University of Alberta

Large-Scale Semi-Supervised Learning for Natural Language Processing

by

Shane Bergsma

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2010
Edmonton, Alberta

Examining Committee

Randy Goebel, Computing Science

Dekang Lin, Computing Science

Greg Kondrak, Computing Science

Dale Schuurmans, Computing Science

Chris Westbury, Psychology

Eduard Hovy, Information Sciences Institute, University of Southern California

Abstract

Natural Language Processing (NLP) develops computational approaches to processing language data. Supervised machine learning has become the dominant methodology of modern NLP. The performance of a supervised NLP system crucially depends on the amount of data available for training. In the standard supervised framework, if a sequence of words was not encountered in the training set, the system can only guess at its label at test time. The cost of producing labeled training examples is a bottleneck for current NLP technology. On the other hand, a vast quantity of unlabeled data is freely available.

This dissertation proposes effective, efficient, versatile methodologies for 1) extracting useful information from very large (potentially web-scale) volumes of unlabeled data and 2) combining such information with standard supervised machine learning for NLP. We demonstrate novel ways to exploit unlabeled data, we scale these approaches to make use of all the text on the web, and we show improvements on a variety of challenging NLP tasks. This combination of learning from both labeled and unlabeled data is often referred to as *semi-supervised learning*.

Although lacking manually-provided labels, the statistics of unlabeled patterns can often distinguish the correct label for an ambiguous test instance. In the first part of this dissertation, we propose to use the counts of unlabeled patterns as features in supervised classifiers, with these classifiers trained on varying amounts of labeled data. We propose a general approach for integrating information from multiple, overlapping sequences of context for lexical disambiguation problems. We also show how standard machine learning algorithms can be modified to incorporate a particular kind of prior knowledge: knowledge of effective weightings for count-based features. We also evaluate performance within and across domains for two generation and two analysis tasks, assessing the impact of combining web-scale counts with conventional features. In the second part of this dissertation, rather than using the aggregate statistics as features, we propose to use them to generate labeled training examples. By automatically labeling a large number of examples, we can train powerful discriminative models, leveraging fine-grained features of input words.

Acknowledgements

I would like to recognize a number of named entities for their contributions to this thesis.

I gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada, the Alberta Ingenuity Fund, and the Alberta Informatics Circle of Research Excellence (the latter two now part of the Alberta Innovates organization).

Thank you to Dekang Lin and Randy Goebel, my excellent supervisors. I admire and appreciate you guys very much. Thanks to the other brilliant collaborators on my thesis publications: Emily Pitler, Greg Kondrak, and Dale Schuurmans. Thanks to Dekang, Randy, Greg, Dale, and Chris Westbury for their very helpful contributions as members of my thesis committee, and to my esteemed external, Eduard Hovy. Thank you to the members of the NLP group at the University of Alberta, including friends and co-authors Colin Cherry, Chris Pinchak, Qing Wang, and Sittichai Jiampojamarn. Thank you, Greg, for showing me how to squeeze extra text into my papers by using *vspace* in LaTeX. A special thanks also to Colin for hosting me at the NRC in Ottawa and helping me throughout my graduate career.

Thank you also to Kristin Musselman and Chris Pinchak for assistance preparing the non-referential pronoun data used in Chapter 3. Thank you to Thorsten Brants, Fernando Pereira, and everyone else at Google Inc. for sharing the Google N-gram data and for making my internships in Mountain View so special. Thank you to Frederick Jelinek and many others at the Center for Language and Speech Processing at Johns Hopkins University for hosting the workshop at which part of the Chapter 5 research was conducted, and thank you to the workshop sponsors. Thank you to the amazing Ken Church, David Yarowsky, Satoshi Sekine, Heng Ji, and my extremely capable fellow students on Team N-gram.

Finally, I provide acknowledgments in the form of a sample of some actual Google 5-gram counts. The frequency ranking is pretty good; I would only suggest that *wife* should be at the top. I would like to thank my . . .

like to thank my family	10068	like to thank my fans	258
like to thank my parents	5447	like to thank my son	245
like to thank my colleagues	4175	like to thank my guests	245
like to thank my wife	4006	like to thank my collaborators	244
like to thank my advisor	3819	like to thank my hairdresser	229
like to thank my supervisor	3779	like to thank my readers	214
like to thank my friends	2375	like to thank my daughter	174
like to thank my mother	1477	like to thank my coach	142
like to thank my committee	1068	like to thank my assistant	96
like to thank my husband	998	like to thank my boyfriend	90
like to thank my mom	872	like to thank my hands	87
like to thank my brother	660	like to thank my uncle	71
like to thank my father	619	like to thank my opponent	71
like to thank my sister	486	like to thank my buddies	67
like to thank my mum	441	like to thank my grandmother	54
like to thank my girlfriend	296	like to thank my computer	50

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Natural language processing (NLP) is a field that develops computational techniques for analyzing human language. NLP provides the algorithms for spelling correction, speech recognition, and automatic translation that are used by millions of people every day.

Recent years have seen an explosion in the availability of language in the form of electronic text. Web pages, e-mail, search-engine queries, and text-messaging have created a staggering and ever-increasing volume of language data. Processing this data is a great challenge. Users of the Internet want to find the right information quickly in a sea of irrelevant pages. Governments, businesses, and hospitals want to discover important trends and patterns in their unstructured textual records.

The challenge of unprecedented volumes of data also presents a significant opportunity. Online text is one of the largest and most diverse bodies of linguistic evidence ever compiled. We can use this evidence to train and test broad and powerful language-processing tools. In this dissertation, I explore ways to extract meaningful statistics from huge volumes of raw text, and I use these statistics to create intelligent NLP systems. Techniques from machine learning play a central role in this work; machine learning provides principled ways to combine linguistic intuitions with evidence from big data.

1.1 What NLP Systems Do

Before we discuss exactly how unlabeled data can help improve NLP systems, it is important to clarify exactly what modern NLP systems do and how they work. NLP systems take sequences of words as input and automatically produce useful linguistic annotations as output. Suppose the following sentence exists on the web somewhere:

- "The movie sucked."

Suppose you work for J.D. Power and Associates Web Intelligence Division. You create systems that automatically analyze blogs and other web pages to find out what people think about particular products, and then you sell this information to the producers of those products (and occasionally surprise them with the results). You might want to annotate the whole sentence for its sentiment: whether the sentence is positive or negative in its tone:

- "The movie sucked $\rightarrow \langle$Sentiment=*NEGATIVE*$\rangle$"

Or suppose you are Google, and you wish to translate this sentence for a German user. The translation of the word *sucked* is ambiguous. Here, it likely does not mean, "to be

drawn in by establishing a partial vacuum," but rather, "to be disagreeable." So another potentially useful annotation is word sense:

- "The movie sucked $\rightarrow$ The movie sucked$\langle$Sense=*IS-DISAGREEABLE*$\rangle$."

More directly, we might consider the German translation itself as the annotation:

- "The movie sucked $\rightarrow$ Der Film war schrecklich.

Finally, if we're the company Powerset, our stated objective is to produce "parse trees" for the entire web as a preprocessing step for our search engine. One part of parsing is to label the syntactic category of each word (i.e., which are nouns, which are verbs, etc.). The part-of-speech annotation might look as follows:

- "The movie sucked $\rightarrow$ The$\backslash$*DT* movie$\backslash$*NN* sucked$\backslash$*VBD*

Where *DT* means determiner, *NN* means a singular or mass noun, and *VBD* means a past-tense verb.[1] Again, note the potential ambiguity for the tag of *sucked*; it could also be labeled *VBN* (verb, past participle). For example, *sucked* is a VBN in the phrase, "the movie sucked into the vacuum cleaner was destroyed."

These outputs are just a few of the possible annotations that can be produced for textual natural language input. Other branches and fields of NLP may operate over speech signals rather than actual text. Also, in the natural language generation (NLG) community, the input may not be text, but information in another form, with the desired output being grammatically-correct English sentences. Most of the work in the NLP community, however, operates exactly in this framework: text comes in, annotations come out. But how does an NLP system produce these annotations automatically?

1.2 Writing Rules vs. Machine Learning

One might imagine writing some rules to produce these annotations automatically. For part-of-speech tagging, we might say, "if the word is *movie*, then label the word as *NN*." These word-based rules fail when the word can have multiple tags (e.g. *saw*, *wind*, etc. can be nouns or verbs). Also, no matter how many rules we write, there will always be new or rare words that didn't make our rule set. For ambiguous words, we could try to use rules that depend on the word's context. Such a rule might be, "if the previous word is *The* and the next word ends in *-ed*, then label as *NN*." But this rule would fail for "the Oilers skated," since here the tag is not *NN* but *NNPS*: a plural proper noun. We could change the rule to: "if the previous word is *The* and the next word ends in *-ed*, *and the word is lower-case*, then label as *NN*." But this would fail for "The begrudgingly viewed movie," where now "begrudgingly" is an adverb, not a noun. We might imagine adding many many more rules. Also, we might wish to attach scores to our rules, to principally resolve conflicting rules. We could say, "if the word is *wind*, give the score for being a *NN* a **ten** and for being a *VB* a **two**," and this score could be combined with other context-based scores, to produce a different cumulative score for each possible tag. The highest-scoring tag would be taken as the output.

[1] Refer to Appendix A for definitions and examples from the Penn Treebank tag set, the most commonly-used part-of-speech tag set.

These rules and scores might depend on many properties of the input sentence: the word itself, the surrounding words, the case, the prefixes and suffixes of the surrounding words, etc. The number of properties of interest (what in machine learning is called "the number of *features*") may be quite large, and it is difficult to choose the set of rules and weights that results in the best performance (See Chapter 2, Section 2.1 for further discussion).

Rather than specifying the rules and weights by hand, the current dominant approach in NLP is to provide a set of *labeled* examples that the system can *learn* from. That is, we *train* the system to make decisions using guidance from labeled data. By labeled data, we simply mean data where the correct, gold-standard answer has been explicitly provided. The properties of the input are typically encoded as numerical features. A score is produced using a weighted combination of the features. The learning algorithm assigns weights to the features so that the correct output scores higher than incorrect outputs on the training set. Or, in cases where the true output can not be generated by the system, so that the highest scoring output (the system prediction) is as close as possible to the known true answer.

For example, feature 96345 might be a binary feature, equal to one if "the word is *wind*," and otherwise equal to zero. This feature (e.g. f_{96345}) may get a high weight for predicting whether the word is a common noun, *NN* (e.g. the corresponding weight parameter, w_{96345}, may be 10). If the weighted-sum-of-features score for the *NN* tag is higher than the scores for the other tags, then *NN* is predicted. Again, the key point is that these weights are chosen, automatically, in order to maximize performance on human-provided, labeled examples. Chapter 2 covers the fundamental equations of machine learning (ML) and discusses how machine learning is used in NLP.

Statistical machine learning works a lot better than specifying rules by hand. ML systems are easier to develop (because a computer program fine-tunes the rules, not a human) and easier to adapt to new domains (because we need only annotate new data, rather than write new rules). ML systems also tend to achieve better performance (again, see Chapter 2, Section 2.1).[2]

The chief bottleneck in developing supervised systems is the manual annotation of data. Historically, most labeled data sets were created by experts in linguistics. Because of the great cost of producing this data, the size and variety of these data sets is quite limited.

Although the amount of labeled data is limited, there is quite a lot of unlabeled data available (as we mentioned above). This dissertation explores various methods to combine very large amounts of unlabeled data with standard supervised learning on a variety of NLP tasks. This combination of learning from both labeled and unlabeled data is often referred to as *semi-supervised learning*.

1.3 Learning from Unlabeled Data

An example from part-of-speech tagging will help illustrate how unlabeled data can be useful. Suppose we are trying to label the parts-of-speech in the following examples. Specifically, there is some ambiguity for the tag of the verb *won*.

(1) "He saw the Bears won yesterday."

[2]Machine learned systems are also more fun to design. At a talk last year at Johns Hopkins University (June, 2009), BBN employee Ralph Weischeidel suggested that one of the reasons that BBN switched to machine learning approaches was because one of their chief designers got so bored writing rules for their information extraction system, he decided to go back to graduate school.

(2) "He saw the trophy won yesterday."

(3) "He saw the boog won yesterday."

Only one word differs in each sentence: the word before the verb *won*. In Example 1, *Bears* is the **subject** of the verb *won* (it was the Bears **who** won yesterday). Here, *won* should get the *VBD* tag. In Example 2, *trophy* is the **object** of the verb *won* (it was the trophy **that was** won). In this sentence, *won* gets a *VBN* tag. In a typical training set (i.e. the training sections of the Penn Treebank [Marcus *et al.*, 1993]), we don't see *Bears won* or *trophy won* at all. In fact, both the words *Bears* and *trophy* are rare enough to essentially look like Example 3 to our system. They might as well be *boog*! Based on even a fairly large set of labeled data, like the Penn Treebank, the correct tag for *won* is ambiguous.

However, the relationship between *Bears* and *won*, and between *trophy* and *won*, is fairly unambiguous if we look at **unlabeled** data. For both pairs of words, I have collected all 2-to-5-grams where the words co-occur in the Google V2 corpus, a collection of N-grams from the entire world wide web. An N-gram corpus states how often each sequence of words (up to length N) occurs (N-grams are discussed in detail in Chapter 3, while the Google V2 corpus is described in Chapter 5; note the Google V2 corpus includes part-of-speech tags). I replace non-stopwords by their part-of-speech tag, and sum the counts for each pattern. The top fifty most frequent patterns for {*Bears, won*} and {*trophy, won*} are given:

Bears won:

- Bears won:3215
- the Bears won:1252
- Bears won the:956
- The Bears won:875
- Bears have won:874
- NNP Bears won:767
- Bears won their:443
- Bears won CD:436
- The Bears have won:328
- Bears won their JJ:321
- Bears have won CD:305
- , the Bears won:305
- the NNP Bears won:305
- The Bears won the:296
- the Bears won the:293
- The NNP Bears won:274
- NNP Bears won the:262
- the Bears have won:255
- NNP Bears have won:217
- as the Bears won:168
- the Bears won CD:168
- Bears won the NNP:162
- Bears have won 00:160
- Bears won the NN:157
- Bears won a:153
- the Bears won their:148
- NNP Bears won their:129
- The Bears have won CD:128
- Bears won ,:124
- Bears had won:121
- The Bears won their:121
- when the Bears won:119
- The NNP Bears have won:117
- Bears have won the:112
- Bears won the JJ:112
- Bears , who won:107
- The Bears won CD:103
- Bears won the NNP NNP:102
- The NNP Bears won the:100
- the NNP Bears won the:96
- Bears have RB won:94
- , the Bears have won:93
- and the Bears won:91
- IN the Bears won:89
- Bears also won:87
- Bears won 00:86
- Bears have won CD of:84
- as the NNP Bears won:80
- Bears won CD .:80
- , the Bears won the:77

trophy won:

- won the trophy:4868
- won a trophy:2770
- won the trophy for:1375
- won the JJ trophy:825
- trophy was won:811
- trophy won:803

- won a trophy for:689
- won the trophy for the:631
- trophy was won by:626
- won a JJ trophy:513
- won the trophy in:511
- won the trophy.:493
- RB won a trophy:439
- trophy they won:421
- won the NN trophy:405
- trophy won by:396
- have won the trophy:396
- won this trophy:377
- the trophy they won:329
- won the NNP trophy:325
- won a trophy .:313
- won the trophy NN:295
- trophy he won:292
- has won the trophy:290
- won the trophy for JJS:284
- won a trophy in:274
- won the trophy in 0000:272
- won the JJ NN trophy:267
- won a trophy and:249
- RB won the trophy:242
- who won the trophy:242
- and won the trophy:240
- won the trophy,:228
- won a trophy ,:215
- won a trophy at:199
- , won the trophy:191
- also won the trophy:189
- had won the trophy:186
- won DT trophy:184
- and won a trophy:178
- the trophy won:173
- won their JJ trophy:169
- JJ trophy:168
- won the trophy RB:161
- won the JJ trophy in:155
- won a JJ NN trophy:155
- I won a trophy:153
- won the trophy CD:145
- won the trophy and:141
- trophy , won:141

In this data, *Bears* is almost always the *subject* of the verb, occurring before *won* and with an object phrase afterwards (like *won the* or *won their*, etc.). On the other hand, *trophy* almost always appears as an object, occurring after *won* or in passive constructions (*trophy was won*, *trophy won by*) or with another noun in the subject role (*trophy they won*, *trophy he won*). If, on the web, a pair of words tends to occur in a particular relationship, then for an ambiguous instance of this pair at test time, it is reasonable to also predict this relationship.

Now think about *boog*. A lot of words look like *boog* to a system that has only seen limited labeled data. Now, if *globally* the words *boog* and *won* occur in the same patterns in which *trophy* and *won* occur, then it would be clear that *boog* is also usually the object of *won*, and thus *won* is likely a past participle (*VBN*) in Example 3. If, on the other hand, *boog* occurs in the same patterns as *Bears*, we would consider it a subject, and label *won* as a past-tense verb (*VBD*).[3]

So, in summary, while a pair of words, like *trophy* and *won*, might be very rare in our labeled data, the patterns in which these words occur (the **distribution** of the words), like *won the trophy*, and *trophy was won*, may be very indicative of a particular relationship. These indicative patterns will likely be shared by other pairs in the labeled training data (e.g., we'll see global patterns like *bought the securities*, *market was closed*, etc. for labeled examples like "the securities bought by" and "the market closed up 134 points"). So, we supplement our sparse information (the identity of individual words) with more-general information (statistics from the distribution of those words on the web). The word's global distribution can provide features just like the features taken from the word's *local* context. By *local*, I mean the contextual information surrounding the words to be classified in a given sentence. Combining local and global sources of information together, we can achieve higher performance.

Note, however, that when the local context is *un*ambiguous, it is usually a better bet to rely on the local information over the global, distributional statistics. For example, if the

[3]Of course, it might be the case that *boog* and *won* don't occur in unlabeled data either, in which case we might back off to even more general global features, but we leave this issue aside for the moment.

actual sentence said, “My son’s simple trophy won their hearts,” then we should guess VBD for *won*, regardless of the global distribution of *trophy won*. Of course, we let the learning algorithm choose the relative weight on global vs. local information. In my experience, when good local features are available, the learning algorithm will usually put most of the weight on them, as the algorithm finds these features to be statistically more reliable. So we must lower our expectations for the possible benefits of purely distributional information. When there are already other good sources of information available locally, the effect of global information is diminished. Section 5.6 presents some experimental results on VBN-VBD disambiguation and discusses this point further.

Using N-grams for Learning from Unlabeled Data

In our work, we make use of aggregate counts over a large corpus; we don’t inspect the individual instances of each phrase. That is, we do not separately process the 4868 sentences where “*won the trophy*” occurs on the web, rather we use the N-gram, *won the trophy*, and its count, 4868, as a single unit of information. We do this mainly because it’s computationally inefficient to process all the instances (that is, the entire web). Very good inferences can be drawn from the aggregate statistics. Chapter 2 describes a range of alternative methods for exploiting unlabeled data; many of these can not scale to web-scale text.

1.4 A Perspective on Statistical vs. Linguistic Approaches

When reading any document, it can be useful to think about the author’s perspective. Sometimes, when we establish the author’s perspective, we might also establish that the document is not worth reading any further. This might happen, for example, if the author’s perspective is completely at odds with our own, or if it seems likely the author’s perspective will prevent them from viewing evidence objectively.

Surely, some readers of this document are also wondering about the perspective of its author. Does he approach language from a purely statistical viewpoint, or is he interested in linguistics itself? The answer: Although I certainly advocate the use of statistical methods and huge volumes of data, I am mostly interested in how these resources can help with real linguistic phenomena. I agree that linguistics has an essential role to play in the future of NLP [Jelinek, 2005; Hajič and Hajičová, 2007]. I aim to be aware of the knowledge of linguists and I try to think about where this knowledge might apply in my own work. I try to gain insight into problems by annotating data myself. When I tackle a particular linguistic phenomenon, I try to think about how that phenomenon serves human communication and thought, how it may work differently in written or spoken language, how it may work differently across human languages, and how a particular computational representation may be inadequate. By doing these things, I hope to not only produce more interesting and insightful research, but to produce systems that work better. For example, while a search on Google Scholar reveals a number of papers proposing “language independent” approaches to tasks such as named-entity recognition, parsing, grapheme-to-phoneme conversion, and information retrieval, it is my experience that approaches that pay attention to language-specific issues tend to work better (e.g., in transliteration [Jiampojamarn *et al.*, 2010]). In fact, exploiting linguistic knowledge can even help the Google statistical translation system [Xu *et al.*, 2009] – a system that is often mentioned as an example of a purely data-driven NLP approach.

On the other hand, mapping language to meaning is a very hard task, and statistical tools help a lot too. It does not seem likely that we will solve the problems of NLP anytime soon. Machine learning allows us to make very good predictions (in the face of uncertainty) by combining multiple, individually inadequate sources of evidence. Furthermore, it is empirically very effective to make predictions based on something previously observed (say, on the web), rather than trying to interpret everything purely on the basis of a very rich linguistic (or multi-modal) model. The observations that we rely on can sometimes be subtle (as in the verb tagging example from Section 1.3) and sometimes obvious (e.g., just count which preposition occurs most frequently in a given context, Section 3.5). Crucially, even if our systems do not really model the underlying linguistic (and other mental) processes,[4] such predictions may still be quite useful for real applications (e.g., in speech, machine translation, writing aids, information retrieval, etc.). Finally, once we understand what can be solved trivially with big data and machine learning, it might better help us focus our attention on the appropriate deeper linguistic issues; i.e., the *long tail* of linguistic behaviour predicted by Zipf's law. Of course, we need to be aware of the limitations of N-gram models and big data, because, as Mark Steedman writes [Steedman, 2008]:

> "One day, either because of the demise of Moore's law, or simply because we have done all the easy stuff, the Long Tail will come back to haunt us."

Not long ago, many in our community were dismissive of applying large volumes of data and machine learning to linguistic problems at all. For example, IBM's first paper on statistical machine translation was met with a famously (anonymous) negative review (1988) (quoted in [Jelinek, 2009]):

> "The crude force of computers is not science. The paper is simply beyond the scope of COLING."

Of course, statistical approaches are now clearly dominant in NLP (see Section 2.1). In fact, what is interesting about the field of NLP today is the growing concern that our field is now *too* empirical. These concerns even come from researchers that were the leaders of the shift to statistical methods. For example, an upcoming talk at COLING 2010 by Ken Church and Mark Johnson discusses the topic, "The Pendulum has swung too far. The revival of empiricism in the 1990s was an exciting time. But now there is no longer much room for anything else."[5] Richard Sproat adds:[6]

> "... the field [of computational linguistics] has devolved in large measure into a group of technicians who are more interested in tweaking the techniques than in the problems they are applied to; who are far more impressed by a clever new ML approach to an old problem, than the application of known techniques to a new problem."

Although my own interests lie in both understanding linguistic problems and in "tweaking" ML techniques, I don't think *everyone* need approach NLP the same way. We need

[4] Our models obviously do not reflect real human cognition since humans do not have access to the trillions of pages of data that we use to train our models. The main objective of this dissertation is to investigate what kinds of useful and scientifically interesting things we can do with computers. In general, my research aims to exploit models of human linguistic processing where possible, as opposed to trying to replicate them.

[5] `http://nlp.stanford.edu/coling10/full-program.html#ring`

[6] `http://www.cslu.ogi.edu/~sproatr/newindex/ncfom.html`

	Uses Web-Scale N-grams			Auto-Creates Examples	
Problem	Ch. 3	Ch. 4	Ch. 5	Ch. 6	Ch. 7
Preposition selection	§ 3.5	§ 4.4.1			
Context-sensitive spelling correction	§ 3.6	§ 4.4.2	§ 5.4		
Non-referential pronoun detection	§ 3.7	§ 4.4.3			
Adjective ordering			§ 5.3		
Noun-compound bracketing			§ 5.5		
VBN-VBD disambiguation			§ 5.6		
Selectional preference				Ch. 6	
Pronoun resolution				§ 6.4.6	
Cognate identification					Ch. 7

Table 1.1: Summary of tasks handled in the dissertation, with pointers to relevant sections, divided by the main method applied (using web-scale N-gram features or automatic creation of training examples)

both tweakers and theorists, *do*ers and *think*ers, those that try to solve everything using ML/big data, and those that feel data-driven successes are ultimately preventing us from solving the real problems. Supporting a diversity of views may be one way to ensure universally better funding for NLP research in the future [Steedman, 2008].

I hope that people from a variety of perspectives will find something they can appreciate in this dissertation.

1.5 Overview of the Dissertation

Chapter 2 provides an introduction to machine learning in NLP, and gives a review of previous supervised and semi-supervised approaches related to this dissertation. The remainder of the dissertation can be divided into two parts that span Chapters 3-5 and Chapters 6-7, respectively. Each chapter is based on a published paper, and so is relatively self-contained. However, reading Chapter 3 first will help clarify Chapter 5 and especially Chapter 4.

We now summarize the specific methods used in each chapter. For easy reference, Table 1.1 also lists the tasks that are evaluated in each of these chapters.

Using Unlabeled Statistics as Features

In the first part of the dissertation, we propose to use the counts of unlabeled patterns as features in supervised systems trained on varying amounts of labeled data. In this part of the dissertation, the unlabeled counts are taken from web-scale N-gram data. Web-scale data has previously been used in a diverse range of language research, but most of this research has used web counts for only short, fixed spans of context. Chapter 3 proposes a unified view of using web counts for lexical disambiguation. We extract the surrounding textual context of a word to be classified and gather, from a large corpus, the distribution of words that occur within that context. Unlike many previous approaches, our supervised and unsupervised systems combine information from multiple and overlapping segments of context. On the tasks of preposition selection and context-sensitive spelling correction, the supervised system reduces disambiguation error by 20-24% over current, state-of-the-art

web-scale systems. This work was published in the proceedings of IJCAI-09 [Bergsma *et al.*, 2009b]. This same method can also be used to determine whether a pronoun in text refers to a preceding noun phrase or is instead *non-referential*. This is the first system for non-referential pronoun detection where all the key information is derived from unlabeled data. The performance of the system exceeds that of (previously dominant) rule-based approaches. The work on non-referential *it* detection was first published in the proceedings of ACL-08: HLT [Bergsma *et al.*, 2008b].

Chapter 4 improves on the lexical disambiguation classifiers of Chapter 3 by using a simple technique for learning better support vector machines (SVMs) using fewer training examples. Rather than using the standard SVM regularization, we regularize toward low weight-variance. Our new SVM objective remains a convex quadratic function of the weights, and is therefore computationally no harder to optimize than a standard SVM. Variance regularization is shown to enable dramatic improvements in the learning rates of SVMs on the three lexical disambiguation tasks that we also tackle in Chapter 3. A version of this chapter was published in the proceedings of CoNLL 2010 [Bergsma *et al.*, 2010b]

Chapter 5 looks at the effect of combining web-scale N-gram features with standard, lexicalized features in supervised classifiers. It extends the work in Chapter 3 both by tackling new problems and by simultaneously evaluating these two very different feature classes. We show that including N-gram count features can advance the state-of-the-art accuracy on standard data sets for adjective ordering, spelling correction, noun compound bracketing, and verb part-of-speech disambiguation. More importantly, when operating on new domains, or when labeled training data is not plentiful, we show that using web-scale N-gram features is essential for achieving robust performance. A version of this chapter was published in the proceedings of ACL 2010 [Bergsma *et al.*, 2010c].

Using Unlabeled Statistics to Generate Training Examples

In the second part of the dissertation, rather than using the unlabeled statistics solely as features, we use them to generate labeled examples. By automatically labeling a large number of examples, we can train powerful discriminative models, leveraging fine-grained features of input words.

Chapter 6 shows how this technique can be used to learn selectional preferences. Models of selectional preference are essential for resolving syntactic, word-sense, and reference ambiguity, and models of selectional preference have received a lot of attention in the NLP community. We turn selectional preference into a supervised classification problem by asking our classifier to predict which predicate-argument pairs should have high association in text. Positive examples are taken from observed predicate-argument pairs, while negatives are constructed from unobserved combinations. We train a classifier to distinguish the positive from the negative instances. Features are constructed from the distribution of the argument in text. We show how to partition the examples for efficient training with 57 thousand features and 6.5 million training instances. The model outperforms other recent approaches, achieving excellent correlation with human plausibility judgments. Compared to mutual information, our method identifies 66% more verb-object pairs in unseen text, and resolves 37% more pronouns correctly in a pronoun resolution experiment. This work was originally published in EMNLP 2008 [Bergsma *et al.*, 2008a].

In Chapter 7, we apply this technique to learning a model of string similarity. A character-based measure of similarity is an important component of many natural language processing systems, including approaches to transliteration, coreference, word alignment,

spelling correction, and the identification of cognates in related vocabularies. We turn string similarity into a classification problem by asking our classifier to predict which bilingual word pairs are translations. Positive pairs are generated automatically from words with a high association in an aligned bitext, or mined from dictionary translations. Negatives are constructed from pairs with a high amount of character overlap, but which are not translations. We gather features from substring pairs consistent with a character-based alignment of the two strings. The main objective of this work was to demonstrate a better model of string similarity, not necessarily to demonstrate our method for generating training examples, however the overall framework of this work fits in nicely with this dissertation. Our model achieves exceptional performance; on nine separate cognate identification experiments using six language pairs, we more than double the average precision of traditional orthographic measures like longest common subsequence ratio and Dice's coefficient. We also show strong improvements over other recent discriminative and heuristic similarity functions. This work was originally published in the proceedings of ACL 2007 [Bergsma and Kondrak, 2007a].

1.6 Summary of Main Contributions

The main contribution of Chapter 3 is to show that we need not restrict ourselves to very limited contextual information simply because we are working with web-scale volumes of text. In particular, by using web-scale N-gram data (as opposed to, for example, search engine data), we can:

- combine information from multiple, overlapping sequences of context of varying lengths, rather than using a single context pattern (Chapter 3), and
- apply either discriminative techniques or simple unsupervised algorithms to integrate information from these overlapping contexts (Chapter 3).

We also make useful contributions by showing how to:

- detect non-referential pronouns by looking at the distribution of fillers that occur in pronominal context patterns (Section 3.7),
- modify the SVM learning algorithm to be biased toward a solution that is *a priori* known to be effective, whenever features are based on counts (Chapter 4), and
- operate on new domains with far greater robustness than approaches that simply use standard lexical features (Chapter 5).
- exploit preprocessing of web-scale N-gram data, either via part-of-speech tags added to the source corpus (Chapter 5), or by truncating/stemming the N-grams themselves (Section 3.7).

The technique of automatically generating training examples has also been used previously in NLP. Our main contributions are showing:

- very clean pseudo-examples can be generated from aggregate statistics rather than individual words or sentences in text, and

- since many more training examples are available when examples are created automatically, we can exploit richer, more powerful, more fine-grained features for a range of problems, from semantics (Chapter 6) to string similarity (Chapter 7).

The new features we proposed include the first use of character-level (string and capitalization) features for selectional preferences (Chapter 6), and the first use of alignment in discriminative string similarity (Chapter 7).

I really do hope you enjoy finding out more about these and the other contributions of this dissertation!

Chapter 2

Supervised and Semi-Supervised Machine Learning in Natural Language Processing

> "We shape our tools. And then our tools shape us."
> - Marshall McLuhan

This chapter outlines the key concepts from machine learning that are used in this dissertation. Section 2.1 provides some musings on why machine learning has risen to be such a dominant force in NLP. Section 2.2 introduces the *linear classifier*, the fundamental statistical model that we use in all later chapters of the dissertation. Section 2.2, and the following Section 2.3, address one important goal of this chapter: to present a simple, detailed explanation of how the tools of supervised machine learning can be used in NLP. Sections 2.4 and 2.5 provide a higher-level discussion of related approaches to unsupervised and semi-supervised learning. In particular, these sections relate past trends in semi-supervised learning to the models presented in the dissertation.

2.1 The Rise of Machine Learning in NLP

It is interesting to trace the historical development of the statistical techniques that are so ubiquitous in NLP today. The following mostly relies on the brief historical sketch in Chapter 1 of Jurafsky and Martin's textbook [Jurafsky and Martin, 2000], with insights from [Church and Mercer, 1993; Manning and Schütze, 1999; Jelinek, 2005; Fung and Roth, 2005; Hajič and Hajičová, 2007].

The foundations of speech and language processing lie in the 1940s and 1950s, when finite-state machines were applied to natural language by Claude Shannon [1948], and subsequently analyzed as a formal language by Noam Chomsky [1956]. During the later 1950s and 1960s, speech and language processing had split into two distinct lines of research: logic-based "symbolic" methods and probabilistic "stochastic" research.

Researchers in the symbolic tradition were both pursuing computational approaches to formal language theory and syntax, and also working with natural language in the logic and reasoning framework then being developed in the new field of artificial intelligence. From about 1960 to 1985, stochastic approaches were generally out-of-favour, and remain

so within some branches of psychology, linguistics and artificial intelligence even today. Manning and Schütze believe that

> "much of the skepticism towards probabilistic models for language (and cognition in general) stems from the fact that the well-known early probabilistic models (developed in the 1940s and 1950s) are extremely simplistic. Because these simplistic models clearly do not do justice to the complexity of human language, it is easy to view probabilistic models in general as inadequate."

The stochastic paradigm became much more influential again after the 1970s and early 1980s when N-gram models were successfully applied to speech recognition by the IBM Thomas J. Watson Research Center [Jelinek, 1976; Bahl *et al.*, 1983] and by James Baker at Carnegie Mellon University [Baker, 1975]. Previous efforts in speech recognition had been rather "*ad hoc* and fragile, and were demonstrated on only a few specially selected examples" [Russell and Norvig, 2003]. The work by Jelinek and others soon made it apparent that data-driven approaches simply *work better*. As Hajič and Hajičová [2007] summarize:

> "[The] IBM Research group under Fred Jelinek's leadership realized (and experimentally showed) that linguistic rules and Artificial Intelligence techniques had inferior results even when compared to very simplistic statistical techniques. This was first demonstrated on phonetic baseforms in the acoustic model for a speech recognition system, but later it became apparent that this can be safely assumed almost for every other problem in the field (e.g., Jelinek [1976]). Statistical learning mechanisms were apparently and clearly superior to any human-designed rules, especially those using any preference system, since humans are notoriously bad at estimating quantitative characteristics in a system with many parameters (such as a natural language)."

Probabilistic and machine learning techniques such as decision trees, clustering, EM, and maximum entropy gradually became the foundation of speech processing [Fung and Roth, 2005]. The successes in speech then inspired a range of empirical approaches to natural language processing. Simple statistical techniques were soon applied to part-of-speech tagging, parsing, machine translation, word-sense disambiguation, and a range of other NLP tasks. While there was only one statistical paper at the ACL conference in 1990, virtually all papers in ACL today employ statistical techniques [Hajič and Hajičová, 2007].

Of course, the fact that statistical techniques currently work better is only partly responsible for their rise to prominence. There was a fairly large gap in time between their proven performance on speech recognition and their widespread acceptance in NLP. Advances in computer technology and the greater availability of data resources also played a role. According to Church and Mercer [1993]:

> "Back in the 1970s, the more data-intensive methods were probably beyond the means of many researchers, especially those working in universities... Fortunately, as a result of improvements in computer technology and the increasing availability of data due to numerous data collection efforts, the data-intensive methods are no longer restricted to those working in affluent industrial laboratories."

Two other important developments were the practical application and commercialization of NLP algorithms and the emphasis that was placed on empirical evaluation. A greater

emphasis on "deliverables and evaluation" [Church and Mercer, 1993] created a demand for robust techniques, empirically-validated on held-out data. Performance metrics from speech recognition and information retrieval were adopted in many NLP sub-fields. People stopped evaluating on their training set, and started using standard test sets. Machine learning researchers, always looking for new sources of data, began evaluating their approaches on natural language, and publishing at high-impact NLP conferences.

Flexible *discriminative* ML algorithms like maximum entropy [Berger *et al.*, 1996] and conditional random fields [Lafferty *et al.*, 2001] arose as natural successors to earlier statistical techniques like naive Bayes and hidden Markov models (*generative* approaches; Section 2.3.3). Indeed, since machine learning algorithms, especially discriminative techniques, could be specifically tuned to optimize a desired performance metric, ML systems achieved superior performance in many competitions and evaluations. This has led to a shift in the overall speech and language processing landscape. Originally, progress in statistical speech processing inspired advances in NLP; today many ML algorithms (such as structured perceptrons and support vector machines) were first developed for NLP and information retrieval applications and then later applied to speech tasks [Fung and Roth, 2005].

In the initial rush to adopt statistical techniques, many NLP tasks were decomposed into sub-problems that could be solved with well-understood and readily-available binary classifiers. In recent years, NLP systems have adopted more sophisticated ML techniques. These algorithms are now capable of producing an entire annotation (like a parse-tree or translation) as a single global output, and suffer less from the propagation of errors common in a pipelined, local-decision approach. These so-called "structured prediction" techniques include conditional random fields [Lafferty *et al.*, 2001], structured perceptrons [Collins, 2002], structured SVMs [Tsochantaridis *et al.*, 2004], and rerankers [Collins and Koo, 2005]. Others have explored methods to produce globally-consistent structured output via linear programming formulations [Roth and Yih, 2004]. While we have also had success in using global optimization techniques like integer linear programming [Bergsma and Kondrak, 2007b] and re-ranking [Dou *et al.*, 2009], the models used in this dissertation are relatively simple linear classifiers, which we discuss in the following section. This dissertation focuses on a) developing better features and b) automatically producing more labeled examples. The advances we make are also applicable when using more sophisticated learning methods.

Finally, we note that recent years have also seen a strong focus on the development of *semi-supervised* learning techniques for NLP. This is also the focus of this dissertation. We describe semi-supervised approaches more generally in Section 2.5.

2.2 The Linear Classifier

A linear classifier is a very simple, unsophisticated concept. We explain it in the context of text categorization, which will help make the equations more concrete for the reader. Text categorization is the problem of deciding whether an input document is a member of a particular category or not. For example, we might want to classify a document as being about *sports* or not.

Let's refer to the input as d. So for text categorization, d is a document. We want to decide if d is about sports or not. On what shall we base this decision? We always base the decision on some **features** of the input. For a document, we base the decision on the words in the document. We define a **feature function** $\mathbf{\Phi}(d)$. This function takes the input d and

produces a **feature vector**. A vector is just a sequence of numbers, like $(0, 34, 2.3)$. We can think of a vector as having multiple *dimensions*, where each dimension is a number in the sequence. So 0 is in the first dimension of $(0, 34, 2.3)$, 34 is in the second dimension, and 2.3 is in the third dimension. For text categorization, each dimension might correspond to a particular word (although character-based representations are also possible [Lodhi *et al.*, 2002]). The **value** at that dimension could be 1 if the word is present in the document, and 0 otherwise. These are **binary** feature values. We sometimes say that a feature *fires* if that feature value is non-zero, meaning, for text categorization, that the word is present in the document. We also sometimes refer to the feature vector as the **feature representation** of the problem.

In machine learning, the feature vector is usually denoted as $\bar{x}$, so $\bar{x} = \mathbf{\Phi}(d)$.

A simple feature representation would be to have the first dimension be for the presence of the word *the*, the second dimension for the presence of *curling*, and the third for the presence of *Obama*. If the document read only "Obama attended yesterday's curling match," then the feature vector would be $(0, 1, 1)$. If the document read "stocks are up today on Wall Street," then the feature vector would be $(0, 0, 0)$. Notice the order of the words in the text doesn't matter. "Curling went Obama" would have the same feature vector as "Obama went curling." So this is sometimes referred to as the **bag-of-words** feature representation. That's not really important but it's a term that is often seen in **bold text** when describing machine learning.

The **linear classifier**, $h(\bar{x})$, works by multiplying the feature vector, $\bar{x} = (x_1, x_2, ...x_N)$ by a set of learned weights, $\bar{w} = (w_1, w_2, ...)$:

$$h(\bar{x}) = \bar{w} \cdot \bar{x} = \sum_i w_i x_i \tag{2.1}$$

where the **dot product** $(\cdot)$ is a mathematical shorthand meaning, as indicated, that each w_i is multiplied with the feature value at dimension i and the results are summed. We can also write a dot product using matrix notation as $\bar{w}^T \bar{x}$. A linear classifier using an N-dimensional feature vector will sum the products of N multiplications. It's known as a linear classifier because this is a **linear combination** of the features. Note, sometimes the weights are also represented using $\boldsymbol{\lambda} = (\lambda_1, ...\lambda_N)$. This is sometimes convenient in NLP when we might want to use w to refer to a word.

The objective of the linear classifier is to produce **labels** on new examples. Labels are almost always represented as y. We choose the label using the output of the linear classifier. In a common paradigm, if the output is positive, that is, $h(\bar{x}) > 0$, then we take this as a positive decision: yes, the document d *does* belong to the sports category, so the label, y equals $+1$ (the positive class). If $h(\bar{x}) < 0$, we say the document does not belong in the sports category, and $y = -1$ (the negative class).

Now, the job of the machine learning algorithm is to learn these weights. That's really it. In the context of the widely-used linear classifier, the weights fully define the classifier. **Training** means choosing the weights, and **testing** means computing the dot product with the weights for new feature vectors. How does the algorithm actually choose the weights? In supervised machine learning, you give some examples of feature vectors and the correct decision on the vector. The index of each training example is usually written as a superscript, so that a training set of M examples can be written as: $\{(\bar{x}^1, y^1), ..., (\bar{x}^M, y^M)\}$. For example, a set of two training examples might be $\{(0, 1, 0), +1\}$, $\{(1, 0, 0), -1\}$ for a positive $(+1)$ and a negative (-1) example. The algorithm tries to choose the **parameters** (a synonym for the weights, $\bar{w}$) that result in the correct decision on this training data

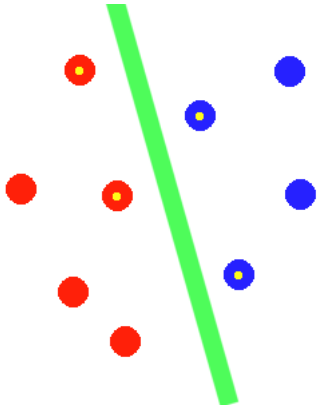


Figure 2.1: The linear classifier hyperplane (as given by an SVM, with support vectors indicated)

when the dot product is computed (here between three weights and three features). For our *sports* example, we would hope that the algorithm would learn, for example, that *curling* should get a positive weight, since documents that contain the word *curling* are usually about sports. It should assign a fairly low weight, perhaps zero weight, to the word *the*, since this word doesn't have much to say one way or the other. Choosing an appropriate weight for the *Obama* feature is left as an exercise for the reader. Note that weights can be negative. Section 2.3 has more details on some of the different algorithms that learn the weights.

If we take a geometric view, and think of the feature vectors as points in N-dimensional space, then learning the weights can also be thought of as learning a separating hyperplane. Once we have any classifier, then all feature vectors that get positive scores will be in one region of space, and all the feature vectors that get negative scores will be in another. With a linear classifier, a hyperplane will divide these two regions. Figure 2.1 depicts this set-up in two dimensions, with the points of one class on the left, the points for the other class on the right, and the dividing hyperplane as a bar down the middle.[1]

In this discussion, we've focused on binary classification: is the document about *sports* or not? In many practical applications, however, we have more than two categories, e.g. *sports*, *finance*, *politics*, etc. It's fairly easy to adapt the binary linear classifier to the **multiclass** case. For K classes, one common approach is the **one-versus-all** strategy: we have K binary classifiers that each predict whether a document is part of a given category or not. Thus we might classify a document about Obama going curling as both a *sports* and a *politics* document. In cases where only one category is possible (i.e., the classes are mutually exclusive, such as the restriction that each word have only one part-of-speech tag), we could take the highest-scoring classifier (the highest $h(\bar{x})$) as the class. There are also multiclass classifiers, like the approach we use in Chapter 3, that essentially jointly optimize the K classifiers (e.g. [Crammer and Singer, 2001]). Chapter 4 defines and evaluates various multi-class learning approaches.

A final point to address: should we be using a linear classifier for our problems at all? Linear classifiers are very simple, extremely fast, and work very well on a range of

[1] From: `www.stat.columbia.edu/~cook/movabletype/archives/2006/02/interesting_cas_1.html`

problems. However, they do not work well in all situations. Suppose we wanted a binary classifier to tell us whether a given Canadian city is either in Manitoba or not in Manitoba. Suppose we had only one feature: distance from the Pacific ocean. It would be difficult to choose a weight and a threshold for a linear classifier such that we could separate Manitoban cities from other Canadian cities with only this one feature. If we took cities below a threshold, we could get all cities west of Ontario. If we took those above, we could get all cities east of Saskatchewan. We would say that the positive and negative examples are not **separable** using this feature representation; the positives and negatives can't be placed nicely onto either side of a hyperplane. There are lots of non-linear classifiers to choose from that might do better.

On the other hand, we're always free to choose whatever feature function we like; for most problems, we can just choose a feature space that does work well with linear classifiers (i.e., a feature space that perhaps does make the training data separable). We could divide *distance-from-Pacific-ocean* into multiple features: say, a binary feature if the distance is between 0 and 100 km, another if it's between 100 and 200, etc. Also, many learning algorithms permit us to use the **kernel trick**, which maps the feature vectors into an implicit higher-dimensional space where a linear hyperplane can better divide the classes. We return to this point briefly in the following section. For many natural language problems, we have thousands of relevant features, and good classification is possible with linear classifiers. Generally, the more features, the more separable the examples.

2.3 Supervised Learning

In this section, we provide a very practical discussion of how the parameters of the linear classifier are chosen. This is the NLP view of machine learning: what you need to know to use it as a tool.

2.3.1 Experimental Set-up

The proper set-up is to have at least three sets of labeled data when designing a supervised machine learning system. First, you have a **training set**, which you use to learn your **model** (yet another word that means the same thing as the weights or parameters: the model is the set of weights). Secondly, you have a **development set**, which serves two roles: a) you can set any of your algorithm's hyperparameters on this set (hyperparameters are discussed below), and b) you can test your system on this set as you are developing. Rather than having a single development set, you could optimize your parameters by ten-fold cross validation on the training set, essentially re-using the training data to set development parameters. Finally, you have a **hold-out set** or **test set** of unseen data which you use for your final evaluation. You only evaluate on the test set once, to generate the final results of your experiments for your paper. This simulates how your algorithm would actually be used in practice: classifying data it has not seen before.

To run machine learning in this framework, we typically begin by converting the three sets into feature vectors and labels. We then supply the training set, in labeled feature vector format, to a standard software package, and this package returns the weights. The package can also be used to multiply the feature vectors by the weights, and return the classification decisions for new examples. It thus can and often does calculate performance on the development or test sets for you.

The above experimental set-up is sometimes referred to as a **batch learning** approach, because the algorithm is given the entire training set at once. A typical algorithm learns a single, static model using the entire training set in one training session (remember: for a linear classifier, by model we just mean the set of weights). This is the approach taken by SVMs and maximum entropy models. This is clearly different than how humans learn; we adapt over time as new data is presented. Alternatively, an **online learning** algorithm is one that is presented with training examples in sequence. Online learning iteratively re-estimates the model each time a new training instance is encountered. The perceptron is the classic example of an online learning approach, while currently MIRA [Crammer and Singer, 2003; Crammer *et al.*, 2006] is a popular maximum-margin online learner (see Section 2.3.4 for more on max-margin classifiers). In practice, there is little difference between how batch and online learners are used; if new training examples become available to a batch learner, the new examples can simply be added to the existing training set and the model can be re-trained on the old-plus-new combined data as another batch process.

It is also worth mentioning another learning paradigm known as **active learning** [Cohn *et al.*, 1994; Tong and Koller, 2002]. Here the learner does not simply train passively from whatever labeled data is available, rather, the learner can request specific examples be labeled if it deems adding these examples to the training set will most improve the classifier's predictive power. Active learning could potentially be used in conjunction with the techniques in this dissertation to get the most benefit out of the smallest amount of training data possible.

2.3.2 Evaluation Measures

Performance is often evaluated in terms of **accuracy**: what percentage of examples did we classify correctly? For example, if our decision is whether a document is about *sports* or not (i.e., *sports* is the **positive class**), then accuracy is the percentage of documents that are correctly labeled as *sports* or *non-sports*. Note it is difficult to compare accuracy of classifiers across tasks, because typically the class balance strongly affects the achievable accuracy. For example, suppose there are 100 documents in our test set, and only five of these are *sports* documents. Then a system could trivially achieve 95% accuracy by assigning every document the *non-sports* label. 95% might be much harder to obtain on another task with a 50-50 balance of the positive and negative classes. Accuracy is most useful as a measure when the performance of the proposed system is compared to a **baseline**: a reasonable, simple and perhaps even trivial classifier, such as one that picks the **majority-class** (the most frequent class in the training data). We use baselines whenever we state accuracy in this dissertation.

Accuracy also does not tell us whether our classifier is predicting one class disproportionately more often than another (that is, whether it has a **bias**). Statistical measures that do identify classifier biases are **Precision**, **Recall**, and **F-Score**. These measures are used together extensively in classifier evaluation.[2] Again, suppose *sports* is the class we're predicting. *Precision* tells us: *of the documents that our classifier predicted to be sports, what percentage are actually sports*? That is, precision is the ratio of true positives (elements we predicted to be of the positive class that truly are positive, where *sports* is the positive class in our running example), divided by the sum of true positives and false positives (to-

[2]Wikipedia has a detailed discussion of these measures: `http://en.wikipedia.org/wiki/Precision_and_recall`

		true class	
		+1	-1
predicted class	+1	TP	FP
	-1	FN	TN

Table 2.1: The classifier confusion matrix. Assuming "1" is the positive class and "-1" is the negative class, each instance assigned a class by a classifier is either a true positive (TP), false positive (FP), false negative (FN), or true negative (TN), depending on its actual class membership (true class) and what was predicted by the classifier (predicted class).

gether, all the elements that we predicted to be members of the positive class). *Recall*, on the other hand, tells us *the percentage of actual sports documents that were also predicted by the classifier to be sports documents*. That is, recall is the ratio of true positives divided by the number of true positives plus the number of false negatives (together, all the true, gold-standard positives). It is possible to achieve 100% recall on any task by predicting all instances to be of the positive class (eliminating false negatives). In isolation, therefore, precision or recall may not be very informative, and so they are often stated together. For a single performance number, precision and recall are often combined into the F-score, which is simply the harmonic mean of precision and recall.

We summarize these measures using Table 2.1 and the following equations:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.3.3 Supervised Learning Algorithms

We want a learning algorithm that will give us the best accuracy on our evaluation data – how do we choose it? As you might imagine, there are many different ways to choose the weights. Some algorithms are better suited to some situations than others. There are **generative** models like naive bayes that work well when you have smaller amounts of training data [Ng and Jordan, 2002]. Generative approaches jointly model both the input and output variables in a probabilistic formulation. They require one to explicitly model the interdependencies between the features of the model. There are also perceptrons, maximum entropy/logistic regression models, support vector machines, and many other **discriminative** techniques that all have various advantages and disadvantages in certain situations. These models are known as discriminative because they are optimized to distinguish the output labels given the input features (to discriminate between the different classes), rather than to jointly model the input and output variables as in the generative approach. As Vapnik [1998] says, (quoted in [Ng and Jordan, 2002]): "One should solve the [classification] problem directly and never solve a more general problem as an intermediate step." Indeed, [Roth, 1998] shows that generative and discriminative classifiers both make use of a linear feature space. Given the same representation, the difference between generative

and discriminative models therefore rests solely in how the weights are chosen. Rather than choosing weights that best fit the generative model on the training data (and satisfy the model's simplifying assumptions, typically concerning the interdependence or independence of different features), a discriminative model chooses the weights that best attain the desired objective: better predictions [Fung and Roth, 2005]. Discriminative models thus tend to perform better, and are correspondingly the preferred approach today in many areas of NLP (including increasingly in semantics, where we recently proposed a discriminative approach to selectional preference; Chapter 6). Unlike generative approaches, when using discriminative algorithms we can generally use arbitrary and interdependent features in our model without worrying about modeling such interdependencies. Use of the word *discriminative* in NLP has thus come to indicate both an approach that optimizes for classification accuracy directly *and* one that uses a wide variety of features. In fact, one kind of feature you might use in a discriminative system is the prediction or output of a generative model. This illustrates another advantage of discriminative learning: competing approaches can always be included as new features.

Note the clear advantages of discriminative models are really only true for supervised learning in NLP. There are now a growing number of generative, Bayesian, unsupervised algorithms that are being developed. It may be the case that the pendulum will soon swing back and generative models will again dominate the supervised playing field as well, particularly if they can provide principled ways to incorporate unlabeled data into a semi-supervised framework.

2.3.4 Support Vector Machines

When you have lots of features and lots of examples, support vector machines [Cortes and Vapnik, 1995] (SVMs) seem to be the best discriminative approach. One reason might be because they perform well in situations, like natural language, where many features are relevant [Joachims, 1999a], as opposed to situations where a few key indicators may be sufficient for prediction. Conceptually, SVMs take a geometric view of the problem, as depicted in Figure 2.1. The training algorithm chooses the hyperplane location such that it is maximally far away from the closest positive and negative points on either side of it (this is known as the *max-margin* solution). These closest vectors are known as support vectors. You can reconstruct the hyperplane from this set of vectors alone. Thus the name *support vector machine*. In fact, Figure 2.1 depicts the hyperplane that would be learned by an SVM, with marks on the corresponding support vectors.

It can be shown that the hyperplane that maximizes the margin corresponds to the weight vector that solves the following constrained optimization problem:

$$\begin{aligned} &\min_{\bar{w}} && \frac{1}{2}||\bar{w}||^2 \\ &\text{subject to}: \forall i, && y^i(\bar{w}\cdot\bar{x}^i) \geq 1 \end{aligned} \tag{2.2}$$

where $||\bar{w}||$ is the Euclidean norm of the weight vector. Note $||\bar{w}||^2 = \bar{w}\cdot\bar{w}$. The $\frac{1}{2}$ is a mathematical convenience so that that coefficient goes away when we take the derivative. The optimization says that we want to find the smallest weight vector (in terms of its Euclidean norm) such that our linear classifier's output ($h(\bar{x}) = \bar{w}\cdot\bar{x}$) is bigger than one when the correct label is a positive class (y = +1), and less than -1 when the correct label is a negative class (y = -1). The constraint in Equation 2.2 is a succinct way of writing these two conditions in one line.

Having the largest possible margin (or, equivalently, the smallest possible weight vector subject to the constraints) that classifies the training examples correctly seems to be a good idea, as it is most likely to **generalize** to new data. Once again, consider text categorization. We may have a feature for each word in each document. There's enough words and few enough documents such that our training algorithm could possibly get all the training examples classified correctly if it just puts all the weight on the rare words in each document. So if *Obama* occurs in a single sports document in our training set, but nowhere else in the training set, our algorithm *could* get that document classified correctly if it were to put all its weight on the word *Obama* and ignore the other features. Although this approach would do well on the training set, it will likely not generalize well to unseen documents. It's likely not the maximum margin (smallest weight vector) solution. If we can instead separate the positive and negative examples using more-frequent words like *score* and *win* and *teams* then we should do so. We will use less weights overall, and the weight vector will have a smaller norm (fewer weights will be non-zero). It intuitively seems like a good idea to rely on more frequent words to make decisions, and the SVM optimization just encodes this intuition in a theoretically well-grounded formulation (it's all based on 'empirical risk minimization' [Vapnik, 1998]).

Sometimes, the positive and negative examples are not separable, and there will be no solution to the above optimization. At other times, even if the data is separable, it may be better to turn the hard constraints in the above equation into soft preferences, and place even greater emphasis on using the frequent features. That is, we may wish to have a weight vector with a small norm even at the expense of not separating the data. In terms of categorizing sports documents, words like *score* and *win* and *teams* may sometimes occur in non-sports documents in the training set (so we may get some training documents wrong if we put positive weight on them), but they are a better bet for getting test documents correct than putting high weight on rare words like *Obama* (blindly enforcing separability). Geometrically, we can view this as saying we might want to allow some points to lie on the opposite side of the hyperplane (or at least closer to it), if we can do this with weights on fewer dimensions.

[Cortes and Vapnik, 1995] give the optimization program for a soft-margin SVM as:

$$\begin{aligned} \min_{\bar{w},\xi^1,\ldots,\xi^M} \quad & \frac{1}{2}||\bar{w}||^2 + C\sum_{i=1}^{m}\xi^i \\ \text{subject to}: \forall i, \quad & \xi^i \geq 0 \\ & y^i(\bar{w}\cdot\bar{x}^i) \geq 1-\xi^i \end{aligned} \tag{2.3}$$

The ξ^i values are known as the slacks. Each example may use some slack. The classification must either be separable and satisfy the margin constraint (in which case $\xi = 0$) or it may instead use its slack to satisfy the inequality. The weighted sum of the slacks are minimized along with the norm of $\bar{w}$.

The relative importance of the slacks (getting the training examples separated nicely) versus the minimization of the weights (using more general features) is controlled by tuning C. If the feature-weights learned by the algorithm are the *parameters*, then this C value is known as a ***hyperparameter***, since it's something done separately from the regular parameter learning. The general practice is to try various values for this hyperparameter, and choose the one that gets the highest performance on the development set. In an SVM, this hyperparameter is known as the **regularization** parameter. It controls how much we penalize training vectors that lie on the opposite side of the hyperplane (with distance given by

their slack value). In practice, I usually try a range of values for this parameter starting at 0.000001 and going up by a factor of 10 to around 100000.

Note you would not want to tune the regularization parameter by measuring performance on the *training* set, as less regularization is always going to lead to better performance on the training data itself. Regularization is a way to prevent **overfitting** the training data, and thus should be set on separate examples, i.e., the development set. However, some people like to do 10-fold cross validation on the training data to set their hyperparameters. I have no problem with this.

Another detail regarding SVM learning is that sometimes it makes sense to scale or normalize the features to enable faster and sometimes better learning. For many tasks, it makes sense to divide all the feature values by the Euclidean norm of the feature vector, such that the resulting vector has a magnitude of one. In the chapters that follow, we specify if we use such a technique. Again, we can test whether such a transformation is worth it by seeing how it affects performance on our development data.

SVMs have been shown to work quite well on a range of tasks. If you want to use a linear classifier, they seem to be a good choice. The SVM formulation is also perfectly suited to using kernels to automatically expand the feature space, allowing for non-linear classification. For all the tasks investigated in this dissertation, however, standard kernels were not found to improve performance. Furthermore, training and testing takes longer when kernels are used.

2.3.5 Software

We view the current best practice in most NLP classification applications as follows: Use as many labeled examples as you can find for the task and domain of interest. Then, carefully construct a linear feature space such that all potentially useful combinations of properties are explicit dimensions in that space (rather than implicitly creating such dimensions through the use of kernels). For training, use the LIBLINEAR package [Fan *et al.*, 2008], an amazingly fast solver that can return the SVM model in seconds even for tens of thousands of features and instances (other fast alternatives exist, but haven't been explored in this dissertation). This set-up allows for very rapid system development and evaluation, allowing us to focus on the features themselves, rather than the learning algorithm.

Since many of the tasks in this dissertation were completed before LIBLINEAR was available, we also present results using older solvers such as the logistic regression package in Weka [Witten and Frank, 2005], the efficient SVM^{multiclass} instance of SVM^{struct} [Tsochantaridis *et al.*, 2004]), and our old stand-by, Thorsten Joachim's SVM^{light} [Joachims, 1999a]. Whatever package is used, it should now be clear that in terms of this dissertation, training simply means learning a set of weights for a linear classifier using a given set of labeled data.

2.4 Unsupervised Learning

There is a way to gather linguistic annotations without using any training data: unsupervised learning. This at first seems rather magical. How can a system produce labels without ever seeing them?

Most current unsupervised approaches in NLP are decidedly unmagical. Probably since so much current work is based on supervised training from labeled data, some rule-based

and heuristic approaches are now being called *unsupervised*, since they are not based on learning from labeled data. For example, in Chapter 1, Section 1.1, we discussed how a part-of-speech tagger could be based on linguistic rules. A rule-based tagger could in some sense be considered *unsupervised*, since a human presumably created the rules from intuition, not from labeled data. However, since the human probably looked at *some* data to come up with the rules (a textbook, maybe?), calling this *unsupervised* is a little misleading from a machine learning perspective. Most people would probably simply call this a "rule-based approach." In Chapter 3, we propose unsupervised systems for lexical disambiguation, where a designer need only specify the words that are correlated with the classes of interest, rather than label any training data. We also discuss previous approaches that use counts derived from Internet search engine results. These approaches have usually been unsupervised.

From a machine learning perspective, true unsupervised approaches are those that induce output structure from properties of the problem, with guidance from probabilistic models rather than human intuition. We can illustrate this concept most clearly again with the example of document classification. Suppose we know there are two classes: documents about sports, and documents that are not about sports. We can generate the feature vectors as discussed above, and then simply form two groups of vectors such that members of each group are close to each other (in terms of Euclidean distance) in N-dimensional space. New feature vectors can be assigned to whatever group or **cluster** they are closest to. The points closest to one cluster will be separated from points closest to the other cluster by a hyperplane in N-dimensional space. Where there's a hyperplane, then there's a corresponding linear classifier, with a set of weights. So clustering can learn a linear classifier as well. We don't know what the clusters represent, but hopefully one of them has all the sports documents (if we inspect the clusters and define one of them as the *sports* class, we're essentially doing a form of *semi-supervised* learning).

Clustering can also be regarded as an "exploratory science" that seeks to discover useful patterns and structures in data [Pantel, 2003]. This structure might later be exploited for other forms of language processing; later we will see how clustering can be used to provide helpful feature information for supervised classifiers (Section 2.5.5).

Clustering is the simplest unsupervised learning algorithm. In more complicated setups, we can define a probability model over our features (and possibly over other *hidden* variables), and then try to learn the parameters of the model such that our unlabeled data has a high likelihood under this model. We previously used such a technique to train a pronoun resolution system using expectation maximization [Cherry and Bergsma, 2005]. Similar techniques can be used to train hidden markov models and other generative models.

These models can provide a very nice way to incorporate lots of unlabeled data. In some sense, however, doing anything beyond an HMM requires one to be a bit of probabilistic-modeling guru. The more features you incorporate in the model, the more you have to account for the interdependence of these features explicitly in your model. Some assumptions you make may not be valid and may impair performance. It's hard to know exactly what's wrong with your model, and how to change it to make it better. Also, when setting the parameters of your model using clustering or expectation maximization, you might reach a point only of local optimum, from which the algorithm can proceed no further to better settings under your model (and you have no idea you've reached this point). But, since these algorithms are not optimizing discriminative performance anyways, it's not clear you want the global maximum even if you can find it.

One way to find a better solution in this bumpy optimization space is to initialize or fix parameters of your model in ways that bias things toward what you know you want. For example, [Haghighi and Klein, 2010] fix a number of parameters in their entity-type / coreference model using *prototypes* of different classes. That is, they ensure, e.g., that *Bush* or *Gore* are in the PERSON class, as are the nominals *president, official*, etc., and that this class is referred to by the appropriate set of pronouns. They also set a number of other parameters to "fixed heuristic values." When the unsupervised learning kicks in, it initially has less freedom to go off the rails, as the hand-tuning has started the model from a good spot in the optimization space.

One argument that is sometimes made against fully unsupervised approaches is that the set-up is a little unrealistic. You will likely want to evaluate your approach. To evaluate your approach, you will need some labeled data. If you can produce labeled data for testing, you can produce some labeled data for training. It seems that *semi*-supervised learning is a more realistic situation: you have lots of unlabeled data, but you also have a few labeled examples to help you configure your parameters. In our unsupervised pronoun resolution work [Cherry and Bergsma, 2005], we also used some labeled examples to re-weight the parameters learned by EM (using the discriminative technique known as maximum entropy).[3]

Another interaction between unsupervised and supervised learning occurs when an unsupervised method provides intermediate structural information for a supervised algorithm. For example, unsupervised algorithms often generate the unseen word-to-word alignments in statistical machine translation [Brown *et al.*, 1993] and also the unseen character-to-phoneme alignments in grapheme-to-phoneme conversion [Jiampojamarn *et al.*, 2007]. This alignment information is then leveraged by subsequent supervised processing.

2.5 Semi-Supervised Learning

Semi-supervised learning is a huge and growing area of interest in many different research communities. The name *semi-supervised learning* has come to essentially mean that a predictor is being created from information from both labeled and unlabeled examples. There are a variety of flavours of semi-supervised learning that are relevant to NLP and merit discussion. A good recent survey of semi-supervised techniques in general is by Zhu [2005].

Semi-supervised learning was the "Special Topic of Interest" at the 2009 Conference on Natural Language Learning. The organizers, Suzanne Stevenson and Xavier Carreras, provided a thoughtful motivation for semi-supervised learning in the call for papers.[4]

> "The field of natural language learning has made great strides over the last 15 years, especially in the design and application of supervised and batch learning methods. However, two challenges arise with this kind of approach. First, in core NLP tasks, supervised approaches require typically large amounts of manually annotated data, and experience has shown that results often depend on the

[3] The line between supervised and unsupervised learning can be a little blurry. We called our use of labeled data and maximum entropy the "*supervised* extension" of our unsupervised system in our EM paper [Cherry and Bergsma, 2005]. A later *unsupervised* approach by Charniak and Elsner [2009], which also uses EM training for pronoun resolution, involved tuning essentially the same number of hyperparameters by hand (to optimize performance on a development set) as the number of parameters we tuned with supervision. Is this still unsupervised learning?

[4] `http://www.cnts.ua.ac.be/conll2009/cfp.html`

> precise make-up and genre of the training text, limiting generalizability of the results and the reach of the annotation effort. Second, in modeling aspects of human language acquisition, the role of supervision in learning must be carefully considered, given that children are not provided explicit indications of linguistic distinctions, and generally do not attend to explicit correction of their errors. Moreover, batch methods, even in an unsupervised setting, cannot model the actual online processes of child learning, which show gradual development of linguistic knowledge and competence."

Theoretical motivations aside, the practical benefit of this line of research is essentially to have the high performance and flexibility of discriminatively-trained systems, without the cost of labeling huge numbers of examples. One can always label more examples to achieve better performance on a particular task and domain but the expense can be severe. Even companies with great resources, like Google and Microsoft, prefer solutions that do not require paying annotators to create labeled data. This is because any cost of annotation would have to be repeated in each language and potentially each domain in which the system might be deployed (because of the dependence on the "precise make-up and genre of the training text" mentioned above). While some annotation jobs can be shipped to cheap overseas annotators at relatively low cost, finding annotation experts in many languages and domains might be more difficult.[5] Furthermore, after initial results, if the objective of the program is changed slightly, then new data would have to be annotated once again. Not only is this expensive, but it slows down the product development cycle. Finally, for many companies and government organizations, data privacy and security concerns prevent the outsourcing of annotation altogether. All labeling must be done by expensive and overstretched internal analysts.

Of course, even when there is plentiful labeled examples and the problem is well-defined and unchanging, it may still boost performance to incorporate statistics from unlabeled data. We have recently seen impressive gains from using unlabeled evidence, even with large amounts of labeled data, for example in the work of Ando and Zhang [2005], Suzuki and Isozaki [2008], and Pitler et al. [2010].

In the remainder of this section, we briefly outline approaches to transductive learning, self-training, bootstrapping, learning with heuristically-labeled examples, and using features derived from unlabeled data. We focus on the work that best characterizes each area, simply noting in passing some research that does not fit cleanly into a particular category.

2.5.1 Transductive Learning

Transductive learning gives us a great opportunity to talk more about document classification (where it was perhaps most famously applied in [Joachims, 1999b]), but otherwise this approach does not seem to be widely used in NLP. Most learners operate in the **inductive** learning framework: you learn your model from the training set, and apply it to unseen data. In the **transductive** framework on the other hand, you assume that, at learning time, you are given access to the test examples you wish to classify (but not their labels).

[5] Another trend worth highlighting is work that leverages large numbers of cheap, non-expert annotations through online services such as Amazon's Mechanical Turk [Snow *et al.*, 2008]. This has been shown to work surprisingly well for a number of simple problems. Combining the benefits of non-expert annotations with the benefits of semi-supervised learning is a potentially rich area for future work.

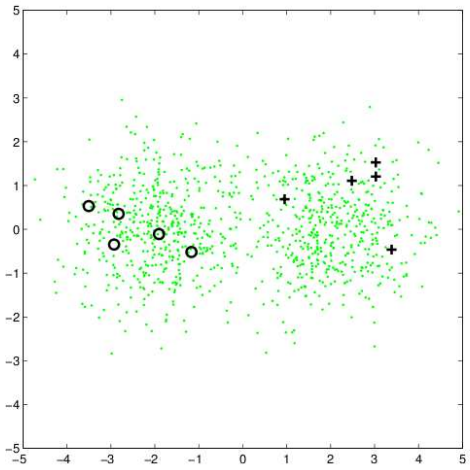


Figure 2.2: Learning from labeled and unlabeled examples, from (Zhu, 2005)

Consider Figure 2.2. In the typical inductive set-up, we would design our classifier based purely on the labeled points for the two classes: the **o**'s and +'s. We would draw the best hyperplane to separate these labeled vectors. However, when we look at all the dots that do not have labels, we may wish to draw a different hyperplane. It appears that there are two clusters of data, one on the left and one on the right. Drawing a hyperplane down the middle would appear to be the optimum choice to separate the two classes. This is only apparent after inspecting unlabeled examples.

We can always train a classifier using both labeled and unlabeled examples in the transductive set-up, but then apply the classifier to unseen data in an inductive evaluation. So in some sense we can group other semi-supervised approaches that make use of labeled and unlabeled examples into this category (e.g. work by Wang et al. [2008]), even if they are not applied transductively *per se*.

There are many computational algorithms that can make use of unlabeled examples when learning the separating hyperplane. The intuition behind them is to say something like: of all combinations of *possible* labels on the unseen examples, find the overall best separating hyperplane. Thus, in some sense we pretend we know the labels on the unlabeled data, and use these labels to train our model via traditional supervised learning. In most semi-supervised algorithms, we either implicitly or explicitly generate labels for unlabeled data in a conceptually similar fashion, to (hopefully) enhance the data we use to train the classifier.

These approaches are not applicable to the problems that we wish to tackle in this dissertation mainly due to practicality. We want to leverage huge volumes of unlabeled data: all the data on the web, if possible. Most transductive algorithms cannot scale to this many examples. Another potential problem is that for many NLP applications, the space of possible labels is simply too large to enumerate. For example, work in parsing aims to produce a tree indicating the syntactic relationships of the words in a sentence. [Church and Patil, 1982] show the number of possible binary trees increases with the Catalan numbers. For twenty-word sentences, there are billions of possible trees. We are currently exploring linguistically-motivated ways to perform a high-precision pruning of the output space for

parsing and other tasks [Bergsma and Cherry, 2010]. One goal of our work is to facilitate more intensive semi-supervised learning approaches. This is an active research area in general.

2.5.2 Self-training

Self-training is a very simple algorithm that has shown some surprising success in natural language parsing [McClosky *et al.*, 2006a]. In this approach, you build a classifier (or parser, or any kind of predictor) on some labeled training examples. You then use the learned classifier to label a large number of unlabeled feature vectors. You then re-train your system on both the original labeled examples and the automatically-labeled examples (and then evaluate on your original development and test data). Again, note that this semi-supervised technique explicitly involves generating labels for unlabeled data to enhance the training of the classifier.

Historically, this approach has not worked very well. Any errors the system makes after the first round of training are just compounded by re-training on those errors. Perhaps it works better in parsing (and especially with a parse reranker) where the constraints of the grammar give some extra guidance to the initial output of the parser. More work is needed in this area.

2.5.3 Bootstrapping

Bootstrapping has a long and rich history in NLP. Bootstrapping is like self-training, but where we avoid the compounding of errors by exploiting different **views** of the problem. We first describe the overall idea in the context of algorithms for multi-view learning. We then consider how related work in bootstrapping from seeds also fits into the multi-view framework.

Bootstrapping with Multiple Views

Consider, once again, classifying documents. However, let's assume that these are online documents. In addition to the words in the documents themselves, we might also classify documents using the text in hyperlinks pointing to the documents, taken from other websites (so-called *anchor text*). In the standard supervised learning framework, we would just use this additional text as additional features, and train on our labeled set. In a bootstrapping approach (specifically, the **co-training** algorithm [Blum and Mitchell, 1998]), we instead train two classifiers: one with features from the document, and one with features from the anchor text in hyperlinks. We use one classifier to label additional examples for the other to learn from, and iterate training and classification with one classifier then the other until all the documents are labeled. Since the classifiers have **orthogonal views** of the problem, the mistakes made by one classifier should not be too detrimental to the learning of the other classifier. That is, the errors should not compound as they do in self-training. Blum and Mitchell [1998] give a PAC Learning-style framework for this approach, and give empirical results on the web-page classification task.

The notion of a problem having orthogonal views or representations is an extremely powerful concept. Many language problems can be viewed in this way, and many algorithms that exploit a dual representation have been proposed. Yarowsky [1995] first implemented this style of algorithm in NLP (and it is now sometimes referred to as the *Yarowsky*

algorithm). Yarowsky used it for word-sense disambiguation. He essentially showed that a bootstrapping approach can achieve performance comparable to full supervised learning. An example from word-sense disambiguation will help illustrate: To disambiguate whether the noun *bass* is used in the *fish* sense or in the *music* sense, we can rely on a just a few key contexts to identify unambiguous instances of the noun in text. Suppose we know that *caught a bass* means the fish sense of *bass*. Now, whenever we see *caught a bass*, we label that noun for the *fish* sense. This is the context-based view of the problem. The other view is a document-based view. It has been shown experimentally that all instances of a unique word type in a single document tend to share the same sense [Gale *et al.*, 1992]. Once we have one instance of *bass* labeled, we can extend this classification to the other instances of *bass* in the same document using this second view. We can then re-learn our context-based classifier from these new examples and repeat the process in new documents and new contexts, until all the instances are labeled.

Multi-view bootstrapping is also used in information extraction [Etzioni *et al.*, 2005]. Collins and Singer [1999] and Cucerzan and Yarowsky [1999] apply bootstrapping to the task of named-entity recognition. Klementiev and Roth [2006] used bootstrapping to extract interlingual named entities. Our research has also been influenced by co-training-style **weakly supervised** algorithms used in coreference resolution [Ge *et al.*, 1998; Harabagiu *et al.*, 2001; Müller *et al.*, 2002; Ng and Cardie, 2003b; 2003a; Bean and Riloff, 2004] and grammatical gender determination [Cucerzan and Yarowsky, 2003].

Bootstrapping from Seeds

A distinct line of bootstrapping research has also evolved in NLP, which we call *Bootstrapping from Seeds*. These approaches all involve starting with a small number of examples, building predictors from these examples, labeling more examples with the new predictors, and then repeating the process to build a large collection of information. While this research generally does not explicitly cast the tasks as exploiting orthogonal views of the data, it is instructive to describe these techniques from the multi-view perspective.

An early example is described by Hearst [1992]. Suppose we wish to find hypernyms in text. A hypernym is a relation between two things such that one thing is a sub-class of the other. It is sometimes known as the *is-a* relation. For example a *wound is-a* type of *injury*, *Ottawa is-a city*, a *Cadillac is-a car*, etc. Suppose we see the words in text, "Cadillacs and other cars..." There are two separate sources of information in this example:

1. The string pair itself: *Cadillac, car*
2. The context: **X***s and other* **Y***s*

We can perform bootstrapping in this framework as follows: First, we obtain a list of seed pairs of words, e.g. *Cadillac/car*, *Ottawa/city*, *wound/injury*, etc. Now, we create a predictor that will label examples as being hypernyms based purely on whether they occur in this seed set. We are thus only using the first view of the problem: the actual string pairs. We use this predictor to label a number of examples in actual text, e.g. "*Cadillacs* and other *cars*, *cars* such as *Cadillacs*, *cars* including *Cadillacs*, etc." We then train a predictor for the other view of the problem: From all the labeled examples, we extract predictive contexts: "**X***s and other* **Y***s*, **Y***s such as* **X***s*, **Y***s including* **X***s*, etc." The contexts extracted in this view can now be used to extract more seeds, and the seeds can then be used to extract more contexts, etc., in an iterative fashion. Hearst described an early form of this algorithm,

which used some manual intervention, but later approaches have essentially differed quite little from her original proposal.

Google co-founder Sergei Brin [1998] used a similar technique to extract relations such as (*author*, *title*) from the web. Similar work was also presented in [Riloff and Jones, 1999] and [Agichtein and Gravano, 2000]. Pantel and Pennacchiotti [2006] used this approach to extract general semantic relations (such as *part-of*, *succession*, *production*, etc.), while Paşca et al. [2006] present extraction results on a web-scale corpus. Another famous variation of this method is Ravichandran and Hovy's system for finding patterns for answering questions [Ravichandran and Hovy, 2002]. They begin with seeds such as (*Mozart*, *1756*) and use these to find patterns that contain the answers to questions such as *When was X born?*

Note the contrast with the traditional supervised machine-learning framework, where we would have annotators mark up text with examples of hypernyms, relations, or question-answer pairs, etc., and then learn a predictor from these labeled examples using supervised learning. In bootstrapping from seeds, we do not label segments of text, but rather pairs of words (labeling only one view of the problem). When we find instances of these pairs in text, we essentially label more data automatically, and then infer a context-based predictor from this labeled set. This context-based predictor can then be used to find more examples of the relation of interest (hypernyms, authors of books, question-answer pairs, etc.). Notice, however, that in contrast to standard supervised learning, we do not label any *negative* examples, only positive instances. Thus, when building a context-based predictor, there is no obvious way to exploit our powerful machinery for feature-based discriminative learning and classification. Very simple methods are instead used to keep track of the best context-based patterns for identifying new examples in text.

In iterative bootstrapping, although the first round of training often produces reasonable results, things often go wrong in later iterations. The first round will inevitably produce some noise, some wrong pairs extracted by the predictor. The contexts extracted from these false predictions will lead to more false pairs being extracted, and so on. In all published research on this topic that we are aware of, the precision of the extractions decreases in each stage.

2.5.4 Learning with Heuristically-Labeled Examples

In the above discussion of bootstrapping, we outlined a number of approaches that extend an existing set of classifications (or seeds) by iteratively classifying and learning from new examples. Another interesting, non-iterative scenario is the situation where, rather than having a few seed examples, we begin with many positive examples of a class or relation, and attempt to classify new relations in this context. With a relatively comprehensive set of seeds, there is little value in iterating to obtain more.[6] Also, having a lot of seeds can also provide a way to generate the negative examples we need for discriminative learning. In this section we look at two flavours: special cases where the examples can be created automatically, and cases where we have only positive seeds, and so create pseudo-negative examples through some heuristic means.

[6]There are also non-iterative approaches that also start with limited seed data. Haghighi and Klein [2006] create a generative, unsupervised sequence prediction model, but add features to indicate if a word to be classified is distributionally-similar to a seed word. Like the approaches presented in our discussion of bootstrapping with seeds, this system achieves impressive results starting with very little manually-provided information.

Learning with Natural Automatic Examples

Some of the lowest-hanging fruit in the history of NLP arose when researchers realized that some important problems in NLP could be solved by generating labeled training examples automatically from raw text.

Consider the task of diacritic or accent restoration. In languages such as French or Spanish, accents are often omitted in informal correspondence, in all-capitalized text such as headlines, and in lower-bit text encodings. Missing accents adversely affect both syntactic and semantic analysis. It would be nice to train a discriminative classifier to restore these accents, but do we need someone to label the accents in unaccented text to provide us with labeled data? Yarowsky [1994] showed that we can simply take (readily-available) *accented* text, take the accents off and use them as labels, and then train predictors using features for everything *except* for the accents. We can essentially generate as many labeled examples as we like this way. The true accent and the text provide the positive example. The unaccented or alternatively-accented text provides negative examples.

We call these *Natural Automatic Examples* since they naturally provide the positive and negative examples needed to solve the problem. We contrast these with problems in the following section where, although one may have plentiful positive examples, one must use some creativity to produce the negative examples.

This approach also works for context-sensitive spelling correction. Here we try to determine, for example, whether someone who typed *whether* actually meant *weather*. We take well-edited text and, each time one of the words is used, we create a training example, with the word-actually-used as the label. We then see if we can predict these words from their confusable alternatives, using the surrounding context for features [Golding and Roth, 1999]. So the word-actually-used is the positive example (e.g. "*whether* or not"), while the alternative, unused words provide the negatives (e.g. "*weather* or not"). Banko and Brill [2001] generate a lot of training data this way to produce their famous results on the relative importance of the learning algorithm versus the amount of training data (the amount of training data is much much more important). In Chapter 3, we use this approach to generate data for both preposition selection and context-sensitive spelling correction.

A similar approach could be used for training systems to segment text into paragraphs, to restore capitalization or punctuation, to do sentence-boundary detection (one must find an assiduous typist, like me, who consistently puts two spaces after periods, but only one after abbreviations...), to convert curse word symbols like %*#@ back into the original curse, etc. (of course, some of these examples may benefit from a channel model rather than exclusively a source/language model). The only limitation is the amount of training data your algorithm can handle. In fact, by summarizing the training examples with N-gram-based features as in Section 2.5.5 (rather than learning from each instance separately), there really is no limitation on the amount of data you might learn from.

There are a fairly limited number of problems in NLP where we can just create examples automatically this way. This is because in NLP, we are usually interested in generating structures over the data that are not surface apparent in naturally-occurring text. We return to this when we discuss analysis and generation problems in Chapter 3. Natural automatic examples abound in many other fields. You can build a discriminative classifier for whether a stock goes up or for whether someone defaults on their loan purely based on previous examples. A search engine can easily predict whether someone will click on a search result using the history of clicks from other users for the same query [Joachims, 2002]. However, despite not having natural automatic examples for some problems, we can sometimes create

automatic examples heuristically. We turn to this in the following subsection.

Learning with Pseudo-Negative Examples

While the previous section described problems where there were natural positive and negative examples (e.g., the correct accent marker is positive, while others, including no accent, are negative), there is a large class of problems in NLP where we only have positive examples and thus it's not clear how to use a discriminative classifier to evaluate new potential examples. This is the situation with seed data: you are presented with a list of only positive seeds, and there's nothing obvious to discriminate these from.

In these situations, researchers have devised various ways to automatically create negative examples. For example, let us return to the example of hypernyms. Although Hearst [1992] started her algorithm with only a few examples, this was an unnecessary handicap. Thousands of examples of hypernym pairs can be extracted automatically from the lexical database WordNet [Miller *et al.*, 1990]. Furthermore, WordNet has good coverage of the relations involving nouns that are actually in WordNet (as opposed to, obviously, no coverage of relations involving words that aren't mentioned in WordNet at all). Thus, pairs of words in WordNet that are *not* linked in a hypernym structure can potentially be taken as reliable examples of words that are *not* hypernyms (since both words are in WordNet, if they were hypernyms, the relation would generally be labeled). These could form our negative examples for discrimination.

Recognizing this, Snow et al. [2005] use WordNet to generate a huge set of both positive and negative hypernym pairs: exactly what we need as training data for a large-scale discriminative classifier. With this resource, we need not iteratively discover contexts that are useful for hypernymy: Snow et al. simply include, as features in the classifier, all the syntactic paths connecting the pair of words in a large parsed corpus. That is, they have features for how often a pair of words occurs in constructions like,"**X***s and other* **Y***s*, **Y***s such as* **X***s*, **Y***s including* **X***s*, etc." Discriminative training, not heuristic weighting, will decide the importance of these patterns in hypernymy. To classify any new example pair (i.e., for nouns that are *not* in WordNet), we can simply construct their feature vector of syntactic paths and apply the classifier. Snow et al. [2005] achieve very good performance using this approach.

This approach could scale to make use of features derived from web-scale data. For any pair of words, we can efficiently extract all the N-grams in which both words occur. This is exactly what we proposed for discriminating object and subject relations for *Bears won* and *trophy won* in our example in Chapter 1, Section 1.3. We can create features from these N-grams, and apply training and classification.

We recently used a similar technique for classifying the natural gender of English nouns [Bergsma *et al.*, 2009a]. Rather than using WordNet to label examples, however, we used co-occurrence statistics in a large corpus to reliably identify the most likely gender of thousands of noun phrases. We then used this list to automatically label examples in raw text, and then proceeded to learn from these automatically-labeled examples. This paper could have served as another chapter in this dissertation, but the dissertation already seemed sufficiently long without it.

Several other recent uses of this approach are also worth mentioning. Okanohara and Tsujii [2007] created examples automatically in order to train a discriminative whole-sentence language model. Language models are designed to tell us whether a sequence of words is valid language (or likely, fluent, *good* English). We can automatically gather

positive examples from any collection of well-formed sentences: they are all valid sentences by definition. But how do we create negative examples? The innovation of Okanohara and Tsujii is to create negative examples from *sentences generated by an N-gram language model*. N-grams are the standard Markovized approximation to English, and their success in language modeling is one of the reasons for the statistical revolution in NLP discussed in Section 2.1 above. However, they often produce ill-formed sentences, and a classifier that can distinguish between valid English sentences and N-gram-model-generated sentences could help us select better output sentences from our speech recognizers, machine translators, curse-word restoration systems, etc.

The results of Okanohara and Tsujii's classifier was promising: about 74% of sentences could be classified correctly. However, they report that a native English speaker was able to achieve 99% accuracy on a 100-sentence sample, indicating that there is much room to improve. It is rare that humans can outperform computers on a task where we have essentially unlimited amounts of training data. Indeed, learning curves in this work indicate that performance is continuously improving up to 500,000 training examples. The main limitation seems to only be computational complexity.

Smith and Eisner [2005] also automatically generate negative examples. They perturb their input sequence (e.g. the sentence word order) to create a neighborhood of *implicit* negative evidence. Structures over the observed sentence should have higher likelihood than structures over the perturbed sequences.

Chapter 6 describes an approach that creates both positive and negative examples of selectional preference from corpus-wide statistics of predicate-argument pairs (rather than only using a local sentence to generate negatives, as in [Smith and Eisner, 2005]). Since the individual training instances encapsulate information from potentially thousands or millions of sentences, this approach can scale better than some of the other semi-supervised approaches described in this chapter. In Chapter 7, we create examples by computing statistics over an aligned bitext, and generate negative examples to be those that have a high string overlap with the positives, but which are not likely to be translations. We use automatically-created examples to mine richer features and demonstrate better models than previous work.

However, note that there is a danger in solving problems on automatically-labeled examples: it is not always clear that the classifier you learn will transfer well to actual tasks, since you're no longer learning a discriminator on manually-labeled examples. In the following section, we describe semi-supervised approaches that train over manually-labeled data, and discuss how perhaps we can have the best of both worlds by including the output of our pseudo-discriminators as features in a supervised model.

2.5.5 Creating Features from Unlabeled Data

We have saved perhaps the simplest form of semi-supervised learning for last: an approach where we simply create features from our unlabeled data and use these features in our supervised learners. Simplicity is good.[7]

The main problem with essentially all of the above approaches is that at some point,

[7] In the words of Mann and McCallum [2007]: "Research in semi-supervised learning has yielded many publications over the past ten years, but there are surprisingly fewer cases of its use in application-oriented research, where the emphasis is on solving a task, not on exploring a new semi-supervised method. This may be partially due to the natural time it takes for new machine learning ideas to propagate to practitioners. We believe it is also due in large part to the complexity and unreliability of many existing semi-supervised methods."

automatically-labeled examples are used to train the classifier. Unfortunately, automatically-labeled examples are often incorrect. The classifier works hard to classify these examples correctly, and subsequently gets similar examples wrong that it encounters at testing. If we have enough manually-labeled examples, it seems that we want the ultimate mediator of the value of our features to be performance on these labeled examples, not performance on any pseudo-examples. This mediation is, of course, exactly what supervised learning does. If we instead create *features* from unlabeled data, rather than using unlabeled data to create new *examples*, standard supervised learning can be used.

How can we include information from unlabeled data as new features in a supervised learner? Section 2.2 described a typical feature representation: each feature is a binary indicator of whether a word is present or not in a document to be classified. When we extract features from unlabeled data, we add new dimensions to the feature representation. These new dimensions are for features that represent what we might call *second-order* interactions – co-occurrences of words with each other in unlabeled text.

In very recent papers, both Huang and Yates [2009] and Turian et al. [2010] provide comparisons of different ways to extract new features from unlabeled data; they both evaluate performance on a range of tasks.

Features Directly From a Word's Distribution in Unlabeled Text

Returning to our sports example, we could have a feature for whether a word in a given document occurs *elsewhere*, in unlabeled data, with the word *score*. A classifier could learn that this feature is associated with the *sports* class, because words like *hockey*, *baseball*, *inning*, *win*, etc. tend to occur with *score*, and some of these likely occur in the training set. So, although we may never see the word *curling* during training, it does occur in unlabeled text with many of the same words that occur with other *sports* terms, like the word *score*. So a document that contains *curling* will have the second-order *score* feature, and thus *curling*, through features created from its distribution, is still an indicator of *sports*. Directly having a feature for each item that co-occurs in a word's distribution is perhaps the simplest way to leverage unlabeled data in the feature representation. Huang and Yates [2009] essentially use this as their multinomial representation. They find it performs worse on sequence-labeling tasks than distributional representations based on HMMs and latent-semantic analysis (two other effective approaches for creating features from unlabeled data). One issue with using the distribution directly is that although sparsity is potentially alleviated at the word level (we can handle words even if we haven't seen them in training data), we increase sparsity at the feature level: there are more features to train but the same amount of training data. This might explain why [Huang and Yates, 2009] see improved performance on rare words but similar performance overall. We return to this issue in Chapter 5 when we present a distributional representation for verb part-of-speech tag disambiguation that may also suffer from these drawbacks (Section 5.6).

Features from Similar Words or Distributional Clusters

There are many other ways to create features from unlabeled data. One popular approach is to summarize the distribution of words (in unlabeled data) using similar words. [Wang *et al.*, 2005] use similar words to help generalization in dependency parsing. [Marton *et al.*, 2009] use similar phrases to help improve the handling of out-of-vocabulary terms in a machine translation system. Another recent trend is to create features from automatically-

generated word clusters. Several researchers have used the hierarchical Brown et al. [1992] clustering algorithm, and then created features for cluster membership at different levels of the hierarchy [Miller *et al.*, 2004; Koo *et al.*, 2008]. Rather than clustering single words, Lin and Wu [2009] use phrasal clusters, and provide features for cluster membership when different numbers of clusters are used in the clustering.

Features for the Output of Auxiliary Classifiers

Another way to create features from unlabeled data is to create features for the output of predictions on auxiliary problems that can be trained solely with unlabeled data [Ando and Zhang, 2005]. For example, we could create a prediction for whether the word *arena* occurs in a document. We can take all the documents where *arena* does and does not occur, and build a classifier using all the other words in the document. This classifier may predict that *arena* does occur if the words *hockey*, *curling*, *fans*, etc. occur. When the predictions are used as features, if they are useful, they will receive high weight at training time. At test time, if we see a word like *curling*, for example, even though it was never seen in our labeled set, it may cause the predictor for *arena* to return a high score, and thus also cause the document to be recognized as *sports*.

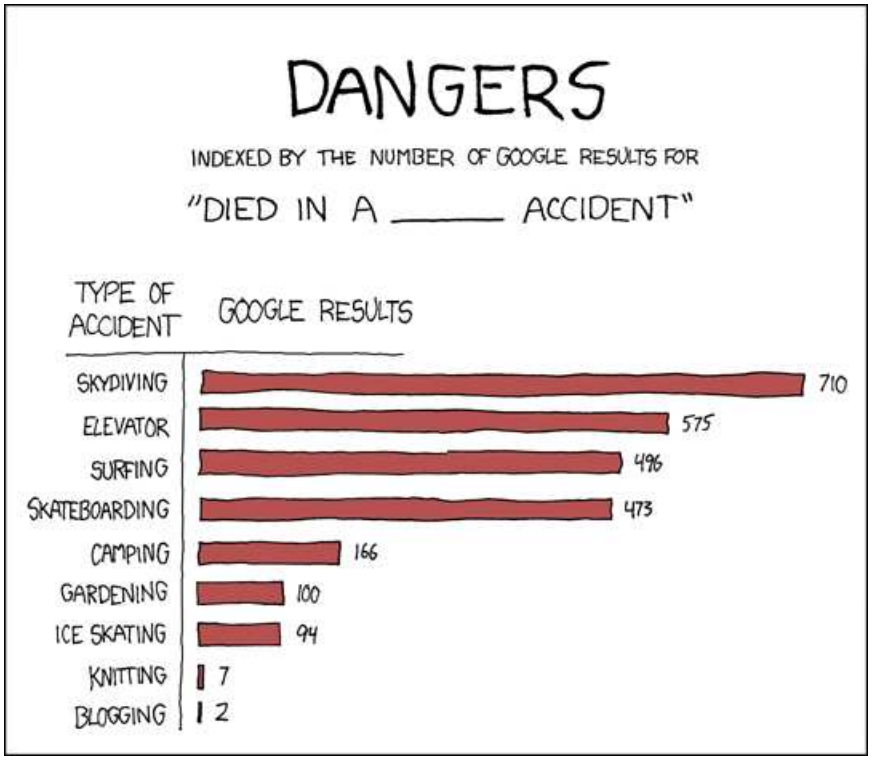
Note that since these examples can be created automatically, this problem (and other auxiliary problems in the Ando and Zhang approach) fall into the category of those with *Natural Automatic Examples* as discussed above. One possible direction for future work is to construct auxiliary problems with pseudo-negative examples. For example, we could include the predictions of various configurations of our selectional-preference classifier (Chapter 6) as a feature in a discriminatively-trained language model. We took a similar approach in our work on gender [Bergsma *et al.*, 2009a]. We trained a classifier on automatically-created examples, but used the output of this classifier as another feature in a classifier trained on a small amount of supervised data. This resulted in a substantial gain in performance over using the original prediction on its own: 95.5% versus 92.6% (but note other features were combined with the prediction of the auxiliary classifier).

Features used in this Dissertation

In this dissertation, we create features from unsupervised data in several chapters and in several different ways. In Chapter 6, to assess whether a noun is compatible with a verb, we create features for the noun's distribution only with *other verbs*. Thus we characterize a noun by its verb contexts, rather than its full distribution, using less features than a naive representation using the noun's full distributional profile. Chapters 3 and 5 also *selectively* use features from parts of the total distribution of a word, phrase, or pair of words (to characterize the relation between words, for noun compound bracketing and verb tag disambiguation in Chapter 5). In Chapter 3, we characterize contexts by using selected types from the distribution of other words that occur in the context. For the adjective-ordering work in Chapter 5, we choose an order based on the distribution of the adjectives individually and combined in a phrase. Our approaches are simple, but effective. Perhaps most importantly, by leveraging the counts in a web-scale N-gram corpus, they scale to make use of all the text data on the web. On the other hand, scaling most other semi-supervised techniques to even moderately-large collections of unlabeled text remains "future work" for a large number of published approaches in the machine learning and NLP literature.

Chapter 3

Learning with Web-Scale N-gram Models



XKCD comic: Dangers, `http://xkcd.com/369/`

3.1 Introduction

Many problems in Natural Language Processing (NLP) can be viewed as assigning labels to particular words in text, given the word's context. If the decision process requires choosing a label from a predefined set of possible choices, called a *candidate set* or *confusion set*, the process is often referred to as *disambiguation* [Roth, 1998]. Part-of-speech tagging, spelling correction, and word sense disambiguation are all lexical disambiguation processes.

[0] A version of this chapter has been published as [Bergsma *et al.*, 2008b; 2009b]

One common disambiguation task is the identification of word-choice errors in text. A language checker can flag an error if a confusable alternative better fits a given context:

(1) The system tried to decide {*among, between*} the two confusable words.

Most NLP systems resolve such ambiguity with the help of a large corpus of text. The corpus indicates which candidate is more frequent in similar contexts. The larger the corpus, the more accurate the disambiguation [Banko and Brill, 2001]. Since few corpora are as large as the world wide web,[1] many systems incorporate web counts into their selection process.

For the above example, a typical web-based system would query a search engine with the sequences "decide *among* the" and "decide *between* the" and select the candidate that returns the most pages [Lapata and Keller, 2005]. Clearly, this approach fails when more context is needed for disambiguation.

We propose a unified view of using web-scale data for lexical disambiguation. Rather than using a single context sequence, we use contexts of various lengths and positions. There are five 5-grams, four 4-grams, three trigrams and two bigrams spanning the target word in Example (1). We gather counts for each of these sequences, with each candidate in the target position. We first show how the counts can be used as features in a supervised classifier, with a count's contribution weighted by its context's size and position. We also propose a novel unsupervised system that simply sums a subset of the (log) counts for each candidate. Surprisingly, this system achieves most of the gains of the supervised approach without requiring any training data.

Since we make use of features derived from the distribution of patterns in large amounts of unlabeled data, this work is an instance of a semi-supervised approach in the category, "Using Features from Unlabeled Data," discussed in Chapter 2, Section 2.5.5.

In Section 3.2, we discuss the range of problems that fit the lexical disambiguation framework, and also discuss previous work using the web as a corpus. In Section 3.3 we discuss our general disambiguation methodology. While all disambiguation problems can be tackled in a common framework, most approaches are developed for a specific task. Like Roth [1998] and Cucerzan and Yarowsky [2002], we take a unified view of disambiguation, and apply our systems to preposition selection (Section 3.5), spelling correction (Section 3.6), and non-referential pronoun detection (Section 3.7). In particular we spend a fair amount of time on non-referential pronoun detection. On each of these applications, our systems outperform traditional web-scale approaches.

3.2 Related Work

3.2.1 Lexical Disambiguation

Yarowsky [1994] defines lexical disambiguation as a task where a system must "disambiguate two or more semantically distinct word-forms which have been conflated into the same representation in some medium." Lapata and Keller [2005] divide disambiguation problems into two groups: generation and analysis. In generation, the confusable candidates are actual words, like *among* and *between*. Generation problems permit learning with

[1] Google recently announced they are now indexing over 1 trillion unique URLs (`http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`). This figure represents a staggering amount of textual data.

"Natural Automatic Examples," as described in Chapter 2, Section 2.5.4. In analysis, we disambiguate semantic labels, such as part-of-speech tags, representing abstract properties of surface words. For these problems, we have historically needed manually-labeled data.

For generation tasks, a model of each candidate's distribution in text is created. The models indicate which usage best fits each context, enabling candidate disambiguation in tasks such as spelling correction [Golding and Roth, 1999], preposition selection [Chodorow *et al.*, 2007; Felice and Pulman, 2007], and diacritic restoration [Yarowsky, 1994]. The models can be large-scale classifiers or standard N-gram language models (LMs).

An N-gram is a sequence of words. A unigram is one word, a bigram is two words, a trigram is three words, and so on. An N-gram language model is a model that computes the probability of a sentence as the product of the probabilities of the N-grams in the sentence. The (maximum likelihood) probability of an N-gram is simply its count divided by the number of times it occurs in the corpus. Higher probability sentences will thus be composed of N-grams that are more frequent. For resolving confusable words, we could select the candidate that results in a higher whole-sentence probability, effectively combining the counts of N-grams at different positions.

The power of an N-gram language model crucially depends on the data from which the counts are taken: the more data, the better. Trigram LMs have long been used for spelling correction, an approach sometimes referred to as the Mays, Damerau, and Mercer model [Wilcox-O'Hearn *et al.*, 2008]. Gamon et al. [2008] use a Gigaword 5-gram LM for preposition selection. While web-scale LMs have proved useful for machine translation [Brants *et al.*, 2007], most web-scale disambiguation approaches compare specific sequence counts rather than full-sentence probabilities. Counts are usually gathered using an Internet search engine [Lapata and Keller, 2005; Yi *et al.*, 2008].

In analysis problems such as part-of-speech tagging, it is not as obvious how a LM can be used to score the candidates, since LMs do not contain the candidates themselves, only surface words. However, large LMs can also benefit these applications, provided there are surface words that correlate with the semantic labels. Essentially, we devise some surrogates for each label, and determine the likelihood of these surrogates occurring with the given context. For example, Mihalcea and Moldovan [1999] perform sense disambiguation by creating label surrogates from similar-word lists for each sense. To choose the sense of *bass* in the phrase "caught a huge bass," we might consider *tenor*, *alto*, and *pitch* for sense one and *snapper*, *mackerel*, and *tuna* for sense two. The sense whose group has the higher web-frequency count in *bass*'s context is chosen. [Yu *et al.*, 2007] use a similar approach to verify the near-synonymy of the words in the *sense pools* of the OntoNotes project [Hovy *et al.*, 2006]. They check whether a word can be substituted into the place of another element in its sense pool, using a few sentences where the sense pool of the original element has been annotated. The substitution likelihood is computed using the counts of N-grams of various orders from the Google web-scale N-gram corpus (discussed in the following subsection).

We build on similar ideas in our unified view of analysis and generation disambiguation problems (Section 3.3). For generation problems, we gather counts for each surface candidate filling our 2-to-5-gram patterns. For analysis problems, we use surrogates as the fillers. We collect our pattern counts from a web-scale corpus.

3.2.2 Web-Scale Statistics in NLP

Exploiting the vast amount of data on the web is part of a growing trend in natural language processing [Keller and Lapata, 2003]. In this section, we focus on some research that has had a particular influence on our own work. We begin by discussing approaches that extract information using Internet search-engines, before discussing recent approaches that have made use of the Google web-scale N-gram corpus.

There were initially three main avenues of research that used the web as a corpus; all were based on the use of Internet search engines.

In the first line of research, search-engine page counts are used as substitutes for counts of a phrase in a corpus [Grefenstette, 1999; Keller and Lapata, 2003; Chklovski and Pantel, 2004; Lapata and Keller, 2005]. That is, a phrase is issued to a search engine as a query, and the count, given by the search engine, of how many pages contain that query is taken as a substitute for the number of times that phrase occurs on the web. Quotation marks are placed around the phrase so that the words are only matched when they occur in their exact phrasal order. By using Internet-derived statistics, these approaches automatically benefit from the growing size and variety of documents on the world wide web. We previously used this approach to collect pattern counts that indicate the gender of noun phrases; this provided very useful information for an anaphora resolution system [Bergsma, 2005]. We also previously showed how a variety of search-engine counts can be used to improve the performance of search-engine query segmentation [Bergsma and Wang, 2007] (a problem closely related to Noun-Compound Bracketing, which we explore in Chapter 5).

In another line of work, search engines are use to assess how often a pair of words occur on the same page (or how often they occur close to each other), irrespective of their order. Thus the page counts returned by a search engine are taken at face value as document co-occurrence counts. Applications in this area include determining the phrasal semantic orientation (good or bad) for sentiment analysis [Turney, 2002] and assessing the coherence of key phrases [Turney, 2003].

A third line of research involves issuing queries to a search engine and then making use of the returned documents. Resnik [1999] shows how the web can be used to gather bilingual text for machine translation, while Jones and Ghani [2000] use the web to build corpora for minority languages. Ravichandran and Hovy [2002] process returned web pages to identify answer patterns for question answering. In an answer-typing system, Pinchak and Bergsma [2007] use the web to find documents that provide information on unit types for how-questions. Many other question-answering systems use the web to assist in finding a correct answer to a question [Brill *et al.*, 2001; Cucerzan and Agichtein, 2005; Radev *et al.*, 2001]. Nakov and Hearst [2005a; 2005b] use search engines both to return counts for N-grams, and also to process the returned results to extract information not available from a search-engine directly, such as punctuation and capitalization.

While a lot of progress has been made using search engines to extract web-scale statistics, there are many fundamental issues with this approach. First of all, since the web changes every day, the results using a search engine are not exactly reproducible. Secondly, some have questioned the reliability of search engine page counts [Kilgarriff, 2007]. Most importantly, using search engines to extract count information is terribly inefficient, and thus search engines restrict the number of queries one can issue to gather web-scale information. With limited queries, we can only use limited information in our systems.

A solution to these issues was enabled by Thorsten Brants and Alex Franz at Google when they released the Google Web 1T 5-gram Corpus Version 1.1 in 2006 [Brants and

Franz, 2006]. This corpus simply lists, for sequences of words from length two to length five, how often the sequence occurs in their web corpus. The web corpus was generated from approximately 1 trillion tokens of online text. In this data, tokens appearing less than 200 times have been mapped to the ⟨UNK⟩ symbol. Also, only N-grams appearing more than 40 times are included. A number of researchers have begun using this N-gram corpus, rather than search engines, to collect their web-scale statistics [Vadas and Curran, 2007a; Felice and Pulman, 2007; Yuret, 2007; Kummerfeld and Curran, 2008; Carlson *et al.*, 2008; Bergsma *et al.*, 2008b; Tratz and Hovy, 2010]. Although this N-gram data is much smaller than the source text from which it was taken, it is still a very large resource, occupying approximately 24 GB compressed, and containing billions of N-grams in hundreds of files. Special strategies are needed to effectively query large numbers of counts. Some of these strategies include pre-sorting queries to reduce passes through the data, hashing [Hawker *et al.*, 2007], storing the data in a database [Carlson *et al.*, 2008], and using a trie structure [Sekine, 2008]. Our work in this area led to our recent participation in the 2009 Johns Hopkins University, Center for Speech and Language Processing, Workshop on *Unsupervised Acquisition of Lexical Knowledge from N-Grams*, led by Dekang Lin.[2] A number of ongoing projects using web-scale N-gram counts have arisen from this workshop, and we discuss some of these in Chapter 5. Lin et al. [2010] provides an overview of our work at the workshop, including the construction of a new web-scale N-gram corpus.

In this chapter, all N-gram counts are taken from the standard Google N-gram data.

One thing that N-gram data does not provide is the *document* co-occurrence counts that have proven useful in some applications discussed above. It could therefore be beneficial to the community to have a resource along the lines of the Google N-gram corpus, but where the corpus simply states how often pairs of words (or phrases) co-occur within a fixed window on the web. I am putting this on my to-do list.

3.3 Disambiguation with N-gram Counts

Section 3.2.1 described how lexical disambiguation, for both generation and analysis tasks, can be performed by scoring various context sequences using a statistical model. We formalize the context used by web-scale systems and then discuss various statistical models that use this information.

For a word in text, $\mathbf{v_0}$, we wish to assign an output, c_i, from a fixed set of candidates, $C = \{c_1, c_2 ..., c_K\}$. Assume that our target word $\mathbf{v_0}$ occurs in a sequence of context tokens: $\mathbf{V}$=$\{v_{-4}, v_{-3}, v_{-2}, v_{-1}, \mathbf{v_0}, v_1, v_2, v_3, v_4\}$. The key to improved web-scale models is that they make use of a variety of context segments, of different sizes and positions, that span the target word $\mathbf{v_0}$. We call these segments *context patterns*. The words that replace the target word are called *pattern fillers*. Let the set of pattern fillers be denoted by $F = \{f_1, f_2, ..., f_{|F|}\}$. Recall that for generation tasks, the filler set will usually be identical to the set of output candidates (e.g., for word selection tasks, F=C=$\{$*among,between*$\}$). For analysis tasks, we must use other fillers, chosen as surrogates for one of the semantic labels (e.g. for WSD of *bass*, C=$\{Sense1, Sense2\}$, F=$\{$*tenor,alto,pitch,snapper,mackerel,tuna*$\}$).

Each length-N context pattern, with a filler in place of $\mathbf{v_0}$, is an N-gram, for which we can retrieve a count from an auxiliary corpus. We retrieve counts from the web-scale Google Web 5-gram Corpus, which includes N-grams of length one to five (Section 3.2.2).

[2] `http://www.clsp.jhu.edu/workshops/ws09/groups/ualkn/`

For each target word $\mathbf{v_0}$, there are five 5-gram context patterns that may span it. For Example (1) in Section 3.1, we can extract the following 5-gram patterns:

system tried to decide $\mathbf{v_0}$
tried to decide $\mathbf{v_0}$ the
to decide $\mathbf{v_0}$ the two
decide $\mathbf{v_0}$ the two confusable
$\mathbf{v_0}$ the two confusable words

Similarly, there are four 4-gram patterns, three 3-gram patterns and two 2-gram patterns spanning the target. With $|F|$ fillers, there are $14|F|$ filled patterns with relevant N-gram counts. For example, for F=$\{among, between\}$, there are two filled 5-gram patterns that begin with the word *decide*: "decide *among* the two confusable" and "decide *between* the two confusable." We collect counts for each of these, along with all the other filled patterns for this example. When F=$\{among, between\}$, there are 28 relevant counts for each example.

We now describe various systems that use these counts.

3.3.1 SUPERLM

We use supervised learning to map a target word and its context to an output. There are two steps in this mapping: a) converting the word and its context into a feature vector, and b) applying a classifier to determine the output class.

In order to use the standard x, y notation for classifiers, we write things as follows: Let $\bar{x} = \mathbf{\Phi}(\mathbf{V})$ be a mapping of the input to a feature representation, $\bar{x}$. We might also think of the feature function as being parameterized by the set of fillers, F and the N-gram corpus, R, so that $\bar{x} = \mathbf{\Phi}_{(F,R)}(\mathbf{V})$. The feature function $\mathbf{\Phi}_{(F,R)}(\cdot)$ outputs the count (in logarithmic form) of the different context patterns with the different fillers. Each of these has a corresponding dimension in the feature representation. If $N = 14|F|$ counts are used, then each $\bar{x}$ is an N-dimensional feature vector.

Now, the classifier outputs the index of the highest-scoring candidate in the set of candidate outputs, $C = \{c_1, c_2 ..., c_K\}$. That is, we let $y \in \{1, ..., K\}$ be the set of classes that can be produced by the classifier. The classifier, H, is therefore a K-class classifier, mapping an attribute vector, $\bar{x}$, to a class, y. Using the standard [Crammer and Singer, 2001]-style multi-class formulation, H is parameterized by a K-by-N matrix of weights, $\mathbf{W}$:

$$H_{\mathbf{W}}(\bar{x}) = \underset{r=1}{\overset{K}{\operatorname{argmax}}}\{\bar{W}_r \cdot \bar{x}\} \tag{3.1}$$

where $\bar{W}_r$ is the rth row of $\mathbf{W}$. That is, the predicted class is the index of the row of $\mathbf{W}$ that has the highest inner-product with the attributes, $\bar{x}$. The weights are optimized using a set of M training examples, $\{(\bar{x}^1, y^1), ..., (\bar{x}^M, y^M)\}$.

This differs a little from the linear classifier that we presented in Section 2.2. Here we actually have K linear classifiers. Although there is only one set of N features, there is a different linear combination for each row of $\mathbf{W}$. Therefore, the weight on a particular count depends on the class we are scoring (corresponding to the row of $\mathbf{W}$, r), as well as the filler, the context position, and the context size, all of which select one of the $14|F|$ base features. There are therefore a total of $14|F|K$ count-weight parameters. Chapter 4 formally describes how these parameters are learned using a multi-class SVM. Chapter 4 also discusses enhancements to this model that can enable better performance with fewer training examples.

Here, we simply provide some intuitions on what kinds of weights will be learned. To be clear, note that $\bar{W}_r$, the rth row of the weight-matrix $\mathbf{W}$, corresponds to the weights for predicting candidate c_r. Recall that in generation tasks, the set C and the set F may be identical. So some of the weights in $\bar{W}_r$ will therefore correspond to features for patterns filled with filler f_r. Intuitively, these weights will be positive. That is, we will predict the class *among* when there are high counts for the patterns filled with the filler *among* (c_r=f_r=*among*). On the other hand, we will choose not to pick *among* if the counts on patterns filled with *between* are high. These tendencies are all learned by the learning algorithm. The learning algorithm can also place higher absolute weights on the more predictive context positions and sizes. For example, for many tasks, the patterns that begin with a filler are more predictive than patterns that end with a filler. The learning algorithm attends to these differences in predictive power as it maximizes prediction accuracy on the training data.

We now note some special features used by our classifier. If a pattern spans outside the current sentence (when $\mathbf{v_0}$ is close to the start or end), we use zero for the corresponding feature value, but fire an indicator feature to flag that the pattern crosses a boundary. This feature provides a kind of smoothing. Other features are possible: for generation tasks, we could also include synonyms of the output candidates as fillers. Features could also be created for counts of patterns processed in some way (e.g. converting one or more context tokens to wildcards, POS-tags, lower-case, etc.), provided the same processing can be done to the N-gram corpus (we do such processing for the non-referential pronoun detection features described in Section 3.7).

We call this approach SUPERLM because it is SUPERvised, and because, like an interpolated language model (LM), it mixes N-gram statistics of different orders to produce an overall score for each filled context sequence. SUPERLM's features differ from previous lexical disambiguation feature sets. In previous systems, attribute-value features flag the presence or absence of a particular word, part-of-speech, or N-gram in the vicinity of the target [Roth, 1998]. Hundreds of thousands of features are used, and pruning and scaling can be key issues [Carlson *et al.*, 2001]. Performance scales logarithmically with the number of examples, even up to one billion training examples [Banko and Brill, 2001]. In contrast, SUPERLM's features are all aggregate counts of events in an external (web) corpus, not specific attributes of the current example. It has only $14|F|K$ parameters, for the weights assigned to the different counts. Much less training data is needed to achieve peak performance. Chapter 5 contrasts the performance of classifiers with N-gram features and traditional features on a range of tasks.

3.3.2 SUMLM

We create an unsupervised version of SUPERLM. We produce a score for each *filler* by summing the (unweighted) log-counts of all context patterns filled with that filler. For example, the score for *among* could be the sum of all 14 context patterns filled with *among*. For generation tasks, the filler with the highest score is taken as the label. For analysis tasks, we compare the scores of different fillers to arrive at a decision; Section 3.7.2 explains how this is done for non-referential pronoun detection.

We refer to this approach in our experiments as SUMLM.

For generation problems where F=C, SUMLM is similar to a naive bayes classifier,

but without counts for the class prior.[3] Naive bayes has a long history in disambiguation problems [Manning and Schütze, 1999], so it is not entirely surprising that our SUMLM system, with a similar form to naive bayes, is also effective.

3.3.3 TRIGRAM

Previous web-scale approaches are also unsupervised. Most use one context pattern for each filler: the trigram with the filler in the middle: $\{v_{-1}, f, v_1\}$. $|F|$ counts are needed for each example, and the filler with the most counts is taken as the label [Lapata and Keller, 2005; Liu and Curran, 2006; Felice and Pulman, 2007]. Using only one count for each label is usually all that is feasible when the counts are gathered using an Internet search engine, which limits the number of queries that can be retrieved. With limited context, and somewhat arbitrary search engine page counts, performance is limited. Web-based systems are regarded as "baselines" compared to standard approaches [Lapata and Keller, 2005], or, worse, as scientifically unsound [Kilgarriff, 2007]. Rather than using search engines, higher accuracy and reliability can be obtained using a large corpus of automatically downloaded web documents [Liu and Curran, 2006]. We evaluate the trigram pattern approach, with counts from the Google 5-gram corpus, and refer to it as TRIGRAM in our experiments.

3.3.4 RATIOLM

Carlson et al. [2008] proposed an unsupervised method for spelling correction that also uses counts for various pattern fillers from the Google 5-gram Corpus. For every context pattern spanning the target word, the algorithm calculates the ratio between the highest and second-highest filler counts. The position with the highest ratio is taken as the "most discriminating," and the filler with the higher count in this position is chosen as the label. The algorithm starts with 5-grams and backs off to lower orders if no 5-gram counts

[3] In this case, we can think of the features, x_i, as being the context patterns, and the classes y as being the fillers. In a naive bayes classifier, we select the class, y, that has the highest score under:

$$
\begin{aligned}
H(\bar{x}) &= \operatorname*{argmax}_{r=1}^{K} \Pr(y_r|\bar{x}) \\
&= \operatorname*{argmax}_{r=1}^{K} \Pr(y_r)\Pr(\bar{x}|y_r) && \text{Bayes decision rule} \\
&= \operatorname*{argmax}_{r=1}^{K} \Pr(y_r) \prod_i \Pr(x_i|y_r) && \text{naive bayes assumption} \\
&= \operatorname*{argmax}_{r=1}^{K} \log(\Pr(y_r)) + \sum_i \log(\Pr(x_i|y_r)) \\
&= \operatorname*{argmax}_{r=1}^{K} \log(\Pr(y_r)) + \sum_i \operatorname{logcnt}(x_i, y_r) - \operatorname{logcnt}(y_r) \\
&= \operatorname*{argmax}_{r=1}^{K} g(y_r) + \sum_i \operatorname{logcnt}(x_i, f_r) && y_r = f_r
\end{aligned}
$$

where we collect all the terms that depend solely on the class into $g(y_r)$. Our SUMLM system is exactly the same as this naive bayes classifier if we drop the $g(y_r)$ term. We tried various ways to model the class priors using N-gram counts and incorporating them into our equations, but nothing performed as well as simply dropping them altogether. Another option we haven't explored is simply having a single class bias parameter for each class, $\lambda_r = g(y_r)$, to be added to the filler counts. We would tune the λ_r's by hand for each task where SUMLM is applied. However, this would make the model require some labeled data to tune, whereas our current SUMLM is parameter-free and entirely unsupervised.

are available. This position-weighting (*viz.* feature-weighting) technique is similar to the decision-list weighting in [Yarowsky, 1994]. We refer to this approach as RATIOLM in our experiments.

3.4 Evaluation Methodology

We compare our supervised and unsupervised systems on three experimental tasks: preposition selection, context-sensitive spelling correction, and non-referential pronoun detection. We evaluate using *accuracy*: the percentage of correctly-selected labels. As a baseline (BASE), we state the accuracy of always choosing the most-frequent class. For spelling correction, we average accuracies across the five confusion sets. We also provide learning curves by varying the number of labeled training examples. It is worth reiterating that this data is used solely to weight the contribution of the different filler counts; the filler counts themselves do not change, as they are always extracted from the full Google 5-gram Corpus.

For training SUPERLM, we use a support vector machine (SVM). SVMs achieve good performance on a range of tasks (Chapter 2, Section 2.3.4). We use a linear-kernel multiclass SVM (the efficient $\text{SVM}^{multiclass}$ instance of SVM^{struct} [Tsochantaridis *et al.*, 2004]). It slightly outperformed one-versus-all SVMs in preliminary experiments (and a later, more extensive study in Chapter 4 confirmed that these preliminary intuitions were justified). We tune the SVM's regularization parameter on the development sets. We apply add-one smoothing to the counts used in SUMLM and SUPERLM, while we add 39 to the counts in RATIOLM, following the approach of Carlson et al. [2008] (40 is the count cut-off used in the Google Corpus). For all unsupervised systems, we choose the most frequent class if no counts are available. For SUMLM, we use the development sets to decide which orders of N-grams to combine, finding orders 3-5 optimal for preposition selection, 2-5 optimal for spelling correction, and 4-5 optimal for non-referential pronoun detection. Development experiments also showed RATIOLM works better starting from 4-grams, not the 5-grams originally used in [Carlson *et al.*, 2008].

3.5 Preposition Selection

3.5.1 The Task of Preposition Selection

Choosing the correct preposition is one of the most difficult tasks for a second-language learner to master, and errors involving prepositions constitute a significant proportion of errors made by learners of English [Chodorow *et al.*, 2007]. Several automatic approaches to preposition selection have recently been developed [Felice and Pulman, 2007; Gamon *et al.*, 2008]. We follow the experiments of Chodorow et al. [2007], who train a classifier to choose the correct preposition among 34 candidates.[4] In [Chodorow *et al.*, 2007], feature vectors indicate words and part-of-speech tags near the preposition, similar to the features used in most disambiguation systems, and unlike the aggregate counts we use in our supervised preposition-selection N-gram model (Section 3.3.1).

[4] Chodorow et al. do not identify the 34 prepositions they use. We use the 34 from the SemEval-07 preposition sense-disambiguation task [Litkowski and Hargraves, 2007]: *about, across, above, after, against, along, among, around, as, at, before, behind, beneath, beside, between, by, down, during, for, from, in, inside, into, like, of, off, on, onto, over, round, through, to, towards, with*

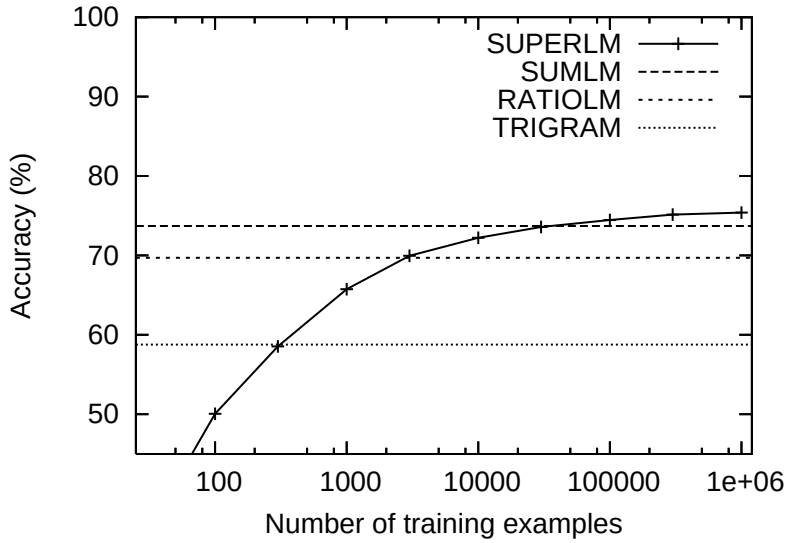


Figure 3.1: Preposition selection learning curve

For preposition selection, like all generation disambiguation tasks, labeled data is essentially free to create (i.e, the problem has *natural automatic examples* as explained in Chapter 2, Section 2.5.4). Each preposition in edited text is assumed to be correct, automatically providing an example of that preposition's class. We extract examples from the New York Times (NYT) section of the Gigaword corpus [Graff, 2003]. We take the first 1 million prepositions in NYT as a training set, 10K from the middle as a development set and 10K from the end as a final unseen test set. We tokenize the corpus and identify prepositions by string-match. Our system uses no parsing or part-of-speech tagging to extract the examples or create the features.

3.5.2 Preposition Selection Results

Preposition selection is a difficult task with a low baseline: choosing the most-common preposition (*of*) in our test set achieves 20.3%. Training on 7 million examples, Chodorow et al. [2007] achieved 69% on the full 34-way selection. Tetreault and Chodorow [2008] obtained a human upper bound by removing prepositions from text and asking annotators to fill in the blank with the best preposition (using the current sentence as context). Two annotators achieved only 75% agreement with each other and with the original text.

In light of these numbers, the accuracy of the N-gram models are especially impressive. SUPERLM reaches 75.4% accuracy, equal to the human agreement (but on different data). Performance continually improves with more training examples, but only by 0.25% from 300K to 1M examples (Figure 3.1). SUMLM (73.7%) significantly outperforms RATIOLM (69.7%), and nearly matches the performance of SUPERLM. TRIGRAM performs worst (58.8%), but note it is the only previous web-scale approach applied to preposition selection [Felice and Pulman, 2007]. All differences are statistically significant (McNemar's test, $p<0.01$).

The order of N-grams used in the SUMLM system strongly affects performance. Using only trigrams achieves 66.8% accuracy, while using only 5-grams achieves just 57.8% (Table 3.1). Note that the performance with only trigrams (66.8%) is not equal to the per-

	Max			
Min	2	3	4	5
2	50.2	63.8	70.4	72.6
3		66.8	72.1	73.7
4			69.3	70.6
5				57.8

Table 3.1: SUMLM accuracy (%) combining N-grams from order *Min* to *Max*

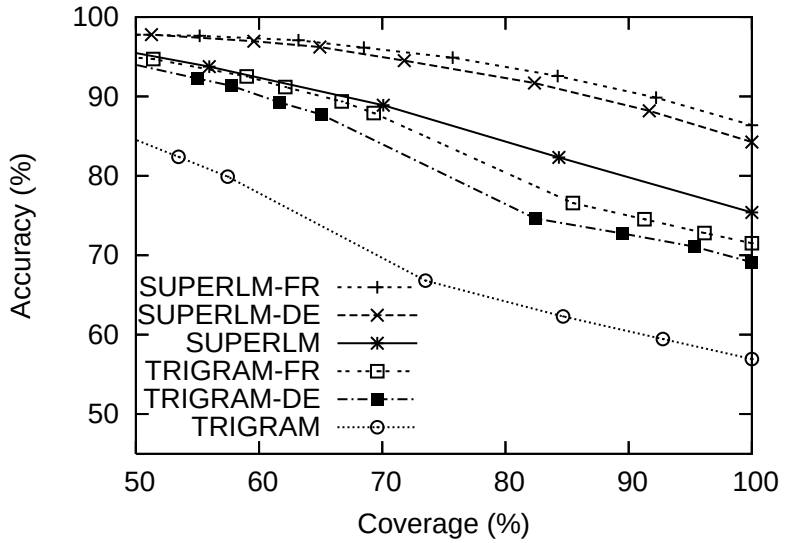


Figure 3.2: Preposition selection over high-confidence subsets, with and without language constraints (-FR,-DE)

formance of the standard TRIGRAM approach (58.8%), because the standard TRIGRAM approach only uses a single trigram (the one centered on the preposition) whereas SUMLM always uses the three trigrams that span the confusable word.

Coverage is the main issue affecting the 5-gram model: only 70.1% of the test examples had a 5-gram count for *any* of the 34 fillers. 93.4% of test examples had at least one 4-gram count and 99.7% of examples had at least one trigram count.

Summing counts from 3-5 results in the best performance on the development and test sets.

We compare our use of the Google Corpus to extracting page counts from a search engine, via the Google API (no longer in operation as of August 2009, but similar services exist). Since the number of queries allowed to the API is restricted, we test on only the first 1000 test examples. Using the Google Corpus, TRIGRAM achieves 61.1%, dropping to 58.5% with search engine page counts. Although this is a small difference, the real issue is the restricted number of queries allowed. For each example, SUMLM would need 14 counts for each of the 34 fillers instead of just one. For training SUPERLM, which has 1 million training examples, we need counts for 267 million *unique* N-grams. Using the Google API with a 1000-query-per-day quota, it would take over 732 years to collect all the counts for training. This is clearly why some web-scale systems use such limited context.

We also follow Carlson et al. [2001] and Chodorow et al. [2007] in extracting a subset of decisions where our system has higher confidence. We only propose a label if the ratio between the highest and second-highest score from our classifier is above a certain threshold, and then vary this threshold to produce accuracy at different coverage levels (Figure 3.2). The SUPERLM system can obtain close to 90% accuracy when deciding on 70% of examples, and above 95% accuracy when deciding on half the examples. The TRIGRAM performance rises more slowly as coverage drops, reaching 80% accuracy when deciding on only 57% of examples.

Many of SUPERLM's errors involve choosing between prepositions that are unlikely to be confused in practice, e.g. *with*/*without*. Chodorow et al. [2007] wrote post-processor rules to prohibit corrections in the case of antonyms. Note that the errors made by an English learner also depend on their native language. A French speaker looking to translate *au-dessus de* has one option in some dictionaries: *above*. A German speaker looking to translate *über* has, along with *above*, many more options. When making corrections, we could combine SUPERLM (a *source* model) with the likelihood of each confusion depending on the writer's native language (a *channel* model). The channel model could be trained on text written by second-language learners who speak, as a first language, the particular language of interest.

In the absence of such data, we only allow our system to make corrections in English if the proposed replacement shares a foreign-language translation in a particular Freelang online bilingual dictionary (`www.freelang.net/dictionary/`). Put another way, we reduce the size of the preposition confusion set dynamically depending on the preposition that was used and the native language of the speaker. A particular preposition is only suggested as a correction if both the correction and the original preposition could have been confused by a foreign-language speaker translating a particular foreign-language preposition without regard to context.

To simulate the use of this module, we randomly flip 20% of our test-set prepositions to confusable ones, and then apply our classifier with the aforementioned confusability (and confidence) constraints. We experimented with French and German lexicons (Figure 3.2). These constraints strongly benefit both SUPERLM and TRIGRAM, with French constraints ($-FR$) helping slightly more than German ($-DE$) for higher coverage levels. There are fewer confusable prepositions in the French lexicon compared to German. As a baseline, if we assign our labels random scores, adding the French and German constraints results in 20% and 14% accuracy, respectively (compared to $\frac{1}{34}$ = 2.9% unconstrained). At 50% coverage, both constrained SUPERLM systems achieve close to 98% accuracy, a level that could provide very reliable feedback in second-language learning software.

3.6 Context-Sensitive Spelling Correction

3.6.1 The Task of Context-Sensitive Spelling Correction

Context-sensitive spelling correction, or real-word error/malapropism detection [Golding and Roth, 1999; Hirst and Budanitsky, 2005], is the task of identifying errors when a misspelling results in a real word in the lexicon, e.g., using *site* when *sight* or *cite* was intended. Contextual spell checkers are among the most widely-used NLP technology, as they are included in commercial word processing software [Church *et al.*, 2007].

For every occurrence of a word in a pre-defined confusion set (like {*among*, *between*}),

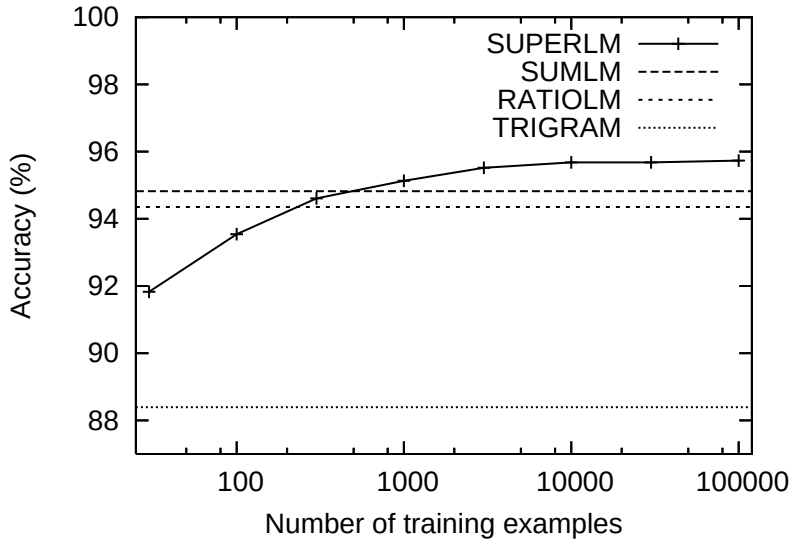


Figure 3.3: Context-sensitive spelling correction learning curve

we select the most likely word from the set. The importance of using large volumes of data has previously been noted [Banko and Brill, 2001; Liu and Curran, 2006]. Impressive levels of accuracy have been achieved on the standard confusion sets, for example, 100% on disambiguating both {*affect, effect*} and {*weather, whether*} by Golding and Roth [1999]. We thus restricted our experiments to the five confusion sets (of twenty-one in total) where the reported performance in [Golding and Roth, 1999] is below 90% (an average of 87%): {*among, between*}, {*amount, number*}, {*cite, sight, site*}, {*peace, piece*}, and {*raise, rise*}. We again create labeled data automatically from the NYT portion of Gigaword. For each confusion set, we extract 100K examples for training, 10K for development, and 10K for a final test set.

3.6.2 Context-sensitive Spelling Correction Results

Figure 3.3 provides the spelling correction learning curve, while Table 3.2 gives results on the five confusion sets. Choosing the most frequent label averages 66.9% on this task (BASE). TRIGRAM scores 88.4%, comparable to the trigram (page count) results reported in [Lapata and Keller, 2005]. SUPERLM again achieves the highest performance (95.7%), and it reaches this performance using many fewer training examples than with preposition selection. This is because the number of parameters grows with the number of fillers *times* the number of labels (recall, there are $14|F|K$ count-weight parameters), and there are 34 prepositions but only two-to-three confusable spellings. Note that we also include the performance reported in [Golding and Roth, 1999], although these results are reported on a different corpus.

SUPERLM achieves a 24% relative reduction in error over RATIOLM (94.4%), which was the previous state-of-the-art [Carlson *et al.*, 2008]. SUMLM (94.8%) also improves on RATIOLM, although results are generally similar on the different confusion sets. On {*raise,rise*}, SUPERLM's supervised weighting of the counts by position and size does not improve over SUMLM (Table 3.2). On all the other sets the performance is higher; for example, on {*among,between*}, the accuracy improves by 2.3%. On this set, counts for

Set	BASE	[Golding and Roth, 1999]	TRIGRAM	SUMLM	SUPERLM
among/between	60.3	86.0	80.8	90.5	92.8
amount/number	75.6	86.2	83.9	93.2	93.7
cite/sight/site	87.1	85.3	94.3	96.3	97.6
peace/piece	60.8	88.0	92.3	97.7	98.0
raise/rise	51.0	89.7	90.7	96.6	96.6
Average	66.9	87.0	88.4	94.8	95.7

Table 3.2: Context-sensitive spelling correction accuracy (%) on different confusion sets

fillers near the beginning of the context pattern are more important, as the object of the preposition is crucial for distinguishing these two classes ("*between* the **two**" but "*among* the **three**"). SUPERLM can exploit the relative importance of the different positions and thereby achieve higher performance.

3.7 Non-referential Pronoun Detection

We now present an application of our approach to a difficult analysis problem: detecting non-referential pronouns. In fact, SUPERLM was originally devised for this task, and then subsequently evaluated as a general solution to all lexical disambiguation problems. More details on this particular application are available in our ACL 2008 paper [Bergsma *et al.*, 2008b].

3.7.1 The Task of Non-referential Pronoun Detection

Coreference resolution determines which noun phrases in a document refer to the same real-world entity. As part of this task, coreference resolution systems must decide which pronouns refer to preceding noun phrases (called antecedents) and which do not. In particular, a long-standing challenge has been to correctly classify instances of the English pronoun *it*. Consider the sentences:

(1) You can make it in advance.

(2) You can make it in Hollywood.

In Example (1), *it* is an anaphoric pronoun referring to some previous noun phrase, like "the sauce" or "an appointment." In Example (2), *it* is part of the idiomatic expression "make it" meaning "succeed." A coreference resolution system should find an antecedent for the first *it* but not the second. Pronouns that do not refer to preceding noun phrases are called *non-anaphoric* or *non-referential* pronouns.

The word *it* is one of the most frequent words in the English language, accounting for about 1% of tokens in text and over a quarter of all third-person pronouns.[5] Usually between a quarter and a half of *it* instances are non-referential. As with other pronouns, the preceding discourse can affect *it*'s interpretation. For example, Example (2) can be interpreted as referential if the preceding sentence is "You want to make a movie?" We show, however,

[5] e.g. `http://ucrel.lancs.ac.uk/bncfreq/flists.html`

Pattern Filler Type	String
#1: 3rd-person pron. sing.	*it/its*
#2: 3rd-person pron. plur.	*they/them/their*
#3: any other pronoun	*he/him/his/*, *I/me/my*, etc.
#4: infrequent word token	⟨*UNK*⟩
#5: any other token	*

Table 3.3: Pattern filler types

that we can reliably classify a pronoun as being referential or non-referential based solely on the local context surrounding the pronoun, using the techniques described in Section 3.3.

The difficulty of non-referential pronouns has been acknowledged since the beginning of computational resolution of anaphora. Hobbs [1978] notes his algorithm does not handle pronominal references to sentences nor cases where *it* occurs in time or weather expressions. Hirst [1981, page 17] emphasizes the importance of detecting non-referential pronouns, "lest precious hours be lost in bootless searches for textual referents." Mueller [2006] summarizes the evolution of computational approaches to non-referential *it* detection. In particular, note the pioneering work of Paice and Husk [1987], the inclusion of non-referential *it* detection in a full anaphora resolution system by Lappin and Leass [1994], and the machine learning approach of Evans [2001].

3.7.2 Our Approach to Non-referential Pronoun Detection

We apply our web-scale disambiguation systems to this task. Like in the above approaches, we turn the context into patterns, with *it* as the word to be labeled. Since the output classes are not explicit words, we devise some surrogate fillers. To illustrate for Example (1), note we can extract the context pattern "make * in advance" and for Example (2) "make * in Hollywood," where "*" represents the filler in the position of *it*. Non-referential instances tend to have the word *it* filling this position in the pattern's distribution. This is because non-referential patterns are fairly unique to non-referential pronouns. Referential distributions occur with many other noun phrase fillers. For example, in the Google N-gram corpus, "make it in advance" and "make them in advance" occur roughly the same number of times (442 vs. 449), indicating a referential pattern. In contrast, "make it in Hollywood" occurs 3421 times while "make them in Hollywood" does not occur at all. This indicates that some useful statistics are counts for patterns filled with the words *it* and *them*.

These simple counts strongly indicate whether another noun can replace the pronoun. Thus we can computationally distinguish between a) pronouns that refer to nouns, and b) all other instances: including those that have no antecedent, like Example (2), and those that refer to sentences, clauses, or implied topics of discourse.

We now discuss our full set of pattern fillers. For identifying non-referential *it* in English, we are interested in how often *it* occurs as a pattern filler versus other *nouns*. As surrogates for nouns, we gather counts for five different classes of words that fill the wild-card position, determined by string match (Table 3.3).[6] The third-person plural *they* (#2) reliably occurs in patterns where referential *it* also resides. The occurrence of *any other*

[6]Note, this work was done before the availability of the POS-tagged Google V2 corpus (Chapter 5). We could directly count noun-fillers using that corpus.

pronoun (#3) guarantees that at the very least the pattern filler is a noun. A match with the *infrequent word token* ⟨UNK⟩ (#4) (explained in Section 3.2.2) will likely be a noun because nouns account for a large proportion of rare words in a corpus. Gathering *any other token* (#5) also mostly finds nouns; inserting another part-of-speech usually results in an unlikely-to-be-observed, ungrammatical pattern.

Unlike our work in preposition selection and spelling correction above, we process our input examples and our N-gram corpus in various way to improve generality. We change the patterns to lower-case, convert sequences of digits to the # symbol, and run the Porter stemmer [Porter, 1980].[7] Our method also works without the stemmer; we simply truncate the words in the pattern at a given maximum length. With simple truncation, all the pattern processing can be easily applied to other languages. To generalize rare names, we convert capitalized words longer than five characters to a special *NE* (named entity) tag. We also added a few simple rules to stem the irregular verbs *be*, *have*, *do*, and *said*, and convert the common contractions *'nt*, *'s*, *'m*, *'re*, *'ve*, *'d*, and *'ll* to their most likely stem. When we extract counts for a processed pattern, we sum all the counts for matching N-grams in the identically-processed Google corpus.

We run SUPERLM using the above fillers and their processed-pattern counts as described in Section 3.3.1. For SUMLM, we decide $NonRef$ if the difference between the SUMLM scores for *it* and *they* is above a threshold. For TRIGRAM, we also threshold the ratio between *it*-counts and *they*-counts. For RATIOLM, we compare the frequencies of *it* and *all*, and decide $NonRef$ if the count of *it* is higher. These thresholds and comparison choices were optimized on the development set.

3.7.3 Non-referential Pronoun Detection Data

We need labeled data for training and evaluation of our system. This data indicates, for every occurrence of the pronoun *it*, whether it refers to a preceding noun phrase or not. Standard coreference resolution data sets annotate all noun phrases that have an antecedent noun phrase in the text. Therefore, we can extract labeled instances of *it* from these sets. We do this for the dry-run and formal sets from MUC-7 [1997], and merge them into a single data set.

Of course, full coreference-annotated data is a precious resource, with the pronoun *it* making up only a small portion of the marked-up noun phrases. We thus created annotated data specifically for the pronoun *it*. We annotated 1020 instances in a collection of Science News articles (from 1995-2000), downloaded from the Science News website. We also annotated 709 instances in the WSJ portion of the DARPA TIPSTER Project [Harman, 1992], and 279 instances in the English portion of the Europarl Corpus [Koehn, 2005]. We take the first half of each of the subsets for training, the next quarter for development and the final quarter for testing, creating an aggregate set with 1070 training, 533 development and 534 test examples.

A single annotator (A_1) labeled all three data sets, while two additional annotators not connected with the project (A_2 and A_3) were asked to separately re-annotate a portion of each, so that inter-annotator agreement could be calculated. A_1 and A_2 agreed on 96% of annotation decisions, while A_1-A_3, and A_2-A_3, agreed on 91% and 93% of decisions, respectively. The *Kappa* statistic [Jurafsky and Martin, 2000, page 315], with $\Pr(E)$ computed from the confusion matrices, was a high 0.90 for A_1-A_2, and 0.79 and 0.81 for the

[7] Adapted from the Bow-toolkit [McCallum, 1996].

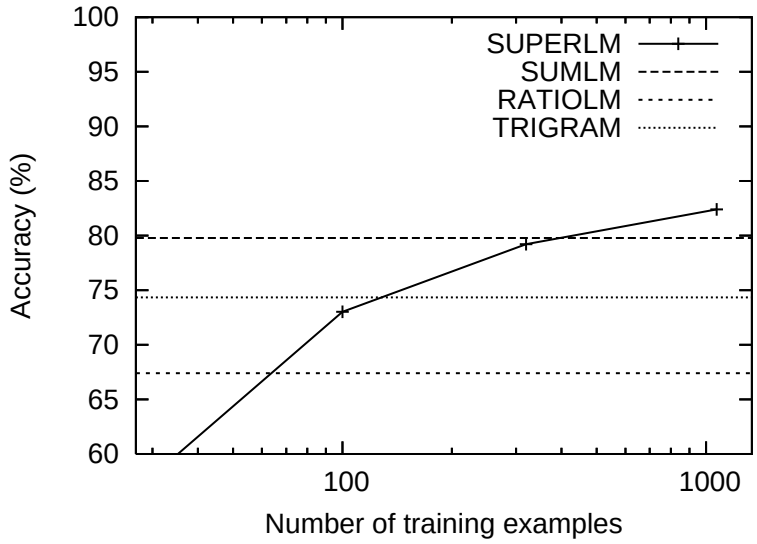


Figure 3.4: Non-referential detection learning curve

other pairs, around the 0.80 considered to be good reliability. These are, perhaps surprisingly, the only *it*-annotation agreement statistics available for written text. They contrast favourably with the low agreement for categorizing *it* in spoken dialog [Müller, 2006].

3.7.4 Non-referential Pronoun Detection Results

Main Results

For non-referential pronoun detection, BASE (always choosing referential) achieves 59.4%, while SUPERLM reaches 82.4%. RATIOLM, with no tuned thresholds, performs worst (67.4%), while TRIGRAM (74.3%) and SUMLM (79.8%) achieve reasonable performance by comparing scores for *it* and *they*. All differences are statistically significant (McNemar's test, $p<0.05$), except between SUPERLM and SUMLM.

In very similar results from [Bergsma *et al.*, 2008b] (but under slightly different experimental conditions; Section 3.7.5), the SUPERLM classifier was shown to strongly outperform rule-based systems for non-referential detection, across a range of text genres.

Learning Curves

As this is our only task for which substantial effort was needed to create training data, we are particularly interested in the learning rate of SUPERLM (Figure 3.4). After 1070 examples, it does not yet show signs of plateauing. Here, SUPERLM uses double the number of fillers (hence double the parameters) that were used in spelling correction, and spelling performance did not level-off until after 10K training examples. Thus labeling an order of magnitude more data will likely also yield further improvements in SUPERLM.

3.7.5 Further Analysis and Discussion

We now describe some work from [Bergsma *et al.*, 2008b] that further analyzes the performance of the non-referential classifier. The performance figures quoted in this section are

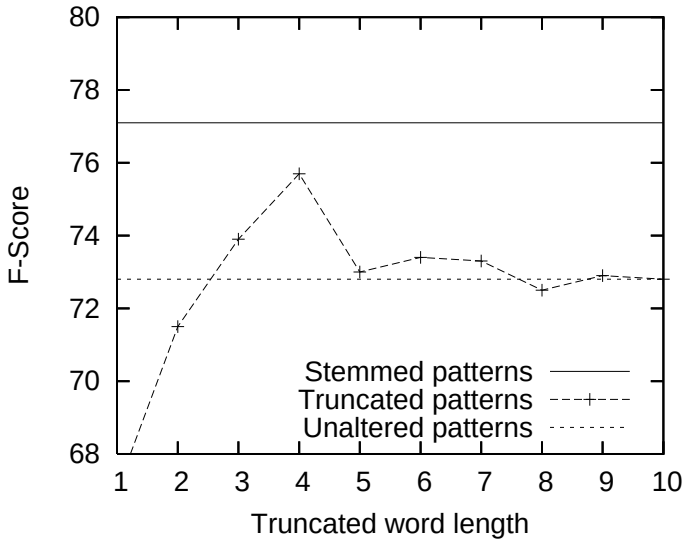


Figure 3.5: Effect of pattern-word truncation on non-referential *it* detection.

not directly comparable to the above work because they used a different split of training and testing data, and because experiments were conducted with a maximum entropy classifier rather than an SVM. However, this previous work nevertheless provides useful insights into the performance of SUPERLM on this task. Full details are available in [Bergsma *et al.*, 2008b]. To analyze the output of our system in greater detail, we now also report on the precision, recall, and F-score of the classifier (defined in Section 2.3.2).

Stemming vs. Simple Truncation

Since applying an English stemmer to the context words (Section 3.7.2) reduces the portability of the distributional technique, we investigated the use of more portable pattern abstraction. Figure 3.5 compares the use of the stemmer to simply truncating the words in the patterns at a certain maximum length. Using no truncation (Unaltered) drops the F-Score by 4.3%, while truncating the patterns to a length of four only drops the F-Score by 1.4%, a difference which is not statistically significant. Simple truncation may be a good option for other languages where stemmers are not readily available. The optimum truncation size will likely depend on the length of the base forms of words in that language. For real-world application of our approach, truncation also reduces the table sizes (and thus storage and look-up costs) of any pre-compiled *it*-pattern database.

A Human Study

We also wondered, what is the effect of making a classification based solely on, in aggregate, four words of context on either side of *it*. Another way to view the limited context is to ask, given the amount of context we have, are we making optimum use of it? We answer this by seeing how well humans can do with the same information. Our system uses 5-gram context patterns that together span from four-to-the-left to four-to-the-right of the pronoun. We thus provide these same nine-token windows to our human subjects, and ask them to decide whether the pronouns refer to previous noun phrases or not, based on these contexts.

System	P	R	F	Acc
SUPERLM	80.0	73.3	76.5	86.5
Human-1	92.7	63.3	75.2	87.5
Human-2	84.0	70.0	76.4	87.0
Human-3	72.2	86.7	78.8	86.0

Table 3.4: Human vs. computer non-referential *it* detection (%).

Subjects first performed a dry-run experiment on separate development data. They were shown their errors and sources of confusion were clarified. They then made the judgments unassisted on a final set of 200 test examples. Three humans performed the experiment. Their results show a range of preferences for precision versus recall, with F-Score and Accuracy broadly similar to SUPERLM (Table 3.4). These results show that our distributional approach is already getting good leverage from the limited context information, around that achieved by our best human.

Error Analysis

It is instructive to inspect the twenty-five test instances that our system classified incorrectly, given human performance on this same set. Seventeen of the twenty-five system errors were also made by one or more human subjects, suggesting system errors are also mostly due to limited context. For example, one of these errors was for the context: "it takes an astounding amount..." Here, the non-referential nature of the instance is not apparent without the infinitive clause that ends the sentence: "... of time to compare very long DNA sequences with each other."

Six of the eight errors unique to the system were cases where the system falsely said the pronoun was non-referential. Four of these could have referred to entire sentences or clauses rather than nouns. These confusing cases, for both humans and our system, result from our definition of a referential pronoun: pronouns with verbal or clause antecedents are considered non-referential. If an antecedent verb or clause is replaced by a nominalization (*Smith researched...* to *Smith's research*), then a neutral pronoun, in the same context, becomes referential. When we inspect the probabilities produced by the maximum entropy classifier, we see only a weak bias for the non-referential class on these examples, reflecting our classifier's uncertainty. It would likely be possible to improve accuracy on these cases by encoding the presence or absence of preceding nominalizations as a feature of our classifier.

Another false non-referential decision is for the phrase "... machine he had installed it on." The *it* is actually referential, but the extracted patterns (e.g. "he had install * on") are nevertheless usually filled with *it* (this example also suggests using filler counts for the word "the" as a feature when *it* is the last word in the pattern). Again, it might be possible to fix such examples by leveraging the preceding discourse. Notably, the first noun-phrase before the context is the word "software." There is strong compatibility between the pronoun-parent "install" and the candidate antecedent "software." In a full coreference resolution system, when the anaphora resolution module has a strong preference to link *it* to an antecedent (which it should when the pronoun is indeed referential), we can override a weak non-referential probability. Non-referential *it* detection should not be a pre-processing step, but rather part of a globally-optimal configuration, as was done for general noun phrase

anaphoricity by [Denis and Baldridge, 2007].

The suitability of this kind of approach to correcting some of our system's errors is especially obvious when we inspect the probabilities of the maximum entropy model's output decisions on the test set. Where the maximum entropy classifier makes mistakes, it does so with less confidence than when it classifies correct examples. The average predicted probability of the incorrect classifications is 76.0% while the average probability of the correct classifications is 90.3%. Many incorrect decisions are ready to switch sides; our next step will be to use features based on the preceding discourse and the candidate antecedents to help give the incorrect classifications a helpful push.

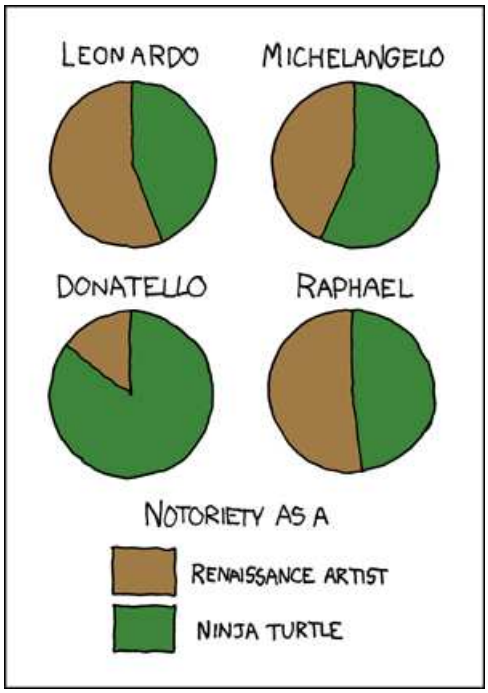
3.8 Conclusion

We proposed a unified view of using web-scale N-gram models for lexical disambiguation. State-of-the-art results by our supervised and unsupervised systems demonstrate that it is not only important to use the largest corpus, but to get maximum information from this corpus. Using the Google 5-gram data not only provides better accuracy than using page counts from a search engine, but facilitates the use of more context of various sizes and positions. The TRIGRAM approach, popularized by Lapata and Keller [2005], clearly underperforms the unsupervised SUMLM system on all three applications.

In each of our tasks, the candidate set was pre-defined, and training data was available to train the supervised system. While SUPERLM achieves the highest performance, the simpler SUMLM, which uses uniform weights, performs nearly as well as SUPERLM, and exceeds it for less training data. Unlike SUPERLM, SUMLM could easily be used in cases where the candidate sets are generated dynamically; for example, to assess the contextual compatibility of preceding-noun candidates for anaphora resolution.

Chapter 4

Improved Natural Language Learning via Variance-Regularization Support Vector Machines



XKCD comic: Ninja Turtles `http://xkcd.com/197/`. The beauty of this comic is that it was also constructed using co-occurrence counts from the Google search engine. That is, the artist counted the number of pages for *Leonardo* and *turtle* vs. the number of pages for *Leonardo* and *artist*.

The previous chapter presented SUPERLM, a supervised classifier that uses web-scale N-gram counts as features. The classifier was trained as a multi-class SVM. In this chap-

[0] A version of this chapter has been published as [Bergsma *et al.*, 2010b]

ter, we present a simple technique for learning better SVMs using fewer training examples. Rather than using the standard SVM regularization, we regularize toward low weight-variance. Our new SVM objective remains a convex quadratic function of the weights, and is therefore computationally no harder to optimize than a standard SVM. Variance regularization is shown to enable improvements in the learning rates of the SVMs on the three lexical disambiguation tasks studied in the previous chapter.

4.1 Introduction

Discriminative training is commonly used in NLP and speech to scale the contribution of different models or systems in a combined predictor. For example, discriminative training can be used to scale the contribution of the language model and translation model in machine translation [Och and Ney, 2002]. Without training data, it is often reasonable to weight the different models equally. We propose a simple technique that exploits this intuition for better learning with fewer training examples. We regularize the feature weights in a support vector machine [Cortes and Vapnik, 1995] toward a low-variance solution. Since the new SVM quadratic program is convex, it is no harder to optimize than the standard SVM objective.

When training data is generated through human effort, faster learning saves time and money. When examples are labeled automatically, through user feedback [Joachims, 2002] or from textual pseudo-examples [Smith and Eisner, 2005; Okanohara and Tsujii, 2007], faster learning can reduce the lag before a new system is useful.

We demonstrate faster learning on the same lexical disambiguation tasks evaluated in the previous chapter. Recall that in a lexical disambiguation task, a system predicts a label for a word in text, based on the word's context. Possible labels include part-of-speech tags, named-entity types, and word senses. A number of disambiguation systems make predictions with the help of N-gram counts from a web-scale auxiliary corpus, typically acquiring these counts via a search-engine or N-gram corpus (Section 3.2.1).

Ultimately, when discriminative training is used to set weights on various counts in order to make good classifications, many of the learned feature weights have similar values. Good weights have low variance.

For example, consider the task of preposition selection. A system selects the most likely preposition given the context, and flags a possible error if it disagrees with the user's choice:

- I worked in Russia **from** 1997 to 2001.
- I worked in Russia ***during** 1997 to 2001.

Chapter 3 presented SUPERLM, which uses a variety of web counts to predict the correct preposition. SUPERLM has features for COUNT(*in Russia* from), COUNT(*Russia* from *1997*), COUNT(from *1997 to*), etc. If these are high, **from** is predicted. Similarly, there are features for COUNT(*in Russia* during), COUNT(*Russia* during *1997*), COUNT(during *1997 to*). These features predict **during**. All counts are in the log domain. The task has thirty-four different prepositions to choose from. A 34-way classifier is trained on examples of correct preposition usage; it learns which context positions and sizes are most reliable and assigns feature weights accordingly.

In Chapter 3, we saw that a very strong unsupervised baseline, however, is to simply weight all the count features equally. In fact, the supervised approach required over 30,000 training examples before it outperformed this baseline. In contrast, we show here that

by regularizing a classifier toward equal weights, a supervised predictor outperforms the unsupervised approach after only ten examples, and does as well with 1000 examples as the standard classifier does with 100,000.

Section 4.2 first describes a general multi-class SVM. We call the base vector of information used by the SVM the *attributes*. A standard multi-class SVM creates features for the cross-product of attributes and classes. E.g., the attribute COUNT(*Russia* during *1997*) is not only a feature for predicting the preposition **during**, but also for predicting the 33 other prepositions. The SVM must therefore learn to disregard many irrelevant features. We observe that this is not necessary, and develop an SVM that only uses the relevant attributes in the score for each class. Building on this efficient framework, we incorporate variance regularization into the SVM's quadratic program.

We apply our algorithms to the three tasks studied in Chapter 3: preposition selection, context-sensitive spelling correction, and non-referential pronoun detection. We reproduce the Chapter 3 results using a multi-class SVM. Our new models achieve much better accuracy with fewer training examples. We also exceed the accuracy of a reasonable alternative technique for increasing the learning rate: including the output of the unsupervised system as a feature in the classifier.

Variance regularization is an elegant addition to the suite of methods in NLP that improve performance when access to labeled data is limited. Section 4.5 discusses some related approaches. While we motivate our algorithm as a way to learn better weights when the features are counts from an auxiliary corpus, there are other potential uses of our method. We outline some of these in Section 4.6, and note other directions for future research.

4.2 Three Multi-Class SVM Models

We describe three max-margin multi-class classifiers and their corresponding quadratic programs. Although we describe linear SVMs, they can be extended to nonlinear cases in the standard way by writing the optimal function as a linear combination of kernel functions over the input examples.

In each case, after providing the general technique, we relate the approach to our motivating application: learning weights for count features in a discriminative web-scale N-gram model.

4.2.1 Standard Multi-Class SVM

We define a K-class SVM following [Crammer and Singer, 2001]. This is a generalization of binary SVMs [Cortes and Vapnik, 1995]. We have a set $\{(\bar{x}^1, y^1), ..., (\bar{x}^M, y^M)\}$ of M training examples. Each $\bar{x}$ is an N-dimensional attribute vector, and $y \in \{1, ..., K\}$ are classes. A classifier, H, maps an attribute vector, $\bar{x}$, to a class, y. H is parameterized by a K-by-N matrix of weights, $\mathbf{W}$:

$$H_{\mathbf{W}}(\bar{x}) = \underset{r=1}{\overset{K}{\operatorname{argmax}}}\{\bar{W}_r \cdot \bar{x}\} \tag{4.1}$$

where $\bar{W}_r$ is the rth row of $\mathbf{W}$. That is, the predicted label is the index of the row of $\mathbf{W}$ that has the highest inner-product with the attributes, $\bar{x}$.

We seek weights such that the classifier makes few errors on training data and generalizes well to unseen data. There are KN weights to learn, for the cross-product of attributes and classes. The most common approach is to train K separate one-versus-all binary SVMs, one for each class. The weights learned for the rth SVM provide the weights $\bar{W}_r$ in (4.1). We call this approach **OVA-SVM**. Note in some settings various one-versus-one strategies may be more effective than one-versus-all [Hsu and Lin, 2002].

The weights can also be found using a single constrained optimization [Vapnik, 1998; Weston and Watkins, 1998]. Following the soft-margin version in [Crammer and Singer, 2001]:

$$\begin{aligned} \min_{\mathbf{W},\xi^1,\ldots,\xi^M} \quad & \frac{1}{2}\sum_{i=1}^{K}||\bar{W}_i||^2 + C\sum_{i=1}^{m}\xi^i \\ \text{subject to} : \forall i \quad & \xi^i \geq 0 \\ \forall r \neq y^i, \quad & \bar{W}_{y^i}\cdot\bar{x}^i - \bar{W}_r\cdot\bar{x}^i \geq 1-\xi^i \end{aligned} \tag{4.2}$$

The constraints require the correct class to be scored higher than other classes by a certain margin, with slack for non-separable cases. Minimizing the weights is a form of regularization. Tuning the C-parameter controls the emphasis on regularization versus separation of training examples.

We call this the **K-SVM**. The K-SVM outperformed the OVA-SVM in [Crammer and Singer, 2001], but see [Rifkin and Klautau, 2004]. The popularity of K-SVM is partly due to convenience; it is included in popular SVM software like SVM-*multiclass*[1] and LIBLINEAR [Fan *et al.*, 2008].

Note that with two classes, K-SVM is less efficient than a standard binary SVM. A binary classifier outputs class 1 if ($\bar{w}\cdot\bar{x} > 0$) and class 2 otherwise. The K-SVM encodes a binary classifier using $\bar{W}_1 = \bar{w}$ and $\bar{W}_2 = -\bar{w}$, therefore requiring twice the memory of a binary SVM. However, both binary and 2-class formulations have the same solution [Weston and Watkins, 1998].

Web-Scale N-gram K-SVM

K-SVM was used to combine the N-gram counts in Chapter 3. This was the SUPERLM model. Recall that for preposition selection, attributes were web counts of patterns filled with 34 prepositions, corresponding to the 34 classes. Each preposition serves as the *filler* of each *context pattern*. Fourteen patterns were used for each filler: all five 5-grams, four 4-grams, three 3-grams, and two 2-grams spanning the position to be predicted. There are $N = 14 * 34 = 476$ total attributes, and therefore $KN = 476 * 34 = 16184$ weights in the $\mathbf{W}$ matrix.

Figure 4.1 depicts the optimization problem for the preposition selection classifier. For the ith training example, the optimizer must set the weights such that the score for the true class (*from*) is higher than the scores of all the other classes by a margin of 1. Otherwise, it must use the slack parameter, ξ^i. The score is the linear product of the preposition-specific weights, $\bar{W}_r$ and all the features, $\bar{x}^i$. For illustration, seven of the thirty-four total classes are depicted. Note these constraints must be collectively satisfied across all training examples.

A K-SVM classifier can potentially exploit very subtle information for this task. Let $\bar{W}_{in}$ and $\bar{W}_{before}$ be weights for the classes **in** and **before**. Notice some of the attributes

[1] `http://svmlight.joachims.org/svm_multiclass.html`

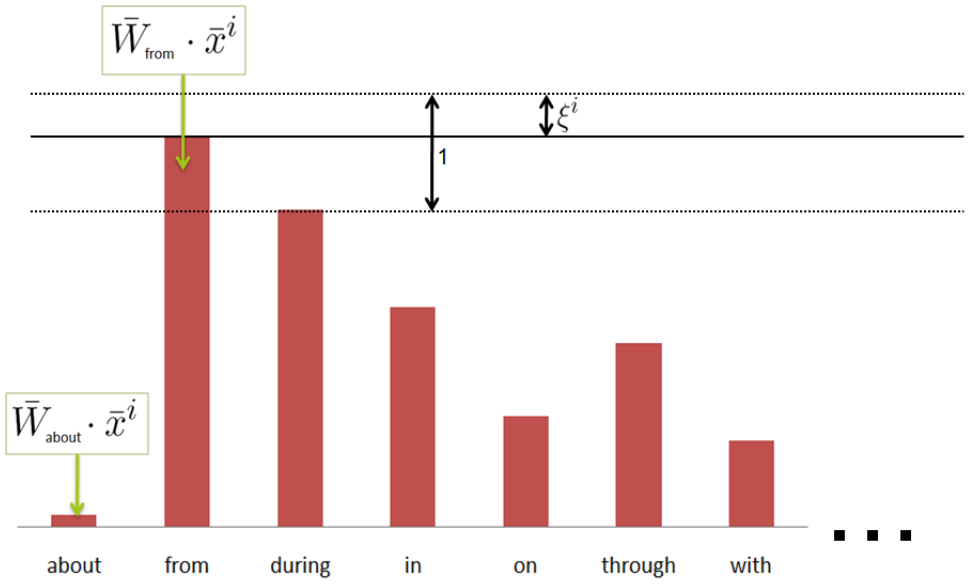


Figure 4.1: Multi-class classification for web-scale N-gram models

weighted in the inner products $\bar{W}_{before} \cdot \bar{x}$ and $\bar{W}_{in} \cdot \bar{x}$ will be for counts of the preposition *after*. Relatively high counts for a context with *after* should deter us from choosing **in** more than from choosing **before**. These correlations can be encoded in the classifier via the corresponding weights on *after*-counts in $\bar{W}_{in}$ and $\bar{W}_{before}$. Our experiments address how useful these correlations are and how much training data is needed before they can be learned and exploited effectively.

4.2.2 SVM with Class-Specific Attributes

Suppose we can partition our attribute vectors into sub-vectors that only include attributes that we declare as relevant to the corresponding class: $\bar{x} = (\bar{x}_1, ..., \bar{x}_K)$. We develop a classifier that only uses the class-specific attributes in the score for each class. The classifier uses an N-dimensional weight vector, $\bar{w}$, which follows the attribute partition, $\bar{w} = (\bar{w}_1, ..., \bar{w}_K)$. The classifier is:

$$H_{\bar{w}}(\bar{x}) = \underset{r=1}{\overset{K}{\text{argmax}}}\{\bar{w}_r \cdot \bar{x}_r\} \tag{4.3}$$

We call this classifier the **CS-SVM** (an SVM with **C**lass-**S**pecific attributes).

The weights can be determined using the follow (soft-margin) optimization:

$$\begin{aligned} \min_{\bar{w},\xi^1,...,\xi^m} \quad & \frac{1}{2}\bar{w}^T\bar{w} + C\sum_{i=1}^{m}\xi^i \\ \text{subject to} : \forall i \quad & \xi^i \geq 0 \\ \forall r \neq y^i, \quad & \bar{w}_{y^i} \cdot \bar{x}^i_{y^i} - \bar{w}_r \cdot \bar{x}^i_r \geq 1 - \xi^i \end{aligned} \tag{4.4}$$

There are several advantages to this formulation. Foremost, rather than having KN weights, it can have only N. For linear classifiers, the number of examples needed to

reach optimum performance is at most linear in the number of weights [Vapnik, 1998; Ng and Jordan, 2002]. In fact, both the total number and number of *active* features per example decrease by K. Thus this reduction saves far more memory than what could be obtained by an equal reduction in dimensionality via pruning infrequent attributes.

Also, note that unlike the K-SVM (Section 4.2.1), in the binary case the CS-SVM is completely equivalent (thus equally efficient) to a standard SVM.

We will not always *a priori* know the class associated with each attribute. Also, some attributes may be predictive of multiple classes. In such cases, we can include ambiguous attributes in every sub-vector (needing N+$D(K$-$1)$ total weights if D attributes are duplicated). In the degenerate case where every attribute is duplicated, CS-SVM is equivalent to K-SVM; both have KN weights.

Optimization as a Binary SVM

We could solve the optimization problem in (4.4) directly using a quadratic programming solver. However, through an equivalent transformation into a binary SVM, we can take advantage of efficient, custom SVM optimization algorithms.

We follow [Har-Peled *et al.*, 2003] in transforming a multi-class example into a set of binary examples, each specifying a constraint from (4.4). We extend the attribute sub-vector corresponding to each class to be N-dimensional. We do this by substituting zero-vectors for all the other sub-vectors in the partition. The attribute vector for the rth class is then $\bar{z}_r = (\bar{0}, ..., \bar{0}, \bar{x}_r, \bar{0}, ..., \bar{0})$. This is known as Kesler's Construction and has a long history in classification [Duda and Hart, 1973; Crammer and Singer, 2003]. We then create binary rank constraints for a ranking SVM [Joachims, 2002] (ranking SVMs reduce to standard binary SVMs). We create K instances for each multi-class example $(\bar{x}^i, y^i)$, with the transformed vector of the true class, $\bar{z}_{y^i}$, assigned a higher-rank than all the other, equally-ranked classes, $\bar{z}_{\{r \neq y^i\}}$. Training a ranking SVM using these constraints gives the same weights as solving (4.4), but allows us to use efficient, custom SVM software.[2] Note the K-SVM can also be trained this way, by including every attribute in every sub-vector, as described earlier.

Web-Scale N-gram CS-SVM

Returning to our preposition selection example, an obvious attribute partition for the CS-SVM is to include as attributes for predicting preposition r only those counts for patterns filled with preposition r. Thus $\bar{x}_{in}$ will only include counts for context patterns filled with *in* and $\bar{x}_{before}$ will only include counts for context patterns filled with *before*. With 34 sub-vectors and 14 attributes in each, there are only $14 * 34 = 476$ total weights. In contrast, K-SVM had 16184 weights to learn.

It is instructive to compare the CS-SVM in (4.3) to the unsupervised SUMLM approach in Chapter 3. That approach can be written as:

$$H(\bar{x}) = \underset{r=1}{\overset{K}{\operatorname{argmax}}} \{\bar{1} \cdot \bar{x}_r\} \tag{4.5}$$

[2] One subtlety is whether to use a single slack, ξ^i, for all K-1 constraints per example i [Crammer and Singer, 2001], or a different slack for each constraint [Joachims, 2002]. Using the former may be better as it results in a tighter bound on empirical risk [Tsochantaridis *et al.*, 2005].

where $\bar{1}$ is an N-dimensional vector of ones. This is CS-SVM with all weights set to unity. The counts for each preposition are simply summed, and whichever one scores the highest is taken as the output (actually only a subset of the counts are used, see Section 4.4.1). As mentioned earlier, this system performs remarkably well on several tasks.

4.2.3 Variance Regularization SVMs

Suppose we choose our attribute partition well and train the CS-SVM on a sufficient number of examples to achieve good performance. It is a reasonable hypothesis that the learned weights will be predominantly positive. This is because each sub-vector $\bar{x}_r$ was chosen to only include attributes that are predictive of class r. Unlike the classifier in (4.1) which weighs positive and negative evidence together for each class, in CS-SVM, negative evidence only plays a roll as it contributes to the score of competing classes.

If all the attributes are equally important, the weights should be equal, as in the unsupervised approach in (4.5). If some are more important than others, the training examples should reflect this and the learner can adjust the weights accordingly.[3] In the absence of this training evidence, it is reasonable to bias the classifier toward an equal-weight solution.

Rather than the standard SVM regularization that minimizes the norm of the weights as in (4.4), we therefore regularize toward weights that have low variance. More formally, we can regard the set of weights, $w_1, ..., w_N$, as the distribution of a discrete random variable, W. We can calculate the mean and variance of this variable from its distribution. We seek a variable that has low variance.

We begin with a more general objective and then explain how a specific choice of covariance matrix, $\mathbf{C}$, minimizes the variance of the weights. We propose the regularizer:

$$\begin{aligned} \min_{\bar{w},\xi^1,...,\xi^m} \quad & \frac{1}{2}\bar{w}^T\mathbf{C}\bar{w} + C\sum_{i=1}^{m}\xi^i \\ \text{subject to} : \forall i \quad & \xi^i \geq 0 \\ \forall r \neq y^i, \quad & \bar{w}_{y^i}\cdot\bar{x}^i_{y^i} - \bar{w}_r\cdot\bar{x}^i_r \geq 1-\xi^i \end{aligned} \tag{4.6}$$

where $\mathbf{C}$ is a normalized covariance matrix such that $\sum_{i,j} C_{i,j} = 0$. This ensures uniform weight vectors receive zero regularization penalty. Since all covariance matrices are positive semi-definite, the quadratic program (QP) remains convex in $\bar{w}$, and thus amenable to general purpose QP-solvers.

Since the unsupervised system in (4.5) has zero weight variance, the SVM learned in (4.6) should do as least as well as (4.5) as we tune the C-parameter on development data. That is, as C approaches zero, variance minimization becomes the sole objective of (4.6), and uniform weights are produced.

We use covariance matrices of the form:

$$\mathbf{C} = \mathrm{diag}(\bar{p}) - \bar{p}\bar{p}^T \tag{4.7}$$

where $\mathrm{diag}(\bar{p})$ is the matrix constructed by putting $\bar{p}$ on the main diagonal. Here, $\bar{p}$ is an arbitrary N-dimensional weighting vector, such that $p \geq 0$ and $\sum_i p_i = 1$. The vector $\bar{p}$ dictates the contribution of each w_i to the mean and variance of the weights in $\bar{w}$. It is easy to see that $\sum_{i,j} C_{i,j} = \sum_i p_i - \sum_i \sum_j p_i p_j = 0$.

[3] E.g., recall from Chapter 3 that the true preposition might be better predicted by the counts of patterns that tend to include the preposition's grammatical object, i.e., patterns that include more right-context.

We now show that $\bar{w}^T(\text{diag}(\bar{p}) - \bar{p}\bar{p}^T)\bar{w}$ expresses the variance of the weights in $\bar{w}$ with respect to the probability weighting $\bar{p}$. The variance of a random variable with mean $E[W] = \mu$ is:

$$\text{Var}[W] = E[(W - \mu)^2] = E[W^2] - E[W]^2$$

The mean of the weights using probability weighting $\bar{p}$ is $E[W] = \bar{w}^T\bar{p} = \bar{p}\bar{w}$. Also, $E[W^2] = \bar{w}^T \text{diag}(\bar{p})\bar{w}$. Thus:

$$\begin{aligned}\text{Var}[W] &= \bar{w}^T \text{diag}(\bar{p})\bar{w} - (\bar{w}^T\bar{p})(\bar{p}\bar{w}) \\ &= \bar{w}^T(\text{diag}(\bar{p}) - \bar{p}\bar{p})\bar{w}\end{aligned}$$

In our experiments, we deem each weight to be equally important to the variance calculation, and set $p_i = \frac{1}{N}, \forall i = 1, \ldots, N$.

The goal of the regularization in (4.6) using $\mathbf{C}$ from (4.7) can be regarded as directing the SVM toward a good unsupervised system, regardless of the constraints (training examples). In some unsupervised systems, however, only a subset of the attributes are used. In other cases, distinct subsets of weights should have low variance, rather than minimizing the variance across all weights. There are examples of these situations in Section 4.4.

We can account for these cases in our QP. We provide separate terms in our quadratic function for the subsets of $\bar{w}$ that should have low variance. Suppose we create L subsets of $\bar{w}$: $\tilde{\omega}_1, ...\tilde{\omega}_L$, where $\tilde{\omega}_j$ is $\bar{w}$ with elements set to zero that are not in subset j. We then minimize $\frac{1}{2}(\tilde{\omega}_1^T\mathbf{C}_1\tilde{\omega}_1 + ... + \tilde{\omega}_L^T\mathbf{C}_L\tilde{\omega}_L)$. If the terms in subset j have low variance, $\mathbf{C}_j$ = $\mathbf{C}$ from (4.7) is used. If the subset corresponds to attributes that are not *a priori* known to be useful, an identity matrix can instead be used, $\mathbf{C}_j = \mathbf{I}$, and these weights will be regularized toward zero as in a standard SVM.[4]

Variance regularization therefore exploits extra knowledge by the system designer. The designer decides which weights should have similar values, and the SVM is biased to prefer this solution.

One consequence of being able to regularize different subsets of weights is that we can also apply variance regularization to the standard multi-class SVM (Section 4.2.1). We can use an identity $\mathbf{C}_i$ matrix for all *irrelevant* weights, i.e., weights that correspond to class-attribute pairs where the attribute is not directly relevant to the class. In our experiments, however, we apply variance regularization to the more efficient CS-SVM.

We refer to a CS-SVM trained using the variance minimization quadratic program as the **VAR-SVM**.

Web-Scale N-gram VAR-SVM

If variance regularization is applied to all weights, attributes COUNT(*in Russia* during), COUNT(*Russia* during *1997*), and COUNT(during *1997 to*) will be encouraged to have similar weights in the score for class **during**. Furthermore, these will be weighted similarly to other patterns, filled with other prepositions, used in the scores for other classes.

Alternatively, we could minimize the variance separately over all 5-gram patterns, then over all 4-gram patterns, etc., or over all patterns with a filler in the same position. In our

[4]Weights must appear in $\geq$1 subsets (possibly only in the $\mathbf{C}_j = \mathbf{I}$ subset). Each occurs in at most one in our experiments. Note it is straightforward to express this as a single covariance matrix regularizer over $\bar{w}$; we omit the details.

experiments, we took a very simple approach: we minimized the variance of all attributes that are weighted equally in the unsupervised baselines. If a feature is not included in an unsupervised baseline, it is regularized toward zero.

4.3 Experimental Details

We use the data sets from Chapter 3. Recall that in each case a classifier makes a decision for a particular word based on the word's surrounding context. The attributes of the classifier are the log counts of different fillers occurring in the context patterns. We retrieve counts from the web-scale Google Web 5-gram Corpus [Brants and Franz, 2006], which includes N-grams of length one to five. We apply add-one smoothing to all counts. Every classifier also has bias features (for every class). We simply include, where appropriate, attributes that are always unity.

We use LIBLINEAR [Fan *et al.*, 2008] to train K-SVM and OVA-SVM, and SVM^{rank} [Joachims, 2006] to train CS-SVM. For VAR-SVM, we solve the primal form of the quadratic program directly in [CPLEX, 2005], a general optimization package.

We vary the number of training examples for each classifier. The C-parameters of all SVMs are tuned on development data. We evaluate using **accuracy**: the percentage of test examples that are classified correctly. We also provide the accuracy of the majority-class baseline and best unsupervised system, as defined in Chapter 3.

As an alternative way to increase the learning rate, we augment a classifier's features using the output of the unsupervised system: For each class, we include one feature for the sum of all counts (in the unsupervised system) that predict that class. We denote these augmented systems with a $+$ as in K-SVM$^+$ and CS-SVM$^+$.

4.4 Applications

4.4.1 Preposition Selection

Following the set-up in Chapter 3, a classifier selects a preposition from 34 candidates, using counts for filled 2-to-5-gram patterns. We again use the same 100K training, 10K development, and 10K test examples. The unsupervised approach sums the counts of all 3-to-5-gram patterns for each preposition. We therefore regularize the variance of the 3-to-5-gram weights in VAR-SVM, and simultaneously minimize the norm of the 2-gram weights.

Results

The majority-class is the preposition **of**; it occurs in 20.3% of test examples. The unsupervised system scores 73.7%. For further perspective on these results, recall that [Chodorow *et al.*, 2007] achieved 69% with 7M training examples, while [Tetreault and Chodorow, 2008] found the human performance was around 75%. However, these results are not directly comparable as they are on different data.

Table 4.1 gives the accuracy for different amounts of training data. Here, as in the other tasks, K-SVM mirrors the learning rate from Chapter 3. There are several distinct phases among the relative ranking of the systems. For smaller amounts of training data ($\leq$1000 examples) K-SVM performs worst, while VAR-SVM is statistically significantly better than all

System	Training Examples 10	100	1K	10K	100K
OVA-SVM	16.0	50.6	66.1	71.1	73.5
K-SVM	13.7	50.0	65.8	72.0	74.7
K-SVM$^+$	22.2	56.8	70.5	73.7	**75.2**
CS-SVM	27.1	58.8	69.0	73.5	74.2
CS-SVM$^+$	39.6	64.8	71.5	74.0	74.4
VAR-SVM	**73.8**	**74.2**	**74.7**	**74.9**	74.9

Table 4.1: Accuracy (%) of preposition-selection SVMs. Unsupervised (SUMLM) accuracy is 73.7%.

System	Training Examples 10	100	1K	10K	100K
CS-SVM	86.0	93.5	95.1	**95.7**	95.7
CS-SVM$^+$	91.0	94.9	95.3	**95.7**	95.7
VAR-SVM	**94.9**	**95.3**	**95.6**	**95.7**	**95.8**

Table 4.2: Accuracy (%) of spell-correction SVMs. Unsupervised (SUMLM) accuracy is 94.8%.

other systems, and always exceeds the performance of the unsupervised approach.[5] Augmenting the attributes with sum counts (the + systems) strongly helps with fewer examples, especially in conjunction with the more efficient CS-SVM. However, VAR-SVM clearly helps more. We noted earlier that VAR-SVM is guaranteed to do as well as the unsupervised system on the development data, but here we confirm that it can also exploit even small amounts of training data to further improve accuracy.

CS-SVM outperforms K-SVM except with 100K examples, while OVA-SVM is better than K-SVM for small amounts of data.[6] K-SVM performs best with all the data; it uses the most expressive representation, but needs 100K examples to make use of it. On the other hand, feature augmentation and variance regularization provide diminishing returns as the amount of training data increases.

4.4.2 Context-Sensitive Spelling Correction

Recall that in context-sensitive spelling correction, for every occurrence of a word in a predefined confusion set (e.g. {*cite, sight, cite*}), the classifier selects the most likely word from the set. We use the five confusion sets from Chapter 3; four are binary and one is a 3-way classification. We use 100K training, 10K development, and 10K test examples for each, and average accuracy across the sets. All 2-to-5 gram counts are used in the unsupervised system, so the variance of all weights is regularized in VAR-SVM.

[5] Significance is calculated using a χ^2 test over the test set correct/incorrect contingency table.

[6] [Rifkin and Klautau, 2004] argue OVA-SVM is as good as K-SVM, but this is "predicated on the assumption that the classes are 'independent'," i.e., that examples from class 0 are no closer to class 1 than to class 2. This is not true of this task (e.g. $\bar{x}_{before}$ is closer to $\bar{x}_{after}$ than $\bar{x}_{in}$, etc.).

System	Training Examples 10	100	1K
CS-SVM	59.0	71.0	84.3
CS-SVM$^+$	59.4	74.9	**84.5**
VAR-SVM	**70.2**	76.2	**84.5**
VAR-SVM+FreeB	64.2	**80.3**	**84.5**

Table 4.3: Accuracy (%) of non-referential detection SVMs. Unsupervised (SUMLM) accuracy is 79.8%.

Results

On this task, the majority-class baseline is much higher, 66.9%, and so is the accuracy of the top unsupervised system: 94.8%. Since four of the five sets are binary classifications, where K-SVM and CS-SVM are equivalent, we only give the accuracy of the CS-SVM (it does perform better than K-SVM on the one 3-way set). VAR-SVM again exceeds the unsupervised accuracy for all training sizes, and generally performs as well as the augmented CS-SVM$^+$ using an order of magnitude less training data (Table 4.2). Differences from $\leq$1K are significant.

4.4.3 Non-Referential Pronoun Detection

We use the same data and fillers as Chapter 3, and preprocess the N-gram corpus in the same way.

Recall that the unsupervised system picks non-referential if the difference between the summed count of *it* fillers and the summed count of *they* fillers is above a threshold (note this no longer fits Equation (4.5), with consequences discussed below). We thus separately minimize the variance of the *it* pattern weights and the *they* pattern weights. We use 1K training, 533 development, and 534 test examples.

Results

The most common class is **referential**, occurring in 59.4% of test examples. The unsupervised system again does much better, at 79.8%.

Annotated training examples are much harder to obtain for this task and we experiment with a smaller range of training sizes (Table 4.3). The performance of VAR-SVM exceeds the performance of K-SVM across all training sizes (bold accuracies are significantly better than either CS-SVM for $\leq$100 examples). However, the gains were not as large as we had hoped, and accuracy remains worse than the unsupervised system when not using all the training data. When using all the data, a fairly large C-parameter performs best on development data, so regularization plays less of a role.

After development experiments, we speculated that the poor performance relative to the unsupervised approach was related to class bias. In the other tasks, the unsupervised system chooses the highest summed score. Here, the difference in *it* and *they* counts is compared to a *threshold*. Since the bias feature is regularized toward zero, then, unlike the other tasks, using a low C-parameter does not produce the unsupervised system, so performance can begin below the unsupervised level.

Since we wanted the system to learn this threshold, even when highly regularized, we removed the regularization penalty from the bias weight, letting the optimization freely set the weight to minimize training error. With more freedom, the new classifier (VAR-SVM+FreeB) performs worse with 10 examples, but exceeds the unsupervised approach with 100 training points. Although this was somewhat successful, developing better strategies for bias remains useful future work.

4.5 Related Work

There is a large body of work on regularization in machine learning, including work that uses positive semi-definite matrices in the SVM quadratic program. The graph Laplacian has been used to encourage geometrically-similar feature vectors to be classified similarly [Belkin *et al.*, 2006]. An appealing property of these approaches is that they incorporate information from unlabeled examples. [Wang *et al.*, 2006] use Laplacian regularization for the task of dependency parsing. They regularize such that features for distributionally-similar words have similar weights. Rather than penalize pairwise differences proportional to a similarity function as they do, we simply penalize weight variance.

In the field of computer vision, [Tefas *et al.*, 2001] (binary) and [Kotsia *et al.*, 2009] (multi-class) also regularize weights with respect to a covariance matrix. They use labeled data to find the sum of the sample covariance matrices from each class, similar to linear discriminant analysis. We propose the idea in general, and instantiate with a different $\mathbf{C}$ matrix: a variance regularizer over $\bar{w}$. Most importantly, our instantiated covariance matrix does not require labeled data to generate.

In a Bayesian setting, [Raina *et al.*, 2006] model feature correlations in a logistic regression classifier. They propose a method to construct a covariance matrix for a multivariate Gaussian prior on the classifier's weights. Labeled data for other, related tasks is used to infer potentially correlated features on the target task. Like in our results, they found that the gains from modeling dependencies diminish as more training data is available.

We also mention two related online learning approaches. Similar to our goal of regularizing toward a good unsupervised system, [Crammer *et al.*, 2006] regularize $\bar{w}$ toward a (different) target vector at each update, rather than strictly minimizing $||\bar{w}||^2$. The target vector is the vector learned from the cumulative effect of previous updates. [Dredze *et al.*, 2008] maintain the variance of each weight and use this to guide the online updates. However, covariance between weights is not considered.

We believe new SVM regularizations in general, and variance regularization in particular, will increasingly be used in combination with related NLP strategies that learn better when labeled data is scarce. These may include: using more-general features, e.g. ones generated from raw text [Miller *et al.*, 2004; Koo *et al.*, 2008], leveraging out-of-domain examples to improve in-domain classification [Blitzer *et al.*, 2007; Daumé III, 2007], active learning [Cohn *et al.*, 1994; Tong and Koller, 2002], and approaches that treat unlabeled data as labeled, such as bootstrapping [Yarowsky, 1995], co-training [Blum and Mitchell, 1998], and self-training [McClosky *et al.*, 2006a].

4.6 Future Work

The primary direction of future research will be to apply the VAR-SVM to new problems and tasks. There are many situations where a system designer has an intuition about the

role a feature will play in prediction; the feature was perhaps added with this role in mind. By biasing the SVM to use features as intended, VAR-SVM may learn better with fewer training examples. The relationship between attributes and classes may be explicit when, e.g., a rule-based system is optimized via discriminative learning, or annotators justify their decisions by indicating the relevant attributes [Zaidan *et al.*, 2007]. Also, if features are *a priori* thought to have different predictive worth, the attribute *values* could be scaled such that variance regularization, as we formulated it, has the desired effect.

Other avenues of future work will be to extend the VAR-SVM in three directions: efficiency, representational power, and problem domain.

While we optimized the VAR-SVM objective in CPLEX, general purpose QP-solvers "do not exploit the special structure of [the SVM optimization] problem," and consequently often train in time super-linear with the number of training examples [Joachims *et al.*, 2009]. It would be useful to fit our optimization problem to efficient SVM training methods, especially for linear classifiers.

VAR-SVM's representational power could be extended by using non-linear SVMs. Kernels can be used with a covariance regularizer [Kotsia *et al.*, 2009]. Since $\mathbf{C}$ is positive semi-definite, the square root of its inverse is defined. We can therefore map the input examples using $(\mathbf{C}^{-\frac{1}{2}}\bar{x})$, and write an equivalent objective function in terms of kernel functions over the transformed examples.

Also, since structured-prediction SVMs build on the multi-class framework [Tsochantaridis *et al.*, 2005], variance regularization can be incorporated naturally into more-complex prediction systems, such as parsers, taggers, and aligners.

VAR-SVM may also help in new domains where annotated data is lacking. VAR-SVM should be stronger cross-domain than K-SVM; regularization with domain-neutral prior-knowledge can offset domain-specific biases. Learned weight vectors from other domains may also provide cross-domain regularization guidance. We discuss the connection between domain adaptation and regularization further in Section 5.7.

4.7 Conclusion

We presented variance-regularization SVMs, an approach to learning that creates better classifiers using fewer training examples. Variance regularization incorporates a bias for known good weights into the SVM's quadratic program. The VAR-SVM can therefore exploit extra knowledge by the system designer. Since the objective remains a convex quadratic function of the weights, the program is computationally no harder to optimize than a standard SVM. We also demonstrated how to design multi-class SVMs using only class-specific attributes, and compared the performance of this approach to standard multi-class SVMs on the task of preposition selection.

While variance regularization is most helpful on tasks with many classes and features, like preposition selection, it achieved gains on all our tasks when training with smaller sample sizes. It should be useful on a variety of other NLP problems.

Chapter 5

Creating Robust Supervised Classifiers via Web-Scale N-gram Data

> "It is said Hugo was on vacation when Les Misérables (which is over 1200 pages) was published. He telegraphed the single-character message '?' to his publisher, who replied with a single '!' [citation needed]"
> `http://en.wikipedia.org/wiki/Victor_hugo`

In this chapter, we systematically assess the value of using web-scale N-gram data in state-of-the-art supervised NLP classifiers, i.e., classifiers using conventional, non-count-based features. We compare classifiers that also include or exclude features for the counts of various N-grams, where the counts are obtained from a new web-scale auxiliary corpus. We show that including N-gram count features can advance the state-of-the-art accuracy on standard data sets for adjective ordering, spelling correction, noun compound bracketing, and verb part-of-speech disambiguation. More importantly, when operating on new domains, or when labeled training data is not plentiful, we show that using web-scale N-gram features is essential for achieving robust performance.

5.1 Introduction

As noted in Chapter 3, many NLP systems use web-scale N-gram counts [Keller and Lapata, 2003; Nakov and Hearst, 2005a; Brants *et al.*, 2007]. [Lapata and Keller, 2005] demonstrate good performance on eight tasks using unsupervised web-based models. They show web counts are superior to counts from a large corpus. In Chapter 3, we proposed unsupervised and supervised systems that use counts from Google's N-gram corpus [Brants and Franz, 2006]. In general, past research has shown that web-based models perform particularly well on *generation* tasks, where systems choose between competing sequences of output text (such as different spellings), as opposed to *analysis* tasks, where systems choose between abstract labels (such as part-of-speech tags, parse trees, or whether a pronoun is referential or not).

In this chapter, we address two natural and related questions which these previous studies leave open:

[0] A version of this chapter has been published as [Bergsma *et al.*, 2010c]

1. Is there a benefit in combining web-scale counts with the standard features used in state-of-the-art supervised approaches?
2. How well do web-based models perform on new domains or when labeled data is scarce?

We address these questions on two generation and two analysis tasks, using both existing N-gram data and a novel web-scale N-gram corpus that includes part-of-speech information (Section 5.2). While previous work has combined web-scale features with other features in specific classification problems [Modjeska *et al.*, 2003; Yang *et al.*, 2005; Vadas and Curran, 2007b; Tratz and Hovy, 2010], we provide a multi-task, multi-domain comparison.

Some may question why supervised learning with standard features is needed at all for generation problems. Why not solely rely on direct evidence from a giant corpus? For example, for the task of prenominal adjective ordering (Section 5.3), a system that needs to describe a ball that is both big and red can simply check that *big red* is more common on the web than *red big*, and order the adjectives accordingly.

It is, however, suboptimal to only use simple counts from N-gram data. For example, ordering adjectives by direct web evidence performs 7% worse than our best supervised system (Section 5.3.2). No matter how large the web becomes, there will always be plausible constructions that never occur. For example, there are currently no pages indexed by Google with the preferred adjective ordering for *bedraggled 56-year-old [professor]*. Also, in a particular domain, words may have a non-standard usage. Systems trained on labeled data can learn the domain usage and leverage other regularities, such as suffixes and transitivity for adjective ordering.

With these benefits, systems trained on labeled data have become the dominant technology in academic NLP. There is a growing recognition, however, that these systems are highly domain dependent. For example, parsers trained on annotated newspaper text perform poorly on other genres [Gildea, 2001]. While many approaches have adapted NLP systems to specific domains [Tsuruoka *et al.*, 2005; McClosky *et al.*, 2006b; Blitzer *et al.*, 2007; Daumé III, 2007; Rimell and Clark, 2008], these techniques assume the system knows on which domain it is being used, and that it has access to representative data in that domain. These assumptions are unrealistic in many real-world situations; for example, when automatically processing a heterogeneous collection of web pages. How well do supervised and unsupervised NLP systems perform when used uncustomized, *out-of-the-box* on new domains, and how can we best design our systems for robust *open-domain* performance?

Our results show that using web-scale N-gram data in supervised systems advances the state-of-the-art performance on standard analysis and generation tasks. More importantly, when operating out-of-domain, or when labeled data is not plentiful, using web-scale N-gram data not only helps achieve good performance – it is essential.

5.2 Experiments and Data

5.2.1 Experimental Design

We again evaluate the benefit of N-gram data on multi-class classification problems. For each task, we have some labeled data indicating the correct output for each example. We evaluate with **accuracy**: the percentage of examples correctly classified in test data. We

§	In-Domain (IN)	Out-of-Domain #1 (O1)	Out-of-Domain #2 (O2)
5.3	BNC [Malouf, 2000]	Gutenberg (new)	Medline (new)
5.4	NYT [Bergsma *et al.*, 2009b]	Gutenberg (new)	Medline (new)
5.5	WSJ [Vadas and Curran, 2007a]	Grolier [Lauer, 1995a]	Medline [Nakov, 2007]
5.6	WSJ [Marcus *et al.*, 1993]	Brown [Marcus *et al.*, 1993]	Medline [Kulick *et al.*, 2004]

Table 5.1: Data, with references, for tasks in § 5.3: Prenominal Adjective Ordering, § 5.4: Context-Sensitive Spelling Correction, § 5.5: Noun Compound Bracketing, and § 5.6: Verb Part-of-Speech Disambiguation.

§	IN-Train	IN-Dev	IN-Test	O1	O2
5.3	237K	13K	13K	13K	9.1K
5.4	100K	50K	50K	7.8K	56K
5.5	2.0K	72	95	244	429
5.6	23K	1.1K	1.1K	21K	6.3K

Table 5.2: Number of labeled examples in in-domain training, development and test sets, and out-of-domain test sets, for tasks in Sections 5.3-5.6.

use one *in-domain* and two *out-of-domain* test sets for each task. Statistical significance is assessed with McNemar's test, $p<0.01$.

We provide results for various unsupervised approaches and also the majority-class baseline for each task.

For our supervised approaches, we represent the examples as feature vectors, and learn a classifier on the training vectors. There are two feature classes: features that use N-grams (N-GM) and those that do not (LEX). N-GM features are real-valued features giving the log-count of a particular N-gram in the auxiliary web corpus. These are just like the features we used for the disambiguation problems in the previous two chapters. LEX features are binary features that indicate the presence or absence of a particular string at a given position in the input. The name LEX emphasizes that they identify specific lexical items. The instantiations of both types of features depend on the task and are described in the corresponding sections.

Each classifier is a linear Support Vector Machine (SVM), trained using LIBLINEAR [Fan *et al.*, 2008] on the standard domain. We use the one-vs-all strategy when there are more than two classes (in Section 5.4). We plot learning curves to measure the accuracy of the classifier when the number of labeled training examples varies. The size of the N-gram data and its counts remain constant. We always optimize the SVM's (L2) regularization parameter on the in-domain development set. We present results with L2-SVM, but achieve similar results with L1-SVM and logistic regression.

5.2.2 Tasks and Labeled Data

We study two generation tasks: prenominal adjective ordering (Section 5.3) and context-sensitive spelling correction (Section 5.4), followed by two analysis tasks: noun compound bracketing (Section 5.5) and verb part-of-speech disambiguation (Section 5.6). Tables 5.1 and 5.2 summarize the sources and sizes of data used in the experiments. For the out-of-domain Gutenberg and Medline data used in Sections 5.3 and 5.4, we generate examples

ourselves.[1] We chose Gutenberg and Medline in order to provide challenging, distinct domains from our training corpora. Our Gutenberg corpus consists of out-of-copyright books, automatically downloaded from the Project Gutenberg website.[2] The Medline data consists of a large collection of online biomedical abstracts. We describe how labeled adjective and spelling examples are created from these corpora in the corresponding sections.

5.2.3 Web-Scale Auxiliary Data

The most widely-used N-gram corpus is the Google 5-gram Corpus [Brants and Franz, 2006].

For our tasks, we also use **Google V2**: a new N-gram corpus (also with N-grams of length one-to-five) that we created from the same one-trillion-word snapshot of the web as the Google 5-gram Corpus, but with several enhancements. These include: 1) Reducing noise by removing duplicate sentences and sentences with a high proportion of non-alphanumeric characters (together filtering about 80% of the source data), 2) pre-converting all digits to the 0 character to reduce sparsity for numeric expressions, and 3) including the part-of-speech (POS) tag distribution for each N-gram. The source data was automatically tagged with TnT [Brants, 2000], using the Penn Treebank tag set. [Lin *et al.*, 2010] provide more details on the N-gram data and N-gram search tools. Other recent projects that have made use of this new data include [Ji and Lin, 2009; Bergsma *et al.*, 2010a; Pitler *et al.*, 2010].

The third enhancement is especially relevant here, as we can use the POS distribution to collect counts for N-grams of mixed words and tags. For example, we have developed an N-gram search engine that can count how often the adjective *unprecedented* precedes another adjective in our web corpus (113K times) and how often it follows one (11K times). Thus, even if we haven't seen a particular adjective pair directly, we can use the positional preferences of each adjective in order to order them.

Early web-based models used search engines to collect N-gram counts, and thus could not use capitalization, punctuation, and annotations such as part-of-speech [Kilgarriff and Grefenstette, 2003]. For example, recall that we had to develop fillers to serve as "surrogates for nouns" for the non-referential pronoun detection system presented in Chapter 3, as nouns were not labeled in the data directly. If we had a POS-tagged web corpus, we could have looked up noun counts directly. Using a POS-tagged web corpus goes a long way to addressing earlier criticisms of web-based NLP.

5.3 Prenominal Adjective Ordering

Prenominal adjective ordering strongly affects text readability. For example, while *the unprecedented statistical revolution* is fluent, *the statistical unprecedented revolution* is not. Many NLP systems need to handle adjective ordering robustly. In machine translation, if a noun has two adjective modifiers, they must be ordered correctly in the target language.

[1] `http://webdocs.cs.ualberta.ca/~bergsma/Robust/` provides our Gutenberg corpus, a link to Medline, and also the generated examples for both Gutenberg and Medline.

[2] `www.gutenberg.org`. All books just released in 2009 and thus unlikely to occur in the source data for our N-gram corpus (from 2006). Of course, with removal of sentence duplicates and also N-gram thresholding, the possible presence of a test sentence in the massive source data is unlikely to affect results. [Carlson *et al.*, 2008] reach a similar conclusion and provide some further justification.

Adjective ordering is also needed in Natural Language Generation systems that produce information from databases; for example, to convey information (in sentences) about medical patients [Shaw and Hatzivassiloglou, 1999].

We focus on the task of ordering a pair of adjectives independently of the noun they modify and achieve good performance in this setting. Following the set-up of [Malouf, 2000], we experiment on the 263K adjective pairs Malouf extracted from the British National Corpus (BNC). We use 90% of pairs for training, 5% for testing, and 5% for development. This forms our in-domain data.[3]

We create out-of-domain examples by tokenizing Medline and Gutenberg (Section 5.2.2), then POS-tagging them with CRFTagger [Phan, 2006]. We create examples from all sequences of two adjectives followed by a noun. Like [Malouf, 2000], we assume that edited text has adjectives ordered fluently. As was the case for preposition selection and spelling correction, adjective ordering also permits the extraction of *natural automatic examples* as explained in Chapter 2, Section 2.5.4. We extract 13K and 9.1K out-of-domain pairs from Gutenberg and Medline, respectively.[4]

The input to the system is a pair of adjectives, (a_1, a_2), ordered alphabetically. The task is to classify this order as correct (the positive class) or incorrect (the negative class). Since both classes are equally likely, the majority-class **baseline** is around 50% on each of the three test sets.

5.3.1 Supervised Adjective Ordering

LEX features

Our adjective ordering model with LEX features is a novel contribution of this work.

We begin with two features for each pair: an indicator feature for a_1, which gets a feature value of $+1$, and an indicator feature for a_2, which gets a feature value of -1. The parameters of the model are therefore weights on specific adjectives. The higher the weight on an adjective, the more it is preferred in the first position of a pair. If the alphabetic ordering is correct, the weight on a_1 should be higher than the weight on a_2, so that the classifier returns a positive score. If the reverse ordering is preferred, a_2 should receive a higher weight. Training the model in this setting is a matter of assigning weights to all the observed adjectives such that the training pairs are maximally ordered correctly. The feature weights thus implicitly produce a linear ordering of all observed adjectives. The examples can also be regarded as rank constraints in a discriminative ranker [Joachims, 2002]. Transitivity is achieved naturally in that if we correctly order pairs $a \prec b$ and $b \prec c$ in the training set, then $a \prec c$ by virtue of the weights on a and c.

While exploiting transitivity has been shown to improve adjective ordering, there are many conflicting pairs that make a strict linear ordering of adjectives impossible [Malouf, 2000]. We therefore provide an indicator feature for the pair $a_1 a_2$, so the classifier can memorize exceptions to the linear ordering, breaking strict order transitivity. Our classifier thus operates along the lines of rankers in the *preference-based setting* [Ailon and Mohri, 2008].

[3] BNC is not a domain *per se* (rather a balanced corpus), but has a style and vocabulary distinct from our OOD data.

[4] Like [Malouf, 2000], we convert our pairs to lower-case. Since the N-gram data includes case, we merge counts from the upper and lower case combinations.

System	IN	O1	O2
[Malouf, 2000]	91.5	65.6	71.6
web $c(a_1, a_2)$ vs. $c(a_2, a_1)$	87.1	83.7	86.0
SVM with N-GM features	90.0	**85.8**	**88.5**
SVM with LEX features	93.0	70.0	73.9
SVM with N-GM + LEX	**93.7**	83.6	85.4

Table 5.3: Adjective ordering accuracy (%). SVM and [Malouf, 2000] trained on BNC, tested on BNC (IN), Gutenberg (O1), and Medline (O2).

Finally, we also have features for all suffixes of length 1-to-4 letters, as these encode useful information about adjective class [Malouf, 2000]. Like the adjective features, the suffix features receive a value of $+1$ for adjectives in the first position and -1 for those in the second.

N-GM features

[Lapata and Keller, 2005] propose a web-based approach to adjective ordering: take the most-frequent order of the words on the web, $c(a_1, a_2)$ vs. $c(a_2, a_1)$. We adopt this as our unsupervised approach. We merge the counts for the adjectives occurring contiguously and separated by a comma.

These are the most useful N-GM features; we include them but also other, tag-based counts from Google V2. Raw counts include cases where one of the adjectives is not used as a modifier: "the *special present* was" vs. "the *present special* issue." We include log-counts for the following, more-targeted patterns:[5] $c(a_1\ a_2$ N.*), $c(a_2\ a_1$ N.*), c(DT $a_1\ a_2$ N.*), c(DT $a_2\ a_1$ N.*). We also include features for the log-counts of each adjective preceded or followed by a word matching an adjective-tag: $c(a_1$ J.*), c(J.* a_1), $c(a_2$ J.*), c(J.* a_2). These assess the positional preferences of each adjective. Finally, we include the log-frequency of each adjective. The more frequent adjective occurs first in 57% of pairs.

As in all tasks, the counts are features in a classifier, so the importance of the different patterns is weighted discriminatively during training.

5.3.2 Adjective Ordering Results

In-domain, with both feature classes, we set a strong new standard on this data: 93.7% accuracy for the N-GM+LEX system (Table 5.3). We trained and tested [Malouf, 2000]'s program on our data; our LEX classifier, which also uses no auxiliary corpus, makes 18% fewer errors than Malouf's system. Our web-based N-GM model is also superior to the direct evidence web-based approach of [Lapata and Keller, 2005], scoring 90.0% vs. 87.1% accuracy. These results show the benefit of both our new lexicalized and our new web-based features.

Figure 5.1 gives the in-domain learning curve. With fewer training examples, the systems with N-GM features strongly outperform the LEX-only system. Note that with tens of thousands of test examples, all differences are highly significant.

Out-of-domain, LEX's accuracy drops a shocking 23% on Gutenberg and 19% on Medline (Table 5.3). [Malouf, 2000]'s system fares even worse. The overlap between training

[5] In this notation, capital letters (and reg-exps) are matched against tags while a_1 and a_2 match words.

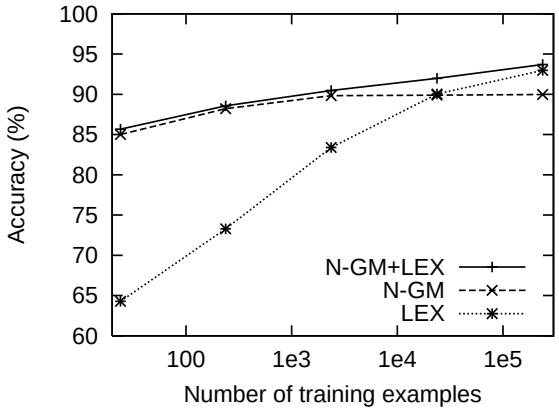


Figure 5.1: In-domain learning curve of adjective ordering classifiers on BNC.

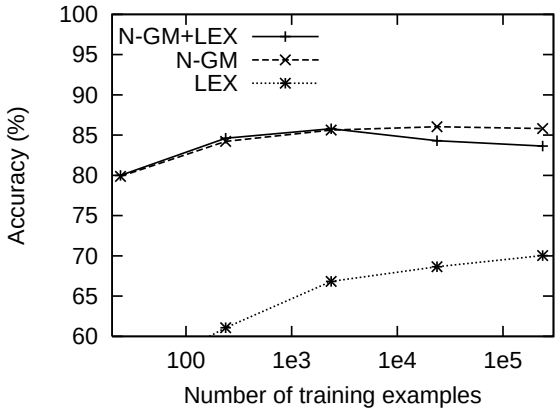


Figure 5.2: Out-of-domain learning curve of adjective ordering classifiers on Gutenberg.

and test pairs helps explain. While 59% of the BNC test pairs were seen in the training corpus, only 25% of Gutenberg and 18% of Medline pairs were seen in training.

While other ordering models have also achieved "very poor results" out-of-domain [Mitchell, 2009], we expected our expanded set of LEX features to provide good generalization on new data. Instead, LEX is very unreliable on new domains.

N-GM features do not rely on specific pairs in training data, and thus remain fairly robust cross-domain. Across the three test sets, 84-89% of examples had the correct ordering appear at least once on the web. On new domains, the learned N-GM system maintains an advantage over the unsupervised $c(a_1, a_2)$ vs. $c(a_2, a_1)$, but the difference is reduced. Note that training with 10-fold cross validation, the N-GM system can achieve up to 87.5% on Gutenberg (90.0% for N-GM + LEX).

The learning curves showing performance on Gutenberg and Medline (but still training on BNC) is particularly instructive (Figures 5.2 and 5.3). The LEX system performs much worse than the web-based models across all training sizes. For our top in-domain system, N-GM + LEX, as you add more labeled examples, performance begins *decreasing* out-of-domain. The system disregards the robust N-gram counts as it is more and more confident in the LEX features, and it suffers the consequences.

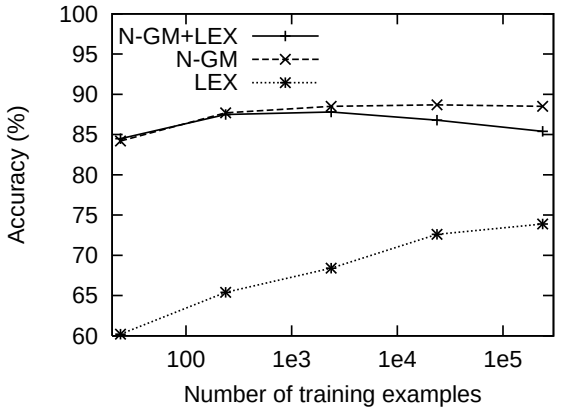


Figure 5.3: Out-of-domain learning curve of adjective ordering classifiers on Medline.

5.4 Context-Sensitive Spelling Correction

We now turn to the generation problem of context-sensitive spelling correction. For those who have read the previous two chapters, you're obviously familiar with the task: For every occurrence of a word in a pre-defined set of confusable words (like *peace* and *piece*), the system must select the most likely word from the set, flagging possible usage errors when the predicted word disagrees with the original.

Our in-domain examples are again from the New York Times (NYT) portion of Gigaword, as described in Chapter 3. Recall that these comprise the 5 confusion sets where accuracy was below 90% in [Golding and Roth, 1999]. There are 100K training, 10K development, and 10K test examples for each confusion set. Our results are averages across confusion sets.

Out-of-domain examples are again drawn from Gutenberg and Medline. We extract all instances of words that are in one of our confusion sets, along with surrounding context. By assuming the extracted instances represent correct usage, we label 7.8K and 56K out-of-domain test examples for Gutenberg and Medline, respectively.

We test three unsupervised systems: 1) TRIGRAM (Chapter 3): use one token of context on the left and one on the right, and output the candidate from the confusion set that occurs most frequently in this pattern [Lapata and Keller, 2005]. 2) SUMLM (Chapter 3), where we measure the frequency of the candidates in all the 3-to-5-gram patterns that span the confusable word. For each candidate, we sum the log-counts of all patterns filled with the candidate, and output the candidate with the highest total. 3) The **baseline** predicts the most frequent member of each confusion set, based on frequencies in the NYT training data.

5.4.1 Supervised Spelling Correction

Our LEX features are typical disambiguation features that flag specific aspects of the context. We have features for the words at all positions in a 9-word window (called collocation features by [Golding and Roth, 1999]), plus indicators for a particular word preceding or following the confusable word. We also include indicators for all N-grams, and their position, in a 9-word window.

For N-GM count features, we follow Chapter 3. We include the log-counts of all

System	IN	O1	O2
Baseline	66.9	44.6	60.6
TRIGRAM	88.4	78.0	87.4
SUMLM	94.8	87.7	94.2
SVM with N-GM features	95.7	**92.1**	93.9
SVM with LEX features	95.2	85.8	91.0
SVM with N-GM + LEX	**96.5**	91.9	**94.8**

Table 5.4: Spelling correction accuracy (%). SVM trained on NYT, tested on NYT (IN) and out-of-domain Gutenberg (O1) and Medline (O2).

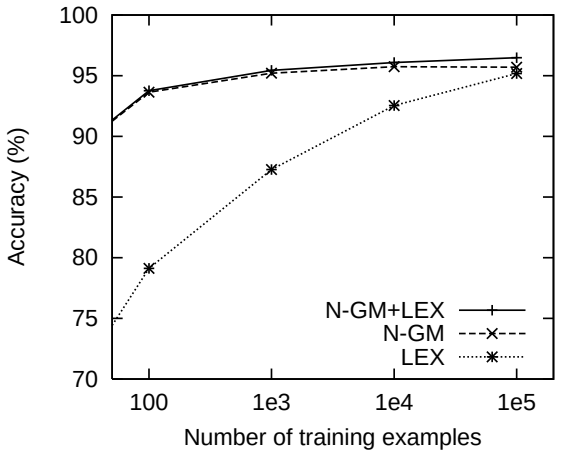


Figure 5.4: In-domain learning curve of spelling correction classifiers on NYT.

N-grams that span the confusable word, with each word in the confusion set filling the N-gram pattern. These features do not use part-of-speech. Following our previous work, we get N-gram counts using the original Google N-gram Corpus.

While neither our LEX nor N-GM features are novel on their own, they have, perhaps surprisingly, not yet been evaluated in a single model.

5.4.2 Spelling Correction Results

The N-GM features outperform the LEX features, 95.7% vs. 95.2% (Table 5.4). Together, they achieve a very strong 96.5% in-domain accuracy. This is 2% higher than the best unsupervised approach, SUMLM. Web-based models again perform well across a range of training data sizes (Figure 5.4).

The error rate of LEX nearly triples on Gutenberg and almost doubles on Medline (Table 5.4). Removing N-GM features from the N-GM + LEX system, errors increase around 75% on both Gutenberg and Medline. The LEX features provide no help to the combined system on Gutenberg, while they do help significantly on Medline. Note the learning curves for N-GM+LEX on Gutenberg and Medline (Figures 5.5, and 5.6) do not display the decrease that we observed in adjective ordering (Figure 5.2).

Both the baseline and LEX perform poorly on Gutenberg. The baseline predicts the majority class from NYT, but it's not always the majority class in Gutenberg. For example, while in NYT *site* occurs 87% of the time for the *(cite, sight, site)* confusion set, *sight* occurs

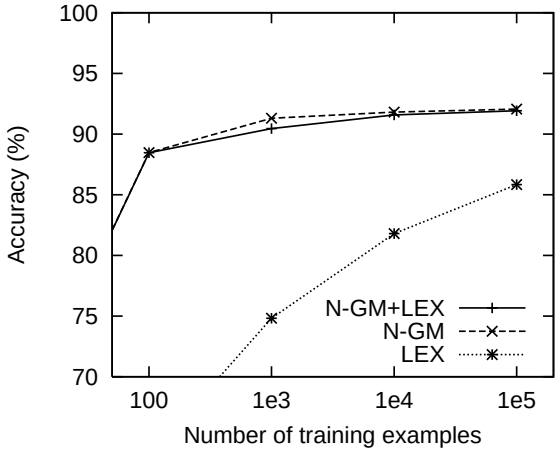


Figure 5.5: Out-of-domain learning curve of spelling correction classifiers on Gutenberg.

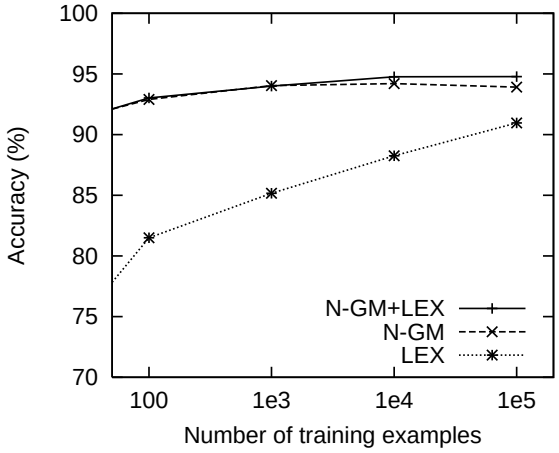


Figure 5.6: Out-of-domain learning curve of spelling correction classifiers on Medline.

90% of the time in Gutenberg. The LEX classifier exploits this bias as it is regularized toward a more economical model, but the bias does not transfer to the new domain.

5.5 Noun Compound Bracketing

About 70% of web queries are noun phrases [Barr *et al.*, 2008] and methods that can reliably parse these phrases are of great interest in NLP. For example, a web query for *zebra hair straightener* should be bracketed as *(zebra (hair straightener))*, a stylish hair straightener with zebra print, rather than *((zebra hair) straightener)*, a useless product since the fur of zebras is already quite straight.

The noun compound (NC) bracketing task is usually cast as a decision whether a 3-word NC has a left or right bracketing. Most approaches are unsupervised, using a large corpus to compare the statistical association between word pairs in the NC. The adjacency model [Marcus, 1980] proposes a left bracketing if the association between words one and two is higher than between two and three. The dependency model [Lauer, 1995a] compares one-two vs. *one-three*. We include dependency model results using PMI as the association measure; results were lower with the adjacency model.

As in-domain data, we use [Vadas and Curran, 2007a]'s Wall-Street Journal (WSJ) data, an extension of the Treebank (which originally left NPs flat). We extract all sequences of three consecutive common nouns, generating 1983 examples from sections 0-22 of the Treebank as training, 72 from section 24 for development and 95 from section 23 as a test set. As out-of-domain data, we use 244 NCs from Grolier Encyclopedia [Lauer, 1995a] and 429 NCs from Medline [Nakov, 2007].

The majority class **baseline** is left-bracketing.

5.5.1 Supervised Noun Bracketing

Our LEX features indicate the specific noun at each position in the compound, plus the three pairs of nouns and the full noun triple. We also add features for the capitalization pattern of the sequence.

N-GM features give the log-count of all subsets of nouns in the compound: (N1), (N2), (N3), (N1 N2), (N1 N3), (N2 N3), and (N1 N2 N3). Counts are from Google V2. Following [Nakov and Hearst, 2005a], we also include counts of noun pairs collapsed into a single token; if a pair occurs often on the web as a single unit, it strongly indicates the pair is a constituent.

[Vadas and Curran, 2007a] use simpler features, e.g. they do not use collapsed pair counts. They achieve 89.9% in-domain on WSJ and 80.7% on Grolier. [Vadas and Curran, 2007b] use comparable features to ours, but do not test out-of-domain.

5.5.2 Noun Compound Bracketing Results

N-GM systems perform much better on this task (Table 5.5). N-GM+LEX is statistically significantly better than LEX on all sets. In-domain, errors more than double without N-GM features. LEX performs poorly here because there are far fewer training examples. The learning curve (Figure 5.7) looks much like earlier in-domain curves (Figures 5.1 and 5.4), but truncated before LEX becomes competitive. The absence of a sufficient amount of labeled data explains why NC-bracketing is generally regarded as a task where corpus counts are crucial.

System	IN	O1	O2
Baseline	70.5	66.8	84.1
Dependency model	74.7	**82.8**	84.4
SVM with N-GM features	89.5	81.6	86.2
SVM with LEX features	81.1	70.9	79.0
SVM with N-GM + LEX	**91.6**	81.6	**87.4**

Table 5.5: NC-bracketing accuracy (%). SVM trained on WSJ, tested on WSJ (IN) and out-of-domain Grolier (O1) and Medline (O2).

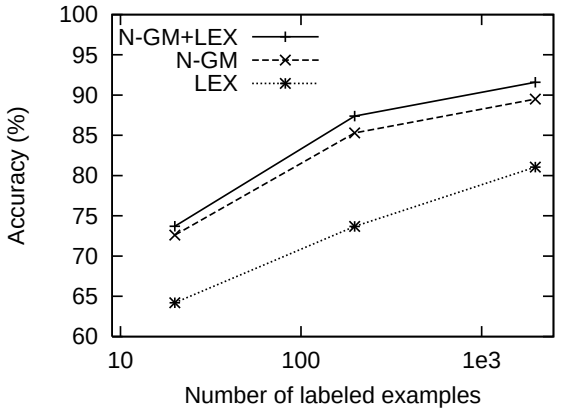


Figure 5.7: In-domain NC-bracketer learning curve

All web-based models (including the dependency model) exceed 81.5% on Grolier, which is the level of human agreement [Lauer, 1995b]. N-GM + LEX is highest on Medline, and close to the 88% human agreement [Nakov and Hearst, 2005a]. Out-of-domain, the LEX approach performs very poorly, close to or below the baseline accuracy. With little training data and cross-domain usage, N-gram features are essential.

5.6 Verb Part-of-Speech Disambiguation

Our final task is POS-tagging. We focus on one frequent and difficult tagging decision: the distinction between a past-tense verb (VBD) and a past participle (VBN). For example, in *the troops stationed in Iraq*, the verb *stationed* is a VBN; *troops* is the head of the (noun) phrase. On the other hand, for *the troops vacationed in Iraq*, the verb *vacationed* is a VBD and also the head. Some verbs make the distinction explicit (*eat* has VBD *ate*, VBN *eaten*), but most require context for resolution. This is exactly the task we presented in Section 1.3 as an example of where unlabeled data can be useful. Recall that the verb in *Bears won* is a VBD while the verb in *trophy won* is a VBN.

Conflating VBN/VBD is damaging because it affects downstream parsers and semantic role labelers. The task is difficult because nearby POS tags can be identical in both cases. When the verb follows a noun, tag assignment can hinge on world-knowledge, i.e., the global lexical relation between the noun and verb (E.g., *troops* tends to be the object of *stationed* but the subject of *vacationed*).[6] Web-scale N-gram data might help improve the

[6]HMM-style taggers, like the fast TnT tagger used on our web corpus, do not use bilexical features, and so

VBN/VBD distinction by providing relational evidence, even if the verb, noun, or verb-noun pair were not observed in training data. This is what we showed in Section 1.3.

We extract nouns followed by a VBN/VBD in the WSJ portion of the Treebank [Marcus *et al.*, 1993], getting 23K training, 1091 development and 1130 test examples from sections 2-22, 24, and 23, respectively. For out-of-domain data, we get 21K examples from the Brown portion of the Treebank and 6296 examples from tagged Medline abstracts in the PennBioIE corpus [Kulick *et al.*, 2004].

The majority class **baseline** is to choose VBD.

5.6.1 Supervised Verb Disambiguation

There are two orthogonal sources of information for predicting VBN/VBD: 1) the noun-verb pair, and 2) the context around the pair. Both N-GM and LEX features encode both these sources.

LEX features

For 1), we use indicators for the noun and verb, the noun-verb pair, whether the verb is on an in-house list of *said*-verb (like *warned*, *announced*, etc.), whether the noun is capitalized and whether it's upper-case. Note that in training data, 97.3% of capitalized nouns are followed by a VBD and 98.5% of *said*-verbs are VBDs. For 2), we provide indicator features for the words before the noun and after the verb.

N-GM features

For 1), we characterize a noun-verb relation via features for the pair's distribution in Google V2. Characterizing a word by its distribution has a long history in NLP; we apply similar techniques to *relations*, like [Turney, 2006], but with a larger corpus and richer annotations. We extract the 20 most-frequent N-grams that contain both the noun and the verb in the pair. For each of these, we convert the tokens to POS-tags, except for tokens that are among the 100 most frequent unigrams in our corpus, which we include in word form. We mask the noun of interest as *N* and the verb of interest as *V*. This converted N-gram is the feature label. The value is the pattern's log-count. A high count for patterns like *(N that V)*, *(N have V)* suggests the relation is a VBD, while patterns *(N that were V)*, *(N V by)*, *(V some N)* indicate a VBN. (Again, refer to Section 1.3 for some more example patterns). As always, the classifier learns the association between patterns and classes.

For 2), we use counts for the verb's context co-occurring with a VBD or VBN tag in our web corpus (again exploiting the fact our web corpus contains tags). E.g., we see whether VBD cases like *troops ate* or VBN cases like *troops eaten* are more frequent. Although our corpus contains many VBN/VBD errors, we hope the errors are random enough for aggregate counts to be useful. The context is an N-gram spanning the VBN/VBD. We have log-count features for all five such N-grams in the (previous-word, noun, verb, next-word) quadruple. The log-count is indexed by the position and length of the N-gram. We include separate count features for contexts matching the specific noun and for when the noun token can match any word tagged as a noun.

perform especially poorly on these cases. One motivation for our work was to develop a fast post-processor to fix VBN/VBD errors.

System	IN	O1	O2
Baseline	89.2	85.2	79.6
ContextSum	92.5	91.1	90.4
SVM with N-GM features	96.1	93.4	93.8
SVM with LEX features	95.8	93.4	93.0
SVM with N-GM + LEX	**96.4**	**93.5**	**94.0**

Table 5.6: Verb-POS-disambiguation accuracy (%) trained on WSJ, tested on WSJ (IN) and out-of-domain Brown (O1) and Medline (O2).

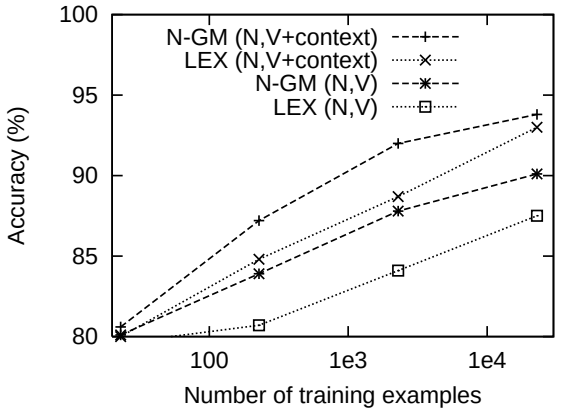


Figure 5.8: Out-of-domain learning curve of verb disambiguation classifiers on Medline.

ContextSum: We also use these context counts in an unsupervised system, ContextSum. Analogously to the SUMLM system from Chapter 3, we separately sum the log-counts for all contexts filled with VBD and then VBN, outputting the tag with the higher total.

5.6.2 Verb POS Disambiguation Results

As in all tasks, N-GM+LEX has the best in-domain accuracy (96.4%, Table 5.6). Out-of-domain, when N-grams are excluded, errors only increase around 14% on Medline and 2% on Brown (the differences are not statistically significant). Why? Figure 5.8, the learning curve for performance on Medline, suggests some reasons. We omit N-GM+LEX from Figure 5.8 as it closely follows N-GM.

Recall that we grouped the features into two views: 1) noun-verb (N,V) and 2) context. If we use just (N,V) features, we do see a large drop out-of-domain: LEX (N,V) lags N-GM (N,V) even using all the training examples. The same is true using only context features (not shown). Using both views, the results are closer: 93.8% for N-GM and 93.0% for LEX. With two views of an example, LEX is more likely to have domain-neutral features to draw on. The effects of data sparsity are reduced.

Also, the Treebank provides an atypical number of labeled examples for analysis tasks. In a more typical situation with less labeled examples, N-GM strongly dominates LEX, even when two views are used. E.g., with 2285 training examples, N-GM+LEX is statistically significantly better than LEX on both out-of-domain sets.

All systems, however, perform log-linearly with training size. This differs especially from our generation tasks where the N-GM performance had levelled off much earlier. In

other tasks we only had a handful of N-GM features; here there are 21K features for the distributional patterns of N,V pairs. With more features, we need more labeled data to make the N-GM features work properly. A similar issue with such high-dimensional global features is explored in [Huang and Yates, 2009]. Reducing this feature space by pruning or performing transformations may improve accuracy in and out-of-domain.

5.7 Discussion and Future Work

Of all classifiers, LEX performs worst on all cross-domain tasks. Clearly, many of the regularities that a typical classifier exploits in one domain do not transfer to new genres. N-GM features, however, do not depend directly on training examples, and thus work better cross-domain. Of course, using web-scale N-grams is not the only way to create robust classifiers. Counts from any large auxiliary corpus may also help, but web counts should help more [Lapata and Keller, 2005]. Section 5.6.2 suggests that another way to mitigate domain-dependence is to have multiple feature views.

[Banko and Brill, 2001] argue "a logical next step for the research community would be to direct efforts towards increasing the size of annotated training collections." Assuming we really do want systems that operate beyond the specific domains on which they are trained, the community also needs to identify which systems behave as in Figure 5.2, where the accuracy of the best in-domain system actually decreases with more training examples. Our results suggest better features, such as web pattern counts, may help more than expanding training data. Also, systems using web-scale unlabeled data will improve automatically as the web expands, without annotation effort.

In some sense, using web counts as features is a form of domain adaptation: adapting a web model to the training domain. How do we ensure these features are adapted well and not used in domain-specific ways (especially with many features to adapt, as in Section 5.6)? One option may be to regularize the classifier specifically for out-of-domain accuracy. We found that adjusting the SVM misclassification penalty (for more regularization) can help or hurt out-of-domain. Other regularizations are possible. In fact, one good option might be to extend the results of Chapter 4 to cross-domain usage. In this chapter, each task had a domain-neutral unsupervised approach. We could encode these systems as linear classifiers with corresponding weights. Rather than a typical SVM that minimizes the weight-norm $||w||$ (plus the slacks), we could regularize toward domain-neutral weights, and otherwise minimize weight variance so that it prefers this system in the absence of other information. This regularization could be optimized on creative splits of the training data, in an effort to simulate the lexical overlap we can expect with a particular test domain.

5.8 Conclusion

We presented results on tasks spanning a range of NLP research: generation, disambiguation, parsing and tagging. Using web-scale N-gram data improves accuracy on each task. When less training data is used, or when the system is used on a different domain, N-gram features greatly improve performance. Since most supervised NLP systems do not use web-scale counts, further cross-domain evaluation may reveal some very brittle systems. Continued effort in new domains should be a priority for the community going forward.

This concludes the part of the dissertation that explored using web-scale N-gram data as features within a supervised classifier. I hope you enjoyed it.

Chapter 6

Discriminative Learning of Selectional Preference from Unlabeled Text

"Saying 'I'm sorry' is the same as saying 'I apologize.' Except at a funeral."
- Demetri Martin

6.1 Introduction

Selectional preferences (SPs) tell us which arguments are plausible for a particular predicate. For example, a *letter*, a *report* or an *e-mail* are all plausible direct-object arguments for the verb predicate *write*. On the other hand, a *zebra*, *intransigence* or *nihilism* are unlikely arguments for this predicate. People tend to *exude confidence* but they don't tend to *exude pencils*. Table 6.2 (Section 6.4.4) gives further examples. SPs can help resolve syntactic, word sense, and reference ambiguity [Clark and Weir, 2002], and so gathering them has received a lot of attention in the NLP community.

One way to determine SPs is from co-occurrences of predicates and arguments in text. Unfortunately, no matter how much text we use (even if, as in the previous chapters, we're using Google N-grams, which come from all the text on the web), many acceptable pairs will be missing. Bikel [2004] found that only 1.49% of the bilexical dependencies considered by Collins' parser during decoding were observed during training. In our parsed corpus (Section 6.4.1), for example, we find *eat* with *nachos*, *burritos*, and *tacos*, but not with the equally tasty *quesadillas*, *chimichangas*, or *tostadas*. Rather than solely relying on co-occurrence counts, we would like to use them to generalize to unseen pairs.

In particular, we would like to exploit a number of arbitrary and potentially overlapping properties of predicates and arguments when we assign SPs. We propose to do this by representing these properties as features in a linear classifier, and training the weights using discriminative learning. Positive examples are taken from observed predicate-argument pairs, while pseudo-negatives are constructed from unobserved combinations. We train a support vector machine (SVM) classifier to distinguish the positives from the negatives. We refer to our model's scores as Discriminative Selectional Preference (DSP). By creating training vectors automatically, DSP enjoys all the advantages of supervised learning, but

[0] A version of this chapter has been published as [Bergsma *et al.*, 2008a]

without the need for manual annotation of examples. Our proposed work here is therefore an example of a semi-supervised system that uses "Learning with Heuristically-Labeled Examples," as described in Chapter 2, Section 2.5.4.

We evaluate DSP on the task of assigning verb-object selectional preference. We encode a noun's textual distribution as feature information. The learned feature weights are linguistically interesting, yielding high-quality similar-word lists as latent information. With these features, DSP is also an example of a semi-supervised system that creates features from unlabeled data (Section 2.5.5). It thus encapsulates the two main thrusts of this dissertation.

Despite its representational power, DSP scales to real-world data sizes: examples are partitioned by predicate, and a separate SVM is trained for each partition. This allows us to efficiently learn with over 57 thousand features and 6.5 million examples. DSP outperforms recently proposed alternatives in a range of experiments, and better correlates with human plausibility judgments. It also shows strong gains over a Mutual Information-based co-occurrence model on two tasks: identifying objects of verbs in an unseen corpus and finding pronominal antecedents in coreference data.

6.2 Related Work

Most approaches to SPs generalize from observed predicate-argument pairs to semantically similar ones by modeling the semantic class of the argument, following Resnik [1996]. For example, we might have a class *Mexican Food* and learn that the entire class is suitable for eating. Usually, the classes are from WordNet [Miller *et al.*, 1990], although they can also be inferred from clustering [Rooth *et al.*, 1999]. Brockmann and Lapata [2003] compare a number of WordNet-based approaches, including Resnik [1996], Li and Abe [1998], and Clark and Weir [2002], and found that more sophisticated class-based approaches do not always outperform frequency-based models.

Another line of research generalizes using similar words. Suppose we are calculating the probability of a particular noun, n, occurring as the object argument of a given verbal predicate, v. Let $\Pr(n|v)$ be the empirical maximum-likelihood estimate from observed text. Dagan et al. [1999] define the similarity-weighted probability, $\Pr_{\text{SIM}}$, to be:

$$\Pr_{\text{SIM}}(n|v) = \sum_{v' \in \text{SIMS}(v)} \textit{Sim}(v', v) \Pr(n|v') \tag{6.1}$$

where $\textit{Sim}(v', v)$ returns a real-valued similarity between two verbs v' and v (normalized over all pair similarities in the sum). In contrast, Erk [2007] generalizes by substituting similar *arguments*, while Wang et al. [2005] use the cross-product of similar pairs. One key issue is how to define the set of similar words, $\text{SIMS}(w)$. Erk [2007] compared a number of techniques for creating similar-word sets and found that both the Jaccard coefficient and Lin [1998a]'s information-theoretic metric work best. Similarity-smoothed models are simple to compute, potentially adaptable to new domains, and require no manually-compiled resources such as WordNet.

Selectional preferences have also been a recent focus of researchers investigating the learning of paraphrases and inference rules [Pantel *et al.*, 2007; Roberto *et al.*, 2007]. Inferences such as "[*X* wins *Y*] $\Rightarrow$ [*X* plays *Y*]" are only valid for certain arguments *X* and *Y*. We follow Pantel et al. [2007] in using automatically-extracted semantic classes to help characterize plausible arguments.

As described in Chapter 2, discriminative techniques are widely used in NLP. They have been applied to the related tasks of word prediction and language modeling. Even-Zohar and Roth [2000] use a classifier to predict the most likely word to fill a position in a sentence (in their experiments: a verb) from a set of candidates (sets of verbs), by inspecting the context of the target token (e.g., the presence or absence of a particular nearby word in the sentence). This approach can therefore learn which specific arguments occur with a particular predicate. In comparison, our features are second-order: we learn what *kinds* of arguments occur with a predicate by encoding features of the arguments. The work of van den Bosch [2006] extends the work of Even-Zohar and Roth [2000] to all-word prediction. Recent distributed and latent-variable models also represent words with feature vectors [Bengio *et al.*, 2003; Blitzer *et al.*, 2005]. Many of these approaches learn both the feature weights and the feature representation. Vectors must be kept low-dimensional for tractability, while learning and inference on larger scales is impractical. By partitioning our examples by predicate, we can efficiently use high-dimensional, sparse vectors.

6.3 Methodology

6.3.1 Creating Examples

To learn a discriminative model of selectional preference, we create positive and negative training examples automatically from raw text. To create the positives, we automatically parse a large corpus, and then extract the predicate-argument pairs that have a statistical association in this data. We measure this association using point-wise Mutual Information (MI) [Church and Hanks, 1990]. The MI between a verb predicate, v, and its object argument, n, is:

$$\text{MI}(v,n) = \log \frac{\Pr(v,n)}{\Pr(v)\Pr(n)} = \log \frac{\Pr(n|v)}{\Pr(n)} \tag{6.2}$$

If MI>0, the probability v and n occur together is greater than if they were independently distributed.

We create sets of positive and negative examples separately for each predicate, v. First, we extract all pairs where MI$(v,n)>\tau$ as positives. For each positive, we create pseudo-negative examples, (v,n'), by pairing v with a new argument, n', that either has MI below the threshold or did not occur with v in the corpus. We require each negative n' to have a similar frequency to its corresponding n. This prevents our learning algorithm from focusing on any accidental frequency-based bias. We mix in K negatives for each positive, sampling without replacement to create all the negatives for a particular predicate. For each v, $\frac{1}{K+1}$ of its examples will be positive. The threshold τ represents a trade-off between capturing a large number of positive pairs and ensuring these pairs have good association. Similarly, K is a trade-off between the number of examples and the computational efficiency. Ultimately, these parameters should be optimized for task performance.

Of course, some negatives will actually be plausible arguments that were unobserved due to sparseness. Fortunately, discriminative methods like soft-margin SVMs can learn in the face of label error by allowing slack, subject to a tunable regularization penalty (Chapter 2, Section 2.3.4).

If MI is a sparse and imperfect model of SP, what can DSP gain by training on MI's scores? We can regard DSP as learning a view of SP that is orthogonal to MI, in a co-training sense [Blum and Mitchell, 1998]. MI labels the data based solely on co-occurrence;

DSP uses these labels to identify other regularities – ones that extend beyond co-occurring words. For example, many instances of n where MI(*eat*, n)>τ also have MI(*buy*, n)>τ and MI(*cook*, n)>τ. Also, compared to other nouns, a disproportionate number of *eat*-nouns are lower-case, single-token words, and they rarely contain digits, hyphens, or begin with a human first name like *Bob*. DSP encodes these interdependent properties as features in a linear classifier. This classifier can score any noun as a plausible argument of *eat* if indicative features are present; MI can only assign high plausibility to observed (*eat*,n) pairs. Similarity-smoothed models can make use of the regularities across similar verbs, but not the finer-grained string- and token-based features.

Our training examples are similar to the data created for *pseudodisambiguation*, the usual evaluation task for SP models [Erk, 2007; Keller and Lapata, 2003; Rooth *et al.*, 1999]. This data consists of triples (v, n, n') where v, n is a predicate-argument pair observed in the corpus and v, n' has not been observed. The models score correctly if they rank observed (thus plausible) arguments above corresponding unobserved (thus implausible) ones. We refer to this as *Pairwise Disambiguation*. Unlike this task, we classify each predicate-argument pair independently as plausible/implausible. We also use MI rather than frequency to define the positive pairs, ensuring that the positive pairs truly have a statistical association, and are not simply the result of parser error or noise.[1]

6.3.2 Partitioning for Efficient Training

After creating our positive and negative training pairs, we must select a feature representation for our examples. Let $\mathbf{\Phi}$ be a mapping from a predicate-argument pair (v, n) to a feature vector, $\mathbf{\Phi} : (v, n) \rightarrow \langle \phi_1 ... \phi_k \rangle$. Predictions are made using a linear classifier, $h(v, n) = \bar{w} \cdot \mathbf{\Phi}(v, n)$, where $\bar{w}$ is our learned weight vector.

We can make training significantly more efficient by using a special form of attribute-value features. Let every feature ϕ_i be of the form $\phi_i(v, n) = \langle v = \hat{v} \wedge f(n) \rangle$. That is, every feature is an intersection of the occurrence of a particular predicate, $\hat{v}$, and some feature of the argument $f(n)$. For example, a feature for a verb-object pair might be, "the verb is *eat* and the object is lower-case." In this representation, features for one predicate will be completely independent from those for every other predicate. Thus rather than a single training procedure, we can actually partition the examples by predicate, and train a classifier for each predicate independently. The prediction becomes $h^v(n) = \bar{w}^v \cdot \mathbf{\Phi}^v(n)$, where $\bar{w}^v$ are the learned weights corresponding to predicate v and all features $\mathbf{\Phi}^v(n)$=$\mathbf{f}(n)$ depend on the argument only.[2]

Some predicate partitions may have insufficient examples for training. Also, a predicate may occur in test data that was unseen during training. To handle these cases, we cluster low-frequency predicates. For assigning SP to verb-object pairs, we cluster all verbs that have less than 250 positive examples, using clusters generated by the CBC algorithm [Pantel and Lin, 2002]. For example, the low-frequency verbs *incarcerate*, *parole*, and

[1] For a fixed verb, MI is proportional to Keller and Lapata [2003]'s conditional probability scores for pseudodisambiguation of (v, n, n') triples: Pr($v|n$) = Pr(v, n)/Pr(n), which was shown to be a better measure of association than co-occurrence frequency $f(v, n)$. Normalizing by Pr(v) (yielding MI) allows us to use a constant threshold across all verbs. MI was also recently used for inference-rule SPs by Pantel et al. [2007].

[2] $h^v(n)$ should not be confused with the multi-class classifiers presented in previous chapters. There, the r in $\bar{W}_r \cdot \bar{x}$ indexed a class and $\bar{W}_r \cdot \bar{x}$ gave the score for each class. All the class scores needed to be computed for each example at test time. Here, for a given v, there is only one linear combination to compute: $\bar{w}^v \cdot \mathbf{f}(n)$. A single binary plausible/implausible decision is made on the basis of this verb-specific decision function.

court-martial are all mapped to the same partition, while frequent verbs like *arrest* and *execute* each have their own partition. About 5.5% of examples are clustered, corresponding to 30% of the 7367 total verbs. 40% of verbs (but only 0.6% of examples) were not in any CBC cluster; these were mapped to a single backoff partition.

The parameters for each partition, $\bar{w}^v$, can be trained with any supervised learning technique. We use SVM (Section 6.4.1) because it is effective in similar high-dimensional, sparse-vector settings (Chapter 2, Section 2.3.4), and has an efficient implementation [Joachims, 1999a]. In an SVM, the sign of $h^v(n)$ gives the classification. We can also use the scalar $h^v(n)$ as our DSP score (i.e. the positive distance of a point from the separating SVM hyperplane).

6.3.3 Features

This section details our argument features, $\mathbf{f}(n)$, for assigning verb-object selectional preference. For a verb predicate (or partition) v and object argument n, the form of our classifier is $h^v(n) = \bar{w}^v \cdot \mathbf{f}(n) = \sum_i w_i^v f_i(n)$.

Verb co-occurrence

We provide features for the empirical probability of the noun occurring as the object argument of other verbs, $\Pr(n|v')$. If we were to only use these features (indexing the feature weights by each verb v'), the form of our classifier would be:

$$h^v(n) = \sum_{v'} w_{v'}^v \Pr(n|v') \tag{6.3}$$

Note the similarity between Equation (6.3) and Equation (6.1). Now the feature weights, $w_{v'}^v$, take the role of the similarity function, $\mathit{Sim}(v', v)$. Unlike Equation (6.1), however, these weights are not set by an external similarity algorithm, but are optimized to discriminate the positive and negative training examples. We need not restrict ourselves to a short list of similar verbs; we include $\Pr_{obj}(n|v')$ features for every verb that occurs more than 10 times in our corpus. $w_{v'}^v$ may be positive or negative, depending on the relation between v' and v. We also include features for the probability of the noun occurring as the *subject* of other verbs, $\Pr_{subj}(n|v')$. For example, nouns that can be the object of *eat* will also occur as the subject of *taste* and *contain*. Other contexts, such as adjectival and nominal predicates, could also aid the prediction, but have not been investigated.

The advantage of tuning similarity to the application of interest has been shown previously by Weeds and Weir [2005]. They optimize a few meta-parameters separately for the tasks of thesaurus generation and pseudodisambiguation. Our approach discriminatively sets millions of individual similarity values. Like Weeds and Weir [2005], our similarity values are asymmetric.

String-based

We include several simple character-based features of the noun string: the number of tokens, the capitalization, and whether the string contains digits, hyphens, an apostrophe, or other punctuation. We also include a feature for the first and last token, and fire indicator features if any token in the noun occurs on in-house lists of given names, family names, cities, provinces, countries, corporations, languages, etc. We also fire a feature if a token is a corporate designation (like *inc.* or *ltd.*) or a human one (like *Mr.* or *Sheik*).

Semantic classes

Motivated by previous SP models that make use of semantic classes, we generate word clusters using CBC [Pantel and Lin, 2002] on a 10 GB corpus, giving 3620 clusters. If a noun belongs in a cluster, a corresponding feature fires. If a noun is in none of the clusters, a *no-class* feature fires.

As an example, CBC cluster 1891 contains:

> sidewalk, driveway, roadway, footpath, bridge, highway, road, runway, street, alley, path, Interstate, ...

In our training data, we have examples like *widen highway*, *widen road* and *widen motorway.* If we see that we can widen a highway, we learn that we can also widen a sidewalk, bridge, runway, etc.

We also made use of the person-name/instance pairs automatically extracted by Fleischman et al. [2003].[3] This data provides counts for pairs such as "Edwin Moses, *hurdler*" and "William Farley, *industrialist*." We have features for all *concepts* and therefore learn their association with each verb.

6.4 Experiments and Results

6.4.1 Set up

We parsed the 3 GB AQUAINT corpus [Vorhees, 2002] using Minipar [Lin, 1998b], and collected verb-object and verb-subject frequencies, building an empirical MI model from this data. Verbs and nouns were converted to their (possibly multi-token) *root*, and string case was preserved. Passive subjects (*the car was bought*) were converted to objects (*bought car*). We set the MI-threshold, τ, to be 0, and the negative-to-positive ratio, K, to be 2.

Numerous previous pseudodisambiguation evaluations only include arguments that occur between 30 and 3000 times [Erk, 2007; Keller and Lapata, 2003; Rooth *et al.*, 1999]. Presumably the lower bound is to help ensure the negative argument is unobserved because it is unsuitable, not because of data sparseness. We wish to use our model on arguments of any frequency, including those that never occurred in the training corpus (and therefore have empty co-occurrence features (Section 6.3.3)). We proceed as follows: first, we exclude pairs whenever the *noun* occurs less than 3 times in our corpus, removing many misspellings and other noise. Next, we omit verb co-occurrence features for nouns that occur less than 10 times, and instead fire a low-count feature. When we move to a new corpus, previously-unseen nouns are treated like these low-count training nouns. Note there are no specific restrictions on the frequency of *pairs*.

This processing results in a set of 6.8 million pairs, divided into 2318 partitions (192 of which are verb clusters (Section 6.3.2)). For each partition, we take 95% of the examples for training, 2.5% for development and 2.5% for a final unseen test set. We provide full results for two models: DSP_{cooc} which only uses the verb co-occurrence features, and DSP_{all} which uses all the features mentioned in Section 6.3.3. Feature values are normalized within each feature type. We train our (linear kernel) discriminative models using SVM^{light} [Joachims, 1999a] on each partition, but set meta-parameters C (regularization) and j (cost of positive vs. negative misclassifications: max at j=2) on the macro-averaged score across all

[3] Available at `http://www.mit.edu/~mbf/instances.txt.gz`

System	*MacroAvg*			*MicroAvg*			*Pairwise*	
	P	R	F	P	R	F	Acc	Cov
[Dagan *et al.*, 1999]	0.36	**0.90**	0.51	0.68	**0.92**	0.78	0.58	0.98
[Erk, 2007]	0.49	0.66	0.56	0.70	0.82	0.76	0.72	0.83
[Keller and Lapata, 2003]	**0.72**	0.34	0.46	**0.80**	0.50	0.62	0.80	0.57
DSP$_{cooc}$	0.53	0.72	0.61	0.73	0.94	**0.82**	0.77	**1.00**
DSP$_{all}$	0.60	0.71	**0.65**	0.77	0.90	**0.83**	**0.81**	**1.00**

Table 6.1: Pseudodisambiguation results averaged across each example (*MacroAvg*), weighted by word frequency (*MicroAvg*), plus coverage and accuracy of pairwise competition (*Pairwise*).

development partitions. Note that we can not use the development set to optimize τ and K because the development examples are obtained *after* setting these values.

6.4.2 Feature weights

It is interesting to inspect the feature weights returned by our system. In particular, the weights on the verb co-occurrence features (Section 6.3.3) provide a high-quality, argument-specific similarity-ranking of other verb contexts. The DSP parameters for *eat*, for example, place high weight on features like Pr(n|*braise*), Pr(n|*ration*), and Pr(n|*garnish*). Lin [1998a]'s similar word list for *eat* misses these but includes *sleep* (ranked 6) and *sit* (ranked 14), because these have similar *subjects* to *eat*. Discriminative, context-specific training seems to yield a better set of similar predicates, e.g. the highest-ranked contexts for DSP$_{cooc}$ on the verb *join*,[4]

> lead 1.42, rejoin 1.39, form 1.34, belong to 1.31, found 1.31, quit 1.29, guide 1.19, induct 1.19, launch (*subj*) 1.18, work at 1.14

give a better SIMS(*join*) for Equation (6.1) than the top similarities returned by [Lin, 1998a]:

> participate 0.164, lead 0.150, return to 0.148, say 0.143, rejoin 0.142, sign 0.142, meet 0.142, include 0.141, leave 0.140, work 0.137

Other features are also weighted intuitively. Note that capitalization is a strong indicator for some arguments, for example the weight on being lower-case is high for *become* (0.972) and *eat* (0.505), but highly negative for *accuse* (-0.675) and *embroil* (-0.573) which often take names of people and organizations.

6.4.3 Pseudodisambiguation

We first evaluate DSP on disambiguating positives from pseudo-negatives, comparing to recently-proposed systems that also require no manually-compiled resources like WordNet. We convert Dagan et al. [1999]'s similarity-smoothed probability to MI by replacing the

[4]Which all correspond to nouns occurring in the object position of the verb (e.g. Pr$_{obj}$(n|*lead*)), except "launch (*subj*)" which corresponds to Pr$_{subj}$(n|*launch*).

empirical $\Pr(n|v)$ in Equation (6.2) with the smoothed $\Pr_{\text{SIM}}$ from Equation (6.1). We also test an MI model inspired by Erk [2007]:

$$\text{MI}_{\text{SIM}}(n, v) = \log \sum_{n' \in \text{SIMS}(n)} \textit{Sim}(n', n) \frac{\Pr(v, n')}{\Pr(v)\Pr(n')}$$

We gather similar words using [Lin, 1998a], mining similar verbs from a comparable-sized parsed corpus, and collecting similar nouns from a broader 10 GB corpus of English text.[5]

We also use Keller and Lapata [2003]'s approach to obtaining predicate-argument counts from the web. Rather than mining parse trees, this technique retrieves counts for the pattern "V Det N" in raw online text, where V is any inflection of the verb, Det is *the*, *a*, or the empty string, and N is the singular or plural form of the noun. We compute a web-based MI by collecting $\Pr(n, v)$, $\Pr(n)$, and $\Pr(v)$ using all inflections, except we only use the root form of the noun. Rather than using a search engine, we obtain counts from the Google Web 5-gram Corpus (Chapter 3, Section 3.2.2).

All systems are thresholded at zero to make a classification. Unlike DSP, the comparison systems may not be able to score each example. The similarity-smoothed examples will be undefined if $\text{SIMS}(w)$ is empty. Also, the Keller and Lapata [2003] approach will be undefined if the pair is unobserved on the web. As a reasonable default for these cases, we assign them a negative decision.

We evaluate disambiguation using precision (P), recall (R), and their harmonic mean, F-Score (F). Table 6.1 gives the results of our comparison. In the *MacroAvg* results, we weight each example equally. For *MicroAvg*, we weight each example by the frequency of the noun. To more directly compare with previous work, we also reproduced *Pairwise Disambiguation* by randomly pairing each positive with one of the negatives and then evaluating each system by the percentage it ranks correctly (Acc). For the comparison approaches, if one score is undefined, we choose the other one. If both are undefined, we abstain from a decision. Coverage (Cov) is the percent of pairs where a decision was made.[6]

Our simple system with only verb co-occurrence features, DSP_{cooc}, outperforms all comparison approaches. Using the richer feature set in DSP_{all} results in a statistically significant gain in performance, up to an F-Score of 0.65 and a pairwise disambiguation accuracy of 0.81.[7] DSP_{all} has both broader coverage and better accuracy than all competing approaches. In the remainder of the experiments, we use DSP_{all} and refer to it simply as DSP.

Some errors are because of plausible but unseen arguments being used as test-set pseudo-negatives. For example, for the verb *damage*, DSP's three most high-scoring false positives are the nouns *jetliner*, *carpet*, and *gear*. While none occur with *damage* in our corpus, all intuitively satisfy the verb's selectional preferences.

MacroAvg performance is worse than *MicroAvg* because all systems perform better on frequent nouns. When we plot F-Score by noun frequency (Figure 6.1), we see that DSP outperforms comparison approaches across all frequencies, but achieves its biggest gains

[5]For both the similar-noun and similar-verb smoothing, we only smooth over similar pairs *that occurred in the corpus*. While averaging over all similar pairs tends to underestimate the probability, averaging over only the observed pairs tends to overestimate it. We tested both and adopt the latter because it resulted in better performance on our development set.

[6]I.e. we use the "half coverage" condition from Erk [2007].

[7]The differences between DSP_{all} and all comparison systems are statistically significant (McNemar's test, $p<0.01$).

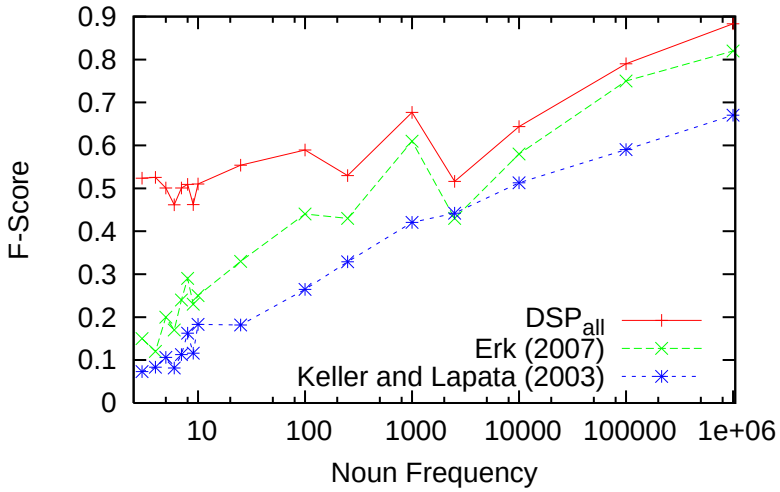


Figure 6.1: Disambiguation results by noun frequency.

on the low-frequency nouns. A richer feature set allows DSP to make correct inferences on examples that provide minimal co-occurrence data. These are also the examples for which we would expect co-occurrence models like MI to fail.

As a further experiment, we re-trained DSP but with only the string-based features removed. Overall macro-averaged F-score dropped from 0.65 to 0.64 (a statistically significant reduction in performance). The system scored nearly identically to DSP on the high-frequency nouns, but performed roughly 15% worse on the nouns that occurred less than ten times. This shows that the string-based features are important for selectional preference, and particularly helpful for low-frequency nouns.

Finally, we note that some suggestions for improving pseudo-word evaluations have been proposed in a very recent paper by Chambers and Jurafsky [Chambers and Jurafsky, 2010]. For example, our evaluation in this section only considers *unseen* pairs in the sense that our training and testing pairs are distinct. It may be more realistic to evaluate on all pairs extracted from a corpus, in which case we can directly compare to co-occurrence (or conditional probability, or MI), as in the following sections.

6.4.4 Human Plausibility

Table 6.2 compares some of our systems on data used by Resnik [1996] (also Appendix 2 in Holmes et al. [1989]). The plausibility of these pairs was initially judged based on the experimenters' intuitions, and later confirmed in a human experiment. We include the scores of Resnik's system, and note that its errors were attributed to sense ambiguity and other limitations of class-based approaches [Resnik, 1996].[8] The other comparison approaches also make a number of mistakes, which can often be traced to a misguided choice of similar word to smooth with.

We also compare to our empirical MI model, trained on our parsed corpus. Although

[8]For example, *warn-engine* scores highly because engines are in the class *entity*, and physical entities (e.g. people) are often objects of *warn*. Unlike DSP, Resnik's approach cannot learn that for *warn*, "the property of being a person is more important than the property of being an entity" [Resnik, 1996].

Verb	Plaus./Implaus.	Resnik	Dagan et al	Erk	MI	DSP
see	friend/method	5.79/-0.01	*0.20/1.40**	0.46/-0.07	1.11/-0.57	0.98/0.02
read	article/fashion	6.80/-0.20	3.00/0.11	3.80/1.90	4.00/—	2.12/-0.65
find	label/fever	1.10/0.22	*1.50/2.20**	0.59/0.01	0.42/0.07	1.61/0.81
hear	story/issue	*1.89/1.89**	*0.66/1.50**	*2.00/2.60**	2.99/-1.03	1.66/0.67
write	letter/market	7.26/0.00	2.50/-0.43	3.60/-0.24	5.06/-4.12	3.08/-1.31
urge	daughter/contrast	*1.14/1.86**	*0.14/1.60**	*1.10/3.60**	-0.95/—	-0.34/-0.62
warn	driver/engine	4.73/3.61	1.20/0.05	2.30/0.62	2.87/—	2.00/-0.99
judge	contest/climate	1.30/0.28	*1.50/1.90**	*1.70/1.70**	3.90/—	1.00/0.51
teach	language/distance	*1.87/1.86*	2.50/1.30	3.60/2.70	3.53/—	1.86/0.19
show	sample/travel	1.44/0.41	1.60/0.14	0.40/-0.82	0.53/-0.49	1.00/-0.83
expect	visit/mouth	*0.59/5.93**	*1.40/1.50**	1.40/0.37	1.05/-0.65	1.44/-0.15
answer	request/tragedy	4.49/3.88	2.70/1.50	3.10/-0.64	2.93/—	1.00/0.01
recognize	author/pocket	*0.50/0.50**	*0.03/0.37**	*0.77/1.30**	0.48/—	1.00/0.00
repeat	comment/journal	*1.23/1.23**	2.30/1.40	2.90/—	2.59/—	1.00/-0.48
understand	concept/session	*1.52/1.51*	2.70/0.25	2.00/-0.28	3.96/—	2.23/-0.46
remember	reply/smoke	1.31/0.20	2.10/1.20	*0.54/2.60**	1.13/-0.06	1.00/-0.42

Table 6.2: Selectional ratings for plausible/implausible direct objects (Holmes, 1989). Mistakes are marked with an asterisk (*), undefined scores are marked with a dash (—). Only DSP is completely defined and completely correct.

Seen Criteria	Unseen Verb-Object Freq.				
	All	$= 1$	$= 2$	$= 3$	> 3
MI > 0	0.44	0.33	0.57	0.70	0.82
Freq. > 0	0.57	0.45	0.76	0.89	0.96
DSP > 0	0.73	0.69	0.80	0.85	0.88

Table 6.3: Recall on identification of Verb-Object pairs from an unseen corpus (divided by pair frequency).

Resnik [1996] reported that 10 of the 16 plausible pairs did not occur in his training corpus, all of them occurred in ours and hence MI gives very reasonable scores on the plausible objects. It has no statistics, however, for many of the implausible ones. DSP can make finer decisions than MI, recognizing that "warning an engine" is more absurd than "judging a climate."

6.4.5 Unseen Verb-Object Identification

We next compare MI and DSP on a much larger set of plausible examples, and also test how well the models generalize across data sets. We took the MI and DSP systems trained on AQUAINT and asked them to rate observed (and thus likely plausible) verb-object pairs taken from an unseen corpus. We extracted the pairs by parsing the San Jose Mercury News (SJM) section of the TIPSTER corpus [Harman, 1992]. Each unique verb-object pair is a single instance in this evaluation.

Table 6.3 gives recall across all pairs **(All)** and grouped by pair-frequency in the unseen corpus (1, 2, 3, >3). DSP accepts far more pairs than MI (73% vs. 44%), even far more than a system that accepts any previously observed verb-object combination as plausible (57%). Recall is higher on more frequent verb-object pairs, but 70% of the pairs occurred

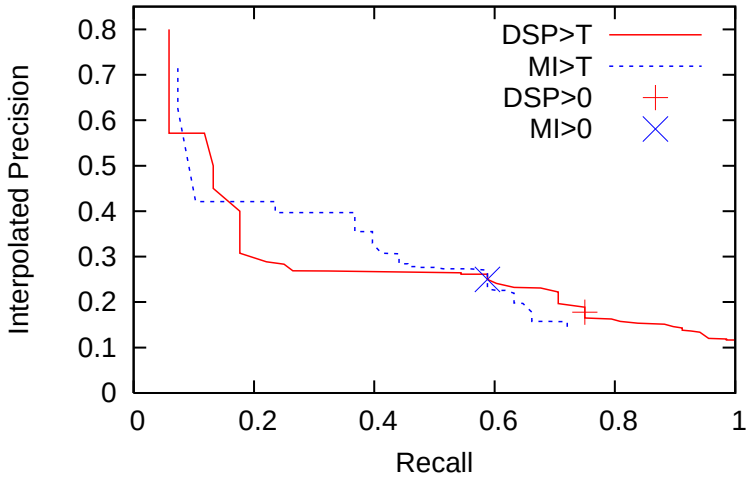


Figure 6.2: Pronoun resolution precision-recall on MUC.

only once in the corpus. Even if we smooth MI by smoothing $\Pr(n|v)$ in Equation 6.2 using modified KN-smoothing [Chen and Goodman, 1998], the recall of MI>0 on SJM only increases from 44.1% to 44.9%, still far below DSP. Frequency-based models have fundamentally low coverage. As further evidence, if we build a model of MI on the SJM corpus and use it in our pseudodisambiguation experiment (Section 6.4.3), MI>0 gets a *MacroAvg* precision of 86% but a *MacroAvg* recall of only 12%.[9]

6.4.6 Pronoun Resolution

Finally, we evaluate DSP on a common application of selectional preferences: choosing the correct antecedent for pronouns in text [Dagan and Itai, 1990; Kehler *et al.*, 2004]. We study the cases where a pronoun is the direct object of a verb predicate, v. A pronoun's antecedent must obey v's selectional preferences. If we have a better model of SP, we should be able to better select pronoun antecedents.[10]

We parsed the MUC-7 [1997] coreference corpus and extracted all pronouns in a direct object relation. For each pronoun, p, modified by a verb, v, we extracted all preceding nouns within the current or previous sentence. Thirty-nine anaphoric pronouns had an antecedent in this window and are used in the evaluation. For each p, let $N(p)^+$ by the set of preceding nouns coreferent with p, and let $N(p)^-$ be the remaining non-coreferent nouns. We take all (v, n^+) where $n^+ \in N(p)^+$ as positive, and all other pairs (v, n^-), $n^- \in N(p)^-$ as negative.

We compare MI and DSP on this set, classifying every (v, n) with MI$>T$ (or DSP$>T$) as positive. By varying T, we get a precision-recall curve (Figure 6.2). Precision is low

[9] Recall that even the Keller and Lapata [2003] system, built on the world's largest corpus, achieves only 34% recall (Table 6.1) (with only 48% of positives and 27% of all pairs previously observed, but note, on the other hand, that low-count N-grams have been filtered from the N-gram corpus, and therefore perhaps this effect is overstated).

[10] Note we're not trying to answer the question of whether selectional preferences are useful [Yang *et al.*, 2005] or not [Kehler *et al.*, 2004] for resolving pronouns when combined with features for recency, frequency, gender, syntactic role of the candidate, etc. We are only using this task as another evaluation for our models of selectional preference.

System	Acc
Most-Recent Noun	17.9%
Maximum MI	28.2%
Maximum DSP	38.5%

Table 6.4: Pronoun resolution accuracy on nouns in current or previous sentence in MUC.

because, of course, there are many nouns that satisfy the predicate's SPs that are not coreferent. DSP>0 has both a higher recall and higher precision than accepting every pair previously seen in text (the right-most point on MI$>T$). The DSP$>T$ system achieves higher precision than MI$>T$ for points where recall is greater than 60% (where MI<0). Interestingly, the recall of MI>0 is higher here than it is for general verb-objects (Section 6.4.5). On the subset of pairs with strong empirical association (MI>0), MI generally outperforms DSP at equivalent recall values.

We next compare MI and DSP as stand-alone pronoun resolution systems (Table 6.4). As a standard baseline, for each pronoun, we choose the most recent noun in text as the pronoun's antecedent, achieving 17.9% resolution accuracy. This baseline is low because many of the most-recent nouns are subjects of the pronoun's verb phrase, and therefore resolution would violate syntactic coreference constraints. If we choose the previous noun with the highest MI as antecedent, we get an accuracy of 28.2%, while choosing the previous noun with the highest DSP achieves 38.5%. DSP resolves 37% more pronouns correctly than MI.

6.5 Conclusions and Future Work

We have proposed a simple, effective model of selectional preference based on discriminative training. Supervised techniques typically achieve better performance than unsupervised models, and we duplicate these gains with DSP. Here, however, these gains come at no additional labeling cost, as training examples are generated automatically from unlabeled text.

DSP allows an arbitrary combination of features, including verb co-occurrence features that yield high-quality similar-word lists as latent output. These lists not only indicate which verbs are associated with a common set of nouns; they provide insight into a chain of narrative events in which a particular noun may participate (e.g., a particular noun may be *bought*, then *cooked*, then *garnished*, and then likely *eaten*). DSP therefore learns similar information to previous approaches that seek such narrative events directly and explicitly [Bean and Riloff, 2004; Chambers and Jurafsky, 2008]. However, note that we learn the association of events across a corpus rather than only in a particular segment of discourse, like a document or paragraph. One option for future work is to model the local and global distribution of nouns separately, allowing for finer-grained (and potentially sense-specific) plausibility predictions.

The features used in DSP only scratch the surface of possible feature mining; information from WordNet relations, Wikipedia categories, or parallel corpora could also provide valuable clues for selectional preference. Also, if any other system were to exceed DSP's performance, it could also be included as one of DSP's features.

It would be interesting to expand our co-occurrence features, including co-occurrence counts across more grammatical relations and using counts from external, unparsed corpora

like the world wide web. We could also reverse the role of noun and verb in our training, having verb-specific features and discriminating separately for each argument noun. The latent information would then be lists of similar nouns.

Finally, one potentially very exciting direction for future work would be to automatically collect and create features for online *images* returned for the noun query. It would be amazing to be able to predict noun-verb plausibility purely on the basis of a system's learned visual recognition of compatible features (e.g., this collection of images seems to depict something that can be eaten, etc.).

DSP provides an excellent framework for such explorations because it generates many training examples and can therefore incorporate fine-grained, overlapping and potentially interdependent features. We look at another system that has these properties in the following chapter.

Chapter 7

Alignment-Based Discriminative String Similarity

> "Kimono... kimono... kimono... Ha! Of course! Kimono is come from the Greek word himona, is mean winter. So, what do you wear in the wintertime to stay warm? A robe. You see: robe, kimono. There you go!"
> - Gus Portokalos, *My Big Fat Greek Wedding*

This chapter proposes a new model of string similarity that exploits a character-based alignment of the two strings. We again adopt a discriminative approach. Positive pairs are generated automatically from word pairs with a high association in an aligned bitext, or else mined from dictionary translations. Negatives are constructed from pairs with a high amount of character overlap, but which are not translations. So in this work, there are three types of information that allow us to generate examples automatically: statistics from a bitext, entries in a dictionary, and characters in the strings. This information is unlabeled in the sense that no human annotator has specifically labeled cognates in the data. It is useful because once a definition is adopted, examples can be generated automatically, and different methods can be empirically evaluated on a level playing field.

7.1 Introduction

String similarity is often used as a means of quantifying the likelihood that two pairs of strings have the same underlying meaning, based purely on the character composition of the two words. [Strube *et al.*, 2002] use edit distance [Levenshtein, 1966] as a feature for determining if two words are coreferent. [Taskar *et al.*, 2005] use French-English common letter sequences as a feature for discriminative word alignment in bilingual texts. [Brill and Moore, 2000] learn misspelled-word to correctly-spelled-word similarities for spelling correction. In each of these examples, a similarity measure can make use of the recurrent substring pairings that reliably occur between words having the same meaning.

Across natural languages, these recurrent substring correspondences are found in word pairs known as cognates: words with a common form and meaning across languages. Cognates arise either from words in a common ancestor language (e.g. *light/Licht, night/Nacht* in English/German) or from foreign word borrowings (e.g. *trampoline/toranporin* in English/Japanese). Knowledge of cognates is useful for a number of applications, including

[0] A version of this chapter has been published as [Bergsma and Kondrak, 2007a]

word alignment [Kondrak *et al.*, 2003], sentence alignment [Simard *et al.*, 1992; Church, 1993; McEnery and Oakes, 1996; Melamed, 1999] and learning translation lexicons [Mann and Yarowsky, 2001; Koehn and Knight, 2002]. The related task of identifying *transliterations* has also received much recent attention [Klementiev and Roth, 2006; Zelenko and Aone, 2006; Yoon *et al.*, 2007; Jiampojamarn *et al.*, 2010]. Extending dictionaries with automatically-acquired knowledge of cognates and transliterations can improve machine translation systems [Knight *et al.*, 1995].

Also, cognates have been used to help assess the readability of a foreign language text by new language learners [Uitdenbogerd, 2005]. Developing automatic ways to identify these cognates is thus a prerequisite for a robust automatic readability assessment.

We propose an alignment-based, discriminative approach to string similarity and we evaluate this approach on the task of cognate identification. Section 7.2 describes previous approaches and their limitations. In Section 7.3, we explain our technique for automatically creating a cognate-identification training set. A novel aspect of this set is the inclusion of *competitive counter-examples* for learning. Section 7.4 shows how discriminative features are created from a character-based, minimum-edit-distance alignment of a pair of strings. In Section 7.5, we describe our bitext and dictionary-based experiments on six language pairs, including three based on non-Roman alphabets. In Section 7.6, we show significant improvements over traditional approaches, as well as significant gains over more recent techniques by [Ristad and Yianilos, 1998], [Tiedemann, 1999], [Kondrak, 2005], and [Klementiev and Roth, 2006].

7.2 Related Work

String similarity is a fundamental concept in a variety of fields and hence a range of techniques have been developed. We focus on approaches that have been applied to words, i.e., uninterrupted sequences of characters found in natural language text. The most well-known measure of the similarity of two strings is the edit distance or Levenshtein distance [Levenshtein, 1966]: the number of insertions, deletions and substitutions required to transform one string into another. In our experiments, we use *normalized* edit distance (NED): edit distance divided by the length of the longer word. Other popular measures include Dice's Coefficient (DICE) [Adamson and Boreham, 1974], and the length-normalized measures longest common subsequence ratio (LCSR) [Melamed, 1999], the length of the longest common subsequence divided by the length of the longer word (used by [Melamed, 1998]), and longest common prefix ratio (PREFIX) [Kondrak, 2005], the length of the longest common prefix divided by the longer word length (four-letter prefix match was used by [Simard *et al.*, 1992]). These baseline approaches have the important advantage of not requiring training data. We can also include in the non-learning category [Kondrak, 2005]'s longest common subsequence formula (LCSF), a probabilistic measure designed to mitigate LCSR's preference for shorter words.

Although simple to use, the untrained measures cannot adapt to the specific spelling differences between a pair of languages. Researchers have therefore investigated adaptive measures that are learned from a set of known cognate pairs. [Ristad and Yianilos, 1998] developed a stochastic transducer version of edit distance learned from unaligned string pairs. [Mann and Yarowsky, 2001] saw little improvement over edit distance when applying this transducer to cognates, even when filtering the transducer's probabilities into different weight classes to better approximate edit distance. [Tiedemann, 1999] used var-

ious measures to learn the recurrent spelling changes between English and Swedish, and used these changes to re-weight LCSR to identify more cognates, with modest performance improvements. [Mulloni and Pekar, 2006] developed a similar technique to improve NED for English/German.

Essentially, all these techniques improve on the baseline approaches by using a set of positive (true) cognate pairs to re-weight the costs of edit operations or the score of sequence matches. Ideally, we would prefer a more flexible approach that can learn positive *or* negative weights on *substring* pairings in order to better identify related strings. One system that can potentially provide this flexibility is a discriminative string-similarity approach to named-entity transliteration by [Klementiev and Roth, 2006]. Although not compared to other similarity measures in the original paper, we show that this discriminative technique can strongly outperform traditional methods on cognate identification.

Unlike many recent generative systems, the Klementiev and Roth approach does not exploit the known positions in the strings where the characters match. For example, [Brill and Moore, 2000] combine a character-based alignment with the expectation maximization (EM) algorithm to develop an improved probabilistic error model for spelling correction. [Rappoport and Levent-Levi, 2006] apply this approach to learn substring correspondences for cognates. [Zelenko and Aone, 2006] recently showed a [Klementiev and Roth, 2006]-style discriminative approach to be superior to alignment-based generative techniques for name transliteration. Our work successfully uses the alignment-based methodology of the generative approaches to enhance the feature set for discriminative string similarity. In work concurrent to our original contribution in [Bergsma and Kondrak, 2007a], Yoon et al. [2007] apply a discriminative approach to recognizing transliterations at the phoneme level. They include binary features over aligned phoneme pairs, but do not use features over phoneme *subsequences* as would be the analog of our work.

Finally, [Munteanu and Marcu, 2005] propose a similar approach to detect *sentences* that are translations in non-parallel corpora. The heart of their algorithm is a classifier that inspects a pair of sentences and decides if they are translations. Like us, they also align the sentences and compute features based on the alignment, but they use more general features (e.g., number of words in a row that are aligned, etc.) rather than, say, phrase pairs that are consistent with the alignment, which would be the direct analogue of our method. Although we originally developed our approach unaware of the connection to this work, the two approaches ultimately face many similar issues and developed similar solutions. In particular, they also automatically generate training pairs from both true sentence translations (positives) and *competitive* counter examples (negatives). Since they can also generate many examples using this technique, it is surprising they did not also explore much richer, finer-grained features like those explored in this chapter.

7.3 The Cognate Identification Task

Given two string lists, E and F, the task of cognate identification is to find all pairs of strings (e, f) that are cognate. In other similarity-driven applications, E and F could be misspelled and correctly spelled words, or the orthographic and the phonetic representation of words, etc. The task remains to link strings with common meaning in E and F using only the string similarity measure.

We can facilitate the application of string similarity to cognates by using a definition of cognation not dependent on etymological analysis. For example, [Mann and Yarowsky,

Foreign Language F	Words $f \in F$	Cognates E_{f+}	False Friends E_{f-}
Japanese (Rômaji)	napukin	napkin	nanking, pumpkin, snacking, sneaking
French	abondamment	abundantly	abandonment, abatement, ... wonderment
German	prozyklische	procyclical	polished, prophylactic, prophylaxis
Spanish	viudos	widows	avoids, idiots, video, videos, virtuous

Table 7.1: Foreign-English cognates and false friend training examples.

2001] define a word pair (e, f) to be cognate if they are a translation pair (same meaning) and their edit distance is less than three (same form). We adopt an improved definition (suggested by [Melamed, 1999] for the French-English Canadian Hansards) that does not over-propose shorter word pairs: (e, f) are cognate if they are translations and their LCSR ≥ 0.58. Note that this cutoff is somewhat conservative: the English/German cognates *light/Licht* (LCSR=0.8) are included, but not the cognates *eight/acht* (LCSR=0.4).

If two words must have LCSR ≥ 0.58 to be cognate, then for a given word $f \in F$, we need only consider as possible cognates the subset of words in E having an LCSR with f larger than 0.58, a set we call E_f. The portion of E_f with the same meaning as f, E_{f+}, are cognates, while the part with different meanings, E_{f-}, are not cognates. The words E_{f-} with similar spelling but different meaning are sometimes called *false friends*. The cognate identification task is, for every word $f \in F$, and a list of similarly spelled words E_f, to distinguish the cognate subset E_{f+} from the false friend set E_{f-}.

To create training data for our learning approaches, and to generate a high-quality labeled test set, we need to annotate some of the $(f, e_f \in E_f)$ word pairs for whether or not the words share a common meaning. In Section 7.5, we explain our two high-precision automatic annotation methods: checking if each pair of words (a) were aligned in a word-aligned bitext, or (b) were listed as translation pairs in a bilingual dictionary.

Table 7.1 provides some labeled examples with non-empty cognate and false friend lists. Note that despite what it may appear from these examples, this is not a ranking task: even in highly related languages, most words in F have empty E_{f+} lists, and many have empty E_{f-} as well. Thus one natural formulation for cognate identification is a pairwise (and symmetric) cognation classification that looks at each pair (f, e_f) separately and individually makes a decision:

+(napukin,napkin)
– (napukin,nanking)
– (napukin,pumpkin)

In this formulation, the benefits of a discriminative approach are clear: it must find substrings that distinguish cognate pairs from word pairs with otherwise similar form. [Klementiev and Roth, 2006], although using a discriminative approach, do not provide their infinite-attribute perceptron with competitive counter-examples. They instead use transliterations as positives and randomly-paired English and Russian words as negative examples. In the following section, we also improve on [Klementiev and Roth, 2006] by using a character-based string alignment to focus the features for discrimination.

7.4 Features for Discriminative String Similarity

As Chapter 2 explained, discriminative training learns a classifier from a set of labeled training examples, each represented as a set of features. In the previous section we showed

how labeled word pairs can be collected. We now address methods of representing these word pairs as sets of features useful for determining cognation.

Consider the Rômaji Japanese/English cognates: *(sutoresu,stress)*. The LCSR is 0.625. Note that the LCSR of *sutoresu* with the English false friend *stories* is higher: 0.75. LCSR alone is too weak a feature to pick out cognates. We need to look at the actual character substrings.

[Klementiev and Roth, 2006] generate features for a pair of words by splitting both words into all possible substrings of up to size two:
sutoresu $\Rightarrow$ { *s, u, t, o, r, e, s, u, su, ut, to, ... su* }
stress $\Rightarrow$ { *s, t, r, e, s, s, st, tr, re, es, ss* }
Then, a feature vector is built from all substring pairs from the two words such that the difference in positions of the substrings is within one:
{*s-s, s-t, s-st, su-s, su-t, su-st, su-tr... r-s, r-s, r-es...*}
This feature vector provides the feature representation used in supervised machine learning.

This example also highlights the limitations of the Klementiev and Roth approach. The learner can provide weight to features like *s-s* or *s-st* at the beginning of the word, but because of the gradual accumulation of positional differences, the learner never sees the *tor-tr* and *es-es* correspondences that really help indicate the words are cognate.

Our solution is to use the minimum-edit-distance alignment of the two strings as the basis for feature extraction, rather than the positional correspondences. We also include beginning-of-word (^) and end-of-word ($) markers (referred to as *boundary markers*) to highlight correspondences at those positions. The pair $(sutoresu, stress)$ can be aligned:

For the feature representation, we only extract substring pairs that are consistent with this alignment.[1] That is, the letters in our pairs can only be aligned to each other and not to letters outside the pairing:
{ ^-^,^s-^s, *s-s, su-s, ut-t, t-t,... es-es, s-s, su-ss...*}
We define *phrase* pairs to be the pairs of substrings consistent with the alignment. A similar use of the term "phrase" exists in machine translation, where phrases are often pairs of word sequences consistent with word-based alignments [Koehn *et al.*, 2003].

By limiting the substrings to only those pairs that are consistent with the alignment, we generate fewer, more-informative features. Computationally, using more-precise features allows a larger maximum substring size L than is feasible with the positional approach. Larger substrings allow us to capture important recurring deletions like the "u" in the pair *sut-st* observed in Japanese-English.

[Tiedemann, 1999] and others have shown the importance of using the mismatching portions of cognate pairs to learn the recurrent spelling changes between two languages. In order to capture mismatching segments longer than our maximum substring size will allow, we include special features in our representation called *mismatches*. *Mismatches* are phrases that span the entire sequence of unaligned characters between two pairs of

[1] If the words are from different writing systems, we can get the alignment by mapping the foreign letters to their closest Roman equivalent, or by using the EM algorithm to learn the edits [Ristad and Yianilos, 1998]. In recent work in recognizing transliterations between different writing systems [Jiampojamarn *et al.*, 2010], we used the output of a many-to-many alignment model [Jiampojamarn *et al.*, 2007] to directly extract the substring-alignment features.

aligned end characters (similar to the "rules" extracted by [Mulloni and Pekar, 2006]). In the above example, *su$-ss$* is a mismatch with "s" and "$" as the aligned end characters. Two sets of features are taken from each mismatch, one that includes the beginning/ending aligned characters as context and one that does not. For example, for the endings of the French/English pair *(économique,economic)*, we include both the substring pairs *ique$:ic$* and *que:c* as features.

One consideration is whether substring features should be binary presence/absence, or the count of the feature in the pair normalized by the length of the longer word. We investigate both of these approaches in our experiments. Also, there is no reason not to include the scores of baseline approaches like NED, LCSR, PREFIX or DICE as features in the representation as well. Features like the lengths of the two words and the difference in lengths of the words have also proved to be useful in preliminary experiments. Semantic features like frequency similarity or contextual similarity might also be included to help determine cognation between words that are not present in a translation lexicon or bitext.

7.5 Experiments

Section 7.3 introduced two high-precision methods for generating labeled cognate pairs: using the word alignments from a bilingual corpus or using the entries in a translation lexicon. We investigate both of these methods in our experiments. In each case, we generate sets of labeled word pairs for training, testing, and development. The proportion of positive examples in the bitext-labeled test sets range between 1.4% and 1.8%, while ranging between 1.0% and 1.6% for the dictionary data.[2]

For the discriminative methods, we use a popular support vector machine (SVM) learning package called SVM^{light} [Joachims, 1999a]. As Chapter 2 describes, SVMs are maximum-margin classifiers that achieve good performance on a range of tasks. In each case, we learn a linear kernel on the training set pairs and tune the parameter that trades-off training error and margin on the development set. We apply our classifier to the test set and score the pairs by their positive distance from the SVM classification hyperplane (also done by [Bilenko and Mooney, 2003] with their token-based SVM similarity measure).

We also score the test sets using traditional orthographic similarity measures PREFIX, DICE, LCSR, and NED, an average of these four, and [Kondrak, 2005]'s LCSF. We also use the log of the edit probability from the stochastic decoder of [Ristad and Yianilos, 1998] (normalized by the length of the longer word) and [Tiedemann, 1999]'s highest performing system (Approach #3). Both use only the positive examples in our training set. Our evaluation metric is 11-pt average precision on the score-sorted pair lists (also used by [Kondrak and Sherif, 2006]).

7.5.1 Bitext Experiments

For the bitext-based annotation, we use publicly-available word alignments from the Europarl corpus, automatically generated by GIZA++ for French-English (Fr), Spanish-English (Es) and German-English (De) [Koehn, 2005; Koehn and Monz, 2006]. Initial cleaning of these noisy word pairs is necessary. We thus remove all pairs with numbers, punctuation, a capitalized English word, and all words that occur fewer than ten times. We also remove

[2]The cognate data sets used in our experiments are available at `http://www.cs.ualberta.ca/~bergsma/Cognates/`

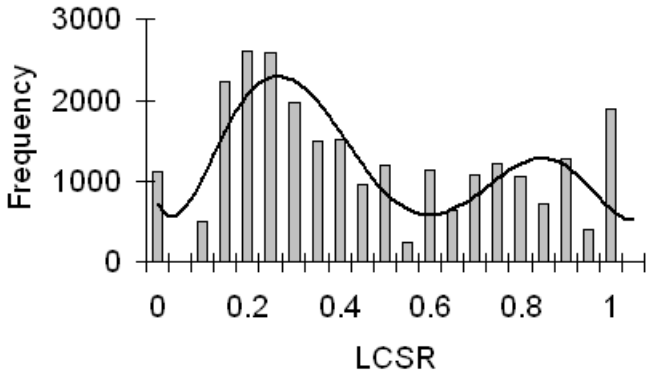


Figure 7.1: LCSR histogram and polynomial trendline of French-English dictionary pairs.

many incorrectly aligned words by filtering pairs where the pointwise Mutual Information between the words is less than 7.5. This processing leaves vocabulary sizes of 39K for French, 31K for Spanish, and 60K for German.

Our labeled set is then generated from pairs with LCSR ≥ 0.58 (using the cutoff from [Melamed, 1999]). Each labeled set entry is a triple of a) the foreign word f, b) the cognates E_{f+} and c) the false friends E_{f-}. For each language pair, we randomly take 20K triples for training, 5K for development and 5K for testing. Each triple is converted to a set of pairwise examples for learning and classification.

7.5.2 Dictionary Experiments

For the dictionary-based cognate identification, we use French, Spanish, German, Greek (Gr), Japanese (Jp), and Russian (Rs) to English translation pairs from the Freelang program.[3] The latter three pairs were chosen so that we can evaluate on more distant languages that use non-Roman alphabets (although the Rômaji Japanese is Romanized by definition). We take 10K labeled-set triples for training, 2K for testing and 2K for development.

The baseline approaches and our definition of cognation require comparison in a common alphabet. Thus we use a simple context-free mapping to convert every Russian and Greek character in the word pairs to their nearest Roman equivalent. We then label a translation pair as cognate if the LCSR between the words' Romanized representations is greater than 0.58. We also operate all of our comparison systems on these Romanized pairs.

7.6 Results

We were interested in whether our working definition of cognation (translations and LCSR ≥ 0.58) reflects true etymological relatedness. We looked at the LCSR histogram for translation pairs in one of our translation dictionaries (Figure 7.1). The trendline suggests a bimodal distribution, with two distinct distributions of translation pairs making up the dictionary: incidental letter agreement gives low LCSR for the larger, non-cognate portion and high LCSR characterizes the likely cognates. A threshold of 0.58 captures most of the cognate distribution while excluding non-cognate pairs. This hypothesis was confirmed by checking the LCSR values of a list of known French-English cognates (randomly collected

[3] `http://www.freelang.net/dictionary/`

System	Prec
Klementiev-Roth (KR) L≤2	58.6
KR L≤2 (normalized, boundary markers)	62.9
phrases L≤2	61.0
phrases L≤3	65.1
phrases L≤3 + *mismatches*	65.6
phrases L≤3 + *mismatches* + NED	**65.8**

Table 7.2: Bitext French-English *development set* cognate identification 11-pt average precision (%).

	Bitext			Dictionary					
System	Fr	Es	De	Fr	Es	De	Gr	Jp	Rs
PREFIX	34.7	27.3	36.3	45.5	34.7	25.5	28.5	16.1	29.8
DICE	33.7	28.2	33.5	44.3	33.7	21.3	30.6	20.1	33.6
LCSR	34.0	28.7	28.5	48.3	36.5	18.4	30.2	24.2	36.6
NED	36.5	**31.9**	32.3	50.1	**40.3**	23.3	**33.9**	28.2	41.4
PREFIX+DICE+LCSR+NED	**38.7**	31.8	**39.3**	**51.6**	40.1	**28.6**	33.7	22.9	37.9
[Kondrak, 2005]: LCSF	29.8	28.9	29.1	39.9	36.6	25.0	30.5	**33.4**	**45.5**
[Ristad and Yianilos, 1998]	37.7	32.5	34.6	56.1	46.9	36.9	38.0	52.7	51.8
[Tiedemann, 1999]	38.8	33.0	34.7	55.3	49.0	24.9	37.6	33.9	45.8
[Klementiev and Roth, 2006]	61.1	55.5	53.2	73.4	62.3	48.3	51.4	62.0	64.4
Alignm-Based Discrim.	**66.5**	**63.2**	**64.1**	**77.7**	**72.1**	**65.6**	**65.7**	**82.0**	**76.9**

Table 7.3: Bitext, Dictionary Foreign-to-English cognate identification 11-pt average precision (%).

from a dictionary for another project): 87.4% were above 0.58. We also checked cognation on 100 randomly-sampled, positively-labeled French-English pairs from both the dictionary and bitext data (i.e. translated or aligned and having LCSR $\geq$ 0.58). 100% of the dictionary pairs and 93% of the bitext pairs were cognate.

Next, we investigate various configurations of the discriminative systems on one of our cognate identification development sets (Table 7.2). The original [Klementiev and Roth, 2006] (KR) system can be improved by normalizing the feature count by the longer string length and including the boundary markers. This is therefore done with all the alignment-based approaches. Also, because of the way its features are constructed, the KR system is limited to a maximum substring length of two (L≤2). A maximum length of three (L≤3) in the KR framework produces millions of features and prohibitive training times, while L≤3 is computationally feasible in the phrasal case, and increases precision by 4.1% over the *phrases L≤2* system.[4] Including *mismatches* results in another small boost in performance (0.5%), while using an edit distance feature again increases performance by a slight margin (0.2%). This ranking of configurations is consistent across all the bitext-based development sets; we therefore take the configuration of the highest scoring system as our Alignment-Based Discriminative system for the remainder of this paper.

[4] At the time of this research, preliminary experiments using even longer phrases (beyond L≤3) produced a computationally prohibitive number of features for SVM learning. Deploying feature selection techniques might enable the use of even more expressive and powerful feature sets with longer phrase lengths.

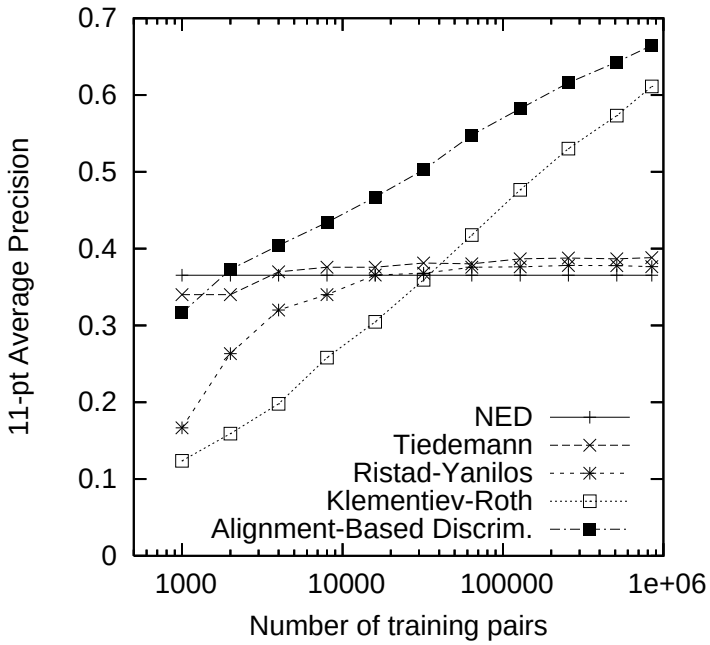


Figure 7.2: Bitext French-English cognate identification learning curve.

We next compare the Alignment-Based Discriminative scorer to the various other implemented approaches across the three bitext and six dictionary-based cognate identification test sets (Table 7.3). The table highlights the top system among both the non-adaptive and adaptive similarity scorers.[5] In each language pair, the alignment-based discriminative approach outperforms all other approaches, but the KR system also shows strong gains over non-adaptive techniques and their re-weighted extensions. This is in contrast to previous comparisons which have only demonstrated minor improvements with adaptive over traditional similarity measures [Kondrak and Sherif, 2006].

We consistently found that the original KR performance could be surpassed by a system that normalizes the KR feature count and adds boundary markers. Across all the test sets, this modification results in a 6% average gain in performance over baseline KR, but is still on average 5% below the Alignment-Based Discriminative technique, with a statistically significantly difference on each of the nine sets.[6]

Figure 7.2 shows the relationship between training data size and performance in our bitext-based French-English data. Note again that the Tiedemann and Ristad & Yanilos systems only use the positive examples in the training data. Our alignment-based similarity function outperforms all the other systems across nearly the entire range of training data. Note also that the discriminative learning curves show no signs of slowing down: performance grows logarithmically from 1K to 846K word pairs.

For insight into the power of our discriminative approach, we provide some of our classifiers' highest and lowest-weighted features (Table 7.4). Note the expected correspondences between foreign spellings and English (*k-c*, *f-ph*), but also features that lever-

[5]Using the training data and the SVM to weight the components of the PREFIX+DICE+LCSR+NED scorer resulted in negligible improvements over the simple average on our development data.

[6]Following [Evert, 2004], significance was computed using Fisher's exact test (at $p = 0.05$) to compare the n-best word pairs from the scored test sets, where n was taken as the number of positive pairs in the set.

Lang.	Feat.	Wt.	Example
Fr (Bitext)	ées - ed	+8.0	vérifiées:verified
Jp (Dict.)	ru - l	+5.9	penaruti:penalty
De (Bitext)	k - c	+5.5	kreativ:creative
De Btxt	eren$ - e$	+5.2	ignorieren:ignore
Fr Btxt	lement$ - ly$	+5.2	admirablement admirably
Es (Bitext)	ar - ating	+5.0	acelerar:accelerating
Rs (Dict.)	irov - _	+4.9	motivirovat:motivate
Gr (Dict.)	f - ph	+4.1	symfonia:symphony
Gr (Dict.)	kos - c	+3.3	anarchikos:anarchic
Gr (Dict.)	os$ - y$	-2.5	*anarchikos:anarchy*
Jp (Dict.)	ou - ou	-2.6	*handoutai:handout*
Es (Dict.)	_ - un	-3.1	*balance:unbalance*
Fr (Bitext)	s$ - ly$	-4.2	*fervents:fervently*
Fr (Dict.)	er$ - er$	-5.0	*former:former*
Es (Bitext)	mos - s	-5.1	*toleramos:tolerates*

Table 7.4: Example features and weights for various Alignment-Based Discriminative classifiers (Foreign-English, negative pairs in *italics*).

age derivational and inflectional morphology. For example, Greek-English pairs with the adjective-ending correspondence *kos-c*, e.g. *anarchikos:anarchic*, are favoured, but pairs with the adjective ending in Greek and noun ending in English, *os$-y$*, are penalized; indeed, by our definition, *anarchikos:anarchy* is not cognate. In a bitext, the feature *ées-ed* captures that feminine-plural inflection of past tense verbs in French corresponds to regular past tense in English. On the other hand, words ending in the Spanish first person plural verb suffix *-amos* are rarely translated to English words ending with the suffix *-s*, causing *mos-s* to be penalized. The ability to leverage negative features, learned from appropriate counter examples, is a key innovation of our discriminative framework.

Table 7.5 gives the top pairs scored by our system on the three bitext and three of the dictionary test sets. Notice that unlike traditional similarity measures that always score identical words higher than all other pairs, by virtue of our feature weighting, our discriminative classifier prefers some pairs with very characteristic spelling changes.

We performed error analysis by looking at all the pairs our system scored quite confidently (highly positive or highly negative similarity), but which were labeled oppositely. Highly-scored false positives arose equally from 1) actual cognates not linked as translations in the data, 2) related words with diverged meanings, e.g. the only error in Table 7.5: *makaroni* in Greek actually means *spaghetti* in English (*makaronada* is *macaroni*), and 3) the same word stem, a different part of speech (e.g. the Greek/English adjective/noun *synonymos:synonym*). Meanwhile, inspection of the highly-confident false negatives revealed some (often erroneously-aligned in the bitext) positive pairs with incidental letter match (e.g. the French/English *recettes:proceeds*) that we would not actually deem to be cognate. Thus the errors that our system makes are often either linguistically interesting or point out mistakes in our automatically-labeled bitext and (to a lesser extent) dictionary data.

Fr-En Bitext	Es-En Bitext	De-En Bitext
film:film	agenda:agenda	akt:act
ambassadeur:ambassador	natural:natural	asthma:asthma
bio:bio	márgenes:margins	lobby:lobby
radios:radios	hormonal:hormonal	homosexuell:homosexual
abusif:abusive	radón:radon	brutale:brutal
irréfutable:irrefutable	higiénico:hygienic	inzidenz:incidence
Gr-En Dict.	**Jp-En Dict.**	**Rs-En Dict.**
alkali:alkali	baiohoronikusu:bioholonics	aerozol:aerosol
makaroni:macaroni	mafia:mafia	gondola:gondola
adrenalini:adrenaline	manierisumu:manierisme	rubidiy:rubidium
flamingko:flamingo	ebonaito:ebonite	panteon:pantheon
spasmodikos:spasmodic	oratorio:oratorio	antonim:antonym
amvrosia:ambrosia	mineraru:mineral	gladiator:gladiator

Table 7.5: Highest scored pairs by Alignment-Based Discriminative classifier (negative pair in *italics*).

7.7 Conclusion and Future Work

This is the first research to apply discriminative string similarity to the task of cognate identification. We have introduced and successfully applied an alignment-based framework for discriminative similarity that consistently demonstrates improved performance in both bitext and dictionary-based cognate identification on six language pairs. Our improved approach can be applied in any of the diverse applications where traditional similarity measures like edit distance and LCSR are prevalent. We have also made available our cognate identification data sets, which will be of interest to general string similarity researchers.

Furthermore, we have provided a natural framework for future cognate identification research. Phonetic, semantic, or syntactic features could be included within our discriminative infrastructure to aid in the identification of cognates in text. In particular, we could investigate approaches that do not require the bilingual dictionaries or bitexts to generate training data. For example, researchers have automatically developed translation lexicons by seeing if words from each language have similar frequencies, contexts [Koehn and Knight, 2002], burstiness, inverse document frequencies, and date distributions [Schafer and Yarowsky, 2002]. Semantic and string similarity might be learned jointly with a co-training or bootstrapping approach [Klementiev and Roth, 2006]. We may also compare alignment-based discriminative string similarity with a more complex discriminative model that learns the alignments as latent structure [McCallum *et al.*, 2005].

Since the original publication of this work, we have also applied the alignment-based string similarity model to the task of transliteration identification [Jiampojamarn *et al.*, 2010] with good results. In that work, we also proposed a new model of string similarity that uses a string kernel to implicitly represent substring pairs of arbitrary length, lifting one of the computational limitations of the model in this chapter.

In addition, we also looked specifically at the cognate identification problem from a multilingual perspective [Bergsma and Kondrak, 2007b]. While the current chapter looks to detect cognates in pairs of languages, we provided a methodology that directly forms

sets of cognates across groups of languages. We showed improvements over simple clustering techniques that do not inherently consider the transitivity of cognate relations. We developed our multi-lingual approach via the global inference framework of integer linear programming. We followed Roth and Yih [2004] in using binary-$\{0, 1\}$ ILP variables to represent the decisions made by our system (cognate or not a cognate), and we optimized as our objective function the sum of the costs/scores of the decisions, with constraints for transitivity and one-to-one mappings across languages. Our formulation was partly based on similar solutions for other tasks by [Barzilay and Lapata, 2006; Denis and Baldridge, 2007]. Application of these techniques should improve the detection of translated *sentences* as well [Munteanu and Marcu, 2005], since transitivity across languages also applies, of course, at the sentence level.

Chapter 8

Conclusions and Future Work

8.1 Summary

This dissertation outlined two simple, scalable, effective methods for large-scale semi-supervised learning: constructing features from web-scale N-gram data, and using unlabeled data to automatically generate training examples.

The availability of web-scale N-gram data was crucial for our improved web-scale feature-based approaches. While the Google N-gram data was originally created to support the language model of an MT system, we confirmed that this data can be useful for a range of tasks, including both analysis and generation problems. Unlike previous work using search engines, it is possible to extract millions of web-scale counts efficiently from N-gram data. We can thus freely exploit numerous overlapping and interdependent contexts for each example, for both training and test instances. Chapter 3 presented a unified framework for integrating such N-gram information for various lexical disambiguation tasks. Excellent results were achieved on three tasks. In particular, we proposed a novel and successful method of using web-scale counts for the identification of non-referential pronouns, a long-standing challenge in the anaphora resolution community.

In Chapter 4, we introduced a new form of SVM training to mitigate the dependence of the discriminative web-N-gram systems on large amounts of training data. Since the unsupervised system was known to achieve good performance with equal weights, we changed the SVM's regularization to prefer low-weight-variance solutions, biasing it toward the unsupervised solution. The optimization problem remained a convex function of the feature weights, and was thus theoretically no harder to optimize than a standard SVM. On smaller amounts of training data, the variance-regularization SVM performed dramatically better than the standard multi-class SVM.

Chapter 5 addressed a pair of open questions on the use of web-scale data in NLP. First, we showed there was indeed a significant benefit in combining web-scale counts with the traditional features used in state-of-the-art supervised approaches. For example, we proposed a novel system for adjective ordering that exceeds the state-of-the-art performance, without using any N-gram data, and then we further improved the performance of this system by adding N-gram features. Secondly, and perhaps much more importantly, models with web-based features were shown to perform much better than traditional supervised systems when moving to new domains or when labeled training data was scarce (realistic situations for the practical application of NLP technology).

In the second part of the dissertation, we showed how to automatically create labeled ex-

amples from unlabeled data in order to train better models of selectional preference (Chapter 6) and string similarity (Chapter 7). The discriminative classifiers trained from this data exploited several novel sources of information, including character-level (string and capitalization) features for selectional preferences, and features derived from a character-based sequence alignment for discriminative string similarity. While automatic example generation was not applied to web-scale unlabeled data in this work, it promises to scale easily to web-scale text. For example, after summarizing web-scale data with N-gram statistics, we can create examples using only several gigabytes of compressed, N-gram-text, rather than using the petabytes of raw web text directly. So automatic example generation from aggregate statistics promises both better scaling and cleaner data (since aggregate statistics naturally exclude phenomena that occur purely due to chance).

The key methods of parts one and two are, of course, compatible in another way: it would be straightforward to use the output of the pseudo-trained models as features in supervised systems. This is similar to the approach of Ando and Zhang [2005], and, in fact, was pursued in some of our concurrent work [Bergsma *et al.*, 2009a] (with good results).

8.2 The Impact of this Work

We hope the straightforward but effective techniques presented in this dissertation will help promote simple, scalable semi-supervised learning as a future paradigm for NLP research. We advocate such a direction for several reasons.

First, only via machine learning can we combine the millions of parameters that interact in natural language processing. Second, only by leveraging unlabeled data can we go beyond the limited models that can be learned from small, hand-annotated training sets.

Furthermore, it is highly advantageous to have an NLP system that both benefits from unlabeled data and that can readily take advantage of even more unlabeled data when it becomes available. Both the volume of text on the web and the power of computer architecture continue to grow exponentially over time. Systems that use unlabeled data will therefore improve *automatically* over time, without any special annotation, research, or engineering effort. For example, in [Pitler *et al.*, 2010], we presented a parser whose performance improves logarithmically with the number of unique N-grams in a web-scale N-gram corpus. A useful direction for future work would be to identify other problems that can benefit from the use of web-scale volumes of unlabeled data. We could hopefully thereby enable an even greater proportion of NLP systems to achieve automatic improvements in performance.

The following section describes some specific directions for future work, and notes some tasks where web-scale data might be productively exploited.

Once we find out, for a range of tasks, just how far we can get with big data and ML alone, we will have a better handle on what other sources of linguistic knowledge might be needed. For example, we can now get to around 75% accuracy on preposition selection using N-grams alone (Section 3.5). To correct preposition errors with even higher accuracy, we needed to exploit knowledge of the speaker's native language (and thus their likely preposition confusions), getting above 95% accuracy in this manner (but also sacrificing a small but perhaps reasonable amount of coverage). It's unlikely N-gram data alone would ever allow us to select the correct preposition in phrases like, "I like to swim *before/after* school." Similarly, we argued that to perform even better on non-referential pronoun detection (Section 3.7), we will need to pay attention to wider segments of discourse.

8.3 Future Work

This section outlines some specific ways to extend or apply insights from this thesis.

8.3.1 Improved Learning with Automatically-Generated Examples

In part two of this thesis, we achieved good results by automatically generating training examples, but we left open some natural questions arising from this work. For example, how many negatives should be generated for each positive? How do we ensure that training with pseudo-examples transfers well to testing on real examples? While the size of the learning problem prevented extensive experiments at the time the research was originally conducted, recent advances in large-scale machine learning enable much faster training. This allows us to perform large-scale empirical studies to address the above questions. In combination with the usual advances in computer speed and memory, large-scale empirical studies will become even easier. In fact, some have even suggested that large-scale learning of linear classifiers is now essentially a *solved problem* [Yu *et al.*, 2010]. This provides even greater impetus to test and exploit large-scale linear pseudo-classifiers in NLP.

8.3.2 Exploiting New ML Techniques

Another interesting direction for future research will be the development of learning algorithms that exploit correlations between local and global features (see Chapter 1 for an example of local and global features for VBN/VBD disambiguation). Often the local and global patterns represent the same linguistic construction, and their weights should thus be similar. For example, suppose at test time we encounter the phrase, "it was the Bears who won." Even if we haven't seen the pattern, "*noun* who *verb*" as local context in the training set, we may have seen it in the *global* context of a VBD training instance. Laplacian regularization (previously used to exploit the distributional similarity of words for syntactic parsing [Wang *et al.*, 2006]) provides a principled way to force global and local features to have similar weights, although simpler feature-based techniques also exist [Daumé III, 2007]. In particular, combining Laplacian regularization with the scaling of feature *values* (to allow the more predictive, local features to have higher weight) is a promising direction to explore. In any case, identifying an effective solution here could have implications on other, related problems, such as multi-task learning [Raina *et al.*, 2006], domain adaptation [McClosky *et al.*, 2010] and sharing feature knowledge across languages [Berg-Kirkpatrick and Klein, 2010].

8.3.3 New NLP Problems

There are a number of other important, but largely unexplored, NLP problems where web-scale solutions could have an impact. One such problem is the detection of functional relations for information extraction. A functional relation is a binary relation where each element of the domain is related to a unique element in the codomain. For example, each person has a unique birthplace and date of birth, but may have multiple children, residences, and alma maters. There are a number of novel contextual clues that could flag these relations. For example, the indefinite articles *a/an* tend not to occur with functional relations; we frequently observe *a cousin of* in text, but we rarely see *a birthplace of*. The latter is functional. Based on our results in Chapter 5, a classifier combining such simple statistics

with standard lexical features could possibly allow robust functional relation identification across different domains and genres.

8.3.4 Improving Core NLP Technologies

I also plan to apply the web-scale semi-supervised framework to core NLP technologies that are in great demand in the NLP community.

I have previously explored a range of enhancements to pronoun resolution systems [Cherry and Bergsma, 2005; Bergsma, 2005; Bergsma and Lin, 2006; Bergsma *et al.*, 2008b; 2008a; 2009a]. My next step will be to develop and distribute an efficient, state-of-the-art, N-gram-enabled pronoun resolution system for academic and industrial applications. In conversation with colleagues at conferences, I have found that many researchers shy away from machine-learned pronoun resolution systems because of a fear they would not work well on new domains (i.e., the specific domain on which the research is being conducted). By incorporating web-scale statistics into pronoun resolvers, I plan to produce a robust system that people can confidently apply wherever needed.

I will also use web-scale resources to make advances in parsing, the cornerstone technology of NLP. A parser gives the structure of a sentence, identifying who is doing what to whom. Parsing digs deeper into text than typical information retrieval technology, extracting richer levels of knowledge. Companies like Google and Microsoft have recognized the need to access these deeper linguistic structures and are making parsing a focus for their next generation of search engines. I will create an accurate open-domain parser: a domain-independent parser that can reliably analyze any genre of text. A few approaches have successfully adapted a parser to a specific domain, such as general non-fiction [McClosky *et al.*, 2006b] or biomedical text [Rimell and Clark, 2008], but these systems make assumptions that would be unrealistic when parsing text in a heterogeneous collection of web pages, for example. A parser that could reliably process a variety of genres, without manual involvement, would be of great practical and scientific value.

I will create an open-domain parser by essentially adapting to all the text on the web, again building on the robust classifiers presented in Chapter 5. Parsing decisions will be based on observations in web-scale N-gram data, rather than observed (and potentially overly-specific) constructions in a particular domain. Custom algorithms could also be used to extract web-scale knowledge for difficult parsing decisions in coordination, noun compounding, and prepositional phrase attachment. Work in open domain parsing will also require the development of new, cross-domain, task-based evaluations; these could facilitate comparison of parsers based on different formalisms.

I have recently explored methods to both improve the speed of highly-accurate graph-based parsers [Bergsma and Cherry, 2010] (thus allowing the incorporation of new features with less overhead) and ways to incorporate web-scale statistics into the subtask of noun phrase parsing [Pitler *et al.*, 2010]. In preliminary experiments, I have identified a number of other simple N-gram-derived features that improve full-sentence parsing accuracy.

I also plan to investigate whether open-domain parsing could be improved by manually annotating parses of the most frequent N-grams in our new web-scale N-gram corpus (Chapter 5). Recall that the new N-gram corpus includes part-of-speech tags. These tags might help identify N-grams that are likely to be both syntactic constituents and syntactically ambiguous (e.g. noun compounds). The annotation could be done either by experts, or by crowdsourcing annotation via Amazon's Mechanical Turk. A similar technique was recently successfully demonstrated for MT [Bloodgood and Callison-Burch, 2010].

My focus is thus on enabling robust, open-domain systems through better features and new kinds of labeled data. These improvements should combine constructively with recent, orthogonal advances in domain detection and adaptation [McClosky *et al.*, 2010].

8.3.5 Mining New Data Sources

While web-scale N-gram data is very effective, future NLP technology will combine information from a variety of other structured and unstructured data sources to make better natural language inferences. Query logs, parallel bilingual corpora, and collaborative projects like Wikipedia will provide crucial knowledge for syntactic and semantic analysis. For example, there is a tremendous amount of untapped information in the Wikipedia edit histories, which record all the changes made to Wikipedia pages. As a first step in harvesting this information, we could extract a database of real spelling corrections made to Wikipedia pages. This data could be used to train and test NLP spelling correction systems at an unprecedented scale.

Furthermore, it also seems likely that information from the massive volume of online images and video will be used to inform automatic language processing. Many simple statistics can also be computed from visual sources and stored, just like N-gram counts, in precompiled databases. For example, we might extract visual descriptors using algorithms like the popular and efficient SIFT algorithm [Lowe, 1999], convert these descriptors to image codewords (i.e., the bag-of-words representation of images), and then store the codeword co-occurrence counts in a large database.

In fact, services like the Google Image Search and Flickr Photo Sharing websites effectively already link caption words to images in a database. This service could be exploited for building special language models, for example, for selectional preference. When creating features for nouns occurring with particular verbs, for example (as in Chapter 6), we might query the image search service using the noun string as the keyword, and then create SIFT-style features for the retrieved images. Could we build a model, for example, of things that can be eaten, purely based on visual images of edible substances?

In general, I envision some breakthroughs once NLP moves beyond solving text processing in isolation and instead adopts an approach that integrates advances in large-scale processing across a variety of disciplines.

– Thanks for reading the dissertation!

Bibliography

[Adamson and Boreham, 1974] George W. Adamson and Jillian Boreham. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information Storage and Retrieval*, 10, 1974.

[Agichtein and Gravano, 2000] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[Ailon and Mohri, 2008] Nir Ailon and Mehryar Mohri. An efficient reduction of ranking to classification. In *COLT*, 2008.

[Ando and Zhang, 2005] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6, 2005.

[Bahl *et al.*, 1983] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2), 1983.

[Baker, 1975] James K. Baker. The DRAGON system - an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1), 1975.

[Banko and Brill, 2001] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *ACL*, 2001.

[Barr *et al.*, 2008] Cory Barr, Rosie Jones, and Moira Regelson. The linguistic structure of English web-search queries. In *EMNLP*, 2008.

[Barzilay and Lapata, 2006] Regina Barzilay and Mirella Lapata. Aggregation via set partitioning for natural language generation. In *HLT-NAACL*, 2006.

[Bean and Riloff, 2004] David L. Bean and Ellen Riloff. Unsupervised learning of contextual role knowledge for coreference resolution. In *HLT-NAACL*, 2004.

[Belkin *et al.*, 2006] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 7:2399–2434, 2006.

[Bengio *et al.*, 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 2003.

[Berg-Kirkpatrick and Klein, 2010] Taylor Berg-Kirkpatrick and Dan Klein. Phylogenetic grammar induction. In *ACL*, 2010.

[Berger *et al.*, 1996] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 1996.

[Bergsma and Cherry, 2010] Shane Bergsma and Colin Cherry. Fast and accurate arc filtering for dependency parsing. In *COLING*, 2010.

[Bergsma and Kondrak, 2007a] Shane Bergsma and Grzegorz Kondrak. Alignment-based discriminative string similarity. In *ACL*, 2007.

[Bergsma and Kondrak, 2007b] Shane Bergsma and Grzegorz Kondrak. Multilingual cognate identification using integer linear programming. In *RANLP Workshop on Acquisition and Management of Multilingual Lexicons*, 2007.

[Bergsma and Lin, 2006] Shane Bergsma and Dekang Lin. Bootstrapping path-based pronoun resolution. In *COLING-ACL*, 2006.

[Bergsma and Wang, 2007] Shane Bergsma and Qin Iris Wang. Learning noun phrase query segmentation. In *EMNLP-CoNLL*, 2007.

[Bergsma *et al.*, 2008a] Shane Bergsma, Dekang Lin, and Randy Goebel. Discriminative learning of selectional preference from unlabeled text. In *EMNLP*, 2008.

[Bergsma *et al.*, 2008b] Shane Bergsma, Dekang Lin, and Randy Goebel. Distributional identification of non-referential pronouns. In *ACL-08: HLT*, 2008.

[Bergsma *et al.*, 2009a] Shane Bergsma, Dekang Lin, and Randy Goebel. Glen, Glenda or Glendale: Unsupervised and semi-supervised learning of English noun gender. In *CoNLL*, 2009.

[Bergsma *et al.*, 2009b] Shane Bergsma, Dekang Lin, and Randy Goebel. Web-scale N-gram models for lexical disambiguation. In *IJCAI*, 2009.

[Bergsma *et al.*, 2010a] Shane Bergsma, Aditya Bhargava, Hua He, and Grzegorz Kondrak. Predicting the semantic compositionality of prefix verbs. In *EMNLP*, 2010.

[Bergsma *et al.*, 2010b] Shane Bergsma, Dekang Lin, and Dale Schuurmans. Improved natural language learning via variance-regularization support vector machines. In *CoNLL*, 2010.

[Bergsma *et al.*, 2010c] Shane Bergsma, Emily Pitler, and Dekang Lin. Creating robust supervised classifiers via web-scale n-gram data. In *ACL*, 2010.

[Bergsma, 2005] Shane Bergsma. Automatic acquisition of gender information for anaphora resolution. In *Proceedings of the 18th Conference of the Canadian Society for Computational Studies of Intelligence (Canadian AI'2005)*, 2005.

[Bikel, 2004] Daniel M. Bikel. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4), 2004.

[Bilenko and Mooney, 2003] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, 2003.

[Blitzer *et al.*, 2005] John Blitzer, Amir Globerson, and Fernando Pereira. Distributed latent variable models of lexical co-occurrences. In *AISTATS*, 2005.

[Blitzer *et al.*, 2007] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007.

[Bloodgood and Callison-Burch, 2010] Michael Bloodgood and Chris Callison-Burch. Bucking the trend: Large-scale cost-focused active learning for statistical machine translation. In *ACL*, 2010.

[Blum and Mitchell, 1998] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, 1998.

[Brants and Franz, 2006] Thorsten Brants and Alex Franz. The Google Web 1T 5-gram Corpus Version 1.1. LDC2006T13, 2006.

[Brants *et al.*, 2007] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *EMNLP*, 2007.

[Brants, 2000] Thorsten Brants. TnT – a statistical part-of-speech tagger. In *ANLP*, 2000.

[Brill and Moore, 2000] Eric Brill and Robert Moore. An improved error model for noisy channel spelling correction. In *ACL*, 2000.

[Brill *et al.*, 2001] Eric Brill, Jimmy Lin, Michele Banko, Susan Dumais, and Andrew Ng. Data-Intensive Question Answering. In *TREC*, 2001.

[Brin, 1998] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extending Database Technology*, 1998.

[Brockmann and Lapata, 2003] Carsten Brockmann and Mirella Lapata. Evaluating and combining approaches to selectional preference acquisition. In *EACL*, 2003.

[Brown *et al.*, 1992] Peter F. Brown, Vincent J. Della Pietra, Peter V. de Souza, Jennifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4), 1992.

[Brown *et al.*, 1993] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2), 1993.

[Carlson *et al.*, 2001] Andrew J. Carlson, Jeffrey Rosen, and Dan Roth. Scaling up context-sensitive text correction. In *AAAI/IAAI*, 2001.

[Carlson *et al.*, 2008] Andrew Carlson, Tom M. Mitchell, and Ian Fette. Data analysis project: Leveraging massive textual corpora using n-gram statistics. Technial Report CMU-ML-08-107, 2008.

[Chambers and Jurafsky, 2008] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. In *ACL*, 2008.

[Chambers and Jurafsky, 2010] Nathanael Chambers and Dan Jurafsky. Improving the use of pseudo-words for evaluating selectional preferences. In *ACL*, 2010.

[Charniak and Elsner, 2009] Eugene Charniak and Micha Elsner. EM works for pronoun anaphora resolution. In *EACL*, 2009.

[Chen and Goodman, 1998] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. TR-10-98, Harvard University, 1998.

[Cherry and Bergsma, 2005] Colin Cherry and Shane Bergsma. An Expectation Maximization approach to pronoun resolution. In *CoNLL*, 2005.

[Chklovski and Pantel, 2004] Timothy Chklovski and Patrick Pantel. Verbocean: Mining the web for fine-grained semantic verb relations. In *EMNLP*, pages 33–40, 2004.

[Chodorow *et al.*, 2007] Martin Chodorow, Joel R. Tetreault, and Na-Rae Han. Detection of grammatical errors involving prepositions. In *ACL-SIGSEM Workshop on Prepositions*, 2007.

[Chomsky, 1956] Noam Chomsky. Three models for the description of language. *IRI Transactions on Information Theory*, 2(3), 1956.

[Church and Hanks, 1990] Kenneth W. Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1), 1990.

[Church and Mercer, 1993] Kenneth W. Church and Rorbert L. Mercer. Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1), 1993.

[Church and Patil, 1982] Kenneth Church and Ramesh Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics*, 8(3-4):139–149, 1982.

[Church *et al.*, 2007] Kenneth Church, Ted Hart, and Jianfeng Gao. Compressing trigram language models with Golomb coding. In *EMNLP-CoNLL*, 2007.

[Church, 1993] Kenneth W. Church. Char_align: A program for aligning parallel texts at the character level. In *Proceedings of ACL 1993*, 1993.

[Clark and Weir, 2002] Stephen Clark and David Weir. Class-based probability estimation using a semantic hierarchy. *Computational Linguistics*, 28(2), 2002.

[Cohn *et al.*, 1994] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Mach. Learn.*, 15(2):201–221, 1994.

[Collins and Koo, 2005] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1), 2005.

[Collins and Singer, 1999] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *EMNLP-VLC*, 1999.

[Collins, 2002] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

[Cortes and Vapnik, 1995] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[CPLEX, 2005] CPLEX. IBM ILOG CPLEX 9.1. `www.ilog.com/products/cplex/`, 2005.

[Crammer and Singer, 2001] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, 2001.

[Crammer and Singer, 2003] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *JMLR*, 3:951–991, 2003.

[Crammer *et al.*, 2006] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *JMLR*, 7:551–585, 2006.

[Cucerzan and Agichtein, 2005] Silviu Cucerzan and Eugene Agichtein. Factoid Question Answering over Unstructured and Structured Web Content. In *TREC*, 2005.

[Cucerzan and Yarowsky, 1999] Silviu Cucerzan and David Yarowsky. Language independent named entity recognition combining morphological and contextual evidence. In *EMNLP-VLC*, 1999.

[Cucerzan and Yarowsky, 2002] Silviu Cucerzan and David Yarowsky. Augmented mixture models for lexical disambiguation. In *EMNLP*, 2002.

[Cucerzan and Yarowsky, 2003] Silviu Cucerzan and David Yarowsky. Minimally supervised induction of grammatical gender. In *NAACL*, 2003.

[Dagan and Itai, 1990] Ido Dagan and Alan Itai. Automatic processing of large corpora for the resolution of anaphora references. In *COLING*, volume 3, 1990.

[Dagan *et al.*, 1999] Ido Dagan, Lillian Lee, and Fernando C. N. Pereira. Similarity-based models of word cooccurrence probabilities. *Mach. Learn.*, 34(1-3), 1999.

[Daumé III, 2007] Hal Daumé III. Frustratingly easy domain adaptation. In *ACL*, 2007.

[Denis and Baldridge, 2007] Pascal Denis and Jason Baldridge. Joint determination of anaphoricity and coreference using integer programming. In *NAACL-HLT*, 2007.

[Dou *et al.*, 2009] Qing Dou, Shane Bergsma, Sittichai Jiampojamarn, and Grzegorz Kondrak. A ranking approach to stress prediction for letter-to-phoneme conversion. In *ACL-IJCNLP*, 2009.

[Dredze *et al.*, 2008] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *ICML*, 2008.

[Duda and Hart, 1973] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[Erk, 2007] Katrin Erk. A simple, similarity-based model for selectional preference. In *ACL*, 2007.

[Etzioni *et al.*, 2005] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1), 2005.

[Evans, 2001] Richard Evans. Applying machine learning toward an automatic classification of *it*. *Literary and Linguistic Computing*, 16(1), 2001.

[Even-Zohar and Roth, 2000] Yair Even-Zohar and Dan Roth. A classification approach to word prediction. In *NAACL*, 2000.

[Evert, 2004] Stefan Evert. Significance tests for the evaluation of ranking methods. In *COLING*, 2004.

[Fan *et al.*, 2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[Felice and Pulman, 2007] Rachele De Felice and Stephen G. Pulman. Automatically acquiring models of preposition use. In *ACL-SIGSEM Workshop on Prepositions*, 2007.

[Fleischman *et al.*, 2003] Michael Fleischman, Eduard Hovy, and Abdessamad Echihabi. Offline strategies for online question answering: answering questions before they are asked. In *ACL*, 2003.

[Fung and Roth, 2005] Pascale Fung and Dan Roth. Guest editors introduction: Machine learning in speech and language technologies. *Machine Learning*, 60(1-3):5–9, 2005.

[Gale *et al.*, 1992] William A. Gale, Kenneth W. Church, and David Yarowsky. One sense per discourse. In *DARPA Speech and Natural Language Workshop*, 1992.

[Gamon *et al.*, 2008] Michael Gamon, Jianfeng Gao, Chris Brockett, Alexandre Klementiev, William B. Dolan, Dmitriy Belenko, and Lucy Vanderwende. Using contextual speller techniques and language modeling for ESL error correction. In *IJCNLP*, 2008.

[Ge *et al.*, 1998] Niyu Ge, John Hale, and Eugene Charniak. A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.

[Gildea, 2001] Dan Gildea. Corpus variation and parser performance. In *EMNLP*, 2001.

[Golding and Roth, 1999] Andrew R. Golding and Dan Roth. A Winnow-based approach to context-sensitive spelling correction. *Mach. Learn.*, 34(1-3):107–130, 1999.

[Graff, 2003] David Graff. English gigaword. LDC2003T05, 2003.

[Grefenstette, 1999] Gregory Grefenstette. The World Wide Web as a resource for example-based machine translation tasks. In *ASLIB Conference on Translating and the Computer*, 1999.

[Haghighi and Klein, 2006] Aria Haghighi and Dan Klein. Prototype-driven learning for sequence models. In *HLT-NAACL*, 2006.

[Haghighi and Klein, 2010] Aria Haghighi and Dan Klein. Coreference resolution in a modular, entity-centered model. In *HLT-NAACL*, 2010.

[Hajič and Hajičová, 2007] Jan Hajič and Eva Hajičová. Some of our best friends are statisticians. In *TSD*, 2007.

[Har-Peled *et al.*, 2003] Sariel Har-Peled, Dan Roth, and Dav Zimak. Constraint classification for multiclass classification and ranking. In *NIPS*, 2003.

[Harabagiu *et al.*, 2001] Sanda Harabagiu, Razvan Bunescu, and Steven Maiorano. Text and knowledge mining for coreference resolution. In *NAACL*, 2001.

[Harman, 1992] Donna Harman. The DARPA TIPSTER project. *ACM SIGIR Forum*, 26(2), 1992.

[Hawker *et al.*, 2007] Tobias Hawker, Mary Gardiner, and Andrew Bennetts. Practical queries of a massive n-gram database. In *Proc. Australasian Language Technology Association Workshop*, 2007.

[Hearst, 1992] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992.

[Hirst and Budanitsky, 2005] Graeme Hirst and Alexander Budanitsky. Correcting real-word spelling errors by restoring lexical cohesion. *Nat. Lang. Eng.*, 11(1):87–111, 2005.

[Hirst, 1981] Graeme Hirst. *Anaphora in Natural Language Understanding: A Survey*. Springer Verlag, 1981.

[Hobbs, 1978] Jerry Hobbs. Resolving pronoun references. *Lingua*, 44(311), 1978.

[Holmes *et al.*, 1989] Virginia M. Holmes, Laurie Stowe, and Linda Cupples. Lexical expectations in parsing complement-verb sentences. *Journal of Memory and Language*, 28, 1989.

[Hovy *et al.*, 2006] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. OntoNotes: the 90% solution. In *HLT-NAACL*, 2006.

[Hsu and Lin, 2002] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Trans. Neur. Networks*, 13(2):415–425, 2002.

[Huang and Yates, 2009] Fei Huang and Alexander Yates. Distributional representations for handling sparsity in supervised sequence-labeling. In *ACL-IJCNLP*, 2009.

[Jelinek, 1976] Fred Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4), 1976.

[Jelinek, 2005] Frederick Jelinek. Some of my best friends are linguists. *Language Resources and Evaluation*, 39, 2005.

[Jelinek, 2009] Frederick Jelinek. The dawn of statistical ASR and MT. *Comput. Linguist.*, 35(4):483–494, 2009.

[Ji and Lin, 2009] Heng Ji and Dekang Lin. Gender and animacy knowledge discovery from web-scale N-grams for unsupervised person mention detection. In *PACLIC*, 2009.

[Jiampojamarn *et al.*, 2007] Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. Applying many-to-many alignments and hidden Markov models to letter-to-phoneme conversion. In *NAACL-HLT*, 2007.

[Jiampojamarn *et al.*, 2010] Sittichai Jiampojamarn, Ken Dwyer, Shane Bergsma, Aditya Bhargava, Qing Dou, Mi-Young Kim, and Grzegorz Kondrak. Transliteration generation and mining with limited training resources. *Named Entities Workshop (NEWS)*, 2010.

[Joachims *et al.*, 2009] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural SVMs. *Mach. Learn.*, 77(1):27–59, 2009.

[Joachims, 1999a] Thorsten Joachims. Making large-scale Support Vector Machine learning practical. In B. Schölkopf and C. Burges, editors, *Advances in Kernel Methods: Support Vector Machines*. MIT-Press, 1999.

[Joachims, 1999b] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML)*, 1999.

[Joachims, 2002] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[Joachims, 2006] Thorsten Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

[Jones and Ghani, 2000] Rosie Jones and Rayid Ghani. Automatically building a corpus for a minority language from the web. In *Proceedings of the Student Research Workshop at the 38th AnnualMeeting of the Association for Computational Linguistics*, 2000.

[Jurafsky and Martin, 2000] Daniel Jurafsky and James H. Martin. *Speech and language processing*. Prentice Hall, 2000.

[Kehler *et al.*, 2004] Andrew Kehler, Douglas Appelt, Lara Taylor, and Aleksandr Simma. The (non)utility of predicate-argument frequencies for pronoun interpretation. In *HLT-NAACL*, 2004.

[Keller and Lapata, 2003] Frank Keller and Mirella Lapata. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484, 2003.

[Kilgarriff and Grefenstette, 2003] Adam Kilgarriff and Gregory Grefenstette. Introduction to the special issue on the Web as corpus. *Computational Linguistics*, 29(3):333–347, 2003.

[Kilgarriff, 2007] Adam Kilgarriff. Googleology is bad science. *Computational Linguistics*, 33(1), 2007.

[Klementiev and Roth, 2006] Alexandre Klementiev and Dan Roth. Named entity transliteration and discovery from multilingual comparable corpora. In *HLT-NAACL*, 2006.

[Knight *et al.*, 1995] Kevin Knight, Ishwar Chander, Matthew Haines, Vasileios Hatzivassiloglou, Eduard Hovy, Masayo Iida, Steve K. Luk, Richard Whitney, and Kenji Yamada. Filling knowledge gaps in a broad coverage machine translation system. In *IJCAI*, 1995.

[Koehn and Knight, 2002] Philipp Koehn and Kevin Knight. Learning a translation lexicon from monolingual corpora. In *ACL Workshop on Unsupervised Lexical Acquistion*, 2002.

[Koehn and Monz, 2006] Philipp Koehn and Christof Monz. Manual and automatic evaluation of machine translation between European languages. In *NAACL Workshop on Statistical Machine Translation*, 2006.

[Koehn *et al.*, 2003] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *HLT-NAACL*, 2003.

[Koehn, 2005] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Summit X*, 2005.

[Kondrak and Sherif, 2006] Grzegorz Kondrak and Tarek Sherif. Evaluation of several phonetic similarity algorithms on the task of cognate identification. In *COLING-ACL Workshop on Linguistic Distances*, 2006.

[Kondrak *et al.*, 2003] Grzegorz Kondrak, Daniel Marcu, and Kevin Knight. Cognates can improve statistical translation models. In *HLT-NAACL*, 2003.

[Kondrak, 2005] Grzegorz Kondrak. Cognates and word alignment in bitexts. In *MT Summit X*, 2005.

[Koo *et al.*, 2008] Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *ACL-08: HLT*, 2008.

[Kotsia *et al.*, 2009] Irene Kotsia, Stefanos Zafeiriou, and Ioannis Pitas. Novel multiclass classifiers based on the minimization of the within-class variance. *IEEE Trans. Neur. Networks*, 20(1):14–34, 2009.

[Kulick *et al.*, 2004] Seth Kulick, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, Lyle Ungar, Scott Winters, and Pete White. Integrated annotation for biomedical information extraction. In *BioLINK 2004: Linking Biological Literature, Ontologies and Databases*, 2004.

[Kummerfeld and Curran, 2008] Jonathan K. Kummerfeld and James R. Curran. Classification of verb particle constructions with the google web1t corpus. In *Australasian Language Technology Association Workshop*, 2008.

[Lafferty *et al.*, 2001] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[Lapata and Keller, 2005] Mirella Lapata and Frank Keller. Web-based models for natural language processing. *ACM Trans. Speech and Language Processing*, 2(1):1–31, 2005.

[Lappin and Leass, 1994] Shalom Lappin and Herbert J. Leass. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4), 1994.

[Lauer, 1995a] Mark Lauer. Corpus statistics meet the noun compound: Some empirical results. In *ACL*, 1995.

[Lauer, 1995b] Mark Lauer. *Designing Statistical Language Learners: Experiments on Compound Nouns*. PhD thesis, Macquarie University, 1995.

[Levenshtein, 1966] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8), 1966.

[Li and Abe, 1998] Hang Li and Naoki Abe. Generalizing case frames using a thesaurus and the MDL principle. *Computational Linguistics*, 24(2), 1998.

[Lin and Wu, 2009] Dekang Lin and Xiaoyun Wu. Phrase clustering for discriminative learning. In *ACL-IJCNLP*, 2009.

[Lin *et al.*, 2010] Dekang Lin, Kenneth Church, Heng Ji, Satoshi Sekine, David Yarowsky, Shane Bergsma, Kailash Patil, Emily Pitler, Rachel Lathbury, Vikram Rao, Kapil Dalwani, and Sushant Narsale. New tools for web-scale N-grams. In *LREC*, 2010.

[Lin, 1998a] Dekang Lin. Automatic retrieval and clustering of similar words. In *COLING-ACL*, 1998.

[Lin, 1998b] Dekang Lin. Dependency-based evaluation of MINIPAR. In *LREC Workshop on the Evaluation of Parsing Systems*, 1998.

[Litkowski and Hargraves, 2007] Ken Litkowski and Orin Hargraves. SemEval-2007 Task 06: Word-sense disambiguation of prepositions. In *SemEval*, 2007.

[Liu and Curran, 2006] Vinci Liu and James R. Curran. Web text corpus for natural language processing. In *EACL*, 2006.

[Lodhi *et al.*, 2002] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *JMLR*, 2:419–444, 2002.

[Lowe, 1999] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[Malouf, 2000] Robert Malouf. The order of prenominal adjectives in natural language generation. In *ACL*, 2000.

[Mann and McCallum, 2007] Gideon S. Mann and Andrew McCallum. Simple, robust, scalable semi-supervised learning via expectation regularization. In *ICML*, 2007.

[Mann and Yarowsky, 2001] Gideon S. Mann and David Yarowsky. Multipath translation lexicon induction via bridge languages. In *NAACL*, 2001.

[Manning and Schütze, 1999] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[Marcus *et al.*, 1993] Mitchell P. Marcus, Beatrice Santorini, and Mary Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[Marcus, 1980] Mitchell P. Marcus. *Theory of Syntactic Recognition for Natural Languages*. MIT Press, Cambridge, MA, USA, 1980.

[Marton *et al.*, 2009] Yuval Marton, Chris Callison-Burch, and Philip Resnik. Improved statistical machine translation using monolingually-derived paraphrases. In *ACL-IJCNLP*, 2009.

[McCallum *et al.*, 2005] Andrew McCallum, Kedar Bellare, and Fernando Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In *UAI*, 2005.

[McCallum, 1996] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/~mccallum/bow, 1996.

[McClosky *et al.*, 2006a] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *HLT-NAACL*, 2006.

[McClosky *et al.*, 2006b] David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *COLING-ACL*, 2006.

[McClosky *et al.*, 2010] David McClosky, Eugene Charniak, and Mark Johnson. Automatic domain adaptation for parsing. In *NAACL HLT*, 2010.

[McEnery and Oakes, 1996] Tony McEnery and Michael P. Oakes. Sentence and word alignment in the CRATER project. In *Using Corpora for Language Research*. Longman, 1996.

[Melamed, 1998] I. Dan Melamed. Manual annotation of translational equivalence. Technical Report IRCS #98-07, University of Pennsylvania, 1998.

[Melamed, 1999] I. Dan Melamed. Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25(1), 1999.

[Mihalcea and Moldovan, 1999] Rada Mihalcea and Dan I. Moldovan. A method for word sense disambiguation of unrestricted text. In *ACL*, 1999.

[Miller *et al.*, 1990] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4), 1990.

[Miller *et al.*, 2004] Scott Miller, Jethran Guinness, and Alex Zamanian. Name tagging with word clusters and discriminative training. In *HLT-NAACL*, 2004.

[Mitchell, 2009] Margaret Mitchell. Class-based ordering of prenominal modifiers. In *12th European Workshop on Natural Language Generation*, 2009.

[Modjeska *et al.*, 2003] Natalia N. Modjeska, Katja Markert, and Malvina Nissim. Using the Web in machine learning for *other*-anaphora resolution. In *EMNLP*, 2003.

[MUC-7, 1997] MUC-7. Coreference task definition (v3.0, 13 Jul 97). In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1997.

[Müller *et al.*, 2002] Christoph Müller, Stefan Rapp, and Michael Strube. Applying co-training to reference resolution. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

[Müller, 2006] Christoph Müller. Automatic detection of nonreferential *It* in spoken multi-party dialog. In *EACL*, 2006.

[Mulloni and Pekar, 2006] Andrea Mulloni and Viktor Pekar. Automatic detection of orthographic cues for cognate recognition. In *LREC*, 2006.

[Munteanu and Marcu, 2005] Dragos S. Munteanu and Daniel Marcu. Improving machine translation performance by exploiting non-parallel corpora. *Computational Linguistics*, 31(4):477–504, 2005.

[Nakov and Hearst, 2005a] Preslav Nakov and Marti Hearst. Search engine statistics beyond the n-gram: Application to noun compound bracketing. In *CoNLL*, 2005.

[Nakov and Hearst, 2005b] Preslav Nakov and Marti Hearst. Using the web as an implicit training set: application to structural ambiguity resolution. In *HLT/EMNLP*, 2005.

[Nakov, 2007] Preslav Ivanov Nakov. *Using the Web as an Implicit Training Set: Application to Noun Compound Syntax and Semantics*. PhD thesis, University of California, Berkeley, 2007.

[Ng and Cardie, 2003a] Vincent Ng and Claire Cardie. Bootstrapping coreference classifiers with multiple machine learning algorithms. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2003.

[Ng and Cardie, 2003b] Vincent Ng and Claire Cardie. Weakly supervised natural language learning without redundant views. In *Proceedings of the HLT-NAACL*, 2003.

[Ng and Jordan, 2002] Andrew Y. Ng and Michael I. Jordan. Discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2002.

[Och and Ney, 2002] Franz J. Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *ACL*, 2002.

[Okanohara and Tsujii, 2007] Daisuke Okanohara and Jun'ichi Tsujii. A discriminative language model with pseudo-negative samples. In *ACL*, 2007.

[Paice and Husk, 1987] Chris D. Paice and Gareth D. Husk. Towards the automatic recognition of anaphoric features in English text: the impersonal pronoun "it". *Computer Speech and Language*, 2:109–132, 1987.

[Pantel and Lin, 2002] Patrick Pantel and Dekang Lin. Discovering word senses from text. In *KDD*, 2002.

[Pantel and Pennacchiotti, 2006] Patrick Pantel and Marco Pennacchiotti. Espresso: leveraging generic patterns for automatically harvesting semantic relations. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, 2006.

[Pantel *et al.*, 2007] Patrick Pantel, Rahul Bhagat, Bonaventura Coppola, Timothy Chklovski, and Eduard Hovy. ISP: Learning inferential selectional preferences. In *NAACL-HLT*, 2007.

[Pantel, 2003] Patrick Pantel. *Clustering by committee*. PhD thesis, University of Alberta, 2003.

[Paşca *et al.*, 2006] Marius Paşca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Names and similarities on the Web: Fact extraction in the fast lane. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, 2006.

[Phan, 2006] Xuan-Hieu Phan. CRFTagger: CRF English POS Tagger. `crftagger.sourceforge.net`, 2006.

[Pinchak and Bergsma, 2007] Christopher Pinchak and Shane Bergsma. Automatic answer typing for how-questions. In *HLT-NAACL*, 2007.

[Pitler *et al.*, 2010] Emily Pitler, Shane Bergsma, Dekang Lin, and Kenneth Church. Using web-scale N-grams to improve base NP parsing performance. In *COLING*, 2010.

[Porter, 1980] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3), 1980.

[Radev *et al.*, 2001] Dragomir R. Radev, Hong Qi, Zhiping Zheng, Sasha Blair-Goldensohn, Zhu Zhang, Weiguo Fan, and John Prager. Mining the Web for Answers to Natural Language Questions. In *CIKM*, 2001.

[Raina *et al.*, 2006] Rajat Raina, Andrew Y. Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *ICML*, 2006.

[Rappoport and Levent-Levi, 2006] Ari Rappoport and Tsahi Levent-Levi. Induction of cross-language affix and letter sequence correspondence. In *EACL Workshop on Cross-Language Knowledge Induction*, 2006.

[Ravichandran and Hovy, 2002] Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002.

[Resnik, 1996] Philip Resnik. Selectional constraints: An information-theoretic model and its computational realization. *Cognition*, 61, 1996.

[Resnik, 1999] Philip Resnik. Mining the web for bilingual text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 1999.

[Rifkin and Klautau, 2004] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *JMLR*, 5:101–141, 2004.

[Riloff and Jones, 1999] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[Rimell and Clark, 2008] Laura Rimell and Stephen Clark. Adapting a lexicalized-grammar parser to contrasting domains. In *EMNLP*, 2008.

[Ristad and Yianilos, 1998] Eric Sven Ristad and Peter N. Yianilos. Learning string-edit distance. *IEEE Trans. Pattern Anal. Machine Intell.*, 20(5), 1998.

[Roberto *et al.*, 2007] Basili Roberto, Diego De Cao, Paolo Marocco, and Marco Pennacchiotti. Learning selectional preferences for entailment or paraphrasing rules. In *RANLP*, 2007.

[Rooth *et al.*, 1999] Mats Rooth, Stefan Riezler, Detlef Prescher, Glenn Carroll, and Franz Beil. Inducing a semantically annotated lexicon via EM-based clustering. In *ACL*, 1999.

[Roth and Yih, 2004] Dan Roth and Wen-Tau Yih. A linear programming formulation for global inference in natural language tasks. In *CoNLL*, 2004.

[Roth, 1998] Dan Roth. Learning to resolve natural language ambiguities: A unified approach. In *AAAI/IAAI*, 1998.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*, chapter 20: Statistical Learning Methods. Prentice Hall, Upper Saddle River, N.J., 2nd edition edition, 2003.

[Schafer and Yarowsky, 2002] Charles Schafer and David Yarowsky. Inducing translation lexicons via diverse similarity measures and bridge languages. In *CoNLL*, 2002.

[Sekine, 2008] Satoshi Sekine. A linguistic knowledge discovery tool: Very large ngram database search with arbitrary wildcards. In *COLING: Companion volume: Demonstrations*, 2008.

[Shannon, 1948] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 1948.

[Shaw and Hatzivassiloglou, 1999] James Shaw and Vasileios Hatzivassiloglou. Ordering among premodifiers. In *ACL*, 1999.

[Simard *et al.*, 1992] Michel Simard, George F. Foster, and Pierre Isabelle. Using cognates to align sentences in bilingual corpora. In *Fourth International Conference on Theoretical and Methodological Issues in Machine Translation*, 1992.

[Smith and Eisner, 2005] Noah A. Smith and Jason Eisner. Contrastive estimation: training log-linear models on unlabeled data. In *ACL*, 2005.

[Snow *et al.*, 2005] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*, 2005.

[Snow *et al.*, 2008] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, 2008.

[Steedman, 2008] Mark Steedman. On becoming a discipline. *Comput. Linguist.*, 34(1):137–144, 2008.

[Strube *et al.*, 2002] Michael Strube, Stefan Rapp, and Christoph Müller. The influence of minimum edit distance on reference resolution. In *EMNLP*, 2002.

[Suzuki and Isozaki, 2008] Jun Suzuki and Hideki Isozaki. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In *Proceedings of ACL-08: HLT*, 2008.

[Taskar *et al.*, 2005] Ben Taskar, Simon Lacoste-Julien, and Dan Klein. A discriminative matching approach to word alignment. In *HLT-EMNLP*, 2005.

[Tefas *et al.*, 2001] Anastasios Tefas, Constantine Kotropoulos, and Ioannis Pitas. Using support vector machines to enhance the performance of elastic graph matching for frontal face authentication. *IEEE Trans. Pattern Anal. Machine Intell.*, 23:735–746, 2001.

[Tetreault and Chodorow, 2008] Joel R. Tetreault and Martin Chodorow. The ups and downs of preposition error detection in ESL writing. In *COLING*, 2008.

[Tiedemann, 1999] Jörg Tiedemann. Automatic construction of weighted string similarity measures. In *EMNLP-VLC*, 1999.

[Tong and Koller, 2002] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *JMLR*, 2:45–66, 2002.

[Tratz and Hovy, 2010] Stephen Tratz and Eduard Hovy. A taxonomy, dataset, and classifier for automatic noun compound interpretation. In *ACL*, 2010.

[Tsochantaridis *et al.*, 2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

[Tsochantaridis *et al.*, 2005] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, 2005.

[Tsuruoka *et al.*, 2005] Yoshimasa Tsuruoka, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou, and Jun'ichi Tsujii. Developing a robust part-of-speech tagger for biomedical text. In *Advances in Informatics*, 2005.

[Turian *et al.*, 2010] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *ACL*, 2010.

[Turney, 2002] Peter D. Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *ACL*, 2002.

[Turney, 2003] Peter D. Turney. Coherent keyphrase extraction via web mining. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), (2003), Acapulco, Mexico*, 2003.

[Turney, 2006] Peter D. Turney. Similarity of semantic relations. *Computational Linguistics*, 32(3):379–416, 2006.

[Uitdenbogerd, 2005] Sandra Uitdenbogerd. Readability of French as a foreign language and its uses. In *Proceedings of the Australian Document Computing Symposium*, 2005.

[Vadas and Curran, 2007a] David Vadas and James R. Curran. Adding noun phrase structure to the Penn Treebank. In *ACL*, 2007.

[Vadas and Curran, 2007b] David Vadas and James R. Curran. Large-scale supervised models for noun phrase bracketing. In *PACLING*, 2007.

[van den Bosch, 2006] Antal van den Bosch. All-word prediction as the ultimate confusable disambiguation. In *Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, 2006.

[Vapnik, 1998] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[Vorhees, 2002] Ellen Vorhees. Overview of the TREC 2002 question answering track. In *Proceedings of the Eleventh Text REtrieval Conference (TREC)*, 2002.

[Wang *et al.*, 2005] Qin Iris Wang, Dale Schuurmans, and Dekang Lin. Strictly lexical dependency parsing. In *International Workshop on Parsing Technologies*, 2005.

[Wang *et al.*, 2006] Qin Iris Wang, Colin Cherry, Dan Lizotte, and Dale Schuurmans. Improved large margin dependency parsing via local constraints and Laplacian regularization. In *CoNLL*, 2006.

[Wang *et al.*, 2008] Qin Iris Wang, Dale Schuurmans, and Dekang Lin. Semi-supervised convex training for dependency parsing. In *ACL-08: HLT*, 2008.

[Weeds and Weir, 2005] Julie Weeds and David Weir. Co-occurrence retrieval: a flexible framework for lexical distributional similarity. *Computational Linguistics*, 31(4), 2005.

[Weston and Watkins, 1998] Jason Weston and Chris Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, 1998.

[Wilcox-O'Hearn *et al.*, 2008] Amber Wilcox-O'Hearn, Graeme Hirst, and Alexander Budanitsky. Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model. In *CICLing*, 2008.

[Witten and Frank, 2005] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, second edition, 2005.

[Xu *et al.*, 2009] Peng Xu, Jaeho Kang, Michael Ringgaard, and Franz Josef Och. Using a dependency parser to improve SMT for subject-object-verb languages. In *HLT-NAACL*, 2009.

[Yang *et al.*, 2005] Xiaofeng Yang, Jian Su, and Chew Lim Tan. Improving pronoun resolution using statistics-based semantic compatibility information. In *ACL*, 2005.

[Yarowsky, 1994] David Yarowsky. Decision lists for lexical ambiguity resolution: application to accent restoration in Spanish and French. In *ACL*, 1994.

[Yarowsky, 1995] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*, 1995.

[Yi *et al.*, 2008] Xing Yi, Jianfeng Gao, and William B. Dolan. A web-based English proofing system for English as a second language users. In *IJCNLP*, 2008.

[Yoon *et al.*, 2007] Su-Youn Yoon, Kyoung-Young Kim, and Richard Sproat. Multilingual transliteration using feature based phonetic method. In *ACL*, pages 112–119, 2007.

[Yu *et al.*, 2007] Liang-Chih Yu, Chung-Hsien Wu, Andrew Philpot, and Eduard Hovy. OntoNotes: Sense pool verification using Google N-gram and statistical tests. In *OntoLex Workshop at the 6th International Semantic Web Conference (ISWC'07)*, 2007.

[Yu *et al.*, 2010] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. In *KDD*, 2010.

[Yuret, 2007] Deniz Yuret. KU: Word sense disambiguation by substitution. In *SemEval-2007: 4th International Workshop on Semantic Evaluations*, June 2007.

[Zaidan *et al.*, 2007] Omar Zaidan, Jason Eisner, and Christine Piatko. Using "annotator rationales" to improve machine learning for text categorization. In *NAACL-HLT*, 2007.

[Zelenko and Aone, 2006] Dmitry Zelenko and Chinatsu Aone. Discriminative methods for transliteration. In *EMNLP*, 2006.

[Zhu, 2005] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

Appendix A

Penn Treebank Tag Set

Tag	Description	Examples
$	dollar	$ -$ –$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
“	opening quotation mark	‘ “
”	closing quotation mark	’ ”
(	opening parenthesis	([{
)	closing parenthesis	)] }
,	comma	,
–	dash	–
.	sentence terminator	. ! ?
:	colon or ellipsis	: ; ...
CC	conjunction, coordinating	’n and both but either et for less minus neither nor or plus so therefore times v. versus vs. whether yet
CD	numeral, cardinal	mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-seven 1987 twenty ’79 zero two 78-degrees eighty-four IX ’60s .025 fifteen 271,124 dozen quintillion DM2,000 ...
DT	determiner	all an another any both del each either every half la many much nary neither no some such that the them these this those
EX	existential there	there
FW	foreign word	gemeinschaft hund ich jeux habeas Haementeria Herr K’ang-si vous lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte terram fiche oui corporis ...
IN	preposition or conjunction, subordinating	astride among uppon whether out inside pro despite on by throughout below within for towards near behind atop around if like until below next into if beside ...
JJ	adjective or numeral, ordinal	third ill-mannered pre-war regrettable oiled calamitous first separable ectoplasmic battery-powered participatory fourth still-to-be-named multilingual multi-disciplinary ...
JJR	adjective, comparative	bleaker braver breezier briefer brighter brisker broader bumper busier calmer cheaper choosier cleaner clearer closer colder commoner costlier cozier creamier crunchier cuter ...
JJS	adjective, superlative	calmest cheapest choicest classiest cleanest clearest closest commonest corniest costliest crassest creepiest crudest cutest darkest deadliest dearest deepest densest dinkiest ...
LS	list item marker	A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005 SP-44007 Second Third Three Two a b c d first five four one six three two
MD	modal auxiliary	can cannot could couldn’t dare may might must need ought shall should shouldn’t will would
NN	noun, common, singular or mass	common-carrier cabbage knuckle-duster Casino afghan shed thermostat investment slide humour falloff slick wind hyena override subhumanity machinist ...

From: `http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html`

NNP	noun, proper, singular	Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA Shannon A.K.C. Meltex Liverpool ...
NNPS	noun, proper, plural	Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists Andalusians Andes Andruses Angels Animals Anthony Antilles Antiques Apache Apaches Apocrypha ...
NNS	noun, common, plural	undergraduates scotches bric-a-brac products bodyguards facets coasts divestitures storehouses designs clubs fragrances averages subjectivists apprehensions muses factory-jobs ...
PDT	pre-determiner	all both half many quite such sure this
POS	genitive marker	’ ’s
PRP	pronoun, personal	hers herself him himself hisself it itself me myself one oneself ours ourselves ownself self she thee theirs them themselves they thou thy us
PRP$	pronoun, possessive	her his mine my our ours their thy your
RB	adverb	occasionally unabatingly maddeningly adventurously professedly stirringly prominently technologically magisterially predominately swiftly fiscally pitilessly ...
RBR	adverb, comparative	further gloomier grander graver greater grimmer harder harsher healthier heavier higher however larger later leaner lengthier less-perfectly lesser lonelier longer louder lower more ...
RBS	adverb, superlative	best biggest bluntest earliest farthest first furthest hardest heartiest highest largest least less most nearest second tightest worst
RP	particle	aboard about across along apart around aside at away back before behind by crop down ever fast for forth from go high i.e. in into just later low more off on open out over per pie raising start teeth that through under unto up up-pp upon whole with you
SYM	symbol	% & ’ ” ”.)). * + ,. ¡ = ¿ A[fj] U.S U.S.S.R
TO	“to” as preposition or infinitive marker	to
UH	interjection	Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly man baby diddle hush sonuvabitch ...
VB	verb, base form	ask assemble assess assign assume atone attention avoid bake balkanize bank begin behold believe bend benefit bevel beware bless boil bomb boost brace break bring broil brush build ...
VBD	verb, past tense	dipped pleaded swiped regummed soaked tidied convened halted registered cushioned exacted snubbed strode aimed adopted belied figgered speculated wore appreciated contemplated ...
VBG	verb, present participle or gerund	telegraphing stirring focusing angering judging stalling lactating hankerin’ alleging veering capping approaching traveling besieging encrypting interrupting erasing wincing ...
VBN	verb, past participle	multihulled dilapidated aerosolized chaired languished panelized used experimented flourished imitated reunifed factored condensed sheared unsettled primed dubbed desired ...
VBP	verb, present tense, not 3rd person singular	predominate wrap resort sue twist spill cure lengthen brush terminate appear tend stray glisten obtain comprise detest tease attract emphasize mold postpone sever return wag ...
VBZ	verb, present tense, 3rd person singular	bases reconstructs marks mixes displeases seals carps weaves snatches slumps stretches authorizes smolders pictures emerges stockpiles seduces fizzes uses bolsters slaps speaks pleads ...
WDT	WH-determiner	that what whatever which whichever
WP	WH-pronoun	that what whatever whatsoever which who whom whosoever
WP$	WH-pronoun, possessive	whose
WRB	Wh-adverb	how however whence whenever where whereby whereever wherein whereof why