1995, page 235] discussed the classification under category Demographic shifts in the U.S. with economic impact of a document that was indeed a positive

<sup>10</sup> Some term selection methods, such as wrapper methods, also address the problem of redundancy.

test instance for the category, and that contained, among others, the quite revealing sentence The nation grew to 249.6 million people in the 1980s as more Americans left the industrial and agricultural heartlands for the South and West. The classifier decision was incorrect when local DR had been performed by  $\chi^2$ -based term selection retaining the top original 200 terms, but was correct when the same task was tackled by means of LSI. This well exemplifies how LSI works: the above sentence does not contain any of the 200 terms most relevant to the category selected by  $\chi^2$ , but quite possibly the words contained in it have concurred to produce one or more of the LSI higher-order terms that generate the document space of the category. As Schütze et al. [1995, page 230] put it, “if there is a great number of terms which all contribute a small amount of critical information, then the combination of evidence is a major problem for a term-based classifier.” A drawback of LSI, though, is that if some original term is particularly good in itself at discriminating a category, that discrimination power may be lost in the new vector space.

Wiener et al. [1995] used LSI in two alternative ways: (i) for local DR, thus creating several category-specific LSI representations, and (ii) for global DR, thus creating a single LSI representation for the entire category set. Their experiments showed the former approach to perform better than the latter, and both approaches to perform better than simple TSR based on Relevancy Score (see Table I).

Schütze et al. [1995] experimentally compared LSI-based term extraction with  $\chi^2$ -based TSR using three different classifier learning techniques (namely, linear discriminant analysis, logistic regression, and neural networks). Their experiments showed LSI to be far more effective than  $\chi^2$  for the first two techniques, while both methods performed equally well for the neural network classifier.

For other TC works that have used LSI or similar term extraction techniques, see Hull [1994], Li and Jain [1998],

Schütze [1998], Weigend et al. [1999], and Yang [1995].

## 6. INDUCTIVE CONSTRUCTION OF TEXT CLASSIFIERS

The inductive construction of text classifiers has been tackled in a variety of ways. Here we will deal only with the methods that have been most popular in TC, but we will also briefly mention the existence of alternative, less standard approaches.

We start by discussing the general form that a text classifier has. Let us recall from Section 2.4 that there are two alternative ways of viewing classification: “hard” (fully automated) classification and ranking (semiautomated) classification.

The inductive construction of a ranking classifier for category  $c_i \in \mathcal{C}$  usually consists in the definition of a function  $CSV_i : \mathcal{D} \rightarrow [0, 1]$  that, given a document  $d_j$ , returns a *categorization status value* for it, that is, a number between 0 and 1 which, roughly speaking, represents the evidence for the fact that  $d_j \in c_i$ . Documents are then ranked according to their  $CSV_i$  value. This works for “document-ranking TC”; “category-ranking TC” is usually tackled by ranking, for a given document  $d_j$ , its  $CSV_i$  scores for the different categories in  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$ .

The  $CSV_i$  function takes up different meanings according to the learning method used: for instance, in the “Naïve Bayes” approach of Section 6.2  $CSV_i(d_j)$  is defined in terms of a probability, whereas in the “Rocchio” approach discussed in Section 6.7  $CSV_i(d_j)$  is a measure of vector closeness in  $|\mathcal{T}|$ -dimensional space.

The construction of a “hard” classifier may follow two alternative paths. The former consists in the definition of a function  $CSV_i : \mathcal{D} \rightarrow \{T, F\}$ . The latter consists instead in the definition of a function  $CSV_i : \mathcal{D} \rightarrow [0, 1]$ , analogous to the one used for ranking classification, followed by the definition of a *threshold*  $\tau_i$  such that  $CSV_i(d_j) \geq \tau_i$  is interpreted

as  $T$  while  $CSV_i(d_j) < \tau_i$  is interpreted as  $F$ .<sup>11</sup>

The definition of thresholds will be the topic of Section 6.1. In Sections 6.2 to 6.12 we will instead concentrate on the definition of  $CSV_i$ , discussing a number of approaches that have been applied in the TC literature. In general we will assume we are dealing with “hard” classification; it will be evident from the context how and whether the approaches can be adapted to ranking classification. The presentation of the algorithms will be mostly qualitative rather than quantitative, that is, will focus on the methods for classifier learning rather than on the effectiveness and efficiency of the classifiers built by means of them; this will instead be the focus of Section 7.

### 6.1. Determining Thresholds

There are various policies for determining the threshold  $\tau_i$ , also depending on the constraints imposed by the application. The most important distinction is whether the threshold is derived *analytically* or *experimentally*.

The former method is possible only in the presence of a theoretical result that indicates how to compute the threshold that maximizes the expected value of the effectiveness function [Lewis 1995a]. This is typical of classifiers that output *probability* estimates of the membership of  $d_j$  in  $c_i$  (see Section 6.2) and whose effectiveness is computed by decision-theoretic measures such as *utility* (see Section 7.1.3); we thus defer the discussion of this policy (which is called *probability thresholding* in Lewis [1995a]) to Section 7.1.3.

When such a theoretical result is not known, one has to revert to the latter method, which consists in testing different values for  $\tau_i$  on a validation set and choosing the value which maximizes effectiveness. We call this policy *CSV thresholding*

[Cohen and Singer 1999; Schapire et al. 1998; Wiener et al. 1995]; it is also called *Scut* in Yang [1999]. Different  $\tau_i$ 's are typically chosen for the different  $c_i$ 's.

A second, popular experimental policy is *proportional thresholding* [Iwayama and Tokunaga 1995; Larkey 1998; Lewis 1992a; Lewis and Ringuette 1994; Wiener et al. 1995], also called *Pcut* in Yang [1999]. This policy consists in choosing the value of  $\tau_i$  for which  $g_{v_a}(c_i)$  is closest to  $g_{T}(c_i)$ , and embodies the principle that the same percentage of documents of both training and test set should be classified under  $c_i$ . For obvious reasons, this policy does not lend itself to document-pivoted TC.

Sometimes, depending on the application, a *fixed thresholding* policy (a.k.a. “*k*-per-doc” thresholding [Lewis 1992a] or *Rcut* [Yang 1999]) is applied, whereby it is stipulated that a fixed number  $k$  of categories, equal for all  $d_j$ 's, are to be assigned to each document  $d_j$ . This is often used, for instance, in applications of TC to automated document indexing [Field 1975; Lam et al. 1999]. Strictly speaking, however, this is not a thresholding policy in the sense defined at the beginning of Section 6, as it might happen that  $d'$  is classified under  $c_i$ ,  $d''$  is not, and  $CSV_i(d') < CSV_i(d'')$ . Quite clearly, this policy is mostly at home with document-pivoted TC. However, it suffers from a certain coarseness, as the fact that  $k$  is equal for all documents (nor could this be otherwise) allows no fine-tuning.

In his experiments Lewis [1992a] found the proportional policy to be superior to probability thresholding when microaveraged effectiveness was tested but slightly inferior with macroaveraging (see Section 7.1.1). Yang [1999] found instead *CSV* thresholding to be superior to proportional thresholding (possibly due to her category-specific optimization on a validation set), and found fixed thresholding to be consistently inferior to the other two policies. The fact that these latter results have been obtained across different classifiers no doubt reinforces them.

In general, aside from the considerations above, the choice of the thresholding

<sup>11</sup> Alternative methods are possible, such as training a classifier for which some standard, predefined value such as 0 is the threshold. For ease of exposition we will not discuss them.

policy may also be influenced by the application; for instance, in applying a text classifier to document indexing for Boolean systems a fixed thresholding policy might be chosen, while a proportional or CSV thresholding method might be chosen for Web page classification under hierarchical catalogues.

## 6.2. Probabilistic Classifiers

Probabilistic classifiers (see Lewis [1998] for a thorough discussion) view  $CSV_i(d_j)$  in terms of  $P(c_i | \vec{d}_j)$ , that is, the probability that a document represented by a vector  $\vec{d}_j = \langle w_{1j}, \dots, w_{|\mathcal{T}|j} \rangle$  of (binary or weighted) terms belongs to  $c_i$ , and compute this probability by an application of Bayes' theorem, given by

$$P(c_i | \vec{d}_j) = \frac{P(c_i)P(\vec{d}_j | c_i)}{P(\vec{d}_j)}. \quad (3)$$

In (3) the event space is the space of documents:  $P(\vec{d}_j)$  is thus the probability that a randomly picked document has vector  $\vec{d}_j$  as its representation, and  $P(c_i)$  the probability that a randomly picked document belongs to  $c_i$ .

The estimation of  $P(\vec{d}_j | c_i)$  in (3) is problematic, since the number of possible vectors  $\vec{d}_j$  is too high (the same holds for  $P(\vec{d}_j)$ , but for reasons that will be clear shortly this will not concern us). In order to alleviate this problem it is common to make the assumption that any two coordinates of the document vector are, when viewed as random variables, statistically independent of each other; this *independence assumption* is encoded by the equation

$$P(\vec{d}_j | c_i) = \prod_{k=1}^{|\mathcal{T}|} P(w_{kj} | c_i). \quad (4)$$

Probabilistic classifiers that use this assumption are called *Naïve Bayes* classifiers, and account for most of the probabilistic approaches to TC in the literature (see Joachims [1998]; Koller and

Sahami [1997]; Larkey and Croft [1996]; Lewis [1992a]; Lewis and Gale [1994]; Li and Jain [1998]; Robertson and Harding [1984]). The “naïve” character of the classifier is due to the fact that usually this assumption is, quite obviously, not verified in practice.

One of the best-known Naïve Bayes approaches is the *binary independence* classifier [Robertson and Sparck Jones 1976], which results from using binary-valued vector representations for documents. In this case, if we write  $p_{ki}$  as short for  $P(w_{kx} = 1 | c_i)$ , the  $P(w_{kj} | c_i)$  factors of (4) may be written as

$$\begin{aligned} P(w_{kj} | c_i) &= p_{ki}^{w_{kj}} (1 - p_{ki})^{1-w_{kj}} \\ &= \left( \frac{p_{ki}}{1 - p_{ki}} \right)^{w_{kj}} (1 - p_{ki}). \end{aligned} \quad (5)$$

We may further observe that in TC the document space is partitioned into two categories,<sup>12</sup>  $c_i$  and its complement  $\bar{c}_i$ , such that  $P(\bar{c}_i | \vec{d}_j) = 1 - P(c_i | \vec{d}_j)$ . If we plug in (4) and (5) into (3) and take logs we obtain

$$\begin{aligned} \log P(c_i | \vec{d}_j) &= \log P(c_i) + \sum_{k=1}^{|\mathcal{T}|} w_{kj} \log \frac{p_{ki}}{1 - p_{ki}} \\ &\quad + \sum_{k=1}^{|\mathcal{T}|} \log(1 - p_{ki}) - \log P(\vec{d}_j) \end{aligned} \quad (6)$$

$$\begin{aligned} \log(1 - P(c_i | \vec{d}_j)) &= \log(1 - P(c_i)) + \sum_{k=1}^{|\mathcal{T}|} w_{kj} \log \frac{p_{k\bar{i}}}{1 - p_{k\bar{i}}} \\ &\quad + \sum_{k=1}^{|\mathcal{T}|} \log(1 - p_{k\bar{i}}) - \log P(\vec{d}_j), \end{aligned} \quad (7)$$

<sup>12</sup> Cooper [1995] has pointed out that in this case the full independence assumption of (4) is not actually made in the Naïve Bayes classifier; the assumption needed here is instead the weaker *linked dependence assumption*, which may be written as  $\frac{P(\vec{d}_j | c_i)}{P(\vec{d}_j | \bar{c}_i)} = \prod_{k=1}^{|\mathcal{T}|} \frac{P(w_{kj} | c_i)}{P(w_{kj} | \bar{c}_i)}$ .

where we write  $p_{k\bar{i}}$  as short for  $P(w_{kx} = 1 \mid \bar{c}_i)$ . We may convert (6) and (7) into a single equation by subtracting componentwise (7) from (6), thus obtaining

$$\begin{aligned} & \log \frac{P(c_i \mid \vec{d}_j)}{1 - P(c_i \mid \vec{d}_j)} \\ &= \log \frac{P(c_i)}{1 - P(c_i)} + \sum_{k=1}^{|\mathcal{T}|} w_{kj} \log \frac{p_{ki}(1 - p_{k\bar{i}})}{p_{k\bar{i}}(1 - p_{ki})} \\ & \quad + \sum_{k=1}^{|\mathcal{T}|} \log \frac{1 - p_{ki}}{1 - p_{k\bar{i}}}. \end{aligned} \quad (8)$$

Note that  $\frac{P(c_i \mid \vec{d}_j)}{1 - P(c_i \mid \vec{d}_j)}$  is an increasing monotonic function of  $P(c_i \mid \vec{d}_j)$ , and may thus be used directly as  $CSV_i(d_j)$ . Note also that  $\log \frac{P(c_i)}{1 - P(c_i)}$  and  $\sum_{k=1}^{|\mathcal{T}|} \log \frac{1 - p_{ki}}{1 - p_{k\bar{i}}}$  are constant for all documents, and may thus be disregarded.<sup>13</sup> Defining a classifier for category  $c_i$  thus basically requires estimating the  $2|\mathcal{T}|$  parameters  $\{p_{1i}, p_{1\bar{i}}, \dots, p_{|\mathcal{T}|i}, p_{|\mathcal{T}|\bar{i}}\}$  from the training data, which may be done in the obvious way. Note that in general the classification of a given document does not require one to compute a sum of  $|\mathcal{T}|$  factors, as the presence of  $\sum_{k=1}^{|\mathcal{T}|} w_{kj} \log \frac{p_{ki}(1 - p_{k\bar{i}})}{p_{k\bar{i}}(1 - p_{ki})}$  would imply; in fact, all the factors for which  $w_{kj} = 0$  may be disregarded, and this accounts for the vast majority of them, since document vectors are usually very sparse.

The method we have illustrated is just one of the many variants of the Naïve Bayes approach, the common denominator of which is (4). A recent paper by Lewis [1998] is an excellent roadmap on the various directions that research on Naïve Bayes classifiers has taken; among these are the ones aiming

—to relax the constraint that document vectors should be binary-valued. This

looks natural, given that weighted indexing techniques (see Fuhr [1989]; Salton and Buckley [1988]) accounting for the “importance” of  $t_k$  for  $d_j$  play a key role in IR.

—to introduce document length normalization. The value of  $\log \frac{P(c_i \mid \vec{d}_j)}{1 - P(c_i \mid \vec{d}_j)}$  tends to be more extreme (i.e., very high or very low) for long documents (i.e., documents such that  $w_{kj} = 1$  for many values of  $k$ ), irrespectively of their semantic relatedness to  $c_i$ , thus calling for length normalization. Taking length into account is easy in non-probabilistic approaches to classification (see Section 6.7), but is problematic in probabilistic ones (see Lewis [1998], Section 5). One possible answer is to switch from an interpretation of Naïve Bayes in which documents are events to one in which terms are events [Baker and McCallum 1998; McCallum et al. 1998; Chakrabarti et al. 1998a; Guthrie et al. 1994]. This accounts for document length naturally but, as noted by Lewis [1998], has the drawback that different occurrences of the same word within the same document are viewed as independent, an assumption even more implausible than (4).

—to relax the independence assumption. This may be the hardest route to follow, since this produces classifiers of higher computational cost and characterized by harder parameter estimation problems [Koller and Sahami 1997]. Earlier efforts in this direction within probabilistic text search (e.g., vanRijsbergen [1977]) have not shown the performance improvements that were hoped for. Recently, the fact that the binary independence assumption seldom harms effectiveness has also been given some theoretical justification [Domingos and Pazzani 1997].

The quotation of text search in the last paragraph is not casual. Unlike other types of classifiers, the literature on probabilistic classifiers is inextricably intertwined with that on probabilistic search systems (see Crestani et al. [1998] for a

<sup>13</sup> This is not true, however, if the “fixed thresholding” method of Section 6.1 is adopted. In fact, for a fixed document  $d_j$  the first and third factor in the formula above are different for different categories, and may therefore influence the choice of the categories under which to file  $d_j$ .

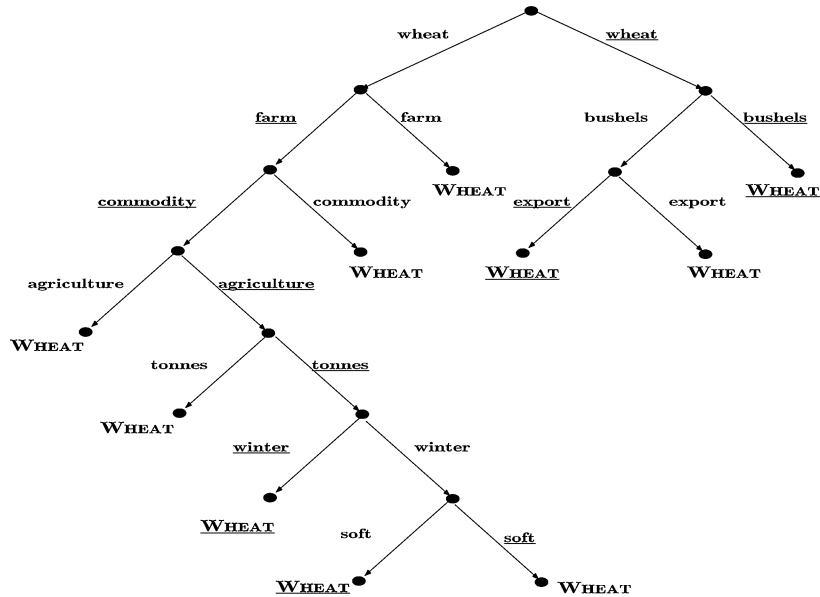


Fig. 2. A decision tree equivalent to the DNF rule of Figure 1. Edges are labeled by terms and leaves are labeled by categories (underlining denotes negation).

review), since these latter attempt to determine the probability that a document falls in the category denoted by the query, and since they are the only search systems that take *relevance feedback*, a notion essentially involving supervised learning, as central.

### 6.3. Decision Tree Classifiers

Probabilistic methods are quantitative (i.e., numeric) in nature, and as such have sometimes been criticized since, effective as they may be, they are not easily interpretable by humans. A class of algorithms that do not suffer from this problem are *symbolic* (i.e., nonnumeric) algorithms, among which inductive rule learners (which we will discuss in Section 6.4) and decision tree learners are the most important examples.

A *decision tree* (DT) text classifier (see Mitchell [1996], Chapter 3) is a tree in which internal nodes are labeled by terms, branches departing from them are labeled by tests on the weight that the term has in the test document, and leaves are labeled by categories. Such a classifier categorizes a test document  $d_j$  by recursively testing for

the weights that the terms labeling the internal nodes have in vector  $\vec{d}_j$ , until a leaf node is reached; the label of this node is then assigned to  $d_j$ . Most such classifiers use binary document representations, and thus consist of binary trees. An example DT is illustrated in Figure 2.

There are a number of standard packages for DT learning, and most DT approaches to TC have made use of one such package. Among the most popular ones are ID3 (used by Fuhr et al. [1991]), C4.5 (used by Cohen and Hirsh [1998], Cohen and Singer [1999], Joachims [1998], and Lewis and Catlett [1994]), and C5 (used by Li and Jain [1998]). TC efforts based on experimental DT packages include Dumais et al. [1998], Lewis and Ringuette [1994], and Weiss et al. [1999].

A possible method for learning a DT for category  $c_i$  consists in a “divide and conquer” strategy of (i) checking whether all the training examples have the same label (either  $c_i$  or  $\bar{c}_i$ ); (ii) if not, selecting a term  $t_k$ , partitioning  $Tr$  into classes of documents that have the same value for  $t_k$ , and placing each such class in a separate subtree. The process is recursively repeated on the subtrees until each

leaf of the tree so generated contains training examples assigned to the same category  $c_i$ , which is then chosen as the label for the leaf. The key step is the choice of the term  $t_k$  on which to operate the partition, a choice which is generally made according to an information gain or entropy criterion. However, such a “fully grown” tree may be prone to overfitting, as some branches may be too specific to the training data. Most DT learning methods thus include a method for growing the tree and one for pruning it, that is, for removing the overly specific branches. Variations on this basic schema for DT learning abound [Mitchell 1996, Section 3].

DT text classifiers have been used either as the main classification tool [Fuhr et al. 1991; Lewis and Catlett 1994; Lewis and Ringuette 1994], or as baseline classifiers [Cohen and Singer 1999; Joachims 1998], or as members of classifier committees [Li and Jain 1998; Schapire and Singer 2000; Weiss et al. 1999].

#### 6.4. Decision Rule Classifiers

A classifier for category  $c_i$  built by an *inductive rule learning* method consists of a *DNF rule*, that is, of a conditional rule with a premise in disjunctive normal form (DNF), of the type illustrated in Figure 1.<sup>14</sup> The literals (i.e., possibly negated keywords) in the premise denote the presence (nonnegated keyword) or absence (negated keyword) of the keyword in the test document  $d_j$ , while the clause head denotes the decision to classify  $d_j$  under  $c_i$ . DNF rules are similar to DTs in that they can encode any Boolean function. However, an advantage of DNF rule learners is that they tend to generate more compact classifiers than DT learners.

Rule learning methods usually attempt to select from all the possible covering rules (i.e., rules that correctly classify all the training examples) the “best” one

according to some minimality criterion. While DTs are typically built by a top-down, “divide-and-conquer” strategy, DNF rules are often built in a bottom-up fashion. Initially, every training example  $d_j$  is viewed as a clause  $\eta_1, \dots, \eta_n \rightarrow \gamma_i$ , where  $\eta_1, \dots, \eta_n$  are the terms contained in  $d_j$  and  $\gamma_i$  equals  $c_i$  or  $\bar{c}_i$  according to whether  $d_j$  is a positive or negative example of  $c_i$ . This set of clauses is already a DNF classifier for  $c_i$ , but obviously scores high in terms of overfitting. The learner applies then a process of generalization in which the rule is simplified through a series of modifications (e.g., removing premises from clauses, or merging clauses) that maximize its compactness while at the same time not affecting the “covering” property of the classifier. At the end of this process, a “pruning” phase similar in spirit to that employed in DTs is applied, where the ability to correctly classify *all* the training examples is traded for more generality.

DNF rule learners vary widely in terms of the methods, heuristics and criteria employed for generalization and pruning. Among the DNF rule learners that have been applied to TC are CHARADE [Moulinier and Ganascia 1996], DL-ESC [Li and Yamanishi 1999], RIPPER [Cohen 1995a; Cohen and Hirsh 1998; Cohen and Singer 1999], SCAR [Moulinier et al. 1996], and SWAP-1 [Apté 1994].

While the methods above use rules of propositional logic (PL), research has also been carried out using rules of first-order logic (FOL), obtainable through the use of *inductive logic programming* methods. Cohen [1995a] has extensively compared PL and FOL learning in TC (for instance, comparing the PL learner RIPPER with its FOL version FLIPPER), and has found that the additional representational power of FOL brings about only modest benefits.

#### 6.5. Regression Methods

Various TC efforts have used regression models (see Fuhr and Pfeifer [1994]; Ittner et al. [1995]; Lewis and Gale [1994]; Schütze et al. [1995]). *Regression* denotes

<sup>14</sup> Many inductive rule learning algorithms build *decision lists* (i.e., arbitrarily nested if-then-else clauses) instead of DNF rules; since the former may always be rewritten as the latter, we will disregard the issue.

the approximation of a *real-valued* (instead than binary, as in the case of classification) function  $\Phi$  by means of a function  $\hat{\Phi}$  that fits the training data [Mitchell 1996, page 236]. Here we will describe one such model, the *Linear Least-Squares Fit* (LLSF) applied to TC by Yang and Chute [1994]. In LLSF, each document  $d_j$  has two vectors associated to it: an *input vector*  $I(d_j)$  of  $|\mathcal{T}|$  weighted terms, and an *output vector*  $O(d_j)$  of  $|\mathcal{C}|$  weights representing the categories (the weights for this latter vector are binary for training documents, and are nonbinary CSV's for test documents). Classification may thus be seen as the task of determining an output vector  $O(d_j)$  for test document  $d_j$ , given its input vector  $I(d_j)$ ; hence, building a classifier boils down to computing a  $|\mathcal{C}| \times |\mathcal{T}|$  matrix  $\hat{M}$  such that  $\hat{M}I(d_j) = O(d_j)$ .

LLSF computes the matrix from the training data by computing a linear least-squares fit that minimizes the error on the training set according to the formula  $\hat{M} = \arg \min_M \|MI - O\|_F$ , where  $\arg \min_M(x)$  stands as usual for the  $M$  for which  $x$  is minimum,  $\|V\|_F \stackrel{def}{=} \sqrt{\sum_{i=1}^{|\mathcal{C}|} \sum_{j=1}^{|\mathcal{T}|} v_{ij}^2}$  represents the so-called *Frobenius norm* of a  $|\mathcal{C}| \times |\mathcal{T}|$  matrix,  $I$  is the  $|\mathcal{T}| \times |\mathcal{Tr}|$  matrix whose columns are the input vectors of the training documents, and  $O$  is the  $|\mathcal{C}| \times |\mathcal{Tr}|$  matrix whose columns are the output vectors of the training documents. The  $\hat{M}$  matrix is usually computed by performing a singular value decomposition on the training set, and its generic entry  $\hat{m}_{ik}$  represents the degree of association between category  $c_i$  and term  $t_k$ .

The experiments of Yang and Chute [1994] and Yang and Liu [1999] indicate that LLSF is one of the most effective text classifiers known to date. One of its disadvantages, though, is that the cost of computing the  $\hat{M}$  matrix is much higher than that of many other competitors in the TC arena.

## 6.6. On-Line Methods

A *linear classifier* for category  $c_i$  is a vector  $\vec{c}_i = \langle w_{1i}, \dots, w_{|\mathcal{T}|i} \rangle$  belonging to the

same  $|\mathcal{T}|$ -dimensional space in which documents are also represented, and such that  $CSV_i(d_j)$  corresponds to the dot product  $\sum_{k=1}^{|\mathcal{T}|} w_{ki}w_{kj}$  of  $\vec{d}_j$  and  $\vec{c}_i$ . Note that when both classifier and document weights are cosine-normalized (see (2)), the dot product between the two vectors corresponds to their *cosine similarity*, that is:

$$\begin{aligned} S(c_i, d_j) &= \cos(\alpha) \\ &= \frac{\sum_{k=1}^{|\mathcal{T}|} w_{ki} \cdot w_{kj}}{\sqrt{\sum_{k=1}^{|\mathcal{T}|} w_{ki}^2} \cdot \sqrt{\sum_{k=1}^{|\mathcal{T}|} w_{kj}^2}}, \end{aligned}$$

which represents the cosine of the angle  $\alpha$  that separates the two vectors. This is the similarity measure between query and document computed by standard vector-space IR engines, which means in turn that once a linear classifier has been built, classification can be performed by invoking such an engine. Practically all search engines have a dot product flavor to them, and can therefore be adapted to doing TC with a linear classifier.

Methods for learning linear classifiers are often partitioned in two broad classes, batch methods and on-line methods.

*Batch methods* build a classifier by analyzing the training set all at once. Within the TC literature, one example of a batch method is *linear discriminant analysis*, a model of the stochastic dependence between terms that relies on the covariance matrices of the categories [Hull 1994; Schütze et al. 1995]. However, the foremost example of a batch method is the Rocchio method; because of its importance in the TC literature, this will be discussed separately in Section 6.7. In this section we will instead concentrate on on-line methods.

*On-line* (a.k.a. *incremental*) *methods* build a classifier soon after examining the first training document, and incrementally refine it as they examine new ones. This may be an advantage in the applications in which  $\mathcal{Tr}$  is not available in its entirety from the start, or in which the “meaning” of the category may change in time, as for example, in adaptive



filtering. This is also apt to applications (e.g., semiautomated classification, adaptive filtering) in which we may expect the user of a classifier to provide feedback on how test documents have been classified, as in this case further training may be performed during the operating phase by exploiting user feedback.

A simple on-line method is the *perceptron* algorithm, first applied to TC by Schütze et al. [1995] and Wiener et al. [1995], and subsequently used by Dagan et al. [1997] and Ng et al. [1997]. In this algorithm, the classifier for  $c_i$  is first initialized by setting all weights  $w_{ki}$  to the same positive value. When a training example  $d_j$  (represented by a vector  $\vec{d}_j$  of binary weights) is examined, the classifier built so far classifies it. If the result of the classification is correct, nothing is done, while if it is wrong, the weights of the classifier are modified: if  $d_j$  was a positive example of  $c_i$ , then the weights  $w_{ki}$  of “active terms” (i.e., the terms  $t_k$  such that  $w_{kj} = 1$ ) are “promoted” by increasing them by a fixed quantity  $\alpha > 0$  (called *learning rate*), while if  $d_j$  was a negative example of  $c_i$  then the same weights are “demoted” by decreasing them by  $\alpha$ . Note that when the classifier has reached a reasonable level of effectiveness, the fact that a weight  $w_{ki}$  is very low means that  $t_k$  has negatively contributed to the classification process so far, and may thus be discarded from the representation. We may then see the perceptron algorithm (as all other incremental learning methods) as allowing for a sort of “on-the-fly term space reduction” [Dagan et al. 1997, Section 4.4]. The perceptron classifier has shown a good effectiveness in all the experiments quoted above.

The perceptron is an *additive weight-updating* algorithm. A *multiplicative* variant of it is POSITIVE WINNOWER [Dagan et al. 1997], which differs from perceptron because two different constants  $\alpha_1 > 1$  and  $0 < \alpha_2 < 1$  are used for promoting and demoting weights, respectively, and because promotion and demotion are achieved by multiplying, instead of adding, by  $\alpha_1$  and  $\alpha_2$ . BALANCED WINNOWER [Dagan et al. 1997] is a further variant of POSITIVE WINNOWER, in which the classifier consists of *two* weights

$w_{ki}^+$  and  $w_{ki}^-$  for each term  $t_k$ ; the final weight  $w_{ki}$  used in computing the dot product is the difference  $w_{ki}^+ - w_{ki}^-$ . Following the misclassification of a positive instance, active terms have their  $w_{ki}^+$  weight promoted and their  $w_{ki}^-$  weight demoted, whereas in the case of a negative instance it is  $w_{ki}^+$  that gets demoted while  $w_{ki}^-$  gets promoted (for the rest, promotions and demotions are as in POSITIVE WINNOWER). BALANCED WINNOWER allows negative  $w_{ki}$  weights, while in the perceptron and in POSITIVE WINNOWER the  $w_{ki}$  weights are always positive. In experiments conducted by Dagan et al. [1997], POSITIVE WINNOWER showed a better effectiveness than perceptron but was in turn outperformed by (Dagan et al.’s own version of) BALANCED WINNOWER.

Other on-line methods for building text classifiers are WIDROW-HOFF, a refinement of it called EXPONENTIATED GRADIENT (both applied for the first time to TC in [Lewis et al. 1996]) and SLEEPING EXPERTS [Cohen and Singer 1999], a version of BALANCED WINNOWER. While the first is an additive weight-updating algorithm, the second and third are multiplicative. Key differences with the previously described algorithms are that these three algorithms (i) update the classifier not only after misclassifying a training example, but also after classifying it correctly, and (ii) update the weights corresponding to all terms (instead of just active ones).

Linear classifiers lend themselves to both category-pivoted and document-pivoted TC. For the former the classifier  $\vec{c}_i$  is used, in a standard search engine, as a query against the set of test documents, while for the latter the vector  $\vec{d}_j$  representing the test document is used as a query against the set of classifiers  $\{\vec{c}_1, \dots, \vec{c}_{|C|}\}$ .

## 6.7. The Rocchio Method

Some linear classifiers consist of an explicit *profile* (or prototypical document) of the category. This has obvious advantages in terms of interpretability, as such a profile is more readily understandable by a human than, say, a neural network

classifier. Learning a linear classifier is often preceded by local TSR; in this case, a profile of  $c_i$  is a weighted list of the terms whose presence or absence is most useful for discriminating  $c_i$ .

The *Rocchio method* is used for inducing linear, profile-style classifiers. It relies on an adaptation to TC of the well-known Rocchio's formula for relevance feedback in the vector-space model, and it is perhaps the only TC method rooted in the IR tradition rather than in the ML one. This adaptation was first proposed by Hull [1994], and has been used by many authors since then, either as an object of research in its own right [Ittner et al. 1995; Joachims 1997; Sable and Hatzivassiloglou 2000; Schapire et al. 1998; Singhal et al. 1997], or as a baseline classifier [Cohen and Singer 1999; Galavotti et al. 2000; Joachims 1998; Lewis et al. 1996; Schapire and Singer 2000; Schütze et al. 1995], or as a member of a classifier committee [Larkey and Croft 1996] (see Section 6.11).

Rocchio's method computes a classifier  $\bar{c}_i = \langle w_{1i}, \dots, w_{|T|i} \rangle$  for category  $c_i$  by means of the formula

$$w_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{w_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NEG_i\}} \frac{w_{kj}}{|NEG_i|},$$

where  $w_{kj}$  is the weight of  $t_k$  in document  $d_j$ ,  $POS_i = \{d_j \in Tr \mid \Phi(d_j, c_i) = T\}$ , and  $NEG_i = \{d_j \in Tr \mid \Phi(d_j, c_i) = F\}$ . In this formula,  $\beta$  and  $\gamma$  are control parameters that allow setting the relative importance of positive and negative examples. For instance, if  $\beta$  is set to 1 and  $\gamma$  to 0 (as in Dumais et al. [1998]; Hull [1994]; Joachims [1998]; Schütze et al. [1995]), the profile of  $c_i$  is the *centroid* of its positive training examples. A classifier built by means of the Rocchio method rewards the closeness of a test document to the centroid of the positive training examples, and its distance from the centroid of the negative training examples. The role of negative examples is usually deempha-

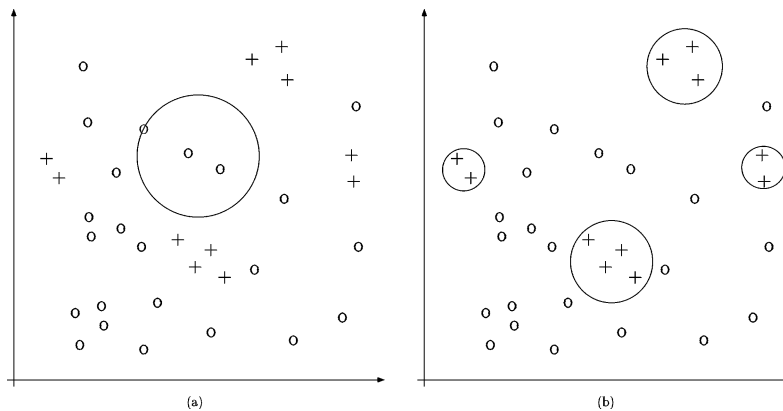
sized, by setting  $\beta$  to a high value and  $\gamma$  to a low one (e.g., Cohen and Singer [1999], Ittner et al. [1995], and Joachims [1997] use  $\beta = 16$  and  $\gamma = 4$ ).

This method is quite easy to implement, and is also quite efficient, since learning a classifier basically comes down to averaging weights. In terms of effectiveness, instead, a drawback is that if the documents in the category tend to occur in disjoint clusters (e.g., a set of newspaper articles labeled with the Sports category and dealing with either boxing or rock-climbing), such a classifier may miss most of them, as the centroid of these documents may fall outside all of these clusters (see Figure 3(a)). More generally, a classifier built by the Rocchio method, as all linear classifiers, has the disadvantage that it divides the space of documents linearly. This situation is graphically depicted in Figure 3(a), where documents are classified within  $c_i$  if and only if they fall within the circle. Note that even most of the positive training examples would not be classified correctly by the classifier.

**6.7.1. Enhancements to the Basic Rocchio Framework.** One issue in the application of the Rocchio formula to profile extraction is whether the set  $NEG_i$  should be considered in its entirety, or whether a well-chosen sample of it, such as the set  $NPOS_i$  of *near-positives* (defined as “the most positive among the negative training examples”), should be selected from it, yielding

$$w_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{w_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NPOS_i\}} \frac{w_{kj}}{|NPOS_i|}.$$

The  $\sum_{\{d_j \in NPOS_i\}} \frac{w_{kj}}{|NPOS_i|}$  factor is more significant than  $\sum_{\{d_j \in NEG_i\}} \frac{w_{kj}}{|NEG_i|}$ , since near-positives are the most difficult documents to tell apart from the positives. Using near-positives corresponds to the *query zoning* method proposed for IR by Singhal et al. [1997]. This method originates from the observation that, when the original



**Fig. 3.** A comparison between the TC behavior of (a) the Rocchio classifier, and (b) the  $k$ -NN classifier. Small crosses and circles denote positive and negative training instances, respectively. The big circles denote the “influence area” of the classifier. Note that, for ease of illustration, document similarities are here viewed in terms of Euclidean distance rather than, as is more common, in terms of dot product or cosine.

Rocchio formula is used for relevance feedback in IR, near-positives tend to be used rather than generic negatives, as the documents on which user judgments are available are among the ones that had scored highest in the previous ranking. Early applications of the Rocchio formula to TC (e.g., Hull [1994]; Ittner et al. [1995]) generally did not make a distinction between near-positives and generic negatives. In order to select the near-positives Schapire et al. [1998] issue a query, consisting of the centroid of the positive training examples, against a document base consisting of the negative training examples; the top-ranked ones are the most similar to this centroid, and are then the near-positives. Wiener et al. [1995] instead equate the near-positives of  $c_i$  to the positive examples of the *sibling* categories of  $c_i$ , as in the application they work on (TC with hierarchically organized category sets) the notion of a “sibling category of  $c_i$ ” is well defined. A similar policy is also adopted by Ng et al. [1997], Ruiz and Srinivasan [1999], and Weigend et al. [1999].

By using query zoning plus other enhancements (TSR, statistical phrases, and a method called *dynamic feedback optimization*), Schapire et al. [1998] have found that a Rocchio classifier can achieve

an effectiveness comparable to that of a state-of-the-art ML method such as “boosting” (see Section 6.11.1) while being 60 times quicker to train. These recent results will no doubt bring about a renewed interest for the Rocchio classifier, previously considered an underperformer [Cohen and Singer 1999; Joachims 1998; Lewis et al. 1996; Schütze et al. 1995; Yang 1999].

## 6.8. Neural Networks

A *neural network* (NN) text classifier is a network of units, where the input units represent terms, the output unit(s) represent the category or categories of interest, and the weights on the edges connecting units represent dependence relations. For classifying a test document  $d_j$ , its term weights  $w_{kj}$  are loaded into the input units; the activation of these units is propagated forward through the network, and the value of the output unit(s) determines the categorization decision(s). A typical way of training NNs is backpropagation, whereby the term weights of a training document are loaded into the input units, and if a misclassification occurs the error is “backpropagated” so as to change the parameters of the network and eliminate or minimize the error.

The simplest type of NN classifier is the perceptron [Dagan et al. 1997; Ng et al. 1997], which is a linear classifier and as such has been extensively discussed in Section 6.6. Other types of linear NN classifiers implementing a form of logistic regression have also been proposed and tested by Schütze et al. [1995] and Wiener et al. [1995], yielding very good effectiveness.

A nonlinear NN [Lam and Lee 1999; Ruiz and Srinivasan 1999; Schütze et al. 1995; Weigend et al. 1999; Wiener et al. 1995; Yang and Liu 1999] is instead a network with one or more additional “layers” of units, which in TC usually represent higher-order interactions between terms that the network is able to learn. When comparative experiments relating nonlinear NNs to their linear counterparts have been performed, the former have yielded either no improvement [Schütze et al. 1995] or very small improvements [Wiener et al. 1995] over the latter.

### 6.9. Example-Based Classifiers

Example-based classifiers do not build an explicit, declarative representation of the category  $c_i$ , but rely on the category labels attached to the training documents similar to the test document. These methods have thus been called *lazy learners*, since “they defer the decision on how to generalize beyond the training data until each new query instance is encountered” [Mitchell 1996, page 244].

The first application of example-based methods (a.k.a. *memory-based reasoning methods*) to TC is due to Creecy, Masand and colleagues [Creecy et al. 1992; Masand et al. 1992]; other examples include Joachims [1998], Lam et al. [1999], Larkey [1998], Larkey [1999], Li and Jain [1998], Yang and Pedersen [1997], and Yang and Liu [1999]. Our presentation of the example-based approach will be based on the *k-NN* (for “ $k$  nearest neighbors”) algorithm used by Yang [1994]. For deciding whether  $d_j \in c_i$ , *k-NN* looks at whether the  $k$  training documents most similar to  $d_j$  also are in  $c_i$ ; if the answer is positive for a large enough proportion of them,

a positive decision is taken, and a negative decision is taken otherwise. Actually, Yang’s is a *distance-weighted* version of *k-NN* (see [Mitchell 1996, Section 8.2.1]), since the fact that a most similar document is in  $c_i$  is weighted by its similarity with the test document. Classifying  $d_j$  by means of *k-NN* thus comes down to computing

$$\begin{aligned} CSV_i(d_j) &= \sum_{d_z \in Tr_k(d_j)} RSV(d_j, d_z) \cdot \llbracket \Phi(d_z, c_i) \rrbracket, \end{aligned} \quad (9)$$

where  $Tr_k(d_j)$  is the set of the  $k$  documents  $d_z$  which maximize  $RSV(d_j, d_z)$  and

$$\llbracket \alpha \rrbracket = \begin{cases} 1 & \text{if } \alpha = T \\ 0 & \text{if } \alpha = F \end{cases}.$$

The thresholding methods of Section 6.1 can then be used to convert the real-valued  $CSV_i$ ’s into binary categorization decisions. In (9),  $RSV(d_j, d_z)$  represents some measure or semantic relatedness between a test document  $d_j$  and a training document  $d_z$ ; any matching function, be it probabilistic (as used by Larkey and Croft [1996]) or vector-based (as used by Yang [1994]), from a ranked IR system may be used for this purpose. The construction of a *k-NN* classifier also involves determining (experimentally, on a validation set) a threshold  $k$  that indicates how many top-ranked training documents have to be considered for computing  $CSV_i(d_j)$ . Larkey and Croft [1996] used  $k = 20$ , while Yang [1994, 1999] has found  $30 \leq k \leq 45$  to yield the best effectiveness. Anyhow, various experiments have shown that increasing the value of  $k$  does not significantly degrade the performance.

Note that *k-NN*, unlike linear classifiers, does not divide the document space linearly, and hence does not suffer from the problem discussed at the end of Section 6.7. This is graphically depicted in Figure 3(b), where the more “local” character of *k-NN* with respect to Rocchio can be appreciated.

This method is naturally geared toward document-pivoted TC, since ranking the training documents for their similarity with the test document can be done once for all categories. For category-pivoted TC, one would need to store the document ranks for each test document, which is obviously clumsy; DPC is thus *de facto* the only reasonable way to use  $k$ -NN.

A number of different experiments (see Section 7.3) have shown  $k$ -NN to be quite effective. However, its most important drawback is its inefficiency at classification time: while, for example, with a linear classifier only a dot product needs to be computed to classify a test document,  $k$ -NN requires the entire training set to be ranked for similarity with the test document, which is much more expensive. This is a drawback of “lazy” learning methods, since they do not have a true training phase and thus defer all the computation to classification time.

*6.9.1. Other Example-Based Techniques.* Various example-based techniques have been used in the TC literature. For example, Cohen and Hirsh [1998] implemented an example-based classifier by extending standard relational DBMS technology with “similarity-based soft joins.” In their WHIRL system they used the scoring function

$$CSV_i(d_j) = 1 - \prod_{d_z \in Tr_k(d_j)} (1 - RSV(d_j, d_z))^{\llbracket \Phi(d_z, c_i) \rrbracket}$$

as an alternative to (9), obtaining a small but statistically significant improvement over a version of WHIRL using (9). In their experiments this technique outperformed a number of other classifiers, such as a C4.5 decision tree classifier and the RIPPER CNF rule-based classifier.

A variant of the basic  $k$ -NN approach was proposed by Galavotti et al. [2000], who reinterpreted (9) by redefining  $\llbracket \alpha \rrbracket$  as

$$\llbracket \alpha \rrbracket = \begin{cases} 1 & \text{if } \alpha = T \\ -1 & \text{if } \alpha = F \end{cases}$$

The difference from the original  $k$ -NN approach is that if a training document  $d_z$  similar to the test document  $d_j$  does not belong to  $c_i$ , this information is not discarded but weights *negatively* in the decision to classify  $d_j$  under  $c_i$ .

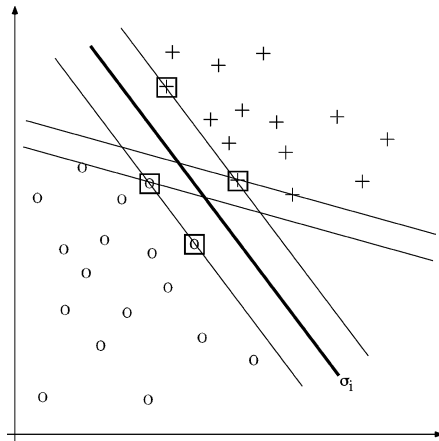
A combination of profile- and example-based methods was presented in Lam and Ho [1998]. In this work a  $k$ -NN system was fed *generalized instances* (GIs) in place of training documents. This approach may be seen as the result of

- clustering the training set, thus obtaining a set of clusters  $K_i = \{k_{i1}, \dots, k_{i|K_i|}\}$ ;
- building a profile  $G(k_{iz})$  (“generalized instance”) from the documents belonging to cluster  $k_{iz}$  by means of some algorithm for learning linear classifiers (e.g., Rocchio, WIDROW-HOFF);
- applying  $k$ -NN with profiles in place of training documents, that is, computing

$$\begin{aligned} CSV_i(d_j) &\stackrel{def}{=} \sum_{k_{iz} \in K_i} RSV(d_j, G(k_{iz})) \cdot \\ &\frac{|\{d_j \in k_{iz} \mid \check{\Phi}(d_j, c_i) = T\}|}{|\{d_j \in k_{iz}\}|} \cdot \\ &\frac{|\{d_j \in k_{iz}\}|}{|Tr|} \\ &= \sum_{k_{iz} \in K_i} RSV(d_j, G(k_{iz})) \cdot \\ &\frac{|\{d_j \in k_{iz} \mid \check{\Phi}(d_j, c_i) = T\}|}{|Tr|}, \end{aligned} \quad (10)$$

where  $\frac{|\{d_j \in k_{iz} \mid \check{\Phi}(d_j, c_i) = T\}|}{|\{d_j \in k_{iz}\}|}$  represents the “degree” to which  $G(k_{iz})$  is a positive instance of  $c_i$ , and  $\frac{|\{d_j \in k_{iz}\}|}{|Tr|}$  represents its weight within the entire process.

This exploits the superior effectiveness (see Figure 3) of  $k$ -NN over linear classifiers while at the same time avoiding the sensitivity of  $k$ -NN to the presence of “outliers” (i.e., positive instances of  $c_i$  that “lie out” of the region where most other positive instances of  $c_i$  are located) in the training set.



**Fig. 4.** Learning support vector classifiers. The small crosses and circles represent positive and negative training examples, respectively, whereas lines represent decision surfaces. Decision surface  $\sigma_i$  (indicated by the thicker line) is, among those shown, the best possible one, as it is the middle element of the widest set of parallel decision surfaces (i.e., its minimum distance to any training example is maximum). Small boxes indicate the support vectors.

#### 6.10. Building Classifiers by Support Vector Machines

The *support vector machine* (SVM) method has been introduced in TC by Joachims [1998, 1999] and subsequently used by Drucker et al. [1999], Dumais et al. [1998], Dumais and Chen [2000], Klinkenberg and Joachims [2000], Taira and Haruno [1999], and Yang and Liu [1999]. In geometrical terms, it may be seen as the attempt to find, among all the surfaces  $\sigma_1, \sigma_2, \dots$  in  $|\mathcal{T}|$ -dimensional space that separate the positive from the negative training examples (*decision surfaces*), the  $\sigma_i$  that separates the positives from the negatives by the widest possible margin, that is, such that the separation property is invariant with respect to the widest possible translation of  $\sigma_i$ .

This idea is best understood in the case in which the positives and the negatives are linearly separable, in which case the decision surfaces are  $(|\mathcal{T}|-1)$ -hyperplanes. In the two-dimensional case of Figure 4, various lines may be chosen as decision surfaces. The SVM method chooses the

middle element from the “widest” set of parallel lines, that is, from the set in which the maximum distance between two elements in the set is highest. It is noteworthy that this “best” decision surface is determined by only a small set of training examples, called the *support vectors*.

The method described is applicable also to the case in which the positives and the negatives are not linearly separable. Yang and Liu [1999] experimentally compared the linear case (namely, when the assumption is made that the categories are linearly separable) with the nonlinear case on a standard benchmark, and obtained slightly better results in the former case.

As argued by Joachims [1998], SVMs offer two important advantages for TC:

- term selection is often not needed, as SVMs tend to be fairly robust to overfitting and can scale up to considerable dimensionalities;
- no human and machine effort in parameter tuning on a validation set is needed, as there is a theoretically motivated, “default” choice of parameter settings, which has also been shown to provide the best effectiveness.

Dumais et al. [1998] have applied a novel algorithm for training SVMs which brings about training speeds comparable to computationally easy learners such as Rocchio.

#### 6.11. Classifier Committees

Classifier *committees* (a.k.a. *ensembles*) are based on the idea that, given a task that requires expert knowledge to perform,  $k$  experts may be better than one if their individual judgments are appropriately combined. In TC, the idea is to apply  $k$  different classifiers  $\Phi_1, \dots, \Phi_k$  to the same task of deciding whether  $d_j \in c_i$ , and then combine their outcome appropriately. A classifier committee is then characterized by (i) a choice of  $k$  classifiers, and (ii) a choice of a combination function.

Concerning Issue (i), it is known from the ML literature that, in order to guarantee good effectiveness, the classifiers

forming the committee should be as independent as possible [Tumer and Ghosh 1996]. The classifiers may differ for the indexing approach used, or for the inductive method, or both. Within TC, the avenue which has been explored most is the latter (to our knowledge the only example of the former is Scott and Matwin [1999]).

Concerning Issue (ii), various rules have been tested. The simplest one is *majority voting* (MV), whereby the binary outputs of the  $k$  classifiers are pooled together, and the classification decision that reaches the majority of  $\frac{k+1}{2}$  votes is taken ( $k$  obviously needs to be an odd number) [Li and Jain 1998; Liere and Tadepalli 1997]. This method is particularly suited to the case in which the committee includes classifiers characterized by a binary decision function  $CSV_i : \mathcal{D} \rightarrow \{T, F\}$ . A second rule is *weighted linear combination* (WLC), whereby a weighted sum of the  $CSV_i$ 's produced by the  $k$  classifiers yields the final  $CSV_i$ . The weights  $w_j$  reflect the expected relative effectiveness of classifiers  $\Phi_j$ , and are usually optimized on a validation set [Larkey and Croft 1996]. Another policy is *dynamic classifier selection* (DCS), whereby among committee  $\{\Phi_1, \dots, \Phi_k\}$  the classifier  $\Phi_l$  most effective on the  $l$  validation examples most similar to  $d_j$  is selected, and its judgment adopted by the committee [Li and Jain 1998]. A still different policy, somehow intermediate between WLC and DCS, is *adaptive classifier combination* (ACC), whereby the judgments of *all* the classifiers in the committee are summed together, but their individual contribution is weighted by their effectiveness on the  $l$  validation examples most similar to  $d_j$  [Li and Jain 1998].

Classifier committees have had mixed results in TC so far. Larkey and Croft [1996] have used combinations of Rocchio, Naïve Bayes, and  $k$ -NN, all together or in pairwise combinations, using a WLC rule. In their experiments the combination of any two classifiers outperformed the best individual classifier ( $k$ -NN), and the combination of the three classifiers improved an all three pairwise combinations. These results would seem to give strong support to the idea that classifier committees

can somehow profit from the complementary strengths of their individual members. However, the small size of the test set used (187 documents) suggests that more experimentation is needed before conclusions can be drawn.

Li and Jain [1998] have tested a committee formed of (various combinations of) a Naïve Bayes classifier, an example-based classifier, a decision tree classifier, and a classifier built by means of their own "subspace method"; the combination rules they have worked with are MV, DCS, and ACC. Only in the case of a committee formed by Naïve Bayes and the subspace classifier combined by means of ACC has the committee outperformed, and by a narrow margin, the best individual classifier (for every attempted classifier combination ACC gave better results than MV and DCS). This seems discouraging, especially in light of the fact that the committee approach is computationally expensive (its cost trivially amounts to the sum of the costs of the individual classifiers plus the cost incurred for the computation of the combination rule). Again, it has to be remarked that the small size of their experiment (two test sets of less than 700 documents each were used) does not allow us to draw definitive conclusions on the approaches adopted.

*6.11.1. Boosting.* The *boosting* method [Schapire et al. 1998; Schapire and Singer 2000] occupies a special place in the classifier committees literature, since the  $k$  classifiers  $\Phi_1, \dots, \Phi_k$  forming the committee are obtained by the *same* learning method (here called the *weak learner*). The key intuition of boosting is that the  $k$  classifiers should be trained not in a conceptually parallel and independent way, as in the committees described above, but sequentially. In this way, in training classifier  $\Phi_i$  one may take into account how classifiers  $\Phi_1, \dots, \Phi_{i-1}$  perform on the training examples, and concentrate on getting right those examples on which  $\Phi_1, \dots, \Phi_{i-1}$  have performed worst.

Specifically, for learning classifier  $\Phi_t$  each  $\langle d_j, c_i \rangle$  pair is given an "importance weight"  $h_{ij}^t$  (where  $h_{ij}^1$  is set to be equal for

all  $\langle d_j, c_i \rangle$  pairs<sup>15</sup>), which represents how hard to get a correct decision for this pair was for classifiers  $\Phi_1, \dots, \Phi_{t-1}$ . These weights are exploited in learning  $\Phi_t$ , which will be specially tuned to correctly solve the pairs with higher weight. Classifier  $\Phi_t$  is then applied to the training documents, and as a result weights  $h_{ij}^t$  are updated to  $h_{ij}^{t+1}$ ; in this update operation, pairs correctly classified by  $\Phi_t$  will have their weight decreased, while pairs misclassified by  $\Phi_t$  will have their weight increased. After all the  $k$  classifiers have been built, a weighted linear combination rule is applied to yield the final committee.

In the BOOSTEXTER system [Schapire and Singer 2000], two different boosting algorithms are tested, using a one-level decision tree weak learner. The former algorithm (ADABOOST.MH, simply called ADABOOST in Schapire et al. [1998]) is explicitly geared toward the maximization of microaveraged effectiveness, whereas the latter (ADABOOST.MR) is aimed at minimizing *ranking loss* (i.e., at getting a correct category ranking for each individual document). In experiments conducted over three different test collections, Schapire et al. [1998] have shown ADABOOST to outperform SLEEPING EXPERTS, a classifier that had proven quite effective in the experiments of Cohen and Singer [1999]. Further experiments by Schapire and Singer [2000] showed ADABOOST to outperform, aside from SLEEPING EXPERTS, a Naive Bayes classifier, a standard (nonenhanced) Rocchio classifier, and Joachims' [1997] PRTFIDF classifier.

A boosting algorithm based on a "committee of classifier subcommittees" that improves on the effectiveness and (especially) the efficiency of ADABOOST.MH was presented in Sebastiani et al. [2000]. An approach similar to boosting was also employed by Weiss et al. [1999], who experimented with committees of decision trees each having an average of 16 leaves (and hence much more complex than the sim-

ple "decision stumps" used in Schapire and Singer [2000]), eventually combined by using the simple MV rule as a combination rule; similarly to boosting, a mechanism for emphasising documents that have been misclassified by previous decision trees is used. Boosting-based approaches have also been employed in Escudero et al. [2000], Iyer et al. [2000], Kim et al. [2000], Li and Jain [1998], and Myers et al. [2000].

## 6.12. Other Methods

Although in the previous sections we have tried to give an overview as complete as possible of the learning approaches proposed in the TC literature, it is hardly possible to be exhaustive. Some of the learning approaches adopted do not fall squarely under one or the other class of algorithms, or have remained somehow isolated attempts. Among these, the most noteworthy are the ones based on *Bayesian inference networks* [Dumais et al. 1998; Lam et al. 1997; Tzeras and Hartmann 1993], *genetic algorithms* [Clack et al. 1997; Masand 1994], and *maximum entropy modelling* [Manning and Schütze 1999].

## 7. EVALUATION OF TEXT CLASSIFIERS

As for text search systems, the evaluation of document classifiers is typically conducted *experimentally*, rather than analytically. The reason is that, in order to evaluate a system analytically (e.g., proving that the system is correct and complete), we would need a formal specification of the problem that the system is trying to solve (e.g., *with respect to what* correctness and completeness are defined), and the central notion of TC (namely, that of membership of a document in a category) is, due to its subjective character, inherently nonformalizable.

The experimental evaluation of a classifier usually measures its *effectiveness* (rather than its efficiency), that is, its ability to take the *right* classification decisions.

<sup>15</sup> Schapire et al. [1998] also showed that a simple modification of this policy allows optimization of the classifier based on "utility" (see Section 7.1.3).



**Table II.** The Contingency Table for Category  $c_i$ 

Category $c_i$		Expert judgments	
		YES	NO
Classifier Judgments	YES	$TP_i$	$FP_i$
	NO	$FN_i$	$TN_i$

### 7.1. Measures of Text Categorization Effectiveness

**7.1.1. Precision and Recall.** Classification effectiveness is usually measured in terms of the classic IR notions of precision ( $\pi$ ) and recall ( $\rho$ ), adapted to the case of TC. *Precision wrt  $c_i$*  ( $\pi_i$ ) is defined as the conditional probability  $P(\Phi(d_x, c_i) = T \mid \Phi(d_x, c_i) = T)$ , that is, as the probability that if a random document  $d_x$  is classified under  $c_i$ , this decision is correct. Analogously, *recall wrt  $c_i$*  ( $\rho_i$ ) is defined as  $P(\Phi(d_x, c_i) = T \mid \Phi(d_x, c_i) = T)$ , that is, as the probability that, if a random document  $d_x$  ought to be classified under  $c_i$ , this decision is taken. These category-relative values may be averaged, in a way to be discussed shortly, to obtain  $\pi$  and  $\rho$ , that is, values global to the entire category set. Borrowing terminology from logic,  $\pi$  may be viewed as the “degree of soundness” of the classifier wrt  $\mathcal{C}$ , while  $\rho$  may be viewed as its “degree of completeness” wrt  $\mathcal{C}$ . As defined here,  $\pi_i$  and  $\rho_i$  are to be understood as *subjective* probabilities, that is, as measuring the expectation of the user that the system will behave correctly when classifying an unseen document under  $c_i$ . These probabilities may be estimated in terms of the *contingency table* for  $c_i$  on a given test set (see Table II). Here,  $FP_i$  (*false positives wrt  $c_i$* , a.k.a. *errors of commission*) is the number of test documents incorrectly classified under  $c_i$ ;  $TN_i$  (*true negatives wrt  $c_i$* ),  $TP_i$  (*true positives wrt  $c_i$* ), and  $FN_i$  (*false negatives wrt  $c_i$* , a.k.a. *errors of omission*) are defined accordingly. Estimates (indicated by carets) of precision wrt  $c_i$  and recall wrt  $c_i$  may thus be obtained as

$$\hat{\pi}_i = \frac{TP_i}{TP_i + FP_i}, \quad \hat{\rho}_i = \frac{TP_i}{TP_i + FN_i}.$$

For obtaining estimates of  $\pi$  and  $\rho$ , two different methods may be adopted:

**Table III.** The Global Contingency Table

Category set $\mathcal{C} = \{c_1, \dots, c_{ \mathcal{C} }\}$		Expert judgments	
		YES	NO
Classifier	YES	$TP = \sum_{i=1}^{ \mathcal{C} } TP_i$	$FP = \sum_{i=1}^{ \mathcal{C} } FP_i$
	Judgments	NO	$FN = \sum_{i=1}^{ \mathcal{C} } FN_i$

—*microaveraging*:  $\pi$  and  $\rho$  are obtained by summing over all individual decisions:

$$\hat{\pi}^\mu = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|} (TP_i + FP_i)},$$

$$\hat{\rho}^\mu = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|} (TP_i + FN_i)},$$

where “ $\mu$ ” indicates microaveraging. The “global” contingency table (Table III) is thus obtained by summing over category-specific contingency tables;

—*macroaveraging*: precision and recall are first evaluated “locally” for each category, and then “globally” by averaging over the results of the different categories:

$$\hat{\pi}^M = \frac{\sum_{i=1}^{|\mathcal{C}|} \hat{\pi}_i}{|\mathcal{C}|}, \quad \hat{\rho}^M = \frac{\sum_{i=1}^{|\mathcal{C}|} \hat{\rho}_i}{|\mathcal{C}|},$$

where “ $M$ ” indicates macroaveraging.

These two methods may give quite different results, especially if the different categories have very different generality. For instance, the ability of a classifier to behave well also on categories with low generality (i.e., categories with few positive training instances) will be emphasized by macroaveraging and much less so by microaveraging. Whether one or the other should be used obviously depends on the application requirements. From now on, we will assume that microaveraging is used; everything we will say in the rest of Section 7 may be adapted to the case of macroaveraging in the obvious way.

**7.1.2. Other Measures of Effectiveness.** Measures alternative to  $\pi$  and  $\rho$  and commonly used in the ML literature, such as *accuracy* (estimated as

$\hat{A} = \frac{TP+TN}{TP+TN+FP+FN}$ ) and *error* (estimated as  $\hat{E} = \frac{FP+FN}{TP+TN+FP+FN} = 1 - \hat{A}$ ), are not widely used in TC. The reason is that, as Yang [1999] pointed out, the large value that their denominator typically has in TC makes them much more insensitive to variations in the number of correct decisions ( $TP + TN$ ) than  $\pi$  and  $\rho$ . Besides, if  $A$  is the adopted evaluation measure, in the frequent case of a very low average generality the *trivial rejector* (i.e., the classifier  $\Phi$  such that  $\Phi(d_j, c_i) = F$  for all  $d_j$  and  $c_i$ ) tends to outperform all nontrivial classifiers (see also Cohen [1995a], Section 2.3). If  $A$  is adopted, parameter tuning on a validation set may thus result in parameter choices that make the classifier behave very much like the trivial rejector.

A nonstandard effectiveness measure was proposed by Sable and Hatzivassiloglou [2000, Section 7], who suggested basing  $\pi$  and  $\rho$  not on “absolute” values of success and failure (i.e., 1 if  $\Phi(d_j, c_i) = \check{\Phi}(d_j, c_i)$  and 0 if  $\Phi(d_j, c_i) \neq \check{\Phi}(d_j, c_i)$ ), but on values of *relative success* (i.e.,  $CSV_i(d_j)$  if  $\check{\Phi}(d_j, c_i) = T$  and  $1 - CSV_i(d_j)$  if  $\check{\Phi}(d_j, c_i) = F$ ). This means that for a correct (respectively wrong) decision the classifier is rewarded (respectively penalized) proportionally to its confidence in the decision. This proposed measure does not reward the choice of a good thresholding policy, and is thus unfit for autonomous (“hard”) classification systems. However, it might be appropriate for interactive (“ranking”) classifiers of the type used in Larkey [1999], where the confidence that the classifier has in its own decision influences category ranking and, as a consequence, the overall usefulness of the system.

### 7.1.3. Measures Alternative to Effectiveness.

In general, criteria different from effectiveness are seldom used in classifier evaluation. For instance, *efficiency*, although very important for applicative purposes, is seldom used as the sole yardstick, due to the volatility of the parameters on which the evaluation rests. However, efficiency may be useful for choosing among

**Table IV.** The Utility Matrix

Category set $\mathcal{C} = \{c_1, \dots, c_{ \mathcal{C} }\}$		Expert judgments	
		YES	NO
Classifier	YES	$u_{TP}$	$u_{FP}$
Judgments	NO	$u_{FN}$	$u_{TN}$

classifiers with similar effectiveness. An interesting evaluation has been carried out by Dumais et al. [1998], who have compared five different learning methods along three different dimensions, namely, effectiveness, *training efficiency* (i.e., the average time it takes to build a classifier for category  $c_i$  from a training set  $Tr$ ), and *classification efficiency* (i.e., the average time it takes to classify a new document  $d_j$  under category  $c_i$ ).

An important alternative to effectiveness is *utility*, a class of measures from decision theory that extend effectiveness by economic criteria such as *gain* or *loss*. Utility is based on a *utility matrix* such as that of Table IV, where the numeric values  $u_{TP}$ ,  $u_{FP}$ ,  $u_{FN}$  and  $u_{TN}$  represent the gain brought about by a true positive, false positive, false negative, and true negative, respectively; both  $u_{TP}$  and  $u_{TN}$  are greater than both  $u_{FP}$  and  $u_{FN}$ . “Standard” effectiveness is a special case of utility, i.e., the one in which  $u_{TP} = u_{TN} > u_{FP} = u_{FN}$ . Less trivial cases are those in which  $u_{TP} \neq u_{TN}$  and/or  $u_{FP} \neq u_{FN}$ ; this is appropriate, for example, in spam filtering, where failing to discard a piece of junk mail (FP) is a less serious mistake than discarding a legitimate message (FN) [Androusoopoulos et al. 2000]. If the classifier outputs probability estimates of the membership of  $d_j$  in  $c_i$ , then decision theory provides *analytical* methods to determine thresholds  $\tau_i$ , thus avoiding the need to determine them experimentally (as discussed in Section 6.1). Specifically, as Lewis [1995a] reminds us, the expected value of utility is maximized when

$$\tau_i = \frac{(u_{FP} - u_{TN})}{(u_{FN} - u_{TP}) + (u_{FP} - u_{TN})},$$

which, in the case of “standard” effectiveness, is equal to  $\frac{1}{2}$ .

Table V. Trivial Cases in TC

		Precision $\frac{TP}{TP+FP}$	Recall $\frac{TP}{TP+FN}$	C-precision $\frac{TN}{FP+TN}$	C-recall $\frac{TN}{TN+FN}$
Trivial rejector	TP = FP = 0	Undefined	$\frac{0}{FN} = 0$	$\frac{TN}{TN} = 1$	$\frac{TN}{TN+FN}$
Trivial acceptor	FN = TN = 0	$\frac{TP}{TP+FP}$	$\frac{TP}{TP} = 1$	$\frac{0}{FP} = 0$	Undefined
Trivial “Yes” collection	FP = TN = 0	$\frac{TP}{TP} = 1$	$\frac{TP}{TP+FN}$	Undefined	$\frac{0}{FN} = 0$
Trivial “No” collection	TP = FN = 0	$\frac{0}{FP} = 0$	Undefined	$\frac{TN}{FP+TN}$	$\frac{TN}{TN} = 1$

The use of utility in TC is discussed in detail by Lewis [1955a]. Other works where utility is employed are Amati and Crestani [1999], Cohen and Singer [1999], Hull et al. [1996], Lewis and Catlett [1994], and Schapire et al. [1998]. Utility has become popular within the text filtering community, and the TREC “filtering track” evaluations have been using it for a while [Lewis 1995c]. The values of the utility matrix are extremely application-dependent. This means that if utility is used instead of “pure” effectiveness, there is a further element of difficulty in the cross-comparison of classification systems (see Section 7.3), since for two classifiers to be experimentally comparable also the two utility matrices must be the same.

Other effectiveness measures different from the ones discussed here have occasionally been used in the literature; these include *adjacent score* [Larkey 1998], *coverage* [Schapire and Singer 2000], *one-error* [Schapire and Singer 2000], *Pearson product-moment correlation* [Larkey 1998], *recall at n* [Larkey and Croft 1996], *top candidate* [Larkey and Croft 1996], and *top n* [Larkey and Croft 1996]. We will not attempt to discuss them in detail. However, their use shows that, although the TC community is making consistent efforts at standardizing experimentation protocols, we are still far from universal agreement on evaluation issues and, as a consequence, from understanding precisely the relative merits of the various methods.

**7.1.4. Combined Effectiveness Measures.** Neither precision nor recall makes sense in isolation from each other. In fact the classifier  $\Phi$  such that  $\Phi(d_j, c_i) = T$  for all  $d_j$  and  $c_i$  (the *trivial acceptor*) has  $\rho = 1$ . When the  $CSV_i$  function has values in  $[0, 1]$ , one only needs to set every threshold  $\tau_i$  to 0 to obtain the trivial acceptor. In this case,  $\pi$  would usually be very low (more precisely, equal to the average test set generality  $\frac{\sum_{i=1}^{|C|} g_{T_e}(c_i)}{|C|}$ ).<sup>16</sup> Conversely, it is well known from everyday IR practice that higher levels of  $\pi$  may be obtained at the price of low values of  $\rho$ .

In practice, by tuning  $\tau_i$  a function  $CSV_i : \mathcal{D} \rightarrow \{T, F\}$  is tuned to be, in the words of Riloff and Lehnert [1994], more *liberal* (i.e., improving  $\rho_i$  to the detriment of  $\pi_i$ ) or more *conservative* (improving  $\pi_i$  to

<sup>16</sup> From this, one might be tempted to infer, by symmetry, that the trivial rejector always has  $\pi = 1$ . This is false, as  $\pi$  is undefined (the denominator is zero) for the trivial rejector (see Table V). In fact, it is clear from its definition ( $\pi = \frac{TP}{TP+FP}$ ) that  $\pi$  depends only on how the positives ( $TP + FP$ ) are split between *true* positives  $TP$  and the *false* positives  $FP$ , and does not depend at all on the cardinality of the positives. There is a breakup of “symmetry” between  $\pi$  and  $\rho$  here because, from the point of view of classifier judgment (positives vs. negatives; this is the dichotomy of interest in trivial acceptor vs. trivial rejector), the “symmetric” of  $\rho$  ( $\frac{TP}{TP+FN}$ ) is not  $\pi$  ( $\frac{TP}{TP+FP}$ ) but *C-precision* ( $\pi^c = \frac{TN}{FP+TN}$ ), the “contrapositive” of  $\pi$ . In fact, while  $\rho = 1$  and  $\pi^c = 0$  for the trivial acceptor,  $\pi^c = 1$  and  $\rho = 0$  for the trivial rejector.

the detriment of  $\rho_i$ ).<sup>17</sup> A classifier should thus be evaluated by means of a measure which combines  $\pi$  and  $\rho$ .<sup>18</sup> Various such measures have been proposed, among which the most frequent are:

- (1) *Eleven-point average precision*: threshold  $\tau_i$  is repeatedly tuned so as to allow  $\rho_i$  to take up values of 0.0, .1, . . . , .9, 1.0;  $\pi_i$  is computed for these 11 different values of  $\tau_i$ , and averaged over the 11 resulting values. This is analogous to the standard evaluation methodology for ranked IR systems, and may be used
  - (a) with categories in place of IR queries. This is most frequently used for document-ranking classifiers (see Schütze et al. [1995]; Yang [1994]; Yang [1999]; Yang and Pedersen [1997]);
  - (b) with test documents in place of IR queries and categories in place of documents. This is most frequently used for category-ranking classifiers (see Lam et al. [1999]; Larkey and Croft [1996]; Schapire and Singer [2000]; Wiener et al. [1995]). In this case, if macroaveraging is used, it needs to be redefined on a per-document, rather than per-category, basis.

This measure does not make sense for binary-valued  $CSV_i$  functions, since in this case  $\rho_i$  may not be varied at will.

- (2) The *breakeven* point, that is, the value at which  $\pi$  equals  $\rho$  (e.g., Apté et al. [1994]; Cohen and Singer [1999]; Dagan et al. [1997]; Joachims [1998];

Joachims [1999]; Lewis [1992a]; Lewis and Ringuette [1994]; Moulinier and Ganascia [1996]; Ng et al. [1997]; Yang [1999]). This is obtained by a process analogous to the one used for 11-point average precision: a plot of  $\pi$  as a function of  $\rho$  is computed by repeatedly varying the thresholds  $\tau_i$ ; breakeven is the value of  $\rho$  (or  $\pi$ ) for which the plot intersects the  $\rho = \pi$  line. This idea relies on the fact that, by decreasing the  $\tau_i$ 's from 1 to 0,  $\rho$  always increases monotonically from 0 to 1 and  $\pi$  usually decreases monotonically from a value near 1 to  $\frac{1}{|C|} \sum_{i=1}^{|C|} g_{Te}(c_i)$ . If for no values of the  $\tau_i$ 's  $\pi$  and  $\rho$  are exactly equal, the  $\tau_i$ 's are set to the value for which  $\pi$  and  $\rho$  are closest, and an *interpolated breakeven* is computed as the average of the values of  $\pi$  and  $\rho$ .<sup>19</sup>

- (3) The  $F_\beta$  function [van Rijsbergen 1979, Chapter 7], for some  $0 \leq \beta \leq +\infty$  (e.g., Cohen [1995a]; Cohen and Singer [1999]; Lewis and Gale [1994]; Lewis [1995a]; Moulinier et al. [1996]; Ruiz and Srinivassan [1999]), where

$$F_\beta = \frac{(\beta^2 + 1)\pi\rho}{\beta^2\pi + \rho}$$

Here  $\beta$  may be seen as the relative degree of importance attributed to  $\pi$  and  $\rho$ . If  $\beta = 0$  then  $F_\beta$  coincides with  $\pi$ , whereas if  $\beta = +\infty$  then  $F_\beta$  coincides with  $\rho$ . Usually, a value  $\beta = 1$  is used, which attributes equal importance to  $\pi$  and  $\rho$ . As shown in Moulinier et al. [1996] and Yang [1999], the breakeven of a classifier  $\Phi$  is always less or equal than its  $F_1$  value.

<sup>17</sup> While  $\rho_i$  can *always* be increased at will by lowering  $\tau_i$ , *usually* at the cost of decreasing  $\pi_i$ ,  $\pi_i$  can *usually* be increased at will by raising  $\tau_i$ , *always* at the cost of decreasing  $\rho_i$ . This kind of tuning is only possible for  $CSV_i$  functions with values in  $[0, 1]$ ; for binary-valued  $CSV_i$  functions tuning is not always possible, or is anyway more difficult (see Weiss et al. [1999], page 66).

<sup>18</sup> An exception is single-label TC, in which  $\pi$  and  $\rho$  are not independent of each other: if a document  $d_j$  has been classified under a wrong category  $c_s$  (thus decreasing  $\pi_s$ ), this also means that it has *not* been classified under the right category  $c_t$  (thus decreasing  $\rho_t$ ). In this case either  $\pi$  or  $\rho$  can be used as a measure of effectiveness.

<sup>19</sup> Breakeven, first proposed by Lewis [1992a, 1992b], has been recently criticized. Lewis himself (see his message of 11 Sep 1997 10:49:01 to the DDLBETA text categorization mailing list—quoted with permission of the author) has pointed out that breakeven is not a good effectiveness measure, since (i) there may be no parameter setting that yields the breakeven; in this case the final breakeven value, obtained by interpolation, is artificial; (ii) to have  $\rho$  equal  $\pi$  is not necessarily desirable, and it is not clear that a system that achieves high breakeven can be tuned to score high on other effectiveness measures. Yang [1999] also noted that when for no value of the parameters  $\pi$  and  $\rho$  are close enough, interpolated breakeven may not be a reliable indicator of effectiveness.

Once an effectiveness measure is chosen, a classifier can be tuned (e.g., thresholds and other parameters can be set) so that the resulting effectiveness is the best achievable by that classifier. Tuning a parameter  $p$  (be it a threshold or other) is normally done experimentally. This means performing repeated experiments on the validation set with the values of the other parameters  $p_k$  fixed (at a default value, in the case of a yet-to-be-tuned parameter  $p_k$ , or at the chosen value, if the parameter  $p_k$  has already been tuned) and with different values for parameter  $p$ . The value that has yielded the best effectiveness is chosen for  $p$ .

## 7.2. Benchmarks for Text Categorization

Standard *benchmark collections* that can be used as initial corpora for TC are publicly available for experimental purposes. The most widely used is the Reuters collection, consisting of a set of newswire stories classified under categories related to economics. The Reuters collection accounts for most of the experimental work in TC so far. Unfortunately, this does not always translate into reliable comparative results, in the sense that many of these experiments have been carried out in subtly different conditions.

In general, different sets of experiments may be used for cross-classifier comparison only if the experiments have been performed

- (1) on exactly the same collection (i.e., same documents and same categories);
- (2) with the same “split” between training set and test set;
- (3) with the same evaluation measure and, whenever this measure depends on some parameters (e.g., the utility matrix chosen), with the same parameter values.

Unfortunately, a lot of experimentation, both on Reuters and on other collections, has not been performed with these *caveats* in mind: by testing three different classifiers on five popular versions of Reuters, Yang [1999] has shown that

a lack of compliance with these three conditions may make the experimental results hardly comparable among each other. Table VI lists the results of all experiments known to us performed on five major versions of the Reuters benchmark: Reuters-22173 “ModLewis” (column #1), Reuters-22173 “ModApté” (column #2), Reuters-22173 “ModWiener” (column #3), Reuters-21578 “ModApté” (column #4), and Reuters-21578[10] “ModApté” (column #5).<sup>20</sup> Only experiments that have computed either a breakeven or  $F_1$  have been listed, since other less popular effectiveness measures do not readily compare with these.

Note that only results belonging to the same column are directly comparable. In particular, Yang [1999] showed that experiments carried out on Reuters-22173 “ModLewis” (column #1) are not directly comparable with those using the other three versions, since the former strangely includes a significant percentage (58%) of “unlabeled” test documents which, being negative examples of all categories, tend to depress effectiveness. Also, experiments performed on Reuters-21578[10] “ModApté” (column #5) are not comparable with the others, since this collection is the restriction of Reuters-21578 “ModApté” to the 10 categories with the highest generality, and is thus an obviously “easier” collection.

Other test collections that have been frequently used are

- the OHSUMED collection, set up by Hersh et al. [1994] and used by Joachims [1998], Lam and Ho [1998], Lam et al. [1999], Lewis et al. [1996], Ruiz and Srinivasan [1999], and Yang

<sup>20</sup> The Reuters-21578 collection may be freely downloaded for experimentation purposes from <http://www.research.att.com/~lewis/reuters21578.html>. A new corpus, called Reuters Corpus Volume 1 and consisting of roughly 800,000 documents, has recently been made available by Reuters for TC experiments (see <http://about.reuters.com/researchandstandards/corpus/>). This will likely replace Reuters-21578 as the “standard” Reuters benchmark for TC.

**Table VI.** Comparative Results Among Different Classifiers Obtained on Five Different Versions of Reuters. (Unless otherwise noted, entries indicate the microaveraged breakeven point; within parentheses, “M” indicates macroaveraging and “ $F_1$ ” indicates use of the  $F_1$  measure; **boldface** indicates the best performer on the collection)

			#1	#2	#3	#4	#5
		# of documents	21,450	14,347	13,272	12,902	12,902
		# of training documents	14,704	10,667	9,610	9,603	9,603
		# of test documents	6,746	3,680	3,662	3,299	3,299
		# of categories	135	93	92	90	10
System	Type	Results reported by					
WORD	(non-learning)	Yang [1999]	.150	.310	.290		
PROP BAYES BIM NB	probabilistic	[Dumais et al. 1998]				.752	.815
	probabilistic	[Joachims 1998]				.720	
	probabilistic	[Lam et al. 1997]	.443 ( $MF_1$ )				
	probabilistic	[Lewis 1992a]	.650				
	probabilistic	[Li and Yamanishi 1999]				.747	
	probabilistic	[Li and Yamanishi 1999]				.773	
	probabilistic	[Yang and Liu 1999]				.795	
C4.5 IND	decision trees	[Dumais et al. 1998]					.884
	decision trees	[Joachims 1998]				.794	
	decision trees	[Lewis and Ringuette 1994]	.670				
SWAP-1 RIPPER SLEEPING EXPERTS DL-ESC CHARADE CHARADE	decision rules	[Apté et al. 1994]		.805			
	decision rules	[Cohen and Singer 1999]	.683	.811		.820	
	decision rules	[Cohen and Singer 1999]	<b>.753</b>	.759		.827	
	decision rules	[Li and Yamanishi 1999]				.820	
	decision rules	[Moulinier and Ganascia 1996]		.738			
	decision rules	[Moulinier et al. 1996]		<b>.783 (<math>F_1</math>)</b>			
LSF LSF	regression	[Yang 1999]		.855	.810		
	regression	[Yang and Liu 1999]				.849	
BALANCED WINNOWER WIDROW-HOFF	on-line linear	[Dagan et al. 1997]	.747 (M)	.833 (M)			
	on-line linear	[Lam and Ho 1998]				.822	
ROCCHIO FINDSIM ROCCHIO ROCCHIO ROCCHIO	batch linear	[Cohen and Singer 1999]	.660	.748		.776	.646
	batch linear	[Dumais et al. 1998]				.617	
	batch linear	[Joachims 1998]				.799	
	batch linear	[Lam and Ho 1998]				.781	
	batch linear	[Li and Yamanishi 1999]				.625	
CLASSI NNET	neural network	[Ng et al. 1997]		.802			
	neural network	Yang and Liu 1999]				.838	
	neural network	[Wiener et al. 1995]			<b>.820</b>		
GIS-W k-NN k-NN k-NN k-NN	example-based	[Lam and Ho 1998]				.860	
	example-based	[Joachims 1998]				.823	
	example-based	[Lam and Ho 1998]				.820	
	example-based	[Yang 1999]	.690	.852	<b>.820</b>		
	example-based	[Yang and Liu 1999]				.856	
SVM LIGHT SVM LIGHT SVM LIGHT SVM LIGHT	SVM	[Dumais et al. 1998]				.870	<b>.920</b>
	SVM	[Joachims 1998]				.864	
	SVM	[Li Yamanishi 1999]				.841	
	SVM	[Yang and Liu 1999]				.859	
ADA BOOST.MH	committee committee	[Schapire and Singer 2000] [Weiss et al. 1999]		<b>.860</b>		<b>.878</b>	
	Bayesian net	[Dumais et al. 1998]				.800	.850
	Bayesian net	[Lam et al. 1997]	.542 ( $MF_1$ )				

and Pedersen [1997].<sup>21</sup> The documents are titles or title-plus-abstracts from medical journals (OHSUMED is actually a subset of the Medline document base);

<sup>21</sup> The OHSUMED collection may be freely downloaded for experimentation purposes from <ftp://medir.ohsu.edu/pub/ohsumed>.

the categories are the “postable terms” of the MESH thesaurus.

—the 20 Newsgroups collection, set up by Lang [1995] and used by Baker and McCallum [1998], Joachims [1997], McCallum and Nigam [1998], McCallum et al. [1998], Nigam et al.

- [2000], and Schapire and Singer [2000]. The documents are messages posted to Usenet newsgroups, and the categories are the newsgroups themselves.
- the AP collection, used by Cohen [1995a, 1995b], Cohen and Singer [1999], Lewis and Catlett [1994], Lewis and Gale [1994], Lewis et al. [1996], Schapire and Singer [2000], and Schapire et al. [1998].

We will not cover the experiments performed on these collections for the same reasons as those illustrated in footnote 20, that is, because in no case have a significant enough number of authors used the same collection in the same experimental conditions, thus making comparisons difficult.

### 7.3. Which Text Classifier Is Best?

The published experimental results, and especially those listed in Table VI, allow us to attempt some considerations on the comparative performance of the TC methods discussed. However, we have to bear in mind that comparisons are reliable only when based on experiments performed by the same author under carefully controlled conditions. They are instead more problematic when they involve different experiments performed by different authors. In this case various “background conditions,” often extraneous to the learning algorithm itself, may influence the results. These may include, among others, different choices in preprocessing (stemming, etc.), indexing, dimensionality reduction, classifier parameter values, etc., but also different standards of compliance with safe scientific practice (such as tuning parameters on the test set rather than on a separate validation set), which often are not discussed in the published papers.

Two different methods may thus be applied for comparing classifiers [Yang 1999]:

- direct comparison*: classifiers  $\Phi'$  and  $\Phi''$  may be compared when they have been tested on the same collection  $\Omega$ , usually by the same researchers and with the

same background conditions. This is the more reliable method.

- indirect comparison*: classifiers  $\Phi'$  and  $\Phi''$  may be compared when
  - (1) they have been tested on collections  $\Omega'$  and  $\Omega''$ , respectively, typically by different researchers and hence with possibly different background conditions;
  - (2) one or more “baseline” classifiers  $\bar{\Phi}_1, \dots, \bar{\Phi}_m$  have been tested on both  $\Omega'$  and  $\Omega''$  by the direct comparison method.

Test 2 gives an indication on the relative “hardness” of  $\Omega'$  and  $\Omega''$ ; using this and the results from Test 1, we may obtain an indication on the relative effectiveness of  $\Phi'$  and  $\Phi''$ . For the reasons discussed above, this method is less reliable.

A number of interesting conclusions can be drawn from Table VI by using these two methods. Concerning the relative “hardness” of the five collections, if by  $\Omega' > \Omega''$  we indicate that  $\Omega'$  is a harder collection than  $\Omega''$ , there seems to be enough evidence that Reuters-22173 “ModLewis”  $\gg$  Reuters-22173 “ModWiener”  $>$  Reuters-22173 “ModApté”  $\approx$  Reuters-21578 “ModApté”  $>$  Reuters-21578[10] “ModApté.” These facts are unsurprising; in particular, the first and the last inequalities are a direct consequence of the peculiar characteristics of Reuters-22173 “ModLewis” and Reuters-21578[10] “ModApté” discussed in Section 7.2.

Concerning the relative performance of the classifiers, remembering the considerations above we may attempt a few conclusions:

- Boosting-based classifier committees, support vector machines, example-based methods, and regression methods deliver top-notch performance. There seems to be no sufficient evidence to decidedly opt for either method; efficiency considerations or application-dependent issues might play a role in breaking the tie.
- Neural networks and on-line linear classifiers work very well, although slightly

worse than the previously mentioned methods.

- Batch linear classifiers (Rocchio) and probabilistic Naïve Bayes classifiers look the worst of the learning-based classifiers. For Rocchio, these results confirm earlier results by Schütze et al. [1995], who had found three classifiers based on linear discriminant analysis, linear regression, and neural networks to perform about 15% better than Rocchio. However, recent results by Schapire et al. [1998] ranked Rocchio along the best performers once near-positives are used in training.
- The data in Table VI is hardly sufficient to say anything about decision trees. However, the work by Dumais et al. [1998], in which a decision tree classifier was shown to perform nearly as well as their top performing system (a SVM classifier), will probably renew the interest in decision trees, an interest that had dwindled after the unimpressive results reported in earlier literature [Cohen and Singer 1999; Joachims 1998; Lewis and Catlett 1994; Lewis and Ringuette 1994].
- By far the lowest performance is displayed by WORD, a classifier implemented by Yang [1999] and not including any learning component.<sup>22</sup>

Concerning WORD and no-learning classifiers, for completeness we should recall that one of the highest effectiveness values reported in the literature for the Reuters collection (a .90 breakeven) belongs to CONSTRUE, a manually constructed classifier. However, this classifier has never been tested on the *standard* variants of Reuters mentioned in Table VI, and it is not clear [Yang 1999] whether the (small) test set of Reuters-22173 “ModHayes” on

which the .90 breakeven value was obtained was chosen randomly, as safe scientific practice would demand. Therefore, the fact that this figure is indicative of the performance of CONSTRUE, and of the manual approach it represents, has been convincingly questioned [Yang 1999].

It is important to bear in mind that the considerations above are not absolute statements (if there may be any) on the comparative effectiveness of these TC methods. One of the reasons is that a particular applicative context may exhibit very different characteristics from the ones to be found in Reuters, and different classifiers may respond differently to these characteristics. An experimental study by Joachims [1998] involving support vector machines,  $k$ -NN, decision trees, Rocchio, and Naïve Bayes, showed all these classifiers to have similar effectiveness on categories with  $\geq 300$  positive training examples each. The fact that this experiment involved the methods which have scored best (support vector machines,  $k$ -NN) and worst (Rocchio and Naïve Bayes) according to Table VI shows that applicative contexts different from Reuters may well invalidate conclusions drawn on this latter.

Finally, a note about the worth of statistical significance testing. Few authors have gone to the trouble of validating their results by means of such tests. These tests are useful for verifying how strongly the experimental results support the claim that a given system  $\Phi'$  is better than another system  $\Phi''$ , or for verifying how much a difference in the experimental setup affects the measured effectiveness of a system  $\Phi$ . Hull [1994] and Schütze et al. [1995] have been among the first to work in this direction, validating their results by means of the ANOVA test and the Friedman test; the former is aimed at determining the significance of the difference in effectiveness between two methods in terms of the ratio between this difference and the effectiveness variability across categories, while the latter conducts a similar test by using instead the rank positions of each method within a category. Yang and Liu [1999] defined a full suite of significance

<sup>22</sup> WORD is based on the comparison between documents and category names, each treated as a vector of weighted terms in the vector space model. WORD was implemented by Yang with the only purpose of determining the difference in effectiveness that adding a learning component to a classifier brings about. WORD is actually called STR in [Yang 1994; Yang and Chute 1994]. Another no-learning classifier was proposed in Wong et al. [1996].



tests, some of which apply to microaveraged and some to macroaveraged effectiveness. They applied them systematically to the comparison between five different classifiers, and were thus able to infer fine-grained conclusions about their relative effectiveness. For other examples of significance testing in TC, see Cohen [1995a, 1995b]; Cohen and Hirsh [1998], Joachims [1997], Koller and Sahami [1997], Lewis et al. [1996], and Wiener et al. [1995].

## 8. CONCLUSION

Automated TC is now a major research area within the information systems discipline, thanks to a number of factors:

- Its domains of application are numerous and important, and given the proliferation of documents in digital form they are bound to increase dramatically in both number and importance.
- It is indispensable in many applications in which the sheer number of the documents to be classified and the short response time required by the application make the manual alternative implausible.
- It can improve the productivity of human classifiers in applications in which no classification decision can be taken without a final human judgment [Larkey and Croft 1996], by providing tools that quickly “suggest” plausible decisions.
- It has reached effectiveness levels comparable to those of trained professionals. The effectiveness of manual TC is not 100% anyway [Cleverdon 1984] and, more importantly, it is unlikely to be improved substantially by the progress of research. The levels of effectiveness of automated TC are instead growing at a steady pace, and even if they will likely reach a plateau well below the 100% level, this plateau will probably be higher than the effectiveness levels of manual TC.

One of the reasons why from the early '90s the effectiveness of text classifiers has dramatically improved is the arrival

in the TC arena of ML methods that are backed by strong theoretical motivations. Examples of these are multiplicative weight updating (e.g., the WINNOWER family, WIDROW-HOFF, etc.), adaptive resampling (e.g., boosting), and support vector machines, which provide a sharp contrast with relatively unsophisticated and weak methods such as Rocchio. In TC, ML researchers have found a challenging application, since datasets consisting of hundreds of thousands of documents and characterized by tens of thousands of terms are widely available. This means that TC is a good benchmark for checking whether a given learning technique can scale up to substantial sizes. In turn, this probably means that the active involvement of the ML community in TC is bound to grow.

The success story of automated TC is also going to encourage an extension of its methods and techniques to neighboring fields of application. Techniques typical of automated TC have already been extended successfully to the categorization of documents expressed in slightly different media; for instance:

- very noisy text resulting from optical character recognition [Ittner et al. 1995; Junker and Hoch 1998]. In their experiments Ittner et al. [1995] have found that, by employing noisy texts also in the training phase (i.e. texts affected by the same source of noise that is also at work in the test documents), effectiveness levels comparable to those obtainable in the case of standard text can be achieved.
- speech transcripts [Myers et al. 2000; Schapire and Singer 2000]. For instance, Schapire and Singer [2000] classified answers given to a phone operator's request “How may I help you?” so as to be able to route the call to a specialized operator according to call type.

Concerning other more radically different media, the situation is not as bright (however, see Lim [1999] for an interesting attempt at image categorization based

on a textual metaphor). The reason for this is that capturing real semantic content of nontextual media by automatic indexing is still an open problem. While there are systems that attempt to detect content, for example, in images by recognizing shapes, color distributions, and texture, the general problem of image semantics is still unsolved. The main reason is that natural language, the language of the text medium, admits far fewer variations than the “languages” employed by the other media. For instance, while the concept of a house can be “triggered” by relatively few natural language expressions such as house, houses, home, housing, inhabiting, etc., it can be triggered by *far more* images: the images of all the different houses that exist, of all possible colors and shapes, viewed from all possible perspectives, from all possible distances, etc. If we had solved the multimedia indexing problem in a satisfactory way, the general methodology that we have discussed in this paper for text would also apply to automated multimedia categorization, and there are reasons to believe that the effectiveness levels could be as high. This only adds to the common sentiment that more research in automated content-based indexing for multimedia documents is needed.

#### ACKNOWLEDGMENTS

This paper owes a lot to the suggestions and constructive criticism of Norbert Fuhr and David Lewis. Thanks also to Umberto Straccia for comments on an earlier draft, to Evgeniy Gabrilovich, Daniela Giorgetti, and Alessandro Moschitti for spotting mistakes in an earlier draft, and to Alessandro Sperduti for many fruitful discussions.

#### REFERENCES

- AMATI, G. AND CRESTANI, F. 1999. Probabilistic learning for selective dissemination of information. *Inform. Process. Man.* 35, 5, 633–654.
- ANDROUTSOPOULOS, I., KOUTSIAS, J., CHANDRINOS, K. V., AND SPYROPOULOS, C. D. 2000. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval* (Athens, Greece, 2000), 160–167.
- APTÉ, C., DAMERAU, F. J., AND WEISS, S. M. 1994. Automated learning of decision rules for text categorization. *ACM Trans. on Inform. Syst.* 12, 3, 233–251.
- ATTARDI, G., DI MARCO, S., AND SALVI, D. 1998. Categorization by context. *J. Univers. Comput. Sci.* 4, 9, 719–736.
- BAKER, L. D. AND MCCALLUM, A. K. 1998. Distributional clustering of words for text classification. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval* (Melbourne, Australia, 1998), 96–103.
- BELKIN, N. J. AND CROFT, W. B. 1992. Information filtering and information retrieval: two sides of the same coin? *Commun. ACM* 35, 12, 29–38.
- BIEBRICHER, P., FUHR, N., KNORZ, G., LUSTIG, G., AND SCHWANTNER, M. 1988. The automatic indexing system AIR/PHYS. From research to application. In *Proceedings of SIGIR-88, 11th ACM International Conference on Research and Development in Information Retrieval* (Grenoble, France, 1988), 333–342. Also reprinted in Sparck Jones and Willett [1997], pp. 513–517.
- BORKO, H. AND BERNICK, M. 1963. Automatic document classification. *J. Assoc. Comput. Mach.* 10, 2, 151–161.
- CAROPRESO, M. F., MATWIN, S., AND SEBASTIANI, F. 2001. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. In *Text Databases and Document Management: Theory and Practice*, A. G. Chin, ed. Idea Group Publishing, Hershey, PA, 78–102.
- CAVNAR, W. B. AND TRENKLE, J. M. 1994. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV, 1994), 161–175.
- CHAKRABARTI, S., DOM, B. E., AGRAWAL, R., AND RAGHAVAN, P. 1998a. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *J. Very Large Data Bases* 7, 3, 163–178.
- CHAKRABARTI, S., DOM, B. E., AND INDYK, P. 1998b. Enhanced hypertext categorization using hyperlinks. In *Proceedings of SIGMOD-98, ACM International Conference on Management of Data* (Seattle, WA, 1998), 307–318.
- CLACK, C., FARRINGTON, J., LIDWELL, P., AND YU, T. 1997. Autonomous document classification for business. In *Proceedings of the 1st International Conference on Autonomous Agents* (Marina del Rey, CA, 1997), 201–208.
- CLEVERDON, C. 1984. Optimizing convenient on-line access to bibliographic databases. *Inform. Serv. Use* 4, 1, 37–47. Also reprinted in Willett [1988], pp. 32–41.
- COHEN, W. W. 1995a. Learning to classify English text with ILP methods. In *Advances in Inductive*

- Logic Programming*, L. De Raedt, ed. IOS Press, Amsterdam, The Netherlands, 124–143.
- COHEN, W. W. 1995b. Text categorization and relational learning. In *Proceedings of ICML-95, 12th International Conference on Machine Learning* (Lake Tahoe, CA, 1995), 124–132.
- COHEN, W. W. AND HIRSH, H. 1998. Joins that generalize: text classification using WHIRL. In *Proceedings of KDD-98, 4th International Conference on Knowledge Discovery and Data Mining* (New York, NY, 1998), 169–173.
- COHEN, W. W. AND SINGER, Y. 1999. Context-sensitive learning methods for text categorization. *ACM Trans. Inform. Syst.* 17, 2, 141–173.
- COOPER, W. S. 1995. Some inconsistencies and misnomers in probabilistic information retrieval. *ACM Trans. Inform. Syst.* 13, 1, 100–111.
- CREECY, R. M., MASAND, B. M., SMITH, S. J., AND WALTZ, D. L. 1992. Trading MIPS and memory for knowledge engineering: classifying census returns on the Connection Machine. *Commun. ACM* 35, 8, 48–63.
- CRESTANI, F., LALMAS, M., VAN RIJSBERGEN, C. J., AND CAMPBELL, I. 1998. “Is this document relevant? ... probably.” A survey of probabilistic models in information retrieval. *ACM Comput. Surv.* 30, 4, 528–552.
- DAGAN, I., KAROV, Y., AND ROTH, D. 1997. Mistake-driven learning in text categorization. In *Proceedings of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing* (Providence, RI, 1997), 55–63.
- DEERWESTER, S., DUMAIS, S. T., FURNAS, G. W., LANDAUER, T. K., AND HARSHMAN, R. 1990. Indexing by latent semantic indexing. *J. Amer. Soc. Inform. Sci.* 41, 6, 391–407.
- DENOYER, L., ZARAGOZA, H., AND GALLINARI, P. 2001. HMM-based passage models for document classification and ranking. In *Proceedings of ECIR-01, 23rd European Colloquium on Information Retrieval Research* (Darmstadt, Germany, 2001).
- DÍAZ ESTEBAN, A., DE BUENAGA RODRÍGUEZ, M., UREÑA LÓPEZ, L. A., AND GARCÍA VEGA, M. 1998. Integrating linguistic resources in a uniform way for text classification tasks. In *Proceedings of LREC-98, 1st International Conference on Language Resources and Evaluation* (Grenada, Spain, 1998), 1197–1204.
- DOMINGOS, P. AND PAZZANI, M. J. 1997. On the the optimality of the simple Bayesian classifier under zero-one loss. *Mach. Learn.* 29, 2–3, 103–130.
- DRUCKER, H., VAPNIK, V., AND WU, D. 1999. Automatic text categorization and its applications to text retrieval. *IEEE Trans. Neural Netw.* 10, 5, 1048–1054.
- DUMAIS, S. T. AND CHEN, H. 2000. Hierarchical classification of Web content. In *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval* (Athens, Greece, 2000), 256–263.
- DUMAIS, S. T., PLATT, J., HECKERMAN, D., AND SAHAMI, M. 1998. Inductive learning algorithms and representations for text categorization. In *Proceedings of CIKM-98, 7th ACM International Conference on Information and Knowledge Management* (Bethesda, MD, 1998), 148–155.
- ESCUDERO, G., MÁRQUEZ, L., AND RIGAU, G. 2000. Boosting applied to word sense disambiguation. In *Proceedings of ECML-00, 11th European Conference on Machine Learning* (Barcelona, Spain, 2000), 129–141.
- FIELD, B. 1975. Towards automatic indexing: automatic assignment of controlled-language indexing and classification from free indexing. *J. Document.* 31, 4, 246–265.
- FORSYTH, R. S. 1999. New directions in text categorization. In *Causal Models and Intelligent Data Management*, A. Gammerman, ed. Springer, Heidelberg, Germany, 151–185.
- FRASCONI, P., SODA, G., AND VULLO, A. 2002. Text categorization for multi-page documents: A hybrid naive Bayes HMM approach. *J. Intell. Inform. Syst.* 18, 2/3 (March–May), 195–217.
- FUHR, N. 1985. A probabilistic model of dictionary-based automatic indexing. In *Proceedings of RIAO-85, 1st International Conference “Recherche d’Information Assistée par Ordinateur”* (Grenoble, France, 1985), 207–216.
- FUHR, N. 1989. Models for retrieval with probabilistic indexing. *Inform. Process. Man.* 25, 1, 55–72.
- FUHR, N. AND BUCKLEY, C. 1991. A probabilistic learning approach for document indexing. *ACM Trans. Inform. Syst.* 9, 3, 223–248.
- FUHR, N., HARTMANN, S., KNORZ, G., LUSTIG, G., SCHWANTNER, M., AND TZERAS, K. 1991. AIR/X—a rule-based multistage indexing system for large subject fields. In *Proceedings of RIAO-91, 3rd International Conference “Recherche d’Information Assistée par Ordinateur”* (Barcelona, Spain, 1991), 606–623.
- FUHR, N. AND KNORZ, G. 1984. Retrieval test evaluation of a rule-based automated indexing (AIR/PHYS). In *Proceedings of SIGIR-84, 7th ACM International Conference on Research and Development in Information Retrieval* (Cambridge, UK, 1984), 391–408.
- FUHR, N. AND PFEIFER, U. 1994. Probabilistic information retrieval as combination of abstraction inductive learning and probabilistic assumptions. *ACM Trans. Inform. Syst.* 12, 1, 92–115.
- FÜRNKRANZ, J. 1999. Exploiting structural information for text classification on the WWW. In *Proceedings of IDA-99, 3rd Symposium on Intelligent Data Analysis* (Amsterdam, The Netherlands, 1999), 487–497.
- GALAVOTTI, L., SEBASTIANI, F., AND SIMI, M. 2000. Experiments on the use of feature selection and negative evidence in automated text categorization. In *Proceedings of ECDL-00, 4th European Conference on Research and*

- Advanced Technology for Digital Libraries* (Lisbon, Portugal, 2000), 59–68.
- GALE, W. A., CHURCH, K. W., AND YAROWSKY, D. 1993. A method for disambiguating word senses in a large corpus. *Comput. Human.* 26, 5, 415–439.
- GÖVERT, N., LALMAS, M., AND FUHR, N. 1999. A probabilistic description-oriented approach for categorising Web documents. In *Proceedings of CIKM-99, 8th ACM International Conference on Information and Knowledge Management* (Kansas City, MO, 1999), 475–482.
- GRAY, W. A. AND HARLEY, A. J. 1971. Computer-assisted indexing. *Inform. Storage Retrieval* 7, 4, 167–174.
- GUTHRIE, L., WALKER, E., AND GUTHRIE, J. A. 1994. Document classification by machine: theory and practice. In *Proceedings of COLING-94, 15th International Conference on Computational Linguistics* (Kyoto, Japan, 1994), 1059–1063.
- HAYES, P. J., ANDERSEN, P. M., NIRENBURG, I. B., AND SCHMANDT, L. M. 1990. Tcs: a shell for content-based text categorization. In *Proceedings of CAIA-90, 6th IEEE Conference on Artificial Intelligence Applications* (Santa Barbara, CA, 1990), 320–326.
- HEAPS, H. 1973. A theory of relevance for automatic document classification. *Inform. Control* 22, 3, 268–278.
- HERSH, W., BUCKLEY, C., LEONE, T., AND HICKMAN, D. 1994. OHSUMED: an interactive retrieval evaluation and new large text collection for research. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval* (Dublin, Ireland, 1994), 192–201.
- HULL, D. A. 1994. Improving text retrieval for the routing problem using latent semantic indexing. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval* (Dublin, Ireland, 1994), 282–289.
- HULL, D. A., PEDERSEN, J. O., AND SCHÜTZE, H. 1996. Method combination for document filtering. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval* (Zürich, Switzerland, 1996), 279–288.
- ITTNER, D. J., LEWIS, D. D., AND AHN, D. D. 1995. Text categorization of low quality images. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV, 1995), 301–315.
- IWAYAMA, M. AND TOKUNAGA, T. 1995. Cluster-based text categorization: a comparison of category search strategies. In *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995), 273–281.
- IYER, R. D., LEWIS, D. D., SCHAPIRE, R. E., SINGER, Y., AND SINGHAL, A. 2000. Boosting for document routing. In *Proceedings of CIKM-00, 9th ACM International Conference on Information and Knowledge Management* (McLean, VA, 2000), 70–77.
- JOACHIMS, T. 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning* (Nashville, TN, 1997), 143–151.
- JOACHIMS, T. 1998. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning* (Chemnitz, Germany, 1998), 137–142.
- JOACHIMS, T. 1999. Transductive inference for text classification using support vector machines. In *Proceedings of ICML-99, 16th International Conference on Machine Learning* (Bled, Slovenia, 1999), 200–209.
- JOACHIMS, T. AND SEBASTIANI, F. 2002. Guest editors' introduction to the special issue on automated text categorization. *J. Intell. Inform. Syst.* 18, 2/3 (March-May), 103–105.
- JOHN, G. H., KOHAVI, R., AND PFLEGER, K. 1994. Irrelevant features and the subset selection problem. In *Proceedings of ICML-94, 11th International Conference on Machine Learning* (New Brunswick, NJ, 1994), 121–129.
- JUNKER, M. AND ABECKER, A. 1997. Exploiting thesaurus knowledge in rule induction for text classification. In *Proceedings of RANLP-97, 2nd International Conference on Recent Advances in Natural Language Processing* (Tzgov Chark, Bulgaria, 1997), 202–207.
- JUNKER, M. AND HOCH, R. 1998. An experimental evaluation of OCR text representations for learning document classifiers. *Internat. J. Document Analysis and Recognition* 1, 2, 116–122.
- KESSLER, B., NUNBERG, G., AND SCHÜTZE, H. 1997. Automatic detection of text genre. In *Proceedings of ACL-97, 35th Annual Meeting of the Association for Computational Linguistics* (Madrid, Spain, 1997), 32–38.
- KIM, Y.-H., HAHN, S.-Y., AND ZHANG, B.-T. 2000. Text filtering by boosting naive Bayes classifiers. In *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval* (Athens, Greece, 2000), 168–175.
- KLINKENBERG, R. AND JOACHIMS, T. 2000. Detecting concept drift with support vector machines. In *Proceedings of ICML-00, 17th International Conference on Machine Learning* (Stanford, CA, 2000), 487–494.
- KNIGHT, K. 1999. Mining online text. *Commun. ACM* 42, 11, 58–61.
- KNORZ, G. 1982. A decision theory approach to optimal automated indexing. In *Proceedings of SIGIR-82, 5th ACM International Conference on Research and Development in Information Retrieval* (Berlin, Germany, 1982), 174–193.
- KOLLER, D. AND SAHAMI, M. 1997. Hierarchically classifying documents using very few words. In

- Proceedings of ICML-97, 14th International Conference on Machine Learning* (Nashville, TN, 1997), 170–178.
- KORFHAGE, R. R. 1997. *Information Storage and Retrieval*. Wiley Computer Publishing, New York, NY.
- LAM, S. L. AND LEE, D. L. 1999. Feature reduction for neural network based text categorization. In *Proceedings of DASFAA-99, 6th IEEE International Conference on Database Advanced Systems for Advanced Application* (Hsinchu, Taiwan, 1999), 195–202.
- LAM, W. AND HO, C. Y. 1998. Using a generalized instance set for automatic text categorization. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval* (Melbourne, Australia, 1998), 81–89.
- LAM, W., LOW, K. F., AND HO, C. Y. 1997. Using a Bayesian network induction approach for text categorization. In *Proceedings of IJCAI-97, 15th International Joint Conference on Artificial Intelligence* (Nagoya, Japan, 1997), 745–750.
- LAM, W., RUIZ, M. E., AND SRINIVASAN, P. 1999. Automatic text categorization and its applications to text retrieval. *IEEE Trans. Knowl. Data Engin.* 11, 6, 865–879.
- LANG, K. 1995. NEWSWEEDER: learning to filter net-news. In *Proceedings of ICML-95, 12th International Conference on Machine Learning* (Lake Tahoe, CA, 1995), 331–339.
- LARKEY, L. S. 1998. Automatic essay grading using text categorization techniques. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval* (Melbourne, Australia, 1998), 90–95.
- LARKEY, L. S. 1999. A patent search and classification system. In *Proceedings of DL-99, 4th ACM Conference on Digital Libraries* (Berkeley, CA, 1999), 179–187.
- LARKEY, L. S. AND CROFT, W. B. 1996. Combining classifiers in text categorization. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval* (Zürich, Switzerland, 1996), 289–297.
- LEWIS, D. D. 1992a. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of SIGIR-92, 15th ACM International Conference on Research and Development in Information Retrieval* (Copenhagen, Denmark, 1992), 37–50.
- LEWIS, D. D. 1992b. *Representation and Learning in Information Retrieval*. Ph. D. thesis, Department of Computer Science, University of Massachusetts, Amherst, MA.
- LEWIS, D. D. 1995a. Evaluating and optimizing autonomous text classification systems. In *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995), 246–254.
- LEWIS, D. D. 1995b. A sequential algorithm for training text classifiers: corrigendum and additional data. *SIGIR Forum* 29, 2, 13–19.
- LEWIS, D. D. 1995c. The TREC-4 filtering track: description and analysis. In *Proceedings of TREC-4, 4th Text Retrieval Conference* (Gaithersburg, MD, 1995), 165–180.
- LEWIS, D. D. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proceedings of ECML-98, 10th European Conference on Machine Learning* (Chemnitz, Germany, 1998), 4–15.
- LEWIS, D. D. AND CATLETT, J. 1994. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of ICML-94, 11th International Conference on Machine Learning* (New Brunswick, NJ, 1994), 148–156.
- LEWIS, D. D. AND GALE, W. A. 1994. A sequential algorithm for training text classifiers. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval* (Dublin, Ireland, 1994), 3–12. See also Lewis [1995b].
- LEWIS, D. D. AND HAYES, P. J. 1994. Guest editorial for the special issue on text categorization. *ACM Trans. Inform. Syst.* 12, 3, 231.
- LEWIS, D. D. AND RINGUETTE, M. 1994. A comparison of two learning algorithms for text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV, 1994), 81–93.
- LEWIS, D. D., SCHAPIRE, R. E., CALLAN, J. P., AND PAKPA, R. 1996. Training algorithms for linear text classifiers. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval* (Zürich, Switzerland, 1996), 298–306.
- LI, H. AND YAMANISHI, K. 1999. Text classification using ESC-based stochastic decision lists. In *Proceedings of CIKM-99, 8th ACM International Conference on Information and Knowledge Management* (Kansas City, MO, 1999), 122–130.
- LI, Y. H. AND JAIN, A. K. 1998. Classification of text documents. *Comput. J.* 41, 8, 537–546.
- LIDDY, E. D., PAIK, W., AND YU, E. S. 1994. Text categorization for multiple users based on semantic features from a machine-readable dictionary. *ACM Trans. Inform. Syst.* 12, 3, 278–295.
- LIERE, R. AND TADEPALLI, P. 1997. Active learning with committees for text categorization. In *Proceedings of AAAI-97, 14th Conference of the American Association for Artificial Intelligence* (Providence, RI, 1997), 591–596.
- LIM, J. H. 1999. Learnable visual keywords for image classification. In *Proceedings of DL-99, 4th ACM Conference on Digital Libraries* (Berkeley, CA, 1999), 139–145.
- MANNING, C. AND SCHÜTZE, H. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

- MARON, M. 1961. Automatic indexing: an experimental inquiry. *J. Assoc. Comput. Mach.* 8, 3, 404–417.
- MASAND, B. 1994. Optimising confidence of text classification by evolution of symbolic expressions. In *Advances in Genetic Programming*, K. E. Kinnear, ed. MIT Press, Cambridge, MA, Chapter 21, 459–476.
- MASAND, B., LINOFF, G., AND WALTZ, D. 1992. Classifying news stories using memory-based reasoning. In *Proceedings of SIGIR-92, 15th ACM International Conference on Research and Development in Information Retrieval* (Copenhagen, Denmark, 1992), 59–65.
- MCCALLUM, A. K. AND NIGAM, K. 1998. Employing EM in pool-based active learning for text classification. In *Proceedings of ICML-98, 15th International Conference on Machine Learning* (Madison, WI, 1998), 350–358.
- MCCALLUM, A. K., ROSENFELD, R., MITCHELL, T. M., AND NG, A. Y. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of ICML-98, 15th International Conference on Machine Learning* (Madison, WI, 1998), 359–367.
- MERKL, D. 1998. Text classification with self-organizing maps: Some lessons learned. *Neurocomputing* 21, 1/3, 61–77.
- MITCHELL, T. M. 1996. *Machine Learning*. McGraw Hill, New York, NY.
- MLADENIĆ, D. 1998. Feature subset selection in text learning. In *Proceedings of ECML-98, 10th European Conference on Machine Learning* (Chemnitz, Germany, 1998), 95–100.
- MLADENIĆ, D. AND GROBELNIK, M. 1998. Word sequences as features in text-learning. In *Proceedings of ERK-98, the Seventh Electrotechnical and Computer Science Conference* (Ljubljana, Slovenia, 1998), 145–148.
- MOULINIER, I. AND GANASCIA, J.-G. 1996. Applying an existing machine learning algorithm to text categorization. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, S. Wermter, E. Riloff, and G. Schaler, eds. Springer Verlag, Heidelberg, Germany, 343–354.
- MOULINIER, I., RAŠKINIS, G., AND GANASCIA, J.-G. 1996. Text categorization: a symbolic approach. In *Proceedings of SDAIR-96, 5th Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV, 1996), 87–99.
- MYERS, K., KEARNS, M., SINGH, S., AND WALKER, M. A. 2000. A boosting approach to topic spotting on subdialogues. In *Proceedings of ICML-00, 17th International Conference on Machine Learning* (Stanford, CA, 2000), 655–662.
- NG, H. T., GOH, W. B., AND LOW, K. L. 1997. Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval* (Philadelphia, PA, 1997), 67–73.
- NIGAM, K., MCCALLUM, A. K., THRUN, S., AND MITCHELL, T. M. 2000. Text classification from labeled and unlabeled documents using EM. *Mach. Learn.* 39, 2/3, 103–134.
- OH, H.-J., MYAENG, S. H., AND LEE, M.-H. 2000. A practical hypertext categorization method using links and incrementally available class information. In *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval* (Athens, Greece, 2000), 264–271.
- PAZIENZA, M. T., ed. 1997. *Information Extraction*. Lecture Notes in Computer Science, Vol. 1299. Springer, Heidelberg, Germany.
- RILOFF, E. 1995. Little words can make a big difference for text classification. In *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995), 130–136.
- RILOFF, E. AND LEHNERT, W. 1994. Information extraction as a basis for high-precision text classification. *ACM Trans. Inform. Syst.* 12, 3, 296–333.
- ROBERTSON, S. E. AND HARDING, P. 1984. Probabilistic automatic indexing by learning from human indexers. *J. Document.* 40, 4, 264–270.
- ROBERTSON, S. E. AND SPARCK JONES, K. 1976. Relevance weighting of search terms. *J. Amer. Soc. Inform. Sci.* 27, 3, 129–146. Also reprinted in Willett [1988], pp. 143–160.
- ROTH, D. 1998. Learning to resolve natural language ambiguities: a unified approach. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence* (Madison, WI, 1998), 806–813.
- RUIZ, M. E. AND SRINIVASAN, P. 1999. Hierarchical neural networks for text categorization. In *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval* (Berkeley, CA, 1999), 281–282.
- SABLE, C. L. AND HATZIVASSILOGLU, V. 2000. Text-based approaches for non-topical image categorization. *Internat. J. Dig. Libr.* 3, 3, 261–275.
- SALTON, G. AND BUCKLEY, C. 1988. Term-weighting approaches in automatic text retrieval. *Inform. Process. Man.* 24, 5, 513–523. Also reprinted in Sparck Jones and Willett [1997], pp. 323–328.
- SALTON, G., WONG, A., AND YANG, C. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11, 613–620. Also reprinted in Sparck Jones and Willett [1997], pp. 273–280.
- SARACEVIC, T. 1975. Relevance: a review of and a framework for the thinking on the notion in information science. *J. Amer. Soc. Inform. Sci.* 26, 6, 321–343. Also reprinted in Sparck Jones and Willett [1997], pp. 143–165.
- SCHAPIRE, R. E. AND SINGER, Y. 2000. BoosTexter: a boosting-based system for text categorization. *Mach. Learn.* 39, 2/3, 135–168.

- SCHAPIRE, R. E., SINGER, Y., AND SINGHAL, A. 1998. Boosting and Rocchio applied to text filtering. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval* (Melbourne, Australia, 1998), 215–223.
- SCHÜTZE, H. 1998. Automatic word sense discrimination. *Computat. Ling.* 24, 1, 97–124.
- SCHÜTZE, H., HULL, D. A., AND PEDERSEN, J. O. 1995. A comparison of classifiers and document representations for the routing problem. In *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995), 229–237.
- SCOTT, S. AND MATWIN, S. 1999. Feature engineering for text classification. In *Proceedings of ICML-99, 16th International Conference on Machine Learning* (Bled, Slovenia, 1999), 379–388.
- SEBASTIANI, F., SPERDUTI, A., AND VALDAMBRINI, N. 2000. An improved boosting algorithm and its application to automated text categorization. In *Proceedings of CIKM-00, 9th ACM International Conference on Information and Knowledge Management* (McLean, VA, 2000), 78–85.
- SINGHAL, A., MITRA, M., AND BUCKLEY, C. 1997. Learning routing queries in a query zone. In *Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval* (Philadelphia, PA, 1997), 25–32.
- SINGHAL, A., SALTON, G., MITRA, M., AND BUCKLEY, C. 1996. Document length normalization. *Inform. Process. Man.* 32, 5, 619–633.
- SLONIM, N. AND TISHBY, N. 2001. The power of word clusters for text classification. In *Proceedings of ECIR-01, 23rd European Colloquium on Information Retrieval Research* (Darmstadt, Germany, 2001).
- SPARCK JONES, K. AND WILLETT, P., eds. 1997. *Readings in Information Retrieval*. Morgan Kaufmann, San Mateo, CA.
- TAIRA, H. AND HARUNO, M. 1999. Feature selection in SVM text categorization. In *Proceedings of AAAI-99, 16th Conference of the American Association for Artificial Intelligence* (Orlando, FL, 1999), 480–486.
- TAURITZ, D. R., KOK, J. N., AND SPRINKHUIZEN-KUYPER, I. G. 2000. Adaptive information filtering using evolutionary computation. *Inform. Sci.* 122, 2–4, 121–140.
- TUMER, K. AND GHOSH, J. 1996. Error correlation and error reduction in ensemble classifiers. *Connection Sci.* 8, 3–4, 385–403.
- TZERAS, K. AND HARTMANN, S. 1993. Automatic indexing based on Bayesian inference networks. In *Proceedings of SIGIR-93, 16th ACM International Conference on Research and Development in Information Retrieval* (Pittsburgh, PA, 1993), 22–34.
- VAN RIJSBERGEN, C. J. 1977. A theoretical basis for the use of co-occurrence data in information retrieval. *J. Document.* 33, 2, 106–119.
- VAN RIJSBERGEN, C. J. 1979. *Information Retrieval*, 2nd ed. Butterworths, London, UK. Available at <http://www.dcs.gla.ac.uk/Keith>.
- WEIGEND, A. S., WIENER, E. D., AND PEDERSEN, J. O. 1999. Exploiting hierarchy in text categorization. *Inform. Retr.* 1, 3, 193–216.
- WEISS, S. M., APTÉ, C., DAMERAU, F. J., JOHNSON, D. E., OLES, F. J., GOETZ, T., AND HAMPP, T. 1999. Maximizing text-mining performance. *IEEE Intell. Syst.* 14, 4, 63–69.
- WIENER, E. D., PEDERSEN, J. O., AND WEIGEND, A. S. 1995. A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV, 1995), 317–332.
- WILLETT, P., ed. 1988. *Document Retrieval Systems*. Taylor Graham, London, UK.
- WONG, J. W., KAN, W.-K., AND YOUNG, G. H. 1996. ACTION: automatic classification for full-text documents. *SIGIR Forum* 30, 1, 26–41.
- YANG, Y. 1994. Expert network: effective and efficient learning from human decisions in text categorisation and retrieval. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval* (Dublin, Ireland, 1994), 13–22.
- YANG, Y. 1995. Noise reduction in a statistical approach to text categorization. In *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995), 256–263.
- YANG, Y. 1999. An evaluation of statistical approaches to text categorization. *Inform. Retr.* 1, 1–2, 69–90.
- YANG, Y. AND CHUTE, C. G. 1994. An example-based mapping method for text categorization and retrieval. *ACM Trans. Inform. Syst.* 12, 3, 252–277.
- YANG, Y. AND LIU, X. 1999. A re-examination of text categorization methods. In *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval* (Berkeley, CA, 1999), 42–49.
- YANG, Y. AND PEDERSEN, J. O. 1997. A comparative study on feature selection in text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning* (Nashville, TN, 1997), 412–420.
- YANG, Y., SLATTERY, S., AND GHANI, R. 2002. A study of approaches to hypertext categorization. *J. Intell. Inform. Syst.* 18, 2/3 (March-May), 219–241.
- YU, K. L. AND LAM, W. 1998. A new on-line learning algorithm for adaptive text filtering. In *Proceedings of CIKM-98, 7th ACM International Conference on Information and Knowledge Management* (Bethesda, MD, 1998), 156–160.

Received December 1999; revised February 2001; accepted July 2001

# Basic Text Processing

## Regular Expressions



# Regular expressions

A formal language for specifying text strings

How can we search for any of these?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks



# Regular Expressions: Disjunctions

Letters inside square brackets []

Pattern	Matches
<code>[wW]oodchuck</code>	Woodchuck, woodchuck
<code>[1234567890]</code>	Any digit

Ranges `[A-Z]`

Pattern	Matches	
<code>[A-Z]</code>	An upper case letter	<u>D</u> renched Blossoms
<code>[a-z]</code>	A lower case letter	<u>m</u> y beans were impatient
<code>[0-9]</code>	A single digit	Chapter <u>1</u> : Down the Rabbit Hole

# Regular Expressions: Negation in Disjunction

## Negations [ ^Ss ]

- Carat means negation only when first in []

Pattern	Matches	
[ ^A-Z ]	Not an upper case letter	O <u>y</u> fn pripetchik
[ ^Ss ]	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
[ ^e^ ]	Neither e nor ^	Look <u>h</u> ere
a^b	The pattern a carat b	Look up <u>a^b</u> now

# Regular Expressions: More Disjunction

Woodchuck is another name for groundhog!

The pipe | for disjunction

Pattern	Matches
<code>groundhog   woodchuck</code>	<code>woodchuck</code>
<code>yours   mine</code>	<code>yours</code>
<code>a   b   c</code>	<code>= [abc]</code>
<code>[gG]roundhog   [Ww]oodchuck</code>	<code>Woodchuck</code>



# Regular Expressions: ? \* + .

Pattern	Matches	
<code>colou?r</code>	Optional previous char	<u>color</u> <u>colour</u>
<code>oo*h!</code>	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<code>o+h!</code>	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<code>baa+</code>		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
<code>beg.n</code>		<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>



Stephen C Kleene

Kleene \*, Kleene +

# Regular Expressions: Anchors <sup>^</sup> <sup>\$</sup>

Pattern	Matches
<sup>^</sup> [A-Z]	<u>P</u> alo Alto
<sup>^</sup> [ <sup>^</sup> A-Za-z]	<u>1</u> <u>"</u> Hello"
\. <sup>\$</sup>	The end <u>.</u>
.\sup>\$	The end <u>?</u> The end <u>!</u>

# Example

Find me all instances of the word “the” in a text.

`the`

Misses capitalized examples

`[tT]he`

Incorrectly returns `other` or `theology`

`[^a-zA-Z][tT]he[^a-zA-Z]`

# Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched  
(there, then, other)  
**False positives (Type I errors)**
2. Not matching things that we should have matched (The)  
**False negatives (Type II errors)**



# Errors cont.

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- **Increasing accuracy or precision** (minimizing false positives)
- **Increasing coverage or recall** (minimizing false negatives).

# Summary

Regular expressions play a surprisingly large role

- Sophisticated sequences of regular expressions are often the first model for any text processing text

For hard tasks, we use machine learning classifiers

- But regular expressions are still used for pre-processing, or as features in the classifiers
- Can be very useful in capturing generalizations

# Basic Text Processing

Regular Expressions

# Basic Text Processing

## More Regular Expressions: Substitutions and ELIZA

# Substitutions

Substitution in Python and UNIX commands:

```
s/regexp1/pattern/
```

e.g.:

```
s/colour/color/
```

# Capture Groups

- Say we want to put angles around all numbers:  
*the 35 boxes* → *the <35> boxes*
- Use parens () to "capture" a pattern into a numbered register (1, 2, 3...)
- Use \1 to refer to the contents of the register  
`s / ( [ 0 - 9 ] + ) / < \ 1 > /`

# Capture groups: multiple registers

```
/the (.*)er they (.*) , the \1er we \2/
```

Matches

*the faster they ran, the faster we ran*

*But not*

*the faster they ran, the faster we ate*

# But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a ?: after paren:

```
/(?:some|a few) (people|cats) like some \1/
```

matches

- some cats like some cats

but not

- some cats like some some



# Lookahead assertions

`(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**

`(?! pattern)` true if a pattern does not match

How to match, at the beginning of a line, any single word that doesn't start with "Volcano":

```
/^(?!Volcano)[A-Za-z]+/
```

# Simple Application: ELIZA

Early NLP system that imitated a Rogerian psychotherapist

- Joseph Weizenbaum, 1966.

Uses pattern matching to match, e.g.,:

- "I need X"

and translates them into, e.g.

- "What would it mean to you if you got X?"

# Simple Application: ELIZA

Men are all alike.

IN WHAT WAY

They're always bugging us about something or other.

CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.

YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.

I AM SORRY TO HEAR YOU ARE DEPRESSED

# How ELIZA works

s/. \* I'M (depressed|sad) . \*/I AM SORRY TO HEAR YOU ARE \1/

s/. \* I AM (depressed|sad) . \*/WHY DO YOU THINK YOU ARE \1/

s/. \* all . \*/IN WHAT WAY?/

s/. \* always . \*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/

# Basic Text Processing

## More Regular Expressions: Substitutions and ELIZA

# Basic Text Processing

## Words and Corpora

# How many words in a sentence?

"I do uh main- mainly business data processing"

- Fragments, filled pauses

"Seuss's **cat** in the hat is different from other **cats!**"

- **Lemma**: same stem, part of speech, rough word sense
  - **cat** and **cats** = same lemma
- **Wordform**: the full inflected surface form
  - **cat** and **cats** = different wordforms

# How many words in a sentence?

they lay back on the San Francisco grass and looked at the stars and their

**Type:** an element of the vocabulary.

**Token:** an instance of that type in running text.

How many?

- 15 tokens (or 14)
- 13 types (or 12) (or 11?)



# How many words in a corpus?

$N$  = number of tokens

$V$  = vocabulary = set of types,  $|V|$  is size of vocabulary

Heaps Law = Herdan's Law =  $|V| = kN^\beta$  where often  $.67 < \beta < .75$

i.e., vocabulary size grows with  $>$  square root of the number of word tokens

	Tokens = $N$	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

# Corpora

Words don't appear out of nowhere!

A text is produced by

- a specific writer(s),
- at a specific time,
- in a specific variety,
- of a specific language,
- for a specific function.

# Corpora vary along dimension like

- **Language:** 7097 languages in the world
- **Variety**, like African American Language varieties.
  - AAE Twitter posts might include forms like "*iont*" (*I don't*)
- **Code switching**, e.g., Spanish/English, Hindi/English:
  - S/E: Por primera vez veo a @username actually being hateful! It was beautiful:  
*[For the first time I get to see @username actually being hateful! it was beautiful:]*
  - H/E: dost tha or ra- hega ... dont worry ... but dherya rakhe  
*["he was and will remain a friend ... don't worry ... but have faith"]*
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics:** writer's age, gender, ethnicity, SES

# Corpus datasheets

Gebru et al (2020), Bender and Friedman (2018)

## **Motivation:**

- Why was the corpus collected?
- By whom?
- Who funded it?

**Situation:** In what situation was the text written?

**Collection process:** If it is a subsample how was it sampled? Was there consent? Pre-processing?

**+Annotation process, language variety, demographics, etc.**

# Basic Text Processing

## Words and Corpora

# Basic Text Processing

## Word tokenization

# Text Normalization

Every NLP task requires text normalization:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

# Space-based tokenization

## A very simple way to tokenize

- For languages that use space characters between words
  - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
- Segment off a token between instances of spaces

## Unix tools for space-based tokenization

- The "tr" command
- Inspired by Ken Church's UNIX for Poets
- Given a text file, output the word tokens and their frequencies



# Simple Tokenization in UNIX

(Inspired by Ken Church's UNIX for Poets.)

Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt
```

Change all non-alpha to newlines

```
| sort
```

Sort in alphabetical order

```
| uniq -c
```

Merge and count each type

```
1945 A
```

```
72 AARON
```

```
19 ABBESS
```

```
5 ABBOT
```

```
... ..
```

```
25 Aaron
```

```
6 Abate
```

```
1 Abates
```

```
5 Abbess
```

```
6 Abbey
```

```
3 Abbot
```

```
.... ..
```

# The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

THE

SONNETS

by

William

Shakespeare

From

fairest

creatures

We

...

# The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

A

A

A

A

A

A

A

A

A

...

# More counting

## Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

## Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954 d
```

What happened here?

# Issues in Tokenization

Can't just blindly remove punctuation:

- [m.p.h.](#), [Ph.D.](#), [AT&T](#), [cap'n](#)
- prices ([\\$45.55](#))
- dates ([01/02/06](#))
- URLs (<http://www.stanford.edu>)
- hashtags ([#nlproc](#))
- email addresses ([someone@cs.colorado.edu](mailto:someone@cs.colorado.edu))

Clitic: a word that doesn't stand on its own

- "are" in [we're](#), French "je" in [j'ai](#), "le" in [l'honneur](#)

When should multiword expressions (MWE) be words?

- [New York](#), [rock 'n' roll](#)

# Tokenization in NLTK

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)          # set flag to allow verbose regexps
...     ([A-Z]\.)+            # abbreviations, e.g. U.S.A.
...     | \w+(-\w+)*          # words with optional internal hyphens
...     | \$?\d+(\.\d+)?%?    # currency and percentages, e.g. $12.40, 82%
...     | \.\.\.              # ellipsis
...     | [][.,;"'()? :-_']  # these are separate tokens; includes ], [
...     '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

# Tokenization in languages without spaces

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

# Word tokenization in Chinese

Chinese words are composed of characters called "**hanzi**" (or sometimes just "**zi**")

Each one represents a meaning unit called a morpheme.

Each word has on average 2.4 of them.

But deciding what counts as a word is complex and not agreed upon.



# How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

# How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明 进入 总决赛

YaoMing reaches finals

# How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明 进入 总决赛

YaoMing reaches finals

5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

# How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明 进入 总决赛

YaoMing reaches finals

5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

7 characters? (don't use words at all):

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

# Word tokenization / segmentation

So in Chinese it's common to just treat each character (zi) as a token.

- So the **segmentation** step is very simple

In other languages (like Thai and Japanese), more complex word segmentation is required.

- The standard algorithms are neural sequence models trained by supervised machine learning.

# Word tokenization

Basic Text  
Processing

# Byte Pair Encoding

Basic Text  
Processing

# Another option for text tokenization

Instead of

- white-space segmentation
- single-character segmentation

**Use the data** to tell us how to tokenize.

**Subword tokenization** (because tokens can be parts of words as well as whole words)



# Subword tokenization

Three common algorithms:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- **Unigram language modeling tokenization** (Kudo, 2018)
- **WordPiece** (Schuster and Nakajima, 2012)

All have 2 parts:

- A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

# Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

= {A, B, C, D,..., a, b, c, d....}

Repeat:

- Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
- Add a new merged symbol 'AB' to the vocabulary
- Replace every adjacent 'A' 'B' in the corpus with 'AB'.

Until  $k$  merges have been done.

# BPE token learner algorithm

**function** BYTE-PAIR ENCODING(strings  $C$ , number of merges  $k$ ) **returns** vocab  $V$

$V \leftarrow$  all unique characters in  $C$                       # initial set of tokens is characters

**for**  $i = 1$  **to**  $k$  **do**    # merge tokens til  $k$  times

$t_L, t_R \leftarrow$  Most frequent pair of adjacent tokens in  $C$

$t_{NEW} \leftarrow t_L + t_R$     # make new token by concatenating

$V \leftarrow V + t_{NEW}$     # update the vocabulary

Replace each occurrence of  $t_L, t_R$  in  $C$  with  $t_{NEW}$                       # and update the corpus

**return**  $V$

# Byte Pair Encoding (BPE) Addendum

Most subword algorithms are run inside space-separated tokens.

So we commonly first add a special end-of-word symbol '\_\_\_' before space in training corpus

Next, separate into letters.

# BPE token learner

Original (very fascinating 🤔) corpus:

low low low low low lowest lowest newer newer newer  
newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

**vocabulary**

\_, d, e, i, l, n, o, r, s, t, w

# BPE token learner

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulary

\_, d, e, i, l, n, o, r, s, t, w

Merge **e r** to **er**

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulary

\_, d, e, i, l, n, o, r, s, t, w, er

# BPE

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r \_  
3 w i d e r \_  
2 n e w \_

## vocabulary

\_, d, e, i, l, n, o, r, s, t, w, e r

Merge **er \_** to **er\_**

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r\_  
3 w i d e r\_  
2 n e w \_

## vocabulary

\_, d, e, i, l, n, o, r, s, t, w, e r, e r\_

# BPE

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r\_  
3 w i d e r\_  
2 n e w \_

## vocabulary

\_, d, e, i, l, n, o, r, s, t, w, e r, e r\_

## Merge **n e** to **ne**

## corpus

5 l o w \_  
2 l o w e s t \_  
6 n e w e r\_  
3 w i d e r\_  
2 n e w \_

## vocabulary

\_, d, e, i, l, n, o, r, s, t, w, e r, e r\_, ne



# BPE

The next merges are:

Merge	Current Vocabulary
(ne, w)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new
(l, o)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo
(lo, w)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo, low
(new, er—)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo, low, newer—
(low, —)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo, low, newer—, low—

# BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every **e r** to **er**, then merge **er \_** to **er\_**, etc.

Result:

- Test set "n e w e r \_" would be tokenized as a full word
- Test set "l o w e r \_" would be two tokens: "low er\_"

# Properties of BPE tokens

Usually include frequent words

And frequent subwords

- Which are often morphemes like *-est* or *-er*

A **morpheme** is the smallest meaning-bearing unit of a language

- *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

# Byte Pair Encoding

Basic Text  
Processing

# Basic Text Processing

## Word Normalization and other issues

# Word Normalization

## Putting words/tokens in a standard format

- U.S.A. or USA
- uhhuh or uh-huh
- Fed or fed
- am, is, be, are

# Case folding

Applications like IR: reduce all letters to lower case

- Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
  - e.g., *General Motors*
  - *Fed* vs. *fed*
  - *SAIL* vs. *sail*

For sentiment analysis, MT, Information extraction

- Case is helpful (*US* versus *us* is important)

# Lemmatization

Represent all words as their lemma, their shared root  
= dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*
- Spanish **quiero** ('I want'), **quieres** ('you want')  
→ **querer** 'want'
- *He is reading detective stories*  
→ *He be read detective story*



# Lemmatization is done by Morphological Parsing

## Morphemes:

- The small meaningful units that make up words
- **Stems**: The core meaning-bearing units
- **Affixes**: Parts that adhere to stems, often with grammatical functions

## Morphological Parsers:

- Parse *cats* into two morphemes *cat* and *s*
- Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.

# Stemming

Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.



Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note .

# Porter Stemmer

Based on a series of rewrite rules run in series

- A cascade, in which output of each pass fed to next pass

Some sample rules:

ATIONAL → ATE (e.g., relational → relate)

ING → ε if stem contains vowel (e.g., motoring → motor)

SSES → SS (e.g., grasses → grass)

# Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
- **Uygarlastiramadiklarimizdanmissinizcasina**
- `(behaving) as if you are among those whom we could not civilize`
- **Uygar** `civilized` + **las** `become`  
+ **tir** `cause` + **ama** `not able`  
+ **dik** `past` + **lar** `plural`  
+ **imiz** `p1pl` + **dan** `abl`  
+ **mis** `past` + **siniz** `2pl` + **casina** `as if`

# Sentence Segmentation

!, ? mostly unambiguous but **period “.”** is very ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.

- An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

# Basic Text Processing

## Word Normalization and other issues

CHAPTER

7

# Neural Networks and Neural Language Models

“[M]achines of this character can behave in a very complicated manner when the number of units is large.”

Alan Turing (1948) “Intelligent Machines”, page 6

Neural networks are a fundamental computational tool for language processing, and a very old one. They are called neural because their origins lie in the **McCulloch-Pitts neuron** (McCulloch and Pitts, 1943), a simplified model of the human neuron as a kind of computing element that could be described in terms of propositional logic. But the modern use in language processing no longer draws on these early biological inspirations.

feedforward

Instead, a modern neural network is a network of small computing units, each of which takes a vector of input values and produces a single output value. In this chapter we introduce the neural net applied to classification. The architecture we introduce is called a **feedforward network** because the computation proceeds iteratively from one layer of units to the next. The use of modern neural nets is often called **deep learning**, because modern networks are often **deep** (have many layers).

deep learning

Neural networks share much of the same mathematics as logistic regression. But neural networks are a more powerful classifier than logistic regression, and indeed a minimal neural network (technically one with a single ‘hidden layer’) can be shown to learn any function.

Neural net classifiers are different from logistic regression in another way. With logistic regression, we applied the regression classifier to many different tasks by developing many rich kinds of feature templates based on domain knowledge. When working with neural networks, it is more common to avoid most uses of rich hand-derived features, instead building neural networks that take raw words as inputs and learn to induce features as part of the process of learning to classify. We saw examples of this kind of representation learning for embeddings in Chapter 6. Nets that are very deep are particularly good at representation learning. For that reason deep neural nets are the right tool for large scale problems that offer sufficient data to learn features automatically.

In this chapter we’ll introduce feedforward networks as classifiers, and also apply them to the simple task of language modeling: assigning probabilities to word sequences and predicting upcoming words. In subsequent chapters we’ll introduce many other aspects of neural models, such as **recurrent neural networks** and the **Transformer** (Chapter 9), contextual embeddings like **BERT** (Chapter 10), and **encoder-decoder** models and **attention** (Chapter 11).

## 7.1 Units

The building block of a neural network is a single computational unit. A unit takes a set of real valued numbers as input, performs some computation on them, and produces an output.

At its heart, a neural unit is taking a weighted sum of its inputs, with one additional term in the sum called a **bias term**. Given a set of inputs  $x_1 \dots x_n$ , a unit has a set of corresponding weights  $w_1 \dots w_n$  and a bias  $b$ , so the weighted sum  $z$  can be represented as:

$$z = b + \sum_i w_i x_i \quad (7.1)$$

Often it's more convenient to express this weighted sum using vector notation; recall from linear algebra that a **vector** is, at heart, just a list or array of numbers. Thus we'll talk about  $z$  in terms of a weight vector  $w$ , a scalar bias  $b$ , and an input vector  $x$ , and we'll replace the sum with the convenient **dot product**:

$$z = w \cdot x + b \quad (7.2)$$

As defined in Eq. 7.2,  $z$  is just a real valued number.

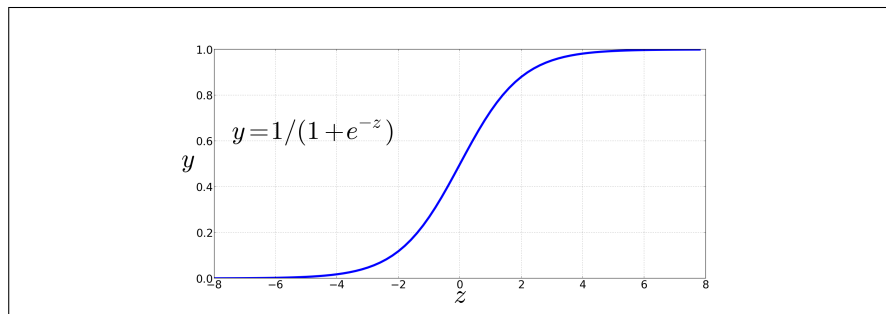
Finally, instead of using  $z$ , a linear function of  $x$ , as the output, neural units apply a non-linear function  $f$  to  $z$ . We will refer to the output of this function as the **activation** value for the unit,  $a$ . Since we are just modeling a single unit, the activation for the node is in fact the final output of the network, which we'll generally call  $y$ . So the value  $y$  is defined as:

$$y = a = f(z)$$

We'll discuss three popular non-linear functions  $f()$  below (the sigmoid, the tanh, and the rectified linear ReLU) but it's pedagogically convenient to start with the **sigmoid** function since we saw it in Chapter 5:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (7.3)$$

The sigmoid (shown in Fig. 7.1) has a number of advantages; it maps the output into the range  $[0, 1]$ , which is useful in squashing outliers toward 0 or 1. And it's differentiable, which as we saw in Section ?? will be handy for learning.



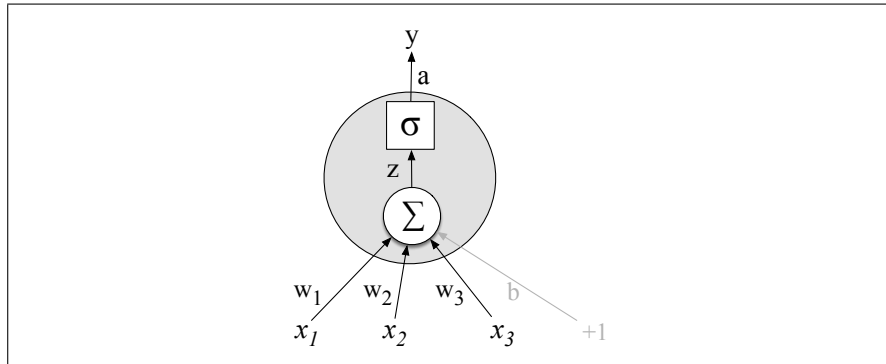
**Figure 7.1** The sigmoid function takes a real value and maps it to the range  $[0, 1]$ . It is nearly linear around 0 but outlier values get squashed toward 0 or 1.



Substituting Eq. 7.2 into Eq. 7.3 gives us the output of a neural unit:

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))} \quad (7.4)$$

Fig. 7.2 shows a final schematic of a basic neural unit. In this example the unit takes 3 input values  $x_1, x_2$ , and  $x_3$ , and computes a weighted sum, multiplying each value by a weight ( $w_1, w_2$ , and  $w_3$ , respectively), adds them to a bias term  $b$ , and then passes the resulting sum through a sigmoid function to result in a number between 0 and 1.



**Figure 7.2** A neural unit, taking 3 inputs  $x_1, x_2$ , and  $x_3$  (and a bias  $b$  that we represent as a weight for an input clamped at +1) and producing an output  $y$ . We include some convenient intermediate variables: the output of the summation,  $z$ , and the output of the sigmoid,  $a$ . In this case the output of the unit  $y$  is the same as  $a$ , but in deeper networks we'll reserve  $y$  to mean the final output of the entire network, leaving  $a$  as the activation of an individual node.

Let's walk through an example just to get an intuition. Let's suppose we have a unit with the following weight vector and bias:

$$\begin{aligned} w &= [0.2, 0.3, 0.9] \\ b &= 0.5 \end{aligned}$$

What would this unit do with the following input vector:

$$x = [0.5, 0.6, 0.1]$$

The resulting output  $y$  would be:

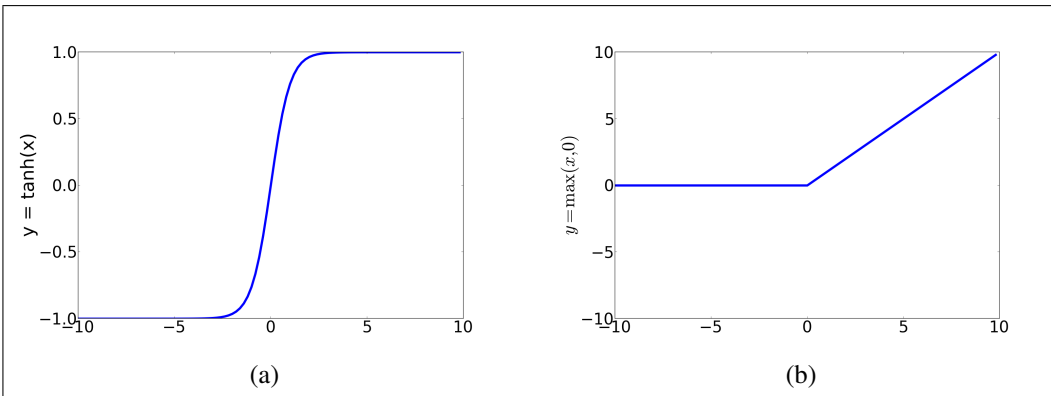
$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{1}{1 + e^{-(.5 \cdot .2 + .6 \cdot .3 + .1 \cdot .9 + .5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

In practice, the sigmoid is not commonly used as an activation function. A function that is very similar but almost always better is the **tanh** function shown in Fig. 7.3a; tanh is a variant of the sigmoid that ranges from -1 to +1:

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (7.5)$$

The simplest activation function, and perhaps the most commonly used, is the rectified linear unit, also called the **ReLU**, shown in Fig. 7.3b. It's just the same as  $x$  when  $x$  is positive, and 0 otherwise:

$$y = \max(x, 0) \quad (7.6)$$



**Figure 7.3** The tanh and ReLU activation functions.

These activation functions have different properties that make them useful for different language applications or network architectures. For example, the tanh function has the nice properties of being smoothly differentiable and mapping outlier values toward the mean. The rectifier function, on the other hand has nice properties that result from it being very close to linear. In the sigmoid or tanh functions, very high values of  $z$  result in values of  $y$  that are **saturated**, i.e., extremely close to 1, and have derivatives very close to 0. Zero derivatives cause problems for learning, because as we'll see in Section 7.4, we'll train networks by propagating an error signal backwards, multiplying gradients (partial derivatives) from each layer of the network; gradients that are almost 0 cause the error signal to get smaller and smaller until it is too small to be used for training, a problem called the **vanishing gradient** problem. Rectifiers don't have this problem, since the derivative of ReLU for high values of  $z$  is 1 rather than very close to 0.

saturated

vanishing gradient

## 7.2 The XOR problem

Early in the history of neural networks it was realized that the power of neural networks, as with the real neurons that inspired them, comes from combining these units into larger networks.

One of the most clever demonstrations of the need for multi-layer networks was the proof by [Minsky and Papert \(1969\)](#) that a single neural unit cannot compute some very simple functions of its input. Consider the task of computing elementary logical functions of two inputs, like AND, OR, and XOR. As a reminder, here are the truth tables for those functions:

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

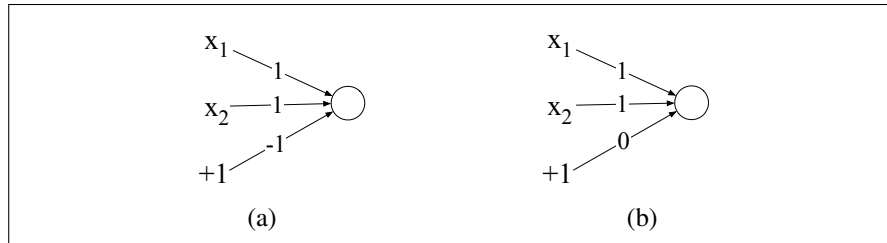
perceptron

This example was first shown for the **perceptron**, which is a very simple neural unit that has a binary output and does **not** have a non-linear activation function. The

output  $y$  of a perceptron is 0 or 1, and is computed as follows (using the same weight  $w$ , input  $x$ , and bias  $b$  as in Eq. 7.2):

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases} \quad (7.7)$$

It's very easy to build a perceptron that can compute the logical AND and OR functions of its binary inputs; Fig. 7.4 shows the necessary weights.



**Figure 7.4** The weights  $w$  and bias  $b$  for perceptrons for computing logical functions. The inputs are shown as  $x_1$  and  $x_2$  and the bias as a special node with value  $+1$  which is multiplied with the bias weight  $b$ . (a) logical AND, showing weights  $w_1 = 1$  and  $w_2 = 1$  and bias weight  $b = -1$ . (b) logical OR, showing weights  $w_1 = 1$  and  $w_2 = 1$  and bias weight  $b = 0$ . These weights/biases are just one from an infinite number of possible sets of weights and biases that would implement the functions.

It turns out, however, that it's not possible to build a perceptron to compute logical XOR! (It's worth spending a moment to give it a try!)

decision  
boundary

The intuition behind this important result relies on understanding that a perceptron is a linear classifier. For a two-dimensional input  $x_1$  and  $x_2$ , the perception equation,  $w_1x_1 + w_2x_2 + b = 0$  is the equation of a line. (We can see this by putting it in the standard linear format:  $x_2 = (-w_1/w_2)x_1 + (-b/w_2)$ .) This line acts as a **decision boundary** in two-dimensional space in which the output 0 is assigned to all inputs lying on one side of the line, and the output 1 to all input points lying on the other side of the line. If we had more than 2 inputs, the decision boundary becomes a hyperplane instead of a line, but the idea is the same, separating the space into two categories.

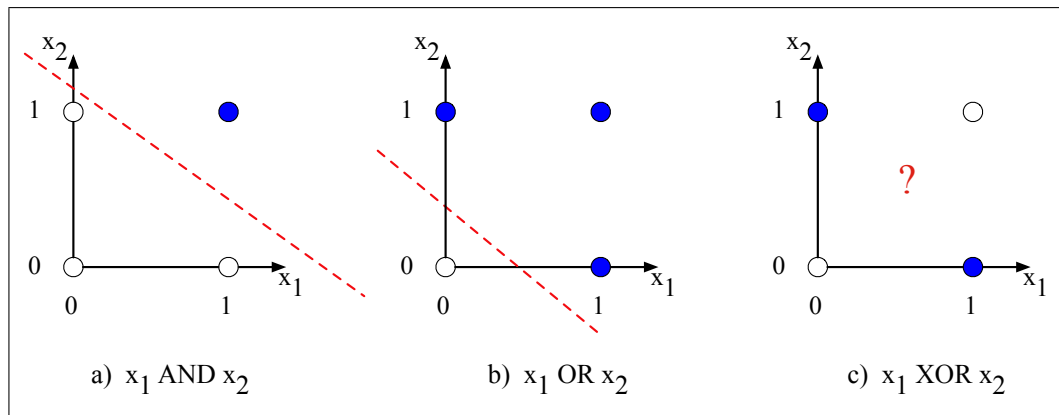
linearly  
separable

Fig. 7.5 shows the possible logical inputs (00, 01, 10, and 11) and the line drawn by one possible set of parameters for an AND and an OR classifier. Notice that there is simply no way to draw a line that separates the positive cases of XOR (01 and 10) from the negative cases (00 and 11). We say that XOR is not a **linearly separable** function. Of course we could draw a boundary with a curve, or some other function, but not a single line.

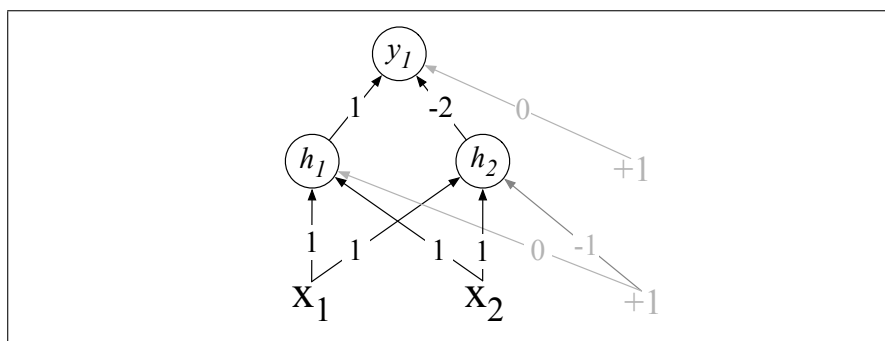
### 7.2.1 The solution: neural networks

While the XOR function cannot be calculated by a single perceptron, it can be calculated by a layered network of units. Let's see an example of how to do this from Goodfellow et al. (2016) that computes XOR using two layers of ReLU-based units. Fig. 7.6 shows a figure with the input being processed by two layers of neural units. The middle layer (called  $h$ ) has two units, and the output layer (called  $y$ ) has one unit. A set of weights and biases are shown for each ReLU that correctly computes the XOR function.

Let's walk through what happens with the input  $x = [0 \ 0]$ . If we multiply each input value by the appropriate weight, sum, and then add the bias  $b$ , we get the



**Figure 7.5** The functions AND, OR, and XOR, represented with input  $x_1$  on the x-axis and input  $x_2$  on the y axis. Filled circles represent perceptron outputs of 1, and white circles perceptron outputs of 0. There is no way to draw a line that correctly separates the two categories for XOR. Figure styled after Russell and Norvig (2002).

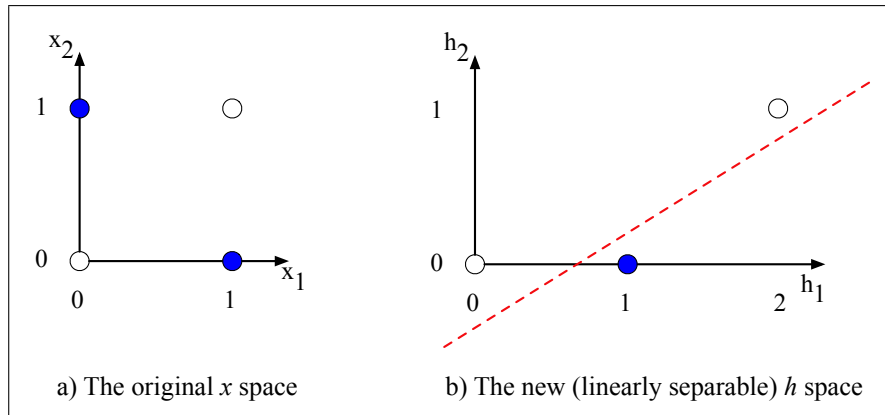


**Figure 7.6** XOR solution after Goodfellow et al. (2016). There are three ReLU units, in two layers; we’ve called them  $h_1$ ,  $h_2$  ( $h$  for “hidden layer”) and  $y_1$ . As before, the numbers on the arrows represent the weights  $w$  for each unit, and we represent the bias  $b$  as a weight on a unit clamped to +1, with the bias weights/units in gray.

vector  $[0 -1]$ , and we then apply the rectified linear transformation to give the output of the  $h$  layer as  $[0 0]$ . Now we once again multiply by the weights, sum, and add the bias (0 in this case) resulting in the value 0. The reader should work through the computation of the remaining 3 possible input pairs to see that the resulting  $y$  values are 1 for the inputs  $[0 1]$  and  $[1 0]$  and 0 for  $[0 0]$  and  $[1 1]$ .

It’s also instructive to look at the intermediate results, the outputs of the two hidden nodes  $h_1$  and  $h_2$ . We showed in the previous paragraph that the  $h$  vector for the inputs  $x = [0 0]$  was  $[0 0]$ . Fig. 7.7b shows the values of the  $h$  layer for all 4 inputs. Notice that hidden representations of the two input points  $x = [0 1]$  and  $x = [1 0]$  (the two cases with XOR output = 1) are merged to the single point  $h = [1 0]$ . The merger makes it easy to linearly separate the positive and negative cases of XOR. In other words, we can view the hidden layer of the network as forming a representation for the input.

In this example we just stipulated the weights in Fig. 7.6. But for real examples the weights for neural networks are learned automatically using the error backpropagation algorithm to be introduced in Section 7.4. That means the hidden layers will learn to form useful representations. This intuition, that neural networks can automatically learn useful representations of the input, is one of their key advantages,



**Figure 7.7** The hidden layer forming a new representation of the input. (b) shows the representation of the hidden layer,  $h$ , compared to the original input representation  $x$  in (a). Notice that the input point  $[0\ 1]$  has been collapsed with the input point  $[1\ 0]$ , making it possible to linearly separate the positive and negative cases of XOR. After Goodfellow et al. (2016).

and one that we will return to again and again in later chapters.

Note that the solution to the XOR problem requires a network of units with non-linear activation functions. A network made up of simple linear (perceptron) units cannot solve the XOR problem. This is because a network formed by many layers of purely linear units can always be reduced (i.e., shown to be computationally identical to) a single layer of linear units with appropriate weights, and we've already shown (visually, in Fig. 7.5) that a single unit cannot solve the XOR problem.

## 7.3 Feed-Forward Neural Networks

feedforward  
network

Let's now walk through a slightly more formal presentation of the simplest kind of neural network, the **feedforward network**. A feedforward network is a multilayer network in which the units are connected with no cycles; the outputs from units in each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers. (In Chapter 9 we'll introduce networks with cycles, called **recurrent neural networks**.)

multi-layer  
perceptrons  
MLP

For historical reasons multilayer networks, especially feedforward networks, are sometimes called **multi-layer perceptrons** (or **MLPs**); this is a technical misnomer, since the units in modern multilayer networks aren't perceptrons (perceptrons are purely linear, but modern networks are made up of units with non-linearities like sigmoids), but at some point the name stuck.

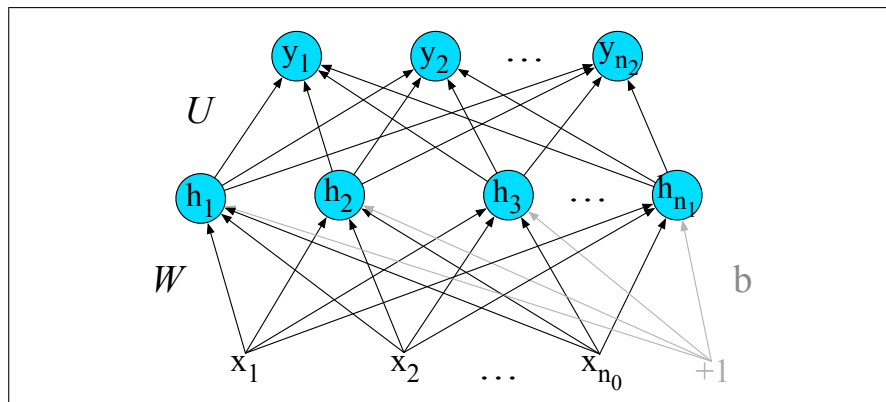
Simple feedforward networks have three kinds of nodes: input units, hidden units, and output units. Fig. 7.8 shows a picture.

hidden layer

The input units are simply scalar values just as we saw in Fig. 7.2.

fully-connected

The core of the neural network is the **hidden layer** formed of **hidden units**, each of which is a neural unit as described in Section 7.1, taking a weighted sum of its inputs and then applying a non-linearity. In the standard architecture, each layer is **fully-connected**, meaning that each unit in each layer takes as input the outputs from all the units in the previous layer, and there is a link between every pair of units from two adjacent layers. Thus each hidden unit sums over all the input units.



**Figure 7.8** A simple 2-layer feedforward network, with one hidden layer, one output layer, and one input layer (the input layer is usually not counted when enumerating layers).

Recall that a single hidden unit has parameters  $w$  (the weight vector) and  $b$  (the bias scalar). We represent the parameters for the entire hidden layer by combining the weight vector  $w_i$  and bias  $b_i$  for each unit  $i$  into a single weight matrix  $W$  and a single bias vector  $b$  for the whole layer (see Fig. 7.8). Each element  $W_{ji}$  of the weight matrix  $W$  represents the weight of the connection from the  $i$ th input unit  $x_i$  to the  $j$ th hidden unit  $h_j$ .

The advantage of using a single matrix  $W$  for the weights of the entire layer is that now the hidden layer computation for a feedforward network can be done very efficiently with simple matrix operations. In fact, the computation only has three steps: multiplying the weight matrix by the input vector  $x$ , adding the bias vector  $b$ , and applying the activation function  $g$  (such as the sigmoid, tanh, or ReLU activation function defined above).

The output of the hidden layer, the vector  $h$ , is thus the following, using the sigmoid function  $\sigma$ :

$$h = \sigma(Wx + b) \quad (7.8)$$

Notice that we're applying the  $\sigma$  function here to a vector, while in Eq. 7.3 it was applied to a scalar. We're thus allowing  $\sigma(\cdot)$ , and indeed any activation function  $g(\cdot)$ , to apply to a vector element-wise, so  $g[z_1, z_2, z_3] = [g(z_1), g(z_2), g(z_3)]$ .

Let's introduce some constants to represent the dimensionalities of these vectors and matrices. We'll refer to the input layer as layer 0 of the network, and have  $n_0$  represent the number of inputs, so  $x$  is a vector of real numbers of dimension  $n_0$ , or more formally  $x \in \mathbb{R}^{n_0}$ , a column vector of dimensionality  $[n_0, 1]$ . Let's call the hidden layer layer 1 and the output layer layer 2. The hidden layer has dimensionality  $n_1$ , so  $h \in \mathbb{R}^{n_1}$  and also  $b \in \mathbb{R}^{n_1}$  (since each hidden unit can take a different bias value). And the weight matrix  $W$  has dimensionality  $W \in \mathbb{R}^{n_1 \times n_0}$ , i.e.  $[n_1, n_0]$ .

Take a moment to convince yourself that the matrix multiplication in Eq. 7.8 will compute the value of each  $h_j$  as  $\sigma(\sum_{i=1}^{n_0} W_{ji}x_i + b_j)$ .

As we saw in Section 7.2, the resulting value  $h$  (for *hidden* but also for *hypothesis*) forms a *representation* of the input. The role of the output layer is to take this new representation  $h$  and compute a final output. This output could be a real-valued number, but in many cases the goal of the network is to make some sort of classification decision, and so we will focus on the case of classification.

If we are doing a binary task like sentiment classification, we might have a single output node, and its value  $y$  is the probability of positive versus negative sentiment.

If we are doing multinomial classification, such as assigning a part-of-speech tag, we might have one output node for each potential part-of-speech, whose output value is the probability of that part-of-speech, and the values of all the output nodes must sum to one. The output layer thus gives a probability distribution across the output nodes.

Let's see how this happens. Like the hidden layer, the output layer has a weight matrix (let's call it  $U$ ), but some models don't include a bias vector  $b$  in the output layer, so we'll simplify by eliminating the bias vector in this example. The weight matrix is multiplied by its input vector ( $h$ ) to produce the intermediate output  $z$ .

$$z = Uh$$

There are  $n_2$  output nodes, so  $z \in \mathbb{R}^{n_2}$ , weight matrix  $U$  has dimensionality  $U \in \mathbb{R}^{n_2 \times n_1}$ , and element  $U_{ij}$  is the weight from unit  $j$  in the hidden layer to unit  $i$  in the output layer.

However,  $z$  can't be the output of the classifier, since it's a vector of real-valued numbers, while what we need for classification is a vector of probabilities. There is a convenient function for **normalizing** a vector of real values, by which we mean converting it to a vector that encodes a probability distribution (all the numbers lie between 0 and 1 and sum to 1): the **softmax** function that we saw on page ?? of Chapter 5. For a vector  $z$  of dimensionality  $d$ , the softmax is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^d e^{z_j}} \quad 1 \leq i \leq d \quad (7.9)$$

Thus for example given a vector  $z=[0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$ ,  $\text{softmax}(z)$  is  $[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$ .

You may recall that softmax was exactly what is used to create a probability distribution from a vector of real-valued numbers (computed from summing weights times features) in logistic regression in Chapter 5.

That means we can think of a neural network classifier with one hidden layer as building a vector  $h$  which is a hidden layer representation of the input, and then running standard logistic regression on the features that the network develops in  $h$ . By contrast, in Chapter 5 the features were mainly designed by hand via feature templates. So a neural network is like logistic regression, but (a) with many layers, since a deep neural network is like layer after layer of logistic regression classifiers, and (b) rather than forming the features by feature templates, the prior layers of the network induce the feature representations themselves.

Here are the final equations for a feedforward network with a single hidden layer, which takes an input vector  $x$ , outputs a probability distribution  $y$ , and is parameterized by weight matrices  $W$  and  $U$  and a bias vector  $b$ :

$$\begin{aligned} h &= \sigma(Wx + b) \\ z &= Uh \\ y &= \text{softmax}(z) \end{aligned} \quad (7.10)$$

We'll call this network a 2-layer network (we traditionally don't count the input layer when numbering layers, but do count the output layer). So by this terminology logistic regression is a 1-layer network.

Let's now set up some notation to make it easier to talk about deeper networks of depth more than 2. We'll use superscripts in square brackets to mean layer numbers, starting at 0 for the input layer. So  $W^{[1]}$  will mean the weight matrix for the

(first) hidden layer, and  $b^{[1]}$  will mean the bias vector for the (first) hidden layer.  $n_j$  will mean the number of units at layer  $j$ . We'll use  $g(\cdot)$  to stand for the activation function, which will tend to be ReLU or tanh for intermediate layers and softmax for output layers. We'll use  $a^{[i]}$  to mean the output from layer  $i$ , and  $z^{[i]}$  to mean the combination of weights and biases  $W^{[i]}a^{[i-1]} + b^{[i]}$ . The 0th layer is for inputs, so the inputs  $x$  we'll refer to more generally as  $a^{[0]}$ .

Thus we can re-represent our 2-layer net from Eq. 7.10 as follows:

$$\begin{aligned} z^{[1]} &= W^{[1]}a^{[0]} + b^{[1]} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= g^{[2]}(z^{[2]}) \\ \hat{y} &= a^{[2]} \end{aligned} \tag{7.11}$$

Note that with this notation, the equations for the computation done at each layer are the same. The algorithm for computing the forward step in an  $n$ -layer feedforward network, given the input vector  $a^{[0]}$  is thus simply:

$$\begin{aligned} &\text{for } i \text{ in } 1..n \\ &\quad z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]} \\ &\quad a^{[i]} = g^{[i]}(z^{[i]}) \\ &\hat{y} = a^{[n]} \end{aligned}$$

The activation functions  $g(\cdot)$  are generally different at the final layer. Thus  $g^{[2]}$  might be softmax for multinomial classification or sigmoid for binary classification, while ReLU or tanh might be the activation function  $g(\cdot)$  at the internal layers.

**Replacing the bias unit** In describing networks, we will often use a slightly simplified notation that represents exactly the same function without referring to an explicit bias node  $b$ . Instead, we add a dummy node  $a_0$  to each layer whose value will always be 1. Thus layer 0, the input layer, will have a dummy node  $a_0^{[0]} = 1$ , layer 1 will have  $a_0^{[1]} = 1$ , and so on. This dummy node still has an associated weight, and that weight represents the bias value  $b$ . For example instead of an equation like

$$h = \sigma(Wx + b) \tag{7.12}$$

we'll use:

$$h = \sigma(Wx) \tag{7.13}$$

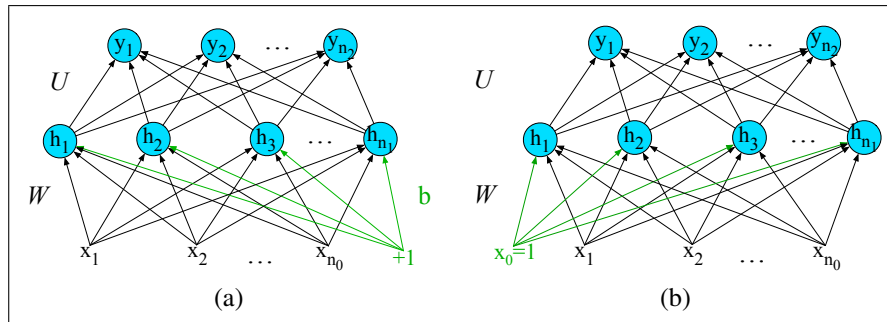
But now instead of our vector  $x$  having  $n$  values:  $x = x_1, \dots, x_n$ , it will have  $n + 1$  values, with a new 0th dummy value  $x_0 = 1$ :  $x = x_0, \dots, x_{n_0}$ . And instead of computing each  $h_j$  as follows:

$$h_j = \sigma \left( \sum_{i=1}^{n_0} W_{ji} x_i + b_j \right), \tag{7.14}$$

we'll instead use:

$$\sigma \left( \sum_{i=0}^{n_0} W_{ji} x_i \right), \tag{7.15}$$





**Figure 7.9** Replacing the bias node (shown in a) with  $x_0$  (b).

where the value  $W_{j0}$  replaces what had been  $b_j$ . Fig. 7.9 shows a visualization.

We'll continue showing the bias as  $b$  for the learning example in the next section, but then we'll switch to this simplified notation without explicit bias terms for the rest of the book.

## 7.4 Training Neural Nets

A feedforward neural net is an instance of supervised machine learning in which we know the correct output  $y$  for each observation  $x$ . What the system produces, via Eq. 7.11, is  $\hat{y}$ , the system's estimate of the true  $y$ . The goal of the training procedure is to learn parameters  $W^{[i]}$  and  $b^{[i]}$  for each layer  $i$  that make  $\hat{y}$  for each training observation as close as possible to the true  $y$ .

In general, we do all this by drawing on the methods we introduced in Chapter 5 for logistic regression, so the reader should be comfortable with that chapter before proceeding.

First, we'll need a **loss function** that models the distance between the system output and the gold output, and it's common to use the loss function used for logistic regression, the **cross-entropy loss**.

Second, to find the parameters that minimize this loss function, we'll use the **gradient descent** optimization algorithm introduced in Chapter 5.

Third, gradient descent requires knowing the **gradient** of the loss function, the vector that contains the partial derivative of the loss function with respect to each of the parameters. Here is one part where learning for neural networks is more complex than for logistic regression. In logistic regression, for each observation we could directly compute the derivative of the loss function with respect to an individual  $w$  or  $b$ . But for neural networks, with millions of parameters in many layers, it's much harder to see how to compute the partial derivative of some weight in layer 1 when the loss is attached to some much later layer. How do we partial out the loss over all those intermediate layers?

The answer is the algorithm called **error backpropagation** or **reverse differentiation**.

### 7.4.1 Loss function

cross-entropy  
loss

The **cross-entropy loss** that is used in neural networks is the same one we saw for logistic regression.

In fact, if the neural network is being used as a binary classifier, with the sig-

moid at the final layer, the loss function is exactly the same as we saw with logistic regression in Eq. ??:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})] \quad (7.16)$$

What about if the neural network is being used as a multinomial classifier? Let  $y$  be a vector over the  $C$  classes representing the true output probability distribution. The cross-entropy loss here is

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^C y_i \log \hat{y}_i \quad (7.17)$$

We can simplify this equation further. Assume this is a **hard classification** task, meaning that only one class is the correct one, and that there is one output unit in  $y$  for each class. If the true class is  $i$ , then  $y$  is a vector where  $y_i = 1$  and  $y_j = 0 \quad \forall j \neq i$ . A vector like this, with one value=1 and the rest 0, is called a **one-hot vector**. The terms in the sum in Eq. 7.17 will be 0 except for the term corresponding to the true class, i.e.:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\sum_{k=1}^K \mathbb{1}\{y = k\} \log \hat{y}_k \\ &= -\sum_{k=1}^K \mathbb{1}\{y = k\} \log \hat{p}(y = k|x) \\ &= -\sum_{k=1}^K \mathbb{1}\{y = k\} \log \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}} \end{aligned} \quad (7.18)$$

negative log likelihood loss

Hence the cross-entropy loss is simply the log of the output probability corresponding to the correct class, and we therefore also call this the **negative log likelihood loss**:

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i, \quad (\text{where } i \text{ is the correct class}) \quad (7.19)$$

Plugging in the softmax formula from Eq. 7.9, and with  $K$  the number of classes:

$$L_{CE}(\hat{y}, y) = -\log \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (\text{where } i \text{ is the correct class}) \quad (7.20)$$

## 7.4.2 Computing the Gradient

How do we compute the gradient of this loss function? Computing the gradient requires the partial derivative of the loss function with respect to each parameter. For a network with one weight layer and sigmoid output (which is what logistic regression is), we could simply use the derivative of the loss that we used for logistic regression in Eq. 7.21 (and derived in Section ??):

$$\begin{aligned} \frac{\partial L_{CE}(w, b)}{\partial w_j} &= (\hat{y} - y) x_j \\ &= (\sigma(w \cdot x + b) - y) x_j \end{aligned} \quad (7.21)$$

Or for a network with one hidden layer and softmax output, we could use the derivative of the softmax loss from Eq. ??:

$$\begin{aligned}\frac{\partial L_{CE}}{\partial w_k} &= (\mathbb{1}\{y = k\} - p(y = k|x))x_k \\ &= \left( \mathbb{1}\{y = k\} - \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^K \exp(w_j \cdot x + b_j)} \right) x_k\end{aligned}\quad (7.22)$$

But these derivatives only give correct updates for one weight layer: the last one! For deep networks, computing the gradients for each weight is much more complex, since we are computing the derivative with respect to weight parameters that appear all the way back in the very early layers of the network, even though the loss is computed only at the very end of the network.

error back-propagation

The solution to computing this gradient is an algorithm called **error backpropagation** or **backprop** (Rumelhart et al., 1986). While backprop was invented specially for neural networks, it turns out to be the same as a more general procedure called **backward differentiation**, which depends on the notion of **computation graphs**. Let's see how that works in the next subsection.

### 7.4.3 Computation Graphs

A computation graph is a representation of the process of computing a mathematical expression, in which the computation is broken down into separate operations, each of which is modeled as a node in a graph.

Consider computing the function  $L(a, b, c) = c(a + 2b)$ . If we make each of the component addition and multiplication operations explicit, and add names ( $d$  and  $e$ ) for the intermediate outputs, the resulting series of computations is:

$$\begin{aligned}d &= 2 * b \\ e &= a + d \\ L &= c * e\end{aligned}$$

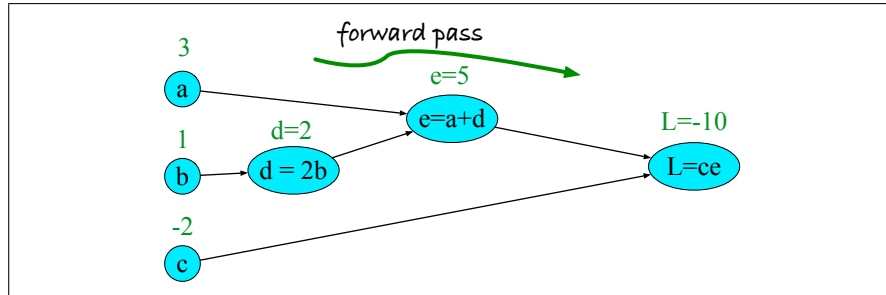
We can now represent this as a graph, with nodes for each operation, and directed edges showing the outputs from each operation as the inputs to the next, as in Fig. 7.10. The simplest use of computation graphs is to compute the value of the function with some given inputs. In the figure, we've assumed the inputs  $a = 3$ ,  $b = 1$ ,  $c = -2$ , and we've shown the result of the **forward pass** to compute the result  $L(3, 1, -2) = -10$ . In the forward pass of a computation graph, we apply each operation left to right, passing the outputs of each computation as the input to the next node.

### 7.4.4 Backward differentiation on computation graphs

The importance of the computation graph comes from the **backward pass**, which is used to compute the derivatives that we'll need for the weight update. In this example our goal is to compute the derivative of the output function  $L$  with respect to each of the input variables, i.e.,  $\frac{\partial L}{\partial a}$ ,  $\frac{\partial L}{\partial b}$ , and  $\frac{\partial L}{\partial c}$ . The derivative  $\frac{\partial L}{\partial a}$ , tells us how much a small change in  $a$  affects  $L$ .

chain rule

Backwards differentiation makes use of the **chain rule** in calculus. Suppose we are computing the derivative of a composite function  $f(x) = u(v(x))$ . The derivative



**Figure 7.10** Computation graph for the function  $L(a, b, c) = c(a + 2b)$ , with values for input nodes  $a = 3$ ,  $b = 1$ ,  $c = -2$ , showing the forward pass computation of  $L$ .

of  $f(x)$  is the derivative of  $u(x)$  with respect to  $v(x)$  times the derivative of  $v(x)$  with respect to  $x$ :

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx} \tag{7.23}$$

The chain rule extends to more than two functions. If computing the derivative of a composite function  $f(x) = u(v(w(x)))$ , the derivative of  $f(x)$  is:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx} \tag{7.24}$$

Let's now compute the 3 derivatives we need. Since in the computation graph  $L = ce$ , we can directly compute the derivative  $\frac{\partial L}{\partial c}$ :

$$\frac{\partial L}{\partial c} = e \tag{7.25}$$

For the other two, we'll need to use the chain rule:

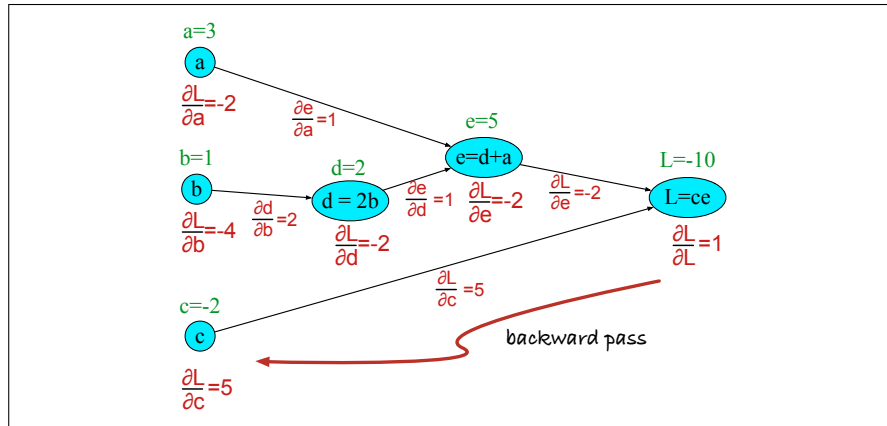
$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b} \end{aligned} \tag{7.26}$$

Eq. 7.26 thus requires five intermediate derivatives:  $\frac{\partial L}{\partial e}$ ,  $\frac{\partial L}{\partial c}$ ,  $\frac{\partial e}{\partial a}$ ,  $\frac{\partial e}{\partial d}$ , and  $\frac{\partial d}{\partial b}$ , which are as follows (making use of the fact that the derivative of a sum is the sum of the derivatives):

$$\begin{aligned} L = ce & : \quad \frac{\partial L}{\partial e} = c, \quad \frac{\partial L}{\partial c} = e \\ e = a + d & : \quad \frac{\partial e}{\partial a} = 1, \quad \frac{\partial e}{\partial d} = 1 \\ d = 2b & : \quad \frac{\partial d}{\partial b} = 2 \end{aligned}$$

In the backward pass, we compute each of these partials along each edge of the graph from right to left, multiplying the necessary partials to result in the final derivative we need. Thus we begin by annotating the final node with  $\frac{\partial L}{\partial L} = 1$ . Moving to the left, we then compute  $\frac{\partial L}{\partial c}$  and  $\frac{\partial L}{\partial e}$ , and so on, until we have annotated the graph all the way to the input variables. The forward pass conveniently already will have computed the values of the forward intermediate variables we need (like  $d$  and  $e$ )

to compute these derivatives. Fig. 7.11 shows the backward pass. At each node we need to compute the local partial derivative with respect to the parent, multiply it by the partial derivative that is being passed down from the parent, and then pass it to the child.



**Figure 7.11** Computation graph for the function  $L(a, b, c) = c(a + 2b)$ , showing the backward pass computation of  $\frac{\partial L}{\partial a}$ ,  $\frac{\partial L}{\partial b}$ , and  $\frac{\partial L}{\partial c}$ .

### Backward differentiation for a neural network

Of course computation graphs for real neural networks are much more complex. Fig. 7.12 shows a sample computation graph for a 2-layer neural network with  $n_0 = 2$ ,  $n_1 = 2$ , and  $n_2 = 1$ , assuming binary classification and hence using a sigmoid output unit for simplicity. The function that the computation graph is computing is:

$$\begin{aligned}
 z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
 a^{[1]} &= \text{ReLU}(z^{[1]}) \\
 z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
 a^{[2]} &= \sigma(z^{[2]}) \\
 \hat{y} &= a^{[2]}
 \end{aligned} \tag{7.27}$$

The weights that need updating (those for which we need to know the partial derivative of the loss function) are shown in orange. In order to do the backward pass, we'll need to know the derivatives of all the functions in the graph. We already saw in Section ?? the derivative of the sigmoid  $\sigma$ :

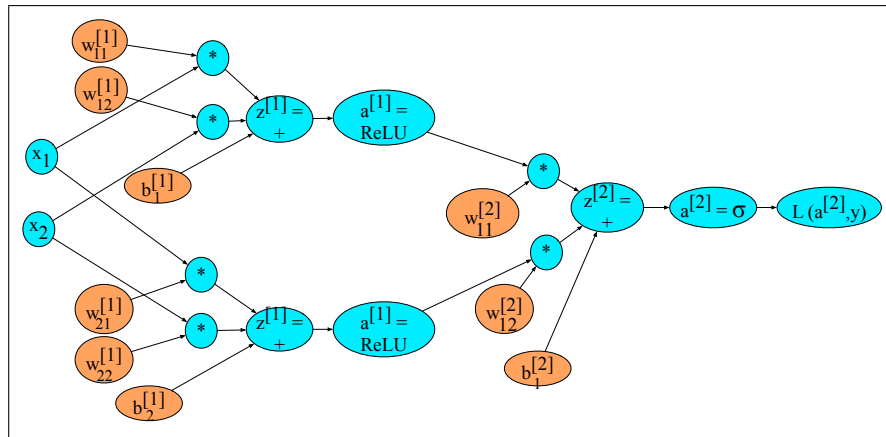
$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)) \tag{7.28}$$

We'll also need the derivatives of each of the other activation functions. The derivative of tanh is:

$$\frac{d \tanh(z)}{dz} = 1 - \tanh^2(z) \tag{7.29}$$

The derivative of the ReLU is

$$\frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{7.30}$$



**Figure 7.12** Sample computation graph for a simple 2-layer neural net (= 1 hidden layer) with two input dimensions and 2 hidden dimensions.

### 7.4.5 More details on learning

Optimization in neural networks is a non-convex optimization problem, more complex than for logistic regression, and for that and other reasons there are many best practices for successful learning.

For logistic regression we can initialize gradient descent with all the weights and biases having the value 0. In neural networks, by contrast, we need to initialize the weights with small random numbers. It's also helpful to normalize the input values to have 0 mean and unit variance.

Various forms of regularization are used to prevent overfitting. One of the most important is **dropout**: randomly dropping some units and their connections from the network during training (Hinton et al. 2012, Srivastava et al. 2014). Tuning of **hyperparameters** is also important. The parameters of a neural network are the weights  $W$  and biases  $b$ ; those are learned by gradient descent. The hyperparameters are things that are chosen by the algorithm designer; optimal values are tuned on a devset rather than by gradient descent learning on the training set. Hyperparameters include the learning rate  $\eta$ , the mini-batch size, the model architecture (the number of layers, the number of hidden nodes per layer, the choice of activation functions), how to regularize, and so on. Gradient descent itself also has many architectural variants such as Adam (Kingma and Ba, 2015).

Finally, most modern neural networks are built using computation graph formalisms that make it easy and natural to do gradient computation and parallelization onto vector-based GPUs (Graphic Processing Units). PyTorch (Paszke et al., 2017) and TensorFlow (Abadi et al., 2015) are two of the most popular. The interested reader should consult a neural network textbook for further details; some suggestions are at the end of the chapter.

## 7.5 Neural Language Models

As our first application of neural networks, let's consider **language modeling**: predicting upcoming words from prior word context.

Neural net-based language models turn out to have many advantages over the n-gram language models of Chapter 3. Among these are that neural language models

don't need smoothing, they can handle much longer histories, and they can generalize over contexts of similar words. For a training set of a given size, a neural language model has much higher predictive accuracy than an  $n$ -gram language model. Furthermore, neural language models underlie many of the models we'll introduce for tasks like machine translation, dialog, and language generation.

On the other hand, there is a cost for this improved performance: neural net language models are strikingly slower to train than traditional language models, and so for many tasks an  $n$ -gram language model is still the right tool.

In this chapter we'll describe simple feedforward neural language models, first introduced by [Bengio et al. \(2003\)](#). Modern neural language models are generally not feedforward but recurrent, using the technology that we will introduce in Chapter 9.

A feedforward neural LM is a standard feedforward network that takes as input at time  $t$  a representation of some number of previous words ( $w_{t-1}, w_{t-2}$ , etc.) and outputs a probability distribution over possible next words. Thus—like the  $n$ -gram LM—the feedforward neural LM approximates the probability of a word given the entire prior context  $P(w_t | w_1 : t-1)$  by approximating based on the  $N$  previous words:

$$P(w_t | w_1, \dots, w_{t-1}) \approx P(w_t | w_{t-N+1}, \dots, w_{t-1}) \quad (7.31)$$

In the following examples we'll use a 4-gram example, so we'll show a net to estimate the probability  $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$ .

### 7.5.1 Embeddings

In neural language models, the prior context is represented by embeddings of the previous words. Representing the prior context as embeddings, rather than by exact words as used in  $n$ -gram language models, allows neural language models to generalize to unseen data much better than  $n$ -gram language models. For example, suppose we've seen this sentence in training:

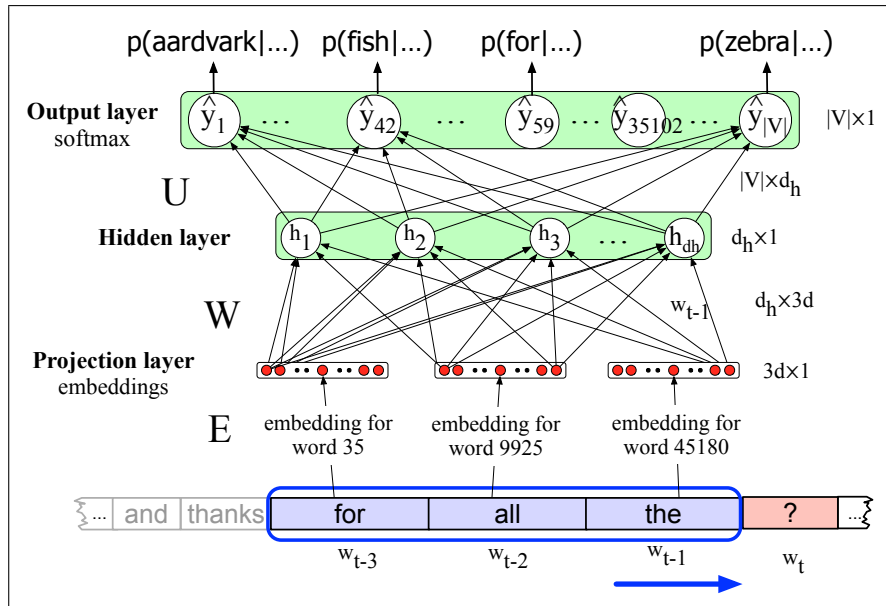
I have to make sure that the cat gets fed.

but have never seen the words “gets fed” after the word “dog”. Our test set has the prefix “I forgot to make sure that the dog gets”. What's the next word? An  $n$ -gram language model will predict “fed” after “that the cat gets”, but not after “that the dog gets”. But a neural LM, knowing that “cat” and “dog” have similar embeddings, will be able to generalize from the “cat” context to assign a high enough probability to “fed” even after seeing “dog”.

Let's see how this works in practice. For now we'll assume we already have an embedding dictionary  $E$  that gives us, for each word in our vocabulary  $V$ , the embedding for that word.

Fig. 7.13 shows a sketch of this simplified feedforward neural language model with  $N=3$ ; we have a moving window at time  $t$  with an embedding vector representing each of the 3 previous words (words  $w_{t-1}$ ,  $w_{t-2}$ , and  $w_{t-3}$ ). These 3 vectors are concatenated together to produce  $x$ , the input layer of a neural network whose output is a softmax with a probability distribution over words. Thus  $y_{42}$ , the value of output node 42 is the probability of the next word  $w_t$  being  $V_{42}$ , the vocabulary word with index 42.

The model shown in Fig. 7.13 is quite sufficient, assuming we have already learned the embeddings separately by a method like the word2vec methods of Chapter 6. Relying on another algorithm to have already learned an embedding represen-



**Figure 7.13** A simplified view of a feedforward neural language model moving through a text. At each timestep  $t$  the network takes the 3 context words, converts each to a  $d$ -dimensional embedding, and concatenates the 3 embeddings together to get the  $1 \times Nd$  unit input layer  $x$  for the network. These units are multiplied by a weight matrix  $W$  and then an activation function is applied element-wise to produce the hidden layer  $h$ , which is then multiplied by another weight matrix  $U$ . Finally, a softmax output layer predicts at each node  $i$  the probability that the next word  $w_t$  will be vocabulary word  $V_i$ . (This picture is simplified because it assumes we just look up in an embedding dictionary  $E$  the  $d$ -dimensional embedding vector for each word, precomputed by an algorithm like word2vec.)

**pretraining** tation for input words is called **pretraining**. If those pretrained embeddings are sufficient for your purposes, then this is all you need.

However, often we'd like to learn the embeddings simultaneously with training the network. This is true when the task the network is designed for (sentiment classification, or translation, or parsing) places strong constraints on what makes a good representation.

Let's therefore show an architecture that allows the embeddings to be learned. To do this, we'll add an extra layer to the network, and propagate the error all the way back to the embedding vectors, starting with embeddings with random values and slowly moving toward sensible representations.

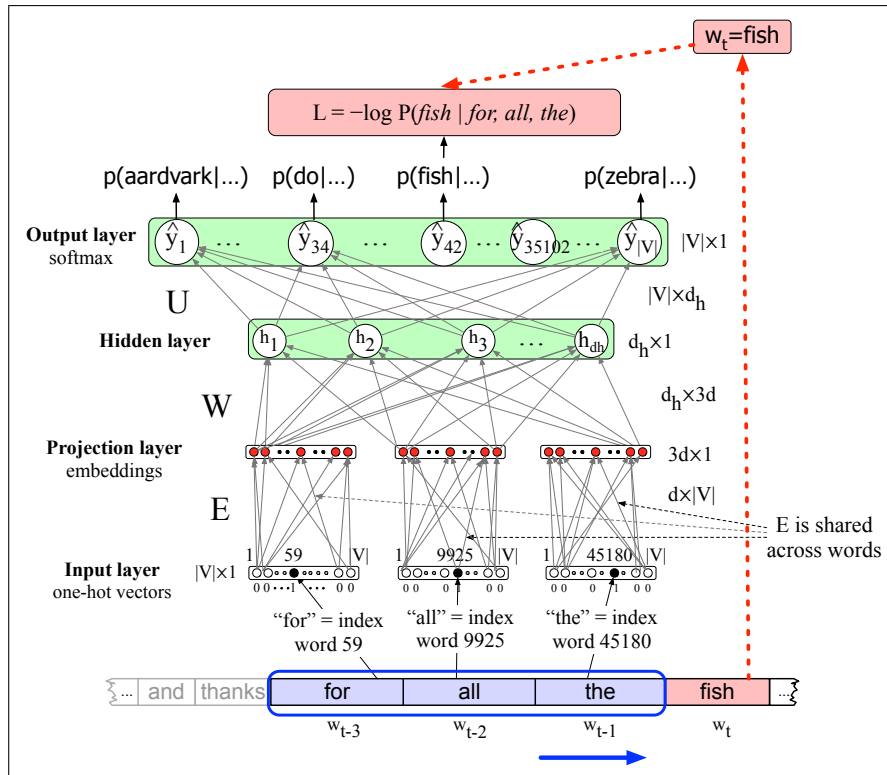
**one-hot vector** For this to work at the input layer, instead of pretrained embeddings, we're going to represent each of the  $N$  previous words as a one-hot vector of length  $|V|$ , i.e., with one dimension for each word in the vocabulary. A **one-hot vector** is a vector that has one element equal to 1—in the dimension corresponding to that word's index in the vocabulary— while all the other elements are set to zero.

Thus in a one-hot representation for the word "toothpaste", supposing it is index 5 in the vocabulary,  $x_5 = 1$ , and  $x_i = 0 \forall i \neq 5$ , as shown here:

$$\begin{bmatrix}
 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots & \dots & |V|
 \end{bmatrix}$$

Fig. 7.14 shows the additional layers needed to learn the embeddings during LM training. Here the  $N=3$  context words are represented as 3 one-hot vectors, fully connected to the embedding layer via 3 instantiations of the embedding matrix  $E$ .





**Figure 7.14** Learning all the way back to embeddings. Notice that the embedding matrix  $E$  is shared among the 3 context words.

Note that we don't want to learn separate weight matrices for mapping each of the 3 previous words to the projection layer, we want one single embedding dictionary  $E$  that's shared among these three. That's because over time, many different words will appear as  $w_{t-2}$  or  $w_{t-1}$ , and we'd like to just represent each word with one vector, whichever context position it appears in. The embedding weight matrix  $E$  thus has a column for each word, each a column vector of  $d$  dimensions, and hence has dimensionality  $d \times |V|$ .

Let's walk through the forward pass of Fig. 7.14.

projection layer

1. **Select three embeddings from  $E$ :** Given the three previous words, we look up their indices, create 3 one-hot vectors, and then multiply each by the embedding matrix  $E$ . Consider  $w_{t-3}$ . The one-hot vector for 'the' (index 35) is multiplied by the embedding matrix  $E$ , to give the first part of the first hidden layer, called the **projection layer**. Since each row of the input matrix  $E$  is just an embedding for a word, and the input is a one-hot column vector  $x_i$  for word  $V_i$ , the projection layer for input  $w$  will be  $Ex_i = e_i$ , the embedding for word  $i$ . We now concatenate the three embeddings for the context words.
2. **Multiply by  $W$ :** We now multiply by  $W$  (and add  $b$ ) and pass through the rectified linear (or other) activation function to get the hidden layer  $h$ .
3. **Multiply by  $U$ :**  $h$  is now multiplied by  $U$
4. **Apply softmax:** After the softmax, each node  $i$  in the output layer estimates the probability  $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$

In summary, if we use  $e$  to represent the projection layer, formed by concatenating the 3 embeddings for the three context vectors, the equations for a neural

language model become:

$$e = (Ex_1, Ex_2, \dots, Ex) \quad (7.32)$$

$$h = \sigma(We + b) \quad (7.33)$$

$$z = Uh \quad (7.34)$$

$$\hat{y} = \text{softmax}(z) \quad (7.35)$$

## 7.5.2 Training the neural language model

To train the model, i.e. to set all the parameters  $\theta = E, W, U, b$ , we do gradient descent (Fig. ??), using error backpropagation on the computation graph to compute the gradient. Training thus not only sets the weights  $W$  and  $U$  of the network, but also as we're predicting upcoming words, we're learning the embeddings  $E$  for each words that best predict upcoming words.

Generally training proceeds by taking as input a very long text, concatenating all the sentences, starting with random weights, and then iteratively moving through the text predicting each word  $w_t$ . At each word  $w_t$ , we use the cross-entropy (negative log likelihood) loss. Recall that the general form for this (repeated from Eq. 7.19) is:

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i, \quad (\text{where } i \text{ is the correct class}) \quad (7.36)$$

For language modeling, the classes are the word in the vocabulary, so  $\hat{y}_i$  here means the probability that the model assigns to the correct next word  $w_t$ :

$$L_{CE} = -\log p(w_t | w_{t-1}, \dots, w_{t-n+1}) \quad (7.37)$$

The parameter update for stochastic gradient descent for this loss from step  $s$  to  $s+1$  is then:

$$\theta^{s+1} = \theta^s - \eta \frac{\partial -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})}{\partial \theta} \quad (7.38)$$

This gradient can be computed in any standard neural network framework which will then backpropagate through  $\theta = E, W, U, b$ .

Training the parameters to minimize loss will result both in an algorithm for language modeling (a word predictor) but also a new set of embeddings  $E$  that can be used as word representations for other tasks.

## 7.6 Summary

- Neural networks are built out of **neural units**, originally inspired by human neurons but now simply an abstract computational device.
- Each neural unit multiplies input values by a weight vector, adds a bias, and then applies a non-linear activation function like sigmoid, tanh, or rectified linear.
- In a **fully-connected, feedforward** network, each unit in layer  $i$  is connected to each unit in layer  $i+1$ , and there are no cycles.
- The power of neural networks comes from the ability of early layers to learn representations that can be utilized by later layers in the network.
- Neural networks are trained by optimization algorithms like **gradient descent**.

- **Error backpropagation**, backward differentiation on a **computation graph**, is used to compute the gradients of the loss function for a network.
- **Neural language models** use a neural network as a probabilistic classifier, to compute the probability of the next word given the previous  $n$  words.
- Neural language models can use pretrained **embeddings**, or can learn embeddings from scratch in the process of language modeling.

## Bibliographical and Historical Notes

The origins of neural networks lie in the 1940s **McCulloch-Pitts neuron** (McCulloch and Pitts, 1943), a simplified model of the human neuron as a kind of computing element that could be described in terms of propositional logic. By the late 1950s and early 1960s, a number of labs (including Frank Rosenblatt at Cornell and Bernard Widrow at Stanford) developed research into neural networks; this phase saw the development of the perceptron (Rosenblatt, 1958), and the transformation of the threshold into a bias, a notation we still use (Widrow and Hoff, 1960).

The field of neural networks declined after it was shown that a single perceptron unit was unable to model functions as simple as XOR (Minsky and Papert, 1969). While some small amount of work continued during the next two decades, a major revival for the field didn't come until the 1980s, when practical tools for building deeper networks like error backpropagation became widespread (Rumelhart et al., 1986). During the 1980s a wide variety of neural network and related architectures were developed, particularly for applications in psychology and cognitive science (Rumelhart and McClelland 1986b, McClelland and Elman 1986, Rumelhart and McClelland 1986a, Elman 1990), for which the term **connectionist** or **parallel distributed processing** was often used (Feldman and Ballard 1982, Smolensky 1988). Many of the principles and techniques developed in this period are foundational to modern work, including the ideas of distributed representations (Hinton, 1986), recurrent networks (Elman, 1990), and the use of tensors for compositionality (Smolensky, 1990).

connectionist

By the 1990s larger neural networks began to be applied to many practical language processing tasks as well, like handwriting recognition (LeCun et al. 1989) and speech recognition (Morgan and Bourlard 1990). By the early 2000s, improvements in computer hardware and advances in optimization and training techniques made it possible to train even larger and deeper networks, leading to the modern term **deep learning** (Hinton et al. 2006, Bengio et al. 2007). We cover more related history in Chapter 9 and Chapter 26.

There are a number of excellent books on the subject. Goldberg (2017) has superb coverage of neural networks for natural language processing. For neural networks in general see Goodfellow et al. (2016) and Nielsen (2015).

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.. Software available from tensorflow.org.
- Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research* 3(Feb), 1137–1155.
- Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. *NeurIPS*.
- Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.
- Feldman, J. A. and Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science* 6, 205–254.
- Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*, Vol. 10 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- Hinton, G. E. (1986). Learning distributed representations of concepts. *COGSCI*.
- Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation* 18(7), 1527–1554.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.
- Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. *ICLR 2015*.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation* 1(4), 541–551.
- McClelland, J. L. and Elman, J. L. (1986). The TRACE model of speech perception. *Cognitive Psychology* 18, 1–86.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5, 115–133.
- Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press.
- Morgan, N. and Bourlard, H. (1990). Continuous speech recognition using multilayer perceptrons with hidden markov models. *ICASSP*.
- Nielsen, M. A. (2015). *Neural networks and Deep learning*. Determination Press USA.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS-W*.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review* 65(6), 386–408.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, Vol. 2, 318–362. MIT Press.
- Rumelhart, D. E. and McClelland, J. L. (1986a). On learning the past tense of English verbs. Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, Vol. 2, 216–271. MIT Press.
- Rumelhart, D. E. and McClelland, J. L. (Eds.). (1986b). *Parallel Distributed Processing*. MIT Press.
- Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach* (2nd Ed.). Prentice Hall.
- Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and brain sciences* 11(1), 1–23.
- Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence* 46(1-2), 159–216.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *JMLR* 15(1), 1929–1958.
- Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. *IRE WESCON Convention Record*, Vol. 4.

# Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation

**Erica Greene**  
Haverford College  
370 Lancaster Ave.  
Haverford, PA 19041  
ericagreene@gmail.com

**Tugba Bodrumlu**  
Dept. of Computer Science  
Univ. of Southern California  
Los Angeles, CA 90089  
bodrumlu@cs.usc.edu

**Kevin Knight**  
Information Sciences Institute  
Univ. of Southern California  
4676 Admiralty Way  
Marina del Rey, CA 90292  
knight@isi.edu

## Abstract

We employ statistical methods to analyze, generate, and translate rhythmic poetry. We first apply unsupervised learning to reveal word-stress patterns in a corpus of raw poetry. We then use these word-stress patterns, in addition to rhyme and discourse models, to generate English love poetry. Finally, we translate Italian poetry into English, choosing target realizations that conform to desired rhythmic patterns.

## 1 Introduction

When it comes to generating creative language (poems, stories, jokes, etc), people have massive advantages over machines:

- people can construct grammatical, sensible utterances,
- people have a wide range of topics to talk about, and
- people experience joy and heart-break.

On the other hand, machines have some minor advantages:

- a machine can easily come up with a five-syllable word that starts with *p* and rhymes with *early*, and
- a machine can analyze very large online text repositories of human works and maintain these in memory.

In this paper we concentrate on statistical methods applied to the analysis, generation, and translation of poetry. By analysis, we mean extracting patterns

from existing online poetry corpora. We use these patterns to generate new poems and translate existing poems. When translating, we render target text in a rhythmic scheme determined by the user.

Poetry generation has received research attention in the past (Manurung et al., 2000; Gervas, 2001; Diaz-Agudo et al., 2002; Manurung, 2003; Wong and Chun, 2008; Tosa et al., 2008; Jiang and Zhou, 2008; Netzer et al., 2009), including the use of statistical methods, although there is a long way to go. One difficulty has been the evaluation of machine-generated poetry—this continues to be a difficulty in the present paper. Less research effort has been spent on poetry analysis and poetry translation, which we tackle here.

## 2 Terms

Meter refers to the rhythmic beat of poetic text when read aloud. Iambic is a common meter that sounds like *da-DUM da-DUM da-DUM*, etc. Each *da-DUM* is called a foot. Anapest meter sounds like *da-da-DUM da-da-DUM da-da-DUM*, etc.

Trimeter refers to a line with three feet, pentameter to a line with five feet, etc. Examples include:

- a VE-ry NAS-ty CUT (iambic trimeter)
- shall I com-PARE thee TO a SUM-mer's DAY? (iambic pentameter)
- twas the NIGHT before CHRIST-mas and ALL through the HOUSE (anapest tetrameter)

Classical English sonnets are poems most often composed of 14 lines of iambic pentameter.

### 3 Analysis

We focus on English rhythmic poetry. We define the following analysis task: *given poetic lines in a known meter (such as sonnets written in iambic pentameter), assign a syllable-stress pattern to each word in each line.* Making such decisions is part of the larger task of reading poetry aloud. Later in the paper, we will employ the concrete statistical tables from analysis to the problems of poetry generation and translation.

We create a test set consisting of 70 lines from Shakespeare’s sonnets, which are written in iambic pentameter. Here is an input line annotated with gold output.

```
shall i compare thee to a summers day
|      |      /\      |      |      |      /\      |
S      S*     S  S*   S      S* S  S* S      S*
```

S refers to an unstressed syllable, and S\* refers to a stressed syllable. One of the authors created gold-standard output by listening to Internet recordings of the 70 lines and marking words according to the speaker’s stress. The task evaluation consists of *per-word accuracy* (how many words are assigned the correct stress pattern) and *per-line accuracy* (how many lines have all words analyzed perfectly).

This would seem simple enough, if we are armed with something like the CMU pronunciation dictionary: we look up syllable-stress patterns for each word token and lay these down on top of the sequence S S\* S S\* S S\* S S\* S S\*. However, there are difficulties:

- The test data contains many words that are unknown to the CMU dictionary.
- Even when all words are known, many lines do not seem to contain 10 syllables. Some lines contain eleven words.
- Spoken recordings include stress reversals, such as poin-TING instead of POIN-ting.
- Archaic pronunciations abound, such as PROV-ed (two syllables) instead of PROVED (one syllable).
- In usage, syllables are often subtracted (PRIS-ner instead of PRIS-o-ner), added (SOV-e-reign instead of SOV-reign), or merged.
- Some one-syllable words are mostly stressed, and others mostly unstressed, but the dictio-

$$e \rightarrow \boxed{P(m|e)} \rightarrow m$$

Figure 1: Finite-state transducer (FST) for mapping sequences of English words (e) onto sequences of S\* and S symbols (m), representing stressed and unstressed syllables.

nary provides no guidance. When we generate rhythmic text, it is important to use one-syllable words properly. For example, we would be happy for an iambic generator to output *big thoughts are not quite here*, but not *quite big thoughts are not here*.

Therefore, we take a different tack and apply unsupervised learning to acquire word-stress patterns directly from raw poetry, without relying on a dictionary. This method easily ports to other languages, where dictionaries may not exist and where morphology is a severe complication. It may also be used for dead languages.

For raw data, we start with all Shakespeare sonnets (17,134 word tokens). Because our learning is unsupervised, we do not mind including our 70-line test set in this data (*open testing*).

Figures 1 and 2 show a finite-state transducer (FST) that converts sequences of English words to sequences of S\* and S symbols. The FST’s transitions initially map each English word onto all output sub-sequences of lengths 1 to 4 (i.e., S, S\*, S-S, S-S\*, S\*-S, S\*-S\*, S-S-S, ...) plus the sequences S-S\*-S-S\*-S and S\*-S-S\*-S-S\*. Initial probabilities are set to 1/32. The FST’s main loop allows it to process a sequence of word tokens. If the same word appears twice in a sequence, then it may receive two different pronunciations, since the mapping is probabilistic. However, a token’s syllable/stress pattern is chosen independently of other tokens in the sequence; we look at relaxing this assumption later.

We next use finite-state EM training<sup>1</sup> to train the machine on input/output sequences such as these:

```
from fairest creatures we desire increase
S S* S S* S S* S S* S S*
```

```
but thou contracted to thine own bright eyes
S S* S S* S S* S S* S S*
```

<sup>1</sup>All operations in this paper are carried out with the generic finite-state toolkit Carmel (Graehl, 1997). For example, the *train-cascade* command uses EM to learn probabilities in an arbitrary FST cascade from end-to-end input/output string pairs.

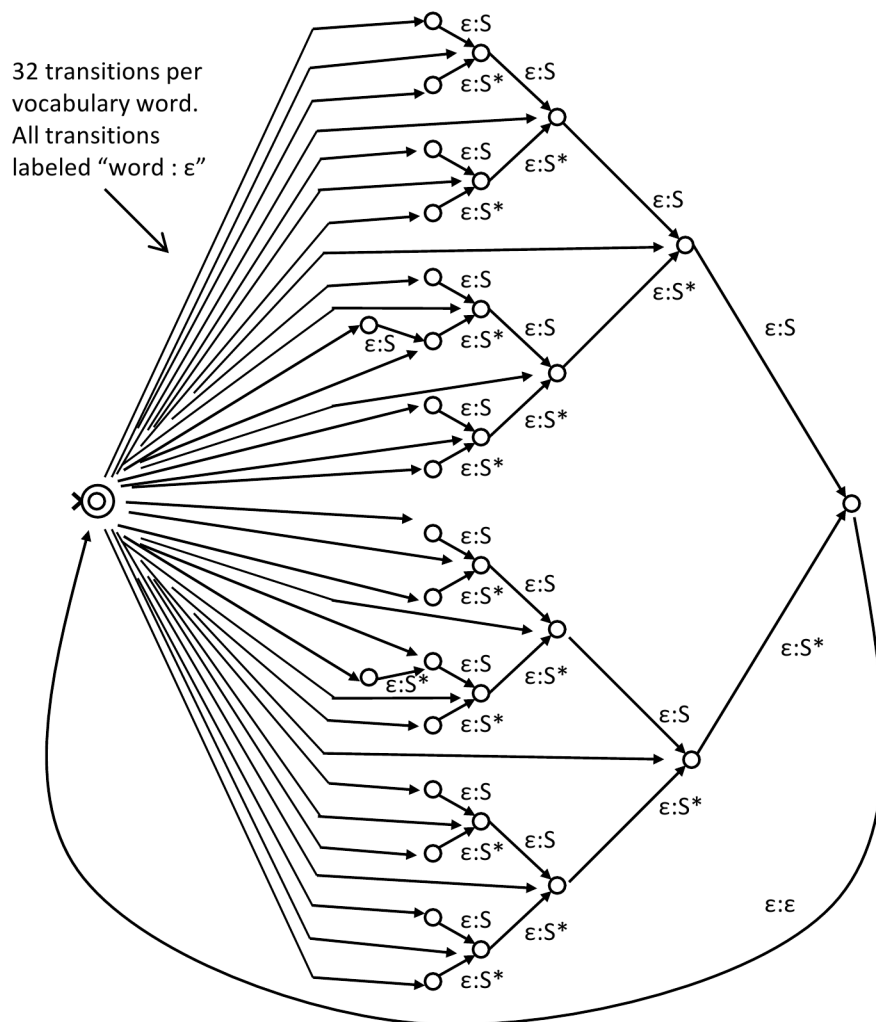


Figure 2: An efficient FST implementing  $P(m|e)$ . This machine maps sequences of English words onto sequences of  $S^*$  and  $S$  symbols, representing stressed and unstressed syllables. Initially every vocabulary word has 32 transitions, each with probability  $1/32$ . After EM training, far fewer transitions remain.

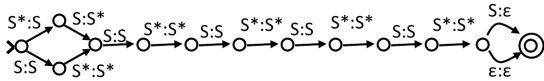


Figure 3: An FST that accepts any of four input meters and deterministically normalizes its input to strict iambic pentameter. We call this FST *norm*.

$$e \rightarrow \boxed{P(m|e)} \rightarrow m \rightarrow \boxed{norm} \rightarrow m$$

Figure 4: FST cascade that encodes a loose interpretation of iambic pentameter. The *norm* FST accepts any of four near-iambic-pentameter sequences and normalizes them into strict iambic pentameter.

Note that the output sequences are all the same, representing our belief that each line should be read as iambic pentameter.<sup>2</sup> After we train the FST, we can use Viterbi decoding to recover the highest-probability alignments, e.g.:

```

from fairest creatures we desire increase
|   |   / \   |   /\   /\
S   S*  S S* S  S* S S*  S S*

but thou contracted to thine own bright eyes
|   |   / \   |   |   |   |
S   S* S S* S  S* S  S* S  S*

```

Note that the first example contains an error—the words *fairest* and *creatures* should each be read with two syllables. There are many such errors. We next improve the system in two ways: more data and better modeling.

First, we augment the Shakespeare sonnets with data from the website *sonnets.org*, increasing the number of word tokens from 17,134 to 235,463. The *sonnets.org* data is noisier, because it contains some non-iambic-pentameter poetry, but overall we find that alignments improve, e.g.:

```

from fairest creatures we desire increase
|   /\   /\   |   /\   /\
S   S* S  S* S  S* S S*  S S*

```

Second, we loosen our model. When we listen to recordings, we discover that not all lines are read  $S^* S S^* S S^* S S^* S S^*$ . Indeed, some lines in our data contain eleven words—these are unexplainable by the EM training system. We also observe that

<sup>2</sup>We can augment the data with lines of poetry written in meters other than iambic pentameter, so long as we supply the desired output pattern for each input line.

Training data	Training tokens	Test token accuracy	Test line accuracy
Shakespeare	17,134	82.3%	55.7%
sonnets.org	235,463	94.2%	81.4%

Figure 5: Analysis task accuracy.

poets often use the word *mother* ( $S^* S$ ) at the beginnings and ends of lines, where it theoretically should not appear.

Two well-known variations explain these facts. One is optional *inversion* of the first foot ( $S S^* \rightarrow S^* S$ ). Second is the optional addition of an eleventh unstressed syllable (the *feminine ending*). These variations yield four possible syllable-stress sequences:

```

S  S* S S* S S* S S* S S*
S* S  S S* S S* S S* S S*
S  S* S S* S S* S S* S S* S
S* S  S S* S S* S S* S S* S

```

We want to offer EM the freedom to analyze lines into any of these four variations. We therefore construct a second FST (Figure 3), *norm*, which maps all four sequences onto the canonical pattern  $S S^* S S^* S S^* S S^* S S^*$ . We then arrange both FSTs in a cascade (Figure 4), and we train the whole cascade on the same input/output sequences as before. Because *norm* has no trainable parameters, we wind up training only the lexical mapping parameters. Viterbi decoding through the two-step cascade now reveals EM’s proposed internal meter analysis as well as token mappings, e.g.:

```

to be or not to be that is the question
| | | | | | | | |
S S* S  S* S  S* S  S* S  S* S
| | | | | | | | |
S S* S  S* S  S* S  S* S  S* S

```

Figure 5 shows accuracy results on the 70-line test corpus mentioned at the beginning of this section. **Over 94% of word tokens are assigned a syllable-stress pattern that matches the pattern transcribed from audio. Over 81% of whole lines are also scanned correctly. The upper limit for whole-line scanning under our constraints is 88.6%, because 11.4% of gold outputs do not match any of the four patterns we allow.**

We further obtain a probabilistic table of word mappings that we can use for generation and trans-



$P(S^* S S^* \mid \text{altitude})$	$= 1.00$
$P(S^* S \mid \text{creatures})$	$= 1.00$
$P(S^* S \mid \text{pointed})$	$= 0.95$
$P(S S^* \mid \text{pointed})$	$= 0.05$
$P(S^* S \mid \text{prisoner})$	$= 0.74$
$P(S^* S S^* \mid \text{prisoner})$	$= 0.26$
$P(S^* S \mid \text{mother})$	$= 0.95$
$P(S^* \mid \text{mother})$	$= 0.03$
$P(S S^* \mid \text{mother})$	$= 0.02$

Figure 6: Sample learned mappings between words and syllable-stress patterns.

word	$P(S^* \mid \text{word})$	$P(S \mid \text{word})$
a	0.04	0.96
the	0.06	0.94
their	0.09	0.91
mens	0.10	0.90
thy	0.10	0.90
be	0.48	0.52
me	0.49	0.51
quick	0.50	0.50
split	0.50	0.50
just	0.51	0.49
food	0.90	0.10
near	0.90	0.10
raised	0.91	0.09
dog	0.93	0.07
thought	0.95	0.05

Figure 7: Sample mappings for one-syllable words.

lation tasks. Figure 6 shows a portion of this table. Note that  $P(S S^* \mid \text{mother})$  has a very small probability of 0.02. We would incorrectly learn a much higher value if we did not loosen the iambic pentameter model, as many *mother* tokens occur line-initial and line-final.

Figure 7 shows which one-syllable words are more often stressed (or unstressed) in iambic pentameter poetry. Function words and possessives tend to be unstressed, while content words tend to be stressed, though many words are used both ways. This useful information is not available in typical pronunciation dictionaries.

Alignment errors still occur, especially in noisy

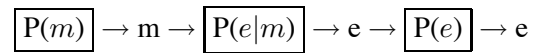


Figure 8: Finite-state cascade for poetry generation.

portions of the data that are not actually written in iambic pentameter, but also in clean portions, e.g.:

the perfect ceremony of loves rite  
 $\begin{array}{cccccccc} | & / \backslash & / \backslash & | & | & / \backslash \\ S & S^* S & S^* S S^* & S & S^* & S & S^* \end{array}$

The word *ceremony* only occurs this once in the data, so it is willing to accept any stress pattern. While *rite* is correctly analyzed elsewhere as a one-syllable word, *loves* prefers  $S^*$ , and this overwhelms the one-syllable preference for *rite*. We can blame our tokenizer for this, as it conflates *loves* and *love's*, despite the fact that these words have different stress probabilities.

## 4 Generation

Figure 8 shows our concept of generation as a cascade of weighted FSTs.

$P(m)$  is a user-supplied model of desired meters—normally it deterministically generates a single string of  $S^*$  and  $S$  symbols. (The user also supplies a rhyme scheme—see below).

$P(e|m)$  is the reverse of Section 3's  $P(m|e)$ , being a model of word selection. Its generative story is: (1) probabilistically select  $n$  tokens ( $n = 1$  to 5) from the input, (2) probabilistically select a word  $w$  that realizes that  $n$ -token sequence, and (3) recurse until the input is consumed.

Instead of asking how a given word is likely to be pronounced (e.g.,  $S$  or  $S^*$ ), we now ask how a given stress-pattern (e.g.,  $S$  or  $S^*$ ) is likely to be realized. This model is trained with the same method described in Section 3 and is augmented with the CMU pronunciation dictionary.

Finally,  $P(e)$  is a word-trigram model built from a 10,000-line corpus of 105 English love poems.

We select the first line of our poem from the FST cascade's 100,000-best list, or by hand. To generate each subsequent line, we modify the cascade and run it again. The first modification is to incorporate a discourse model. From our poetry corpus, we estimate a word's unigram probability given the words on the previous line, via IBM Model 1 (Brown et al., 1993). We modify  $P(e)$  by interpolating in these probabilities. Second, we check if any previous line

The women of the night  
 Again and all the way  
 Like a mouse in the white  
 Not the heart of the day.  
 - - -  
 Of the bed to trust me  
 Around her twists the string  
 But i will not tell thee  
 Fire changes everything.  
 - - -  
 A son of the right hand confines  
 His uncle could have broken in  
 Towards the high bank and the pines  
 Upon the eyes and i have been  
 - - -  
 Into one of her hundred year old  
 Or the house in a house in a cold  
 The first time she met him  
 Like a mouse in the dim  
 For me to the moon and when i told  
 - - -  
 Into one of them some years before  
 His own man or the house in a more  
 The moon and when the day  
 Into one of the way  
 With the breath from the first time  
 she swore

Figure 9: Sample poems generated with a weighted FST cascade.

$w_1, w_2, \dots, w_n$  needs to be rhymed with, according to the user-supplied scheme. If so, we build an additional FST that accepts only strings whose final word rhymes with  $w_n$ . This is a reasonable approach, though it will not, for example, rhyme *...tar me* with *...army*. We say two non-identical words rhyme if their phoneme strings share a common suffix that includes the last stressed vowel.

Figure 9 shows several poems that we automatically generate with this scheme.

## 5 Translation

Automatically generated poetry can sound good when read aloud, but it often has a “nonsense” feel to it. According to (Gervas, 2010), creative-language researchers interested in realization and surface language statistics (“how to say”) have tended to gravitate to poetry generation, while researchers interested in characters, goals, and story-line (“what to say”) have tended to gravitate to prose story generation.

Translation provides one way to tie things to-

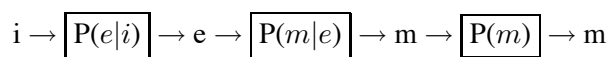


Figure 10: Finite-state cascade for poetry translation.

gether. The source language provides the input (“what to say”), and the target language can be shaped to desired specifications (“how to say”). For example, we may want to translate Italian sonnets into fluent English iambic pentameter. This is certainly a difficult task for people, and one which is generally assumed to be impossible for computers.

Here we investigate translating Dante’s *Divine Comedy* (DC) from Italian into English by machine. The poem begins:

nel mezzo del cammin di nostra vita  
 mi ritrovai per una selva oscura  
 che la via diritta era smarrita.

DC is a long sequence of such three-line stanzas (*tercets*). The meter in Italian is hendecasyllabic, which has ten syllables and ensures three beats. Dante’s Italian rhyme scheme is: **ABA, BCB, CDC**, etc, meaning that lines 2, 4, and 6 rhyme with each other; lines 5, 7, and 9 rhyme with each other, and so forth. There is also internal rhyme (e.g., *diritta/smarrita*).

Because DC has been translated many times into English, we have examples of good outputs. Some translations target iambic pentameter, but even the most respected translations give up on rhyme, since English is much harder to rhyme than Italian. Longfellow’s translation begins:

midway upon the journey of our life  
 i found myself within a forest dark  
 for the straightforward pathway had  
 been lost.

We arrange the translation problem as a cascade of WFSTs, as shown in Figure 10. We call our Italian input  $i$ . In lieu of the first WFST, we use the statistical phrase-based machine translation (PBMT) system Moses (Koehn et al., 2007), which generates a target-language lattice with paths scored by  $P(e|i)$ . We send this lattice through the same  $P(m|e)$  device we trained in Section 3. Finally, we filter the resulting syllable sequences with a strict, single-path, deterministic iambic pentameter acceptor,  $P(m)$ .<sup>3</sup> Our

<sup>3</sup>It is also possible to use a looser iambic  $P(m)$  model, as described in Section 3.

### Parallel Italian/English Data

Collection	Word count (English)
DC-train	400,670
Il Fiore	25,995
Detto Damare	2,483
Egloghe	3,120
Misc.	557
<i>Europarl</i>	32,780,960

### English Language Model Data

Collection	Word count (English)
DC-train	400,670
poemhunter.com poetry.eserver.org poetrymountain.com	686,714
poetryarchive.org	58,739
everypoet.com	574,322
sonnets.org	166,465
<i>Europarl</i>	32,780,960

### Tune and Blind Test Data (4 reference)

Collection	Word count (Italian)
DC-tune	7,674
DC-test	2,861

Figure 11: Data for Italian/English statistical translation.

finite-state toolkit’s top-k paths represent the translations with the highest product of scores  $P(e|i) \cdot P(m|e) \cdot P(m)$ .

In general, the  $P(e|i)$  and  $P(m|e)$  models fight each other in ranking candidate outputs. In experiments, we find that the  $P(e|i)$  preference is sometimes so strong that the  $P(m|e)$  model is pushed into using a low-probability word-to-stress mapping. This creates output lines that do not scan easily. We solve this problem by assigning a higher weight to the  $P(m|e)$  model.<sup>4</sup>

Figure 11 shows the data we used to train the PBMT system. The vast majority of parallel Italian/English poetry is DC itself, for which we have four English translations. We break DC up into DC-train, DC-tune, and DC-test. We augment our target language model with English poetry collected from many sources. We also add Europarl data, which

<sup>4</sup>We set this weight manually to 3.0, i.e., we raise all probabilities in the  $P(m|e)$  model to the power of 3.0. Setting the weight too high results in lines that scan very well, but whose translation quality is low.

<i>Original:</i>
nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che la via diritta era smarrita.
<i>Phrase-based translation (PBMT):</i>
midway in the journey of our life i found myself within a forest dark for the straight way was lost.
<i>PBMT + meter model:</i>
midway upon the journey of our life i found myself within a forest dark for the straightforward pathway had been lost.

Figure 12: Automatic translation of lines from Dante’s *Divine Comedy*. In this test-on-train scenario, the machine reproduces lines from human translations it has seen.

is out of domain, but which reduces the unknown word-token rate in DC-test from 9% to 6%, and the unknown word-type rate from 22% to 13%.

We first experiment in a test-on-train scenario, where we translate parts of DC that are in our training set. This is a normal scenario in human poetry translation, where people have access to previous translations.

Figure 12 shows how we translate the first lines of DC, first using only PBMT, then using the full system. When we use the full system, we not only get an output string, but also the system’s intended scan, e.g.:

```
midway upon the journey of our life
/\      /\      |      /\      |      |      |
S  S*  S  S* S  S* S  S*  S  S*
```

The machine’s translation here is the same as Longfellow’s, which is in the training data. In other cases, we observe the machine combining existing translations, e.g.:

```
i:  bedi la bestia per cu io mi volsi
I5: behold the beast that made me turn aside

H1: BEHOLD THE BEAST for which i have turned back
H2: you see the beast THAT MADE ME TURN ASIDE
H3: see the beast that forced me to turn back
H4: look at the beast that drove me to turn back
```

I5 refs to the machine’s iambic pentameter transla-

tion, while H1-4 refer to human translations. The machine also creates new translations:

i: diro' de laltre cose chi vho scorte  
 I5: i shall explain the other things i saw

H1: speak will i of the other things i saw there  
 H2: ill also tell THE OTHER THINGS I SAW  
 H3: i will recount the other things i saw  
 H4: i here will tell the other things i saw

We can further change the target meter to anything we desire. To obtain iambic tetrameter (4-beat) translations, we delete the last two transitions of the  $P(m)$  model. We then get:

I4: in our life the journey way  
 i found myself deep on dark wood  
 that lost straightforward pathway had.

ah how to say the what is hard  
 this forest savage rough and stern  
 the very thought renews the fear.

Translations and scans are uneven, but we have significant flexibility. We can even request translations that avoid the English letter A, by adding a filter to the end of the FST cascade, obtaining:

I5: in midst upon the journey of our life  
 i found myself within the wood obscure  
 <fail>

To steer clear of the adjective *dark* in the second line, the system switches from *forest* to *wood*, so obtain a proper scan. The third line fails because all paths through the translation lattice contain an A somewhere.

Translating blind-test data proves to be more difficult. We hold out Canto XXXI of DC's *Paradiso* section for testing. Figure 13 shows a portion of the translation results. The MT system handles unknown Italian words by passing them through to the output. The  $P(m|e)$  meter model cannot process those words, accounting for the I5 failure rate.

Here, we get a first look at statistical MT translating poetry into rhythmic structures—as with all MT, there are successes and problems, and certainly more to do.

## 6 Future Work

We plan to release all our data in useful, processed form. Below we list directions for future research. In general, we see many interesting paths to pursue.

**Analysis.** Proper use of one-syllable words remains tricky. Lines coming out of generation

<p><i>Original:</i></p> <p>in forma dunque di candida rosa  mi si mostrava la milizia santa  che nel suo sangue cristo fece sposa</p> <p>ma laltra che volando vede e canta  la gloria di colui che la nnamora  e la bonta' che la fece cotanta</p>
<p><i>Human translation:</i></p> <p>in fashion then as of a snow white rose  displayed itself to me the saintly host  whom christ in his own blood had made his bride</p> <p>but the other host that flying sees and sings  the glory of him who doth enamour it  and the goodness that created it so noble</p>
<p><i>Phrase-based translation (PBMT):</i></p> <p>in the form so rose candida  i now was shown the militia holy  that in his blood christ did bride</p> <p>but the other that flying sees and sings  the glory of him that the nnamora  and the goodness that the made cotanta</p>
<p><i>PBMT + meter model:</i></p> <p>&lt;fail&gt;  i now was shown the holy soldiery  that in his blood he married jesus christ</p> <p>but flying sees and sings the other which  &lt;fail&gt;  &lt;fail&gt;</p>

Figure 13: Automatic translation of blind-test data from Dante's *Divine Comedy*.

and translation do not always scan naturally when read aloud by a person. We trace such errors to the fact that our lexical probabilities are context-independent. For example, we have:

$$\begin{aligned}P(S \mid \text{off}) &= 0.39 \\ P(S^* \mid \text{off}) &= 0.61\end{aligned}$$

When we look at Viterbi alignments from the analysis task, we see that when *off* is preceded by the word *far*, the probabilities reverse dramatically:

$$\begin{aligned}P(S \mid \text{off, after far}) &= 0.95 \\ P(S^* \mid \text{off, after far}) &= 0.05\end{aligned}$$

Similarly, the probability of stressing *at* is 40% in general, but this increases to 91% when the next word is *the*. Developing a model with context-dependent probabilities may be useful not only for improving generation and translation, but also for improving poetry analysis itself, as measured by analysis task accuracy.

Other potential improvements include the use of prior knowledge, for example, taking word length and spelling into account, and exploiting incomplete pronunciation dictionary information.

**Generation.** Evaluation is a big open problem for automatic poetry generation—even evaluating human poetry is difficult. Previous suggestions for automatic generation include acceptance for publication in some established venue, or passing the Turing test, i.e., confounding judges attempts to distinguish machine poetry from human poetry. The Turing test is currently difficult to pass with medium-sized Western poetry.

**Translation.** The advantage of translation over generation is that the source text provides a coherent sequence of propositions and images, allowing the machine to focus on “how to say” instead of “what to say.” However, translation output lattices offer limited material to work with, and as we dig deeper into those lattices, we encounter increasingly disfluent ways to string together renderings of the source substrings.

An appealing future direction is to combine translation and generation. Rather than translating the source text, a program may instead use the source text for inspiration. Such a hybrid translation/generation program would not be bound to translate every word, but rather it could more freely combine lexical material from its translation tables

with other grammatical and lexical resources. Interestingly, human translators sometimes work this way when they translate poetry—many excellent works have been produced by people with very little knowledge of the source language.

**Paraphrasing.** Recently,  $e \rightarrow f$  translation tables have been composed with  $f \rightarrow e$  tables, to make  $e \rightarrow e$  tables that can paraphrase English into English (Bannard and Callison-Burch, 2005). This makes it possible to consider statistical translation of English prose into English poetry.

## Acknowledgments

This work was partially supported by NSF grant IIS-0904684.

## References

- C. Bannard and C. Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proc. ACL*.
- P. Brown, V. Della Pietra, S. Della Pietra, and R. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2).
- B. Diaz-Agudo, P. Gervas, and P. A. Gonzalez-Calero. 2002. Poetry generation in COLIBRI. In *Proc. EC-CBR*.
- P. Gervas. 2001. An expert system for the composition of formal Spanish poetry. *Journal of Knowledge-Based Systems*, 14:200–1.
- P. Gervas. 2010. Engineering linguistic creativity: Bird flight and jet planes. Invited talk, CALC-10.
- J. Graehl. 1997. Carmel finite-state toolkit. <http://www.isi.edu/licensed-sw/carmel>.
- L. Jiang and M. Zhou. 2008. Generating Chinese couplets using a statistical MT approach. In *Proc. COLING*.
- P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: open source toolkit for statistical machine translation. In *Proc. ACL*.
- H. Manurung, G. Ritchie, and H. Thompson. 2000. Towards a computational model of poetry generation. In *Proc. AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*.
- H. Manurung. 2003. *An evolutionary algorithm approach to poetry generation*. Ph.D. thesis, University of Edinburgh.
- Y. Netzer, D. Gabay, Y. Goldberg, and M. Elhadad. 2009. Gaiku : Generating Haiku with word associations

- norms. In *Proc. NAACL Workshop on Computational Approaches to Linguistic Creativity*.
- N. Tosa, H. Obara, and M. Minoh. 2008. Hitch Haiku: An interactive supporting system for composing Haiku poem. In *Proc. International Conference on Entertainment Computing*.
- M. T. Wong and A. H. W. Chun. 2008. Automatic Haiku generation using VSM. In *Proc. ACACOS*.

# Automatic Prediction of Text Aesthetics and Interestingness

**Debasis Ganguly**  
CNGL,  
School of Computing,  
Dublin City University,  
Dublin 9, Ireland  
dganguly@computing.dcu.ie

**Johannes Leveling**  
CNGL,  
School of Computing,  
Dublin City University,  
Dublin 9, Ireland  
jleveling@computing.dcu.ie

**Gareth J.F. Jones**  
CNGL,  
School of Computing,  
Dublin City University,  
Dublin 9, Ireland  
gjones@computing.dcu.ie

## Abstract

This paper investigates the problem of automated text aesthetics prediction. The availability of user generated content and ratings, e.g. Flickr, has induced research in aesthetics prediction for non-text domains, particularly for photographic images. This problem, however, has yet not been explored for the text domain. Due to the very subjective nature of text aesthetics, it is difficult to compile human annotated data by methods such as crowd sourcing with a fair degree of inter-annotator agreement. The availability of the Kindle “popular highlights” data has motivated us to compile a dataset comprised of human annotated aesthetically pleasing and interesting text passages. We then undertake a supervised classification approach to predict text aesthetics by constructing real-valued feature vectors from each text passage. In particular, the features that we use for this classification task are word length, repetitions, polarity, part-of-speech, semantic distances; and topic generality and diversity. A traditional binary classification approach is not effective in this case because non-highlighted passages surrounding the highlighted ones do not necessarily represent the other extreme of unpleasant quality text. Due to the absence of real negative class samples, we employ the MC algorithm, in which training can be initiated with instances only from the positive class. On each successive iteration the algorithm selects new *strong negative* samples from the unlabeled class and retrains itself. The results show that the mapping convergence (MC) algorithm with a Gaussian and a linear kernel used for the mapping and convergence phases, respectively, yields the best results, achieving satisfactory accuracy, precision and recall values of about 74%, 42% and 54% respectively.

## 1 Introduction

Since their inception, Amazon Kindle device<sup>1</sup> and Apps for other general purpose hand-held devices, have led to a massive increase in the trend of reading e-books over paper printed ones. The Amazon Kindle and the Kindle Apps provide a very simple mechanism for highlighting a piece of text and sharing it on social media. The most popular highlighted pieces of text are shown in the Kindle device with an intention to help readers focus on passages that are pleasing or interesting to the greatest number of people. Every month, Kindle customers highlight millions of book passages that are meaningful to them<sup>2</sup>. The general trend among Kindle readers, while reading the classic English literary works, is to highlight text passages that are associated with a high aesthetic quality. An example highlighted passage is shown in Figure 1.

With the availability of such highlighted text, which may be considered as text passages which most readers find pleasing to read, an interesting research problem is to attempt automatic prediction of highlighted pieces of text. In other words, given a text passage, the objective is to

---

This work is licensed under Creative Commons Attribution 4.0 International Licence. Page numbers and proceedings footer are added by the organisers. Licence details: <http://creativecommons.org/licenses/by/4.0/>

<sup>1</sup><https://kindle.amazon.com/>

<sup>2</sup>[https://kindle.amazon.com/most\\_popular](https://kindle.amazon.com/most_popular)

*It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.*

Figure 1: Passage from *A tale of two cities* (Charles Dickens), highlighted by 6843 Kindle readers.

determine the likelihood of it being aesthetically pleasing and interesting. Such an automated approach of identifying aesthetically pleasing text passages may potentially be used to endorse a newly released book on e-commerce websites with an aim to increase its sales. Moreover, such an approach may also, in principle, be used as a tool by an author to determine how likely it is for readers to appreciate a newly written text passage.

The key challenge in solving this problem is to determine the characteristic attributes of a popular highlighted text passage. An intuitive assumption is that the popularity of a highlighted passage depends on its aesthetic quality. Generally speaking, passages inclined towards expressing an author’s view on a subject, which may often be philosophical in nature, with considerable application of atypical figures of speech, e.g. anaphora, alliteration, antithesis, metaphor, simile, personification etc., are more likely to be highlighted than a straight-forward story narrative passage. For example, the highlighted passage in Figure 1 is rich in anaphora (repetition of the same word or group of words in a paragraph, e.g. “times”, “age”, “epoch” etc.) and antithesis (juxtaposition of opposing or contrasting ideas, e.g. “best of times”, “worst of times”; “wisdom”, “foolishness” etc). An automated approach of aesthetic quality prediction thus has to take into account these different features of a text passage. The idea of using these features for text aesthetics prediction, in fact, forms a core part of our work.

It is particularly interesting to see that this problem of automatically predicting text aesthetics is largely different from the standard well researched problem of document text classification (Sebastiani, 2002). The reason is as follows. The problem of text categorization can effectively be solved by the application of discrete categorical features, such as character n-gram frequencies and word frequencies. In other words, the presence of characteristic words from a particular domain is a good indicator of the class of a document, e.g. the presence of the words “soccer”, “goal” etc. in a document is a good indicator that the document is of the sports genre, whereas the presence of words such as “money”, “bank” etc. would indicate that the genre is finance. Consequently, the generative framework of a multinomial Naive Bayes (NB) model with character n-gram and word n-grams based features works effectively for this class of problems (McCallum and Nigam, 1998).

In the case of aesthetic quality prediction, however, the mere presence of a particular word or character n-gram can hardly be a good indicator of the inherent literary quality of the text. The output classes of this classification problem, namely *aesthetic* or *not aesthetic*, do not comprise a small vocabulary of domain-specific representative terms such as in the case of the sports or finance domains. The vocabularies of the respective classes in this classification problem are largely unrestricted and mutually indistinguishable.

The rest of the paper is organized as follows. Section 2 presents related research. In Section 3, we present our proposed approach to solve the text aesthetics problem. Section 4 describes our experimental settings, following which Section 5 presents the results. In Section 6, we investigate the contribution from individual features and then the relative importance of the features when used in combination. Finally, Section 7 concludes the paper.

## 2 Related Work

A computational viewpoint of aesthetic quality, in general, takes into account the subjectivity of an observer and postulates that among several observations, the aesthetically most pleasing one



is the one with the shortest description, given the observer’s previous knowledge (Schmidhuber, 2010). An agent driven reinforcement based learning algorithm can then be used in principle to produce creative (novel and interesting) outputs (Schmidhuber, 2010). Our work in this paper is largely different from the general reinforcement learning paradigm, because we focus on the particular problem of text aesthetics viewing the problem as a supervised classification task. Moreover, the proposition of minimum description length as an attribute of aesthetic quality (Schmidhuber, 2010) is counter-intuitive for literary works.

There has been considerable research interest in automatically predicting visual aesthetic quality of images (Dhar et al., 2011) and layout of web pages (Reinecke et al., 2013). Most empirically successful approaches to image aesthetics prediction first transform an image into a feature vector of characteristic attributes that play a pivotal role in differentiating an *interesting* image from a *non-interesting* one. Generally speaking, some of these attributes which determine whether an image is aesthetically pleasing are the presence of salient objects (indicated by a low depth of field), compositional attributes (e.g. the rule of thirds), the effect of light in natural landscapes, etc. The next step is to apply a supervised learning algorithm, e.g. support vector machine (SVM), to learn a two-class prediction model. Useful features, extracted from images for this classification task include: i) colourfulness, contrast, symmetry, vanishing point and facial features (Jiang et al., 2010); ii) face poses, between-face distances, and the consistency of expressions on multiple faces (Li et al., 2010); iii) high level describable attributes, such as compositional attributes (e.g. rule of thirds image layout), content attributes related to the presence of people, animals, sky illumination attributes etc. (Dhar et al., 2011).

Our proposed method of text aesthetics prediction is similarly based on extracting characteristic features from the text passages. However, in the case of literature, it is worth mentioning that in contrast to image aesthetics it is more difficult to describe the subtle attributes which differentiate an aesthetically pleasing text from its counterpart.

Although the authors are not aware of any reported research on text aesthetics, there has been a considerable amount of research in the somewhat closely related problem of detecting metaphors in text. Automated approaches to metaphor detection involve both supervised and unsupervised approaches, some of which include: i) supervised classification on extracted verbal target feature vectors of sentences (Gedigian et al., 2006); ii) expectation maximization (EM) based unsupervised approach to non-literal word sense detection (Birke and Sarkar, 2006); iii) unsupervised approach using hierarchical graph factorization clustering (Shutova and Sun, 2013).

In general, it is intuitive to assume that metaphorical or figurative parts of text are aesthetically pleasing and interesting, which makes the problem of text aesthetics prediction somewhat similar to that of metaphor detection. Unfortunately, this assumption is not often true, and this is particularly the case for literary works due to the availability of a large number of figures of speech at an author’s disposal (metaphor just being one of them). For example, the sample Kindle highlighted passage shown in Section 1 has an obvious aesthetic appeal to a large number of readers, in spite of it being not metaphorical.

### 3 Our Approach to the Text Aesthetics Prediction Problem

In this section, we describe the details of our approach to text aesthetics prediction. We hypothesize that a NB classifier with word or character n-gram based features is not suitable for this particular problem due to the mutual overlap and lack of domain specific restriction in the vocabulary of the output classes (i.e. aesthetic and non-aesthetic). One thus needs to extract a set of characteristic features from the text passages which may be useful to solve the classification problem. We describe the features used in our approach in Section 3.1. In Section 3.2, we propose to use the mapping convergence (MC) algorithm for the text aesthetics problem, where the intention is to learn a classifier only from positive samples.

*The truth is rarely pure and never simple. Modern life would be very tedious if it were either, and modern literature a complete impossibility!*

Figure 2: Passage from *The Importance of Being Earnest* (Oscar Wilde).

### 3.1 Feature Vector Encoding of Text Passages

In this section, we introduce the various features used for the text aesthetics classification task. Each feature is a function which maps a passage of text  $P = \{w_1 \dots w_N\}$  comprising  $N$  words into a real number.

#### 3.1.1 Word-based Features

In Section 1, we illustrated that an anaphora is a rhetorical device used by authors to emphasize a text passage, which in turn indicates that such a passage is likely to attract the attention of readers and hence are likely to be highlighted by them. Moreover, the closer the repetitions are, the stronger is the emphasis.

On the basis of this reasoning, we employ an average positional difference weighted count of word repetitions in a passage. To be more precise, for each word in a passage we compute the number of times a word  $w_i$  is repeated, divide this count by the difference between the repeating position (say at position  $j$ ), and average the sum of counts for all repeating words over the passage length, as shown in Equation 1. In Equation 1,  $\mathbb{1}(w_i = w_j)$  is the indicator function which is 1 if and only if  $w_i = w_j$  and 0 otherwise.

The second word level feature which we use, is the average length of words in a passage. The reasoning behind using this feature is that authors tend to use relatively longer words (e.g. superlatives) to emphasize a passage. Equation 2 shows how this is computed.

$$W_1(P) = \frac{2}{N(N-1)} \sum_{i=1}^N \sum_{j=i+1}^N \frac{\mathbb{1}(w_i = w_j)}{j-i} \quad (1)$$

$$W_2(P) = \frac{1}{N} \sum_{i=1}^N \text{len}(w_i) \quad (2)$$

#### 3.1.2 Topic-based Features

An attribute which can be considered responsible for the aesthetic quality of a text passage is the diversity of topics it expresses. It is reasonable to assume that a text passage expressing a broad idea or opinion of an author, often philosophical in nature, is likely to be appealing to readers. Such general themed text passages typically cover a broad range of topics, as a result of which the constituent words of such text passages involve collocation of seemingly unrelated terms. For example, in the text passage shown in Figure 2, the word pairs (*truth*, *tedious*), and (*literature*, *impossibility*) would typically appear in different topic classes, where by a topic we mean a set of words with high co-occurrence likelihood estimated from a collection of documents by standard topic modelling techniques such as the Latent Dirichlet allocation (LDA) (Blei et al., 2003). To encode this diversity of topics as a real valued feature function, we use Equation 3.

$$T_1(P) = \frac{2}{N(N-1)} \sum_{i=1}^N \sum_{j=i+1}^N \frac{\mathbb{1}[z(w_i) \neq z(w_j)]}{(j-i)} \quad (3)$$

In Equation 3,  $z(w)$  denotes the topic class of the word  $w$  obtained with the help of LDA. A mismatch in the topic class is divided by the distance between the mismatches to assign more weight to the close mismatches. As an example, the mismatch between (*literature*, *impossibility*) bears more importance than the mismatch between (*modern*, *impossibility*).

The second topic-based feature which we use pertains to predicting the abstractness of the content of a passage. It has been reported that words highly representative of topics are generally

not metaphorical. We apply a similar reasoning to hypothesize that since an interesting piece of text is more likely to be philosophical or abstract in nature in comparison to a story narrative, the constituent words are less likely to be the representatives of their topic classes. Formally speaking in terms of LDA, these words are expected to have smaller values of  $\max_k \phi_k(w)$ . Recall that a topic representative word in LDA exhibits a skewed distribution with a peak for one topic class (with a high value of  $\max_k \phi_k(w)$ ), whereas a less representative word exhibits a more uniform distribution of  $\phi_k(w)$  values over the topic classes (thus a low value of  $\max_k \phi_k(w)$ ). We use Equation 4 to compute the average topic concreteness of a text passage.

$$T_2(P) = \frac{1}{N} \sum_{i=1}^N \max_k \phi_k(w_i) \quad (4)$$

### 3.1.3 Part of Speech Feature

We hypothesize that another attribute of an aesthetic passage is that it is likely to contain a rich usage of adjectives (mostly of superlative type for the sake of emphasis) and adverbs. We therefore employ the part of speech tag (POS) information of the constituent words of a text passage as one of our features. To be more specific, we use the average number of adjectives and adverbs of a text passage as the feature value. This is shown in Equation 5.

$$POS(P) = \frac{1}{N} \sum_{i=1}^N (\#adjectives + \#adverbs) \quad (5)$$

### 3.1.4 Sentiment Feature

We pointed out in Section 1 that authors often use the *antithesis* figure of speech to express contrasting concepts. Thus, another feature which we can use is the aggregated absolute difference values between the sentiment polarities of words in a text paragraph. This again is weighted by the difference in position between a positive sentiment word and its negative counterpart to assign more importance to closely occurring opposite sentiment concepts.

To obtain the sentiment values of the constituent words, we used the SentiWordNet<sup>3</sup>. To illustrate with an example, consider the closely occurring opposite sentiment word pairs (*best* (0.75), *worst* (-0.75)), (*wisdom* (0.375), *foolishness* (-0.375)) etc. of Figure 1 and the word pairs (*complete* (0.625), *impossibility* (-0.25)) of Figure 2, where the numbers in the parentheses show the positive or the negative sentiment value (a normalized number between 0 and 1). Equation 6 shows the real-valued function derived from the sentiment information of word pairs, where the function  $s(w)$  denotes the sentiment value associated with the word  $w$ .

$$SENT(P) = \frac{2}{N(N-1)} \sum_{i=1}^N \sum_{j=i+1}^N \frac{|s(w_i) - s(w_j)|}{(j-i)} \quad (6)$$

### 3.1.5 Inter-word Semantic Distance Feature

An alternative way to represent the topic diversity is to capture the likelihood of the event of occurrence of two words in close vicinity. The higher this likelihood is, the better is the semantic relation or coherence between the words. We make use of the DISCO<sup>4</sup> tool to compute the semantic relation between two words in a word pair. In DISCO, these semantic relations between the words are precomputed on the basis of co-occurrence likelihoods from a large corpus, e.g. the Wikipedia (Kolb, 2008). DISCO provides two similarity measurements (named the first order and the second order similarities) between two input words. While the first order similarity between two input words is computed based on their collocation sets, the second order similarity is computed based on their sets of distributionally similar words (Kolb, 2008). We denote the

<sup>3</sup><http://sentiwordnet.isti.cnr.it/>

<sup>4</sup>[http://www.linguatools.de/disco/disco\\_en.html](http://www.linguatools.de/disco/disco_en.html)

first order and the second order similarities between words  $w_i$  and  $w_j$  respectively as  $ds_1(w_i, w_j)$  and  $ds_2(w_i, w_j)$  respectively.

In relation to text aesthetics, we expect a small value of average first order and second order similarity values between word pairs in a highlighted piece of text in comparison to a non-highlighted one. Similar to our earlier features, we divide these similarity values by the positional difference between the words in order to put more emphasis on semantic diversity between closely occurring words. Equation 7 shows the two features extracted making use of these similarity values.

$$SD_k(P) = \frac{2}{N(N-1)} \sum_{i=1}^N \sum_{j=i+1}^N \frac{|ds_k(w_i) - ds_k(w_j)|}{(j-i)}, \quad k = \{1, 2\} \quad (7)$$

### 3.2 Learning from Positive Examples: The MC Algorithm

Binary classifiers, such as SVMs, work particularly well with a sufficient number of both positive and negative class instances for training. In the case of text aesthetics prediction problem, the passages highlighted by Kindle readers serve as the positive class samples. Although it might be intuitive to use the non-highlighted passages as instances of the negative type, there can be problems associated with this approach.

Firstly, the non-highlighted passages are not essentially instances of the negative class because the non-highlighted passages are not necessarily aesthetically unpleasing. Secondly, there is an element of cognitive bias associated with the highlighting process because a reader, who can already see popular highlights while reading a page, may be biased to highlight the same passage himself, and may not in fact highlight some other passage which he himself found interesting.

Note that this observation in fact makes our problem more challenging to solve in comparison to aesthetics prediction in other domains, such as images, where information such as Flickr<sup>5</sup> photo ratings can be used as strong positive or negative indicators of an image interestingness or aesthetic quality, leading to effective classification results using a standard binary classification approach (Dhar et al., 2011).

Due to the presence of incompletely labeled examples, we apply the *mapping convergence* (MC) algorithm (Yu et al., 2003) for this task. The objective of the MC algorithm is to predict the positive samples from a test data, given a mixture of positive and unlabeled samples. These unlabeled samples in the MC algorithm can be treated as instances of either the positive or the negative class in order to obtain maximum classification effectiveness.

The two stages of the MC algorithm are summarized as follows.

1. The *mapping* stage identifies from the unlabeled samples the strong negative ones, i.e. the points distinctly different from the positive samples.
2. The *convergence* stage is an iterative step to learn a binary classification model, e.g. SVM, using the positive and the strong negative samples. Each iterative step of convergence classifies the remaining unlabeled samples to collect more strong negative samples. The convergence step is repeated until no more strong negative samples are found.

The objective of the convergence step of the MC algorithm is to maximize margin to make progressively better approximation of the negative data. At the end of the iteration, the class boundary eventually converges to the boundary around the positive data set in the feature space (Yu et al., 2003).

In our approach to the text aesthetics prediction task, we implement the mapping stage of the MC algorithm with the help of standard one-class classifiers, namely the one class SVM (OSVM) (Schölkopf et al., 1999) and the support vector data descriptor (SVDD) (Tax and Duin, 2004). The OSVM separates all the data points in the feature space from the origin, with the help of a separating hyperplane with maximum distance from the origin. The OSVM is thus

<sup>5</sup><https://www.flickr.com/>

able to separate out regions in the input space with high probability densities (Schölkopf et al., 1999). SVDD, on the other hand, instead of a planar, takes a spherical approach to the one class problem. The algorithm obtains a spherical boundary in feature space around the data. The volume of this hypersphere is minimized to minimize the effect of incorporating outliers in the solution (Tax and Duin, 2004).

It is worth mentioning here that although the OSVM and the SVDD can be trained with positive samples only, these models are prone to over-fitting or under-fitting due to a small number of support vectors modeled from a small number of positive samples (Yu et al., 2003). In contrast, a binary SVM can model data more robustly due to the presence of the additional negative samples. Hence, OSVM and the SVDD are typically used as a weak classifier to obtain a set of initial strong negative samples in order to initiate the convergence step of the MC algorithm.

## 4 Experiment Settings

In this section, we describe the dataset and the tools used for our experiments.

### 4.1 Dataset Construction

The standard practice to evaluate the metaphor detection problem, which is somewhat similar to the text aesthetics prediction, is to make extensive use of manually annotated data typically obtained under controlled user-based studies, where the users or the participants are instructed to perform some given objectives, such as manually label metaphors in a collection of documents, e.g. (Hovy et al., 2013). The main difficulties with this approach are that: i) it takes a considerable amount of time to collect data; ii) the quality of the data depends largely on controlled experimental settings, e.g. the data quality may be susceptible to errors caused by targeted, malicious work efforts, since there is often a financial incentive to complete tasks quickly rather than effectively (Ipeirotis et al., 2010); and iii) it is very difficult to compare the effectiveness of two methods on two different datasets obtained under different controlled user study settings.

The availability of fairly large amounts of highlighted text on the Amazon website has ensured a reliable and fast way to construct the dataset for carrying out the text aesthetics experiments. The advantages are as follows. Firstly, it is not necessary to conduct crowd sourcing experiments for data collection. Secondly, since the data is not generated by controlled crowd sourcing, the quality of the data is more reliable because there is no financial incentive to complete tasks quickly. Thirdly, since the data is publicly available, it is possible to achieve a fair comparison between different problem solving approaches.

The Amazon “Popular Highlights”<sup>6</sup> web page presents a ranked list of the most highlighted passages, sorted in descending order by the number of highlights. However, at the time of writing this paper, Amazon has neither made the data publicly downloadable nor provided an API to access it. For conducting our experiments with this data, we therefore had to automatically crawl data from the Popular Highlights web page.

In addition to the highlighted passages (serving as the positive class samples in our dataset), we also need the non-highlighted ones (meant to serve as the unlabeled samples). The text from the non-highlighted passages, however, are not available in the Popular Highlights web page. This data was thus extracted from those books, the passages of which are popularly highlighted. In order to ensure free access to book content, we had to restrict our dataset to the 50 most popular highlighted classic English fictions.

More precisely speaking, for every highlighted passage found while crawling the Amazon Popular Highlights page, our crawler checks if the book is available on project Gutenberg<sup>7</sup>. If not, then we examine the next highlighted passage, otherwise we crawl the full text of the book,

---

<sup>6</sup>[https://kindle.amazon.com/most\\_popular/highlights\\_all\\_time/](https://kindle.amazon.com/most_popular/highlights_all_time/)

<sup>7</sup><http://www.gutenberg.org/>

in which the current highlighted passages belongs, from project Gutenberg website. The crawler continued to run until we had collected highlighted passages from 50 different literature classics.

The dataset for the prediction task is then constructed as follows. First, we add the text of all highlighted passages as instances of the positive class. Next, for each highlighted passage, we add the paragraph preceding and succeeding it into the dataset as the unlabeled samples. Note that selecting the unlabeled samples this way is better than random selection of non-highlighted passages from full text, because this way of choosing negative samples ensures a meaningful representation of reader judgments to highlight a particular passage of text from within a surrounding context.

We then partition the dataset comprised of the positive and unlabeled samples into equal sized training and test sets. In Table 1, we outline the characteristics of the dataset.

Dataset	# Books	Vocab. Size	# Passages		
			Highlighted	Unhighlighted	Total
Train	25	9560	168	305	473
Test	25	7883	169	319	488
Total	50	13496	337	624	961

Table 1: Dataset characteristics

## 4.2 Implementation Details

For each passage in the dataset, we extract the features described in Section 3.1. To compute the topic modeling based features we used Mallet<sup>8</sup>. The number of topics ( $K$ ) in LDA was set to 100. The POS tag feature was extracted with the help of the Stanford POS tagger<sup>9</sup>. For extracting the sentiment feature, we made use of the Java API of the SentiWordNet<sup>10</sup>. For the semantic word distance feature, we used the DISCO Java API<sup>11</sup>.

For the naive Bayes experiment, we used the Stanford classifier<sup>12</sup>. The SVM experiments (binary SVM, one-class SVM, SVDD) were conducted with the libSVM software<sup>13</sup>.

## 4.3 Evaluation Metrics

For all the experiments reported in this paper, the classification effectiveness mainly focuses on precision and recall with respect to the positive class. Consequently, precision, recall and the F-score measures, shown in Tables 2 and 3, are measured with respect to the positive class only.

Ideally, for this problem one would want to obtain a high recall, i.e. identify as many highlighted passages correctly as possible. In this situation, recall is thus more important than precision. Achieving a good precision is desirable, nonetheless, to minimize the false positives. Although we report accuracy, we emphasize that accuracy alone is not a good measure of classification effectiveness in this case, because correct identification of negative instances is not important for this problem.

## 5 Results

Before conducting experiments with the MC algorithm, we obtained baseline results by classifying the dataset using NB and SVMs. In the case of NB, instead of using the real valued features from the text passages (as proposed in Section 3.1), we simply used the character n-gram and word n-gram features (maximum value of  $n$  was set to 5) from the text, automatically extracted

<sup>8</sup><http://mallet.cs.umass.edu/>

<sup>9</sup><http://nlp.stanford.edu/software/tagger.shtml>

<sup>10</sup><http://sentiwordnet.isti.cnr.it/code/SentiWordNetDemoCode.java>

<sup>11</sup>[http://www.linguatools.de/disco/disco\\_en.html](http://www.linguatools.de/disco/disco_en.html)

<sup>12</sup><http://nlp.stanford.edu/software/classifier.shtml>

<sup>13</sup><http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/>

Classifier	Kernel	Accuracy	Precision	Recall	F-score
NB	N/A	67.40	54.40	36.70	43.80
BSVM	Linear	66.19	35.71	5.92	10.15
BSVM	Gaussian	67.00	39.39	15.38	22.13
OSVM	Linear	38.32	32.46	51.48	39.82
OSVM	Gaussian	53.68	41.87	50.29	45.70
SVDD	Linear	35.04	34.77	100.00	51.60
SVDD	Gaussian	37.91	35.56	97.63	52.13

Table 2: Text aesthetics prediction results with Naive Bayes and SVM.

Classifier		Kernel		Accuracy	Precision	Recall	F-score
Mapping	Convergence	Mapping	Convergence				
OSVM	BSVM	Linear	Linear	66.18	35.71	5.92	10.15
OSVM	BSVM	Linear	Gaussian	64.96	40.26	36.69	38.39
OSVM	BSVM	Gaussian	Linear	66.80	44.44	11.83	18.69
OSVM	BSVM	Gaussian	Gaussian	64.34	36.87	39.05	37.93
SVDD	BSVM	Linear	Linear	40.98	35.76	92.90	51.64
SVDD	BSVM	Linear	Gaussian	43.44	36.17	90.53	51.69
SVDD	BSVM	Gaussian	Linear	56.76	42.90	74.64	<b>54.42</b>
SVDD	BSVM	Gaussian	Gaussian	47.34	38.60	88.17	53.69

Table 3: Text aesthetics prediction results by the MC algorithm with different settings.

by the Stanford classifier. The result of this experiment (see Table 2) shows that the recall value is very low, which in turn indicates that word vocabulary based features, typically used for text categorization, are not effective for this task.

The next classification method that we employ is standard binary class SVM (denoted as BSVM). The training phase of the BSVM used the non-highlighted passages as negative class instances. We experimented with both linear and Gaussian kernels. For all reported results which use the Gaussian kernel, the parameter  $\gamma$  was set to the default value of  $1/(\#\text{features})$  as per the libSVM implementation. Although the accuracy achieved is comparable to NB, the recall achieved is worse, which shows that treating non-highlighted passages as negative class instances is not reasonable for this problem (see Section 6.2 for an illustration).

The recall value is significantly increased with the help of one-class SVM (OSVM). SVDD performs even better in terms of recall. However, SVDD significantly underfits the data because it classifies almost every test data point as an instance of the positive class, thus achieving low accuracy and precision due to the presence of too many false positives.

Our next set of experiments involves the MC algorithm for classification. Since, the mapping phase makes use of only the positive data, we employed both the one-class classifiers used in the experiments of Table 2, i.e. OSVM and SVDD, for this purpose. Mapping with OSVM results in an improvement in the accuracy at the cost of sacrificing recall, which is not desirable for this problem. However, note that the negative samples obtained with the OSVM mapping (with Gaussian kernel) improves the classification effectiveness of the BSVM (compare the fourth row of Table 3 with the second row of Table 2), which indicates that the MC algorithm does improve the classification effectiveness, confirming our hypothesis that it is reasonable not to consider every non-highlighted passage as negative samples.

The problem of SVDD underfitting (as evident from the SVDD results of Table 2) is alleviated by the MC approach. The most effective MC approach uses Gaussian/linear kernels for mapping/convergence (see the seventh row of Table 3). Accuracy is increased to around 56% with a satisfactory recall of around 74%. The use of Gaussian kernel during both the mapping and convergence steps yields a higher recall but at the cost of more false positives (lower accuracy, precision and F-score).

Feature combination vector				Evaluation Metrics			
Word	Topics	POS/Polarity	Semantic	Accuracy	Precision	Recall	F-score
1	0	0	0	36.06	34.88	97.63	51.40
0	1	0	0	37.91	35.74	99.40	52.58
0	0	1	0	36.05	35.01	98.81	51.70
0	0	0	1	42.41	37.03	94.67	53.24
1	1	1	1	56.76	42.90	74.64	<b>54.42</b>

Table 4: Individual feature contributions for identifying text aesthetics.

Feature	igain
Topic diversity ( $T_1$ )	0.3684
Sentiment ( $SENT$ )	0.2685
Word repetition ( $W_1$ )	0.2509
First-order semantic distance ( $SD_1$ )	0.1543
Part-of-speech ( $POS$ )	0.1448
Second-order semantic distance ( $SD_2$ )	0.1141
Word length ( $W_2$ )	0.0732
Topic abstractness ( $T_2$ )	0.0526

Table 5: Ranking features by their igain values.

## 6 Posthoc Analysis

In this section, we comment on the importance of the features used for classification, and also illustrate how the MC algorithm helps in increasing the separability between the classes.

### 6.1 Feature Importance

First, we investigate the importance of the different features by a selective choice of only one group of features at a time for the classification. The classifier we use for this experiment is MC with a Gaussian SVDD kernel for mapping and a linear SVM kernel for convergence (as per the best settings of Table 3). The results are shown in Table 4 from which it can be seen that the best accuracy is obtained with the use of the semantic distance features.

It can be observed that the accuracy values obtained with a single category of features, such as word-based (length and repetition), topic-based (generality and diversity) and so on, are considerably lower than the accuracy value obtained with a combination of all the features (the last row of Table 4). The precision values achieved with these individual feature groups are also considerably lower than the precision of 42.90% of the overall combination.

Next, we find out the relative importance of each feature in their overall combination by ranking the features with the help of a standard feature quality estimator, called *information gain* (*igain*) (Quinlan, 1986). The results are presented in Table 5. It can be seen that the topic diversity is the most discriminative feature having an igain value significantly higher than the second most important one in the list. This observation verifies our hypothesis that aesthetically appealing passages are those constituting terms from diverse topics.

The sentiment and the word repetition features, having close igain values, are second and third respectively in the list. The usefulness of the sentiment feature suggests that contrasting concepts packed in close vicinity of a sentence are likely to be aesthetically pleasing to read. The word repetition feature, on the other hand, suggests that the anaphora figure of speech is likely to be associated with aesthetically pleasing text.

### 6.2 Illustration of the usefulness of the MC Algorithm

This section investigates the usefulness of the MC algorithm for the text aesthetics classification. In particular, we show that for this one class classification problem, the MC algorithm can selectively refine the set of unlabeled samples and retrain the model for better separability



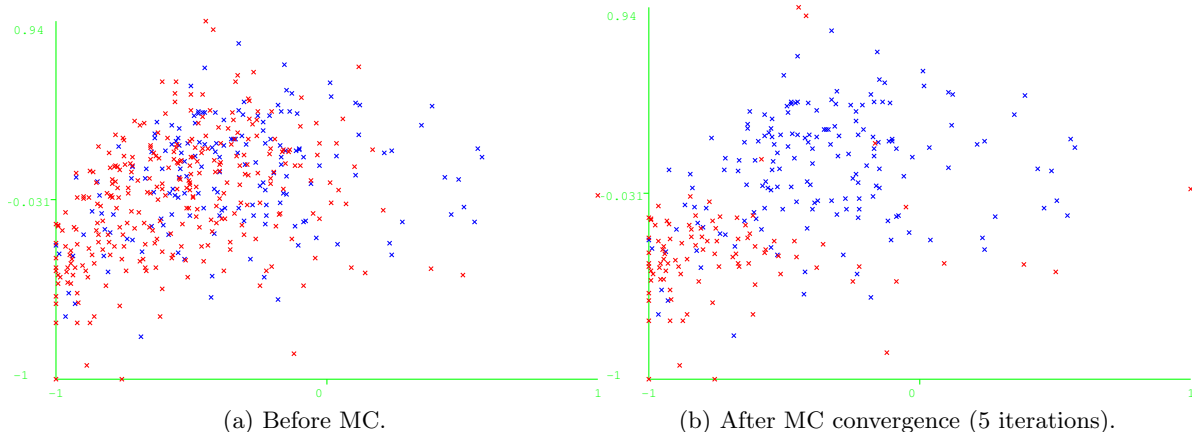


Figure 3: Visualization of the training set in the two most discriminating dimensions, i.e. topic diversity (Y-axis) and sentiment (X-axis).

between the positive and the unlabeled classes.

To illustrate our claim, we first plot the initial training set in two dimensional subspace before the application of MC, i.e. when all the unlabelled instances are treated as negative class samples; this is shown in Figure 3a. The two dimensions that we use for plotting this figure, are the two features having the highest igitain values, i.e. the topic diversity ( $T_1$ ) and sentiment ( $SENT$ ) features. Figure 3a shows that the highlighted text passages (shown in blue) are not well separated from the non-highlighted ones (shown in red).

Next, in Figure 3b, we plot the training set with a reduced number of samples from the negative (non-aesthetic) class obtained after running the MC algorithm. Figure 3b clearly shows that after convergence the MC algorithm has retained only the strong negative samples for training, as is evident from a better visual separation between the classes. A binary classifier, trained on the dataset of Figure 3b, is thus likely to be more effective than that trained with Figure 3a.

## 7 Conclusions

This paper investigated the problem of automated text aesthetics prediction. As distinguishing features for text aesthetics identification, we applied different statistical features such as word repetitions, topic diversity, part-of-speech, word polarity etc. We collected aesthetically pleasing text passages from the Kindle “popular highlights” website for conducting our experiments. Due to the presence of only positive class samples, i.e. the highlighted passages, in this dataset, we apply the MC algorithm to iteratively train a binary classifier with the strongly negative samples.

The results of our experiments show that the MC algorithm with a Gaussian and a linear kernel applied for the mapping and convergence phases respectively, yields the best results achieving satisfactory recall, precision and F-score values of about 74%, 42% and 54% respectively. Moreover, the results also demonstrate that the topic diversity, word polarity and word repetition are the three most distinguishing features for text aesthetics identification. Furthermore, our results are comparable to those of a somewhat similar problem of figurative text detection where the best reported F-score values achieved are about 54% (Birke and Sarkar, 2006) and 64% (Shutova and Sun, 2013).

## Acknowledgments

This research is supported by Science Foundation Ireland (SFI) as a part of the CNGL Centre for Global Intelligent Content at DCU (Grant No: 12/CE/I2267).

## References

- Julia Birke and Anoop Sarkar. 2006. A clustering approach for nearly unsupervised recognition of nonliteral language. In *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy*. The Association for Computer Linguistics.
- David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, March.
- Sagnik Dhar, Vicente Ordonez, and Tamara L. Berg. 2011. High level describable attributes for predicting aesthetics and interestingness. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, pages 1657–1664.
- Matt Gedigian, John Bryant, Srinu Narayanan, and Branimir Cicic. 2006. Catching metaphors. In *Proceedings of the Third Workshop on Scalable Natural Language Understanding, ScaNaLU '06*, pages 41–48, Stroudsburg, PA, USA. Association for Computational Linguistics.
- Dirk Hovy, Shashank Shrivastava, Sujay Jauhar, Mrinmaya Sachan, Kartik Goyal, Huying Li, Whitney Sanders, and Eduard Hovy. 2013. Identifying metaphorical expressions with tree kernels. In *Proceedings of NAACL-HLT Meta4NLP Workshop*.
- Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. 2010. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP '10*, pages 64–67, New York, NY, USA. ACM.
- Wei Jiang, Alexander C. Loui, and Cathleen Daniels Cerosaletti. 2010. Automatic aesthetic value assessment in photographic images. In *Proceedings of the 2010 IEEE International Conference on Multimedia and Expo, ICME 2010, 19-23 July 2010, Singapore*, pages 920–925.
- Peter Kolb. 2008. DISCO: A Multilingual Database of Distributionally Similar Words. In *KONVENS 2008 – Ergänzungsband: Textressourcen und lexikalisches Wissen*, pages 37–44.
- Congcong Li, Alexander C. Loui, and Tsuhan Chen. 2010. Towards aesthetics: A photo quality assessment and photo selection system. In *Proceedings of the International Conference on Multimedia, MM '10*, pages 827–830, New York, NY, USA. ACM.
- Andrew McCallum and Kamal Nigam. 1998. A comparison of event models for naive bayes text classification. In *AAAI/ICML Workshop on Learning for Text Categorization*, pages 41–48.
- J. R. Quinlan. 1986. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March.
- Katharina Reinecke, Tom Yeh, Luke Miratrix, Rahmatri Mardiko, Yuechen Zhao, Jenny Liu, and Krzysztof Z. Gajos. 2013. Predicting users’ first impressions of website aesthetics with a quantification of perceived visual complexity and colorfulness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '13*, pages 2049–2058, New York, NY, USA. ACM.
- Jürgen Schmidhuber. 2010. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE T. Autonomous Mental Development*, 2(3):230–247.
- Bernhard Schölkopf, Robert C. Williamson, Alex J. Smola, John Shawe-Taylor, and John C. Platt. 1999. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 582–588. The MIT Press.
- Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March.
- Ekaterina Shutova and Lin Sun. 2013. Unsupervised metaphor identification using hierarchical graph factorization clustering. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 978–988. The Association for Computational Linguistics.
- David M. J. Tax and Robert P. W. Duin. 2004. Support vector data description. *Mach. Learn.*, 54(1):45–66, January.
- Hwanjo Yu, ChengXiang Zhai, and Jiawei Han. 2003. Text classification from positive and unlabeled documents. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pages 232–239. ACM.

# Multilabel Subject-based Classification of Poetry

Andrés Lou, Diana Inkpen and Chris Tănăsescu (Margento)

University of Ottawa

School of Electrical Engineering and Computer Science  
800 King Edward, Ottawa, ON, Canada, K1N 6N5

## Abstract

Oftentimes, the question “what is this poem about?” has no trivial answer, regardless of length, style, author, or context in which the poem is found. We propose a simple system of multi-label classification of poems based on their subjects following the categories and subcategories as laid out by the Poetry Foundation. We make use of a model that combines the methodologies of tf-idf and Latent Dirichlet Allocation for feature extraction, and a Support Vector Machine model for the classification task. We determine how likely it is for our models to correctly classify each poem they read into one or more main categories and subcategories. Our contribution is, thus, a new method to automatically classify poetry given a set and various subsets of categories.

## Introduction

Poetry computational analysis is becoming more and more popular, though the field remains largely unexplored, as evidenced by the lack of a substantial body of work published (Kaplan and Blei 2007). Text classification methods, however efficient or at least effective when processing prose, often have to be modified and fine-tuned in a very different manner when dealing with poetry. While it may appear at first that any sort of in-depth analysis applied to poetry is a monumental task for a machine (because of the richness of meanings and information that can be contained in a single poem, a single verse, or sometimes even a single word), studies like those of Greene, Bodrumlu, and Knight (2010) and Kao and Jurafsky (2012) show that this is indeed possible, and that tasks such as machine translation and natural language generation can be carried out to a certain degree of effectiveness even when the data involved is poetry.

While poetry can be classified using many different evaluation metrics, such as subject, historical period, author, school, place of origin, etc, we focus entirely on a subject-based classification task, making exclusive use of the lexical content of each poem in our corpus to determine the categories to which it belongs.

## Related Work

While there exists a volume of work related to computational poetry, the field is still relatively unexplored. Kaplan and

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Blei (2007) showed that it is possible to classify a group of poems in terms of style and to visualize them as clusters. Kao and Jurafsky (2012) showed that both concrete features, such as rhyme and alliteration, and abstract features, like positive emotions and psychological well-being, can be used to determine whether a certain poem was written in the style of prestigious, award-winning poets or amateur poets. Features such as rhyme or rhythm can be extracted and abstracted from verses into syllable-stress patterns for further processing, as shown by (2010) and (2010). Jamal, Mohd, and Noah (2012) used a Support Vector Machine model to classify traditional Malay poetry called pantun, which is a form of art used to express ideas, emotions and feelings in the form of rhyming lines. The authors classified the poems by theme; they also trained a classifier to distinguish poetry from non-poetry. A total of 1500 pantun divided into 10 themes and 214 Malaysian folklore documents were used as the training and testing datasets. This work is similar to our work since the themes are similar to our categories, but we also have subcategories, and our models use additional features.

We note that many of the resources employed in the crafting of poetry can indeed be processed, or “understood”, by a machine, even if there are many gaps yet to be filled: Genzel, Uszkoreit, and Och (2010) point out that the task of preserving the form and meaning of a poem is an example of an area where machine translation might never replace a human translator, though they point out that there is work to be done in the field.

## Classifying poetry

In this work, we focus on how the vocabulary of a poem determines its subject. While seemingly intuitive, this notion is a much more difficult task to perform than what it seems at first glance. As an example, let us consider the following excerpt from *The Love Song of J. Alfred Prufrock*, by T. S. Eliot:

Let us go then, you and I,  
When the evening is spread out against the sky  
Like a patient etherized upon a table;  
Let us go, through certain half-deserted streets,  
The muttering retreats  
Of restless nights in one-night cheap hotels  
And sawdust restaurants with oyster-shells:

<b>Total</b>	<b>No. of Poems</b>	<b>Fraction</b>
	<b>7214</b>	<b>%</b>
Love	1250	17.3
Nature	2218	30.7
Social Commentaries	2258	31.3
Religion	848	11.8
Living	3103	43
Relationships	2524	35
Activities	1144	15.9
Arts & Sciences	1723	23.9
Mythology & Folklore	356	4.9

Table 1: The nine categories and the total number of poems in in our training set.

Streets that follow like a tedious argument  
Of insidious intent  
To lead you to an overwhelming question ...  
Oh, do not ask, "What is it?"  
Let us go and make our visit.

As is the case with many modern and contemporary poems, the subject of this celebrated high modernist piece is problematic, elusive, and multilayered. The question of what category this poem belongs to has a nebulous answer. The title, while indicative, cannot be used to readily classify it as a "Love" poem. Furthermore, the fact that it belongs to a certain category such as "Love" does not imply that it does not belong to a different category as well, such as "Living", nor does it imply whether it belongs to a subcategory thereof, specifically, the subcategory of "Marriage & Companionship" (indeed, as we will see, unequivocal single categorization is rare). Furthermore, is the speaker's insistent urge to travel and discover (new?) places actually a facetious one, as some of his diction strongly suggests, and then what is the target of his irony? Are possibly capital existential questions as the one in the penultimate line muffled by the modern condition of pointless rambling, indiscriminating consumerism, and chronic disorientation? And where is the announced love in the "tedious argument" of the alienating placeless cityscape? The task of determining whether a poem belongs to any given number of categories and subcategories, by means of analyzing its lexical content, is the objective of our work.

## Data

The Poetry Foundation's goal since its establishment in 2003 is "to discover and celebrate the best poetry and to place it before the largest possible audience."<sup>1</sup> The foundation is a large organization, and its website includes a corpus of several thousand poems categorized by subject, occasion, holiday, and several others.

The foundation is the successor to the Modern Poetry Association, founded in 1941 and the previous publisher of *Poetry* magazine. Today, the Poetry Foundation is one of the largest literary foundations in the world.

The corpus we used to train our classifying models was the Poetry Foundation's archive of poetry as of November

2014<sup>2</sup>. We developed a method of parsing and downloading the poem embedded on the HTML of every page in the poetry archives. Thus we produced a large corpus of unprocessed documents (more than 7,000 poems), each one of them annotated with its author, title, and its subjects.

**Tokenization** is the process of breaking down a string of characters into substrings comprised of individual words and punctuation signs, called tokens. A token is a sequence of characters that we treat as a string; the vocabulary of a text is the set of tokens that appear in it. We do not focus on all tokens, but instead on **word types**, which are "the form or spelling of [a] word independently of its specific occurrences in the text" (Bird, Klein, and Loper 2009).

As an example of a tokenization process, we consider the following verses of Edgar Allen Poe's *The Raven*:

Once upon a midnight dreary, while I pondered, weak and weary,  
Over many a quaint and curious volume of forgotten lore—

Splitting these into tokens, we obtain the following set of unique types: ":", "-", "I", "Once", "Over", "a", "and", "curious", "dreary", "forgotten", "lore", "many", "midnight", "of", "pondered", "quaint", "upon", "volume", "weak", "weary", and "while".

Each poem in our corpus was tokenized and mined for types, a task from which we built a word list containing all the types in the corpus and the probability associated to each type. To reduce the dimensionality of our vector, we removed stopwords, punctuation signs, capitalization, and types that did not appear in the whole corpus at least twice. Thus, we were left with a word list containing 29,537 unique types.

Table 1 shows the total number of poems in our training set and the break-down of each category. Since a given poem may belong to more than one main category, the percentages do not add up to 100%.

## Methodology

Our methodology involves three distinct phases: 1) Determining the number of categories and subcategories, and their nature, in which to place each poem; 2) Determine a method to extract relevant features from each document, and 3) Selecting an appropriate classifying algorithm.

## Main Categories and Subcategories

The nine main categories as laid out by the Poetry Foundation's archive are as follows: "Love", "Nature", "Social Commentaries", "Religion", "Living", "Relationships", "Activities", "Arts & Sciences", and "Mythology & Folklore".

The same archive divides each main category into several subcategories, each of which do not appear outside their parent category. Because of time constraints, we only examine the subcategories of three main categories:

*Love*: "Desire", "Heartache & Loss", "Realistic & Complicated", "Romantic Love", "Classic Love", "Infatuation &

<sup>1</sup><http://www.poetryfoundation.org/foundation/about>

<sup>2</sup><http://www.poetryfoundation.org/browse/>

Crushes”, “Unrequited Love”, “Break-ups & Vexed Love”, “First Love”.

*Living*: “Birth & Birthdays”, “Infancy”, “Youth”, “Coming of Age”, “Marriage & Companionship”, “Parent-hood”, “Separation & Divorce”, “Midlife”, “Growing Old”, “Health & Illness”, “Death”, “Sorrow & Grieving”, “Life Choices”, “The Body”, “The Mind”, “Time & Brevity”.

*Mythology & Folklore*: “Ghosts & the Supernatural”, “Horror”, “Heroes & Patriotism”, “Greek & Roman Mythology”, “Fairy-tales & Legends”.

## Feature Extraction

The content-based nature of the classification task makes it ideal to use two models to extract features from our corpus: Term Frequency-Inverse Document Frequency (tf-idf) as applied to a Bag-of-Words model, and Latent Dirichlet Allocation (LDA).

**Bag of Word features** Each word type is a feature for the classification and its value could be binary (1 if the word appears in the document and 0 if not) or based on frequency. We used **tf-idf**, because it was shown to work better in text classification tasks (Sebastiani 2002). tf-idf relates the frequency of a given word within a document to the frequency of the same word across all documents of a corpus, essentially determining how important the word is within the corpus. Several ways exist to calculate  $tf(t, d)$  of a given word; we used the simple approach of calculating the number of times the term  $t$  appears in a given poem  $d$ . idf is given by:

$$idf(t, \mathcal{D}) = \ln \frac{N}{|\{d \in \mathcal{D} \mid t \in d\}|}$$

where  $N$  is the total number of documents in a corpus,  $d$  is a document belonging to the corpus set  $\mathcal{D}$  and  $t$  is a term. Thus the set in the denominator represents all the documents in the corpus that contain the term  $t$  and the  $||$  operator denotes the cardinality of the set. tf-idf is then given by:

$$tf - idf(t, d, \mathcal{D}) = tf(t, d) \times idf(t, \mathcal{D})$$

**LDA features** Latent Dirichlet Allocation was first described by Blei, Ng, and Jordan (2003) as a “generative probabilistic model for collections of discrete data, such as text corpora” (Blei, Ng, and Jordan 2003). The idea of LDA is to represent each document in a corpus as a collection of topics, where each topic is characterized by a distribution over a set of words. LDA assumes the following generative process for each document  $d$  in a corpus  $\mathcal{D}$  (Blei, Ng, and Jordan 2003):

- Choose an  $N$  number of words in the form a Poisson distribution.
- Choose  $\theta \sim \text{Dir}(\alpha)$
- For each word  $w_n$  in  $N$ :
  - Choose a topic  $z_n \sim \text{Multinomial}(\theta)$
  - Choose a word  $w_n$  from  $p(w_n | z_n, \beta)$

With this and the diagram in Figure 1, we present the full probability equation as written by Savov (2009):

$$P(W, Z, \theta, \alpha, \beta) = \prod_{k=1}^T P(\varphi_k; \beta) \prod_{d=1}^{\mathcal{D}} P(\theta_d; \alpha) \prod_{w=1}^{N_d} P(Z_{d,w} | \theta_d) P(W_{d,w} | \varphi_{Z_{d,w}})$$

where  $P(Z_{j,w} | \theta_d)$  is the probability of picking a topic  $Z$  for a word  $w$  from a document  $d$ , given the topic proportion of  $d$  is  $\theta_d$ , and  $P(W_{d,w} | \varphi_{Z_{d,w}})$  is the probability of picking word  $W$  for the  $w$ -th word in document  $d$  assuming we were drawing it from the topic distribution for topic  $Z_{j,w}$ .

In practice, the challenge of using LDA lies in either empirically or experimentally estimating the parameters from which the model would produce our corpus. For our task, we used the Gensim Python module to implement an LDA model using Gibbs sampling for parameter estimation. For a detailed analysis on utilizing this method, see (Griffiths and Steyvers 2004). LDA has been shown to be an efficient way of performing text classification tasks and has become a popular tool in different areas of the subject. See (Li et al. 2011) and (Zhou, Li, and Liu 2009) for examples.

The documents can be represented as a collection of topics and each word in each document is associated with a distribution of these topics. The topics look like clusters of words with certain probabilities /weights that reflect the importance of each word for the topic. Each document is assigned a number of topics, each having a certain probability/weight. Thus, the topics will be used as features for our classifiers, and each document will be represented by the topics assigned to it by LDA, while the values of the features are the assigned probabilities/weights.

**Feature Selection** We filtered the resulting feature set with a  $\chi^2$  ranking algorithm. **Pearson’s  $\chi^2$  test** is a statistical test used to determine whether two events are independent of each other: the higher the  $\chi^2$  statistic, the more likely it is that the two events are dependent of each other.  $\chi^2$  is actually somewhat inaccurate when it comes to determining the level of independence between two events to one degree of independence, and it is prone to rank as dependent a number of features with little actual dependence; however, Manning, Raghavan, and Schtze (2008) showed that the noise produced by these is not important for a classification task as long as no statements about statistical dependence is made. Using this method, we kept the 1000 highest ranking word-types as features.

## Classifiers

To build our classifiers, we used a Support Vector Machine model, namely Weka’s SMO classifier with a polynomial kernel (Hall et al. 2009). A **Support Vector Machine (SVM)** is a model wherein input vectors are non-linearly mapped to a very high-dimension feature space, [w]here a linear decision surface is constructed. Special properties of the decision surface ensures high generalization ability of the learning machine (Cortes and Vapnik 1995). SVM has shown to be very efficient in text classification tasks, and has been a standard in the field for over a decade (Joachims 1998).

tf-idf	Accuracy	Precision	Recall	F-Measure	AUC
Love	0.888	0.883	0.888	0.873	0.711
Nature	0.831	0.83	0.831	0.823	0.764
Social Commentaries	0.809	0.806	0.809	0.798	0.738
Religion	0.908	0.896	0.908	0.895	0.678
Living	0.748	0.754	0.748	0.74	0.728
Relationships	0.769	0.775	0.769	0.749	0.697
Activities	0.875	0.864	0.875	0.85	0.642
Arts & Sciences	0.849	0.85	0.849	0.83	0.711
Mythology & Folklore	0.958	0.949	0.958	0.948	0.625
<b>Average</b>	<b>0.848</b>	<b>0.845</b>	<b>0.848</b>	<b>0.834</b>	<b>0.699</b>
Baseline	0.794	0.788	0.794	0.790	0.616

Table 2: Binary model output for each of the main categories using only bag-of-words. Baseline denotes the result obtained without feature selection. Note that using feature selection produces a considerably higher AUC value.

tf-idf + LDA <sub>k=100</sub>	Accuracy	Precision	Recall	F-Measure	AUC
Love	0.888	0.884	0.888	0.873	0.71
Nature	0.831	0.829	0.831	0.822	0.764
Social Commentaries	0.807	0.804	0.807	0.797	0.737
Religion	0.909	0.898	0.909	0.897	0.682
Living	0.745	0.750	0.745	0.738	0.726
Relationships	0.770	0.777	0.770	0.750	0.699
Activities	0.875	0.865	0.875	0.851	0.644
Arts & Sciences	0.849	0.850	0.849	0.830	0.711
Mthology & Folklore	0.957	0.949	0.957	0.948	0.622
<b>Average</b>	<b>0.848</b>	<b>0.845</b>	<b>0.848</b>	<b>0.834</b>	<b>0.699</b>
tf-idf + LDA <sub>k=500</sub>					
Love	0.887	0.884	0.887	0.872	0.709
Nature	0.832	0.83	0.832	0.823	0.765
Social Commentaries	0.806	0.802	0.806	0.795	0.734
Religion	0.91	0.899	0.91	0.897	0.678
Living	0.749	0.754	0.749	0.742	0.73
Relationships	0.770	0.776	0.770	0.751	0.700
Activities	0.874	0.865	0.874	0.849	0.638
Arts & Sciences	0.849	0.850	0.849	0.831	0.712
Mythology & Folklore	0.957	0.948	0.957	0.947	0.622
<b>Average</b>	<b>0.848</b>	<b>0.845</b>	<b>0.848</b>	<b>0.834</b>	<b>0.699</b>
tf-idf + LDA <sub>k=1000</sub>					
Love	0.889	0.885	0.889	0.874	0.712
Nature	0.833	0.832	0.833	0.825	0.766
Social Commentaries	0.805	0.801	0.805	0.794	0.733
Religion	0.909	0.898	0.909	0.896	0.68
Living	0.751	0.757	0.751	0.744	0.732
Relationships	0.77	0.776	0.77	0.751	0.7
Activities	0.873	0.863	0.873	0.848	0.638
Arts & Sciences	0.851	0.852	0.851	0.833	0.715
Mythology & Folklore	0.958	0.949	0.958	0.948	0.628
<b>Average</b>	<b>0.849</b>	<b>0.846</b>	<b>0.849</b>	<b>0.835</b>	<b>0.700</b>

Table 3: Binary model outputs for each of the main categories using tf-idf and LDA.

The experiments we ran consisted of two separate tasks: the classification of poems into one or more of the nine main categories, and the classification of poems inside one or more subcategories belonging to a main category.

The binary nature of a SVM classifier meant that each document, given a category or subcategory  $a$ , had to be classified as either “belonging to  $a$ ” or “not belonging to  $a$ ”. We therefore had to train several binary models, one for

each category and each subcategory analyzed. Each model is evaluated using the standard measures: accuracy, precision, recall (all for positive values for each classifier), and area under the ROC curve (AUC)<sup>3</sup>. For our evaluation, we performed a 10-fold cross-validation (the data is split into  $k = 10$  equal-sized subsets; 1 subset is used for validation

<sup>3</sup>The ROC curve plots the true positive rate against the false positive rate at various threshold settings.

<b>Living</b> tf-idf+LDA <sub>k=500</sub>	Accuracy	Precision	Recall	F-Measure	AUC
Birth & Birthdays	0.976	0.975	0.976	0.97	0.648
Infancy	0.982	0.802	0.979	0.977	0.587
Youth	0.906	0.905	0.906	0.896	0.757
Coming of Age	0.951	0.953	0.951	0.935	0.611
Marriage & Companionship	0.955	0.957	0.955	0.941	0.614
Parenthood	0.924	0.928	0.924	0.908	0.678
Separation & Divorce	0.984	0.984	0.984	0.979	0.591
Midlife	0.973	0.940	0.973	0.973	0.516
Growing Old	0.902	0.909	0.902	0.875	0.618
Health & Illness	0.939	0.937	0.939	0.925	0.658
Death	0.859	0.864	0.859	0.847	0.766
Sorrow & Grieving	0.901	0.909	0.901	0.875	0.632
Life Choices	0.952	0.939	0.952	0.937	0.560
The Body	0.939	0.923	0.939	0.912	0.514
The Mind	0.929	0.929	0.929	0.905	0.570
Time & Brevity	0.882	0.885	0.882	0.868	0.750
<b>Average</b>	<b>0.935</b>	<b>0.921</b>	<b>0.934</b>	<b>0.920</b>	<b>0.629</b>
<b>Mythology &amp; Folklore</b> tf-idf+LDA <sub>k=100</sub>	Accuracy	Precision	Recall	F-Measure	AUC
Ghosts & the Supernatural	0.905	0.916	0.905	0.897	0.818
Horror	0.952	0.907	0.952	0.929	0.500
Heroes & Patriotism	0.810	0.851	0.810	0.779	0.692
Greek & Roman Mythology	0.960	0.676	0.69	0.673	0.626
Fairy-tales & Legends	1.000	1.000	1.000	1.000	0.000
<b>Average</b>	<b>0.925</b>	<b>0.870</b>	<b>0.871</b>	<b>0.855</b>	<b>0.527</b>
<b>Love</b> tf-idf+LDA <sub>k=500</sub>	Accuracy	Precision	Recall	F-Measure	AUC
Desire	0.837	0.841	0.837	0.822	0.741
Heartache & Loss	0.892	0.901	0.892	0.861	0.616
Realistic & Complicated	0.816	0.822	0.816	0.798	0.720
Romantic Love	0.837	0.835	0.837	0.824	0.735
Classic Love	0.942	0.938	0.942	0.93	0.664
Infatuation & Crushes	0.884	0.882	0.884	0.873	0.751
Unrequited Love	0.915	0.913	0.915	0.893	0.615
Break-ups & Vexed Love	1.000	1.000	1.000	1.000	0.000
First Love	0.971	0.969	0.971	0.962	0.593
<b>Average</b>	<b>0.899</b>	<b>0.900</b>	<b>0.899</b>	<b>0.885</b>	<b>0.604</b>

Table 4: Binary model outputs for each of the subcategories of Living, Mythology & Folklore and “Love”.

while the remaining  $k - 1$  are used as training data; the process is repeated  $k$  times, each time a different subset being used for training). Our results are shown in Tables 2-4.

## Results and Discussion

The issue of data imbalance could not be sorted out without decreasing the size of our corpus; there is a disproportionately larger amount of instances under the “Living” category and a disproportionately smaller amount of instances under “Mythology & Folklore”. Overall, the results are acceptable, with all AUC measures well above 0.6 but none over 0.8. Further repetitions of the experiments and fine-tuning the parameters of the SVM classifier do not significantly improve the data. The subcategories show overall similar results, while presenting scarcity as an additional limiting factor.

### Main Categories

We performed an experimental run with the entirety of the word-types extracted from the corpus, without including the

LDA models in our training data. Results are shown in Table 2. The average AUC of this run is the lowest of all our experiments with the main categories.

After performing feature selection, we performed two sets of experimental runs: using only Bag-of-Words to extract features, and integrating both Bag-of-Words and LDA. Our purpose was to determine the impact the latter would have on our results, since the literature has shown the model to be popular in classification tasks. Our results, however, show that, while tf-idf alone delivers better results for some categories and tf-idf+LDA delivers better results for others, the average AUC is identical between the models, with all other statistics leaning, however slightly, towards tf-idf+LDA. The results of tf-idf are shown in Table 2.

The Gibbs sampling method of estimating LDA parameters leaves the task of selecting the number of LDA topics,  $k$ , to the experimenter. We made three experimental runs of tf-idf+LDA and  $k = 100, 500, 1000$ . Results are shown in Table 3. We also attempted to fully represent our corpus as an LDA distribution of topics by using nothing but the

$k = 500$  number of topics in our feature-space; they clearly show that stand-alone LDA topics are insufficient for any useful practical result<sup>4</sup>.

## Subcategories

The three main categories we selected to perform an experimental run were “Living”, “Mythology & Folklore”, and “Love”, the first being the largest category in terms of number of poems, the second being the smallest, and the third falling somewhere in between.

Table 4 present our results for the subcategories. The average AUC measurement for the subcategories is noticeably lower when compared to the main categories. This decrement reflects the relative scarcity of each subcategory, as there are much fewer instances with which to train each classifier. “Living” has the highest average AUC, which, again, reflects the relative scarcity of data for the subcategories of Love and Mythology & Folklore. The results suggest that a major increase in the available instances of each category or subcategory would further improve the performance of the classifier.

## Conclusion and Future Work

We have shown a simple method of determining whether a given poem belongs to an established category by listing its vocabulary in relation to the frequency of each term that belongs to it. While the idea of poems that do not approach a single given topic is not controversial, the categories themselves are not a universal convention. The very existence (or lack) of content-based categories of any sort might sometimes be a point of contention against subject-based classification tasks. SVM classifiers with feature selection achieved the best results for the main categories and subcategories.

Future work focusing on the content of poems for classifying purposes should refine models and account for both poetic diction and form. Style and the use of metaphors are both content-based concepts that should also be used in the task of classifying poetry. Advances in metaphor comprehension and development, as shown by Levy and Markovitch (2012), show that metaphors represented as mapping functions over a feature-space are a viable tool to make a machine “understand” a concept. Advances in rhyme and rhythm analysis (Genzel, Uszkoreit, and Och 2010) – which we shall complement with our own work on both meter and more euphonic techniques (alliteration, assonance, slant rhymes, etc.) as well as established poetic forms (sonnets, villanelles, terza rimas, etc.)– are steadily paving the road for automatic classification in such a deeply human field as poetry.

Never say, your lexicon exhausted,  
that for lack of content the lyre is silent  
There may not be a poet but  
There always shall be poetry  
— Gustavo Adolfo Bécquer

## References

- Bird, S.; Klein, E.; and Loper, E. 2009. *Natural Language Processing with Python*. O’Reilly.
- Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning*.
- Cortes, C., and Vapnik, V. 1995. Support-Vector Network. *Machine Learning*, 20, 273–297.
- Genzel, D.; Uszkoreit, J.; and Och, F. 2010. Poetic statistical machine translation: Rhyme and meter. In *Conference on Empirical Methods in Natural Language Processing*, 158–166.
- Greene, E.; Bodrumlu, T.; and Knight, K. 2010. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Conference on Empirical Methods in Natural Language Processing*, 524–533.
- Griffiths, T. L., and Steyvers, M. 2004. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*.
- Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA data mining software: An update. In *SIGKDD Explorations*, volume 11.
- Jamal, N.; Mohd, M.; and Noah, S. A. 2012. Poetry classification using Support Vector Machines. *Journal of Computer Science* 8, Issue 9:1441–1446.
- Joachims, T. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of ECML-98*, volume 1398 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 137–142.
- Kao, J., and Jurafsky, D. 2012. A computational analysis of style, affect, and imagery in contemporary poetry. In *Workshop on Computational Linguistics for Literature*, 8–17.
- Kaplan, D. M., and Blei, D. M. 2007. A computational approach to style in american poetry. In *Seventh IEEE International Conference on Data Mining*, 553–558.
- Levy, O., and Markovitch, S. 2012. Teaching machines to learn by metaphors. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.
- Li, K.; Xie, J.; Sun, X.; Ma, Y.; and Bai, H. 2011. Multi-class text categorization based on LDA and SVM. *Procedia Engineering* 1963–1967.
- Manning, C. D.; Raghavan, P.; and Schtze, H. 2008. *Assessing  $\chi^2$  as a feature selection method*. Cambridge University Press.
- Savov, I. 2009. Latent Dirichlet Allocation for scientific topic extraction. Unpublished.
- Sebastiani, F. 2002. Machine learning in automated text categorization. *ACM Computing Surveys* 34(1):1–47.
- Zhou, S.; Li, K.; and Liu, Y. 2009. Text categorization based on topic model. *Atlantis Press* 398–409.

<sup>4</sup>An average AUC of 0.508



# Metaphor Detection by Deep Learning and the Place of Poetic Metaphor in Digital Humanities

Chris Tanasescu,\* Vaibhav Kesarwani, Diana Inkpen

School of Electrical Engineering and Computer Science

University of Ottawa

Ottawa, Ontario, Canada

margento.official@gmail.com, vkesa079@uottawa.ca, diana.inkpen@uottawa.ca

## Abstract

The paper presents the work that has been done as part of the *Graph Poem* project in developing metaphor classifiers, now by deep learning methods (after previously having developed rule-based and machine learning algorithms), and a web-based metaphor detection tool. After reviewing the existing work on metaphor in natural language processing (NLP), digital humanities (DH), and artificial intelligence (AI), we present our own research and argue in favor of adopting data-intensive approaches, developing NLP classifiers, and applying graph theory (and particularly networks of networks) in computational literary or poetry analysis, while also highlighting the relevance of such work to DH, NLP, and AI in general.

keywords: poetry; metaphor; natural language processing; deep learning; digital humanities; graph theory applications; networks of networks

## Introduction

The purpose of this paper is to present the results the authors have obtained in applying deep learning methods in poetic metaphor detection as a continuation and expansion of our previous work on automatic metaphor classification in poetry. As both our present and previous work on metaphor are part of a larger project—The *Graph Poem*—involving a holistic comprehensive computational approach to poetry, we will dedicate this section to briefly introducing the overall project and discussing its place in Digital Humanities (DH). Then in the second section we will zoom in on our previous work on metaphor detection in the comparative context of the relevant metaphor computational analysis literature and DH metaphor-relevant projects and approaches. The subsequent sections will focus on the current research and results, and the last part will present our conclusions and work to be done in the future.

The concept behind the *Graph Poem* project was initially a creative writing and generative one, writing, assembling, (post)digitally or X-algorithmically generating and/or expanding poetry corpora based on commonalities between poems. Poems are nodes in a network graph while edges

represent commonalities initially drawn manually (MARGENTO 2012) and for which later on computational tools have been developed. The creative and generative purposes have thus naturally switched to critical and analytical ones, while the project joined a more general trend in DH regarding—in what was called the second and third wave in DH (Berry 2011)—a transition from classification or retrieval to more creative approaches and tools, or moreover, a fusion of the two. The latter was foregrounded in a significant way by N. Katherine Hayles and Jessica Pressman (2013) who underscored the complementarity if not congruence between these two sides of the coin in advancing a pun-like desideratum: making; critique. And this is perhaps particularly relevant to poetry and literature in general as it came against the background of a proposed new subject: comparative textual media.

In the specific case of our project, textual-medium-related concerns are occasioned by the more general critical approach to data. Our tenet is that generally speaking for digital tools to be relevant—in poetry computational analysis and not only—they need to be trained and applicable to substantial amount of data. And while it is naturally debatable what magnitude would that specifically entail in the case of poetry, and since big data is at the same time a concept that has features and implications that are hardly applicable to literature, there is indeed possible to adapt another subject-relevant concept to poetry: data intensive research (Critchlow and Kleese van Dam 2013). Such an approach entails working with as large amounts of data as possible (and we will refine this statement in a bit) while also applying methods and developing tools that are specifically sensitive to data in quantitative and qualitative fashions. We therefore looked for large available databases and archives, and settled on the Poetry Foundation browser that contains tens of thousands of poems and has been manually annotated for various poetic features ranging from topic to form to period and region. The quantitative component is hence obvious, and that is part of our contribution since other work in the field has involved significantly smaller datasets, at the scale of sometimes even one hundred poems (Kao and Jurafsky 2012) (Dalvean 2013), with otherwise significant results in poetry automated analysis).

While in the existing work it is true that the magnitude of the explored data is perhaps relative and not necessarily

---

\*Margento

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

consequential in terms of the quality of the results, this aspect becomes of the essence in developing deep learning approaches, as is the case of the work presented in this paper. As will see below, deep learning proves to provide better results than other methods, but in order for this method to work, the amount of data has to be increased beyond certain limits.

The issue of data and data intensive research is also of great importance for the *Graph Poem* project as a whole. The focus is on poetry corpora and the features of the assembled network graphs—such as connectivity, the existence of cliques and of cut vertices, percolation, etc.—tell us significant things about a specific corpus or corpora (be it a random selection, a volume, a whole oeuvre or several oeuvres of several poets) and are obtainable depending on the quantity, quality, and computational analysis of data, since the output of our poetic-feature-focused tools (the processing of form, topic, style, etc.) is the input data for the graph visualization and analysis component. Graph theory has never been to our knowledge deployed in poetry, but network graphs have been used in other literature-focused chapters of DH and computational linguistics, specifically in social network analysis as applied in modeling fiction characters (Agarwal, Kotalwar, and Rambow 2013) (also see (Vala et al. 2016) for a comprehensive literature review) but also in structure-based clustering of novels (Ardanuy and Sporleder 2014) and in generating narrative (Sack 2012). Besides the inevitable genre-related differences between these applications and the ones pertaining to the *Graph Poem* project, there is another significant distinction: all of the graphs in fiction analysis are plain (single-layer) networks whereas the ones assembled and analyzed in the latter are multiplex networks (or multigraphs). Multiplex networks are graphs in which vertices are connected by different types (layers) of edges, therefore a particular case of networks of networks that behave in certain critical respects differently from plain networks (D’Agostino and Scala 2014) and whose importance for the study of literature has been explored recently, for instance, in an article (Tanasescu and Tanasescu 2018) deploying multiplex networks in translation studies. The vertices in our network graphs are connected, therefore, in various layers, and analyzing those graphs involves processing and classifying the features in these layers independently and then analyzing the whole as a multiplex network with its specific characteristics. One of these layers is metaphor.

## Context and Previous Work

In our previous work (Kesarwani et al. 2017) we have developed a classifier combining rule-based and machine learning methods, and in this paper we are presenting the deep learning tool deploying convoluted neural networks we have developed for the same purpose but obtaining better results.

Metaphor has been touched on in a number of recent DH publications that are, if not developing tools, posing theoretical questions regarding, for instance, metaphorical thinking in media studies analysis (Graham and Brook 2016), the place of metaphor in digital hermeneutics and as embedded in an interface (Armaselu and van den Heuvel 2017) or the role of metaphors in shaping public opinion (Núñez et

al. 2017). But perhaps the most elaborate metaphor-focused DH project to this day is the *Metaphor Map of English* at the University of Glasgow (Hamilton, Bramwell, and Hough 2016) that visualizes the use of metaphor in the Historical Thesaurus with links between various categories, such as People or Travel and Travelling, all taken from the Thesaurus and falling under three general classes, external, mental, and social world. While this project involves an impressive database and interface, it does not work as an automatic classifier, and it is limited to visualizing and extracting data from the Thesaurus, with no option to run the tool on any other data. A poetry search for instance will produce a table of examples of metaphors from the Historical Thesaurus alongside their category, (the source texts) year of occurrence, and strength (strong or weak).

The only other DH project that has aimed to process poetic metaphor computationally is POEMAGE. The project has actually started as a visualization system for exploring the sonic topology of a poem (McCurdy et al. 2016) in the same period when the *Graph Poem* team began working on a rhyme classifier (Tanasescu, Paget, and Inkpen 2016), but has more recently turned to considering metaphor processing as well. Unlike the *Graph Poem*, though, POEMAGE does not involve “creating a tool that will algorithmically identify and visualize metaphor” (Coles 2017), as the authors themselves admit, but “getting at metaphor” in an “oblique way” by pointing the human reader to places in the text the machine is uncertain of in terms of pronunciation and meaning and thus signaling potential metaphor occurrences.

In our previous work on metaphor (Kesarwani et al. 2017) we researched what has been done in this respect in machine learning and NLP, and since we were the first to approach metaphor in poetry (and literature in general), we turned to what had been accomplished in processing metaphor in non-literary texts and drew on what was usable in poetry as well. Turney’s (Turney et al. 2011) notion of tracking down abstract-concrete discrepancies as indication of possible occurrences of metaphors became a feature in our model. We combined that with concrete category overlap (Assaf et al. 2013), and ConceptNet (Speer and Havasi 2012), and one of our alternative computations of metaphor expanded on the part of speech tagging used by Neuman (2013) who in his turn borrowed the method from Krishnakumaran and Zhu (2007). The expansion involved updating one of their syntactic sequences and adding two more (noun-verb and verb-verb on top of their adjective-noun and noun-verb-noun with two subcategories, copulative and regular verb) and deploying word embeddings. So far, in (Kesarwani et al. 2017), we have developed a classifier for the first type, noun-verb-noun, which we have refined by allowing the nouns to have determinants, while our future work will involve both developing classifiers for the other types and a generic classifier non-dependent on any given syntactic patterns. With regards to word embeddings, which are a novel way of representing words as vectors aimed at redefining the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus (Kesarwani et al. 2017), they also played a key role in our approach. In

fact, as we will explain in a bit, we trained our own word embeddings.

The above-mentioned syntactic structural sequences were developed into a rule-based method, and then, after researching another major contribution to metaphor natural language processing (Shutova, Kiela, and Maillard 2016) we also developed a statistical model and compared the results. It turned out, quite predictably, that the latter works significantly better. We also had to develop our own corpus and do manual annotation for training and testing our model: 680 sentences out of 1500 extracted only for type 1 metaphor [noun-copulative verb-(determiner)-noun] with two annotators and an arbiter deciding on disagreements.

Yet this was not our only training data. We combined our dataset with two other sets, with the notations TroFi (Birke and Sarkar 2006) and Shutova (Mohammad, Shutova, and Turney 2016), while the abbreviation we have picked for ours is PoFo. The results for PoFo + TroFi + Shutova were significantly better than those on PoFo alone, which proved an interesting twofold point, namely that the above-mentioned data intensive approach improves outcomes indeed, and that in detecting metaphor in poetry, non-poetry data are as helpful as poetry data.

We also reached in certain other respects a converse conclusion, that is, that tools trained on poetry data are sometimes better for non-poetry tasks than the ones trained on non-poetry data. We trained our own word embeddings on the PoFo poems and named the resulting model the GraphPoem model, which we did not use for metaphor detection, as we wanted the embeddings to be rooted in corpora that contained as few metaphors as possible. The options in terms of available generally used embeddings were GloVe (Pennington, Socher, and Manning 2014) and word2vec (Mikolov et al. 2013) vectors, yet we chose the former as they had been shown to work better for many lexical-semantic tasks (Pennington, Socher, and Manning 2014). Still, these word embeddings of ours turned out to be better than the GloVe ones, and we deployed them in our poetic diction processing tasks. We are currently working on comprehensive technical evaluations, but the superiority of our own embeddings is already intuitively obvious for all examples we have run our tool on as compared to GloVe.

The results in Table 1 and Table 2 show the words related to the word *love* in the GloVe and GraphPoem models in decreasing order of similarity score. It can be seen that the words in the GloVe model are more conversational (pronouns like *me*, *my*, *you*, *I* and *she* reinforce this) and less thematic, whereas the words from the GraphPoem model are more consistently—semantically and logically—related to the query *love*. Even an antonym like *hate* popping up in the GraphPoem model fits better in the list (on the 12th position nevertheless) than extraneous results such as *mind* in the GloVe one, given the general (psychological and not only) potential relevance of the love-hate complex and ambivalence, which is hardly to be found in the other case.

## Deep Learning Classification

In what follows we will describe our metaphor deep learning classifier and our web-based application for metaphor

Word	Score
me	0.738
passion	0.735
my	0.733
life	0.729
dream	0.727
you	0.718
always	0.711
wonder	0.709
i	0.708
dreams	0.707
mind	0.706
friends	0.704
true	0.703
loves	0.700
feel	0.698
happy	0.698
fun	0.697
kind	0.696
soul	0.695
she	0.695

Table 1: LOVE in the GloVe model

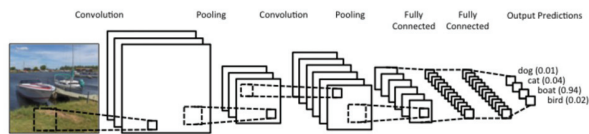


Figure 1: Layers in a convolutional neural network, from (Kim 2014).

detection.

The first requirement for deep learning classification is lots of examples / data points (by the thousands). Earlier, we tested our PoFo dataset (containing 680 data points with 340 for training and 340 for test) and the F-score for deep learning classifier was worse than machine learning ones. Therefore, when we were able to collect more data (around 4870 instances) with metaphor (poetry and non-poetry) by adding data from (Mohammad, Shutova, and Turney 2016) and (Birke and Sarkar 2006), we experimented with Convolutional Neural Networks (CNN) (shown in Figure 1) (Kim 2014) to examine whether we can get any gains in F-score when compared to the standard machine learning classifiers. Figure 2 shows the details of the CNN text classifier schema.

We used the Keras (Chollet and others 2015) deep learning framework with a Tensorflow (Abadi et al. 2015) backend and used a local GPU to accelerate the training process. The parameters that we tested on are given in Table 3.

The results of our experiments are given in Table 4. The best result, i.e., F-score 0.833 for metaphor and F-score 0.744 for the non-metaphor class was seen with epochs 300, batch 70, neurons 206 and inputs 103. Though we tested on hundreds of combinations of hyper-parameters, only the top

Word	Score
joy	0.791
sorrow	0.783
hope	0.781
desire	0.764
grief	0.759
despair	0.742
delight	0.737
pleasure	0.730
beauty	0.730
pain	0.729
bliss	0.729
hate	0.716
pity	0.714
true	0.709
comfort	0.706
shame	0.702
passion	0.701
faith	0.697
fear	0.697
hunger	0.695

Table 2: LOVE in the GraphPoem model

Parameter	Range
Inputs	103 - 106
Input activation function	ReLU, TANH
Hidden layers	1 - 4
Neurons in 1st layer	6 - 306
Output activation function	Softmax, Sigmoid
Dropout	0 - 0.9
Outputs	2
Epochs	20 - 1000
Loss function	Cat./Binary Cross Entropy
Optimizer	ADAM
Batch size	20 - 200

Table 3: Range of parameters tested.

results are being reported here for brevity.

It can be observed that with the same training set and test set (the test set being the 340 poetry examples), CNN performed significantly better than Support Vector Machines (SVM) and other machine learning algorithms. A baseline classifier that always outputs the most frequent class is also included for comparison. The best F-score for metaphor class was 0.781, seen with SVM with Pearson Universal Kernel (Puk). For CNN, we get a gain of 5.2% and we get a high F-score of 0.833. For the non-metaphor class, KNN obtained a F-score of 0.711. For CNN, we get a gain of 3.3% and we get a higher F-score of 0.744. For both classes, the performance is better with CNN.

The major drawback for the CNN classification (when compared to the other machine learning algorithms) is that a lot of data points are needed for training. Consequently, though we got better results for PoFo+TroFi+Shutova

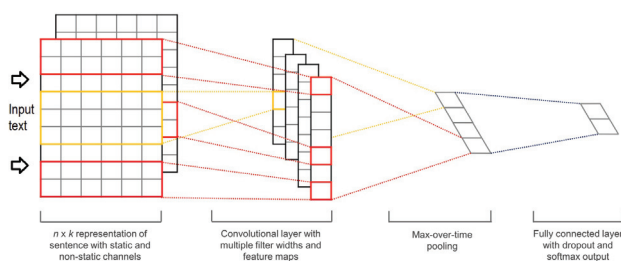


Figure 2: Schema diagram of the CNN text classifier.

dataset collectively, results on the rest of the datasets individually were worse than the SVM results for both the classes. We empirically observed that, in our case, anything less than 3,000 instances was insufficient for training the CNN and the results are much worse (close to the baseline of 50%). If we are able to overcome this soft threshold of 3,000 data points, deep learning classification works appreciably better.

### Web Application for Generic Metaphor Detection

We used our best performing machine learning model (SVM with F-score 0.781) and developed a web application for Multi-line metaphor detection. This web application works for not just poetry, but also for any natural language text and will be available for public use shortly once we complete the hosting process. For this application, we have not used POS tag sequence for Type 1 metaphor, instead we use Stanford NLP parser dependencies given below to extract the potential word pairs:

- nsubj
- dobj
- nsubjpass
- acl:relcl

Moreover, the pre-trained model (SVM with F-score 0.781) used for prediction in these two applications is serialized to decrease the execution time. It normally takes 10 - 12 seconds for execution. If serialization is not used, execution time can be as high as 40 seconds.

The application accepts multi-line text and outputs line-by-line result for the analysis. There can be multiple word-pairs for each line that are analysed for metaphoric intent. Figure 3 shows a screenshot of the web application. The poem (excerpt) (Lorde 2000) entered in the application is given below:

Poem Title : Afterimages (excerpt)  
 Author : Audre Lorde

A woman measures her life's damage  
 my eyes are caves, chunks of etched rock  
 tied to the ghost of a black boy  
 whistling  
 crying and frightened

Parameters	<i>metaphor</i>			<i>literal</i>		
	Precision	Recall	F-score	Precision	Recall	F-score
Baseline	0.565	1.000	0.722	0.000	0.000	0.000
SVM (Puk)	0.759	0.804	0.781	0.724	0.670	0.696
e:200 b:5 n:202 i:102	0.812	0.795	0.804	0.698	0.720	0.709
e:200 b:50 n:202 i:102	0.810	0.826	0.818	0.727	0.704	0.715
e:100 b:150 n:202 i:102	0.805	0.833	0.819	0.732	0.694	0.712
e:100 b:70 n:202 i:102	0.811	0.850	0.830	0.754	0.699	0.725
e:100 b:70 n:206 i:103	0.823	0.840	0.831	0.748	0.725	0.736
e:250 b:70 n:206 i:103	0.837	0.823	0.830	0.737	0.756	0.746
e:300 b:70 n:206 i:103	0.831	0.836	<b>0.833</b>	0.748	0.740	<b>0.744</b>

Table 4: Top results for SVM & CNN classification. Puk denotes Pearson Universal Kernel. The CNN was tested on various hyper-parameters, where e denotes epochs, b denotes batch size, n denotes the number of neurons in 1st layer and i denotes the number of inputs. All other hyper-parameters remained constant, input activation : RELU and output activation : SOFTMAX.

her tow-headed children cluster  
 like little mirrors of despair  
 their father’s hands upon them  
 and soundlessly  
 a woman begins to weep.

The complete result from the application is given below:

Line : A woman measures her life’s damage  
 Processing measures : woman  
 Prediction : metaphor  
 Processing measures : damage  
 Prediction : metaphor

Line : my eyes are caves, chunks of etched rock  
 Processing caves : eyes  
 Prediction : metaphor

Line : her tow-headed children cluster  
 Processing cluster : children  
 Prediction : metaphor

Line : like little mirrors of despair  
 Processing like : mirrors  
 Prediction : metaphor

Line : their father’s hands upon them  
 Processing hands : father  
 Prediction : literal

Line : a woman begins to weep.  
 Processing begins : woman  
 Prediction : literal

## Conclusions and Future Work

While work has been done in artificial intelligence and in digital humanities (DH) on processing poetry computationally, see for instance the repository of digital-pedagogy-relevant poetry projects put together by Chuck Rybak (Rybak 2016) this is the first initiative in assembling a comprehensive holistic conglomerate of poetry algorithms and tools. These tools are meant for both poetry analysis and creative writing/generative purposes and tackle the multifaceted

features of the genre consistently and coordinately, from topic to form to diction and style. For the latter, so far we have developed metaphor classifiers that deploy rule-based, statistical (machine learning), and deep learning methods, and we have launched a web-based metaphor natural language processing (NLP) application. While metaphor has been tackled in NLP before, the focus of that research has never been poetic metaphor (or literary tropes in general), and nor have any DH projects set as (part of) their goal(s) developing poetic metaphor processing tools. In our own work in metaphor deep learning classification we have established that deep learning, and particularly convoluted neural networks (CNN) output better results than the other previously deployed methods, given that the amount of data fed to the CNN is big enough, which also gave us the opportunity to verify the validity of the data intensive approach we have adopted generally in working on the overarching *Graph Poem* project. Our future work will include developing a metaphor classifier not depending on any syntactical pattern, and integrating the resulting web-based application into the overall network graph analysis and visualization tool.

## References

- Abadi, M.; Agarwal, A.; Barham, P.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Agarwal, A.; Kotalwar, A.; and Rambow, O. 2013. Automatic extraction of social networks from literary text: A case study on alice in wonderland. In *IJCNLP*, 1202–1208.
- Ardanuy, M. C., and Sporleder, C. 2014. Structure-based clustering of novels. In *CLfL@ EACL*, 31–39.
- Armaselu, F., and van den Heuvel, C. 2017. Metaphors in digital hermeneutics: Zooming through literary, didactic and historical representations of imaginary and existing cities. *Digital Humanities Quarterly* 11(3).
- Assaf, D.; Neuman, Y.; Cohen, Y.; Argamon, S.; Howard, N.; Last, M.; Frieder, O.; and Koppel, M. 2013. Why “dark thoughts” aren’t really dark: A novel algorithm for metaphor identification. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2013 IEEE Symposium on*, 60–65. IEEE.

## Metaphor Detection in Natural Language

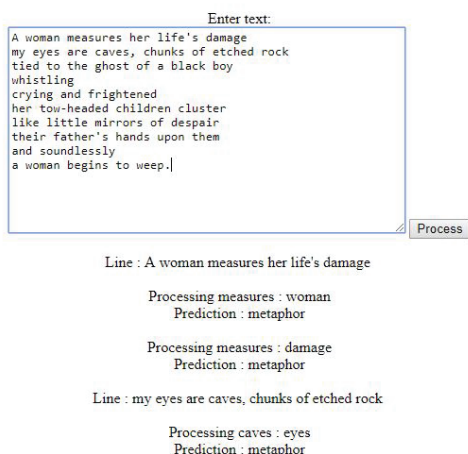


Figure 3: Screenshot of the multi-line metaphor detection web application. The full text is not captured in this screenshot.

Berry, D. M. 2011. The computational turn: Thinking about the digital humanities. *Culture Machine* 12.

Birke, J., and Sarkar, A. 2006. A Clustering Approach for Nearly Unsupervised Recognition of Nonliteral Language. In *Proc. EACL*, 329–336.

Chollet, F., et al. 2015. Keras. <https://github.com/fchollet/keras>.

Coles, K. 2017. Getting at metaphor. paper presented at the digital humanities 2017 conference.

D’Agostino, G., and Scala, A. 2014. *Networks of networks: the last frontier of complexity*. Springer.

Dalvean, M. 2013. Ranking contemporary american poems. *Digital Scholarship in the Humanities* 30(1):6–19.

Graham, E., and Brook, S. S. 2016. The printing press as metaphor. *Digital Humanities Quarterly* 10(3).

Hamilton, R.; Bramwell, E.; and Hough, C. 2016. Mapping metaphor with the historical thesaurus: a new resource for investigating metaphor in names.

Hayles, N. K., and Pressman, J. 2013. *Comparative textual media: Transforming the humanities in the postprint era*. University of Minnesota Press.

Kao, J., and Jurafsky, D. 2012. A computational analysis of style, affect, and imagery in contemporary poetry. In *CLJL@NAACL-HLT*, 8–17.

Kesarwani, V.; Inkpen, D.; Szpakowicz, S.; and Tanasescu, C. 2017. Metaphor detection in a poetry corpus. In *Proceedings of the Joint SIGHUM Workshop at Association for Computational Linguistics*, 1–9.

Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Krishnakumar, S., and Zhu, X. 2007. Hunting elusive metaphors using lexical resources. In *Proceedings of the*

*Workshop on Computational approaches to Figurative Language*, 13–20. Association for Computational Linguistics.

Lorde, A. 2000. *The collected poems of Audre Lorde*. WW Norton & Company.

MARGENTO. 2012. *Nomadosophy*. Max Blecher Press.

McCurdy, N.; Lein, J.; Coles, K.; and Meyer, M. 2016. Poemage: Visualizing the sonic topology of a poem. *IEEE transactions on visualization and computer graphics* 22(1):439–448.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.

Mohammad, S. M.; Shutova, E.; and Turney, P. D. 2016. Metaphor as a Medium for Emotion: An Empirical Study. In *Proc. \*SEM*, 23–33.

Neuman, Y.; Assaf, D.; Cohen, Y.; Last, M.; Argamon, S.; Howard, N.; and Frieder, O. 2013. Metaphor Identification in Large Texts Corpora. *PLOS ONE* 8(4):1–9.

Núñez, A.; Gerloff, M.; Do Dinh, E.-L.; Rapp, A.; Gehring, P.; and Gurevych, I. 2017. A wind of change-shaping public opinion of the arab spring using metaphors. paper presented at the digital humanities 2017 conference.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global Vectors for Word Representation. In *Proc. EMNLP*, volume 14, 1532–1543.

Rybak, C. 2016. Poetry. *Digital Pedagogy in the Humanities: Concepts, Models, and Experiments*.

Sack, G. 2012. Character networks for narrative generation. In *Intelligent Narrative Technologies: Papers from the 2012 AIIDE Workshop, AAAI Technical Report WS-12-14*, 38–43.

Shutova, E.; Kiela, D.; and Maillard, J. 2016. Black Holes and White Rabbits: Metaphor Identification with Visual Features. In *Proc. 2016 NAACL: HLT*, 160–170.

Speer, R., and Havasi, C. 2012. Representing General Relational Knowledge in ConceptNet 5. In *Proc. LREC*, 3679–3686.

Tanasescu, R., and Tanasescu, C. 2018. Translator networks of networks. the case of *Asymptote* journal. *Complexity Theory (K. Marais and R. Maylaerts, Eds.)*.

Tanasescu, C.; Paget, B.; and Inkpen, D. 2016. Automatic classification of poetry by meter and rhyme. In *FLAIRS Conference*, 244–249.

Turney, P. D.; Neuman, Y.; Assaf, D.; and Cohen, Y. 2011. Literal and Metaphorical Sense Identification through Concrete and Abstract Context. In *Proc. EMNLP*, 680–690. Association for Computational Linguistics.

Vala, H.; Dimitrov, S.; Jurgens, D.; Piper, A.; and Ruths, D. 2016. Annotating characters in literary corpora: A scheme, the charles tool, and an annotated novel. In *LREC*.

# Deep Learning for Political Science<sup>1</sup>

Kakia Chatsiou (University of Essex) and Slava Jankin Mikhaylov (Hertie School)<sup>2</sup>

## Introduction

Political science, and social science in general, have traditionally been using computational methods to study areas such as voting behavior, policy making, international conflict, and international development. More recently, increasingly available quantities of data are being combined with improved algorithms and affordable computational resources to predict, learn, and discover new insights from data that is large in volume and variety. New developments in the areas of machine learning, deep learning, natural language processing (NLP), and, more generally, artificial intelligence (AI) are opening up new opportunities for testing theories and evaluating the impact of interventions and programs in a more dynamic and effective way. Applications using large volumes of structured and unstructured data are becoming common in government and industry, and increasingly also in social science research.

This chapter offers an introduction to such methods drawing examples from political science. Focusing on the areas where the strengths of the methods coincide with challenges in these fields, the chapter first presents an introduction to AI and its core technology – machine learning, with its rapidly developing subfield of deep learning. The discussion of deep neural networks is illustrated with the NLP tasks that are relevant to political science. The latest

---

<sup>1</sup> Forthcoming in Cuirini, Luigi and Robert Franzese, eds. *Handbook of Research Methods in Political Science and International Relations*. Thousand Oaks: Sage.

<sup>2</sup> Email address: [jankin@hertie-school.org](mailto:jankin@hertie-school.org)

advances in deep learning methods for NLP are also reviewed, together with their potential for improving information extraction and pattern recognition from political science texts.

We conclude by reflecting on issues of algorithmic bias – often overlooked in political science research. We also discuss the issues of fairness, accountability, and transparency in machine learning, which are being addressed at the academic and public policy levels.

## **AI: Machine Learning and NLP**

The European Commission (2019) defines AI as ‘systems that display intelligent behaviour by analysing their environment and taking actions – with some degree of autonomy – to achieve specific goals’. As a scientific discipline, AI includes several techniques like machine learning (with deep learning and reinforcement learning as specific examples), machine reasoning, and robotics (European Commission, 2019). However, much of what is discussed as AI in the public sphere is machine learning, which is an ‘algorithmic field that blends ideas from statistics, computer science and many other disciplines [...] to design algorithms that process data, make predictions, and help make decisions’ (Jordan, 2019).

Machine learning has a history of successful deployment in both industry and academia, going back several decades. Deep learning has more recently made great progress in such applications as speech and language understanding, computer vision, and event and behavior prediction (Goodfellow et al., 2016). These rapid technological advances and the promise of automation and human-intelligence augmentation (Jordan, 2019) reignited debates on AI’s impact on jobs and markets (Brynjolfsson et al., 2018; Samothrakis, 2018; Schlogl and Sumner, 2018) and the need for AI governance (Aletas et al., 2016; Benjamins et al., 2005).



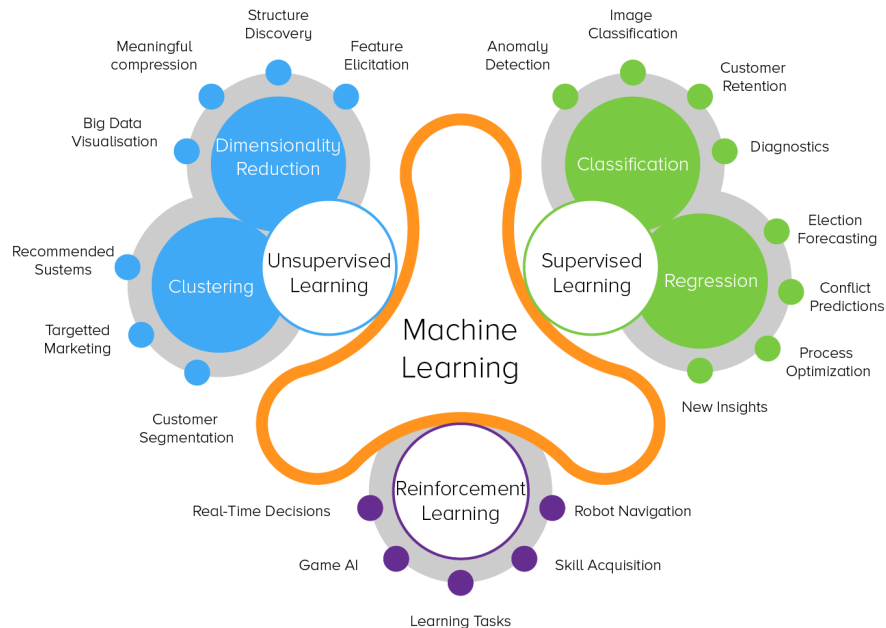
Machine learning (and deep learning as its subfield) is defined as the ‘field of study that gives computers the ability to learn without being explicitly programmed’ (Samuel, 1959). In this context, ‘learning’ can be viewed as the use of statistical techniques to enable computer systems to progressively improve their performance on a specific task using data without being explicitly programmed (Goldberg and Holland, 1988). To be able to learn how to perform a task and become better at it, a machine should:

- be provided with a set of example information (inputs) and the desired outputs. The goal is then to learn a general rule that can take us from the inputs to the outputs. This type of learning is called *Supervised Learning*. This works well even in cases when the input information is not available in full;
- be provided with an incomplete set of example information to learn from, where some of the target outputs are missing. This type of learning is called *Semi-supervised Learning*. When example information is available in one domain and we want to apply the knowledge to another domain with no available example information, this is called *Transfer Learning*;
- obtain training labels for a small number of instances while at the same time optimize which elements it needs to learn labels for. This is called *Active Learning*, and, in some cases, it can be implemented interactively in order to ask a human user for information on how best to label different elements;
- be asked to find structure in the input without having any labels provided in advance (as input). This type of learning is called *Unsupervised Learning* and can be used both for discovering hidden patterns in the data as well as learning features or parameters from the data;

be given information not about the structure of the data itself but rather about whether it has learned something correctly or incorrectly, in the form of rewards and punishments. This is called *Reinforcement Learning* and is the type of learning best performed in dynamic

environments such as when driving a vehicle or playing a game against an opponent (Bishop, 2006).

Figure 55.1 summarizes different types of learning and how they relate to their subtasks.



One of the most fruitful areas of machine learning applications in political science relates to work that treats text as data. Such quantitative text analysis could involve the following tasks: Assign a category to a group of documents or other elements (*'classification'*): this is useful when, for example, there is a need to understand audience sentiment from social media or customer reviews or sort party manifestos into predefined categories on the ideological spectrum. Spam filtering is an example of classification from our contemporary daily life, where the inputs are email (or other) messages and the classes are 'spam' and 'not spam'. The task involves a dataset containing text documents with labels, which is then used to train a classifier aiming to automatically classify the text documents into one or more predefined categories. Inputs are divided into two or more classes, and the algorithm assigns unseen inputs to one or more (multi-

label classification) of these classes. This is typically tackled via supervised learning. In political science work, such models have been used, for example, to understand US Supreme Court decisions (Evans et al., 2007), party affiliation (Yu et al., 2008), and in measuring polarization (Peterson and Spirling, 2018).

Separate elements into groups (*'clustering'*): this is similar to classification, only the groups are not known beforehand, hence this task usually involves unsupervised learning. Sanders et al. (2017) and Preoțiu-Pietro et al. (2017) are examples of the potential use of clustering to better understand political ideologies and parliamentary topics.

Reduce the complexity of data: *dimensionality reduction* simplifies inputs by mapping them into a lower-dimensional space. Principal-components analysis and related methods like correspondence analysis have been used to analyze preferences for foreign aid (Baker, 2015) and the ideological mapping of candidates and campaign contributors (Bonica, 2014). Topic modelling is a related problem, where multiple documents are reduced to a smaller set of underlying themes or topics. Feature extraction is a type of dimensionality reduction task and can be accomplished using either semi-supervised or unsupervised learning. Selection and extraction of text features from documents or words is essential for text mining and information retrieval, where learning is done by seeking to reduce the dimension of the learning set into a set of features (Uysal, 2016; Nguyen et al., 2015).

Perform structured predictions: *structured prediction* or *structured (output) learning* is an umbrella term for supervised machine learning techniques that involve predicting structured objects, rather than scalar discrete or real values (BakIr, 2007). In Lafferty et al. (2001), for example, the issue of translating a natural-language sentence into a syntactic representation such

as a parse tree can be seen as a structured-prediction problem in which the structured-output domain is the set of all possible parse trees.

The table below summarizes some of these techniques:

Method	Type of learning	Examples
Classification	Supervised	<ul style="list-style-type: none"> <li>• understand audience sentiment from social media</li> <li>• sort party manifestos into predefined categories on the ideological spectrum</li> <li>• understand US Supreme Court decisions (Evans et al., 2007),</li> <li>• extract party affiliation (Yu et al., 2008),</li> <li>• measure polarization (Peterson and Spirling, 2018)</li> </ul>
Clustering	Unsupervised	<ul style="list-style-type: none"> <li>• understand political ideologies and parliamentary topics (Sanders et al., 2017; Preoțiu-Pietro et al., 2017)</li> </ul>
Dimensionality Reduction e.g. Topic modelling, Feature Extraction	Semi-supervised Unsupervised	<ul style="list-style-type: none"> <li>• preferences for foreign aid (Baker, 2015)</li> <li>• ideological mapping of candidates and campaign contributors (Bonica, 2014)</li> <li>• extraction of text features from documents (Uysal, 2016; Nguyen et al., 2015)</li> </ul>

*Table 55.1 Overview of machine learning methods and examples from political science*

These political text-as-data applications are related to the broader field of NLP, which is concerned with the interactions between computers and human or natural languages (rather than formal languages). After the 1980s and alongside the developments in machine learning and advances in hardware and technology, NLP has mostly evolved around the use of statistical models to automatically identify patterns and structures in language, through the analysis of large sets of annotated texts or corpora. In addition to document classification and dimensionality-reduction applications in political science, leveraging the latest developments in machine learning and deep learning methods, the NLP field has made significant progress on several additional tasks:

- *Extracting text from an image.* Such a task usually involves a form of *Optical Character Recognition*, which can help with determining the corresponding text characters from an image of printed or handwritten text.
- *Identifying boundaries and segment text into smaller units (for example from documents to characters).* Examples of such tasks include morphological segmentation, word segmentation, and sentence-boundary disambiguation.

*Morphological segmentation* is the field of separating words into individual morphemes and identifying the class of the morphemes is an essential step of text pre-processing before textual data can be used as an input in some machine learning algorithms. Some such tasks can be quite challenging to perform automatically, sometimes depending on morphological complexity (i.e. the internal structure of words) of the language being considered.

*Word segmentation* or *tokenization* makes possible the separation of continuous text into separate words.

*Sentence-boundary disambiguation* helps identify where a sentence starts and where it ends. This is not as simple as identifying where a period or other punctuation mark is, since not all punctuation signals the end of a sentence (consider abbreviations, for example) and not all sentences have punctuation.

*Assigning meaning to units. Part-of-speech tagging,* involves automatically determining and assigning a part of speech (e.g., a verb or a noun) to a word is usually the first step to looking at word context and meaning. Of course many words have more than one meaning or could be assigned different parts of speech, which can prove challenging for NLP, as it needs to select the meaning which makes more sense in the current context. With the emergence of deep learning methods, word embeddings have been used to capture semantic properties of words and their context (see the next section for a more detailed presentation).

*Extracting information from the text and synthesizing it.* NLP tasks such as *Named Entity Recognition*, *Sentiment Analysis*, *Machine Translation* and *Automated Text Summarization* build on the above tasks in order to identify and extract specific content from texts and synthesize it to generate new insights or content.

*Machine Translation* studies ways to automate the translation between languages. Deep learning methods are improving the accuracy of algorithms for this task (Nallapati et al., 2016). This leads to scaling-up opportunities in comparative politics research (de Vries et al., 2018).

*Named Entity Recognition* helps determine the elements in a text that are proper names (such as people or places) and what type of elements they are (for example, person, location, organization, etc.).

*Sentiment Analysis* is the automatic extraction of opinions or subjective information from a set of documents or reviews, to determine ‘polarity’ about specific ideas. For example, scholars have used *Sentiment Analysis* to identify trends of public opinion in social media (Ceron et al., 2014; Proksch et al., 2015).

*Automated Text Summarization* is a common dimensionality-reduction task in machine learning and NLP. It involves producing a readable, coherent, and fluent summary of a longer text, which should include the main points outlined in the document. *Extractive summarization* involves techniques such as identifying key words from the source document and combining them into a continuous text to make a summary. *Abstractive summarization* involves automatically paraphrasing or shortening parts of the original text.

With the deep learning methods being extremely data hungry, we believe that a primary area where the field will benefit from the latest technology is in the text-as-data or broader NLP domain. In what follows, we outline several deep learning models that have made recent advances in NLP possible and highlight how they can be used in political science research.

# Deep Learning NLP for Political Analysis

## Understanding ‘Learning’

To define *deep learning* and understand the difference between deep learning and other machine learning approaches, first we need some idea of what machine learning algorithms *do*. As mentioned above, the field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

But what does *learning* mean in this context?

A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$  if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ . (Mitchell, 1997; our emphasis)

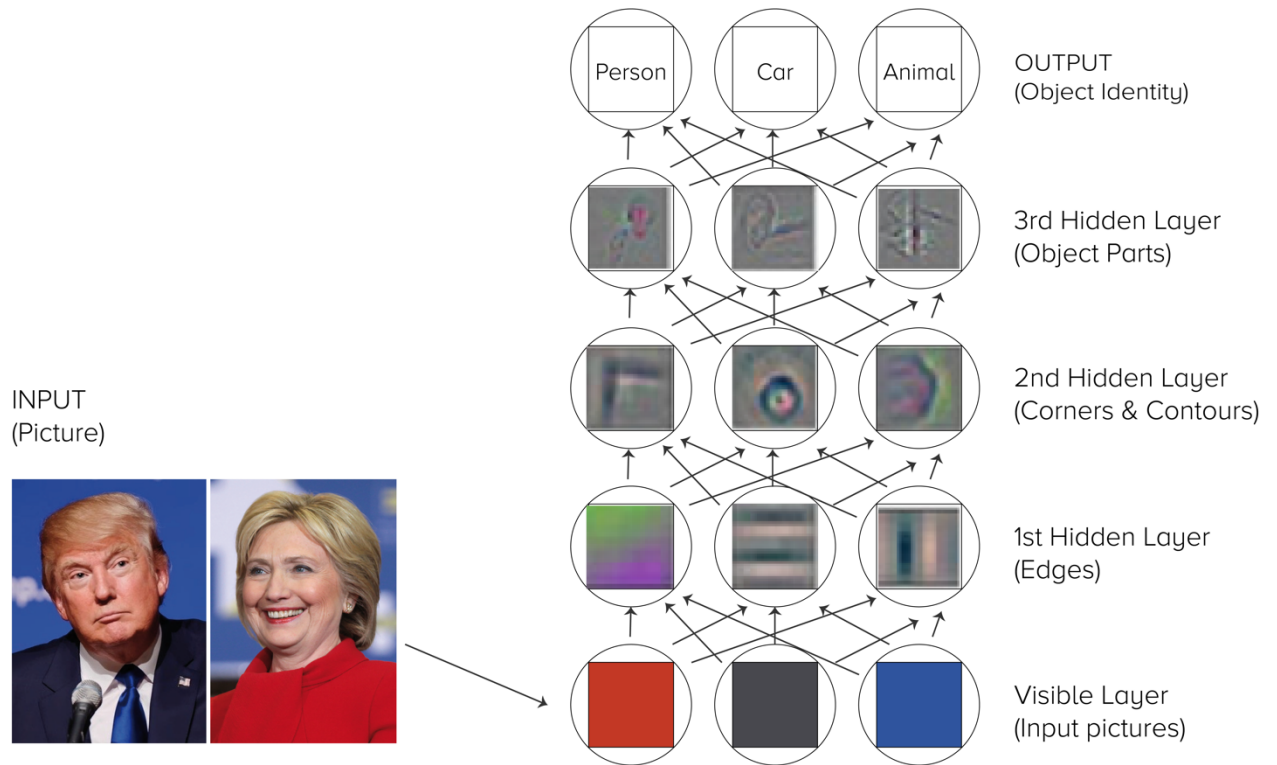
This type of learning that particularly pertains to NLP regardless of the type of learning (supervised, unsupervised, active, etc.) is very much based on a ‘bag-of-words’ approach that only considers one dimension of the text, without taking onboard any of the contextual information – a rather ‘shallow’ type of learning.

Deep learning, on the other hand, offers the potential to combine multiple *layers* of representation of information, sometimes grouped in a hierarchical way.

## Understanding ‘Deep’

Deep learning is a type of machine learning (representation learning) that enables a machine to automatically learn the patterns needed to perform regression or classification when provided with raw data. The approach puts an emphasis on learning successive *layers* of increasingly meaningful representations. It involves multiple levels of representation. Deng (2014: 199–200) define deep learning as a class of machine learning algorithms that

use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation, and each successive layer uses the output from the previous layer as input; learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners; learn multiple levels of representations that correspond to different levels of abstraction – the levels form a hierarchy of concepts.



In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image-recognition application, the raw input may be a matrix of pixels, the first representational layer may abstract the pixels and encode edges, the second layer may compose and encode the arrangements of edges, the third layer may encode eyes and a nose, and the fourth layer may recognize that the image contains a face (for more information about feature visualizations from computer-vision deep neural networks, see Olah et al., 2017 and Zhang and Zhu, 2018). Importantly, a deep learning process can learn which



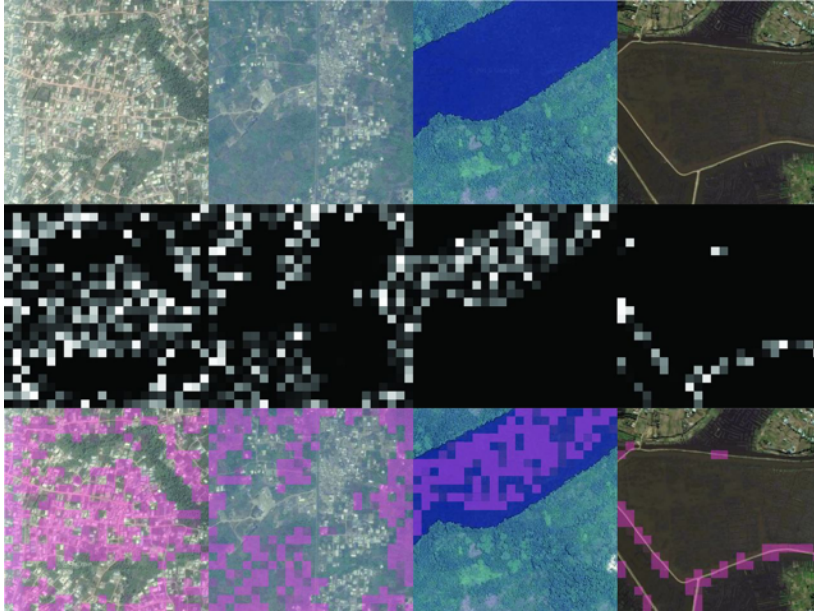
features to optimally place in which level *on its own*. Figure 55.2 shows how a deep learning hierarchy of complex concepts can be built from simpler concepts.

We will next discuss the application of deep learning algorithms in generating insights from images and text data.

## **Working with Image Data**

Convolutional neural networks (CNNs) are a category of artificial neural networks that have proven very effective when trying to classify or detect features in images. CNNs have been very successful at identifying objects, faces, and traffic signs in images and are currently advancing computer vision in robotics and self-driving vehicles.

CNNs have been trained on satellite imagery to map and estimate poverty, where data on economic livelihoods are scarce and where outcomes cannot be studied via other data. Jean et al. (2016) combine satellite imagery with survey data from five African countries (Nigeria, Tanzania, Uganda, Malawi, and Rwanda) to train a CNN to identify image features that can explain up to 75% of the variation in the local-level economic outcomes by estimating consumption expenditure. Figure 55.3 shows four different convolutional filters used for extracting these features, which identify (from left to right) features corresponding to urban areas, non-urban areas, water and roads. Babenko et al. (2017) focus on an urban subsample of satellite images in Mexico (using images from Digital Globe and Planet) identifying rural and urban ‘pockets’ of poverty that are inaccessible and changing frequently – areas that are unlikely to integrate without the support of the necessary policy measures (Figure 55.4).



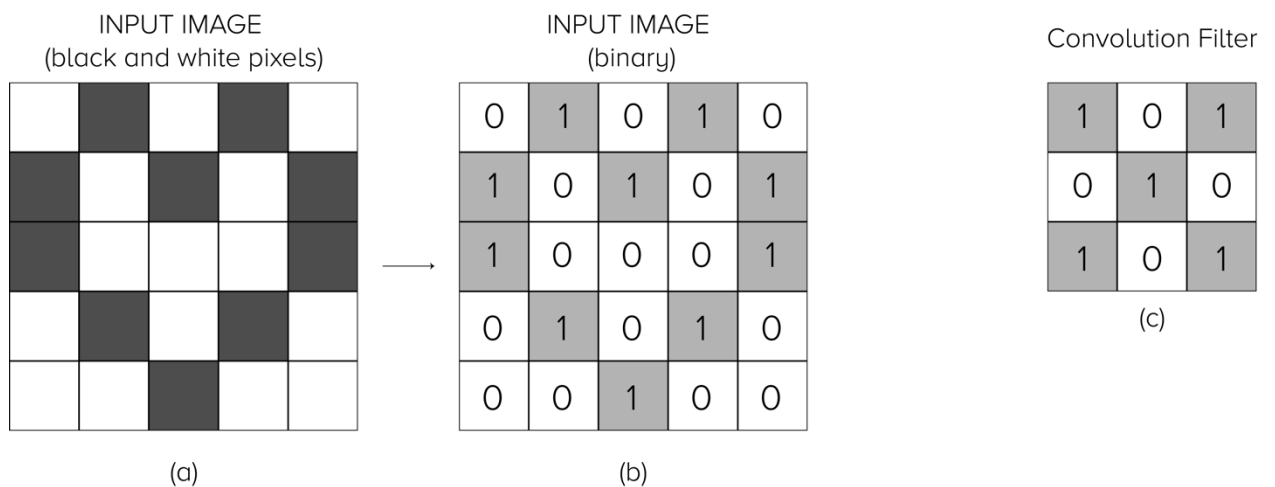
CNNs have also been used to map informal settlements (‘slums’) in developing countries, using high- and low-resolution satellite imagery (Helber et al., 2018), to help international aid organizations to provide effective social and economic aid.

But how do they work?

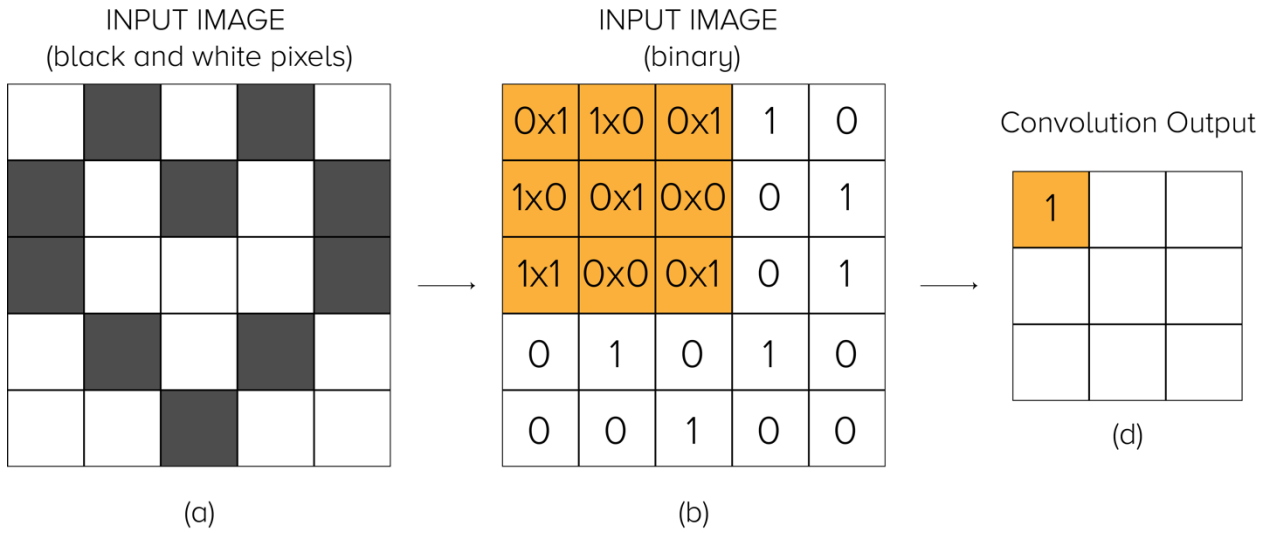
Analogous to how children learn to recognize a cat from a dog, we need to ‘show’ an algorithm millions of pictures (‘input’) of a dog before it can reliably make generalizations and predictions for images it has never seen before. However, machines do not ‘see’ in the same way

we do – their ‘language’ consists of numbers. One way around this is to represent every image as multi-dimensional arrays of numbers, and CNNs offer a way to move from an image to a set of vectors.

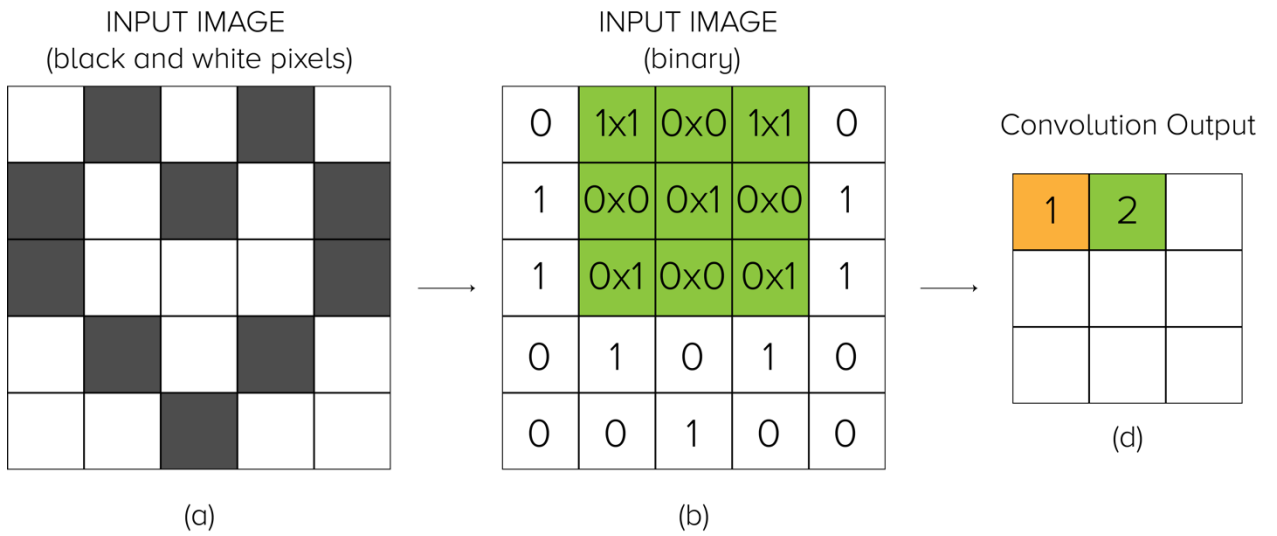
The main building block of CNN is the *convolutional layer, filter, or kernel*. Convolution is a mathematical operation that allows us to condense information by combining two functions into one. Take the very simple, pixelated representation of a black and white heart in Figure 55.5 element (a) for example. If each cell is a pixel, then we could represent black pixels with value 1 and white pixels with value 0 (see Figure 55.5, element (b)) – this is the ‘input’.



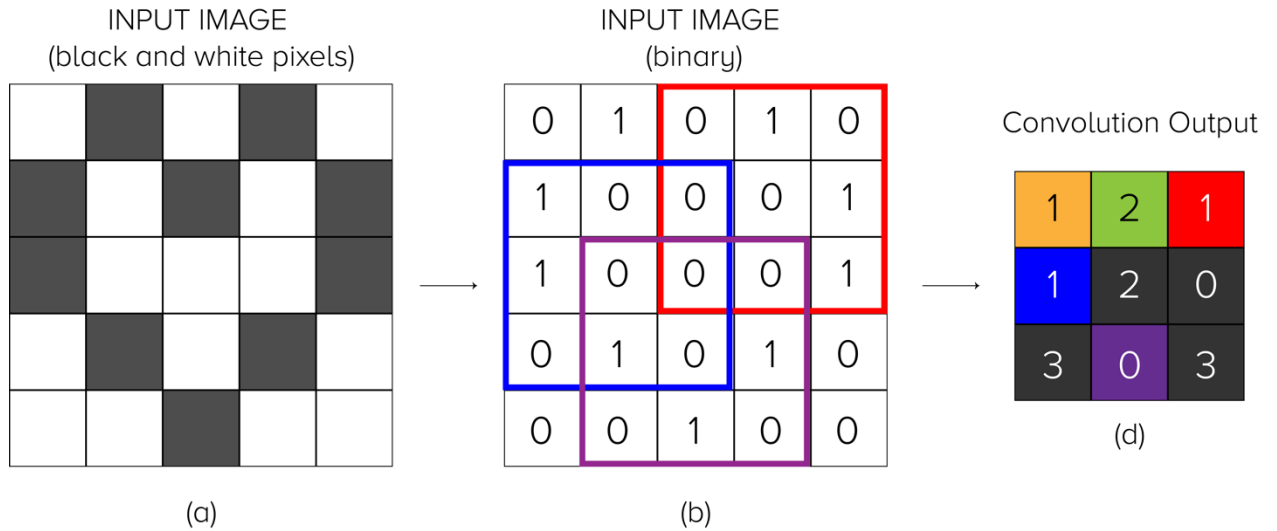
Using a filter, as in Figure 55.5 element (c), with predefined black and white pixels, we can now perform a convolution and create a ‘feature map’ (Figure 55.6, element (d)) by layering the filter on top of the input and sliding it for each row. At every step, we perform element-wise matrix multiplication and sum the result, which goes into the feature map – represented in the black background in Figure 55.6.



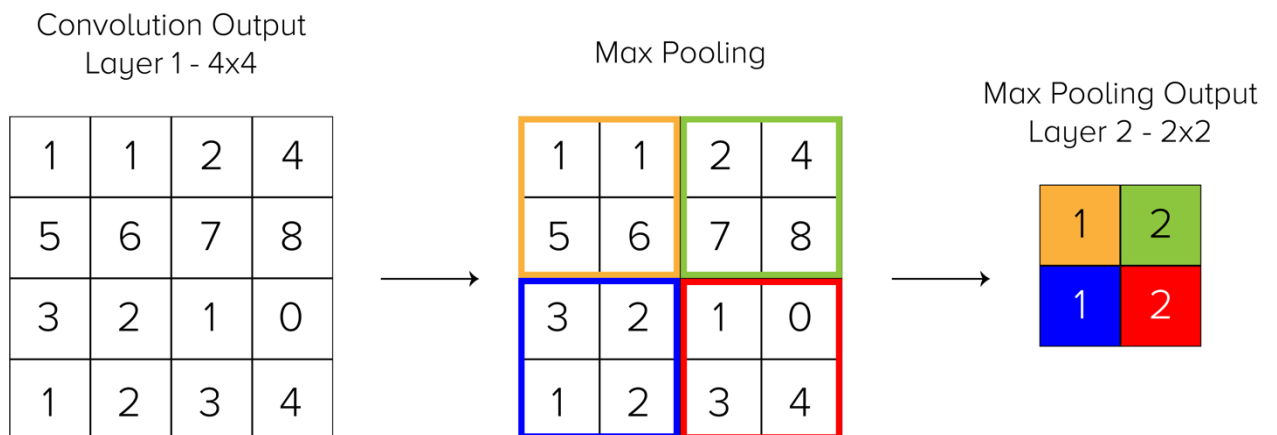
We then slide the filter over the next position and perform the same multiplication (see Figure 55.7).



We do the same until the 'input' is reduced from a 5x5 matrix to a 3x3 feature map.

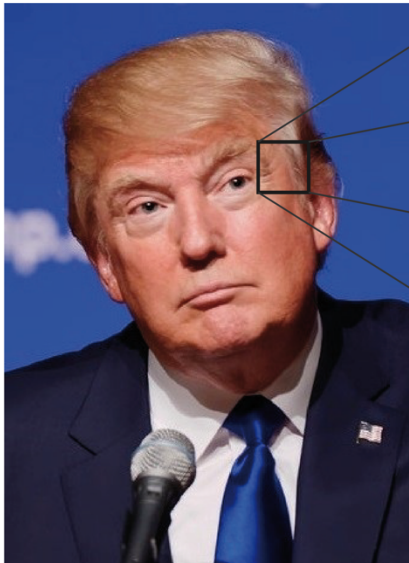


We repeat until the 'input' is reduced from a 5x5 matrix to a 3x3 feature map, as in Figure 55.8 element (c) above. The example above is a two-dimensional convolution using a 3x3 filter – in reality, these convolutions are performed in three dimensions (width, height, and RGB color channel) with the filter being 3D as well. Multiple convolutions take place on an input, each using a different filter with a distinct feature map as the output. After a convolution operation, we usually perform *pooling* (usually *max pooling*, i.e. taking the max value in the pooling window) to reduce the dimensionality and reduce the number of parameters (see Figure 55.9).



This is crucial when dealing with the volume of data that is fed to the algorithm, as it both speeds training time and helps avoid overfitting of the algorithm.

INPUT  
(Picture)



WHAT THE COMPUTER SEES

11	113	95	91	98	91	85	82	91	104	101	101	136	206	203	178	187	145	127	136	16
13	104	95	91	82	88	85	91	104	107	95	95	111	165	199	183	133	104	120	120	11
20	104	111	91	82	88	91	98	101	101	101	124	140	136	129	111	111	101	101	104	9
07	107	107	88	79	95	104	95	101	104	111	107	104	113	117	104	98	98	98	98	10
98	111	95	85	85	95	98	91	95	98	101	104	107	111	113	104	98	98	91	95	10
04	107	85	82	85	91	91	95	95	107	107	111	113	107	111	111	104	95	98	101	10
01	91	79	82	82	88	88	98	101	104	111	113	107	104	113	111	98	104	104	101	12
98	82	79	79	82	85	88	95	107	107	107	107	113	111	127	127	104	111	107	129	16
98	79	82	79	79	91	88	98	111	104	111	113	117	133	187	219	183	161	165	152	14
82	82	79	85	85	91	88	101	111	107	111	113	168	190	255	255	228	212	183	161	16
82	85	82	91	98	95	95	95	107	107	111	142	241	255	255	235	255	238	232	248	21
98	85	85	98	104	104	104	104	117	113	104	197	255	255	255	215	255	255	251	212	21
82	85	85	95	101	107	101	124	120	113	104	181	241	251	235	199	212	212	219	206	18
82	85	98	101	107	101	111	117	120	117	107	165	210	219	199	199	203	199	212	241	25
98	91	101	107	111	107	117	101	101	113	107	140	194	215	181	215	226	210	206	241	25
95	91	95	104	111	113	117	111	98	113	129	133	210	210	203	210	210	199	178	199	23
91	88	101	101	104	117	111	117	111	111	118	127	215	232	212	199	194	187	203	206	19
91	95	104	98	107	117	120	111	111	101	113	158	206	194	206	203	181	178	212	199	18
01	104	98	98	129	161	124	113	113	107	101	181	199	197	210	199	181	183	199	199	17
13	101	111	117	219	228	197	133	107	98	95	107	187	203	228	171	149	161	174	199	14

image classification →  
 82% person  
 15% animal  
 2% hat  
 1% vase

CNNs seem to suit the task of image classification, as they can help us predict a distribution over specific labels (as in Figure 55.10) to indicate confidence of prediction for a given image. But what about text data?

## Working with Text Data

The study of political discourse using text as data has a long tradition in political science. Political texts have long been used as an important form of social practice that contributes to the construction of social identities and relations (Fairclough, 1989, 1992; Laclau and Mouffe, 1985). Text as a representation of discourses has been studied systematically to derive information about actors and combine them with additional resources such as surveys and observations, as well as knowledge and reflective understanding of the context by scholars, yet not in a reproducible and quantifiable way (see Blommaert and Bulcaen, 2000, for a review).

Over the past two decades, scholars have sought to extract information such as policy and ideology positions and gauge citizen political engagement by treating words as data in a more

consistent way. Since some of the earliest implementations of text-scaling methods such as *Wordscores* (Laver et al., 2003) and *Wordfish* (Slapin and Proksch, 2008) to estimate party positions from texts and the increasing availability of annotated political corpora, the availability and complexity of quantitative text-analysis methods have increased dramatically (Barberá, 2015; Grimmer and Stewart, 2013; Herzog and Benoit, 2015; Lauderdale and Herzog, 2016). Most of these methods tend to involve a ‘bag-of-words’ approach to determine relevance and cluster documents or their parts in groups (see also Laver, 2014). Such approaches assume that each document can be represented by a multiset (‘bag’) of its words, that ignores word order and grammar. Word frequencies in the document are then used to classify the document into a category. Some methods like *Wordscores* employ a version of the Naive Bayes classifier (Benoit and Nulty, 2013) in a supervised learning setting by leveraging pre-labelled training data, whereas others, like *WordFish*, are based on a Poisson distribution of word frequencies, with ideological positions estimated using an expectation-maximization algorithm (Proksch and Slapin, 2009; Slapin and Proksch, 2008).

What these approaches do not capture, though, is the linguistic and semiological context, i.e. the information provided by the words around the target elements. Such a context would allow for a better representation of that context and offer a richer understanding of word relationships in a political text. One way to do that is by using *word embeddings*, a set of methods to model language, combining concepts from NLP and graph theory.

## Representing Words in Context: Word Embeddings

Word embeddings are a set of language modelling and dimensionality-reduction techniques, where words or phrases from a document are mapped to vectors or numbers. They usually involve a mathematical embedding from a space with a single dimension for each word to a

continuous vector space with a reduced dimension. The underlying idea is that ‘[y]ou shall know a word by the company it keeps’ (Firth, 1957: 11), and it has evolved from ideas in structuralist linguistics and ordinary language philosophy, as expressed in the work of Zellig Harris, John Firth, Ludwig Wittgenstein, and vector-space models for information retrieval in the late 1960s to the 1980s. In the 2000s, Bengio et al. (2006) and Holmes and Jain (2006) provided a series of papers on the ‘Neural Probabilistic Language Models’ in order to address the issues of dimensionality of word representations in contexts, by facilitating learning of a ‘distributed representation of words’. The method developed gradually and really took off after 2010, partly due to major advances in the quality of vectors and the training speeds of the models.

There are many variations of word-embedding implementations, and many research groups have created similar but slightly different types of word embeddings that can be used in the deep learning pipelines. Popular implementations include Google’s Word2Vec (Mikolov et al., 2013), Stanford University’s GloVe (Pennington et al., 2014), and Facebook’s fastText (Bojanowski et al., 2016). For a recent discussion of word embeddings in a political science context, see Spirling and Rodriguez (2019).

Now that we have a mechanism to turn text into dense vectors (very much like we did with the image of the heart in the previous section), let’s see how CNNs can be applied to NLP tasks for political texts.

## CNNs for Text Analysis

CNNs have recently been applied to various NLP tasks with very good results in accuracy and precision (Johnson and Zhang, 2014; Kalchbrenner et al., 2014; Kim, 2014).



Instead of image pixels, each row of the matrix corresponds to one token (usually a word, but it could also be a character; see Jacovi et al., 2018 and Zhang et al., 2015) or rather a vector that represents a word. These vectors are typically *word embeddings* such as Word2Vec or GloVe (see previous section). Kim (2014) describes the general approach of using CNNs for NLP, assuming a single layer of networks and pretrained static word vectors on very large corpora (Word2Vec vectors from Google, trained on 100 billion tokens from Google News). Sentences are mapped to embedding vectors and are available as a matrix input to the model. Convolutions are performed across the input word-wise using differently sized kernels, such as two or three words at a time. The resulting feature maps are then processed using a max pooling layer to condense or summarize the extracted features. Figure 55.11 shows a single-layer CNN architecture for sentence classification from Kim (2014).

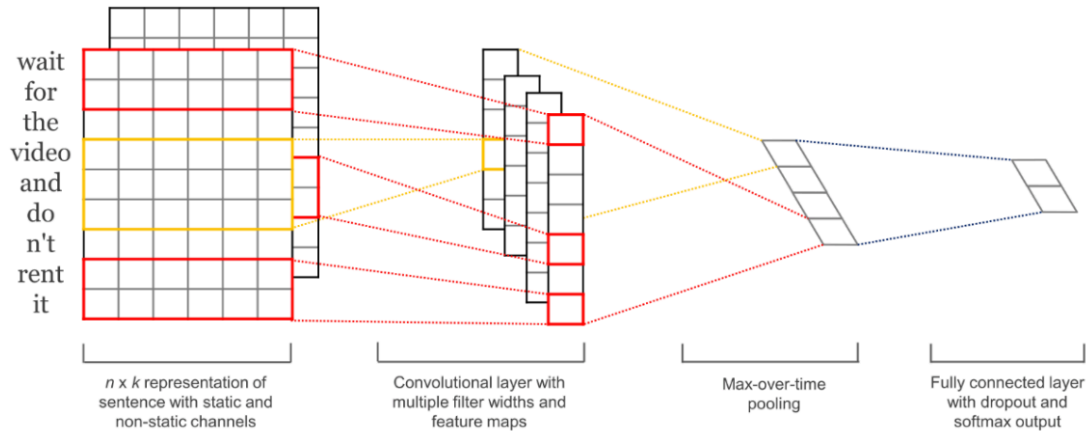
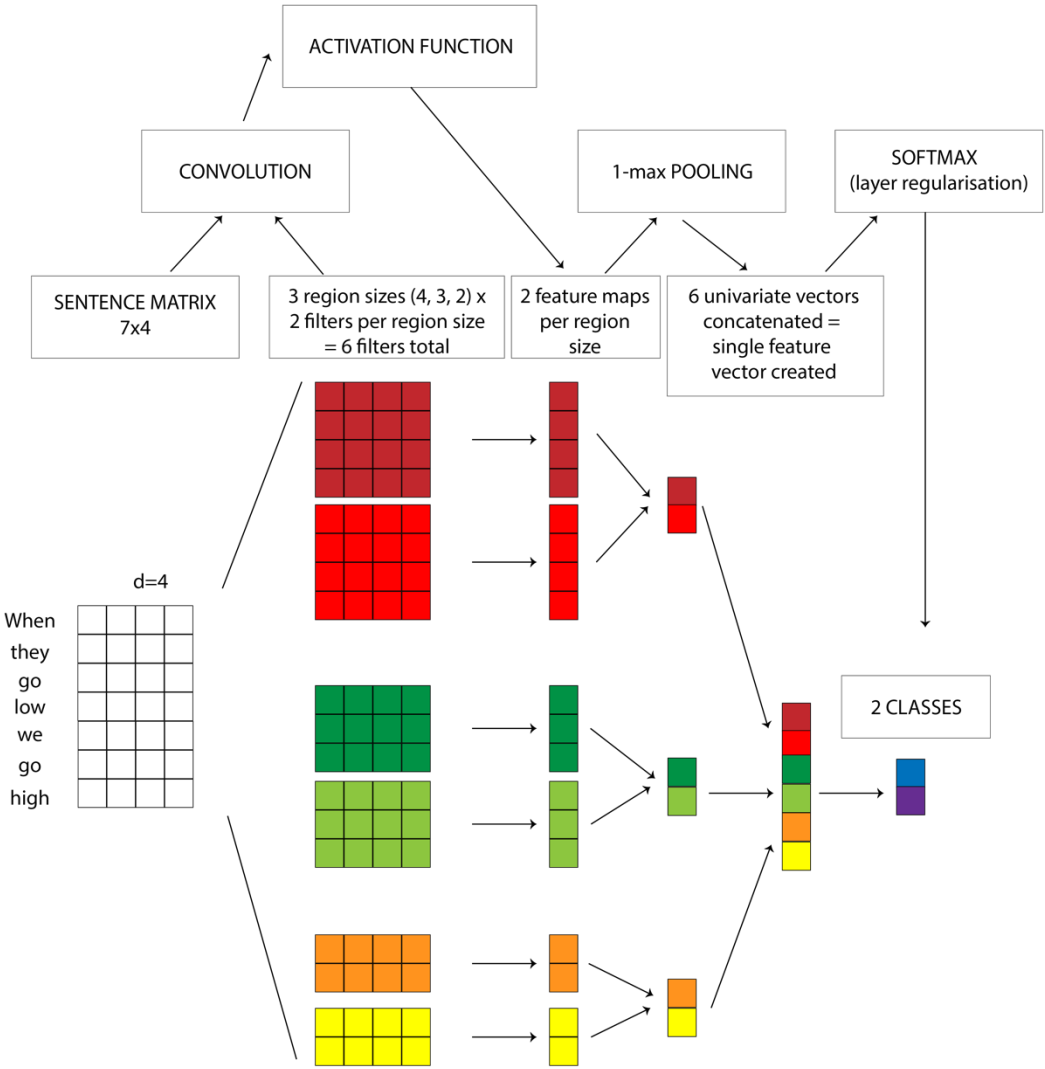


Figure 11. Illustration of a single-layer Convolutional Neural Network (CNN) architecture for sentence classification from (Kim, 2014)

Figure 55.12 shows how a CNN would work for a sentence-classification task adapted from Zhang and Wallace (2015). Assuming the sentence we wanted to classify was Michelle Obama's 'When they go low, we go high', this would generate a  $7 \times 4$  sentence matrix, with three filter region sizes: 2, 3, and 4, each of which has two filters for each region size. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then,

1-max pooling is performed over each map, i.e. the largest number from each feature map is recorded. Thus, a univariate feature vector is generated from all six maps, and these six features are concatenated to form a feature vector for the penultimate layer. The final *softmax* layer then receives this feature vector as input and uses it to classify the sentence; here, we assume binary classification and hence depict two possible output states.



Despite CNNs being a little unintuitive in their language implementation, they perform really well on tasks like text classification. They are very fast, as convolutions are highly parallelizable, form an integral part of computer graphics, and are implemented on graphical processing units (GPUs). They also work much better compared to other ‘bag-of-words’ approaches such as n-grams, as they can learn representations automatically without the need to represent the whole vocabulary (whereas in the case of n-grams, for example, if we had a large vocabulary, computing anything beyond tri-grams would become quite expensive in terms of computational power), with architectures as deep as 29 layers performing sufficiently well (Zhang et al., 2015).

CNNs have been successfully deployed for NLP tasks such as automatic summarization, fake news detection and text classification. Narayan et al. (2018), for example, apply CNNs to automatically summarize a real-world, large-scale dataset of online articles from the British Broadcasting Corporation (BBC). They demonstrate experimentally that this architecture captures long-range relationships in a document and recognizes related content, outperforming other state-of-the-art abstractive approaches when evaluated automatically and by humans.

Yamshchikov and Rezagholi (2018) develop a model of binary text classifiers based on CNNs, which helps them label statements in the political programs of the Democratic and Republican parties in the United States, whereas Bilbao-Jayo and Almeida (2018) propose a new approach to automate the analysis of texts in the Manifestos Project, to allow for a quicker and more streamlined classification of such types of political texts.

The Manifesto Project (Lehmann et al., 2018) includes data on parties’ policy positions, derived from content analysis of parties’ electoral manifestos. It covers over 1,000 parties from 1945 until today in over 50 countries on five continents. The corpus includes manually annotated

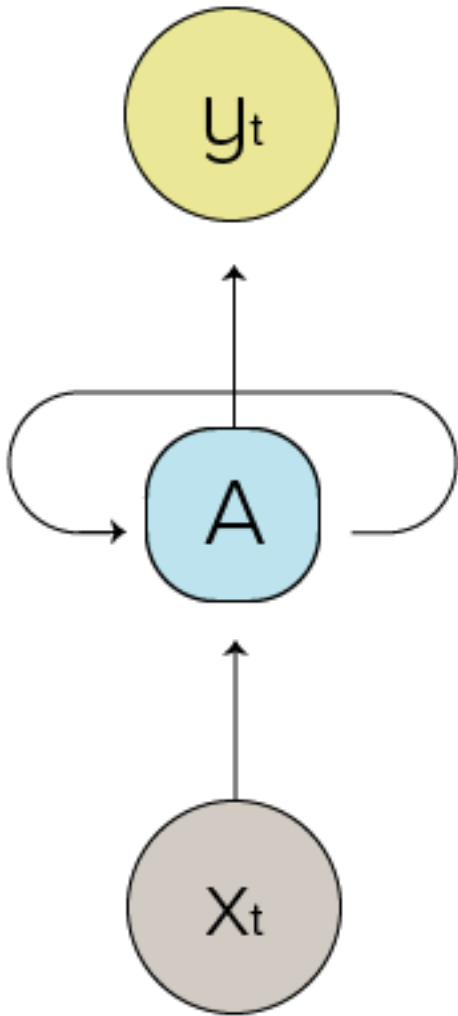
election manifestos using the Manifesto Project coding scheme, which is widely used in comparative politics research. Bilbao-Jayo and Almeida (2018) use multi-scale CNNs with word embeddings and two types of context data as extra features, like the previous sentence in the manifesto and the political party. Their model achieves reasonably high performance of the classifier across several languages of the Manifesto Project.

Another type of neural network that has shown good performance in NLP tasks are recurrent neural networks (RNNs) and, in particular, a variation of that algorithm, the long short-term memory (LSTM) RNNs.

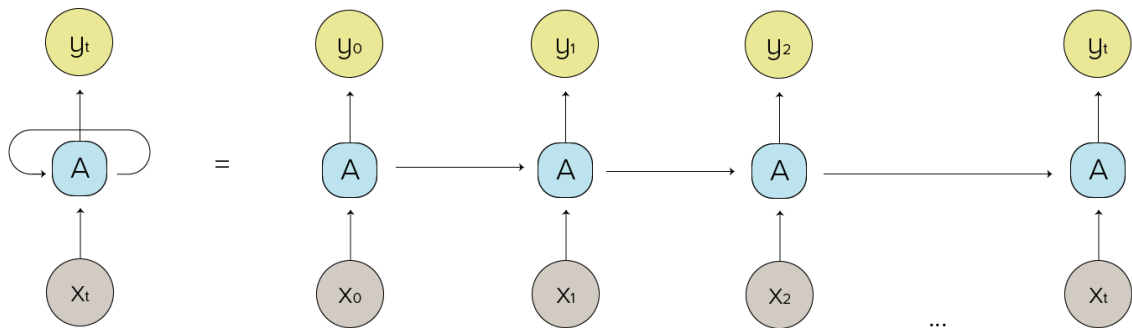
## LSTM RNNs for Text Analysis

As you read this paragraph, you understand each word based on your understanding of previous words – those right before this word, words expressed in the paragraphs and sections above, as well as words that you might have read in the previous chapters of this *Handbook* (or even words that you have read in other books and articles).

Every time we read a new word, we do not forget what we read before – our understanding has some degree of *persistence*. Unfortunately, CNNs cannot reason about previous steps in the learning process to inform later ones. RNNs overcome this issue because they permit loops, thus allowing for the information in the neural network to persist. A simple RNN is a class of artificial neural networks where connections between nodes form a directed graph along a sequence, incorporating previous knowledge (see Figure 55.13, adapted from Olah, 2015).

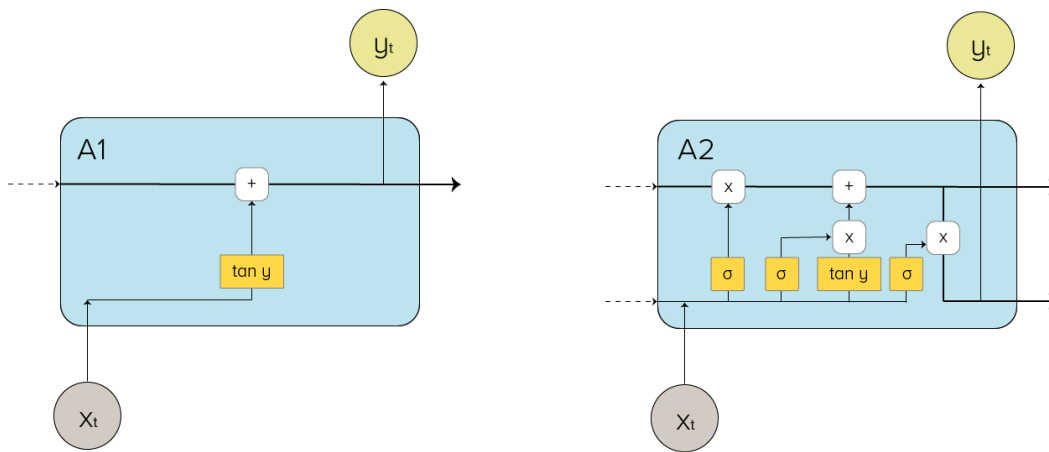


A sequence of RNN blocks can be regarded as multiple copies of the same network, linked to one another like a chain, each passing an input to its future self (Figure 55.14). This enables it to display dynamic temporal behavior for a time sequence and make these networks work really robustly with sequence data such as text, time-series data, videos, and even DNA sequences.



This suits textual data, which for the most part is sequence or list data, and which has been applied with success to NLP tasks such as speech recognition, language modelling, translation, and image captioning (Ba et al., 2014; Gregor et al., 2015). However, simple RNNs are not well suited for remembering information that is not close to the current node they are in (also called long-distance dependencies), a problem detailed in Bengio et al. (1994).

LSTM neural networks (Hochreiter and Schmidhuber, 1997) provide a solution to this issue. LSTMs also have the RNN chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, all interacting in a special way. Figure 55.15 shows the repeating module in a standard RNN with a single layer (A1) and an LSTM with 4 interacting layers (A2). The LSTM has the advantage of incorporating context from both the input ( $x$ ) and the previous knowledge (represented with dashed lines in A2) and also feed the augmented knowledge to the next iteration.



Standard LSTMs (like those in Figure 55.15) are *unidirectional* – in other words, they preserve information from the past inputs that have already passed through the different iterations of the hidden layers of the neural network. Take for example the following word sequence:

‘Let’s make ...’

There are a lot of possibilities for what word sequences could follow. All the sentences below are possible:

‘Let’s make some cake!’

‘Let’s make fun of Bob!’

‘Let’s make my friend see some sense, because I think she is making a huge mistake!’

What if you knew that the words that followed the first word sequence were actually these?

“Let’s make ... great again!”

Now the range of options is narrower, and it is easy to predict that the next word is probably a noun phrase such as ‘America’ or ‘this business’.

A unidirectional LSTM will only be able to consider past input ('let's make'). If you wish to see the future, you would need to use a *bidirectional* LSTM, which will run the input in two ways: one from the past to the future and one from the future to the past. When running backwards, it preserves information from the future, and by combining this knowledge with the past, it provides improved and more contextualized predictions.

Both types of LSTMs have been used to detect fake news and propaganda discourse in traditional and social media text, where the problem of detecting bots – automated social media accounts governed by software but disguised as human users – has strong societal and political implications.

Kudugunta and Ferrara (2018) propose a deep neural network based on contextual LSTM architecture, that exploits both content and metadata to detect bots at the tweet level. Their proposed technique is based on synthetic minority oversampling to generate a large labelled dataset suitable for deep nets training, from a minimal amount of labelled data (roughly 3,000 examples of sophisticated Twitter bots). The proposed model can, from the first tweet, achieve high classification accuracy (> 96%) in separating bots from humans.

Event detection using neural-network algorithms on tweets describing an event is another area of application of particular interest to media agencies and policy makers. Iyyer et al. (2014) assume that an individual's words often reveal their political ideology, and they use RNNs to identify the political position demonstrated at the sentence level, reporting that their model outperforms 'bag of words' or wordlists models in both the training and a newly annotated dataset. Makino et al. (2018), for example, propose a method to input and concatenate character and word sequences in Japanese tweets by using CNNs and reporting an improved accuracy score, whereas Rao and Spasojevic (2016) apply word embeddings and LSTM to text



classification problems, where the classification criteria are decided by the context of the application. They show that using LSTMs with word embeddings vastly outperforms traditional techniques, particularly in the domain of text classification of social media messages' political leaning. The research reports an accuracy of classification of 87.57%, something that has been used in practice to help company agents provide customer support by prioritizing which messages to respond to.

Other scholars have used hybrid neural-network approaches to work with text, by combining aspects of the CNN and RNN algorithms. Ajao et al. (2018), for example, propose a framework that detects and classifies fake news messages from Twitter posts, using such a hybrid of CNNs and LSTM RNNs, an approach that allows them to identify relevant features associated with fake news stories without previous knowledge of the domain. Singh et al. (2018) use a combination of the CNN, LSTM, and bidirectional LSTM to detect (overt and covert) aggression and hate speech on Facebook and social media comments, where the rise of user-generated content in social media coupled with almost non-existent moderation in many such systems has seen aggressive content rise.

Hybrid neural-network approaches also perform well in the task of automatic identification and verification of political claims. The task assumes that given a debate or political speech, we can produce a ranked list of all of the sentences based on their worthiness for fact checking – potential uses of this would be to predict which claims in a debate should be prioritized for fact-checking. As outlined in Atanasova et al. (2018), of a total of seven models compared, the most successful approaches used by the participants relied on recurrent and multi-layer neural networks, as well as combinations of distributional representations, matching claims' vocabulary against lexicons, and measures of syntactic dependency.

## Working with Multimodal Data

With the resurgence of deep learning for modeling data, the parallel progress in fields of computer vision and NLP, as well as with the increasing availability of text/image datasets, there has been a growing interest in using multimodal data that combines text with images. The popularity of crowd-sourcing tools for generating new, rich datasets combining visual and language content has been another important factor favoring multimodal input approaches.

Ramisa et al. (2018), for example, have compiled a large-scale dataset of news articles with rich metadata. The dataset, *BreakingNews*, consists of approximately 100,000 news articles collected over 2014, illustrated with one to three images and their corresponding captions. Each article is enriched with other data like related images from Google Images, tags, shallow and deep linguistic features (e.g., parts of speech, semantic topics, or outcomes of a sentiment analyzer), GPS latitude/longitude coordinates, and reader comments. The dataset is an excellent benchmark for taking joint vision and language developments a step further. Figure 55.16 illustrates the different components of the Ramisa et al. (2018) *BreakingNews* corpus, which contains a variety of news-related information for about 100K news articles. The figure shows two sample images. Such a volume of heterogeneous data makes *BreakingNews* a good benchmark for several tasks exploring the relation between text and images.



Figure 17. The BreakingNews dataset (Ramisa, Yan, Moreno-Noguer, & Mikolajczyk, 2018). The dataset contains a variety of news-related information including: the text of the article, captions, related images, part-of-speech tagging, GPS coordinates, semantic topics list or results of sentiment analysis, for about 100K news articles. The figure shows two sample images. All this volume of heterogeneous data makes BreakingNews an appropriate benchmark for several tasks exploring the relation between text and images.

The paper used CNN for source detection, geolocation prediction, and article illustration, and a mixed LSTM/CNNs model for caption generation. Overall results were very promising, especially for the tasks of source detection, article illustration, and geolocation. The automatic caption-generation task, however, demonstrated sensitivity to loosely related text and images.

Ajao et al. (2018) also fed mixed data inputs (text and images) to CNNs in order to detect fake news in political-debate speech, and they noted that except for the usual patterns in what would be considered misinformation, there also exists some hidden patterns in the words and images that can be captured with a set of latent features extracted via the multiple convolutional layers in the model. They put forward the TI-CNN (text and image information based convolutional neural network) model, whereby explicit and latent features can be projected into a unified feature space, with the TI-CNN able to be trained with both the text and image information simultaneously.

## Recent Developments

Deep neural networks have revolutionized the field of NLP. Furthermore, deep learning in NLP is undergoing an ‘ImageNet’ moment. In a paradigm shift, instead of using word embeddings as

initializations of the first layer of the networks, we are now moving to pretraining the entire models that capture hierarchical representations and bring us closer to solving complex language-understanding tasks. When the ImageNet challenge AlexNet (Krizhevsky et al., 2012) solution showed a dramatically improved performance of deep learning models compared to traditional competitors, it arguably spurred the whole deep learning research wave. Over the last 18 months, pretrained language models have blown out of the water previous state-of-the-art results across many NLP tasks. These advances can be characterized within the broader framework of transfer learning, where the weights learned in state-of-the-art models can be used to initialize models for different datasets, and this ‘fine-tuning’ achieves superior performance even with as little as one positive example per category (Ruder et al., 2019).

One of the assumptions of standard word embeddings like Word2Vec is that the meaning of the word is relatively stable across sentences. An alternative is to develop contextualized embeddings as part of the language models. Embeddings from language models (ELMo) (Peters et al., 2018), universal language model fine-tuning (ULMFiT) (Howard and Ruder, 2018), and generative pretraining transformer (OpenAI GPT) (Radford et al., 2018) were initial extremely successful pretrained language models.

More recently GPT2 (Radford et al. 2019) extended the previous GPT model and was used to generate realistic-sounding artificial text. Bullock and Luengo-Oroz (2019) used the pretrained GPT2 model to generate fake but natural-sounding speeches in the United Nations General Debate (see Baturu et al., 2017, for more details about the data and a substantive example). Bidirectional encoder representations from transformers (BERT) (Devlin et al., 2019) extended GPT through bi-directional training and dramatically improved performance on various metrics.

While BERT was the reigning champion for several months, it may have recently been overtaken by XLNet (Yang et al., 2019), which outperforms BERT on about 20 NLP tasks.

In parallel with the advances in transfer learning, we are also further understanding what we are learning with the deep neural networks. Liu et al. (2019) show that RNNs (and LSTMs in particular) pick up general linguistic properties, with the lowest layers representing morphology and being the most transferable between tasks, middle layers representing syntax, and the highest layers representing task-specific semantics. Large pretrained language models do not exhibit the same monotonic increase in task specificity, with the middle layers being the most transferrable. Tenney et al. (2019) focus on BERT and show that the model represents the steps of the traditional NLP pipeline, with the parts-of-speech tagging followed by parsing, named-entity recognition, semantic roles, and, finally, coreference. Furthermore, the model adjusts the pipeline dynamically, taking into account complex interactions between different levels of hierarchical information.

Detailed discussion of the above models is beyond the scope of this chapter. Instead, we want to emphasize the pace of development in NLP research, which is leveraging pretrained language models for downstream tasks. Instead of downloading pretrained word embeddings like Word2Vec or GloVe as discussed earlier in the chapter, we are now in a position to download pretrained language models and fine-tune them to a specific task.

## **Conclusion**

It is appealing to think of machine learning algorithms as objective, unbiased actors that are beyond the influence of human prejudices. It is also appealing to think of empirical research in political science that utilizes machine learning algorithms as being sufficiently removed from any potential bias. Unfortunately, this is rarely the case.

Algorithms are designed by humans and learn by observing patterns in the data that very often represent biased human behavior. It is no surprise that algorithms tend to adopt and, in some occasions, perpetuate and reinforce the experiences and predispositions of the humans that have constructed them and those of society as a whole; this is also known as *algorithmic bias*. Although machine learning has been transformative in many fields, it has received criticism in the areas of causal inference, algorithmic bias, and data privacy. This is forming into a distinct area of social science research, focusing on the lack of (suitable) training data, difficulties of data access and data sharing, data bias and data provenance, privacy preserving data usage, and inadequate tasks, tools and evaluation settings (Danks and London, 2017).

The quality of insights delivered by algorithms crucially depends on data quality and data provenance. In particular, in each case, we need to effectively query very distinct (heterogeneous) data sources before we can extract and transform them for input into the data models. Common aspects of data quality that may affect the robustness of insights include consistency, integrity, accuracy, and completeness. How image or textual data is pre-processed may affect how data is interpreted and may also lead to biases. For example, dataset biases in computer vision can lead to feature representation flaws where CNNs, despite high accuracy, learn from unreliable co-appearing contexts (Zhang et al., 2018).

The consequences of biased algorithms can be quite real and severe. In 2016, an investigative study by ProPublica (Angwin et al., 2016) provided evidence that a risk-assessment machine learning algorithm used by US courts wrongly flagged non-white defendants at almost twice the rate of white defendants. More recently, Wang and Kosinski (2018) showed how deep neural networks can outperform humans in detecting sexual orientation. Apart from the ethical

issues of the study, the ease of deployment of such ‘AI Gaydar’ raises issues of people’s privacy and safety.

The issues of algorithmic bias are also highlighted in the Wellcome Trust Report (Matthew Fenech et al., 2018) with a focus on how AI has been used for health research. The report identifies, among other ethical, social, and political challenges, issues around implications of algorithmic transparency and explainability on health, the difference between an algorithmic decision and a human decision, and what makes algorithms, and the entities that create them, trustworthy. The report highlights the importance of stakeholders across the public- and private-sector organizations collaborating in the development of AI technology, and it raises awareness of the need for AI to be regulated.

Such algorithmic-bias issues may seem to be removed from everyday political science research. However, various methodological approaches discussed earlier in this chapter are not bias free. Word embeddings have been shown to carry societal biases that are encoded in human language (Garg et al., 2018). These range from biased analogies (Bolukbasi et al., 2016; Manzini et al., 2019; Nissim et al., 2019) to bias in language ID (Blodgett and O’Connor, 2017), natural-language inference (Rudinger et al., 2017), coreference resolution (Rudinger et al., 2018), and automated essay scoring (Amorim et al., 2018).

There are corresponding efforts to reduce algorithmic bias in deep neural-network applications, for example through postprocessing (Bolukbasi et al., 2016) or directly modeling the problem (Zhao et al., 2018). However, the bias still remains encoded implicitly (Gonen and Goldberg, 2019), and transparency and awareness about the problem may be better as a research and deployment strategy (Caliskan et al., 2017; Dwork et al., 2012; Gonen and Goldberg, 2019).

There are legitimate concerns about algorithmic bias and discrimination, algorithmic accountability and transparency, and general ‘black box’ perception of deep neural-network models (Knight, 2017; Mayernik, 2017). In order to address these issues, scholars (Fiesler and Proferes, 2018; Mittelstadt et al., 2016; Olhede and Wolfe, 2018; Prates et al., 2018), AI technologists, international organizations (European Group on Ethics in Science and New Technologies (EGE), 2018), and national governments (House of Lords Select Committee, 2018) have been recently advocating for a more ‘ethical’ and ‘beneficial’ AI that will be programmed to have humans’ interests at heart and could never hurt anyone.

Kusner et al. (2017), for example, provide an ethical framework for machine decision-making, whereby a ‘decision is considered fair towards an individual if it is the same in both the actual world and a “counterfactual” world, where the individual would belong to a different demographic group’. In addition, it is vital to think about who is being excluded from AI systems and what is missing from the datasets that drive machine learning algorithms. Often, these blind spots tend to produce disparate impacts on vulnerable and marginalized groups. This leads to the invisibility of these communities and their needs because there are not enough feedback loops for individuals to give their input. While the collection of even more personal data might make algorithmic models better, it would also increase the threats to privacy.

Russell et al. (2015) present relevant questions to be considered: what are the power dynamics between different industry and research groups? Will the interests of the research community change with greater state funding? Will government intervention encourage AI research to become less transparent and accountable? What organizational principles and institutional mechanisms exist to best promote beneficial AI? What would international cooperation look like in the research, regulation, and use of AI? Will transnational efforts to



regulate AI fall to the same collective-action problems that have undermined global efforts to address climate change?

To ensure that future iterations of the ethical principles are adopted widely around the world, further research will be needed to investigate long-standing political questions such as collective action, power, and governance, as well as the global governance of AI, to name a few.

## References

Ajao, O., Bhowmik, D. and Zargari, S. (2018) Fake News Identification on Twitter with Hybrid CNN and RNN Models. In: Proceedings of the 9th International Conference on Social Media and Society – SMSociety, Copenhagen, Denmark, pp. 226–230. New York: ACM.

Aletras, N., Tsarapatsanis, D., Preoțiuc-Pietro, D. and Lampos, V. (2016) Predicting judicial decisions of the European Court of Human Rights: A natural language processing perspective. *PeerJ Computer Science* 2: e93.

Amorim, E., Cançado, M. and Veloso, A. (2018) Automated Essay Scoring in the Presence of Biased Ratings. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long Papers), New Orleans, United States: Association for Computational Linguistics, pp. 229–237. Available at: <https://doi.org/10.18653/v1/N18-1021> (accessed 17 December 2018).

Angwin, J., Larson, J., Mattu, S. and Kirchner, L. (2016) Machine Bias. *ProPublica*, 23 May, 2016. Available at: <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing> (accessed 17 December 2018).

Atanasova, P., Barron-Cedeno, A., Elsayed, T., Suwaileh, R., Zaghouani, W., Kyuchukov, S., Da San Martino, G. and Nakov, P. (2018) Overview of the CLEF-2018 CheckThat! Lab on Automatic Identification and Verification of Political Claims. Task 1: Check-Worthiness. arXiv:1808.05542 [cs]. Available at: <http://arxiv.org/abs/1808.05542> (accessed 19 December 2018).

Ba, J., Mnih, V. and Kavukcuoglu, K. (2014) Multiple Object Recognition with Visual Attention. arXiv:1412.7755 [cs]. Available at: <http://arxiv.org/abs/1412.7755> (accessed 19 December 2018).

Babenko, B., Hersh, J., Newhouse, D., Ramakrishnan, A. and Swartz, T. (2017) Poverty Mapping Using Convolutional Neural Networks Trained on High and Medium Resolution Satellite Images, With an Application in Mexico. arXiv:1711.06323 [cs, stat]. Available at: <http://arxiv.org/abs/1711.06323> (accessed 18 December 2018).

Baker, A. (2015) Race, paternalism, and foreign aid: Evidence from US public opinion. *American Political Science Review* 109(1): 93–109.

BakIr, G. (ed.) (2007) Predicting Structured Data. *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press.

Barberá, P. (2015) Birds of the same feather tweet together: Bayesian ideal point estimation using Twitter data. *Political Analysis* 23(1): 76–91.

Baturo, A., Dasandi, N. and Mikhaylov, S. J. (2017) Understanding state preferences with text as data: Introducing the un general debate corpus. *Research & Politics* 4(2): 1-9. Available at <https://journals.sagepub.com/doi/pdf/10.1177/2053168017712821> (accessed 19 December 2019)

Bengio, Y., Simard, P. and Frasconi, P. (1994) Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2): 157–166.

Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F. and Gauvin, J.-L. (2006) Neural Probabilistic Language Models. In: Holmes, D. E. and Jain, L. C. (eds), *Innovations in machine learning*. Berlin/Heidelberg: Springer-Verlag, pp. 137–186.

Benjamins, V. R., Selic, B, Casanovas, P., Breuker, J. and Gangemi, A. (2005) *Law and the Semantic Web: Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications*. Berlin Heidelberg: Springer.

Benoit, K. and Nulty, P. (2013) Classification methods for scaling latent political traits. In: Presentation at the Annual Meeting of the Midwest Political Science Association, Chicago, United States, pp. 11–13.

Bilbao-Jayo, A. and Almeida, A. (2018) Automatic political discourse analysis with multi-scale convolutional neural networks and contextual data. *International Journal of Distributed Sensor Networks* 14(11): 1-11.

Bishop, C. M. (2006) *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer.

Blodgett, S. L. & O'Connor, B. (2017) Racial Disparity in Natural Language Processing: A Case Study of Social Media African-American English. In: 2017 Workshop on Fairness, Accountability, and Transparency in Machine Learning, Nova Scotia, Canada. arXiv preprint arXiv:1707.00061: 1-4.

Blommaert, J. and Bulcaen, C. (2000) Critical discourse analysis. *Annual Review of Anthropology* 29, : 447–466.

Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2016) Enriching Word Vectors with Subword Information. arXiv:1607.04606 [cs]. Available at: <http://arxiv.org/abs/1607.04606> (accessed 19 December 2018).

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V. and Kalai, A. T. 2016. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In: Lee, D. D., von Luxburg, R. Garnett, M. Sugiyama, and Guyon, I, (eds). *Advances in Neural Information Processing Systems 29: 30th Annual Conference on Neural Information Processing Systems 2016: Barcelona, Spain, 5-10 December 2016*, Red Hook, NY: Curran Associates, Inc.: 4349–4357.

Bonica, A. (2014) Mapping the ideological marketplace. *American Journal of Political Science* 58(2): 367–386.

Brynjolfsson, E., Mitchell, T. and Rock, D. (2018) What Can Machines Learn, and What Does It Mean for Occupations and the Economy? In: *AEA Papers and Proceedings*, Nashville, TN: American Economic Association, 108: 43–47.

Bullock, J. and Luengo-Oroz, M. (2019) Automated Speech Generation from UN General Assembly Statements: Mapping Risks in AI Generated Texts. In: *The 2019 International Conference on Machine Learning AI for Social Good Workshop*, Long Beach, United States: 1-5. Available at <http://arxiv.org/abs/1906.01946v1> (accessed 15 October 2019).

Caliskan, A., Bryson, J. J. and Narayanan, A. (2017) Semantics derived automatically from language corpora contain human-like biases. *Science* 356(6334): 183–186.

Ceron, A., Curini, L., Iacus, S. M. and Porro, G. (2014) Every tweet counts? How sentiment analysis of social media can improve our knowledge of citizens' political preferences with an application to Italy and France. *New Media & Society* 16(2): 340–358.

Danks, D. and London, A. J. (2017) Algorithmic bias in autonomous systems. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, Melbourne Australia. Red Hook, NY: Curran Associates, Inc: 4691–4697.

de Vries, E., Schoonvelde, M. and Schumacher, G. (2018) No longer lost in translation: Evidence that Google Translate works for comparative bag-of-words text applications. *Political Analysis* 26(4): 417–430.

Deng, L. (2014) Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing* 7(3–4): 197–387.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol 1 (Long and Short papers)*, Minneapolis, Minnesota: Association for Computational Linguistics: 4171 – 4186.

Dwork, C., Hardt, M., Pitassi, T., Reingold, O. and Zemel, R. (2012) Fairness through awareness. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, Cambridge, Massachusetts: ACM. pp. 214–226.

European Commission (2019) A Definition of AI: Main Capabilities and Scientific Disciplines. *High-Level Expert Group on Artificial Intelligence*, 8 April. Available at <https://web.archive.org/web/20191014134019/https://ec.europa.eu/digital-single-market/en/news/definition-artificial-intelligence-main-capabilities-and-scientific-disciplines> (accessed 14 October 2019).

European Group on Ethics in Science and New Technologies (EGE) (2018) Statement on Artificial Intelligence, Robotics and ‘Autonomous’ Systems. EU report. Available at: [https://web.archive.org/web/20191014135156/http://ec.europa.eu/research/ege/pdf/ege\\_ai\\_statement\\_2018.pdf](https://web.archive.org/web/20191014135156/http://ec.europa.eu/research/ege/pdf/ege_ai_statement_2018.pdf) (accessed 14 October 2019).

Evans, M., McIntosh, W., Lin, J. and Cates, C. L. (2007) Recounting the courts? Applying automated content analysis to enhance empirical legal research. *Journal of Empirical Legal Studies* 4(4): 1007–1039.

Fairclough, N. (1989) *Language and Power*. London; New York: Longman.

Fairclough, N. (1992) Discourse and text: Linguistic and intertextual analysis within discourse analysis. *Discourse & Society* 3(2): 193–217.

Fenech, M., Strukelj, N. and Buston, O. (2018) *Ethical, Social and Political Challenges of Artificial Intelligence in Health*. London: Wellcome Trust and Future Advocacy. Available at: <https://wellcome.ac.uk/sites/default/files/ai-in-health-ethical-social-political-challenges.pdf> (accessed 21 December 2018).

Fiesler, C. and Proferes, N. (2018) ‘Participant’ Perceptions of Twitter Research Ethics. *Social Media + Society* 4(1): 1-14.

Firth, J. (1957) A synopsis of linguistic theory 1930–1955. In: *Studies in Linguistic Analysis*. Oxford: Philological Society.

Garg, N., Schiebinger, L., Jurafsky, D. and Zou, J. (2018) Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16): E3635–E3644.

Goldberg, D. E. and Holland, J. H. (1988) Genetic algorithms and machine learning. *Machine Learning* 3(2): 95–99.

Gonen, H. and Goldberg, Y. (2019) Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. arXiv preprint arXiv:1903.03862.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, Massachusetts London, England: MIT Press.

Gregor, K., Danihelka, I., Graves, A., Jimenez Rezende, D. and Wierstra, D. (2015) DRAW: A Recurrent Neural Network For Image Generation. arXiv:1502.04623 [cs]. Available at: <http://arxiv.org/abs/1502.04623> (accessed 19 December 2018).

Grimmer, J. and Stewart, B. M. (2013) Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis* 21(3): 267–297.

Helber, P., Gram-Hansen, B., Varatharajan, I., Azam, F., Coca-Castro, A., Kopackova, V. and Bilinski, P. (2018) Mapping Informal Settlements in Developing Countries with Multi-resolution, Multi-spectral Data. arXiv:1812.00812 [cs, stat]. Available at: <http://arxiv.org/abs/1812.00812> (accessed 18 December 2018).

Herzog, A. and Benoit, K. (2015) The most unkindest cuts: Speaker selection and expressed government dissent during economic crisis. *The Journal of Politics* 77(4): 1157–1175.

Hochreiter, S. and Schmidhuber, J. (1997) Long short-term memory. *Neural Computation* 9(8): 1735–80.

Holmes, D. E. and Jain, L. C. (eds) (2006) *Innovations in Machine Learning: Theory and Applications*. Studies in Fuzziness and Soft Computing 194. Berlin: Springer.

House of Lords Select Committee (2018) AI in the UK: ready, willing and able? *House of Lords* 36. Available at: <https://publications.parliament.uk/pa/ld201719/ldselect/ldai/100/100.pdf> (accessed 14 October 2019).

Howard, J. and Ruder, S. (2018) Universal Language Model Fine-tuning for Text Classification. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers)*, Melbourne, Australia: ACL, pp. 328–339.

Iyyer, M., Enns, P., Boyd-Graber, J. L. and Resnik, P. (2014) Political Ideology Detection Using Recursive Neural Networks. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, United States: ACL, pp. 1113–1122.

Jacovi, A., Shalom, O. S. and Goldberg, Y. (2018) Understanding Convolutional Neural Networks for Text Classification. arXiv:1809.08037 [cs]. Available at: <http://arxiv.org/abs/1809.08037> (accessed 19 December 2018).

Jean, N., Burke, M., Xie, M., Davis, W. M., Lobell, D. B. and Ermon, S. (2016) Combining satellite imagery and machine learning to predict poverty. *Science* 353(6301): 790–794.



Johnson, R. and Zhang, T. (2014) Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. arXiv:1412.1058 [cs, stat]. Available at: <http://arxiv.org/abs/1412.1058> (accessed 25 October 2018).

Jordan, M. I. (2019) Artificial intelligence – The revolution hasn't happened yet. Harvard Data Science Review (1). Available at <https://doi.org/10.1162/99608f92.f06c6e61> (accessed 14 October 2019).

Kalchbrenner, N., Grefenstette, E. and Blunsom, P. (2014) A Convolutional Neural Network for Modelling Sentences. arXiv:1404.2188 [cs]. Available at: <http://arxiv.org/abs/1404.2188> (accessed 25 October 2018).

Kim, Y. (2014) Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882 [cs]. Available at: <http://arxiv.org/abs/1408.5882> (accessed 25 October 2018).

Knight, W. (2017, April) The dark secret at the heart of AI: No one really knows how the most advanced algorithms do what they do-that could be a problem. MIT Technology Review 120(2). Available at: <https://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai/> (accessed 14 October 2019).

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K.Q. (eds). *Advances in Neural Information Processing Systems 25: Neural Information Processing Systems 2012*, Red Hook, NY: Curran Associates, Inc, pp. 1097–1105.

Kudugunta, S. and Ferrara, E. (2018) Deep neural networks for bot detection. Information Sciences 467: 312–322.

Kusner, M. J., Loftus, J. R., Russell, C. and Silva, R. (2017) Counterfactual Fairness. arXiv:1703.06856 [cs, stat]. Available at: <http://arxiv.org/abs/1703.06856> (accessed 18 December 2018).

Laclau, E. and Mouffe, C. (1985) *Hegemony and Socialist Strategy: Towards a Radical Democratic Politics*, 1st ed.: Radical Thinkers. London; New York: Verso.

Lafferty, J. D., McCallum, A. and Pereira, F. C. N (2001) Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: *Proceedings of the Eighteenth International Conference on Machine Learning*, San Francisco, United States: Morgan Kaufmann Publishers Inc., pp. 282–289.

Lauderdale, B. E. and Herzog, A. (2016) Measuring political positions from legislative speech. *Political Analysis* 24(3): 374–394.

Laver, M. (2014) Measuring policy positions in political space. *Annual Review of Political Science* 17(1): 207–223.

Laver, M., Benoit, K. and Garry, J. (2003) Extracting policy positions from political texts using words as data. *American Political Science Review* 97(2): 311–331.

Lehmann, P., Werner, K., Lewandowski, J., Matthieß, T., Merz, N., Regel, S. and Werner, A. (2018) *Manifesto Corpus*. Version: 2018-01. Berlin: WZB Berlin Social Science Center. Available at: <https://manifesto-project.wzb.eu> (accessed 14 October 2019)

Liu, N. F., Gardner, M., Belinkov, Y., Peters, M. and Smith, N. A. (2019) Linguistic Knowledge and Transferability of Contextual Representations. In: *NAACL 2019*, Minneapolis, United

States. arXiv preprint arXiv:1903.08855. Available at <https://arxiv.org/abs/1903.08855> (accessed 14 October 2019).

Makino, K., Takei, Y., Miyazaki, T. and Goto, J. (2018) Classification of Tweets about Reported Events using Neural Networks. In: Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text, Brussels, Belgium: Association for Computational Linguistics, pp. 153–163. Available at: <http://aclweb.org/anthology/W18-6121> (accessed 14 October 2019).

Manzini, T., Lim, Y. C., Tsvetkov, Y. and Black, A. W. (2019) Black is to criminal as Caucasian is to police: Detecting and removing multiclass bias in word embeddings. In: NAACL 2019, Minneapolis, United States, pp. 1-5. arXiv preprint arXiv:1904.04047. Available at <https://arxiv.org/abs/1904.04047> (accessed 14 October 2019).

Mayernik, M. S. (2017) Open data: Accountability and transparency. *Big Data & Society* 4(2), pp. 1-5. Available at <https://journals.sagepub.com/doi/pdf/10.1177/2053951717718853> (accessed 14 October 2019).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. and Dean, J. (2013) Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546 [cs, stat]. Available at: <http://arxiv.org/abs/1310.4546> (accessed 19 December 2018).

Mitchell, T. M. (1997) *Machine Learning*. New York: McGraw-Hill.

Mittelstadt, B. D., Allo, P., Taddeo, M., Wachter, S. and Floridi, L. (2016) The ethics of algorithms: Mapping the debate. *Big Data & Society* 3(2), pp. 1-21. Available at <https://doi.org/10.1177/2053951716679679> (accessed 14 October 2019).

Nallapati, R., Zhou, B., dos Santos, C. N., Gulcehre, C. and Xiang, B. (2016) Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond. arXiv:1602.06023 [cs]. Available at: <http://arxiv.org/abs/1602.06023> (accessed 25 October 2018).

Narayan, S., Cohen, S. B. and Lapata, M. (2018) Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. arXiv:1808.08745 [cs]. Available at: <http://arxiv.org/abs/1808.08745> (accessed 19 December 2018).

Nguyen, V.-A., Boyd-Graber, J., Resnik, P. and Miler, K. (2015) Tea Party in the House: A Hierarchical Ideal Point Topic Model and its Application to Republican Legislators in the 112th Congress. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (vol. 1: Long Papers), Beijing, China: Association for Computational Linguistics, pp. 1438–1448.

Nissim, M., van Noord, R. and van der Goot, R. (2019) Fair is Better than Sensational: Man is to Doctor as Woman is to Doctor. arXiv preprint arXiv:1905.09866. Available at <https://arxiv.org/abs/1905.09866> (accessed 14 October 2019).

Olah, C. (2015) Understanding LSTM Networks. Available at: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> (accessed 21 December 2018).

Olah, C., Mordvintsev, A. and Schubert, L. (2017) Feature Visualization. *Distill*. Available at <https://distill.pub/2017/feature-visualization> (accessed 14 October 2019).

Olhede, S. C. and Wolfe, P. J. (2018) The Growing Ubiquity of Algorithms in Society: Implications, Impacts and Innovations. In: Philosophical Transactions of the Royal Society,

Series A: Mathematical, Physical, and Engineering Sciences 376(2128), pp. 1 – 16. Available at <https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2017.0364> (accessed 14 October 2019).

Pennington, J., Socher, R. and Manning, C. D. (2014) Glove: Global Vectors for Word Representation. In: EMNLP, Doha, Qatar.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In: NAACL, New Orleans, United States.

Peterson, A. and Spirling, A. (2018) Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems. *Political Analysis* 26(1): 120–128.

Prates, M., Avelar, P. and Lamb, L. C. (2018) On Quantifying and Understanding the Role of Ethics in AI Research: A Historical Account of Flagship Conferences and Journals. arXiv:1809.08328 [cs]: 188–173. DOI: 10.29007/74gj.

Preoțiuc-Pietro, D., Liu, Y., Hopkins, D. and Ungar, L. (2017) Beyond Binary Labels: Political Ideology Prediction of Twitter Users. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (vol. 1: Long Papers), Vancouver, Canada, pp. 729–740.

Proksch, S.-O. and Slapin, J. B. (2009) How to avoid pitfalls in statistical analysis of political texts: The case of Germany. *German Politics* 18(3): 323–344.

Proksch, S.-O., Lowe, W. and Soroka, S. (2015) Multilingual sentiment analysis: A new approach to measuring conflict in parliamentary speeches. *Legislative Studies Quarterly* 44(1): 97–131.

Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. (2018) Improving language understanding by generative pre-training. OpenAI.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. (2019) Language models are unsupervised multitask learners. OpenAI Blog 1(8).

Ramisa, A., Yan, F., Moreno-Noguer, F. and Mikolajczyk, K. (2018) BreakingNews: Article annotation by image and text processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(5): 1072–1085.

Rao, A. and Spasojevic, N. (2016) Actionable and Political Text Classification using Word Embeddings and LSTM. Available at: <https://arxiv.org/abs/1607.02501> (accessed 29 November 2018).

Ruder, S., Peters, M., Swayamdipta, S. and Wolf T. (2019) Transfer Learning in Natural Language Processing. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 15–18.

Rudinger, R., May, C. and Van Durme, B. (2017) Social Bias in Elicited Natural Language Inferences. In: *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pp. 74–79.

Rudinger, R., Naradowsky, J., Leonard, B. and Van Durme, B. (2018) Gender Bias in Coreference Resolution. arXiv preprint arXiv:1804.09301.

Russell, S., Dewey, D. and Tegmark, M. (2015) Research priorities for robust and beneficial artificial intelligence. *AI Magazine* 36(4): 105–114.

Samothrakis, S. (2018) Viewpoint: Artificial Intelligence and Labour. arXiv:1803.06563 [cs]. Available at: <https://www.ijcai.org/proceedings/2018/0803.pdf> (accessed 12 December 2018).

Samuel, A. L. (1959) Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3(3): 210–229.

Sanders, J., Lisi, G. and Schonhardt-Bailey, C. (2017) Themes and topics in parliamentary oversight hearings: a new direction in textual data analysis. *Statistics, Politics and Policy* 8(2): 153–194.

Schlogl, L. and Sumner, A. (2018) The Rise of the Robot Reserve Army: Automation and the Future of Economic Development, Work, and Wages in Developing Countries. ID 3208816, SSRN Scholarly Paper, 2 July. Rochester, NY: Social Science Research Network. Available at: <https://papers.ssrn.com/abstract=3208816> (accessed 19 December 2018).

Singh, V., Varshney, A., Akhtar, S. S. Vijay, D. and Shrivastava, M. (2018) Aggression Detection on Social Media Text Using Deep Neural Networks. In: *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, Brussels, Belgium, pp. 43–50. Association for Computational Linguistics. Available at: <http://aclweb.org/anthology/W18-5106>.

Slapin, J. B. and Proksch, S.-O. (2008) A scaling model for estimating time-series party positions from texts. *American Journal of Political Science* 52(3): 705–722.

Spirling, A. and Rodriguez, P. L. (2019) Word Embeddings: What Works, What Doesn't, and How to Tell the Difference for Applied Research. NYU manuscript. <https://github.com/ArthurSpirling/EmbeddingsPaper>

Tenney, I., Das, D. and Pavlick, E. (2019) BERT Rediscovered the Classical NLP Pipeline. ACL 2019.

Uysal, A. K. (2016) An improved global feature selection scheme for text classification. *Expert Systems with Applications* 43: 82–92.

Wang, Y. and Kosinski, M. (2018) Deep neural networks are more accurate than humans at detecting sexual orientation from facial images. *Journal of Personality and Social Psychology*, 114(2): 246.

Yamshchikov, I. P. and Rezagholi, S. (2018) Elephants, Donkeys, and Colonel Blotto. In: *Proceedings of the 3rd International Conference on Complexity, Future Information Systems and Risk*, pp. 113–119.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. and Le, Q. V. (2019) XLNet: Generalized Autoregressive Pretraining for Language Understanding. arXiv 1906.08237.

Yu, B., Kaufmann, S. and Diermeier, D. (2008) Classifying party affiliation from political speech. *Journal of Information Technology & Politics* 5(1): 33–48.

Zhang, X., Zhao, J. and LeCun, Y. (2015) Character-level Convolutional Networks for Text Classification. arXiv:1509.01626 [cs]. Available at: <http://arxiv.org/abs/1509.01626> (accessed 25 October 2018).

Zhang, Y. and Wallace, B. (2015) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Available at: <https://arxiv.org/abs/1510.03820> (accessed 25 October 2018).



Zhang, Q. and Zhu, S. (2018) Visual interpretability for deep learning: A survey. *Frontiers of Information Technology and Electronic Engineering* 19(1): 27–39.

Zhang, Q., Wang, W. and Zhu, S. C. (2018) Examining CNN Representations with Respect to Dataset Bias. In: *32nd AAAI Conference on Artificial Intelligence*.

Zhao, J., Zhou, Y. Li, Z., Wang, W. and Chang, K.-W. 2018. Learning Gender-Neutral Word Embeddings. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium*, pp. 4847–4853, Association for Computational Linguistics.

CHAPTER

6

# Vector Semantics and Embeddings

荃者所以在鱼，得鱼而忘荃 Nets are for fish;  
Once you get the fish, you can forget the net.  
言者所以在意，得意而忘言 Words are for meaning;  
Once you get the meaning, you can forget the words  
庄子(Zhuangzi), Chapter 26

The asphalt that Los Angeles is famous for occurs mainly on its freeways. But in the middle of the city is another patch of asphalt, the La Brea tar pits, and this asphalt preserves millions of fossil bones from the last of the Ice Ages of the Pleistocene Epoch. One of these fossils is the *Smilodon*, or saber-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different

sabre-tooth tiger called *Thylacosmilus* lived in Argentina and other parts of South America. *Thylacosmilus* was a marsupial whereas *Smilodon* was a placental mammal, but *Thylacosmilus* had the same long upper canines and, like *Smilodon*, had a protective bone flange on the lower jaw. The similarity of these two mammals is one of many examples



of parallel or convergent evolution, in which particular contexts or environments lead to the evolution of very similar structures in different species (Gould, 1980).

The role of context is also important in the similarity of a less biological kind of organism: the word. Words that occur in *similar contexts* tend to have *similar meanings*. This link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis**. The hypothesis was first formulated in the 1950s by linguists like Joos (1950), Harris (1954), and Firth (1957), who noticed that words which are synonyms (like *oculist* and *eye-doctor*) tended to occur in the same environment (e.g., near words like *eye* or *examined*) with the amount of meaning difference between two words “corresponding roughly to the amount of difference in their environments” (Harris, 1954, 157).

distributional hypothesis

In this chapter we introduce **vector semantics**, which instantiates this linguistic hypothesis by learning representations of the meaning of words, called **embeddings**, directly from their distributions in texts. These representations are used in every natural language processing application that makes use of meaning, and the **static embeddings** we introduce here underlie the more powerful dynamic or **contextualized embeddings** like **BERT** that we will see in Chapter 10.

vector semantics embeddings

These word representations are also the first example in this book of **representation learning**, automatically learning useful representations of the input text. Finding such **self-supervised** ways to learn representations of the input, instead of creating representations by hand via **feature engineering**, is an important focus of NLP research (Bengio et al., 2013).

representation learning

## 6.1 Lexical Semantics

Let's begin by introducing some basic principles of word meaning. How should we represent the meaning of a word? In the n-gram models of Chapter 3, and in classical NLP applications, our only representation of a word is as a string of letters, or an index in a vocabulary list. This representation is not that different from a tradition in philosophy, perhaps you've seen it in introductory logic classes, in which the meaning of words is represented by just spelling the word with small capital letters; representing the meaning of "dog" as DOG, and "cat" as CAT.

Representing the meaning of a word by capitalizing it is a pretty unsatisfactory model. You might have seen a joke due originally to semanticist Barbara Partee (Carlson, 1977):

Q: What's the meaning of life?

A: LIFE'

Surely we can do better than this! After all, we'll want a model of word meaning to do all sorts of things for us. It should tell us that some words have similar meanings (*cat* is similar to *dog*), others are antonyms (*cold* is the opposite of *hot*), some have positive connotations (*happy*) while others have negative connotations (*sad*). It should represent the fact that the meanings of *buy*, *sell*, and *pay* offer differing perspectives on the same underlying purchasing event (If I buy something from you, you've probably sold it to me, and I likely paid you). More generally, a model of word meaning should allow us to draw inferences to address meaning-related tasks like question-answering or dialogue.

In this section we summarize some of these desiderata, drawing on results in the linguistic study of word meaning, which is called **lexical semantics**; we'll return to and expand on this list in Chapter 18 and Chapter 10.

**Lemmas and Senses** Let's start by looking at how one word (we'll choose *mouse*) might be defined in a dictionary (simplified from the online dictionary WordNet):

mouse (N)

1. any of numerous small rodents...
2. a hand-operated device that controls a cursor...

lexical  
semantics

lemma  
citation form

Here the form *mouse* is the **lemma**, also called the **citation form**. The form *mouse* would also be the lemma for the word *mice*; dictionaries don't have separate definitions for inflected forms like *mice*. Similarly *sing* is the lemma for *sing*, *sang*, *sung*. In many languages the infinitive form is used as the lemma for the verb, so Spanish *dormir* "to sleep" is the lemma for *duermes* "you sleep". The specific forms *sung* or *carpets* or *sing* or *duermes* are called **wordforms**.

wordform

As the example above shows, each lemma can have multiple meanings; the lemma *mouse* can refer to the rodent or the cursor control device. We call each of these aspects of the meaning of *mouse* a **word sense**. The fact that lemmas can be **polysemous** (have multiple senses) can make interpretation difficult (is someone who types "mouse info" into a search engine looking for a pet or a tool?). Chapter 18 will discuss the problem of polysemy, and introduce **word sense disambiguation**, the task of determining which sense of a word is being used in a particular context.

**Synonymy** One important component of word meaning is the relationship between word senses. For example when one word has a sense whose meaning is identical to a sense of another word, or nearly identical, we say the two senses of those two words are **synonyms**. Synonyms include such pairs as

synonym

*couch/sofa vomit/throw up filbert/hazelnut car/automobile*

A more formal definition of synonymy (between words rather than senses) is that two words are synonymous if they are substitutable for one another in any sentence without changing the *truth conditions* of the sentence, the situations in which the sentence would be true. We often say in this case that the two words have the same **propositional meaning**.

propositional  
meaning

While substitutions between some pairs of words like *car / automobile* or *water / H<sub>2</sub>O* are truth preserving, the words are still not identical in meaning. Indeed, probably no two words are absolutely identical in meaning. One of the fundamental tenets of semantics, called the **principle of contrast** (Girard 1718, Bréal 1897, Clark 1987), states that a difference in linguistic form is always associated with some difference in meaning. For example, the word *H<sub>2</sub>O* is used in scientific contexts and would be inappropriate in a hiking guide—*water* would be more appropriate— and this genre difference is part of the meaning of the word. In practice, the word *synonym* is therefore used to describe a relationship of approximate or rough synonymy.

principle of  
contrast

**Word Similarity** While words don't have many synonyms, most words do have lots of *similar* words. *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly similar words. In moving from synonymy to similarity, it will be useful to shift from talking about relations between word senses (like synonymy) to relations between words (like similarity). Dealing with words avoids having to commit to a particular representation of word senses, which will turn out to simplify our task.

similarity

The notion of word **similarity** is very useful in larger semantic tasks. Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are, a very important component of natural language understanding tasks like question answering, paraphrasing, and summarization. One way of getting values for word similarity is to ask humans to judge how similar one word is to another. A number of datasets have resulted from such experiments. For example the SimLex-999 dataset (Hill et al., 2015) gives values on a scale from 0 to 10, like the examples below, which range from near-synonyms (*vanish, disappear*) to pairs that scarcely seem to have anything in common (*hole, agreement*):

vanish	disappear	9.8
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

**Word Relatedness** The meaning of two words can be related in ways other than similarity. One such class of connections is called word **relatedness** (Budanitsky and Hirst, 2006), also traditionally called word **association** in psychology.

relatedness  
association

Consider the meanings of the words *coffee* and *cup*. Coffee is not similar to cup; they share practically no features (coffee is a plant or a beverage, while a cup is a manufactured object with a particular shape). But coffee and cup are clearly related; they are associated by co-participating in an everyday event (the event of drinking coffee out of a cup). Similarly *scalpel* and *surgeon* are not similar but are related eventively (a surgeon tends to make use of a scalpel).

semantic field

One common kind of relatedness between words is if they belong to the same **semantic field**. A semantic field is a set of words which cover a particular semantic domain and bear structured relations with each other. For example, words might be related by being in the semantic field of hospitals (*surgeon, scalpel, nurse, anesthetic, hospital*), restaurants (*waiter, menu, plate, food, chef*), or houses (*door, roof,*

**topic models** *kitchen, family, bed*). Semantic fields are also related to **topic models**, like **Latent Dirichlet Allocation, LDA**, which apply unsupervised learning on large sets of texts to induce sets of associated words from text. Semantic fields and topic models are very useful tools for discovering topical structure in documents.

In Chapter 18 we'll introduce more relations between senses like **hypernymy** or **IS-A**, **antonymy** (opposites) and **meronymy** (part-whole relations).

**semantic frame** **Semantic Frames and Roles** Closely related to semantic fields is the idea of a **semantic frame**. A semantic frame is a set of words that denote perspectives or participants in a particular type of event. A commercial transaction, for example, is a kind of event in which one entity trades money to another entity in return for some good or service, after which the good changes hands or perhaps the service is performed. This event can be encoded lexically by using verbs like *buy* (the event from the perspective of the buyer), *sell* (from the perspective of the seller), *pay* (focusing on the monetary aspect), or nouns like *buyer*. Frames have semantic roles (like *buyer, seller, goods, money*), and words in a sentence can take on these roles.

Knowing that *buy* and *sell* have this relation makes it possible for a system to know that a sentence like *Sam bought the book from Ling* could be paraphrased as *Ling sold the book to Sam*, and that Sam has the role of the *buyer* in the frame and Ling the *seller*. Being able to recognize such paraphrases is important for question answering, and can help in shifting perspective for machine translation.

**connotations** **Connotation** Finally, words have *affective meanings* or **connotations**. The word *connotation* has different meanings in different fields, but here we use it to mean the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations. For example some words have positive connotations (*happy*) while others have negative connotations (*sad*). Even words whose meanings are similar in other ways can vary in connotation; consider the difference in connotations between *fake, knockoff, forgery*, on the one hand, and *copy, replica, reproduction* on the other, or *innocent* (positive connotation) and *naive* (negative connotation). Some words describe positive evaluation (*great, love*) and others negative evaluation (*terrible, hate*). Positive or negative evaluation language is called

**sentiment** **sentiment**, as we saw in Chapter 4, and word sentiment plays a role in important tasks like sentiment analysis, stance detection, and applications of NLP to the language of politics and consumer reviews.

Early work on affective meaning (Osgood et al., 1957) found that words varied along three important dimensions of affective meaning:

**valence:** the pleasantness of the stimulus

**arousal:** the intensity of emotion provoked by the stimulus

**dominance:** the degree of control exerted by the stimulus

Thus words like *happy* or *satisfied* are high on valence, while *unhappy* or *annoyed* are low on valence. *Excited* is high on arousal, while *calm* is low on arousal. *Controlling* is high on dominance, while *awed* or *influenced* are low on dominance. Each word is thus represented by three numbers, corresponding to its value on each of the three dimensions:

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

Osgood et al. (1957) noticed that in using these 3 numbers to represent the meaning of a word, the model was representing each word as a point in a three-dimensional space, a vector whose three dimensions corresponded to the word's rating on the three scales. This revolutionary idea that word meaning could be represented as a point in space (e.g., that part of the meaning of *heartbreak* can be represented as the point [2.45, 5.65, 3.58]) was the first expression of the vector semantics models that we introduce next.

## 6.2 Vector Semantics

### vector semantics

**Vectors semantics** is the standard way to represent word meaning in NLP, helping us model many of the aspects of word meaning we saw in the previous section. The roots of the model lie in the 1950s when two big ideas converged: Osgood's (1957) idea mentioned above to use a point in three-dimensional space to represent the connotation of a word, and the proposal by linguists like Joos (1950), Harris (1954), and Firth (1957) to define the meaning of a word by its **distribution** in language use, meaning its neighboring words or grammatical environments. Their idea was that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.

For example, suppose you didn't know the meaning of the word *ongchoi* (a recent borrowing from Cantonese) but you see it in the following contexts:

- (6.1) Ongchoi is delicious sauteed with garlic.
- (6.2) Ongchoi is superb over rice.
- (6.3) ...ongchoi leaves with salty sauces...

And suppose that you had seen many of these context words in other contexts:

- (6.4) ...spinach sauteed with garlic over rice...
- (6.5) ...chard stems and leaves are delicious...
- (6.6) ...collard greens and other salty leafy greens

The fact that *ongchoi* occurs with words like *rice* and *garlic* and *delicious* and *salty*, as do words like *spinach*, *chard*, and *collard greens* might suggest that *ongchoi* is a leafy green similar to these other leafy greens.<sup>1</sup> We can do the same thing computationally by just counting words in the context of *ongchoi*.

### embeddings

The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived (in ways we'll see) from the distributions of word neighbors. Vectors for representing words are called **embeddings** (although the term is sometimes more strictly applied only to dense vectors like word2vec (Section 6.8), rather than sparse tf-idf or PPMI vectors (Section 6.3-Section 6.6)). The word "embedding" derives from its mathematical sense as a mapping from one space or structure to another, although the meaning has shifted; see the end of the chapter.

Fig. 6.1 shows a visualization of embeddings learned for sentiment analysis, showing the location of selected words projected down from 60-dimensional space into a two dimensional space. Notice the distinct regions containing positive words, negative words, and neutral function words.

<sup>1</sup> It's in fact *Ipomoea aquatica*, a relative of morning glory sometimes called *water spinach* in English.



**Figure 6.1** A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

The fine-grained model of word similarity of vector semantics offers enormous power to NLP applications. NLP applications like the sentiment classifiers of Chapter 4 or Chapter 5 depend on the same words appearing in the training and test sets. But by representing words as embeddings, classifiers can assign sentiment as long as it sees some words with *similar meanings*. And as we’ll see, vector semantic models can be learned automatically from text without supervision.

In this chapter we’ll introduce the two most commonly used models. In the **tf-idf** model, an important baseline, the meaning of a word is defined by a simple function of the counts of nearby words. We will see that this method results in very long vectors that are **sparse**, i.e. mostly zeros (since most words simply never occur in the context of others). We’ll introduce the **word2vec** model family for constructing short, **dense** vectors that have useful semantic properties. We’ll also introduce the **cosine**, the standard way to use embeddings to compute *semantic similarity*, between two words, two sentences, or two documents, an important tool in practical applications like question answering, summarization, or automatic essay grading.

## 6.3 Words and Vectors

“The most important attributes of a vector in 3-space are {Location, Location, Location}”  
 Randall Munroe, <https://xkcd.com/2358/>

Vector or distributional models of meaning are generally based on a **co-occurrence matrix**, a way of representing how often words co-occur. We’ll look at two popular matrices: the term-document matrix and the term-term matrix.

### 6.3.1 Vectors and documents

term-document  
matrix

In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents. Fig. 6.2 shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.

vector space  
model

The term-document matrix of Fig. 6.2 was first defined as part of the **vector space model** of information retrieval (Salton, 1971). In this model, a document is

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

**Figure 6.2** The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

represented as a count vector, a column in Fig. 6.3.

**vector** To review some basic linear algebra, a **vector** is, at heart, just a list or array of numbers. So *As You Like It* is represented as the list [1,114,36,20] (the first **column vector** in Fig. 6.3) and *Julius Caesar* is represented as the list [7,62,1,2] (the third column vector). A **vector space** is a collection of vectors, characterized by their **dimension**. In the example in Fig. 6.3, the document vectors are of dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality  $|V|$ , the vocabulary size.

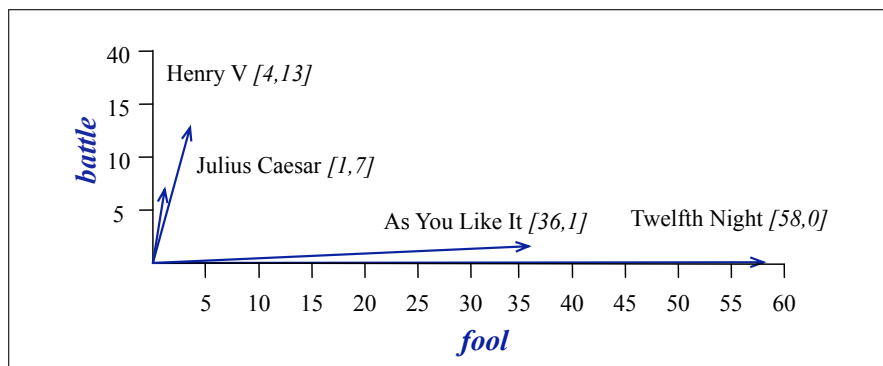
**vector space**  
**dimension**

The ordering of the numbers in a vector space indicates different meaningful dimensions on which documents vary. Thus the first dimension for both these vectors corresponds to the number of times the word *battle* occurs, and we can compare each dimension, noting for example that the vectors for *As You Like It* and *Twelfth Night* have similar values (1 and 0, respectively) for the first dimension.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

**Figure 6.3** The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

We can think of the vector for a document as a point in  $|V|$ -dimensional space; thus the documents in Fig. 6.3 are points in 4-dimensional space. Since 4-dimensional spaces are hard to visualize, Fig. 6.4 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.



**Figure 6.4** A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Term-document matrices were originally defined as a means of finding similar documents for the task of document **information retrieval**. Two documents that are



similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar. The vectors for the comedies *As You Like It* [1,114,36,20] and *Twelfth Night* [0,80,58,15] look a lot more like each other (more fools and wit than battles) than they look like *Julius Caesar* [7,62,1,2] or *Henry V* [13,89,4,3]. This is clear with the raw numbers; in the first dimension (battle) the comedies have low numbers and the others have high numbers, and we can see it visually in Fig. 6.4; we'll see very shortly how to quantify this intuition more formally.

A real term-document matrix, of course, wouldn't just have 4 rows and columns, let alone 2. More generally, the term-document matrix has  $|V|$  rows (one for each word type in the vocabulary) and  $D$  columns (one for each document in the collection); as we'll see, vocabulary sizes are generally in the tens of thousands, and the number of documents can be enormous (think about all the pages on the web).

information  
retrieval

**Information retrieval (IR)** is the task of finding the document  $d$  from the  $D$  documents in some collection that best matches a query  $q$ . For IR we'll therefore also represent a query by a vector, also of length  $|V|$ , and we'll need a way to compare two vectors to find how similar they are. (Doing IR will also require efficient ways to store and manipulate these vectors by making use of the convenient fact that these vectors are sparse, i.e., mostly zeros).

Later in the chapter we'll introduce some of the components of this vector comparison process: the tf-idf term weighting, and the cosine similarity metric.

### 6.3.2 Words as vectors: document dimensions

We've seen that documents can be represented as vectors in a vector space. But vector semantics can also be used to represent the meaning of *words*. We do this by associating each word with a word vector—a **row vector** rather than a column vector, hence with different dimensions, as shown in Fig. 6.5. The four dimensions of the vector for *fool*, [36,58,1,4], correspond to the four Shakespeare plays. Word counts in the same four dimensions are used to form the vectors for the other 3 words: *wit*, [20,15,2,3]; *battle*, [1,0,7,13]; and *good* [114,80,62,89].

row vector

	As You Like It	Twelfth Night	Julius Caesar	Henry V
<b>battle</b>	1	0	7	13
<b>good</b>	114	80	62	89
<b>fool</b>	36	58	1	4
<b>wit</b>	20	15	2	3

**Figure 6.5** The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

### 6.3.3 Words as vectors: word dimensions

An alternative to using the term-document matrix to represent words as vectors of document counts, is to use the **term-term matrix**, also called the **word-word matrix** or the **term-context matrix**, in which the columns are labeled by words rather than documents. This matrix is thus of dimensionality  $|V| \times |V|$  and each cell records

word-word  
matrix

the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, generally a window around the word, for example of 4 words to the left and 4 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a  $\pm 4$  word window around the row word. For example here is one example each of some words in their windows:

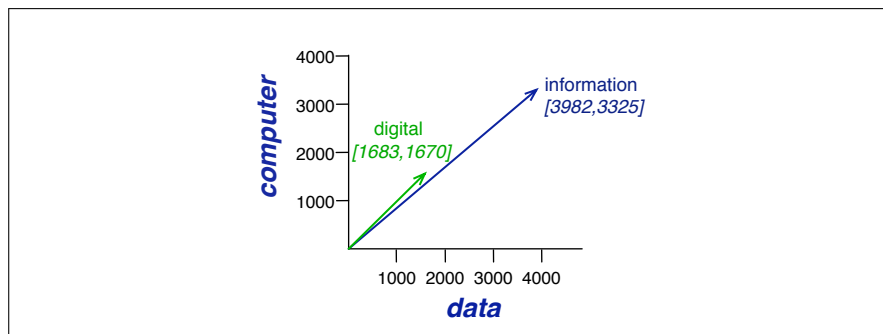
is traditionally followed by **cherry** pie, a traditional dessert  
 often mixed, such as **strawberry** rhubarb pie. Apple pie  
 computer peripherals and personal **digital** assistants. These devices usually  
 a computer. This includes **information** available on the internet

If we then take every occurrence of each word (say **strawberry**) and count the context words around it, we get a word-word co-occurrence matrix. Fig. 6.6 shows a simplified subset of the word-word co-occurrence matrix for these four words computed from the Wikipedia corpus (Davies, 2015).

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

**Figure 6.6** Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Note in Fig. 6.6 that the two words *cherry* and *strawberry* are more similar to each other (both *pie* and *sugar* tend to occur in their window) than they are to other words like *digital*; conversely, *digital* and *information* are more similar to each other than, say, to *strawberry*. Fig. 6.7 shows a spatial visualization.



**Figure 6.7** A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

Note that  $|V|$ , the length of the vector, is generally the size of the vocabulary, often between 10,000 and 50,000 words (using the most frequent words in the training corpus; keeping words after about the most frequent 50,000 or so is generally not helpful). Since most of these numbers are zero these are **sparse** vector representations; there are efficient algorithms for storing and computing with sparse matrices.

Now that we have some intuitions, let's move on to examine the details of computing word similarity. Afterwards we'll discuss methods for weighting cells.

## 6.4 Cosine for measuring similarity

To measure similarity between two target words  $v$  and  $w$ , we need a metric that takes two vectors (of the same dimensionality, either both with words as dimensions, hence of length  $|V|$ , or both with documents as dimensions as documents, of length  $|D|$ ) and gives a measure of their similarity. By far the most common similarity metric is the **cosine** of the angle between the vectors.

dot product  
inner product

The cosine—like most measures for vector similarity used in NLP—is based on the **dot product** operator from linear algebra, also called the **inner product**:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N \quad (6.7)$$

As we will see, most metrics for similarity between vectors are based on the dot product. The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong dissimilarity.

vector length

This raw dot product, however, has a problem as a similarity metric: it favors **long** vectors. The **vector length** is defined as

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2} \quad (6.8)$$

The dot product is higher if a vector is longer, with higher values in each dimension. More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them. The raw dot product thus will be higher for frequent words. But this is a problem; we'd like a similarity metric that tells us how similar two words are regardless of their frequency.

We modify the dot product to normalize for the vector length by dividing the dot product by the lengths of each of the two vectors. This **normalized dot product** turns out to be the same as the cosine of the angle between the two vectors, following from the definition of the dot product between two vectors  $\mathbf{a}$  and  $\mathbf{b}$ :

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}| |\mathbf{b}| \cos \theta \\ \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} &= \cos \theta \end{aligned} \quad (6.9)$$

cosine

The **cosine** similarity metric between two vectors  $\mathbf{v}$  and  $\mathbf{w}$  thus can be computed as:

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}| |\mathbf{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}} \quad (6.10)$$

unit vector

For some applications we pre-normalize each vector, by dividing it by its length, creating a **unit vector** of length 1. Thus we could compute a unit vector from  $\mathbf{a}$  by dividing it by  $|\mathbf{a}|$ . For unit vectors, the dot product is the same as the cosine.

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for orthogonal vectors, to -1 for vectors pointing in opposite directions. But since raw frequency values are non-negative, the cosine for these vectors ranges from 0–1.

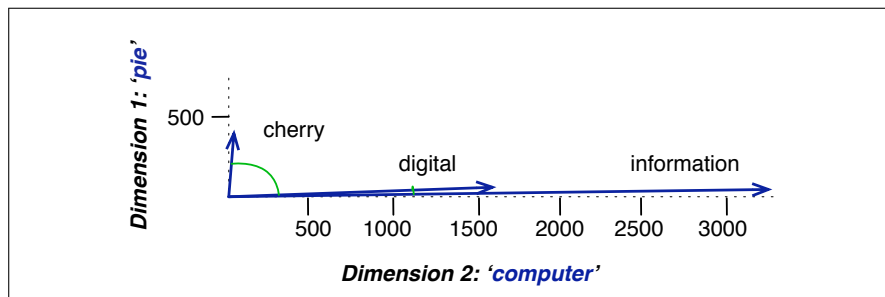
Let’s see how the cosine computes which of the words *cherry* or *digital* is closer in meaning to *information*, just using raw counts from the following shortened table:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry}, \text{information}) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

The model decides that *information* is way closer to *digital* than it is to *cherry*, a result that seems sensible. Fig. 6.8 shows a visualization.



**Figure 6.8** A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. Note that the angle between *digital* and *information* is smaller than the angle between *cherry* and *information*. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest ( $0^\circ$ ); the cosine of all other angles is less than 1.

## 6.5 TF-IDF: Weighing terms in the vector

The co-occurrence matrices above represent each cell by frequencies, either of words with documents (Fig. 6.5), or words with other words (Fig. 6.6). But raw frequency is not the best measure of association between words. Raw frequency is very skewed and not very discriminative. If we want to know what kinds of contexts are shared by *cherry* and *strawberry* but not by *digital* and *information*, we’re not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words and aren’t informative about any particular word. We saw this also in Fig. 6.3 for the Shakespeare corpus; the dimension for the word *good* is not very discriminative between plays; *good* is simply a frequent word and has roughly equivalent high frequencies in each of the plays.

It’s a bit of a paradox. Words that occur nearby frequently (maybe *pie* nearby *cherry*) are more important than words that only appear once or twice. Yet words

that are too frequent—ubiquitous, like *the* or *good*— are unimportant. How can we balance these two conflicting constraints?

There are two common solutions to this problem: in this section we'll describe the **tf-idf** algorithm, usually used when the dimensions are documents. In the next we introduce the **PPMI** algorithm (usually used when the dimensions are words).

The **tf-idf algorithm** (the '-' here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions:

term frequency

The first is the **term frequency** (Luhn, 1957): the frequency of the word  $t$  in the document  $d$ . We can just use the raw count as the term frequency:

$$\text{tf}_{t,d} = \text{count}(t,d) \quad (6.11)$$

More commonly we squash the raw frequency a bit, by using the  $\log_{10}$  of the frequency instead. The intuition is that a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document. Because we can't take the log of 0, we normally add 1 to the count:<sup>2</sup>

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d) + 1) \quad (6.12)$$

If we use log weighting, terms which occur 0 times in a document would have  $\text{tf} = \log_{10}(1) = 0$ , 10 times in a document  $\text{tf} = \log_{10}(11) = 1.4$ , 100 times  $\text{tf} = \log_{10}(101) = 2.004$ , 1000 times  $\text{tf} = 3.00044$ , and so on.

document frequency

The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful. The **document frequency**  $\text{df}_t$  of a term  $t$  is the number of documents it occurs in. Document frequency is not the same as the **collection frequency** of a term, which is the total number of times the word appears in the whole collection in any document. Consider in the collection of Shakespeare's 37 plays the two words *Romeo* and *action*. The words have identical collection frequencies (they both occur 113 times in all the plays) but very different document frequencies, since *Romeo* only occurs in a single play. If our goal is to find documents about the romantic tribulations of *Romeo*, the word *Romeo* should be highly weighted, but not *action*:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

We emphasize discriminative words like *Romeo* via the **inverse document frequency** or **idf** term weight (Sparck Jones, 1972). The idf is defined using the fraction  $N/\text{df}_t$ , where  $N$  is the total number of documents in the collection, and  $\text{df}_t$  is the number of documents in which term  $t$  occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. It's usually clear what counts as a document: in Shakespeare we would use a play; when processing a collection of encyclopedia articles like Wikipedia, the document is a Wikipedia page; in processing newspaper articles, the document is a single article. Occasionally your corpus might not have appropriate document divisions and you might need to break up the corpus into documents yourself for the purposes of computing idf.

<sup>2</sup> Or we can use this alternative:  $\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$

Because of the large number of documents in many collections, this measure too is usually squashed with a log function. The resulting definition for inverse document frequency (idf) is thus

$$\text{idf}_t = \log_{10} \left( \frac{N}{\text{df}_t} \right) \quad (6.13)$$

Here are some idf values for some words in the Shakespeare corpus, ranging from extremely informative words which occur in only one play like *Romeo*, to those that occur in a few like *salad* or *Falstaff*, to those which are very common like *fool* or so common as to be completely non-discriminative since they occur in all 37 plays like *good* or *sweet*.<sup>3</sup>

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

**tf-idf** The **tf-idf** weighted value  $w_{t,d}$  for word  $t$  in document  $d$  thus combines term frequency  $\text{tf}_{t,d}$  (defined either by Eq. 6.11 or by Eq. 6.12) with idf from Eq. 6.13:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \quad (6.14)$$

Fig. 6.9 applies tf-idf weighting to the Shakespeare term-document matrix in Fig. 6.2, using the tf equation Eq. 6.12. Note that the tf-idf values for the dimension corresponding to the word *good* have now all become 0; since this word appears in every document, the tf-idf algorithm leads it to be ignored. Similarly, the word *fool*, which appears in 36 out of the 37 plays, has a much lower weight.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
<b>battle</b>	0.074	0	0.22	0.28
<b>good</b>	0	0	0	0
<b>fool</b>	0.019	0.021	0.0036	0.0083
<b>wit</b>	0.049	0.044	0.018	0.022

**Figure 6.9** A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of  $\text{tf} = \log_{10}(20 + 1) = 1.322$  and  $\text{idf} = .037$ . Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

The tf-idf weighting is the way for weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing. It's also a great baseline, the simple thing to try first. We'll look at other weightings like PPMI (Positive Pointwise Mutual Information) in Section 6.6.

<sup>3</sup> *Sweet* was one of Shakespeare's favorite adjectives, a fact probably related to the increased use of sugar in European recipes around the turn of the 16th century (Jurafsky, 2014, p. 175).

## 6.6 Pointwise Mutual Information (PMI)

An alternative weighting function to tf-idf, PPMI (positive pointwise mutual information), is used for term-term-matrices, when the vector dimensions correspond to words rather than documents. PPMI draws on the intuition that the best way to weigh the association between two words is to ask how much **more** the two words co-occur in our corpus than we would have a priori expected them to appear by chance.

pointwise  
mutual  
information

**Pointwise mutual information** (Fano, 1961)<sup>4</sup> is one of the most important concepts in NLP. It is a measure of how often two events  $x$  and  $y$  occur, compared with what we would expect if they were independent:

$$I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} \quad (6.16)$$

The pointwise mutual information between a target word  $w$  and a context word  $c$  (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$\text{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)} \quad (6.17)$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated.

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring *less often* than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each  $10^{-6}$  occur together less often than chance, we would need to be certain that the probability of the two occurring together is significantly different than  $10^{-12}$ , and this kind of granularity would require an enormous corpus. Furthermore it's not clear whether it's even possible to evaluate such scores of 'unrelatedness' with human judgments. For this reason it is more common to use Positive PMI (called **PPMI**) which replaces all negative PMI values with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994)<sup>5</sup>:

PPMI

$$\text{PPMI}(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0\right) \quad (6.18)$$

More formally, let's assume we have a co-occurrence matrix  $F$  with  $W$  rows (words) and  $C$  columns (contexts), where  $f_{ij}$  gives the number of times word  $w_i$  occurs in

<sup>4</sup> PMI is based on the **mutual information** between two random variables  $X$  and  $Y$ , defined as:

$$I(X, Y) = \sum_x \sum_y P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)} \quad (6.15)$$

In a confusion of terminology, Fano used the phrase *mutual information* to refer to what we now call *pointwise mutual information* and the phrase *expectation of the mutual information* for what we now call *mutual information*

<sup>5</sup> Positive PMI also cleanly solves the problem of what to do with zero counts, using 0 to replace the  $-\infty$  from  $\log(0)$ .

context  $c_j$ . This can be turned into a PPMI matrix where  $ppmi_{ij}$  gives the PPMI value of word  $w_i$  with context  $c_j$  as follows:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad p_{i*} = \frac{\sum_{j=1}^C f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad p_{*j} = \frac{\sum_{i=1}^W f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad (6.19)$$

$$PPMI_{ij} = \max(\log_2 \frac{p_{ij}}{p_{i*} p_{*j}}, 0) \quad (6.20)$$

Let's see some PPMI calculations. We'll use Fig. 6.10, which repeats Fig. 6.6 plus all the count marginals, and let's pretend for ease of calculation that these are the only words/contexts that matter.

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

**Figure 6.10** Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other words/contexts matter.

Thus for example we could compute  $PPMI(w=\text{information},c=\text{data})$ , assuming we pretended that Fig. 6.6 encompassed all the relevant word contexts/dimensions, as follows:

$$\begin{aligned} P(w=\text{information},c=\text{data}) &= \frac{3982}{11716} = .3399 \\ P(w=\text{information}) &= \frac{7703}{11716} = .6575 \\ P(c=\text{data}) &= \frac{5673}{11716} = .4842 \\ ppmi(\text{information},\text{data}) &= \log_2(.3399/ (.6575 * .4842)) = .0944 \end{aligned}$$

Fig. 6.11 shows the joint probabilities computed from the counts in Fig. 6.10, and Fig. 6.12 shows the PPMI values. Not surprisingly, *cherry* and *strawberry* are highly associated with both *pie* and *sugar*, and *data* is mildly associated with *information*.

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

**Figure 6.11** Replacing the counts in Fig. 6.6 with joint probabilities, showing the marginals around the outside.

PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values. One way to reduce this bias toward low frequency



	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

**Figure 6.12** The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 6.11. Note that most of the 0 PPMI values are ones that had a negative PMI; for example  $\text{PMI}(\text{cherry}, \text{computer}) = -6.7$ , meaning that *cherry* and *computer* co-occur on Wikipedia less often than we would expect by chance, and with PPMI we replace negative values by zero.

events is to slightly change the computation for  $P(c)$ , using a different function  $P_\alpha(c)$  that raises the probability of the context word to the power of  $\alpha$ :

$$\text{PPMI}_\alpha(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P_\alpha(c)}, 0\right) \quad (6.21)$$

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}(c)^\alpha} \quad (6.22)$$

Levy et al. (2015) found that a setting of  $\alpha = 0.75$  improved performance of embeddings on a wide range of tasks (drawing on a similar weighting used for skip-grams described below in Eq. 6.32). This works because raising the count to  $\alpha = 0.75$  increases the probability assigned to rare contexts, and hence lowers their PMI ( $P_\alpha(c) > P(c)$  when  $c$  is rare).

Another possible solution is Laplace smoothing: Before computing PMI, a small constant  $k$  (values of 0.1-3 are common) is added to each of the counts, shrinking (discounting) all the non-zero values. The larger the  $k$ , the more the non-zero counts are discounted.

## 6.7 Applications of the tf-idf or PPMI vector models

In summary, the vector semantics model we've described so far represents a target word as a vector with dimensions corresponding either to the documents in a large collection (the term-document matrix) or to the counts of words in some neighboring window (the term-term matrix). The values in each dimension are counts, weighted by tf-idf (for term-document matrices) or PPMI (for term-term matrices), and the vectors are sparse (since most values are zero).

The model computes the similarity between two words  $x$  and  $y$  by taking the cosine of their tf-idf or PPMI vectors; high cosine, high similarity. This entire model is sometimes referred to as the **tf-idf** model or the **PPMI** model, after the weighting function.

The tf-idf model of meaning is often used for document functions like deciding if two documents are similar. We represent a document by taking the vectors of all the words in the document, and computing the **centroid** of all those vectors. The centroid is the multidimensional version of the mean; the centroid of a set of vectors is a single vector that has the minimum sum of squared distances to each of the vectors in the set. Given  $k$  word vectors  $w_1, w_2, \dots, w_k$ , the centroid **document vector**  $d$  is:

$$d = \frac{w_1 + w_2 + \dots + w_k}{k} \quad (6.23)$$

centroid

document  
vector

Given two documents, we can then compute their document vectors  $d_1$  and  $d_2$ , and estimate the similarity between the two documents by  $\cos(d_1, d_2)$ . Document similarity is also useful for all sorts of applications; information retrieval, plagiarism detection, news recommender systems, and even for digital humanities tasks like comparing different versions of a text to see which are similar to each other.

Either the PPMI model or the tf-idf model can be used to compute word similarity, for tasks like finding word paraphrases, tracking changes in word meaning, or automatically discovering meanings of words in different corpora. For example, we can find the 10 most similar words to any target word  $w$  by computing the cosines between  $w$  and each of the  $V - 1$  other words, sorting, and looking at the top 10.

## 6.8 Word2vec

In the previous sections we saw how to represent a word as a sparse, long vector with dimensions corresponding to words in the vocabulary or documents in a collection. We now introduce a more powerful word representation: **embeddings**, short dense vectors. Unlike the vectors we've seen so far, embeddings are **short**, with number of dimensions  $d$  ranging from 50-1000, rather than the much larger vocabulary size  $|V|$  or number of documents  $D$  we've seen. These  $d$  dimensions don't have a clear interpretation. And the vectors are **dense**: instead of vector entries being sparse, mostly-zero counts or functions of counts, the values will be real-valued numbers that can be negative.

It turns out that dense vectors work better in every NLP task than sparse vectors. While we don't completely understand all the reasons for this, we have some intuitions. Representing words as 300-dimensional dense vectors requires our classifiers to learn far fewer weights than if we represented words as 50,000-dimensional vectors, and the smaller parameter space possibly helps with generalization and avoiding overfitting. Dense vectors may also do a better job of capturing synonymy. For example, in a sparse vector representation, dimensions for synonyms like *car* and *automobile* dimension are distinct and unrelated; sparse vectors may thus fail to capture the similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor.

**skip-gram**  
**SGNS**  
**word2vec**

In this section we introduce one method for computing embeddings: **skip-gram with negative sampling**, sometimes called **SGNS**. The skip-gram algorithm is one of two algorithms in a software package called **word2vec**, and so sometimes the algorithm is loosely referred to as word2vec (Mikolov et al. 2013, Mikolov et al. 2013a). The word2vec methods are fast, efficient to train, and easily available online with code and pretrained embeddings. Word2vec embeddings are **static embeddings**, meaning that the method learns one fixed embedding for each word in the vocabulary. In Chapter 10 we'll introduce methods for learning dynamic **contextual embeddings** like the popular **BERT** or **ELMO** representations, in which the vector for each word is different in different contexts.

The intuition of word2vec is that instead of counting how often each word  $w$  occurs near, say, *apricot*, we'll instead train a classifier on a binary prediction task: "Is word  $w$  likely to show up near *apricot*?" We don't actually care about this prediction task; instead we'll take the learned classifier *weights* as the word embeddings.

The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier; a word  $c$  that occurs near the target word *apricot* acts as gold 'correct answer' to the question "Is word  $c$  likely to show

**static**  
**embeddings**



We model the probability that word  $c$  is a real context word for target word  $w$  as:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)} \quad (6.28)$$

The sigmoid function returns a number between 0 and 1, but to make it a probability we'll also need the total probability of the two possible events ( $c$  is a context word, and  $c$  isn't a context word) to sum to 1. We thus estimate the probability that word  $c$  is not a real context word for  $w$  as:

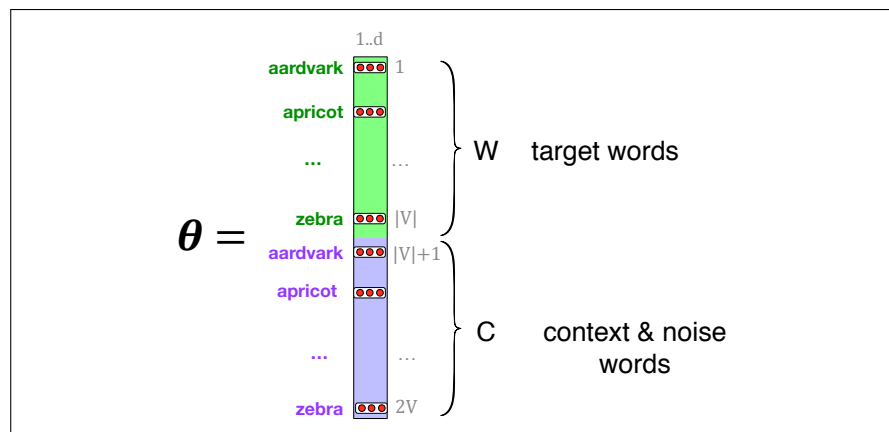
$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned} \quad (6.29)$$

Equation 6.28 gives us the probability for one word, but there are many context words in the window. Skip-gram makes the simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(-c_i \cdot w) \quad (6.30)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(-c_i \cdot w) \quad (6.31)$$

In summary, skip-gram trains a probabilistic classifier that, given a test target word  $w$  and its context window of  $L$  words  $c_{1:L}$ , assigns a probability based on how similar this context window is to the target word. The probability is based on applying the logistic (sigmoid) function to the dot product of the embeddings of the target word with each context word. To compute this probability, we just need embeddings for each target word and context word in the vocabulary.



**Figure 6.13** The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter  $\theta$  that the algorithm learns is thus a matrix of  $2|V|$  vectors, each of dimension  $d$ , formed by concatenating two matrices, the target embeddings  $W$  and the context+noise embeddings  $C$ .

Fig. 6.13 shows the intuition of the parameters we'll need. Skip-gram actually stores two embeddings for each word, one for the word as a target, and one for the

word considered as context. Thus the parameters we need to learn are two matrices  $W$  and  $C$ , each containing an embedding for every one of the  $|V|$  words in the vocabulary  $V$ .<sup>6</sup> Let's now turn to learning these embeddings (which is the real goal of training this classifier in the first place).

### 6.8.2 Learning skip-gram embeddings

Skip-gram learns embeddings by starting with random embedding vectors and then iteratively shifting the embedding of each word  $w$  to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby. Let's start by considering a single piece of training data:

... lemon, a [tablespoon of apricot jam, a] pinch ...  
                   c1          c2      w      c3          c4

This example has a target word  $w$  (apricot), and 4 context words in the  $L = \pm 2$  window, resulting in 4 positive training instances (on the left below):

positive examples +		negative examples -			
$w$	$c_{pos}$	$w$	$c_{neg}$	$w$	$c_{neg}$
apricot	tablespoon	apricot	aardvark	apricot	seven
apricot	of	apricot	my	apricot	forever
apricot	jam	apricot	where	apricot	dear
apricot	a	apricot	coaxial	apricot	if

For training a binary classifier we also need negative examples. In fact skip-gram with negative sampling (SGNS) uses more negative examples than positive examples (with the ratio between them set by a parameter  $k$ ). So for each of these  $(w, c_{pos})$  training instances we'll create  $k$  negative samples, each consisting of the target  $w$  plus a 'noise word'  $c_{neg}$ . A noise word is a random word from the lexicon, constrained not to be the target word  $w$ . The right above shows the setting where  $k = 2$ , so we'll have 2 negative examples in the negative training set – for each positive example  $w, c_{pos}$ .

The noise words are chosen according to their weighted unigram frequency  $p_\alpha(w)$ , where  $\alpha$  is a weight. If we were sampling according to unweighted frequency  $p(w)$ , it would mean that with unigram probability  $p(\text{"the"})$  we would choose the word *the* as a noise word, with unigram probability  $p(\text{"aardvark"})$  we would choose *aardvark*, and so on. But in practice it is common to set  $\alpha = .75$ , i.e. use the weighting  $p^{\frac{3}{4}}(w)$ :

$$P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha} \quad (6.32)$$

Setting  $\alpha = .75$  gives better performance because it gives rare noise words slightly higher probability: for rare words,  $P_\alpha(w) > P(w)$ . To illustrate this intuition, it might help to work out the probabilities for an example with two events,  $P(a) = .99$  and  $P(b) = .01$ :

$$\begin{aligned} P_\alpha(a) &= \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \\ P_\alpha(b) &= \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03 \end{aligned} \quad (6.33)$$

<sup>6</sup> In principle the target matrix and the context matrix could use different vocabularies, but we'll simplify by assuming one shared vocabulary  $V$ .

Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings to

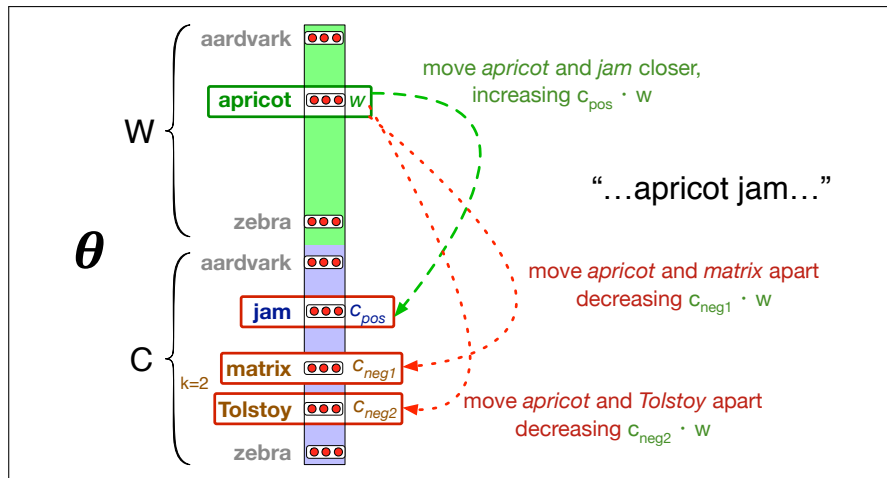
- Maximize the similarity of the target word, context word pairs  $(w, c_{pos})$  drawn from the positive examples
- Minimize the similarity of the  $(w, c_{neg})$  pairs from the negative examples.

If we consider one word/context pair  $(w, c_{pos})$  with its  $k$  noise words  $c_{neg_1} \dots c_{neg_k}$ , we can express these two goals as the following loss function  $L$  to be minimized (hence the  $-$ ); here the first term expresses that we want the classifier to assign the real context word  $c_{pos}$  a high probability of being a neighbor, and the second term expresses that we want to assign each of the noise words  $c_{neg_i}$  a high probability of being a non-neighbor, all multiplied because we assume independence:

$$\begin{aligned}
 L_{CE} &= -\log \left[ P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\
 &= - \left[ \log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\
 &= - \left[ \log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\
 &= - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \tag{6.34}
 \end{aligned}$$

That is, we want to maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the  $k$  negative sampled non-neighbor words.

We minimize this loss function using stochastic gradient descent. Fig. 6.14 shows the intuition of one step of learning.



**Figure 6.14** Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

To get the gradient, we need to take the derivative of Eq. 6.34 with respect to the different embeddings. It turns out the derivatives are the following (we leave the

proof as an exercise at the end of the chapter):

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w \quad (6.35)$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w \quad (6.36)$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i} \quad (6.37)$$

The update equations going from time step  $t$  to  $t + 1$  in stochastic gradient descent are thus:

$$c_{pos}^{t+1} = c_{pos}^t - \eta[\sigma(c_{pos}^t \cdot w) - 1]w \quad (6.38)$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta[\sigma(c_{neg}^t \cdot w)]w \quad (6.39)$$

$$w^{t+1} = w^t - \eta[\sigma(c_{pos} \cdot w^t) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)]c_{neg_i} \quad (6.40)$$

Just as in logistic regression, then, the learning algorithm starts with randomly initialized  $W$  and  $C$  matrices, and then walks through the training corpus using gradient descent to move  $W$  and  $C$  so as to maximize the objective in Eq. 6.34 by making the updates in (Eq. 6.39)-(Eq. 6.40).

target  
embedding  
context  
embedding

Recall that the skip-gram model learns **two** separate embeddings for each word  $i$ : the **target embedding**  $w_i$  and the **context embedding**  $c_i$ , stored in two matrices, the **target matrix**  $W$  and the **context matrix**  $C$ . It's common to just add them together, representing word  $i$  with the vector  $w_i + c_i$ . Alternatively we can throw away the  $C$  matrix and just represent each word  $i$  by the vector  $w_i$ .

As with the simple count-based methods like tf-idf, the context window size  $L$  affects the performance of skip-gram embeddings, and experiments often tune the parameter  $L$  on a devset.

### 6.8.3 Other kinds of static embeddings

fasttext

There are many kinds of static embeddings. An extension of word2vec, **fasttext** (Bojanowski et al., 2017), deals with unknown words and sparsity in languages with rich morphology, by using subword models. Each word in fasttext is represented as itself plus a bag of constituent  $n$ -grams, with special boundary symbols  $<$  and  $>$  added to each word. For example, with  $n = 3$  the word *where* would be represented by the sequence  $<where>$  plus the character  $n$ -grams:

$<wh, whe, her, ere, re>$

Then a skipgram embedding is learned for each constituent  $n$ -gram, and the word *where* is represented by the sum of all of the embeddings of its constituent  $n$ -grams. A fasttext open-source library, including pretrained embeddings for 157 languages, is available at <https://fasttext.cc>.

The most widely used static embedding model besides word2vec is GloVe (Pennington et al., 2014), short for Global Vectors, because the model is based on capturing global corpus statistics. GloVe is based on ratios of probabilities from the word-word co-occurrence matrix, combining the intuitions of count-based models like PPMI while also capturing the linear structures used by methods like word2vec.

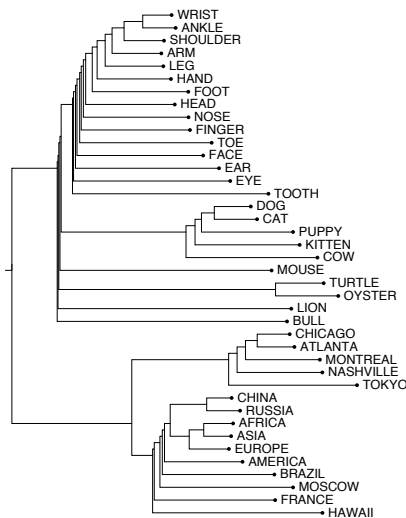
It turns out that dense embeddings like word2vec actually have an elegant mathematical relationships with sparse embeddings like PPMI, in which word2vec can be seen as implicitly optimizing a shifted version of a PPMI matrix (Levy and Goldberg, 2014c).

## 6.9 Visualizing Embeddings

“I see well in many dimensions as long as the dimensions are around two.”

The late economist Martin Shubik

Visualizing embeddings is an important goal in helping understand, apply, and improve these models of word meaning. But how can we visualize a (for example) 100-dimensional vector?



The simplest way to visualize the meaning of a word  $w$  embedded in a space is to list the most similar words to  $w$  by sorting the vectors for all words in the vocabulary by their cosine with the vector for  $w$ . For example the 7 closest words to *frog* using the GloVe embeddings are: *frogs*, *toad*, *litoria*, *leptodactylidae*, *rana*, *lizard*, and *eleutherodactylus* (Pennington et al., 2014).

Yet another visualization method is to use a clustering algorithm to show a hierarchical representation of which words are similar to others in the embedding space. The uncaptioned figure on the left uses hierarchical clustering of some embedding vectors for nouns as a visualization method (Rohde et al., 2006).

Probably the most common visualization method, however, is to project the 100 dimensions of a word down into 2 dimensions. Fig. 6.1 showed one such visualization, as does Fig. 6.16, using a projection method called t-SNE (van der Maaten and Hinton, 2008).

## 6.10 Semantic properties of embeddings

In this section we briefly summarize some of the semantic properties of embeddings that have been studied.

**Different types of similarity or association:** One parameter of vector semantic models that is relevant to both sparse tf-idf vectors and dense word2vec vectors is the size of the context window used to collect counts. This is generally between 1 and 10 words on each side of the target word (for a total context of 2-20 words).

The choice depends on the goals of the representation. Shorter context windows tend to lead to representations that are a bit more syntactic, since the information is coming from immediately nearby words. When the vectors are computed from short context windows, the most similar words to a target word  $w$  tend to be semantically similar words with the same parts of speech. When vectors are computed from long context windows, the highest cosine words to a target word  $w$  tend to be words that are topically related but not similar.



For example [Levy and Goldberg \(2014a\)](#) showed that using skip-gram with a window of  $\pm 2$ , the most similar words to the word *Hogwarts* (from the *Harry Potter* series) were names of other fictional schools: *Sunnydale* (from *Buffy the Vampire Slayer*) or *Evernight* (from a vampire series). With a window of  $\pm 5$ , the most similar words to *Hogwarts* were other words topically related to the *Harry Potter* series: *Dumbledore*, *Malfoy*, and *half-blood*.

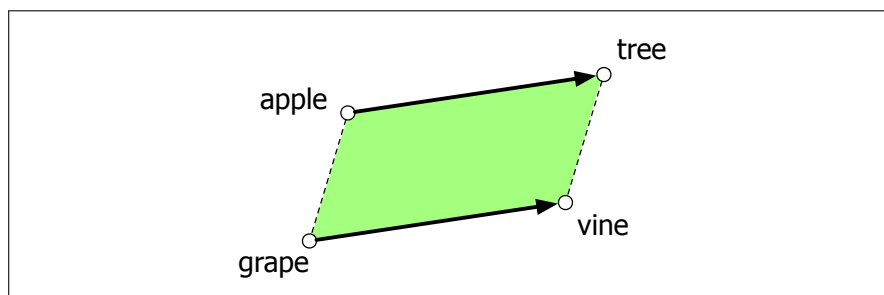
first-order  
co-occurrence

It's also often useful to distinguish two kinds of similarity or association between words ([Schütze and Pedersen, 1993](#)). Two words have **first-order co-occurrence** (sometimes called **syntagmatic association**) if they are typically nearby each other. Thus *wrote* is a first-order associate of *book* or *poem*. Two words have **second-order co-occurrence** (sometimes called **paradigmatic association**) if they have similar neighbors. Thus *wrote* is a second-order associate of words like *said* or *remarked*.

second-order  
co-occurrence

**Analogy/Relational Similarity:** Another semantic property of embeddings is their ability to capture relational meanings. In an important early vector space model of cognition, [Rumelhart and Abrahamson \(1973\)](#) proposed the **parallelogram model** for solving simple analogy problems of the form *a is to b as a\* is to what?*. In such problems, a system given a problem like *apple:tree::grape:?*, i.e., *apple is to tree as grape is to \_\_\_\_\_*, and must fill in the word *vine*. In the parallelogram model, illustrated in Fig. 6.15, the vector from the word *apple* to the word *tree* ( $= \text{apple} - \text{tree}$ ) is added to the vector for *grape* ( $\overrightarrow{\text{grape}}$ ); the nearest word to that point is returned.

parallelogram  
model



**Figure 6.15** The parallelogram model for analogy problems ([Rumelhart and Abrahamson, 1973](#)): the location of  $\overrightarrow{\text{vine}}$  can be found by subtracting  $\overrightarrow{\text{tree}}$  from  $\overrightarrow{\text{apple}}$  and adding  $\overrightarrow{\text{grape}}$ .

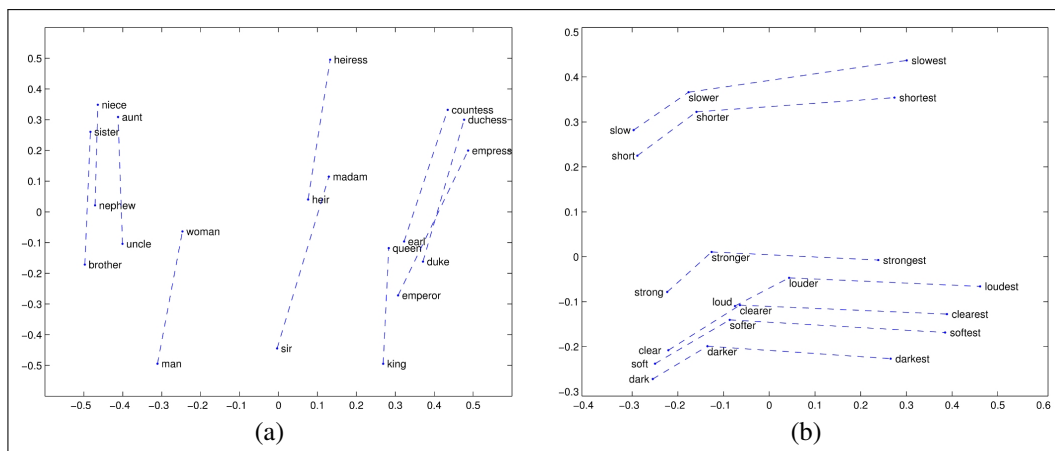
In early work with sparse embeddings, scholars showed that sparse vector models of meaning could solve such analogy problems ([Turney and Littman, 2005](#)), but the parallelogram method received more modern attention because of its success with word2vec or GloVe vectors ([Mikolov et al. 2013b](#), [Levy and Goldberg 2014b](#), [Pennington et al. 2014](#)). For example, the result of the expression  $\overrightarrow{(\text{king})} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$  is a vector close to  $\overrightarrow{\text{queen}}$ . Similarly,  $\text{Paris} - \text{France} + \text{Italy}$  results in a vector that is close to  $\overrightarrow{\text{Rome}}$ . The embedding model thus seems to be extracting representations of relations like MALE-FEMALE, or CAPITAL-CITY-OF, or even COMPARATIVE/SUPERLATIVE, as shown in Fig. 6.16 from GloVe.

For a  $a:b::a*:b^*$  problem, meaning the algorithm is given  $a$ ,  $b$ , and  $a^*$  and must find  $b^*$ , the parallelogram method is thus:

$$\hat{b}^* = \underset{x}{\operatorname{argmax}} \operatorname{distance}(x, a^* - a + b) \quad (6.41)$$

with the distance function defined either as cosine or as Euclidean distance.

There are some caveats. For example, the closest value returned by the parallelogram algorithm in word2vec or GloVe embedding spaces is usually not in fact  $b^*$  but one of the 3 input words or their morphological variants (i.e., *cherry:red ::*



**Figure 6.16** Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions. (a)  $\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}}$  is close to  $\vec{\text{queen}}$ . (b) offsets seem to capture comparative and superlative morphology (Pennington et al., 2014).

*potato:x* returns *potato* or *potatoes* instead of *brown*), so these must be explicitly excluded. Furthermore while embedding spaces perform well if the task involves frequent words, small distances, and certain relations (like relating countries with their capitals or verbs/nouns with their inflected forms), the parallelogram method with embeddings doesn't work as well for other relations (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a), and indeed Peterson et al. (2020) argue that the parallelogram method is in general too simple to model the human cognitive process of forming analogies of this kind.

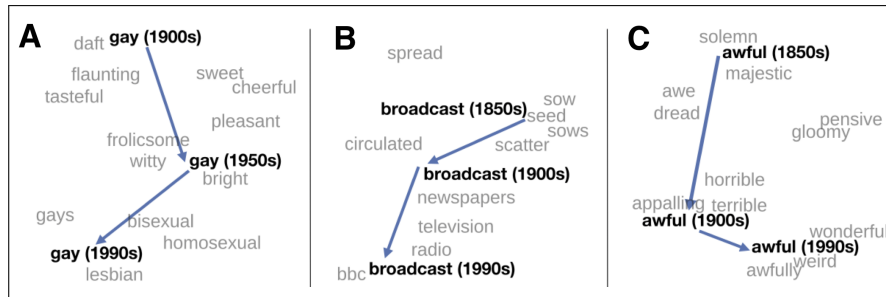
### 6.10.1 Embeddings and Historical Semantics

Embeddings can also be a useful tool for studying how meaning changes over time, by computing multiple embedding spaces, each from texts written in a particular time period. For example Fig. 6.17 shows a visualization of changes in meaning in English words over the last two centuries, computed by building separate embedding spaces for each decade from historical corpora like Google N-grams (Lin et al., 2012) and the Corpus of Historical American English (Davies, 2012).

## 6.11 Bias and Embeddings

In addition to their ability to learn word meaning from text, embeddings, alas, also reproduce the implicit biases and stereotypes that were latent in the text. As the prior section just showed, embeddings can roughly model relational similarity: 'queen' as the closest word to 'king' - 'man' + 'woman' implies the analogy *man:woman::king:queen*. But these same embedding analogies also exhibit gender stereotypes. For example Bolukbasi et al. (2016) find that the closest occupation to 'man' - 'computer programmer' + 'woman' in word2vec embeddings trained on news text is 'homemaker', and that the embeddings similarly suggest the analogy 'father' is to 'doctor' as 'mother' is to 'nurse'. This could result in what Crawford (2017) and Blodgett et al. (2020) call an **allocational harm**, when a system allocates resources (jobs or credit) unfairly to different groups. For example algorithms

allocational  
harm



**Figure 6.17** A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. The visualizations show the changes in the word *gay* from meanings related to “cheerful” or “frolicsome” to referring to homosexuality, the development of the modern “transmission” sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning “full of awe” to meaning “terrible or appalling” (Hamilton et al., 2016).

that use embeddings as part of a search for hiring potential programmers or doctors might thus incorrectly downweight documents with women’s names.

bias  
amplification

It turns out that embeddings don’t just reflect the statistics of their input, but also **amplify** bias; gendered terms become **more** gendered in embedding space than they were in the input text statistics (Zhao et al. 2017, Ethayarajh et al. 2019b, Jia et al. 2020), and biases are more exaggerated than in actual labor employment statistics (Garg et al., 2018).

representational  
harm

Embeddings also encode the implicit associations that are a property of human reasoning. The Implicit Association Test (Greenwald et al., 1998) measures people’s associations between concepts (like ‘flowers’ or ‘insects’) and attributes (like ‘pleasantness’ and ‘unpleasantness’) by measuring differences in the latency with which they label words in the various categories.<sup>7</sup> Using such methods, people in the United States have been shown to associate African-American names with unpleasant words (more than European-American names), male names more with mathematics and female names with the arts, and old people’s names with unpleasant words (Greenwald et al. 1998, Nosek et al. 2002a, Nosek et al. 2002b). Caliskan et al. (2017) replicated all these findings of implicit associations using GloVe vectors and cosine similarity instead of human latencies. For example African-American names like ‘Leroy’ and ‘Shaniqua’ had a higher GloVe cosine with unpleasant words while European-American names (‘Brad’, ‘Greg’, ‘Courtney’) had a higher cosine with pleasant words. These problems with embeddings are an example of a **representational harm** (Crawford 2017, Blodgett et al. 2020), which is a harm caused by a system demeaning or even ignoring some social groups. Any embedding-aware algorithm that made use of word sentiment could thus exacerbate bias against African Americans.

Recent research focuses on ways to try to remove these kinds of biases, for example by developing a transformation of the embedding space that removes gender stereotypes but preserves definitional gender (Bolukbasi et al. 2016, Zhao et al. 2017)

<sup>7</sup> Roughly speaking, if humans associate ‘flowers’ with ‘pleasantness’ and ‘insects’ with ‘unpleasantness’, when they are instructed to push a green button for ‘flowers’ (daisy, iris, lilac) and ‘pleasant words’ (love, laughter, pleasure) and a red button for ‘insects’ (flea, spider, mosquito) and ‘unpleasant words’ (abuse, hatred, ugly) they are faster than in an incongruous condition where they push a red button for ‘flowers’ and ‘unpleasant words’ and a green button for ‘insects’ and ‘pleasant words’.

or changing the training procedure (Zhao et al., 2018). However, although these sorts of **debiasing** may reduce bias in embeddings, they do not eliminate it (Gonen and Goldberg, 2019), and this remains an open problem.

Historical embeddings are also being used to measure biases in the past. Garg et al. (2018) used embeddings from historical texts to measure the association between embeddings for occupations and embeddings for names of various ethnicities or genders (for example the relative cosine similarity of women’s names versus men’s to occupation words like ‘librarian’ or ‘carpenter’) across the 20th century. They found that the cosines correlate with the empirical historical percentages of women or ethnic groups in those occupations. Historical embeddings also replicated old surveys of ethnic stereotypes; the tendency of experimental participants in 1933 to associate adjectives like ‘industrious’ or ‘superstitious’ with, e.g., Chinese ethnicity, correlates with the cosine between Chinese last names and those adjectives using embeddings trained on 1930s text. They also were able to document historical gender biases, such as the fact that embeddings for adjectives related to competence (‘smart’, ‘wise’, ‘thoughtful’, ‘resourceful’) had a higher cosine with male than female words, and showed that this bias has been slowly decreasing since 1960. We return in later chapters to this question about the role of bias in natural language processing.

## 6.12 Evaluating Vector Models

The most important evaluation metric for vector models is extrinsic evaluation on tasks, i.e., using vectors in an NLP task and seeing whether this improves performance over some other model.

Nonetheless it is useful to have intrinsic evaluations. The most common metric is to test their performance on **similarity**, computing the correlation between an algorithm’s word similarity scores and word similarity ratings assigned by humans. **WordSim-353** (Finkelstein et al., 2002) is a commonly used set of ratings from 0 to 10 for 353 noun pairs; for example (*plane*, *car*) had an average score of 5.77. **SimLex-999** (Hill et al., 2015) is a more difficult dataset that quantifies similarity (*cup*, *mug*) rather than relatedness (*cup*, *coffee*), and including both concrete and abstract adjective, noun and verb pairs. The **TOEFL dataset** is a set of 80 questions, each consisting of a target word with 4 additional word choices; the task is to choose which is the correct synonym, as in the example: *Levied is closest in meaning to: imposed, believed, requested, correlated* (Landauer and Dumais, 1997). All of these datasets present words without context.

Slightly more realistic are intrinsic similarity tasks that include context. The Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012) and the Word-in-Context (WiC) dataset (Pilehvar and Camacho-Collados, 2019) offer richer evaluation scenarios. SCWS gives human judgments on 2,003 pairs of words in their sentential context, while WiC gives target words in two sentential contexts that are either in the same or different senses; see Section ?? . The *semantic textual similarity* task (Agirre et al. 2012, Agirre et al. 2015) evaluates the performance of sentence-level similarity algorithms, consisting of a set of pairs of sentences, each pair with human-labeled similarity scores.

Another task used for evaluation is the analogy task, discussed on page 24, where the system has to solve problems of the form *a is to b as a\* is to b\**, given *a*, *b*, and *a\** and having to find *b\** (Turney and Littman, 2005). A number of sets of tuples have

been created for this task, (Mikolov et al. 2013, Mikolov et al. 2013b, Gladkova et al. 2016), covering morphology (*city:cities::child:children*), lexicographic relations (*leg:table::spout::teapot*) and encyclopedia relations (*Beijing:China::Dublin:Ireland*), some drawing from the SemEval-2012 Task 2 dataset of 79 different relations (Jurgens et al., 2012).

All embedding algorithms suffer from inherent variability. For example because of randomness in the initialization and the random negative sampling, algorithms like word2vec may produce different results even from the same dataset, and individual documents in a collection may strongly impact the resulting embeddings (Hellrich and Hahn 2016, Antoniak and Mimno 2018). When embeddings are used to study word associations in particular corpora, therefore, it is best practice to train multiple embeddings with bootstrap sampling over documents and average the results (Antoniak and Mimno, 2018).

## 6.13 Summary

- In vector semantics, a word is modeled as a vector—a point in high-dimensional space, also called an **embedding**. In this chapter we focus on **static embeddings**, in each each word is mapped to a fixed embedding.
- Vector semantic models fall into two classes: **sparse** and **dense**. In sparse models each dimension corresponds to a word in the vocabulary  $V$  and cells are functions of **co-occurrence counts**. The **term-document** matrix has a row for each word (**term**) in the vocabulary and a column for each document. The **word-context** or **term-term** matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary. Two sparse weightings are common: the **tf-idf** weighting which weights each cell by its **term frequency** and **inverse document frequency**, and **PPMI** (pointwise positive mutual information) most common for word-context matrices.
- Dense vector models have dimensionality 50–1000. **Word2vec** algorithms like **skip-gram** are a popular way to compute dense embeddings. Skip-gram trains a logistic regression classifier to compute the probability that two words are ‘likely to occur nearby in text’. This probability is computed from the dot product between the embeddings for the two words.
- Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.
- Other important embedding algorithms include **GloVe**, a method based on ratios of word co-occurrence probabilities.
- Whether using sparse or dense vectors, word and document similarities are computed by some function of the **dot product** between vectors. The cosine of two vectors—a normalized dot product—is the most popular such metric.

## Bibliographical and Historical Notes

The idea of vector semantics arose out of research in the 1950s in three distinct fields: linguistics, psychology, and computer science, each of which contributed a

fundamental aspect of the model.

The idea that meaning is related to the distribution of words in context was widespread in linguistic theory of the 1950s, among distributionalists like Zellig Harris, Martin Joos, and J. R. Firth, and semioticians like Thomas Sebeok. As [Joos \(1950\)](#) put it,

the linguist's "meaning" of a morpheme... is by definition the set of conditional probabilities of its occurrence in context with all other morphemes.

The idea that the meaning of a word might be modeled as a point in a multi-dimensional semantic space came from psychologists like Charles E. Osgood, who had been studying how people responded to the meaning of words by assigning values along scales like *happy/sad* or *hard/soft*. [Osgood et al. \(1957\)](#) proposed that the meaning of a word in general could be modeled as a point in a multidimensional Euclidean space, and that the similarity of meaning between two words could be modeled as the distance between these points in the space.

mechanical  
indexing

A final intellectual source in the 1950s and early 1960s was the field then called **mechanical indexing**, now known as **information retrieval**. In what became known as the **vector space model** for information retrieval ([Salton 1971](#), [Sparck Jones 1986](#)), researchers demonstrated new ways to define the meaning of words in terms of vectors ([Switzer, 1965](#)), and refined methods for word similarity based on measures of statistical association between words like mutual information ([Giuliano, 1965](#)) and idf ([Sparck Jones, 1972](#)), and showed that the meaning of documents could be represented in the same vector spaces used for words.

Some of the philosophical underpinning of the distributional way of thinking came from the late writings of the philosopher Wittgenstein, who was skeptical of the possibility of building a completely formal theory of meaning definitions for each word, suggesting instead that "the meaning of a word is its use in the language" ([Wittgenstein, 1953, PI 43](#)). That is, instead of using some logical language to define each word, or drawing on denotations or truth values, Wittgenstein's idea is that we should define a word by how it is used by people in speaking and understanding in their day-to-day interactions, thus prefiguring the movement toward embodied and experiential models in linguistics and NLP ([Glenberg and Robertson 2000](#), [Lake and Murphy 2020](#), [Bisk et al. 2020](#), [Bender and Koller 2020](#)).

semantic  
feature

More distantly related is the idea of defining words by a vector of discrete features, which has roots at least as far back as Descartes and Leibniz ([Wierzbicka 1992](#), [Wierzbicka 1996](#)). By the middle of the 20th century, beginning with the work of Hjelmslev ([Hjelmslev, 1969](#)) (originally 1943) and fleshed out in early models of generative grammar ([Katz and Fodor, 1963](#)), the idea arose of representing meaning with **semantic features**, symbols that represent some sort of primitive meaning. For example words like *hen*, *rooster*, or *chick*, have something in common (they all describe chickens) and something different (their age and sex), representable as:

*hen*    +female, +chicken, +adult  
*rooster* -female, +chicken, +adult  
*chick*    +chicken, -adult

The dimensions used by vector models of meaning to define words, however, are only abstractly related to this idea of a small fixed number of hand-built dimensions. Nonetheless, there has been some attempt to show that certain dimensions of embedding models do contribute some specific compositional aspect of meaning like these early semantic features.

The use of dense vectors to model word meaning, and indeed the term **embedding**, grew out of the **latent semantic indexing** (LSI) model ([Deerwester et al.](#),

1988) recast as **LSA (latent semantic analysis)** (Deerwester et al., 1990). In LSA **SVD singular value decomposition—SVD**—is applied to a term-document matrix (each cell weighted by log frequency and normalized by entropy), and then the first 300 dimensions are used as the LSA embedding. Singular Value Decomposition (SVD) is a method for finding the most important dimensions of a data set, those dimensions along which the data varies the most. LSA was then quickly widely applied: as a cognitive model Landauer and Dumais (1997), and for tasks like spell checking (Jones and Martin, 1997), language modeling (Bellegarda 1997, Cocco and Jurafsky 1998, Bellegarda 2000) morphology induction (Schone and Jurafsky 2000, Schone and Jurafsky 2001b), multiword expressions (MWEs) (Schone and Jurafsky, 2001a), and essay grading (Rehder et al., 1998). Related models were simultaneously developed and applied to word sense disambiguation by Schütze (1992). LSA also led to the earliest use of embeddings to represent words in a probabilistic classifier, in the logistic regression document router of Schütze et al. (1995). The idea of SVD on the term-term matrix (rather than the term-document matrix) as a model of meaning for NLP was proposed soon after LSA by Schütze (1992). Schütze applied the low-rank (97-dimensional) embeddings produced by SVD to the task of word sense disambiguation, analyzed the resulting semantic space, and also suggested possible techniques like dropping high-order dimensions. See Schütze (1997).

A number of alternative matrix models followed on from the early SVD work, including Probabilistic Latent Semantic Indexing (PLSI) (Hofmann, 1999), Latent Dirichlet Allocation (LDA) (Blei et al., 2003), and Non-negative Matrix Factorization (NMF) (Lee and Seung, 1999).

The LSA community seems to have first used the word “embedding” in Landauer et al. (1997), in a variant of its mathematical meaning as a mapping from one space or mathematical structure to another. In LSA, the word embedding seems to have described the mapping from the space of sparse count vectors to the latent space of SVD dense vectors. Although the word thus originally meant the mapping from one space to another, it has metonymically shifted to mean the resulting dense vector in the latent space. and it is in this sense that we currently use the word.

By the next decade, Bengio et al. (2003) and Bengio et al. (2006) showed that neural language models could also be used to develop embeddings as part of the task of word prediction. Collobert and Weston (2007), Collobert and Weston (2008), and Collobert et al. (2011) then demonstrated that embeddings could be used to represent word meanings for a number of NLP tasks. Turian et al. (2010) compared the value of different kinds of embeddings for different NLP tasks. Mikolov et al. (2011) showed that recurrent neural nets could be used as language models. The idea of simplifying the hidden layer of these neural net language models to create the skip-gram (and also CBOW) algorithms was proposed by Mikolov et al. (2013). The negative sampling training algorithm was proposed in Mikolov et al. (2013a). There are numerous surveys of static embeddings and their parameterizations (Bullinaria and Levy 2007, Bullinaria and Levy 2012, Lapesa and Evert 2014, Kiela and Clark 2014, Levy et al. 2015).

See Manning et al. (2008) for a deeper understanding of the role of vectors in information retrieval, including how to compare queries with documents, more details on tf-idf, and issues of scaling to very large datasets. See Kim (2019) for a clear and comprehensive tutorial on word2vec. Cruse (2004) is a useful introductory linguistic text on lexical semantics.

## Exercises



- Agirre, E., Banea, C., Cardie, C., Cer, D., Diab, M., Gonzalez-Agirre, A., Guo, W., Lopez-Gazpio, I., Maritxalar, M., Mihalcea, R., Rigau, G., Uria, L., and Wiebe, J. (2015). 2015 SemEval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability. *SemEval-15*.
- Agirre, E., Diab, M., Cer, D., and Gonzalez-Agirre, A. (2012). Semeval-2012 task 6: A pilot on semantic textual similarity. *SemEval-12*.
- Antoniak, M. and Mimno, D. (2018). Evaluating the stability of embedding-based word similarities. *TACL 6*, 107–119.
- Bellegarda, J. R. (1997). A latent semantic analysis framework for large-span language modeling. *EUROSPEECH*.
- Bellegarda, J. R. (2000). Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE 89*(8), 1279–1296.
- Bender, E. M. and Koller, A. (2020). Climbing towards NLU: On meaning, form, and understanding in the age of data. *ACL*.
- Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*(8), 1798–1828.
- Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research 3*(Feb), 1137–1155.
- Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., and Gauvain, J.-L. (2006). Neural probabilistic language models. *Innovations in Machine Learning*, 137–186. Springer.
- Bisk, Y., Holtzman, A., Thomason, J., Andreas, J., Bengio, Y., Chai, J., Lapata, M., Lazaridou, A., May, J., Nisnevich, A., Pinto, N., and Turian, J. (2020). Experience grounds language.. arXiv preprint arXiv:2004.10151.
- Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *JMLR 3*(5), 993–1022.
- Blodgett, S. L., Barocas, S., Daumé III, H., and Wallach, H. (2020). Language (technology) is power: A critical survey of “bias” in NLP. *ACL*.
- Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL 5*, 135–146.
- Bolukbasi, T., Chang, K.-W., Zou, J., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. *NeurIPS*.
- Bréal, M. (1897). *Essai de Sémantique: Science des significations*. Hachette.
- Budanitsky, A. and Hirst, G. (2006). Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics 32*(1), 13–47.
- Bullinaria, J. A. and Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods 39*(3), 510–526.
- Bullinaria, J. A. and Levy, J. P. (2012). Extracting semantic representations from word co-occurrence statistics: stoplists, stemming, and SVD. *Behavior research methods 44*(3), 890–907.
- Caliskan, A., Bryson, J. J., and Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. *Science 356*(6334), 183–186.
- Carlson, G. N. (1977). *Reference to kinds in English*. Ph.D. thesis, University of Massachusetts, Amherst. Forward.
- Church, K. W. and Hanks, P. (1989). Word association norms, mutual information, and lexicography. *ACL*.
- Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics 16*(1), 22–29.
- Clark, E. (1987). The principle of contrast: A constraint on language acquisition. MacWhinney, B. (Ed.), *Mechanisms of language acquisition*, 1–33. LEA.
- Coccaro, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. *ICSLP*.
- Collobert, R. and Weston, J. (2007). Fast semantic extraction using a novel neural network architecture. *ACL*.
- Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML*.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *JMLR 12*, 2493–2537.
- Crawford, K. (2017). The trouble with bias. Keynote at NeurIPS.
- Cruse, D. A. (2004). *Meaning in Language: an Introduction to Semantics and Pragmatics*. Oxford University Press. Second edition.
- Dagan, I., Marcus, S., and Markovitch, S. (1993). Contextual word similarity and estimation from sparse data. *ACL*.
- Davies, M. (2012). Expanding horizons in historical linguistics with the 400-million word Corpus of Historical American English. *Corpora 7*(2), 121–157.
- Davies, M. (2015). The Wikipedia Corpus: 4.6 million articles, 1.9 billion words. Adapted from Wikipedia. <https://www.english-corpora.org/wiki/>.
- Deerwester, S. C., Dumais, S. T., Furnas, G. W., Harshman, R. A., Landauer, T. K., Lochbaum, K. E., and Streeter, L. (1988). Computer information retrieval using latent semantic structure: US Patent 4,839,853..
- Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantics analysis. *JASIS 41*(6), 391–407.
- Ethayarajh, K., Duvenaud, D., and Hirst, G. (2019a). Towards understanding linear word analogies. *ACL*.
- Ethayarajh, K., Duvenaud, D., and Hirst, G. (2019b). Understanding undesirable word embedding associations. *ACL*.
- Fano, R. M. (1961). *Transmission of Information: A Statistical Theory of Communications*. MIT Press.
- Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems 20*(1), 116—131.
- Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. *Studies in Linguistic Analysis*. Philological Society. Reprinted in Palmer, F. (ed.) 1968. Selected Papers of J. R. Firth. Longman, Harlow.

- Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences* 115(16), E3635–E3644.
- Girard, G. (1718). *La justesse de la langue françoise: ou les différentes significations des mots qui passent pour synonymes*. Laurent d’Houry, Paris.
- Giuliano, V. E. (1965). The interpretation of word associations. Stevens, M. E., Giuliano, V. E., and Heilprin, L. B. (Eds.), *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings*. Washington, D.C., USA, March 17, 1964. <https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf>.
- Gladkova, A., Drozd, A., and Matsuoka, S. (2016). Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn’t. *NAACL Student Research Workshop*.
- Glenberg, A. M. and Robertson, D. A. (2000). Symbol grounding and meaning: A comparison of high-dimensional and embodied theories of meaning. *Journal of memory and language* 43(3), 379–401.
- Gonen, H. and Goldberg, Y. (2019). Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. *NAACL HLT*.
- Gould, S. J. (1980). *The Panda’s Thumb*. Penguin Group.
- Greenwald, A. G., McGhee, D. E., and Schwartz, J. L. K. (1998). Measuring individual differences in implicit cognition: the implicit association test. *Journal of personality and social psychology* 74(6), 1464–1480.
- Hamilton, W. L., Leskovec, J., and Jurafsky, D. (2016). Diachronic word embeddings reveal statistical laws of semantic change. *ACL*.
- Harris, Z. S. (1954). Distributional structure. *Word* 10, 146–162. Reprinted in J. Fodor and J. Katz, *The Structure of Language*, Prentice Hall, 1964 and in Z. S. Harris, *Papers in Structural and Transformational Linguistics*, Reidel, 1970, 775–794.
- Hellrich, J. and Hahn, U. (2016). Bad company—Neighborhoods in neural embedding spaces considered harmful. *COLING*.
- Hill, F., Reichart, R., and Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics* 41(4), 665–695.
- Hjelmslev, L. (1969). *Prologomena to a Theory of Language*. University of Wisconsin Press. Translated by Francis J. Whitfield; original Danish edition 1943.
- Hofmann, T. (1999). Probabilistic latent semantic indexing. *SIGIR-99*.
- Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. *ACL*.
- Jia, S., Meng, T., Zhao, J., and Chang, K.-W. (2020). Mitigating gender bias amplification in distribution by posterior regularization. *ACL*.
- Jones, M. P. and Martin, J. H. (1997). Contextual spelling correction using latent semantic analysis. *ANLP*.
- Joos, M. (1950). Description of language design. *JASA* 22, 701–708.
- Jurafsky, D. (2014). *The Language of Food*. W. W. Norton, New York.
- Jurgens, D., Mohammad, S. M., Turney, P., and Holyoak, K. (2012). SemEval-2012 task 2: Measuring degrees of relational similarity. *\*SEM 2012*.
- Katz, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language* 39, 170–210.
- Kiela, D. and Clark, S. (2014). A systematic study of semantic vector space model parameters. *EACL 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*.
- Kim, E. (2019). Optimize computational efficiency of skip-gram with negative sampling. [https://aegis4048.github.io/optimize\\_computational\\_efficiency\\_of\\_skip-gram\\_with\\_negative\\_sampling](https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling).
- Lake, B. M. and Murphy, G. L. (2020). Word meaning in minds and machines..
- Landauer, T. K. and Dumais, S. T. (1997). A solution to Plato’s problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review* 104, 211–240.
- Landauer, T. K., Laham, D., Rehder, B., and Schreiner, M. E. (1997). How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans. *COGSCI*.
- Lapesa, G. and Evert, S. (2014). A large scale evaluation of distributional semantic models: Parameters, interactions and model selection. *TACL* 2, 531–545.
- Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature* 401(6755), 788–791.
- Levy, O. and Goldberg, Y. (2014a). Dependency-based word embeddings. *ACL*.
- Levy, O. and Goldberg, Y. (2014b). Linguistic regularities in sparse and explicit word representations. *CoNLL*.
- Levy, O. and Goldberg, Y. (2014c). Neural word embedding as implicit matrix factorization. *NeurIPS*.
- Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *TACL* 3, 211–225.
- Li, J., Chen, X., Hovy, E. H., and Jurafsky, D. (2015). Visualizing and understanding neural models in NLP. *NAACL HLT*.
- Lin, Y., Michel, J.-B., Lieberman Aiden, E., Orwant, J., Brockman, W., and Petrov, S. (2012). Syntactic annotations for the google books ngram corpus. *ACL*.
- Linzen, T. (2016). Issues in evaluating semantic spaces using word analogies. *1st Workshop on Evaluating Vector-Space Representations for NLP*.
- Luhn, H. P. (1957). A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development* 1(4), 309–317.
- Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge.
- Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. *ICLR 2013*.
- Mikolov, T., Kombrink, S., Burget, L., Černocký, J. H., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. *ICASSP*.

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. *NeurIPS*.
- Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. *NAACL HLT*.
- Niwa, Y. and Nitta, Y. (1994). Co-occurrence vectors from corpora vs. distance vectors from dictionaries. *ACL*.
- Nosek, B. A., Banaji, M. R., and Greenwald, A. G. (2002a). Harvesting implicit group attitudes and beliefs from a demonstration web site. *Group Dynamics: Theory, Research, and Practice* 6(1), 101.
- Nosek, B. A., Banaji, M. R., and Greenwald, A. G. (2002b). Math=male, me=female, therefore math $\neq$  me. *Journal of personality and social psychology* 83(1), 44.
- Osgood, C. E., Suci, G. J., and Tannenbaum, P. H. (1957). *The Measurement of Meaning*. University of Illinois Press.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *EMNLP*.
- Peterson, J. C., Chen, D., and Griffiths, T. L. (2020). Parallelograms revisited: Exploring the limitations of vector space models for simple analogies. *Cognition* 205.
- Pilehvar, M. T. and Camacho-Collados, J. (2019). WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. *NAACL HLT*.
- Rehder, B., Schreiner, M. E., Wolfe, M. B. W., Laham, D., Landauer, T. K., and Kintsch, W. (1998). Using Latent Semantic Analysis to assess knowledge: Some technical considerations. *Discourse Processes* 25(2-3), 337–354.
- Rohde, D. L. T., Gonnerman, L. M., and Plaut, D. C. (2006). An improved model of semantic similarity based on lexical co-occurrence. *CACM* 8, 627–633.
- Rumelhart, D. E. and Abrahamson, A. A. (1973). A model for analogical reasoning. *Cognitive Psychology* 5(1), 1–28.
- Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice Hall.
- Schone, P. and Jurafsky, D. (2000). Knowledge-free induction of morphology using latent semantic analysis. *CoNLL*.
- Schone, P. and Jurafsky, D. (2001a). Is knowledge-free induction of multiword unit dictionary headwords a solved problem?. *EMNLP*.
- Schone, P. and Jurafsky, D. (2001b). Knowledge-free induction of inflectional morphologies. *NAACL*.
- Schütze, H. (1992). Dimensions of meaning. *Proceedings of Supercomputing '92*. IEEE Press.
- Schütze, H. (1997). *Ambiguity Resolution in Language Learning – Computational and Cognitive Models*. CSLI, Stanford, CA.
- Schütze, H., Hull, D. A., and Pedersen, J. (1995). A comparison of classifiers and document representations for the routing problem. *SIGIR-95*.
- Schütze, H. and Pedersen, J. (1993). A vector model for syntagmatic and paradigmatic relatedness. *9th Annual Conference of the UW Centre for the New OED and Text Research*.
- Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28(1), 11–21.
- Sparck Jones, K. (1986). *Synonymy and Semantic Classification*. Edinburgh University Press, Edinburgh. Republication of 1964 PhD Thesis.
- Switzer, P. (1965). Vector images in document retrieval. Stevens, M. E., Giuliano, V. E., and Heilprin, L. B. (Eds.), *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*. <https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf>.
- Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. *ACL*.
- Turney, P. D. and Littman, M. L. (2005). Corpus-based learning of analogies and semantic relations. *Machine Learning* 60(1-3), 251–278.
- van der Maaten, L. and Hinton, G. E. (2008). Visualizing high-dimensional data using t-sne. *JMLR* 9, 2579–2605.
- Wierzbicka, A. (1992). *Semantics, Culture, and Cognition: University Human Concepts in Culture-Specific Configurations*. Oxford University Press.
- Wierzbicka, A. (1996). *Semantics: Primes and Universals*. Oxford University Press.
- Wittgenstein, L. (1953). *Philosophical Investigations. (Translated by Anscombe, G.E.M.)*. Blackwell.
- Zhao, J., Wang, T., Yatskar, M., Ordonez, V., and Chang, K.-W. (2017). Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *EMNLP*.
- Zhao, J., Zhou, Y., Li, Z., Wang, W., and Chang, K.-W. (2018). Learning gender-neutral word embeddings. *EMNLP*.

# How Can We Accelerate Progress Towards Human-like Linguistic Generalization?

Tal Linzen

Department of Cognitive Science  
Johns Hopkins University  
tal.linzen@jhu.edu

## Abstract

This position paper describes and critiques the Pretraining-Agnostic Identically Distributed (PAID) evaluation paradigm, which has become a central tool for measuring progress in natural language understanding. This paradigm consists of three stages: (1) pre-training of a word prediction model on a corpus of arbitrary size; (2) fine-tuning (transfer learning) on a training set representing a classification task; (3) evaluation on a test set drawn from the same distribution as that training set. This paradigm favors simple, low-bias architectures, which, first, can be scaled to process vast amounts of data, and second, can capture the fine-grained statistical properties of a particular data set, regardless of whether those properties are likely to generalize to examples of the task outside the data set. This contrasts with humans, who learn language from several orders of magnitude less data than the systems favored by this evaluation paradigm, and generalize to new tasks in a consistent way. We advocate for supplementing or replacing PAID with paradigms that reward architectures that generalize as quickly and robustly as humans.

## 1 Introduction

The special session of the 2020 Annual Meeting of Association for Computational Linguistics invites us to take stock of the progress made in the field in the last few years. There is no question that we have made significant progress in a range of applications: current machine translation systems for high-resource languages, for example, are undeniably better than those we had a decade ago. This opinion piece will focus on a different question: are we making progress towards the classic goal of mimicking human linguistic abilities in machines—towards a model that acquires language as efficiently as humans, and generalizes it as humans do to new structures and contexts (“tasks”)?

I will argue that an evaluation paradigm that has rapidly established itself as one of the main tools for measuring progress in the field—a paradigm I will term, for want of a catchier name, Pretraining-Agnostic Identically Distributed evaluation (PAID)—encourages progress in a direction that is at best orthogonal to the goal of human-like generalization. Because it does not consider sample efficiency, this approach rewards models that can be trained on massive amounts of data, several orders of magnitude more than a human can expect to be exposed to. And because benchmark scores are computed on test sets drawn from the same distribution as their respective training sets, this paradigm favors models that excel in capturing the statistical patterns of particular data sets over models that generalize as a human would.

## 2 Human-like Generalization

Humans learn language from much more limited exposure than most contemporary NLP systems. An analysis of recordings taken in the environment of the child of an MIT professor between the ages of 9 and 24 months found that the child heard or produced approximately eight million words over this 15-month period (Roy et al., 2015). Children in lower socioeconomic status families in Western societies receive significantly less linguistic input than that (around 3 million words per year, Hart and Risley 1995); even more strikingly, members of the Tsimane community in Bolivia spend about 15 times less time per hour speaking to their children than do highly educated American families (Cristia et al., 2019). If NLP systems were as sample-efficient as Tsimane children, far fewer languages would be considered “low-resource languages”.

Despite the limited amount of exposure to their language, humans generalize their linguistic knowl-

edge in a consistent way to structures that are infrequent or non-existent in corpora (Sprouse et al., 2013), and quickly learn to do new things with language (what we sometimes refer to in NLP as “tasks”). As I discuss below, this is not the case for current deep learning systems: when tested on cases sampled from a distribution that differs from the one they were trained on, their behavior is unpredictable and inconsistent with that of humans (Jia and Liang, 2017; McCoy et al., 2019b), and they require extensive instruction on each new task (Yogatama et al., 2019). Humans’ rapid and consistent generalization abilities rely on powerful inductive biases, which likely arise from a combination of innate building blocks and experience with diverse learning problems (Lake et al., 2017).

Systems that generalize like humans would be useful not only for NLP, but also for the scientific study of human language acquisition and processing (Keller, 2010; Dupoux, 2018). But, as I will argue in the next two sections, it is unclear whether our dominant evaluation paradigms are getting us closer to this goal.

### 3 Pretraining-Agnostic Evaluation

Over the last two years, deep learning systems have obtained rapidly increasing scores on language understanding benchmarks such as GLUE (Wang et al., 2019b) or SuperGLUE (Wang et al., 2019a). These benchmarks aggregate multiple supervised classification tasks—such as sentiment analysis, linguistic acceptability judgments, or entailment detection—and collate the scores obtained on those tasks into a leaderboard, with a single headline score for each model averaging its scores on each individual task. For each of these classification tasks, a data set that was generated by a particular process, often involving crowdsourcing, is randomly split into two: a training set, which the system is allowed to observe, and a held-out test set, on which it is evaluated.

A standard recipe has emerged for achieving high scores on such benchmarks. A neural network—typically, one based on the transformer architecture (Vaswani et al., 2017)—is pretrained on a denoising objective, such as filling in one or more blanks in a vast number of sentences. This network is then fine-tuned (performs transfer learning) on the benchmark’s supervised tasks, each of which include a much smaller number of training examples than the pretraining corpus

(Howard and Ruder, 2018; Peters et al., 2018). The T5 model (Raffel et al., 2019)—the system that boasted the highest score on SuperGLUE at the time of writing—achieved an average accuracy of 88.9% on this benchmark, slightly lower than that of untrained human annotators (89.8%), and more than 20 percentage points higher than the score obtained just a few months earlier by BERT (Devlin et al., 2019; Wang et al., 2019a). This jump in accuracy does not reflect significant modeling innovations: both BERT and T5 are transformers trained on similar objectives that differ primarily in their scale.

When ranking systems, leaderboards such as SuperGLUE do not take into account the amount of pretraining data provided to each model. Pretraining corpora are not standardized, and the amount of pretraining data is not always easy to discern from the papers reporting on such systems. Here is my attempt to reconstruct the recent evolution of pretraining corpus sizes.<sup>1</sup> BERT, uploaded to arXiv in October 2018, was trained on 3.3 billion words; XLNet (Yang et al., June 2019), was trained on 78 GB of text, or approximately 13 billion words; RoBERTa (Liu et al., July 2019) was trained on 160 GB of text, or around 28 billion words; and T5 (Raffel et al., October 2019) was trained on 750 GB of text, or approximately 130 billion words.

When we rely on a single leaderboard to compare systems trained on corpora with such a large range of sizes, we are not comparing architectures, but rather interactions of architectures, corpus sizes, and computational resources available for training. While this may be a useful comparison for an engineer who seeks to plug an existing trained model into a larger pipeline, this approach is unlikely to advance us towards the goal advocated in this article. The 130 billion word corpus that T5 was trained on is much larger than the corpus that a human can expect to be exposed to before adulthood (fewer than 100 million words, see Section 2). But a leaderboard that evaluates only bottom-line transfer learning accuracy inherently disadvantages a sample-efficient model pretrained on a few dozen million words compared to a model such as T5. For all we know, it is possible that architectures

---

<sup>1</sup>Corpus sizes reported in massive-corpus pretraining papers are often specified in gigabytes, or number of model-specific subword units, instead of measures such as number of words that are easier to compare across articles. My estimates are based on an average English word length of 4.7 characters and a space or punctuation mark after each word.

rewarded by PAID, such as massive transformers, *only* work well when given an amount of data that is orders of magnitude greater than that available to humans. If that is the case, our exploration of the space of possible models could be going in a direction that is orthogonal to the one that might lead us to models that can imitate humans’ sample efficiency (one example of such direction is neural networks with explicit symbolic structure, which are harder to scale up, but perform well on smaller data sets: [Kuncoro et al. 2018](#); [Wilcox et al. 2019](#)).

#### 4 Identically Distributed Training Set and Test Set

The remaining two letters of the PAID acronym refer to the practice of evaluating success on classification tasks using training and test set generated using the same process. Typically, a single data set is collected and is randomly split into a training portion and test portion. While this may seem reasonable from a machine learning perspective, it has become clear that this form of evaluation obscures possible mismatches between the generalizations that we as humans believe a system performing the task should acquire, and the generalizations that the system in fact extracts from the data.

Consider, for example, crowdsourced natural language inference (NLI) data sets, in which workers are asked to generate a sentence that contradicts the prompt shown to them ([Bowman et al., 2015](#)). One strategy that crowdworkers adopt when generating a contradiction is to simply negate the prompt, for example by inserting the word *not*. This strategy is often effective: *the man is sleeping* contradicts *the man is not sleeping*. Conversely, it is much less likely that the worker would use the word *not* when asked to generate a sentence that is entailed by the prompt. Taken together, such worker choices lead to a strong correlation between the presence of the word *not* in the hypothesis and the label CONTRADICTION. It would be surprising if low-bias learners such as neural networks did not notice such a correlation, and indeed they do, leading them to respond CONTRADICTION with high probability any time the hypothesis contains a negation word ([Gururangan et al., 2018](#); [Poliak et al., 2018](#)). Of course, relying on the presence of the word *not* is not a generally valid inference strategy; for example, *the man is awake* entails, rather than contradicts, *the man is not sleeping*.

Numerous generalization issues of this sort have

been documented, for NLI and for other tasks. In the syntactic domain, [McCoy et al. \(2019b\)](#) showed that BERT fine-tuned on the crowdsourced MultiNLI data set ([Williams et al., 2018](#)) achieves high accuracy on the MultiNLI test set, but shows very little sensitivity to word order when tested on constructed examples that require an analysis of the structure of the sentence; for example, this model is likely to conclude that *the detective followed the suspect* entails *the suspect followed the detective*.

In short, the models, unable to discern the intentions of the data set’s designers, happily recapitulate any statistical patterns they find in the training data. With a random training/test split, any correlation observed in the training set will hold approximately for the test set, and a system that learned it could achieve high test set accuracy. And indeed, we have models that excel in the PAID paradigm, even exceeding the performance of human annotators on the test portion of the corpus used for fine-tuning ([Nangia and Bowman, 2019](#)), but, when tested on controlled examples, make mistakes that a human would rarely make.<sup>2</sup>

The generalizations that a statistical model extracts from the data are always the result of the interaction between the model’s inductive biases and the statistical properties of the data set. In the case of BERT’s insensitivity to word order in NLI, the model does not seem to have a strong inductive bias one way or another; its sensitivity to word order varies widely depending on the weight initialization of the fine-tuning classifier and the order of the fine-tuning examples ([McCoy et al., 2019a](#)), and its syntactic behavior in the inference task can be made to be more consistent with human intuitions if the training set is augmented to include a larger number of examples illustrating the importance of word order ([Min et al., 2020](#)). While BERT is capable of learning to use syntax for inference given a sufficiently strong signal, then, it prefers to use other heuristics, if possible. This contrasts with human-like generalization in this task, which would likely start from the assumption that any language understanding task should recruit our

---

<sup>2</sup>Comparisons between human annotators and transformers are arguably unfair: before observing the test set, the models receive hundreds of thousands of examples of the output of the data-generating process. This contrasts with humans annotators, who need to perform the task based on their general language understanding skills. It would be an entertaining though somewhat cruel experiment to repeat the comparison after matching the amount of exposure that humans and pre-trained transformers receive to the quirks of the data set.

knowledge of syntax: it would most likely be difficult to convince humans to *ignore* syntax when understanding a sentence, as BERT does.

## 5 The Generalization Leaderboard

What is the way forward? My goal is not to argue that there is no value to the leaderboard approach, where a single number or a small set of numbers can be used to quickly compare models. Despite the drawbacks of this approach—in particular, its tendency to obscure the fine-grained strengths and weaknesses of particular models, as I discuss below—hill climbing on a metric can enable a productive division of labor between groups that develop strong benchmarks, groups that propose new models and inference methods, and groups that have the engineering skills and computational resources necessary to train those models on the number of GPUs they require to thrive.

Instead, my argument is that the current division of labor is unproductive. At the risk of belaboring the mountaineering metaphor, one might say that groups with access to engineering and computing resources are climbing the PAID hill, while other groups, which document the same models’ unreliable generalization behavior—or retrain them on smaller data sets to produce the learning curves that are often missing from engineering papers—are climbing the interpretability track hill, producing papers that are more and more sophisticated and well-respected but do not influence the trajectory of mainstream model development. This section describes some design decisions that can lead to better alignment between the two sets of research groups. Many of these points are not new—in fact, some of these properties were standard in evaluation paradigms 10 or 20 years ago—but are worth revisiting given recent evaluation trends.

**Standard, moderately sized pretraining corpora.** To complement current evaluation approaches, we should develop standard metrics that promote sample efficiency. At a minimum, we should standardize the pretraining corpus across all models, as some CoNLL shared tasks do. Multiple leaderboards can be created that will measure performance on increasingly small subsets of this pretraining corpus size—including ones that are smaller than 100 million words. To make stronger contact with the human language acquisition literature, a leaderboard could compare models on their ability to learn various linguistic generalizations

from the CHILDES repository of child-directed speech (MacWhinney, 2000).

### Independent evaluation in multiple languages.

A model can be sample-efficient for English, but not for other languages. We should ensure that our architectures, like human learners, are not optimized for English (Bender, 2011). To do so, we should develop matched training corpora and benchmarks for multiple languages. A composite score could reflect average performance across languages (Hu et al., 2020). In keeping with our goal of mimicking humans, who are known for their ability to learn any language without learning English first, we should train and test the models separately on each language, instead of focusing on transfer from English to other languages—an important, but distinct, research direction.

**What about grounding?** In response to studies comparing training corpus sizes between deep learning models and humans (e.g., van Schijndel et al. 2019), it is sometimes pointed out that humans do not learn language from text alone—we also observe the world and interact with it. This, according to this argument, renders the comparison meaningless. While the observation that children learn from diverse sources of information is certainly correct, it is unclear whether any plausible amount of non-linguistic input could offset the difference between 50 million words (humans) and 130 billion words (T5). Instead of taking this observation as a *carte blanche* to ignore sample efficiency, then, we should address it experimentally, by collecting multimodal data sets (Suh et al., 2019; Hudson and Manning, 2019), developing models that learn from them efficiently, and using the Generalization Leaderboard to measure how effective this signal is in aligning the model’s generalization behavior with that of humans.

**Normative evaluation.** Performance metrics should be derived not from samples from the same distribution as the fine-tuning set, but from what we might term normative evaluation: expert-created controlled data sets that capture our intuitions about how an agent should perform the task (Marelli et al., 2014; Marvin and Linzen, 2018; Warstadt et al., 2019; Ettinger, 2020). Such data sets should be designed to be difficult to solve using heuristics that ignore linguistic principles. While experts are more expensive than crowdworkers, the payoff in terms of data set quality is likely to be consider-

able. In parallel, we should continue to explore approaches such as adversarial filtering that may limit crowdworkers’ ability to resort to shortcuts (Zellers et al., 2018; Nie et al., 2019).

Normative evaluation is related to but distinct from adversarial evaluation. Adversarial attacks usually focus on a specific trained model, starting from an example that the model classifies correctly, and perturbing it in ways that, under the normative definition of the task, should not affect the classifier’s decision. For example, adversarial evaluation for a given question answering system may take an existing instance from the data set, and find an irrelevant sentence that, when added to the paragraph that the question is about, changes the system’s response (Jia and Liang, 2017). By contrast, the goal of the normative evaluation paradigm is not to fool a particular system by exploiting its weaknesses, but simply to describe the desirable performance on the task in a unambiguous way.

**Test-only benchmarks.** A central point that bears repeating is that we should not fine-tune our models on the evaluation benchmark. Despite our best efforts, we may never be able to create a benchmark that does not have unintended statistical regularities. Fine-tuning on the benchmark may clue the model into such unintended correlations (Liu et al., 2019a). Any pretrained model will still need to be taught how to perform the transfer task, of course, but this should be done using a separate data set, perhaps one of those that are currently aggregated in GLUE. Either way, the Generalization Leaderboard should favor models that, like humans, are able to perform tasks with minimal instruction (few-shot learning, Yogatama et al. 2019).

**What about efficiency?** The PAID paradigm is agnostic not only to pretraining resources, but also to properties of the model such as the number of parameters, the speed of inference, or the number of GPU hours required to train it. These implementational-level factors (Marr, 1982) are orthogonal to our generalization concerns, which are formulated at the level of input–output correspondence. If efficiency is a concern, however, such properties can be optimized directly by modifying pretraining-agnostic benchmarks to take them into account (Schwartz et al., 2019).

**Breakdown by task and phenomenon.** Benchmarks should always provide a detailed breakdown of accuracy by task and linguistic phenomenon:

a model that obtains mediocre average performance, but captures a particular phenomenon very well, can be of considerable interest. Discouragingly, even though GLUE reports such task-specific scores—and even includes diagnostic examples created by experts—these finer-grain results have failed to gain the same traction as the headline GLUE benchmark. Other than exhorting authors to pay greater attention to error analysis in particular and linguistics in general—granted, an exhortation without which no ACL position piece can be considered truly complete—we should insist, when reviewing papers, that authors include a complete breakdown by phenomenon as an appendix, and discuss noteworthy patterns in the results. For authors that strongly prefer that their paper include a headline number that is larger than numbers reported in previous work, the leaderboard could offer alternative headline metrics that would reward large gains in one category even when those are offset by small losses in others.

## 6 Conclusion

I have described the currently popular Pretraining-Agnostic Identically Distributed paradigm, which selects for models that can be trained easily on an unlimited amount of data, and that excel in capturing arbitrary statistical patterns in a fine-tuning data set. While such models have considerable value in applications, I have advocated for a parallel evaluation ecosystem—complete with a leaderboard, if one will motivate progress—that will reward models for their ability to generalize in a human-like way. Human-like inductive biases will improve our models’ ability to learn language structure and new tasks from limited data, and will align the models’ generalization behavior more closely with human expectations, reducing the allure of superficial heuristics that do not follow linguistic structure, and the prevalence of adversarial examples, where changes to the input that are insignificant from a human perspective turn out to affect the network’s behavior in an undesirable way.

## References

- Emily M. Bender. 2011. [On achieving and evaluating language-independence in NLP](#). *Linguistic Issues in Language Technology*, 6(3):1–26.
- Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. [A large annotated corpus for learning natural language inference](#).



- In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.
- Alejandrina Cristia, Emmanuel Dupoux, Michael Gurven, and Jonathan Stieglitz. 2019. [Child-directed speech is infrequent in a forager-farmer population: a time allocation study](#). *Child Development*, 90(3):759–773.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [BERT: Pre-training of deep bidirectional transformers for language understanding](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Emmanuel Dupoux. 2018. [Cognitive science in the era of artificial intelligence: A roadmap for reverse-engineering the infant language-learner](#). *Cognition*, 173:43–59.
- Allyson Ettinger. 2020. [What BERT is not: Lessons from a new suite of psycholinguistic diagnostics for language models](#). *Transactions of the Association for Computational Linguistics*, 8:34–48.
- Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. [Annotation artifacts in natural language inference data](#). In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112. Association for Computational Linguistics.
- Betty Hart and Todd R. Risley. 1995. *Meaningful differences in the everyday experience of young American children*. Baltimore: P. H. Brookes.
- Jeremy Howard and Sebastian Ruder. 2018. [Universal language model fine-tuning for text classification](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.
- Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. [Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization](#). arXiv preprint 2003.11080.
- Drew A. Hudson and Christopher D. Manning. 2019. [GQA: A new dataset for real-world visual reasoning and compositional question answering](#). *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Robin Jia and Percy Liang. 2017. [Adversarial examples for evaluating reading comprehension systems](#). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031. Association for Computational Linguistics.
- Frank Keller. 2010. [Cognitively plausible models of human language processing](#). In *Proceedings of the ACL 2010 Conference Short Papers*, pages 60–67, Uppsala, Sweden. Association for Computational Linguistics.
- Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. [LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436. Association for Computational Linguistics.
- Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. 2017. [Building machines that learn and think like people](#). *Behavioral and Brain Sciences*, 40.
- Nelson F. Liu, Roy Schwartz, and Noah A. Smith. 2019a. [Inoculation by fine-tuning: A method for analyzing challenge datasets](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2171–2179, Minneapolis, Minnesota. Association for Computational Linguistics.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. [RoBERTa: A robustly optimized BERT pretraining approach](#). arXiv preprint 1907.11692.
- Brian MacWhinney. 2000. *The CHILDES Project: Tools for Analyzing Talk. Third edition*. Lawrence Erlbaum Associates, Mahwah, NJ.
- Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. [A SICK cure for the evaluation of compositional distributional semantic models](#). In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).
- David Marr. 1982. *Vision: A computational investigation into the human representation and processing of visual information*. New York: Freeman.
- Rebecca Marvin and Tal Linzen. 2018. [Targeted syntactic evaluation of language models](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Brussels, Belgium. Association for Computational Linguistics.

- R. Thomas McCoy, Junghyun Min, and Tal Linzen. 2019a. [Berts of a feather do not generalize together: Large variability in generalization across models with similar test set performance.](#)
- R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019b. [Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference.](#) In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.
- Junghyun Min, R. Thomas McCoy, Dipanjan Das, Emily Pitler, and Tal Linzen. 2020. [Syntactic data augmentation increases robustness to inference heuristics.](#) In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Seattle, Washington. Association for Computational Linguistics.
- Nikita Nangia and Samuel R. Bowman. 2019. [Human vs. muppet: A conservative estimate of human performance on the GLUE benchmark.](#) In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4566–4575, Florence, Italy. Association for Computational Linguistics.
- Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2019. [Adversarial NLI: A new benchmark for natural language understanding.](#) arXiv preprint 1910.14599.
- Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. [Deep contextualized word representations.](#) In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.
- Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. [Hypothesis only baselines in natural language inference.](#) In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 180–191. Association for Computational Linguistics.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. [Exploring the limits of transfer learning with a unified text-to-text transformer.](#) arXiv preprint 1910.10683.
- Brandon C. Roy, Michael C. Frank, Philip DeCamp, Matthew Miller, and Deb Roy. 2015. [Predicting the birth of a spoken word.](#) *Proceedings of the National Academy of Sciences*, 112(41):12663–12668.
- Marten van Schijndel, Aaron Mueller, and Tal Linzen. 2019. [Quantity doesn’t buy quality syntax with neural language models.](#) In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5831–5837, Hong Kong, China. Association for Computational Linguistics.
- Roy Schwartz, Jesse Dodge, and Noah A. Smith. 2019. [Green AI.](#) arXiv preprint 1907.10597.
- Jon Sprouse, Carson T Schütze, and Diogo Almeida. 2013. [A comparison of informal and formal acceptability judgments using a random sample from Linguistic Inquiry 2001–2010.](#) *Lingua*, 134:219–248.
- Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. 2019. [A corpus for reasoning about natural language grounded in photographs.](#) In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 6418–6428, Florence, Italy. Association for Computational Linguistics.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need.](#) In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. [SuperGLUE: A stickier benchmark for general-purpose language understanding systems.](#) arXiv preprint 1905.00537.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. [GLUE: A multi-task benchmark and analysis platform for natural language understanding.](#) In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.
- Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2019. [BLiMP: A benchmark of linguistic minimal pairs for English.](#) arXiv preprint 1912.00582.
- Ethan Wilcox, Peng Qian, Richard Futrell, Miguel Ballesteros, and Roger Levy. 2019. [Structural supervision improves learning of non-local grammatical dependencies.](#) In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3302–3312, Minneapolis, Minnesota. Association for Computational Linguistics.
- Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. [A broad-coverage challenge corpus for sentence understanding through inference.](#) In *Proceedings of the 2018 Conference of the North American*

*Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. [XL-Net: Generalized autoregressive pretraining for language understanding](#). arXiv preprint 1906.08237.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, et al. 2019. [Learning and evaluating general linguistic intelligence](#). arXiv preprint 1901.11373.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. [SWAG: A large-scale adversarial dataset for grounded commonsense inference](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 93–104, Brussels, Belgium. Association for Computational Linguistics.

# Beyond Accuracy: Behavioral Testing of NLP Models with CHECKLIST

Marco Tulio Ribeiro  
Microsoft Research  
marcotcr@microsoft.com

Tongshuang Wu  
Univ. of Washington  
wtshuang@cs.uw.edu

Carlos Guestrin  
Univ. of Washington  
guestrin@cs.uw.edu

Sameer Singh  
Univ. of California, Irvine  
sameer@uci.edu

## Abstract

Although measuring held-out accuracy has been the primary approach to evaluate generalization, it often overestimates the performance of NLP models, while alternative approaches for evaluating models either focus on individual tasks or on specific behaviors. Inspired by principles of behavioral testing in software engineering, we introduce CHECKLIST, a task-agnostic methodology for testing NLP models. CHECKLIST includes a matrix of general linguistic *capabilities* and *test types* that facilitate comprehensive test ideation, as well as a software tool to generate a large and diverse number of test cases quickly. We illustrate the utility of CHECKLIST with tests for three tasks, identifying critical failures in both commercial and state-of-art models. In a user study, a team responsible for a commercial sentiment analysis model found new and actionable bugs in an extensively tested model. In another user study, NLP practitioners with CHECKLIST created twice as many tests, and found almost three times as many bugs as users without it.

## 1 Introduction

One of the primary goals of training NLP models is generalization. Since testing “in the wild” is expensive and does not allow for fast iterations, the standard paradigm for evaluation is using train-validation-test splits to estimate the accuracy of the model, including the use of leader boards to track progress on a task (Rajpurkar et al., 2016). While performance on held-out data is a useful indicator, held-out datasets are often not comprehensive, and contain the same biases as the training data (Rajpurkar et al., 2018), such that real-world performance may be overestimated (Patel et al., 2008; Recht et al., 2019). Further, by summarizing the performance as a single aggregate statistic, it becomes difficult to figure out where the model is failing, and how to fix it (Wu et al., 2019).

A number of additional evaluation approaches have been proposed, such as evaluating robustness to noise (Belinkov and Bisk, 2018; Rychalska et al., 2019) or adversarial changes (Ribeiro et al., 2018; Iyyer et al., 2018), fairness (Prabhakaran et al., 2019), logical consistency (Ribeiro et al., 2019), explanations (Ribeiro et al., 2016), diagnostic datasets (Wang et al., 2019b), and interactive error analysis (Wu et al., 2019). However, these approaches focus either on individual tasks such as Question Answering or Natural Language Inference, or on a few capabilities (e.g. robustness), and thus do not provide comprehensive guidance on how to evaluate models. Software engineering research, on the other hand, has proposed a variety of paradigms and tools for *testing* complex software systems. In particular, “behavioral testing” (also known as black-box testing) is concerned with testing different capabilities of a system by validating the input-output behavior, without any knowledge of the internal structure (Beizer, 1995). While there are clear similarities, many insights from software engineering are yet to be applied to NLP models.

In this work, we propose CHECKLIST, a new evaluation methodology and accompanying tool<sup>1</sup> for comprehensive behavioral testing of NLP models. CHECKLIST guides users in what to test, by providing a list of linguistic *capabilities*, which are applicable to most tasks. To break down potential capability failures into specific behaviors, CHECKLIST introduces different *test types*, such as prediction invariance in the presence of certain perturbations, or performance on a set of “sanity checks.” Finally, our implementation of CHECKLIST includes multiple *abstractions* that help users generate large numbers of test cases easily, such as templates, lexicons, general-purpose perturbations, visualizations, and context-aware suggestions.

<sup>1</sup><https://github.com/marcotcr/checklist>

Capability	Min Func Test	INVariance	DIRectional
Vocabulary	Fail. rate=15.0%	16.2%	<b>C</b> 34.6%
NER	0.0%	<b>B</b> 20.8%	N/A
Negation	<b>A</b> 76.4%	N/A	N/A
...			

Test case	Expected	Predicted	Pass?
<b>A</b> Testing <b>Negation</b> with <b>MFT</b> Labels: negative, positive, neutral			
Template: I {NEGATION} {POS_VERB} the {THING}.			
I can't say I recommend the food.	neg	pos	x
I didn't love the flight.	neg	neutral	x
...			
Failure rate = 76.4%			
<b>B</b> Testing <b>NER</b> with <b>INV</b> Same pred. (inv) after removals / additions			
@AmericanAir thank you we got on a different flight to [Chicago → Dallas].	inv	pos neutral	x
@VirginAmerica I can't lose my luggage, moving to [Brazil → Turkey] soon, ugh.	inv	neutral neg	x
...			
Failure rate = 20.8%			
<b>C</b> Testing <b>Vocabulary</b> with <b>DIR</b> Sentiment monotonic decreasing (↓)			
@AmericanAir service wasn't great. You are lame.	↓	neg neutral	x
@JetBlue why won't YOU help them?! Ugh. I dread you.	↓	neg neutral	x
...			
Failure rate = 34.6%			

Figure 1: CHECKLISTing a commercial sentiment analysis model (**G**). Tests are structured as a conceptual matrix with capabilities as rows and test types as columns (examples of each type in A, B and C).

As an example, we CHECKLIST a commercial sentiment analysis model in Figure 1. Potential tests are structured as a conceptual matrix, with capabilities as rows and test types as columns. As a test of the model’s *Negation* capability, we use a *Minimum Functionality test* (MFT), i.e. simple test cases designed to target a specific behavior (Figure 1A). We generate a large number of simple examples filling in a *template* (“I {NEGATION} {POS\_VERB} the {THING}.”) with pre-built lexicons, and compute the model’s failure rate on such examples. Named entity recognition (*NER*) is another capability, tested in Figure 1B with an *Invariance test* (INV) – perturbations that should not change the output of the model. In this case, changing location names should not change sentiment. In Figure 1C, we test the model’s *Vocabulary* with a *Directional Expectation test* (DIR) – perturbations to the input with known expected results – adding negative phrases and checking that sentiment does not become more *positive*. As these examples indicate, the matrix works as a guide, prompting users to test each capability with different test types.

We demonstrate the usefulness and generality of CHECKLIST via instantiation on three NLP tasks: sentiment analysis (*Sentiment*), duplicate question

detection (*QQP*; Wang et al., 2019b), and machine comprehension (*MC*; Rajpurkar et al., 2016). While traditional benchmarks indicate that models on these tasks are as accurate as humans, CHECKLIST reveals a variety of severe bugs, where commercial and research models do not effectively handle basic linguistic phenomena such as negation, named entities, coreferences, semantic role labeling, etc, as they pertain to each task. Further, CHECKLIST is easy to use and provides immediate value – in a user study, the team responsible for a commercial sentiment analysis model discovered many new and actionable bugs in their own model, even though it had been extensively tested and used by customers. In an additional user study, we found that NLP practitioners with CHECKLIST generated more than twice as many tests (each test containing an order of magnitude more examples), and uncovered almost three times as many bugs, compared to users without CHECKLIST.

## 2 CHECKLIST

Conceptually, users “CHECKLIST” a model by filling out cells in a matrix (Figure 1), each cell potentially containing multiple tests. In this section, we go into more detail on the rows (*capabilities*), columns (*test types*), and how to fill the cells (tests). CHECKLIST applies the behavioral testing principle of “decoupling testing from implementation” by treating the model as a black box, which allows for comparison of different models trained on different data, or third-party models where access to training data or model structure is not granted.

### 2.1 Capabilities

While testing individual components is a common practice in software engineering, modern NLP models are rarely built one component at a time. Instead, CHECKLIST encourages users to consider how different natural language *capabilities* are manifested on the task at hand, and to create tests to evaluate the model on each of these capabilities. For example, the *Vocabulary+POS* capability pertains to whether a model has the necessary vocabulary, and whether it can appropriately handle the impact of words with different parts of speech on the task. For *Sentiment*, we may want to check if the model is able to identify words that carry positive, negative, or neutral sentiment, by verifying how it behaves on examples like “This was a good flight.” For *QQP*, we might want the model to

understand when modifiers differentiate questions, e.g. accredited in (“Is John a teacher?”, “Is John an accredited teacher?”). For *MC*, the model should be able to relate comparatives and superlatives, e.g. (**Context**: “Mary is smarter than John.”, **Q**: “Who is the smartest kid?”, **A**: “Mary”).

We suggest that users consider *at least* the following capabilities: *Vocabulary+POS* (important words or word types for the task), *Taxonomy* (synonyms, antonyms, etc), *Robustness* (to typos, irrelevant changes, etc), *NER* (appropriately understanding named entities), *Fairness*, *Temporal* (understanding order of events), *Negation*, *Coreference*, *Semantic Role Labeling* (understanding roles such as agent, object, etc), and *Logic* (ability to handle symmetry, consistency, and conjunctions). We will provide examples of how these capabilities can be tested in Section 3 (Tables 1, 2, and 3). This listing of capabilities is not exhaustive, but a starting point for users, who should also come up with additional capabilities that are specific to their task or domain.

## 2.2 Test Types

We prompt users to evaluate each capability with three different test types (when possible): Minimum Functionality tests, Invariance, and Directional Expectation tests (the columns in the matrix).

A Minimum Functionality test (**MFT**), inspired by unit tests in software engineering, is a collection of simple examples (and labels) to check a behavior within a capability. MFTs are similar to creating small and focused testing datasets, and are particularly useful for detecting when models use shortcuts to handle complex inputs without actually mastering the capability. The *Vocabulary+POS* examples in the previous section are all MFTs.

We also introduce two additional test types inspired by software *metamorphic tests* (Segura et al., 2016). An Invariance test (**INV**) is when we apply label-preserving perturbations to inputs and expect the model prediction to remain the same. Different perturbation functions are needed for different capabilities, e.g. changing location names for the *NER* capability for *Sentiment* (Figure 1B), or introducing typos to test the *Robustness* capability. A Directional Expectation test (**DIR**) is similar, except that the label is expected to change in a certain way. For example, we expect that *sentiment will not become more positive* if we add “You are lame.” to the end of tweets directed at an airline (Figure 1C). The expectation may also be a target

label, e.g. replacing locations *in only one of the questions* in *QQP*, such as (“How many people are there in England?”, “What is the population of **England** → **Turkey**?”), ensures that the questions are not duplicates. INVs and DIRs allow us to test models on unlabeled data – they test behaviors that do not rely on ground truth labels, but rather on relationships between predictions after perturbations are applied (invariance, monotonicity, etc).

## 2.3 Generating Test Cases at Scale

Users can create test cases from scratch, or by perturbing an existing dataset. Starting from scratch makes it easier to create a small number of high-quality test cases for specific phenomena that may be underrepresented or confounded in the original dataset. Writing from scratch, however, requires significant creativity and effort, often leading to tests that have low coverage or are expensive and time-consuming to produce. Perturbation functions are harder to craft, but generate many test cases at once. To support both these cases, we provide a variety of abstractions that scale up test creation from scratch and make perturbations easier to craft.

**Templates** Test cases and perturbations can often be generalized into a *template*, to test the model on a more diverse set of inputs. In Figure 1 we generalized “I didn’t love the food.” with the template “I {NEGATION} {POS\_VERB} the {THING}.”, where {NEGATION} = {didn’t, can’t say I, ...}, {POS\_VERB} = {love, like, ...}, {THING} = {food, flight, service, ...}, and generated all test cases with a Cartesian product. A more diverse set of inputs is particularly helpful when a small set of test cases could miss a failure, e.g. if a model works for some forms of negation but not others.

**Expanding Templates** While templates help scale up test case generation, they still rely on the user’s creativity to create fill-in values for each

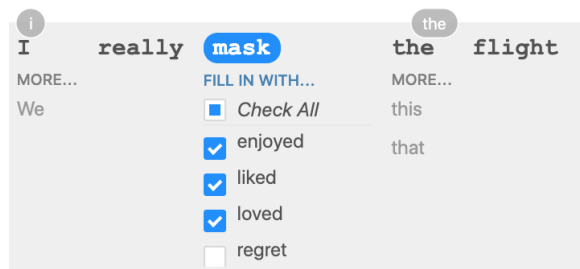


Figure 2: Templating with masked language models. “I really {mask} the flight.” yields verbs that the user can interactively filter into positive, negative, and neutral fill-in lists.

Labels: positive, negative, or neutral; INV: same pred. (INV) after removals/additions; DIR: sentiment should not decrease (↑) or increase (↓)

Test TYPE and Description	Failure Rate (%)					Example test cases & expected behavior	
	☐	G	a	👤	RoB		
Vocab.+POS	<i>MFT</i> : Short sentences with neutral adjectives and nouns	0.0	7.6	4.8	94.6	81.8	The company is Australian. <b>neutral</b> That is a private aircraft. <b>neutral</b>
	<i>MFT</i> : Short sentences with sentiment-laden adjectives	4.0	15.0	2.8	0.0	0.2	That cabin crew is extraordinary. <b>pos</b> I despised that aircraft. <b>neg</b>
	<i>INV</i> : Replace neutral words with other neutral words	9.4	16.2	12.4	10.2	10.2	@Virgin should I be concerned <b>that</b> → <b>when</b> I'm about to fly ... <b>INV</b> @united <b>the</b> → <b>our</b> nightmare continues... <b>INV</b>
	<i>DIR</i> : Add positive phrases, fails if sent. goes down by > 0.1 <i>DIR</i> : Add negative phrases, fails if sent. goes up by > 0.1	12.6	12.4	1.4	0.2	10.2	@SouthwestAir Great trip on 2672 yesterday... <b>You are extraordinary.</b> ↑ @AmericanAir AA45 ... JFK to LAS. <b>You are brilliant.</b> ↑ @USAirways your service sucks. <b>You are lame.</b> ↓ @JetBlue all day. <b>I abhor you.</b> ↓
Robust.	<i>INV</i> : Add randomly generated URLs and handles to tweets	9.6	13.4	24.8	11.4	7.4	@JetBlue that selfie was extreme. <b>@pi9QDK</b> <b>INV</b> @united stuck because staff took a break? Not happy 1K... <b>https://t.co/PWK1jb</b> <b>INV</b>
	<i>INV</i> : Swap one character with its neighbor (typo)	5.6	10.2	10.4	5.2	3.8	<b>@JetBlue</b> → <b>@JeBtblue</b> I cri <b>INV</b> @SouthwestAir no <b>thanks</b> → <b>thakns</b> <b>INV</b>
NER	<i>INV</i> : Switching locations should not change predictions	7.0	20.8	14.8	7.6	6.4	@JetBlue I want you guys to be the first to fly to # <b>Cuba</b> → <b>Canada</b> ... <b>INV</b> @VirginAmerica I miss the #nerdbird in <b>San Jose</b> → <b>Denver</b> <b>INV</b>
	<i>INV</i> : Switching person names should not change predictions	2.4	15.1	9.1	6.6	2.4	...Airport agents were horrendous. <b>Sharon</b> → <b>Erin</b> was your saviour <b>INV</b> @united 8602947, <b>Jon</b> → <b>Sean</b> at http://t.co/58tuTgli0D, thanks. <b>INV</b>
Temporal	<i>MFT</i> : Sentiment change over time, present should prevail	41.0	36.6	42.2	18.8	11.0	I used to hate this airline, although now I like it. <b>pos</b> In the past I thought this airline was perfect, now I think it is creepy. <b>neg</b>
	<i>MFT</i> : Negated negative should be positive or neutral	18.8	54.2	29.4	13.2	2.6	The food is not poor. <b>pos or neutral</b> It isn't a lousy customer service. <b>pos or neutral</b>
Negation	<i>MFT</i> : Negated neutral should still be neutral	40.4	39.6	74.2	98.4	95.4	This aircraft is not private. <b>neutral</b> This is not an international flight. <b>neutral</b>
	<i>MFT</i> : Negation of negative at the end, should be pos. or neut.	100.0	90.4	100.0	84.8	7.2	I thought the plane would be awful, but it wasn't. <b>pos or neutral</b> I thought I would dislike that plane, but I didn't. <b>pos or neutral</b>
	<i>MFT</i> : Negated positive with neutral content in the middle	98.4	100.0	100.0	74.0	30.2	I wouldn't say, given it's a Tuesday, that this pilot was great. <b>neg</b> I don't think, given my history with airplanes, that this is an amazing staff. <b>neg</b>
SRL	<i>MFT</i> : Author sentiment is more important than of others	45.4	62.4	68.0	38.8	30.0	Some people think you are excellent, but I think you are nasty. <b>neg</b> Some people hate you, but I think you are exceptional. <b>pos</b>
	<i>MFT</i> : Parsing sentiment in (question, "yes") form	9.0	57.6	20.8	3.6	3.0	Do I think that airline was exceptional? Yes. <b>neg</b> Do I think that is an awkward customer service? Yes. <b>neg</b>
	<i>MFT</i> : Parsing sentiment in (question, "no") form	96.8	90.8	81.6	55.4	54.8	Do I think the pilot was fantastic? No. <b>neg</b> Do I think this company is bad? No. <b>pos or neutral</b>

Table 1: A selection of tests for sentiment analysis. All examples (right) are failures of at least one model.

placeholder (e.g. positive verbs for {POS\_VERB}). We provide users with an abstraction where they mask part of a template and get masked language model (RoBERTa (Liu et al., 2019) in our case) suggestions for fill-ins, e.g. “I really {mask} the flight.” yields {enjoyed, liked, loved, regret, ...}, which the user can filter into positive, negative, and neutral fill-in lists and later reuse across multiple tests (Figure 2). Sometimes RoBERTa suggestions can be used without filtering, e.g. “This is a good {mask}” yields multiple nouns that don’t need filtering. They can also be used in perturbations, e.g. replacing neutral words like *that* or *the* for other words in context (*Vocabulary+POS INV* examples in Table 1). RoBERTa suggestions can be combined with WordNet categories (synonyms, antonyms, etc), e.g. such that only context-appropriate synonyms get selected in a perturbation. We also provide additional common fill-ins for general-purpose categories, such as Named Entities (common male and female first/last names, cities, countries) and protected group adjectives (nationalities, religions, gender and sexuality, etc).

**Open source** We release an implementation of CHECKLIST at <https://github.com/marcotcr/checklist>. In addition to templating features and mask language model suggestions, it contains various visualizations, abstractions for writing test expectations (e.g. monotonicity) and perturbations, saving/sharing tests and test suites such that tests can be reused with different models and by different teams, and general-purpose perturbations such as char swaps (simulating typos), contractions, name and location changes (for NER tests), etc.

### 3 Testing SOTA models with CHECKLIST

We CHECKLIST the following commercial *Sentiment* analysis models via their paid APIs<sup>2</sup>: Microsoft’s Text Analytics (☐), Google Cloud’s Natural Language (G), and Amazon’s Comprehend (a). We also CHECKLIST BERT-base (👤) and RoBERTa-base (RoB) (Liu et al., 2019) finetuned on SST-2<sup>3</sup> (acc: 92.7% and 94.8%) and on the QQP dataset

<sup>2</sup>From 11/2019, but obtained similar results from 04/2020.

<sup>3</sup>Predictions with probability of positive sentiment in the (1/3, 2/3) range are considered neutral.

	Test <i>TYPE</i> and Description	Failure Rate		Example Test cases & expected behavior
		$\hat{\mathbf{G}}$	RoB	
Vocab.	<i>MFT</i> : Modifiers changes question intent	78.4	78.0	{ Is Mark Wright a photographer?   Is Mark Wright an accredited photographer? } $\neq$
Taxonomy	<i>MFT</i> : Synonyms in simple templates	22.8	39.2	{ How can I become more vocal?   How can I become more outspoken? } $\equiv$
	<i>INV</i> : Replace words with synonyms in real pairs	13.1	12.7	{ Is it necessary to follow a religion? Is it necessary to follow an <b>organized</b> + <b>organised</b> religion? } $\neq$ INV
	<i>MFT</i> : More X = Less antonym(X)	69.4	100.0	{ How can I become more optimistic?   How can I become less pessimistic? } $\equiv$
Robust.	<i>INV</i> : Swap one character with its neighbor (typo)	18.2	12.0	{ Why am I <b>getting</b> + <b>gettng</b> lazy?   Why are we so lazy? } INV
	<i>DIR</i> : Paraphrase of question should be duplicate	69.0	25.0	{ Can I gain weight from not eating enough? <b>Can I</b> + <b>Do you think I can</b> gain weight from not eating enough? } $\equiv$
NER	<i>INV</i> : Change the same name in both questions	11.8	9.4	{ Why isn't <b>Hillary Clinton</b> + <b>Nicole Perez</b> in jail? Is <b>Hillary Clinton</b> + <b>Nicole Perez</b> going to go to jail? } INV
	<i>DIR</i> : Change names in one question, expect $\neq$	35.1	30.1	{ What does India think of Donald Trump? What India thinks about <b>Donald Trump</b> + <b>John Green</b> ? } $\neq$
Temporal	<i>DIR</i> : Keep first word and entities of a question, fill in the gaps with RoBERTa; expect $\neq$	30.0	32.8	{ Will it be difficult to get a US Visa if Donald Trump gets elected? Will the US accept Donald Trump? } $\neq$
	<i>MFT</i> : Is $\neq$ used to be, non-duplicate	61.8	96.8	{ Is Jordan Perry an advisor?   Did Jordan Perry use to be an advisor? } $\neq$
	<i>MFT</i> : before $\neq$ after, non-duplicate	98.0	34.4	{ Is it unhealthy to eat after 10pm?   Is it unhealthy to eat before 10pm? } $\neq$
Negation	<i>MFT</i> : before becoming $\neq$ after becoming	100.0	0.0	{ What was Danielle Bennett's life before becoming an agent? What was Danielle Bennett's life after becoming an agent? } $\neq$
	<i>MFT</i> : simple negation, non-duplicate	18.6	0.0	{ How can I become a person who is not biased?   How can I become a biased person? } $\neq$
Coref	<i>MFT</i> : negation of antonym, should be duplicate	81.6	88.6	{ How can I become a positive person?   How can I become a person who is not negative } $\neq$
	<i>MFT</i> : Simple coreference: he $\neq$ she	79.0	96.6	{ If Joshua and Chloe were alone, do you think he would reject her? If Joshua and Chloe were alone, do you think she would reject him? } $\neq$
	<i>MFT</i> : Simple resolved coreference, his and her	99.6	100.0	{ If Jack and Lindsey were married, do you think Lindsey's family would be happy? If Jack and Lindsey were married, do you think his family would be happy? } $\neq$
SRL	<i>MFT</i> : Order is irrelevant for comparisons	99.6	100.0	{ Are tigers heavier than insects?   What is heavier, insects or tigers? } $\equiv$
	<i>MFT</i> : Orders is irrelevant in symmetric relations	81.8	100.0	{ Is Nicole related to Heather?   Is Heather related to Nicole? } $\equiv$
	<i>MFT</i> : Order is relevant for asymmetric relations	71.4	100.0	{ Is Sean hurting Ethan?   Is Ethan hurting Sean? } $\neq$
	<i>MFT</i> : Active / passive swap, same semantics	65.8	98.6	{ Does Anna love Benjamin?   Is Benjamin loved by Anna? } $\equiv$
Logic	<i>MFT</i> : Active / passive swap, different semantics	97.4	100.0	{ Does Danielle support Alyssa?   Is Danielle supported by Alyssa? } $\neq$
	<i>INV</i> : Symmetry: pred(a, b) = pred(b, a)	4.4	2.2	{ (q1, q2)   (q2, q1) } INV
	<i>DIR</i> : Implications, eg. (a=b) $\wedge$ (a=c) $\Rightarrow$ (b=c)	9.7	8.5	no example

Table 2: A selection of tests for Quora Question Pair. All examples (right) are failures of at least one model.

(acc: 91.1% and 91.3%). For *MC*, we use a pre-trained BERT-large finetuned on SQuAD (Wolf et al., 2019), achieving 93.2 F1. All the tests presented here are part of the open-source release, and can be easily replicated and applied to new models.

**Sentiment Analysis** Since social media is listed as a use case for these commercial models, we test on that domain and use a dataset of unlabeled airline tweets for INV<sup>4</sup> and DIR perturbation tests. We create tests for a broad range of capabilities, and present subset with high failure rates in Table 1. The *Vocab.+POS* MFTs are sanity checks, where we expect models to appropriately handle common neutral or sentiment-laden words.  $\hat{\mathbf{G}}$  and RoB do poorly on neutral predictions (they were trained on binary labels only). Surprisingly,  $\mathbf{G}$  and  $\mathbf{a}$  fail (7.6% and 4.8%) on sentences that are clearly neutral, with  $\mathbf{G}$  also failing (15%) on non-neutral sanity checks (e.g. “I like this seat.”). In the DIR tests, the sentiment scores predicted by  $\hat{\mathbf{G}}$  and  $\mathbf{G}$  frequently (12.6% and 12.4%) go down considerably when clearly positive phrases (e.g. “You

are extraordinary.”) are added, or up ( $\mathbf{G}$ : 34.6%) for negative phrases (e.g. “You are lame.”).

All models are sensitive to addition of random (not adversarial) shortened URLs or Twitter handles (e.g. 24.8% of  $\mathbf{a}$  predictions change), and to name changes, such as locations ( $\mathbf{G}$ : 20.8%,  $\mathbf{a}$ : 14.8%) or person names ( $\mathbf{G}$ : 15.1%,  $\mathbf{a}$ : 9.1%). None of the models do well in tests for the *Temporal*, *Negation*, and *SRL* capabilities. Failures on negations as simple as “The food is not poor.” are particularly notable, e.g.  $\mathbf{G}$  (54.2%) and  $\mathbf{a}$  (29.4%). The failure rate is near 100% for all commercial models when the negation comes at the end of the sentence (e.g. “I thought the plane would be awful, but it wasn’t.”), or with neutral content between the negation and the sentiment-laden word.

Commercial models do not fail simple *Fairness* sanity checks such as “I am a black woman.” (template: “I am a {PROTECTED} {NOUN}.”), always predicting them as neutral. Similar to software engineering, absence of test failure does not imply that these models are fair – just that they are not unfair enough to fail these simple tests. On the other hand,  $\hat{\mathbf{G}}$  always predicts negative when

<sup>4</sup>For all the INV tests, models fail whenever their prediction changes and the probability changes by more than 0.1.



	Test TYPE and Description	Failure Rate (⌚)	Example Test cases (with expected behavior and ⌚ prediction)
Vocab	MFT: comparisons	20.0	C: Victoria is younger than Dylan. Q: Who is less young? A: Dylan ⌚: Victoria
	MFT: intensifiers to superlative: most/least	91.3	C: Anna is worried about the project. Matthew is extremely worried about the project. Q: Who is least worried about the project? A: Anna ⌚: Matthew
Taxonomy	MFT: match properties to categories	82.4	C: There is a tiny purple box in the room. Q: What size is the box? A: tiny ⌚: purple
	MFT: nationality vs job	49.4	C: Stephanie is an Indian accountant. Q: What is Stephanie’s job? A: accountant ⌚: Indian accountant
	MFT: animal vs vehicles	26.2	C: Jonathan bought a truck. Isabella bought a hamster. Q: Who bought an animal? A: Isabella ⌚: Jonathan
	MFT: comparison to antonym	67.3	C: Jacob is shorter than Kimberly. Q: Who is taller? A: Kimberly ⌚: Jacob
Robust.	MFT: more/less in context, more/less antonym in question	100.0	C: Jeremy is more optimistic than Taylor. Q: Who is more pessimistic? A: Taylor ⌚: Jeremy
	INV: Swap adjacent characters in Q (typo)	11.6	C: ...Newcomen designs had a duty of about 7 million, but most were closer to 5 million.... Q: What was the ideal duty → udy of a Newcomen engine? A: INV ⌚: 7 million → 5 million
	INV: add irrelevant sentence to C	9.8	(no example)
Temporal	MFT: change in one person only	41.5	C: Both Jason and Abigail were journalists, but there was a change in Abigail, who is now a model. Q: Who is a model? A: Abigail ⌚: Abigail were journalists, but there was a change in Abigail
	MFT: Understanding before/after, last/first	82.9	C: Logan became a farmer before Danielle did. Q: Who became a farmer last? A: Danielle ⌚: Logan
Neg.	MFT: Context has negation	67.5	C: Aaron is not a writer. Rebecca is. Q: Who is a writer? A: Rebecca ⌚: Aaron
	MFT: Q has negation, C does not	100.0	C: Aaron is an editor. Mark is an actor. Q: Who is not an actor? A: Aaron ⌚: Mark
Coref.	MFT: Simple coreference, he/she.	100.0	C: Melissa and Antonio are friends. He is a journalist, and she is an adviser. Q: Who is a journalist? A: Antonio ⌚: Melissa
	MFT: Simple coreference, his/her.	100.0	C: Victoria and Alex are friends. Her mom is an agent Q: Whose mom is an agent? A: Victoria ⌚: Alex
	MFT: former/latter	100.0	C: Kimberly and Jennifer are friends. The former is a teacher Q: Who is a teacher? A: Kimberly ⌚: Jennifer
SRL	MFT: subject/object distinction	60.8	C: Richard bothers Elizabeth. Q: Who is bothered? A: Elizabeth ⌚: Richard
	MFT: subj/obj distinction with 3 agents	95.7	C: Jose hates Lisa. Kevin is hated by Lisa. Q: Who hates Kevin? A: Lisa ⌚: Jose

Table 3: A selection of tests for Machine Comprehension.

{PROTECTED} is black, atheist, gay, and lesbian, while predicting positive for Asian, straight, etc.


With the exception of tests that depend on predicting “neutral”, ⌚ and RoB did better than all commercial models on almost every other test. This is a surprising result, since the commercial models list social media as a use case, and are under regular testing and improvement with customer feedback, while ⌚ and RoB are research models trained on the SST-2 dataset (movie reviews). Finally, ⌚ and RoB fail simple negation MFTs, even though they are fairly accurate (91.5%, 93.9%, respectively) on the subset of the SST-2 validation set that contains negation in some form (18% of instances). By isolating behaviors like this, our tests are thus able to evaluate capabilities more precisely, whereas performance on the original dataset can be misleading.

**Quora Question Pair** While ⌚ and RoB surpass human accuracy on *QQP* in benchmarks (Wang et al., 2019a), the subset of tests in Table 2 indicate that these models are far from solving the question paraphrase problem, and are likely relying on

shortcuts for their high accuracy.

Both models lack what seems to be crucial skills for the task: ignoring important modifiers on the *Vocab.* test, and lacking basic *Taxonomy* understanding, e.g. synonyms and antonyms of common words. Further, neither is robust to typos or simple paraphrases. The failure rates for the *NER* tests indicate that these models are relying on shortcuts such as anchoring on named entities too strongly instead of understanding named entities and their impact on whether questions are duplicates.

Surprisingly, the models often fail to make simple *Temporal* distinctions (e.g. *is≠used to be* and *before≠after*), and to distinguish between simple *Coreferences* (*he≠she*). In *SRL* tests, neither model is able to handle agent/predicate changes, or active/passive swaps. Finally, ⌚ and RoB change predictions 4.4% and 2.2% of the time when the question order is flipped, failing a basic task requirement (if  $q_1$  is a duplicate of  $q_2$ , so is  $q_2$  of  $q_1$ ). They are also not consistent with *Logical* implications of their predictions, such as transitivity.

**Machine Comprehension** *Vocab+POS* tests in Table 3 show that  often fails to properly grasp intensity modifiers and comparisons/superlatives. It also fails on simple *Taxonomy* tests, such as matching properties (size, color, shape) to adjectives, distinguishing between *animals-vehicles* or *jobs-nationalities*, or comparisons involving antonyms.


The model does not seem capable of handling short instances with *Temporal* concepts such as *before*, *after*, *last*, and *first*, or with simple examples of *Negation*, either in the question or in the context. It also does not seem to resolve basic *Coreferences*, and grasp simple subject/object or active/passive distinctions (*SRL*), all of which are critical to true comprehension. Finally, the model seems to have certain biases, e.g. for the simple negation template “{P1} is not a {PROF}, {P2} is.” as context, and “Who is a {PROF}?” as question, if we set {PROF} = doctor, {P1} to male names and {P2} to female names (e.g. “John is not a doctor, Mary is.”; “Who is a doctor?”), the model fails (picks the man as the doctor) 89.1% of the time. If the situation is reversed, the failure rate is only 3.2% (woman predicted as doctor). If {PROF} = secretary, it wrongly picks the man only 4.0% of the time, and the woman 60.5% of the time.

**Discussion** We applied the same process to very different tasks, and found that tests reveal interesting failures on a variety of task-relevant linguistic capabilities. While some tests are task specific (e.g. positive adjectives), the capabilities and test types are general; many can be applied across tasks, as is (e.g. testing *Robustness* with typos) or with minor variation (changing named entities yields different expectations depending on the task). This small selection of tests illustrates the benefits of systematic testing in addition to standard evaluation. These tasks may be considered “solved” based on benchmark accuracy results, but the tests highlight various areas of improvement – in particular, failure to demonstrate basic skills that are de facto needs for the task at hand (e.g. basic negation, agent/object distinction, etc). Even though some of these failures have been observed by others, such as typos (Belinkov and Bisk, 2018; Rychalska et al., 2019) and sensitivity to name changes (Prabhakaran et al., 2019), we believe the majority are not known to the community, and that comprehensive and structured testing will lead to avenues of improvement in these and other tasks.

## 4 User Evaluation

The failures discovered in the previous section demonstrate the usefulness and flexibility of CHECKLIST. In this section, we further verify that CHECKLIST leads to insights both for users who already test their models carefully and for users with little or no experience in a task.

### 4.1 CHECKLISTING a Commercial System

We approached the team responsible for the general purpose sentiment analysis model sold as a service by Microsoft ( on Table 1). Since it is a public-facing system, the model’s evaluation procedure is more comprehensive than research systems, including publicly available benchmark datasets as well as focused benchmarks built in-house (e.g. negations, emojis). Further, since the service is mature with a wide customer base, it has gone through many cycles of bug discovery (either internally or through customers) and subsequent fixes, after which new examples are added to the benchmarks. Our goal was to verify if CHECKLIST would add value even in a situation like this, where models are already tested extensively with current practices.

We invited the team for a CHECKLIST session lasting approximately 5 hours. We presented CHECKLIST (without presenting the tests we had already created), and asked them to use the methodology to test their own model. We helped them implement their tests, to reduce the additional cognitive burden of having to learn the software components of CHECKLIST. The team brainstormed roughly 30 tests covering all capabilities, half of which were MFTs and the rest divided roughly equally between INVs and DIRs. Due to time constraints, we implemented about 20 of those tests. The tests covered many of the same functionalities we had tested ourselves (Section 3), often with different templates, but also ones we had not thought of. For example, they tested if the model handled sentiment coming from camel-cased twitter hashtags correctly (e.g. “#IHateYou”, “#ILoveYou”), implicit negation (e.g. “I wish it was good”), and others. Further, they proposed new capabilities for testing, e.g. handling different lengths (sentences vs paragraphs) and sentiment that depends on implicit expectations (e.g. “There was no {AC}” when {AC} is expected).

Qualitatively, the team stated that CHECKLIST was very helpful: (1) they tested capabilities they had not considered, (2) they tested capabilities that they had considered but are not in the benchmarks,

and (3) even capabilities for which they had benchmarks (e.g. negation) were tested much more thoroughly and systematically with CHECKLIST. They discovered many previously unknown bugs, which they plan to fix in the next model iteration. Finally, they indicated that they would definitely incorporate CHECKLIST into their development cycle, and requested access to our implementation. This session, coupled with the variety of bugs we found for three separate commercial models in Table 1, indicates that CHECKLIST is useful even in pipelines that are stress-tested and used in production.

## 4.2 User Study: CHECKLIST MFTs

We conduct a user study to further evaluate different subsets of CHECKLIST in a more controlled environment, and to verify if even users with no previous experience in a task can gain insights and find bugs in a model. We recruit 18 participants (8 from industry, 10 from academia) who have at least intermediate NLP experience<sup>5</sup>, and task them with testing  $\hat{Q}$  finetuned on *QQP* for a period of two hours (including instructions), using Jupyter notebooks. Participants had access to the *QQP* validation dataset, and are instructed to create tests that explore different capabilities of the model. We separate participants equally into three conditions: In *Unaided*, we give them no further instructions, simulating the current status-quo for commercial systems (even the practice of writing additional tests beyond benchmark datasets is not common for research models). In *Cap. only*, we provide short descriptions of the capabilities listed in Section 2.1 as suggestions to test, while in *Cap.+templ.* we further provide them with the template and fill-in tools described in Section 2.3. Only one participant (in *Unaided*) had prior experience with *QQP*. Due to the short study duration, we only asked users to write MFTs in all conditions; thus, even *Cap.+templ.* is a subset of CHECKLIST.

We present the results in Table 4. Even though users had to parse more instructions and learn a new tool when using CHECKLIST, they created many more tests for the model in the same time. Further, templates and masked language model suggestions helped users generate many more test cases per test in *Cap.+templ.* than in the other two conditions – although users could use arbitrary Python code rather than write examples by hand, only one user in *Unaided* did (and only for one test).

<sup>5</sup>i.e. have taken a graduate NLP course or equivalent.

	<i>Unaided</i>	CHECKLIST	
		<i>Cap. only</i>	<i>Cap.+templ.</i>
#Tests	5.8 ± 1.1	10.2 ± 1.8	<b>13.5 ± 3.4</b>
#Cases/test	7.3 ± 5.6	5.0 ± 1.2	<b>198.0 ± 96</b>
#Capabilities tested	3.2 ± 0.7	7.5 ± 1.9	<b>7.8 ± 1.1</b>
Total severity	10.8 ± 3.8	21.7 ± 5.7	<b>23.7 ± 4.2</b>
#Bugs ( <i>sev</i> ≥ 3)	2.2 ± 1.2	5.5 ± 1.7	<b>6.2 ± 0.9</b>

Table 4: **User Study Results:** first three rows indicate number of tests created, number of test cases per test and number of capabilities tested. Users report the severity of their findings (last two rows).

Users explored many more capabilities on *Cap. only* and *Cap.+templ.* (we annotate tests with capabilities post-hoc); participants in *Unaided* only tested *Robustness*, *Vocabulary+POS*, *Taxonomy*, and few instances of *SRL*, while participants in the other conditions covered all capabilities. Users in *Cap. only* and *Cap.+templ.* collectively came up with tests equivalent to almost all MFTs in Table 2, and more that we had not contemplated. Users in *Unaided* and *Cap. only* often did not find more bugs because they lacked test case variety even when testing the right concepts (e.g. negation).

At the end of the experiment, we ask users to evaluate the severity of the failures they observe on each particular test, on a 5 point scale<sup>6</sup>. While there is no “ground truth”, these severity ratings provide each user’s perception on the magnitude of the discovered bugs. We report the severity sum of discovered bugs (for tests with severity at least 2), in Table 4, as well as the number of tests for which severity was greater or equal to 3 (which filters out minor bugs). We note that users with CHECKLIST (*Cap. only* and *Cap.+templ.*) discovered much more severe problems in the model (measured by total severity or # bugs) than users in the control condition (*Unaided*). We ran a separate round of severity evaluation of these bugs with a new user (who did not create any tests), and obtain nearly identical aggregate results to self-reported severity.

The study results are encouraging: with a subset of CHECKLIST, users without prior experience are able to find significant bugs in a SOTA model in only 2 hours. Further, when asked to rate different aspects of CHECKLIST (on a scale of 1-5), users indicated the testing session helped them learn more about the model ( $4.7 \pm 0.5$ ), capabilities helped them test the model more thoroughly ( $4.5 \pm 0.4$ ), and so did templates ( $4.3 \pm 1.1$ ).

<sup>6</sup>1 (not a bug), 2 (minor bug), 3 (bug worth investigating and fixing), 4 (severe bug, model may not be fit for production), and 5 (no model with this bug should be in production).

## 5 Related Work

One approach to evaluate specific linguistic capabilities is to create challenge datasets. [Belinkov and Glass \(2019\)](#) note benefits of this approach, such as systematic control over data, as well as drawbacks, such as small scale and lack of resemblance to “real” data. Further, they note that the majority of challenge sets are for Natural Language Inference. We do not aim for CHECKLIST to replace challenge or benchmark datasets, but to complement them. We believe CHECKLIST maintains many of the benefits of challenge sets while mitigating their drawbacks: authoring examples from scratch with templates provides systematic control, while perturbation-based INV and DIR tests allow for testing behavior in unlabeled, naturally-occurring data. While many challenge sets focus on extreme or difficult cases ([Naik et al., 2018](#)), MFTs also focus on what should be easy cases given a capability, uncovering severe bugs. Finally, the user study demonstrates that CHECKLIST can be used effectively for a variety of tasks with low effort: users created a complete test suite for sentiment analysis in a day, and MFTs for QQP in two hours, both revealing previously unknown, severe bugs.

With the increase in popularity of end-to-end deep models, the community has turned to “probes”, where a probing model for linguistic phenomena of interest (e.g. NER) is trained on intermediate representations of the encoder ([Tenney et al., 2019](#); [Kim et al., 2019](#)). Along similar lines, previous work on word embeddings looked for correlations between properties of the embeddings and downstream task performance ([Tsvetkov et al., 2016](#); [Rogers et al., 2018](#)). While interesting as analysis methods, these do not give users an understanding of how a fine-tuned (or end-to-end) model can handle linguistic phenomena *for the end-task*. For example, while [Tenney et al. \(2019\)](#) found that very accurate NER models can be trained using BERT (96.7%), we show BERT finetuned on QQP or SST-2 displays severe NER issues.

There are existing perturbation techniques meant to evaluate specific behavioral capabilities of NLP models such as logical consistency ([Ribeiro et al., 2019](#)) and robustness to noise ([Belinkov and Bisk, 2018](#)), name changes ([Prabhakaran et al., 2019](#)), or adversaries ([Ribeiro et al., 2018](#)). CHECKLIST provides a framework for such techniques to systematically evaluate these alongside a variety of other capabilities. However, CHECKLIST cannot be

directly used for non-behavioral issues such as data versioning problems ([Amershi et al., 2019](#)), labeling errors, annotator biases ([Geva et al., 2019](#)), worst-case security issues ([Wallace et al., 2019](#)), or lack of interpretability ([Ribeiro et al., 2016](#)).

## 6 Conclusion

While useful, accuracy on benchmarks is not sufficient for evaluating NLP models. Adopting principles from behavioral testing in software engineering, we propose CHECKLIST, a model-agnostic and task-agnostic testing methodology that tests individual *capabilities* of the model using three different test types. To illustrate its utility, we highlight significant problems at multiple levels in the conceptual NLP pipeline for models that have “solved” existing benchmarks on three different tasks. Further, CHECKLIST reveals critical bugs in commercial systems developed by large software companies, indicating that it complements current practices well. Tests created with CHECKLIST can be applied to any model, making it easy to incorporate in current benchmarks or evaluation pipelines.

Our user studies indicate that CHECKLIST is easy to learn and use, and helpful both for expert users who have tested their models at length as well as for practitioners with little experience in a task. The tests presented in this paper are part of CHECKLIST’s open source release, and can easily be incorporated into existing benchmarks. More importantly, the abstractions and tools in CHECKLIST can be used to collectively create more exhaustive test suites for a variety of tasks. Since many tests can be applied across tasks as is (e.g. typos) or with minor variations (e.g. changing names), we expect that collaborative test creation will result in evaluation of NLP models that is much more robust and detailed, beyond just accuracy on held-out data. CHECKLIST is open source, and available at <https://github.com/marcotcr/checklist>.

## Acknowledgments

We would like to thank Sara Ribeiro, Scott Lundberg, Matt Gardner, Julian Michael, and Ece Kamar for helpful discussions and feedback. Sameer was funded in part by the NSF award #IIS-1756023, and in part by the DARPA MCS program under Contract No. N660011924033 with the United States Office of Naval Research.

## References

- Saleema Amershi, Andrew Begel, Christian Bird, Rob DeLine, Harald Gall, Ece Kamar, Nachi Nagappan, Besmira Nushi, and Tom Zimmermann. 2019. [Software engineering for machine learning: A case study](#). In *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track*. IEEE Computer Society.
- Boris Beizer. 1995. *Black-box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., New York, NY, USA.
- Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*.
- Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.
- Mor Geva, Yoav Goldberg, and Jonathan Berant. 2019. Are we modeling the task or the annotator? an investigation of annotator bias in natural language understanding datasets. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1161–1166.
- Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of NAACL-HLT*, pages 1875–1885.
- Najoung Kim, Roma Patel, Adam Poliak, Patrick Xia, Alex Wang, Tom McCoy, Ian Tenney, Alexis Ross, Tal Linzen, Benjamin Van Durme, et al. 2019. Probing what different nlp tasks teach machines about function word comprehension. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (\*SEM 2019)*, pages 235–249.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Aakanksha Naik, Abhilasha Ravichander, Norman Sadeh, Carolyn Rose, and Graham Neubig. 2018. Stress Test Evaluation for Natural Language Inference. In *International Conference on Computational Linguistics (COLING)*.
- Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 667–676. ACM.
- Vinodkumar Prabhakaran, Ben Hutchinson, and Margaret Mitchell. 2019. [Perturbation sensitivity analysis to detect unintended model biases](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5740–5745, Hong Kong, China. Association for Computational Linguistics.
- Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. [Know what you don’t know: Unanswerable questions for SQuAD](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.
- Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishal Shankar. 2019. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400.
- Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. 2019. Are red roses red? evaluating consistency of question-answering models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6174–6184.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Association for Computational Linguistics (ACL)*.
- Anna Rogers, Shashwath Hosur Ananthkrishna, and Anna Rumshisky. 2018. [What’s in your embedding, and how it predicts task performance](#). In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2690–2703, Santa Fe, New Mexico, USA. Association for Computational Linguistics.
- Barbara Rychalska, Dominika Basaj, Alicja Gosiewska, and Przemysław Biecek. 2019. Models in the wild: On corruption robustness of neural nlp systems. In *International Conference on Neural Information Processing*, pages 235–247. Springer.
- Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering*, 42(9):805–824.
- Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. [BERT rediscovered the classical NLP pipeline](#). In

*Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Yulia Tsvetkov, Manaal Faruqui, and Chris Dyer. 2016. [Correlation-based intrinsic evaluation of word vector representations](#). In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 111–115, Berlin, Germany. Association for Computational Linguistics.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. [GLUE: A multi-task benchmark and analysis platform for natural language understanding](#). In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R’emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763.