EMPIRICAL METHODS FOR FINE-GRAINED OPINION EXTRACTION FROM TEXT

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Eric John Breck

August 2008

EMPIRICAL METHODS FOR FINE-GRAINED OPINION EXTRACTION FROM TEXT

Eric John Breck, Ph.D.

Cornell University 2008

Opinions are everywhere. The op/ed pages of newspapers, political blogs, and consumer websites like epinions.com are just some examples of the textual opinions available to readers. And there are many consumers who are interested in following these opinions – intelligence analysts who track the opinions of foreign countries, public relation firms who want to ensure positive opinions for their clients, pollsters who want to know the public's opinions about politicians, and companies who want to know customers' opinions about their products. The problem faced by all of these consumers of opinion is that there is such a wealth of text to process that it is hard to read it all. Central to processing the opinions in these text will be solving two specific problems - identifying expressions of opinion, and identifying their hierarchical structure. We demonstrate solutions involving empirical natural language processing techniques.

Although empirical, data-driven methods such as these have become the norm in natural language processing, little work has been done in analyzing their impact on the reproducibility, efficiency, and effectiveness of research. We address two specific problems in this area. We introduce a lightweight computational workflow system to improve the reproducibility and efficiency of machine learning and natural language processing experiments. And we investigate the process of feature generation, setting out desiderata for an ideal process and exploring the effectiveness of several alternatives. Both are investigated in the context of the natural language learning tasks set out earlier.

BIOGRAPHICAL SKETCH

Eric Breck received a Bachelor of Science degree in Mathematics and Linguistics from the University of Michigan. He then worked at the MITRE Corporation in the Intelligent Information Access group for three years. After that, he came to Cornell to pursue this PhD in Computer Science. While at Cornell, he served as an intern in the Natural Language Processing group at Microsoft Research.

This thesis is dedicated to James & Sandra Breck, the best parents anyone could possibly have.

ACKNOWLEDGMENTS

Claire Cardie has been a fantastically supportive mentor over the past nearly seven years. She has cheered me on through successful and unsuccessful projects and taught me about science and writing and what it means to be a researcher. I am glad to have produced work to pass Lillian Lee's keen analytical standards. Mats Rooth has tried to keep me linguistically honest. And I am grateful for Charlie Van Loan's presence and for his leadership as our department chair through most of my term at Cornell.

I've also greatly benefited from the wisdom and expertise of Thorsten Joachims and Rich Caruana. The students in the Cornell natural language processing group and the machine learning discussion group have been valued colleagues. Marc Light and Lynette Hirschman were my first mentors in computational linguistics, and they and the other folks at MITRE provided a wonderful introduction to life as a researcher and to the field of NLP.

John Lawler taught the first linguistics class I ever took, and stoked the fires of my interest in language. I still use some of the anecdotes he told in that class today when I try to share my excitement. Bill Harris and Rick Bednarz taught me calculus and geometry in high school and middle school, but more than that they showed me how much fun mathematics can be.

A couple of students have been especially helpful to me in completing this process. Dan Grossman offered his just-a-bit-ahead experience, providing sage advice to me as a young student. Sharon Goldwater walked a parallel path to mine and we shared our different takes on similar programs.

Graduate school is about more than research, and many students have helped make this experience richer, more productive, and more fun. The Books-n-Cooks book club forced me to read some fantastic (and some terrible) litera-

ture, as well as providing a wide sampling of delectable cuisine. The Cornell Chordials welcomed me into a world entirely distinct from my usual one, filled with music and laughter and stories. And there's really nothing like the experience I had with them of hearing our name called and rushing onstage for a victory encore, or standing in Town Hall and getting chills from the performance of my fellow singers. Thanks to Joss Whedon and Rob Thomas, Amy, Alexa, Dave, Mohan, Riccardo, Rif, Tom, Vicky, and I spent many nights enjoying the exploits of Buffy and Veronica. And over all the years, I've been glad to share an office with Alex, Filip, Jeff, Matt, and Steve, as we weathered the ups and downs of grad school together.

Amy, Cynthia, Gretchen, Janine, Jasmine, Justin, Karen, Kyla, Michelle, Sarah, Sean, and Will have all enriched my life in ways too numerous to count. Emily has brought me joy I never knew. My brother Jason knows everything I do and more, despite having had six fewer years to learn it all in — he's amazing and I'm so proud to be his big brother. And I owe everything I am and have to my ever-loving parents, Jim and Sandy.

This work was supported in part by the Advanced Research and Development Activity (ARDA), by NSF Grants IIS-0535099 and IIS-0208028, by Department of Homeland Security Grant N0014-07-1-0152, by gifts from Google and the Xerox Foundation, by an NSF Graduate Research Fellowship, and by a Cornell Cognitive Studies Fellowship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AIC	Akaike Information Criterion, *see page 105*
BIC	Bayes Information Criterion, *see page 105*
CASS	A partial parser developed by Steve Abney, *see page 28*
CRF	Conditional Random Field, *see page 42*
DSE	Direct Subjective Expression, *see page 22*
DSESE	Direct Subjective and Speech Expression, *see page 23*
ESE	Expressive subjective element, *see page 23*
F-measure	The harmonic mean of precision and recall. Sometimes referred to in the literature as F1 or F-score, *see page 30*
GATE	General Architecture for Text Engineering, *see page 27*
IIC	Information Investment Criterion, *see page 105*
IND	A decision tree package developed by Wray Buntine, *see page 60*
IOB	In / Out / Begin, an encoding of extent tagging as per-word tagging. Sometimes called BIO., *see page 41*
MALLET	Machine Learning for Language Toolkit, *see page 42*
MPQA	Multi-Perspective Question Answering - a workshop and resulting annotated corpus, *see page 29*
NLP	Natural Language Processing, *see page 1*
OSE	Objective Speech Expression, *see page 23*

PSE Potentially Subjective Element, *see page 17*

RIC Risk Inflation Criterion, *see page 105*

SE Speech Expression, *see page 23*

SQL Structured Query Language, *see page 96*

SVM Support Vector Machine, *see page 31*

XML eXtensible Markup Language, *see page 92*

CHAPTER 1

INTRODUCTION

There are several reasons for pursuing research into natural language processing (NLP). One is that for those who enjoy puzzles, language is a rich source of fun problems to solve[1]. Another is that since everyone speaks a language, everyone can relate to the issues that arise in researching it. Moreover, language is a pillar of cognition, lending hope that language research will advance our understanding of mind, and natural language processing research will advance our understanding of artificial intelligence.

There are compelling practical motivations for studying natural language processing as well. The ever-more-technological world abounds with linguistic problems begging for practical solutions. Machine translation promises to break down communication barriers among the people of the world(MTX, 2007). Automatic question answering could provide straightforward access to the wealth of information now avalable online (Voorhees, 1999). Summarization may allow a reader to get the gist of far more information than they have time to read (DUC, 2007).

Most research in these areas has been focused on analyzing factual content. Recently, however, there has been increasing interest in going beyond facts to subjective, opinionated content.

[1]The North American Computational Linguistics Olympiad, organized since 2007, provides quite a variety of such puzzles for the interested reader.

1.1 Opinion analysis

There are several reasons why studying the automatic analysis of opinionated text is important. First of all, there are many users who would potentially benefit. Second, there is an ever-growing amount of data available to process. And third, the tasks are different in a number of respects from the factual text analysis that has been common in the field up until recently.

Users Intelligence analysts are interested in the opinions of foreign governments, and of other entities that might affect them (e.g., terrorists). Public relations firms want to make sure the public and other businesses have a positive opinion of their clients. Pollsters are interested in the opinions of citizens about issues and candidates. Companies want to know what their customers think of their products. Finally, individuals are interested in the opinions of other people about the products and issues they care about, for help in making decisions

Data Newspapers have op/ed pages, with opinionated content from anonymous editors and recurring columnists, in addition to often opinionated letters from their readers. Blogs offer opinions on everything from politics to games to hobbies. Many retailers such as Amazon.com offer their customers the opportunity to comment on the products they buy. This data offers rich, interesting phenomena, such as metaphor, idioms, and other creative uses of language.

Non-factual Researchers have found that the techniques developed for extracting purely factual information serve as a starting point for analyzing opinions, but that opinions present unique challenges. To take just one recent ex-

ample, Pang and Lee (2004) find that while in news summarization using the first n sentences is a good baseline, on a movie-review domain using the last n sentences performs much better.

This dissertation focuses on key problems in opinion analysis that will be necessary for creating applications for any of the above users. While the domain evaluated is news text, the techniques applied should largely carry over to other domains. Finally, since the problems discussed here have not previously been directly studied, we introduce novel problem encodings and feature sets.

1.1.1 Opinion-oriented information extraction

Consider the following sentences:

1. Philip Clapp, president of the National Environment Trust, sums up well the general thrust of the reaction of environmental movements: "There is no reason at all to believe that the polluters are suddenly going to become reasonable."
2. John McCain will be the 2008 Republican nominee for President.
3. *Harold and Kumar go to White Castle* is one of the finest films in American cinema.

The prevailing tasks in the natural language processing literature involve answering questions about opinions.

- *Is this sentence positive, negative, or neutral?* Sentence 3 expresses a positive opinion about the movie. Sentence 1 is in general negative. Sentence 2 is neutral.

- *Is this sentence subjective or objective?* Sentence 2 reports an objective fact, while the other two sentences contain subjectivity.

These questions may also be asked of larger or smaller linguistic units – for example, is a particular movie review positive or negative, or is a particular word positive or negative? These questions are important and useful and much work has gone into being able to answer them. However, the work we will discuss in this thesis aims to answer a wider variety of questions.

We are interested in fine-grained extraction of opinions. As such, we want to be able to answer questions such as the following.

- *Who is it that holds a given opinion?* Sentence 1 presents an opinion held by the environmental movement. Sentence 3 presents an opinion held by the author of the sentence.
- *What is the opinion about?* Sentence 1 presents an opinion about polluters, and Sentence 3 presents an opinion about a movie.
- *How strong is the opinion?* Sentence 3 presents a quite strong opinion, while the opinion in Sentence 1 is milder.
- *How is the opinion filtered?* Sentence 3 presents an opinion directly to us by the author of the sentence. Sentence 1 presents the opinion of the environmental movements, but only as it is reported to us via Philip Clapp, and then by the writer of the sentence.

The eventual goal of our research is to be able to answer all of these questions about any sentence. There are a number of potential uses for such a system. One is simply a question answering system, like those described earlier, that

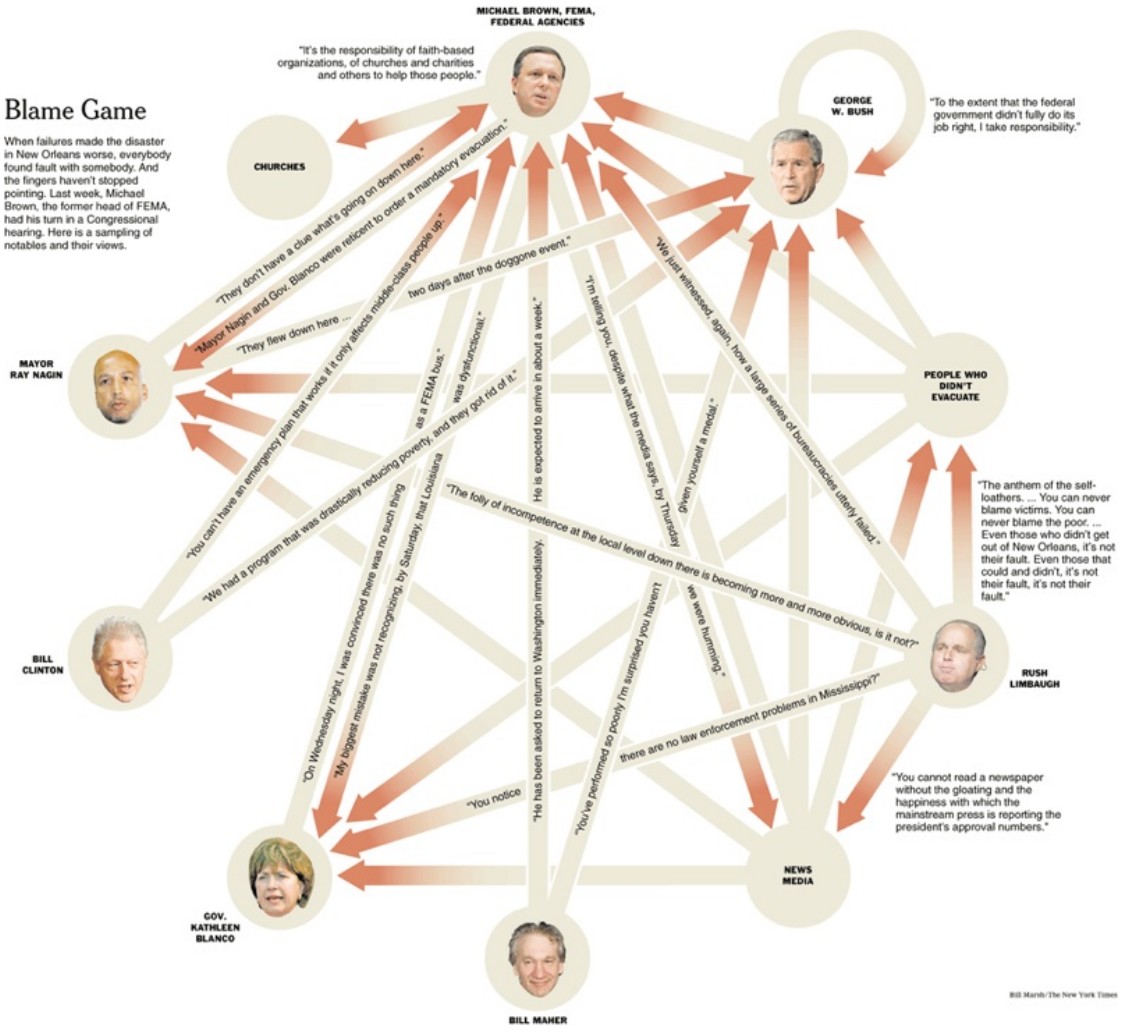

Figure 1.1: An Opinion Summary (Marsh, 2005)

would answer not questions about facts, but questions like these about opinions. Another is a system that could produce a summary of the opinions in one or more documents. Figure 1.1 presents an example summary of negative opinions expressed following the Hurricane Katrina disaster. This summary allows the reader to quickly see an overview of the parties involved and how they react to one another.

1.1.2 Tasks addressed here

In this thesis, we will address two specific tasks. The first is to build a system to automatically identify the regions of a text corresponding to expressions of opinion. This is a necessary precursor to further analysis, since all of the other questions in which we are interested relate to the expression of opinion – for example, the opinion holder and topic of the opinion can be identified as the agent and topic semantic roles once the opinion expression is identified. Although much previous work has relied on the identification of opinion expressions for a variety of sentiment-based NLP tasks (Riloff and Wiebe, 2003; Wiebe and Wilson, 2002), none has focused directly on this important supporting task. Moreover, none of the proposed methods for identification of opinion expressions has been evaluated at the task that they were designed to perform.

The second task addressed by this thesis is to identify how the opinions are filtered. The facts, events, and opinions appearing in text are often known only second- or third-hand, and as any child who has played "telephone" knows, this relaying of facts often garbles the original message. Properly understanding the information filtering structures that govern the interpretation of these facts, then, is critical to appropriately analyzing them. As with the first task, no previous work has addressed a computational solution to it.

1.2 Methods

The previous sections have introduced the natural language processing problems addressed in this thesis. Here we briefly discuss the methods used to

solve them. The field of natural language processing has changed over the past twenty years from using top-down, knowledge-driven methods (Norvig, 1987; Hobbs et al., 1988, e.g.) to focus on empirical, data-driven methods (Church and Mercer, 1993). We follow this trend, and use machine-learning methods based on human-annotated corpora in our work. One question that arises with these methods is that, since they are relatively recent, how effective are they? A scientific method should allow for reproducibility of results, it should support researchers in efficiently conducting their research, and it should lead to a high-quality result. We are aware of little research that has considered the extent to which typical empirical methods address these concerns of reproducibility, researcher efficiency, and quality of output. This thesis addresses these issues in two ways.

1.2.1 Computational workflow

Experiments in natural language processing and machine learning typically involve running a complicated network of programs to process data and evaluate algorithms. Ensuring that the workflow of these processes is done in a manner that is reproducible and efficient is critical to maintaining the goals for good methods set out above.

Researchers often write one or more UNIX shell scripts to "glue" together these various pieces, but such scripts are suboptimal for several reasons. Without significant additional work, a script does not handle recovering from failures, it requires keeping track of complicated filenames, and it does not support running processes in parallel. We present `zymake` as a solution to all these

problems. `zymake` scripts look like shell scripts, but have semantics similar to makefiles. Using `zymake` improves repeatability and scalability of running experiments, and provides a clean, simple interface for assembling components. A `zymake` script also serves as documentation for the complete workflow. We present a `zymake` script for a published set of NLP experiments, and demonstrate that it is superior to alternative solutions, including shell scripts and makefiles, while being far simpler to use than complex scientific grid computing systems.

1.2.2 Feature generation

One crucial part of using machine-learning methods for natural language processing is problem encoding, and specifically generating features. Since the tasks addressed in this thesis are new problems, no feature set is yet standard. Coming up with features for a new task is time-consuming, and also critical to producing a high-quality learned system.

In the introduction to a recent special issue of the Journal of Machine Learning Research on feature selection, Guyon and Elisseef (2003) provide a heuristic checklist on how to proceed in choosing features for a new problem. The very first item on their list is the following:

1. **Do you have domain knowledge?** If yes, construct a better set of "ad hoc" features. (Guyon and Elisseef, 2003, page 1159)

Despite the apparent importance of such domain-specific ad hoc features, no further advice is given in this special issue on how to construct them. This

is typically treated as a problem for the domain expert (not the learning researcher) to solve. Yet in natural language processing (NLP) work, the learning researcher is also the domain expert. We are interested in discovering to what extent we can exploit this dual role to allow the researcher to more quickly and effectively solve natural language learning problems. We set out desiderata for an ideal process of feature generation, and explore the effectiveness of a number of alternatives.

1.3 Contributions and Structure of This Thesis

This thesis provides several contributions to the field of natural language processing.

Identifying expressions of opinion We present the first published work that produces and evaluates a system on the task of identifying opinion expressions. Our approach achieves expression-level performance within 5% of the human interannotator agreement. This will be a basic building block for future opinion-extraction work, just as identifying and classifying proper names is a building block for factual information extraction. This is discussed in Chapter 3.

Identifying opinion hierarchies We present the first work that identifies the structure where one opinion is filtered through another. This work will be important in analyzing the reliability of expressed facts and opinions as they are passed from one source to another. This is discussed in Chapter 4.

A lightweight system for natural language processing and machine learning workflows We introduce a tool that we have found quite useful in organizing the execution of complex experiments in natural language processing and machine learning. This is discussed in Chapter 5.

Feature engineering for natural language processing We present work improving the effectiveness of a researcher's efforts in feature engineering. This is discussed in Chapter 6.

In addition, we present related work in Chapter 2, and conclude in Chapter 7.

CHAPTER 2

RELATED WORK

There has been a tremendous amount of research in opinion-oriented information extraction in recent years. It touches on a wide variety of research topics, so no single keyword would suffice to find related work. Such work might include terms such as *affect, commonsense psychology, genre, opinion, opinion mining, perspective, point of view, polarity, private state, reputation, semantic orientation, sentiment*, *subjectivity*, or *valence*[1]. For simplicity, we will refer to all of this work under the umbrella term *opinion*.

Although boundaries are hard to draw, we divide research in opinion-oriented information extraction into three subareas: classification, extraction, and lexicon building. The work discussed and proposed in this thesis most naturally falls into the extraction area, but all the following research is potentially relevant. We note that Pang and Lee (2008) provide a comprehensive overview of many areas of opinion-related research. In Chapter 4, we discuss work not focusing on opinions in particular but relevant to the notion of reporting discussed there.

This thesis also discusses problems in the empirical methods used in natural language processing research, namely feature engineering and coordination of experiments. Related work for experiment coordination is discussed in Chapter 5. In Chapter 6, we discuss feature engineering research as it relates to our work.

[1]Note that the most common meaning of *valence* in linguistics has to do with the number and type of arguments taken by a verb. We refer here to the meaning more common in psychology having to do with the difference between positive and negative semantic orientation.

2.1 Classification

Research in this area seeks to classify linguistic units (documents or sentences) into a small number of opinion-based categories. The document classification is sometimes referred to as *genre* or *style*. We do not here attempt to survey all research in this area but simply to indicate some of the areas of emphasis. While some of the work proposes a larger set of categories, most focus on subjective versus objective and positive versus negative. This work is complementary to our research in two ways. First, work that can classify areas of text as subjective versus objective can be used to identify where our opinion identification can best proceed. Second, work that classifies texts as positive versus negative might be adapted to classify the polarity of our recognized opinions.

2.1.1 Document classification

Dave et al. (2003) approach the problem of classifying online reviews from C|Net and Amazon as positive or negative. Beginning with just unigram bag of words, they try a wide variety of techniques to develop more complex models. They find approaches based on document metadata, WordNet, negation and the MiniPar parser prove ineffective, but get some benefit from stemming. N-gram features prove useful, and they get a slight benefit from feature weighting, but not from smoothing. Finally, they try several machine learning algorithms, but find in almost all variations that a method that returns the sign of an average of simple term scores performs better. As a follow-on task, they attempt to apply their classifier to general documents to find reviews, with mixed results.

Pang et al. (2002) learn to classify movie reviews as positive or negative. A random baseline gives 50% accuracy, and brief human-elicited word-lists achieve up to 64% accuracy. However, perusing the corpus including the test data to find a (probably suboptimal) set of just 14 keywords provides 70% performance. Using a bag of unigrams with three learning methods achieves about 82% performance. Adding a variety of more complex features, such as bigrams, part of speech, etc. does not improve performance.

2.1.2 Sentence classification

Wiebe et al. (2004) learn lists of subjectivity clues, learn to disambiguate these clues in context, and then apply the disambiguated clues to sentence- and document-level subjectivity classification. The clues are based on hapax legomena as well as n-grams learned from corpora of documents annotated for subjectivity at the expression and document level. Clues are measured for effectiveness by their "precision", the fraction of documents (or sentences) in which they appear that are opinionated. After using these techniques to find high-precision clues, the authors describe methods for determining whether, in a given context, a clue actually represents an opinion. Finally, the disambiguated clues are used to identify subjective or objective sentences and documents, with quite good results.

Wiebe and Riloff (2005) approach sentimental sentence classification using an unsupervised method. They begin by creating rule-based classifiers, whose input is fed to a system for learning extraction patterns. This system generates higher-recall classifiers, which are finally used to produce training data

for a Naive Bayes classifier, which is self-trained to higher performance. The rule-based classifiers combine large sets of clues learned in previous work or extracted from manually-created knowledge resources. These clues are used as a baseline in Chapter 3.

Yu and Hatzivassiloglou (2003) motivate their opinion-oriented classification by the problem of opinion-oriented question answering, but do not actually directly address question answering in this work. Instead, they address three components of opinion detection and organization: document-level classification as to subjective versus objective, sentence-level classification subjective versus objective, and the polarity of words and sentences. Document level classification achieves 97% results distinguishing the news from editorial sections of the Wall Street Journal using a Naive Bayes classifier. A much more complicated model also using a similarity classifer and a bootstrapping method achieves mixed results on sentence-level subjectivity classification. They measure the polarity of words by the log-likelihood of cooccurrence with between 1 and 600 seed terms, measuring sentences by the average word polarity.

2.2 Extraction

This subarea, rather than focusing on the classification of documents or sentences, focuses on learning to extract smaller pieces of text, words or phrases, and/or create more complex annotation schemes. This is the category into which our work falls. Section 3.1 discusses one particularly close piece of work, that of Bethard et al. (2004). Here we discuss two categories of work, divided by the type of text analyzed: reviews and newswire.

2.2.1 Extraction from reviews

Review text classification was discussed in the previous section, but another set of work seeks to extract more detailed information from reviews. Much of this work centers around identifying *features* or *attributes* of a product that a customer might have an opinion about, and then extracting particular opinions about each feature. This is rather different from the news domain discussed in the next section, in which an opinion might be about any topic (not just a feature of the target of the review) and held by any entity (not just the writer of the review).

In Kobayashi et al. (2004), the goal is to find *attribute* and *value* expressions for a given domain. Examples from an automotive domain might be *seat, window* or *door* for the former category and *comfortable* for the latter. The semi-automatic process proceeds by applying a set of 8 hand-written co-occurrence patterns to web data, which are then filtered by a human. The authors compare this approach to a purely manual collection of terms, and find that the semiautomatic approach is much faster but doesn't find all the human-selected terms, because of a poor treatment of complex phrases.

Yi, Nasukawa and colleagues (Nasukawa and Yi, 2003; Yi et al., 2003) identify phrasal sentiments expressed towards a particular product or company. Nasukawa and Yi (2003) create a hand-built lexicon of sentiment terms. Given a target term, such as *IBM* or *Range Rover*, plus context, their system determines the polarity of the sentiment expressed in that context, along with the particular sentiment terms used there. They evaluate their system by precision and recall on some small labeled corpora and by precision alone on larger corpora. Their system achieves high precision (around 95%) but lower recall (around

20%). They also note that more complex sentence structure confuses their simple parser.

Yi et al. (2003) extend this approach by allowing the target terms to be not just products and companies but also "feature terms" such as *lens* or *battery life* that are related to the target term. They introduce two statistical algorithms for finding feature terms for a given domain, and find that one achieves extremely high precision for digital camera and music reviews. They then evaluate their sentiment analysis algorithm on the review corpus.

Jindal and Liu (2006) extend the task of feature extraction to identify comparative relations. A comparative relation is a four-tuple, such as (better, optics, Canon, Sony, Nikon), which one might gloss as "Canon's optics are better than the optics of Sony or Nikon." The authors first identify sentences containing comparatives, using a combination of part-of-speech-based keyword and a machine learning method, and then extract the four parts of the tuple from the comparative sentences using a technique called label sequential rules. They find their method outperforms one based on conditional random fields, and achieves 72% F-measure across the latter three elements of the tuple.

Morinaga et al. (2002) introduce "Reputation Analysis", a tool for marketing researchers to examine how their product is seen by customers. Essentially, they learn a set of probabilistically ranked if-then rules: based on a conjunction of lexical items, assign a sentence the label positive or negative and which product it describes. They then find cooccurring words, rank opinions by typicality for a given category, and perform principal component analysis on categories and words. The output is thus intended for a human user, rather than applied to a specific quantifiably evaluable task.

2.2.2 Extraction from news

In this section we discuss extraction of opinions from news text. Since news covers many different domains, and reports the opinions of many different parties, there are a wide variety of potential tasks, such as identifying low-level opinions and their holders, the strength and polarity of the opinions, as well as higher-level tasks such as question-answering and summarization. Our work fits neatly into this section, with the other work here filling out other parts of the story.

Wiebe and Wilson (2002) note that while one can create a list of potentially subjective elements (PSEs), in context a PSE might not actually be subjective. They evaluate on several Wall Street Journal corpora and find that features based on the density of nearby PSEs help in determining whether a given PSE is actually subjective.

Wilson et al. (2005b) introduce the notion of "contextual polarity" – that is, the polarity of an opinion expression in context may differ from its prior polarity due to negations, modality, or other factors. They hand-annotate a corpus for this task, and then collect a large prior polarity lexicon based on clues used in prior work as well as resources such as a dictionary and a thesaurus. Using only this prior polarity lexicon achieves performance on their task of just 48%, but using a two-step machine learning method results in 65.7% performance. The clue dictionary developed in this work is used as a baseline and as features in Chapter 3.

Munson et al. (2005) use two opinion extraction problems (along with two other problems) to evaluate the utility of "ensemble selection" to optimize a

machine learning method to a variety of different metrics (not just error). The two opinion extraction problems used are DSESE identification (discussed in Section 3.2) and opinion hierarchy identification (discussed in Chapter 4).

Choi et al. (2005) attempt to identify entities that are sources of opinions. Defining the problem as an I/O/B tagging task, the authors use both automatically derived extraction patterns and conditional random fields to solve it. The best results come from combining the two approaches.

Choi, Breck, and Cardie (2006) combine the system described in the previous paragraph with the system described in Chapter 3, along with a global inference system using integer linear programming, to jointly identify (a) expressions of opinion, (b) sources of opinion, and (c) the relation that a particular source expressed a particular opinion.

Wilson et al. (2004) take on the task of classifying the strength of opinions at the clause level. The authors amass a large stable of opinion "clues" from previous work, and also add some new syntax-based clues for this problem. They discuss how the clues can be used as features, and show that re-organizing the clues for the current task helps performance. They evaluate several machine learning algorithms and provide an ablation study to examine several variations in the feature set.

Stoyanov et al. (2005) approach the problem of answering opinion-oriented questions. They introduce the OpQA corpus, containing both opinion and fact questions along with text spans corresponding to their answers. Corpus analysis shows that answers to opinion questions differ substantially in form from answers to fact questions, in that they are longer, more apt to be partial, and less

likely to be confined to a single constituent. The authors use manual and automatic filters to remove factual sentences returned by the information retrieval component when their system is faced with opinion questions, and find that performance is improved.

Kim and Hovy (2006) seek to identify not just opinion holders but also the topics of opinions. They manually construct an opinion lexicon, then identify the frame from FrameNet associated with each word in the lexicon (using a clustering method to assign frames to words not in FrameNet). Then, using a semantic role labeling system, they select the frame elements corresponding to topic and holder. They evaluate their system on a selection of about 2000 sentences from FrameNet as well as 100 sentences from online news text, achieving good results on both.

2.3 Lexicon building

In this section we describe a small sample of existing research on methods for classifying word *types* (as opposed to tokens) based on their opinion content, independent of context. In addition to the work described here that focuses on learning lexica, much of the other opinion research uses lexica in service of another task, either lexica built by hand (e.g. Tong (2001; Subasic and Huettner (2001; Das and Chen (2001)) or learned (e.g. (Wilson et al., 2005b; Wiebe and Riloff, 2005)). We discuss work in two categories – polarity lexica, where the lexica include a categorization as to the polarity of each item, and subjectivity lexica, where the words and phrases in the lexicon are typically subjective. The former category may be useful in extending our work

to categorize the polarity of opinions; some lexica in the latter category are used in this work as features for identifying opinion expressions in context.

2.3.1 Polarity lexica

Hatzivassiloglou and McKeown (1997) learn to predict the semantic orientation of adjectives (positive or negative). They use a Wall Street Journal corpus, with automatically assigned part-of-speech labels, and form a graph. The nodes of the graph are adjectives, and edges are created for adjectives found linked by conjunctions or by morphological similarity. The authors then cluster the nodes and label as positive the cluster with higher average frequency.

Turney and Littman (2002; 2003) learn the polarity (here, "semantic orientation") of individual words using seeded unsupervised learning. The basic method is to assign a score to each word a score $SOA(w) = \sum_{p \in Pos} assoc(w, p) - \sum_{n \in Neg} assoc(w, n)$, where Pos and Neg are sets of positive and negative words. The function $assoc$ is either pointwise mutual information (PMI) based on AltaVista NEAR queries, or latent semantic analysis (LSA). LSA appears to give better results than PMI, but does not scale to larger corpora.

2.3.2 Subjectivity lexica

Riloff and Wiebe (2003) note that hand-built lists of subjective expressions are always going to be low-recall, if high-precision. They therefore begin with a list of such expressions and use a bootstrapping method (AutoSlog-TS) to learn further patterns. Unlike typical usages of AutoSlog, since the goal is not to

extract phrases but simply to use the patterns to detect subjectivity, a ranking measure and threshold was used to produce a fully automatic bootstrapping method. The resulting method achieves significantly higher recall with only a small drop in precision compared to the original lists.

Riloff et al. (2003) use two varieties of bootstrapping to learn lists of subjective nouns. With 20 seed words, the two algorithms produce about 4000 words, which with a few hours work are filtered to a list of a thousand good indicators. Applying these new indicators to the task of Wiebe et al. (1999), the authors achieve a solid 2% precision increase over that previous work.

CHAPTER 3

IDENTIFYING EXPRESSIONS OF OPINION IN CONTEXT[1]

The goal of our research is to create a system for fine-grained information extraction of opinions. As we have argued in Chapter 1, central to all such analysis is first identifying the expressions that denote opinions.

Consider the following sentences, in which we denote two kinds of opinion expression in boldface and italic (described below).

1: Minister Vedrine **criticized** the White House reaction.

2: 17 persons were killed by sharpshooters **faithful to** the president.

3: Tsvangirai **said** the election result was "*illegitimate*" and a clear case of "*highway robbery*".

4: Criminals have been *preying* on Korean travellers in China.

5: The speaker **argued** that the committee **has rejected** the president's bid to open a dialogue with China.

Wiebe et al. (2005) distinguish two types of opinion expressions, and we follow their definitions here. We also define three further types of expressions related to opinions.

Direct subjective expressions (DSEs), shown in boldface, are spans of text that explicitly express an attitude or opinion. "Criticized" and "faithful to" (examples 1 and 2), for example, directly denote negative and positive attitudes towards the "White House reaction" and "the president", respectively. Speech

[1]Portions of this chapter are adapted from Breck et al. (2007).

events like "said" in example 3 can be DSEs if the propositions that they introduce express subjectivity.

Expressive subjective elements (ESEs), shown in italics, are spans of text that indicate, merely by the specific choice of words, a degree of subjectivity on the part of the speaker. The phrases "illegitimate" and "highway robbery", for example, indirectly relay Tsvangirai's negative opinion of "the election result" (example 3), and the use of "preying on" (instead of, say, "robbing") indicates the writer's sympathy for the Korean travellers in example 4.

Speech expressions (SE), such as "said" or "criticized" in the examples above, can play a role in the information flow of a sentence whether they are subjective or not. Chapter 4 discusses this information flow among opinions and speech expressions.

Objective speech expressions (OSEs) are defined as as $SE - DSE$ (set difference), i.e. speech expressions that only express factual content.

Direct subjective and speech expressions (DSESEs) are defined as $DSE \cup SE$, i.e. any of the expressions discussed so far except ESEs[2].

Table 3.1: More examples of opinion expression types

types	expression
DSE, DSESE	Peter *believes* that Narnia is real.
ESE	Casablanca is *the best film of all time*.
OSE, SE, DSESE	Jack *told* Susan it was raining.
DSE, SE, DSESE	"That's a terrible idea", Murray *said angrily*.

Table 3.1 presents further examples of DSEs, ESEs, OSEs, SEs, and DSESEs.

[2]What we are calling here DSESEs have been referred to in the past as *on*s (Wiebe et al., 2003), *pse*s (Breck and Cardie, 2004), (I apologize for having used that abbreviation as it might be confused with PSE, meaning potentially subjective expression) and *psf*s (Munson et al., 2005).

While some previous work identifies opinion expressions in support of sentence- or clause-level subjectivity classification (Riloff and Wiebe, 2003; Wiebe and Wilson, 2002), none has directly tackled the problem of opinion expression identification, developed methods for the task, and evaluated performance at the expression level. Instead, previous work in this area focuses its evaluation on the sentence-level subjectivity classification task.

In this chapter, we will present four approaches to the problem of identifying opinion expressions. Section 3.1 discusses one piece of related work that identifies some expressions of opinion. In Section 3.2, we review our initial approach to identifying single-word DSESEs, and later extensions by others. In Section 3.3, we extend this model with additional features. In Section 3.4, we extend our approach to include multi-word DSESESs. Finally, in Section 3.5, we describe an approach to identifying just the opinionated expression types defined above, DSEs and ESEs. We achieve F-measures of 63.43% for ESEs and 70.65% for DSEs, within 5% of the human interannotator agreement for DSEs and within 10% for ESEs.

3.1 Related work

Before presenting our own approach to identifying opinion expressions, we review one piece of closely related work. Bethard et al. (2004) seek to extract propositional opinions and their holders. They define an opinion as "a sentence, or part of a sentence that would answer the question 'How does X feel about Y?' " A propositional opinion is an opinion "localized in the propositional argument" of certain verbs, such as "believe" or "realize". Expressed in

Table 3.2: Breakdown of classes of DSESEs. "writer" denotes DSESEs with the writer as source. "No parse" denotes DSESEs in sentences where the parse failed, and so the part of speech could not be determined.

DSESE class	count
writer	9808
verb	7623
noun	2293
no parse	278
adjective	197
adverb	50
other	370

Table 3.3: Breakdown of number of DSESEs per sentence

number of DSESEs	number of sentences
1	3612
2	3256
3	1810
4	778
5	239
>5	113

our nomenclature, their task corresponds to identifying a DSESE, its associated direct source, and the content of the private state. However, they consider as DSESEs only verbs, and further restrict attention to verbs with a propositional argument, which is a subset of the direct subjective and speech expressions that we consider here. Table 3.2, for example, shows the diversity of word classes that correspond to DSESEs in the MPQA corpus introduced in Section 3.2. Perhaps more importantly for the purposes of this work, their work does not address information filtering issues, i.e. problems that arise when an opinion has been filtered through multiple sources. Namely, Bethard et al. (2004) do not consider sentences that contain multiple DSESEs, and do not, therefore, need to identify any indirect sources of opinions. As shown in Table 3.3, however, we find that sentences with multiple non-writer DSESEs (i.e. sentences that contain 3 or more total DSESEs) comprise a significant portion (29.98%) of the MPQA

corpus. An advantage over our work, however, is that Bethard et al. (2004) do not require separate solutions to DSESE identification and the identification of their direct sources.

3.2 Identifying single-word direct subjective and speech expressions

We chose to begin by identifying a broad category of opinion-related expressions, namely direct subjective and speech expressions (DSESEs). In this section, we present a problem encoding, features, data, evaluations, and machine learning methods used by our initial experiments (Wiebe et al., 2003; Wiebe et al., 2002) as well as by some follow-on work by others (Tenenbaum, 2004; Munson, 2004).

Problem encoding Most of the DSESEs in the data (like the first one in Figure 3.1) consist of a single word – 56.8% in the development data. Therefore, as a first step we chose to focus on the problem of *single-word DSESE identification*. That is, every token in the corpus is an instance, and the class of an instance is 1 if that token alone is a DSESE, and 0 otherwise. Figure 3.2 shows example sentence 5 with the class labels that would be given to each token. Importantly, note that all tokens in a multi-word DSESE are assigned a class label of 0. All previous work had used this problem encoding[3].

[3]OpinionFinder (Wilson et al., 2005a) does have one rule which finds multiple word phrases.

The speaker **argued** that the committee **has rejected** the president's bid to **open a dialogue** with China.

Figure 3.1: Example sentence 5, with direct subjective expressions and speech expressions (DSESEs) marked in **bold**.

The speaker **argued** that the committee **has rejected** the president's bid to **open a dialogue** with China.
The/0 speaker/0 **argued/1** that/0 the/0 committee/0 has/0 rejected/0 the/0 president's/0 bid/0 to/0 open/0 a/0 dialogue/0 with/0 China/0 ./0

Figure 3.2: Identifying single-word DSESEs: binary classification

Features We include features to allow the model to learn at various levels of generality. We include lexical features to capture specific phrases, local syntactic features to learn syntactic context, and dictionary-based features to capture both more general patterns and expressions already known to be opinion-related. For pedagogical reasons, we present the features as categorically valued, but in our model we encode all features in binary. That is, for every value v of every feature f, we create a binary feature (f, v) whose value is 1 for a token t if $f(t) = v$ and 0 otherwise. We do not create binary features for the special value **null**. Table 3.4 shows feature values for the first three tokens of example sentence 5.

Lexical features We include features w_i, defined to be the word at position i relative to the current token. We include $w_{-2}, w_{-1}, \ldots w_2$.

Syntactic features We include a feature **pos**, defined to be the part of speech of the current token according to the GATE part-of-speech tagger (Cunningham et al., 2002). We also include three features **prev, cur,** and **next**, defined to be the previous, current, or following constituent type, respectively, according to the

Table 3.4: Original features for first three tokens (t_1, t_2, t_3) of example sentence 5. w_{-2} through w_2 are a word-window about the target word, **pos** is the part of speech of the target word, **cur**, **next**, and **prev** are based on a partial parse, and **lev** and **fn** are based on the Levin and FrameNet word lists.

feature f	$f(t_1)$	$f(t_2)$	$f(t_3)$
class	0	0	1
w_{-2}	-	-	The
w_{-1}	-	The	speaker
w_0	**The**	**speaker**	**argued**
w_1	speaker	argued	that
w_2	argued	that	the
pos	DT	NN	VBD
prev	-	nx	nx
cur	nx	vx	vx
next	vx	vx	vx
lev	-	-	verb of communication
fn	-	-	comm

CASS partial parser (Abney, 1996)[4]. Examples of these features are presented in Table 3.4.

Dictionary-based features In this model, we include two dictionary-based features. The first feature, referred to as **lev** in Table 3.4, is based on Levin's (1993) categorization of English verbs. A verb may appear in one or more sections of Levin's book. Paul Davis, a linguist then at the Ohio State University, identified three sections that may be likely to represent speech event verbs: section 37 ("verbs of communication"), section 33 ("judgment verbs"), and subsections 1-5 of section 29 ("verbs with predicative complements"). For each word w, the **lev** returns the name of the major section in which w appears, or **null** if w appears in none of the sections[5]. The second dictionary-based featured, labeled

[4]Cass is available for download at `http://www.vinartus.net/spa/`.

[5]If a verb appears in more than one section, 37 takes precedence over 33, which takes precedence over 29. This is based on Davis' judgment that verbs in section 37 are most likely to be speech events, followed by 33 and then 29.

Table 3.5: Tenenbaum's features for first three tokens (t_1, t_2, t_3) of example sentence 5

feature f	$f(t_1)$	$f(t_2)$	$f(t_3)$
target word	The	speaker	argued
noun synset	-	{speaker, talker}07596348	-
verb synset	-	-	{argue, reason}00524590
adjective synset	-	-	-
adverb synset	-	-	-

fn in Table 3.4, is based on the categorization of nouns and verbs in FrameNet[6]. Davis selected certain frames in FrameNet[7] as being indicative of speech events, and so the value of the **fn** feature for a word w is f if FrameNet lists w as a lexical unit for the frame f and f is one of the selected frames, and **null** otherwise.

Wordnet features Tenenbaum (2004) extended the feature set with four additional features based on the WordNet database (see Table 3.5). One feature is created for each of the four open-class parts of speech (noun, verb, adjective, adverb). Given a part of speech, she looks up the target word in WordNet (or its lemma if the target word is not present), and returns the synset ID of the first (most frequent) sense of the word or lemma.

Data The Multi-Perspective Question Answering (MPQA) corpus (Wiebe et al., 2005)[8] consists of 535 newswire documents annotated with a variety of annotations of interest for subjectivity research. In particular, all DSESEs, DSEs and ESEs in the documents have been manually identified. 135 documents are reserved for feature engineering and model development, with the remaining 400

[6] `http://www.icsi.berkeley.edu/~framenet/`

[7] Davis selected the *body-movement* frame from the body domain, the *candidness, cogitation, coming-to-believe, invention*, and *judgment* frames from the cognition domain, and the *commitment, communication_noise, communication_response, conversation, encoding, gesture, hear, manner, questioning, request, statement*, and *volubility* frames from the communication domain.

[8] Available at `http://www.cs.pitt.edu/mpqa/`. We use version 1.1 of the corpus.

Table 3.6: Statistics for test data

number of sentences	8297
number of DSESEs	8992
number of DSEs	6712
number of ESEs	8640
average length of DSESEs	1.78 words
average length of DSEs	1.86 words
average length of ESEs	3.33 words

kept blind for testing. Munson (2004) used 10-fold cross-validation of these 400 documents for evaluation, and Tenenbaum (2004) followed the same document splits and evaluation procedure. We use the same splits and cross-validation for the remainder of the experiments in this chapter. Table 3.6 presents some statistics on these 400 documents.

Evaluation Since this problem involves binary classification with a large class skew, accuracy is not an appropriate evaluation metric. Therefore, recall (R^{single}), precision (P^{single}), and F-measure (F^{single}) have been used for evaluation. Munson et al. (2005) also employed breakeven point and a number of other standard machine learning metrics, which we will not present. We define

$$R^{single} = \frac{|S \cap W|}{|W|}$$

$$P^{single} = \frac{|S \cap W|}{|S|}$$

$$F^* = \frac{1}{\frac{\frac{1}{P^*}+\frac{1}{R^*}}{2}} = \frac{2P^*R^*}{P^*+R^*}$$

where S is the set of tokens identified by the system as being *single* word DSESEs and W is the set of tokens that are single word DSESEs according to the manual annotations. F^* is defined generically for any variant of precision and recall, since later in this chapter we will define additional metrics.

Methods and Results Our initial experiments on this task were conducted as part of an 8-week summer workshop called the Multi-Perspective Question Answering workshop, led by Janyce Wiebe (Wiebe et al., 2002). The full corpus described above was not available during the workshop, and in fact annotated data only became available near the end. As a result, we were only able to conduct a small set of initial experiments on 92 documents. We used the k-nearest-neighbor and naive Bayes learning methods, both using default settings of the Weka[9] machine learning package. We compare to a baseline that predicts any word found on the Levin/FrameNet lists to be a DSESE. The results are presented in Table 3.7, copied from Wiebe et al. (2002). We see that the learning methods outperform the baseline according to recall and F-measure and that results are encouragingly high.

Further experiments were performed by others using the same feature set but different learning algorithms and with the larger amount of data described earlier. Table 3.8 presents the results of Munson (2004), who used a number of methods: nearest-neighbor or memory-based learning[10], support vector machines (SVM)[11], decision trees[12], and rule learning[13], and found that again, the learning methods outperformed the baseline according to recall and F-measure, and often according to precision as well. Table 3.8 also presents the results of Tenenbaum (2004), who performed experiments using the additional WordNet-based feature described above, and found that using certain learning methods, a large performance improvement could be found.

[9]Weka (Witten and Frank, 1999), available at `http://www.cs.waikato.ac.nz/ml/weka`.

[10]TiMBL, (Daelemans et al., 2000), available at `http://ilk.uvt.nl/timbl`.

[11]SVM^{light}, available at `svmlight.joachims.org`.

[12]C4.5 (Quinlan, 1986)

[13]RIPPER (Cohen, 1995)

Table 3.7: Results for identifying single-word DSESEs using the approach of Wiebe et al. (2003)

method	P^{single}	R^{single}	F^{single}
Baseline	69.9	47.7	56.7
Naive Bayes	46.7	76.6	58.0
k-NN	69.6	63.4	66.4

Table 3.8: Results for identifying DSESEs using the approaches described in Section 3.2.

method	P^{single}	R^{single}	F^{single}
Baseline	74.5	41.1	53.2
Results from Munson (2004)			
`TiMBL` (k=1)	74.5	65.3	69.6
`SVM`^{light}	88.7	55.2	68.0
`RIPPER`	80.1	53.4	64.0
`C4.5`	77.9	42.8	55.1
Results from Tenenbaum (2004)			
`TiMBL`	73.9	65.9	69.7
`SVM`^{light}	85.6	64.3	73.4
`RIPPER`	83.9	42.7	56.6
`C4.5`	88.0	41.7	56.5

3.3 New features for identifying single-word direct subjective and speech expressions

The results in the previous section show that the DSESE identification problem is amenable to automatic analysis. However, the features there were, as mentioned, developed quickly during a short summer workshop. In this section, we explore the result of adding a number of new features. We intend to include more features both to provide additional context to the learner and to allow it to better generalize from the training data.

The new features we add are listed for the third word of example sentence 5 in Table 3.9. First of all, while Tenenbaum had previously included features

based on the synsets of the target word in WordNet, we hoped to allow additional generalization by including more of the hierarchy, so we included as features every synset that is an ancestor of a synset containing any sense of the target word. These features are referred to subsequently as `wordnet`. We also expanded the word window to four words on either side of the target word, included the lemmas[14] of all the words in the window, and word bigrams beginning with all the words in the window. Finally, we included the original categories from Levin's work, as opposed to Davis' selected categories, as well as all prefixes of the original category. These additional features are referred to subsequently as `other`.

Methods In previous work, while the performance of the different methods varied, linear support vector machines were either the best or second best, as measured by F^{single}. Due to their performance, quick training speed, and easy handling of thousands of features, we chose to perform the experiments reported in this section using linear support vector machines with default parameters.

Results and Discussion Table 3.10 presents the results of using the new `other` features and the new `wordnet` feature. Row 1 is equivalent to the SVM^{light} results of Munson (2004) presented in Table 3.2, and row 3 is equivalent to the SVM^{light} results of Tenenbaum (2004)[15]. Row 5 presents a system called OpinionFinder (Wilson et al., 2005a), which has a fixed set of words pre-

[14]We use Abney's program `stemmer`, part of the SCOL package, to identify inflectional lemmas.

[15]The small differences between row 1 and the Munson results are due to small differences in data encoding.

Table 3.9: Additional features for third token (t_3) of example sentence 5 (*argued*). The list of WordNet hypernyms is abbreviated. For the SVM classifier, all features are encoded as binary indicator features: 1 if a given (feature name, feature value) appears for the instance in question, and 0 otherwise.

feature f	$f(t_3)$
hypernym for lemma 0	{00524590} argue, reason
hypernym for lemma 0	{00525080} present, lay out
hypernym for lemma 0	{00682542} state, say, tell
hypernym for lemma 0	{00636716} express, utter, give tongue to
hypernym for lemma 0	{00525378} argue, contend, debate, fence
hypernym for lemma 0	⋮
word -2	The
word -1	speaker
word 0	argued
word 1	that
word 2	the
word 3	committee
word 4	has
lemma -2	The
lemma -1	speaker
lemma 0	argue
lemma 1	that
lemma 2	the
lemma 3	committee
lemma 4	have
bigram -2	The_speaker
bigram -1	speaker_argued
bigram 0	argue_that
bigram 1	that_the
bigram 2	the_committee
bigram 3	committee_has
bigram 4	has_rejected
Levin category for lemma 0	29.2
Levin category for lemma 0	29

Table 3.10: Results for DSESE identification. All rows include the lexical and syntactic features and the Levin/FrameNet dictionary features. The first row is the base system, identical to that used in previous work. `wordnet` indicates adding our WordNet features. `other` denotes adding our additional new features. Superscripts denote one standard deviation.

row#	`other`	`wordnet`	R^{single}	P^{single}	F^{single}
1	.	.	$54.84^{4.2}$	$88.96^{1.6}$	$67.79^{3.5}$
2	✓	.	$60.06^{4.4}$	$\mathbf{89.34^{1.8}}$	$71.77^{3.6}$
3	.	✓	$64.52^{4.2}$	$86.35^{2.8}$	$73.83^{3.6}$
4	✓	✓	$\mathbf{64.57^{4.5}}$	$86.82^{2.6}$	$\mathbf{74.02^{3.8}}$
5	OpinionFinder		$47.39^{3.8}$	$68.78^{2.8}$	$56.07^{3.4}$

judged to be likely opinions, and predicts any word on its list to be an opinion[16].

While we had hoped that our different encoding of information from WordNet would be helpful, in fact the differences between our `wordnet` feature (row 3) and Tenenbaum's (see Table 3.2) are quite small. Also while both new features are useful, we see that the `wordnet` feature is more helpful (row 3 - row 1) than the `other` feature (row 2 - row 1). Moreover, using both features together (row 4) does not provide a significant improvement over just using the `wordnet` feature (row 3)[17]. As a result, in subsequent experiments, we omit the `other` feature, except for expanding the word-window to 4 words on either side of the target.

3.4 Identifying all direct subjective and speech expressions

The previous approaches have sought to identify single-word DSESEs, because most DSESEs consist of a single word. However, there are two problems with

[16]OpinionFinder has since been updated and now uses a variant of the classifier described in Section 3.5.

[17]Using a paired t-test of the cross-validation folds at a $p < 0.05$ significance level; all other differences between pairs of rows are significant, except for the precisions of rows one and two.

The/0 speaker/0 **argued/1** that/0 the/0 committee/0 *has/0 rejected/0* the/0 president's/0 bid/0 to/0 *open/0 a/0 dialogue/0* with/0 China/0 ./0

Figure 3.3: The `filter` modification: remove the italicized tokens from the training data.

this – first, this is not really what we want, since not all DSESEs are a single word, and second, it may actually be harder for a learner to identify just single-word DSESEs, because the training data is confusing. In this section, we introduce two modifications to the training procedure designed to reduce the confusion to the learner. We measure this by evaluating using a new metric that judges the learner on its ability to identify all DSESEs, not just single-word DSESEs.

Training modifications: `filter` Words like *rejected* sometimes occur as a single word DSESE, but sometimes (as in sentence 5) occur in a multiple word DSESE, which are considered negative examples for our single-word DSESE identification task. We feared this might confuse the learner, so we implemented a simple filter which removes from the training data all instances (tokens) that occur inside multiple-word DSESEs. In Figure 3.3, the italicized tokens would be removed from the training (but not test data). We refer to this modification subsequently as `filter`.

Training modifications: `core` We next observed that many DSESEs, while consisting of multiple words, in fact appear to have a single "core" word plus a few auxiliaries. In example sentence 5, *has rejected* consists of the core *rejected* plus the auxiliary *has*. Another example is *remark* in the common DSESE *the remark*. To utilize this observation, we modify the training data as follows. Pre-

The/0 speaker/0 **argued/1** that/0 the/0 committee/0 has/0 ***rejected/1*** the/0 president's/0 bid/0 to/0 open/0 a/0 dialogue/0 with/0 China/0 ./0

Figure 3.4: The **core** modification: change the class variable (in training data) of the italicized words to positive.

viously, the class variable for an instance (token) was 1 if the token, *by itself, was a DSESE*, and 0 otherwise. Now, the class variable is 1 if the token *is the core of a DSESE*, and 0 otherwise. Figure 3.4 shows the new class values for example sentence 5. To find the core word of a DSESE, we employ a simple heuristic. Any single word DSESE is its own core. If a multiple word DSESE consists of an optional series of determiners, auxiliaries and negations, followed by one word, followed by an optional preposition or complementizer, then that single word in the middle is the core. Other DSESEs simply have no core – as before, all their tokens are marked with class label 0. However, in the development data, this heuristic finds a core for 77.6% of the DSESEs. We refer to this training data modification subsequently as `core`.

Evaluating all DSESEs The existing single word evaluation metrics are reasonable for the single word DSESE problem. However, they produce unintuitive results when considered as evaluating the system on all, possibly multi-word DSESEs. Consider the hypothetical system response in Figure 3.5. On this sentence, $P^{single} = \frac{1}{2}$, that is, $\left(\frac{|\{\text{argued}\}|}{|\{\text{argued,rejected}\}|}\right)$, even though every token proposed by the system is actually contained in a DSESE. Also $R^{single} = 1$, that is, $\left(\frac{|\{\text{argued}\}|}{|\{\text{argued}\}|}\right)$, even though one DSESE in the sentence is completely missed by the system.

As a result, we introduce additional metrics that evaluate against not just single-word DSESEs, but all DSESEs. However, since our method only iden-

human annotation
The/0 speaker/0 *argued/1* that/0 the/0 committee/0 *has/0* **rejected/0** the/0 president's/0 bid/0 to/0 *open/0 a/0 dialogue/0* with/0 China/0 ./0
system response
The/0 speaker/0 *argued/1* that/0 the/0 committee/0 *has/0* **rejected/1** the/0 president's/0 bid/0 to/0 *open/0 a/0 dialogue/0* with/0 China/0 ./0

Figure 3.5: A possible system response. The true DSESEs are marked in italics, and the class assigned either by the original problem encoding or by the system is marked after the token by 0 or 1.

tifies single words, we soften the standard definitions of precision and recall. We define soft precision as $SP^a = \frac{|\{s|s \in S \wedge \exists\ c \in C\ s.t.\ a(c,s)\}|}{|S|}$ and soft recall as $SR^a = \frac{|\{c|c \in C \wedge \exists\ s \in S\ s.t.\ a(c,s)\}|}{|C|}$, where $a(c, p)$ is a predicate that is true just when expression c "aligns" to expression p in a sense defined by a. Here S is again the set of words predicted to be DSESEs, but C is the set of *all* DSESEs. We report results according to the $overlap(c, s)$ predicate, true when s is one of the words in the expression c.

3.4.1 Results and Discussion

Table 3.11 presents the results of our modifications to the approach. Our revised model is intended to identify all DSESEs, not just single-word DSESEs, so we present results using the new all-DSESE metrics; however, we also present results using the single-word DSESE metrics, for comparison. Also for comparison, we again include the OpinionFinder baseline, and the systems described in the previous section, equivalent to previous work. The rest of the table presents the performance of the base system, supplemented by various combinations of our WordNet features (`wordnet`) and the two training modifications (`core` and `filter`). Note that model complexity roughly increases as one reads down the

rows of the table. Nearly all differences between rows are significant, using a paired t-test on the cross-validation folds[18].

First of all, note that according to the $SF^{overlap}$ metric, our three main modifications (`wordnet`, `core`, and `filter`), raise the performance from 51% to over 78% (row 1 to row 7). The recall (measured by $SR^{overlap}$) doubles from 35% to 70%. The individual contributions of each of these three modifications are also quite large (compare row 1 to rows 2, 3, or 4).

We note that the differences in precision between the various metrics are quite large, especially with the more complex models. The final model (row 7) in particular is judged to have 57% precision by P^{single} but 89% precision by $SP^{overlap}$, confirming our suspicion that P^{single} is drastically underestimating the true precision of the model. We are also quite excited to note that as the models increase in complexity (moving down the table), recall steadily increases (according to all of the recall versions), while precision (as measured by P^{single}) decreases only slightly. Note that since the support vector machine produces a prediction that can be interpreted as a confidence, we can straightforwardly tune our model to perform at a desired precision level, trading off recall.

The reader may wonder at two features of Table 3.11. First, the performance on the single-word DSESE metrics steadily decreases as one reads down the table, indicating that our more complex models are hurting, rather than helping, performance. In fact, the recall (R^{single}) does steadily increase, and the decrease is explained by a loss in precision (P^{single}). This is, as mentioned, due to the metric penalizing the more complex models for in fact identifying DSESEs which

[18] At a $p < 0.05$ significance level, all differences between rows are significant except for the following: R^{single}between rows 2 and 4; P^{single}between rows 3 and 8; F^{single}between rows 1 and 4, 1 and 6, 1 and 7, 3 and 5, 4 and 6, 6 and 7; $SP^{overlap}$between rows 3 and 4, 3 and 8, 4 and 8, 5 and 6, 5 and 8, 6 and 8; and $SF^{overlap}$between rows 1 and 8.

Table 3.11: Results for DSESE identification. All rows include the lexical and syntactic features and the Levin/FrameNet dictionary features. The first row is the base system, identical to that used in previous work. `wordnet` indicates adding our WordNet features. `filter` denotes filtering out multi-word DSESEs from the training data. `core` denotes classifying as positive training instances the core words of some multi-word DSESEs. See Section 3.4 for more detail and for descriptions of the various evaluation metrics. Superscripts denote one standard deviation.

row#	wordnet	core	filter	single-word DSESEs			all DSESEs		
				R^{single}	P^{single}	F^{single}	$SR^{overlap}$	$SP^{overlap}$	$SF^{overlap}$
1	.	.	.	$54.84^{4.2}$	$\mathbf{88.96^{1.6}}$	$67.79^{3.5}$	$34.79^{3.6}$	$\mathbf{96.45^{0.7}}$	$51.03^{3.8}$
2	✓	.	.	$64.52^{4.2}$	$86.35^{2.8}$	$\mathbf{73.83^{3.6}}$	$41.51^{3.4}$	$95.09^{1.1}$	$57.72^{3.4}$
3	.	✓	.	$58.57^{4.9}$	$69.48^{2.3}$	$63.49^{3.5}$	$46.19^{4.7}$	$93.94^{1.4}$	$61.80^{4.2}$
4	.	.	✓	$63.63^{4.1}$	$72.31^{2.9}$	$67.66^{3.4}$	$47.78^{3.5}$	$93.68^{1.6}$	$63.21^{3.1}$
5	.	✓	✓	$66.36^{4.1}$	$62.07^{2.6}$	$64.13^{3.2}$	$56.34^{3.7}$	$91.25^{1.4}$	$69.61^{3.0}$
6	✓	✓	.	$70.40^{4.3}$	$64.43^{2.6}$	$67.26^{3.2}$	$58.45^{3.7}$	$92.10^{1.3}$	$71.46^{2.9}$
7	✓	✓	✓	$\mathbf{78.84^{2.7}}$	$57.00^{2.6}$	$66.15^{2.5}$	$\mathbf{70.53^{2.4}}$	$88.90^{1.1}$	$\mathbf{78.64^{1.7}}$
8	OpinionFinder			$47.39^{3.8}$	$68.78^{2.8}$	$56.07^{3.4}$	$36.06^{3.4}$	$92.73^{1.7}$	$51.85^{3.6}$

happen not to be single-words. Second, while the "soft" precision $SP^{overlap}$ metric always has a higher value than P^{single}, the soft recall $SR^{overlap}$ is lower than R^{single}, which is counter to the usual meaning of "soft". Again, this has to do with the fact that the metrics on the right-hand-side penalize the learned models for not identifying all DSESEs, while the metrics on the left-hand-side only require that the models identify single-word DSESEs.

In summary, we have reanalyzed the problem of identifying DSESEs. By considering a more precise definition of the problem rather than the original, single-word problem, we have developed simple modifications to the learning-based approach that increase the performance from an F-measure of 51% to over 78%.

IOB	. . . **faithful/B to/I** the/O president/O ./O
IO	. . . **faithful/I to/I** the/O president/O ./O

Figure 3.6: How to encode the class variable: The IOB method and the IO method. IO is used in this work.

3.5 Identifying direct subjective expressions and expressive-subjective elements

The approach described in the previous section worked well for DSESEs (direct subjective and speech expressions), and we now wish to tackle the expressions that are purely subjective, DSEs (direct subjective expressions) and ESEs (expressive-subjective elements). However, as Table 3.6 suggests, ESEs are significantly longer than DSESEs, so an approach like the previous one based on predicting single words will not work well. Therefore, we take a different approach for both of these expression types, instead using a model that predicts multi-word prhases.

3.5.1 The class variable

A common encoding for extent-identification tasks such as named entity recognition is the so-called IOB encoding. Using IOB, each word is tagged as either Beginning an entity, being In an entity (i.e. an opinion expression), or being Outside of an entity (see Figure 3.6). While we initially used this encoding, preliminary investigation on separate development data revealed that a simpler binary encoding produces better results for opinion identification. We suspect this is because it is rarely the case in our data that two opinion entities are ad-

jacent, and so the simpler model is easier to fit. Thus, we tag each token as being either In an entity or Outside of an entity. When predicting, a sequence of consecutive tokens tagged as In constitutes a single predicted entity.

3.5.2 New feature

Since the `other` features added in the previous section turned out not to be very useful, we do not use them here, except that we do include the wider word-window. In addition to the dictionary-based features from WordNet, FrameNet, and Levin described earlier, we include an additional feature that specifically targets subjective expressions. Wilson et al. (2005b) identify a set of clues as being either strong or weak cues to the subjectivity of a clause or sentence. We identify any sequence of tokens included on this list, and then define a feature *Wilson* that returns the value '-' if the current token is not in any recognized clue, or **strong** or **weak** if the current token is in a recognized clue of that strength.

3.5.3 The learning method

The previous learning methods have all been per-token classifiers, while now we need to predict multi-word phrases. We chose to use a linear-chain conditional random field (CRF) model for all of our experiments, using the MALLET toolkit (McCallum, 2002). This discriminatively-trained sequence model has been found to perform extremely well on tagging tasks such as ours (Lafferty et al., 2001). Based on pilot experiments on development data, we chose a Gaussian prior of 0.25.

3.5.4 Evaluation

In addition to the $overlap$ predicate used in the prior section, we also report results according to the $exact(c, p)$ predicate, true just when c and p are the same spans - this yields the usual notions of precision and recall. Note that since we are now predicting not single words but extents, there is a potential issue with soft precision and recall, that the measures may drastically overestimate the system's performance. A system predicting a single entity overlapping with every token of a document would achieve 100% soft precision and recall with the overlap predicate. We can ensure the performance overestimate does not happen by measuring the average number of expressions to which each correct or predicted expression is aligned (excluding expressions not aligned at all). In our data, this does not exceed 1.13, so we can conclude these evaluation measures are behaving reasonably.

3.5.5 Baselines

For baselines, we compare to two dictionaries of subjectivity clues identified by previous work (Wilson et al., 2005b; Wiebe and Riloff, 2005). These clues were collected to help recognize subjectivity at the sentence or clause level, not at the expression level, but the clues often correspond to subjective expressions. Each clue is one to six consecutive tokens, possibly allowing for a gap, and matching either stemmed or unstemmed tokens, possibly of a fixed part of speech. In the following experiments, we report results of the *Wiebe* baseline, which predicts any sequence of tokens matching a clue from Wiebe and Riloff (2005) to be a subjective expression, and the *Wilson* baseline, using similar predictions based

on clues from Wilson et al. (2005b). When predicting DSEs using either list, we remove all clues from the list that never match a DSE in the test data, to make the baseline's precision as high as possible (although since many potentially subjective expressions are often not subjective, the precision is still quite low). We similarly trim the lists when predicting the other targets below. Apart from this trimming, the lists were not derived from the MPQA corpus. Note that the higher-performing of these two baselines, from Wilson et al. (2005b), was incorporated into the feature set used in our CRF models[19].

3.5.6 Results

Tables 3.12 and 3.13 present experimental results on identifying direct subjective expressions and expressive subjective elements in several settings, as well as presenting the two baselines for comparison purposes. We experiment with two variants of conditional random fields, one with potentials (features) for Markov order 1+0 (similar to the features in a hidden Markov model, labeled crf-1 in the tables), and one with features only for order 0 (equivalent to a maximum entropy model, labeled crf-0 in the tables). Orthogonally, we compare models trained separately on each task (classifying each token as in a DSE versus not or in an ESE versus not, labeled just DSE or ESE in the tables) to models trained to do both tasks at once (classifying each token into one of three classes: in a DSE, in an ESE, or neither[20], labeled DSE&ESE in the tables).

[19]The CRF features based on the Wilson dictionary were based on the entire dictionary, including clues not relevant for the particular problem being tested. Also, the choice to use only the Wilson dictionary and not the Wiebe for features was made during development of the model on a separate development dataset. So the model tested was in no way developed using the test data.

[20]A small number of tokens are manually annotated as being part of both a DSE and an ESE. For training, we label these tokens as DSEs, while for testing, we (impossibly) require the model

Because the baselines were not designed to distinguish between DSEs and ESEs, we run another set of experiments where the two categories are lumped together. The rows labeled DSE&ESE use the models trained previously to distinguish three categories, but are here evaluated only on the binary decision of opinion expression or not. The rows labeled DSE+ESE are trained to classify a token as I if it is in either a DSE or ESE, or O otherwise. The results of these experiments are reported in Table 3.14.

Finally, to determine the effect of the various dictionaries, we examine all combinations of the various dictionaries - WordNet, Framenet, Levin, and the clues from Wilson et al. (2005b) (to save space, we combine the two smallest dictionaries, Framenet and Levin, into one). These results, on the DSE task, are reported in Table 3.15.

3.5.7 Discussion

Tables 3.12, 3.13, and 3.14 present experiments studying the effect of varying four variables, so we will discuss these each in turn - Markov order, three-versus-two class training, fixed rules versus learning, and the varying target class. All statistical significance results are computed using a paired t-test between the metric values on ten cross-validation folds, with a threshold of $p < 0.05$.

The order-0 models outperform the order-1 models in Tables 3.12, 3.13, and 3.14 according to overlap F-measure and recall, but by exact F-measure and either precision metric, the order-1 models are superior[21]. The creators of the

to annotate both entities.

[21] All differences are statistically significant, except the difference in exact F-measure between

dataset state "we did not attempt to define rules for boundary agreement in the annotation instructions, nor was boundary agreement stressed during training." (Wiebe et al., 2005, page 35). For example, whether a DSE ought to be annotated as "firmly said" or just "said" is left up to the annotator. Therefore, we hypothesize that the model with greater capacity (the order 0+1) may overfit to the somewhat inconsistent training data.

Looking at Tables 3.12 and 3.13, the differences between the two-class training (target class versus non-opinion) and three-class training (one model that distinguishes DSE versus ESE versus non-opinion) are small (0-2% absolute), and in many cases, not statistically significant [22]. As with the Markov order, we find a difference in recall versus precision – the two-way training yields slightly greater precision, and the three-way training yields slightly better recall. I do not have an explanation for this result.

All of the learned models in Tables 3.12 and 3.13, significantly outperform the rule-based baselines according to all metrics, except for recall on the ESE task, confirming our hypothesis that learning methods were necessary for this task. The slightly higher recall of the baselines on the ESE task is likely due to the very large number of different phrases that are used as ESEs; note that the baseline precision is quite low on this task, even when the baselines were allowed to "cheat" at precision as described in Section 3.5.5. Table 3.14 presents results which are even more fair to the baselines, not requiring them to make the DSE-versus-ESE distinction which they cannot do, which does raise their precision, but they are still significantly outperformed by the learning methods[23].

crf-1-ESE and crf-0-ESE.

[22] $p \not< 0.05$ for the following metrics and settings: exact recall and F for crf-1-DSE, exact F for crf-0-DSE, overlap precision, exact recall, and exact F for crf-1-ESE, all exact metrics for crf-0-ESE, overlap precision for crf-1-both, and both Fs and exact recall for crf-0-both.

[23] According to the same test, except for the 1% difference between the Wilson baseline and

Finally, while the results are not strictly comparable, we note that the results on the DSE task, Table 3.12 are superior to those of the ESE task, Table 3.13 The interannotator agreement results for these tasks are relatively low; 0.75 for DSEs and 0.72 for ESEs, according to a metric very close to overlap F-measure[24]. Our results are thus quite close to the human performance level for both of these tasks.

The ablation results in Table 3.15 indicate that the WordNet features are by far the most helpful. The other two dictionary sets are individually useful [25] (with the Wilson features being more useful than the Levin/Framenet ones), but beyond the WordNet features the others make no significant difference[26]. This is interesting, especially since the WordNet dictionary is entirely general, and the Wilson dictionary was built specifically for the task of recognizing subjectivity. Ablation tables for the other two targets (ESEs and DSE&ESE) look similar and are omitted.

In looking at errors on the development data, we found several causes which we could potentially fix to yield higher performance. The category of DSEs includes speech events like "said" or "a statement," but not all occurrences of speech events are DSEs, since some are simply statements of objective fact. Adding features to help the model make this distinction should help performance. Also, as others have observed, expressions of subjectivity tend to clus-

the crf-1-DSE&ESE model, which is not significant.

[24] Using the agr statistic, the interannotator agreement for ESEs on the MPQA data is 0.72 (Wiebe et al., 2005, page 36), and for DSEs is 0.75 (Theresa Wilson, personal communication). agr is the arithmetic (rather than harmonic) mean of overlap recall and precision between two annotators.

[25] Providing a statistically significant improvement in recall, with no significant difference in overlap precision but a small significant loss according to exact precision.

[26] Except for adding the Wilson feature to either model already containing WordNet, which does result in a small ($< 1\%$) but statistically significant improvement according to overlap F and overlap recall.

ter, so incorporating features based on the density of expressions might help as well (Wiebe and Wilson, 2002).

Table 3.12: Results for identifying direct subjective expressions. Superscripts designate one standard deviation.

	overlap			exact		
method	recall	precision	F	recall	precision	F
Wiebe baseline	$45.69^{2.4}$	$31.10^{2.5}$	$36.97^{2.3}$	$21.52^{1.8}$	$13.91^{1.4}$	$16.87^{1.4}$
Wilson baseline	$55.15^{2.2}$	$30.73^{1.9}$	$39.44^{1.9}$	$25.65^{1.7}$	$13.32^{1.0}$	$17.52^{1.2}$
crf-1-DSE	$60.22^{1.8}$	$\mathbf{79.34^{3.2}}$	$68.44^{2.0}$	$42.65^{2.9}$	$\mathbf{57.65^{2.8}}$	$\mathbf{49.01^{2.8}}$
crf-1-DSE&ESE	$62.73^{2.3}$	$77.99^{3.1}$	$69.51^{2.4}$	$\mathbf{43.23^{2.9}}$	$55.38^{2.8}$	$48.54^{2.8}$
crf-0-DSE	$65.48^{2.0}$	$74.85^{3.5}$	$69.83^{2.4}$	$39.95^{2.4}$	$44.52^{2.2}$	$42.10^{2.2}$
crf-0-DSE&ESE	$\mathbf{69.22^{1.8}}$	$72.16^{3.2}$	$\mathbf{70.65^{2.4}}$	$42.13^{2.3}$	$42.69^{2.5}$	$42.40^{2.3}$

Table 3.13: Results for identifying expressive subjective elements. Superscripts designate one standard deviation.

	overlap			exact		
method	recall	precision	F	recall	precision	F
Wiebe baseline	$56.36^{2.1}$	$43.03^{4.5}$	$48.66^{3.3}$	$15.09^{1.1}$	$9.91^{1.6}$	$11.92^{1.4}$
Wilson baseline	$\mathbf{66.10^{2.6}}$	$40.94^{4.7}$	$50.38^{4.0}$	$17.23^{1.9}$	$8.76^{1.5}$	$11.56^{1.6}$
crf-1-ESE	$46.36^{4.1}$	$\mathbf{75.21^{6.6}}$	$57.14^{3.6}$	$15.11^{1.7}$	$\mathbf{27.28^{2.3}}$	$19.35^{1.5}$
crf-1-DSE&ESE	$48.79^{3.2}$	$74.09^{6.7}$	$58.70^{3.7}$	$15.58^{1.1}$	$26.18^{2.1}$	$\mathbf{19.46^{0.8}}$
crf-0-ESE	$61.22^{3.4}$	$64.84^{5.4}$	$62.82^{3.3}$	$18.31^{1.7}$	$17.11^{3.0}$	$17.61^{2.2}$
crf-0-DSE&ESE	$63.46^{3.3}$	$63.76^{5.7}$	$\mathbf{63.43^{3.3}}$	$\mathbf{18.96^{1.4}}$	$16.79^{2.5}$	$17.73^{1.8}$

Table 3.14: Results for identifying expressions that are either DSEs or ESEs. Superscripts designate one standard deviation. DSE&ESE indicates a model trained to make a three-way distinction among DSEs, ESEs, and other tokens, while DSE+ESE indicates a model trained to make a two-way distinction between DSEs or ESEs and all other tokens.

	overlap			exact		
method	recall	precision	F	recall	precision	F
Wiebe baseline	$51.59^{2.0}$	$61.35^{4.6}$	$55.99^{2.8}$	$17.70^{0.8}$	$19.61^{2.0}$	$18.58^{1.2}$
Wilson baseline	$61.23^{2.1}$	$58.48^{4.7}$	$59.73^{3.1}$	$20.61^{1.4}$	$17.68^{1.5}$	$19.00^{1.3}$
crf-1-DSE+ESE	$64.77^{2.2}$	$81.33^{4.4}$	$72.03^{2.2}$	$26.68^{2.7}$	$39.23^{2.6}$	$31.70^{2.4}$
crf-1-DSE&ESE	$62.36^{2.1}$	$\mathbf{81.90^{4.1}}$	$70.74^{2.2}$	$28.24^{2.7}$	$\mathbf{42.64^{1.9}}$	$\mathbf{33.92^{2.3}}$
crf-0-DSE+ESE	$\mathbf{74.70^{2.5}}$	$71.64^{4.5}$	$\mathbf{73.05^{2.8}}$	$\mathbf{30.93^{2.5}}$	$28.20^{2.3}$	$29.44^{2.0}$
crf-0-DSE&ESE	$71.91^{2.2}$	$74.04^{4.5}$	$72.88^{2.6}$	$30.30^{2.2}$	$29.64^{2.3}$	$29.91^{1.8}$

Table 3.15: Results for feature ablation for identifying DSEs. All rows include the lexical features and the syntactic features. The bottom line represents the same model as CRF-0-DSE&ESE in Table 3.12.

feature set			overlap			exact		
Levin/FrameNet	Wilson	WordNet	recall	precision	F	recall	precision	F
.	.	.	$47.14^{2.6}$	$70.91^{4.4}$	$56.60^{3.0}$	$30.55^{2.7}$	$\mathbf{45.12^{3.1}}$	$36.41^{2.8}$
✓	.	.	$50.57^{3.1}$	$70.51^{4.1}$	$58.86^{3.3}$	$32.20^{3.1}$	$44.11^{3.3}$	$37.20^{3.1}$
.	✓	.	$54.92^{2.4}$	$70.73^{4.0}$	$61.81^{2.9}$	$34.61^{2.5}$	$43.60^{2.9}$	$38.57^{2.5}$
✓	✓	.	$57.21^{2.6}$	$70.79^{4.1}$	$63.26^{3.0}$	$35.77^{2.4}$	$43.42^{2.8}$	$39.21^{2.5}$
.	.	✓	$68.29^{2.4}$	$71.82^{3.5}$	$70.00^{2.8}$	$41.80^{2.5}$	$42.71^{2.5}$	$42.24^{2.4}$
.	✓	✓	$68.93^{2.1}$	$72.06^{3.3}$	$70.45^{2.6}$	$42.10^{2.5}$	$42.71^{2.6}$	$\mathbf{42.40^{2.5}}$
✓	.	✓	$68.48^{2.4}$	$71.87^{3.3}$	$70.13^{2.8}$	$41.92^{2.2}$	$42.80^{2.5}$	$42.34^{2.3}$
✓	✓	✓	$\mathbf{69.22^{1.8}}$	$\mathbf{72.16^{3.2}}$	$\mathbf{70.65^{2.4}}$	$\mathbf{42.13^{2.3}}$	$42.69^{2.5}$	$\mathbf{42.40^{2.3}}$

3.5.8 Conclusions

Extracting information about subjectivity is an area of great interest to a variety of public and private interests. We have argued that successfully pursuing this research will require the same expression-level identification as in factual information extraction. Our method is the first to directly approach the task of extracting these expressions, achieving performance within 5% of human inter-annotator agreement.

CHAPTER 4

DETERMINING THE HIERARCHICAL STRUCTURE OF OPINIONS[1]

The next step toward our goal of creating a system for fine-grained information extraction of opinions is to accurately identify the hierarchical structure of direct subjective and speech expressions. Consider for example, the following sentences (in which direct subjective expressions are denoted in **bold**, speech expressions are underlined, and sources are denoted in *italics*):

1. *Charlie* was **angry** at *Alice*'s **claim** that *Bob* was **unhappy**.
2. *Philip Clapp*, president of the National Environment Trust, sums up well the general thrust of the **reaction** of *environmental movements*: "There is no reason at all to believe that the polluters are suddenly going to become reasonable."

Direct subjective expressions in Sentence 1 describe the emotions or opinion of three sources: Charlie's anger, Bob's unhappiness, and Alice's belief. Direct subjective expressions in Sentence 2, on the other hand, introduce the explicit opinion of one source, i.e. the reaction of the environmental movements.

In this chapter, we focus on the *filtering* of information through sources. By filtering, we mean the fact that while a speech event or subjective expression passes information to the reader, the information is affected by the biases and perspective of the source of the speech event or expression. Both direct subjective expressions and speech expressions perform filtering in these examples. The reaction of the environmental movements is filtered by Clapp's summarization, which, in turn, is filtered by the writer's choice of quotation. In addition, the fact that Bob was unhappy is filtered through Alice's claim, which, in turn,

[1]This chapter is adapted from Breck and Cardie (2004).

is filtered by the writer's choice of words for the sentence. Similarly, it is only according to the writer that Charlie is angry[2].

Given sentences 1 and 2 and their DSESEs (direct subjective and speech events), for example, we will present methods that produce the structures shown in Figure 4.1, which represent the multi-stage information filtering that should be taken into account in the interpretation of the text.

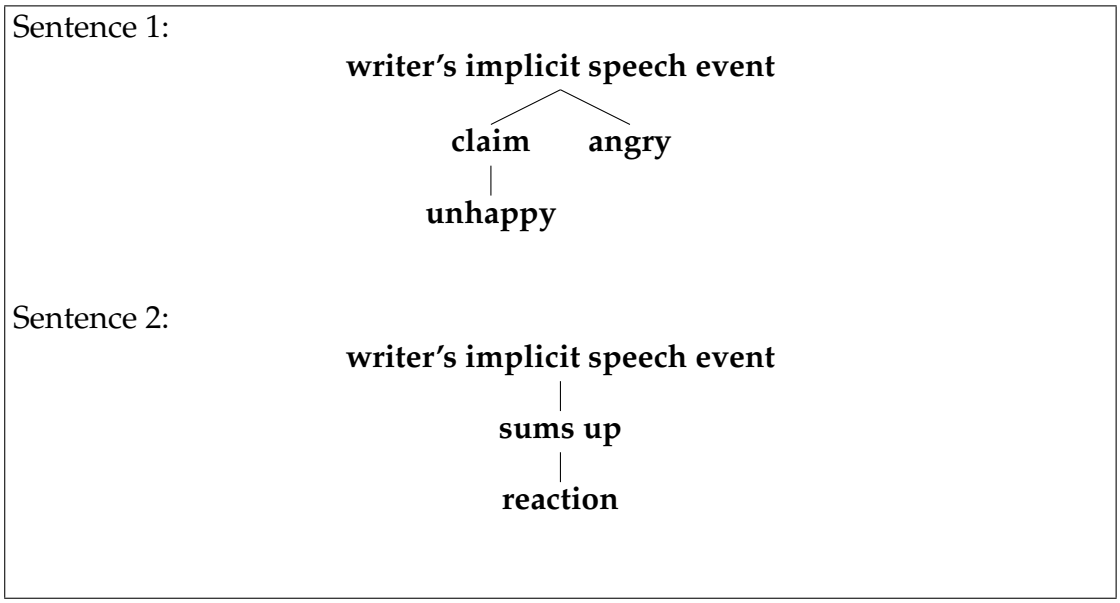

Figure 4.1: Hierarchical structure of the direct subjective and speech expressions in sentences 1 and 2

We propose a supervised machine learning approach to the problem that relies on a small set of syntactically-based features. We compare the approach to two heuristic-based baselines — one that simply assumes that every DSESE is filtered only through the writer (the `writer` baseline), and a second that is based on syntactic dominance relations in the associated parse tree (the `syntax` baseline). In an evaluation using the opinion-annotated MPQA corpus (described in Section 3.2), the learning-based approach achieves an accuracy of

[2]As with any linguistic construction, there may be multiple readings of these sentences. We present here analyses for the most salient reading according to our judgment.

78.30%, significantly higher than both the `writer` baseline (65.57%) and the `syntax` baseline (71.64%).

4.1 Related Work

While no approach has been proposed for the specific task here, there is some relevant prior work. Wiebe (1994) describes an algorithm to track the change of "point of view" in narrative text (fiction). That is, the "writer" of one sentence may not correspond to the writer of the next sentence. Although this is not as frequent in newswire text as in fiction, it may occur, and is relevant because in this chapter, we only find the filtering structure for each sentence, leaving open the question of whether the writer or speaker of one sentence is the same as the next.

Bergler et al (Anick and Bergler, 1991; Bergler, 1991; Bergler, 1993) have examined several aspects of the lexical semantics of reporting verbs using the framework of the generative lexicon theory. Briefly, the generative lexicon asserts that the lexicon isn't simply a flat mapping from words to some knowledge representat ion, but rather that the lexicon has significant structure. In Anick and Bergler (1991), the authors focus on treating selectional restricti on violations in this framework (rather than using pragmatics). In particular, they investigate logical metonymies of the source of reporting verbs - for examp le, a reporting verb's subject, while theoretically an agent, may be expressed using a city, an organization, a building, etc. Bergler (1991) relates these metonymies to the notion of a "semantic collocation." A semantic collocation is similar to a collocation, but in semantic space, so *insist* requires that some proposition oppos-

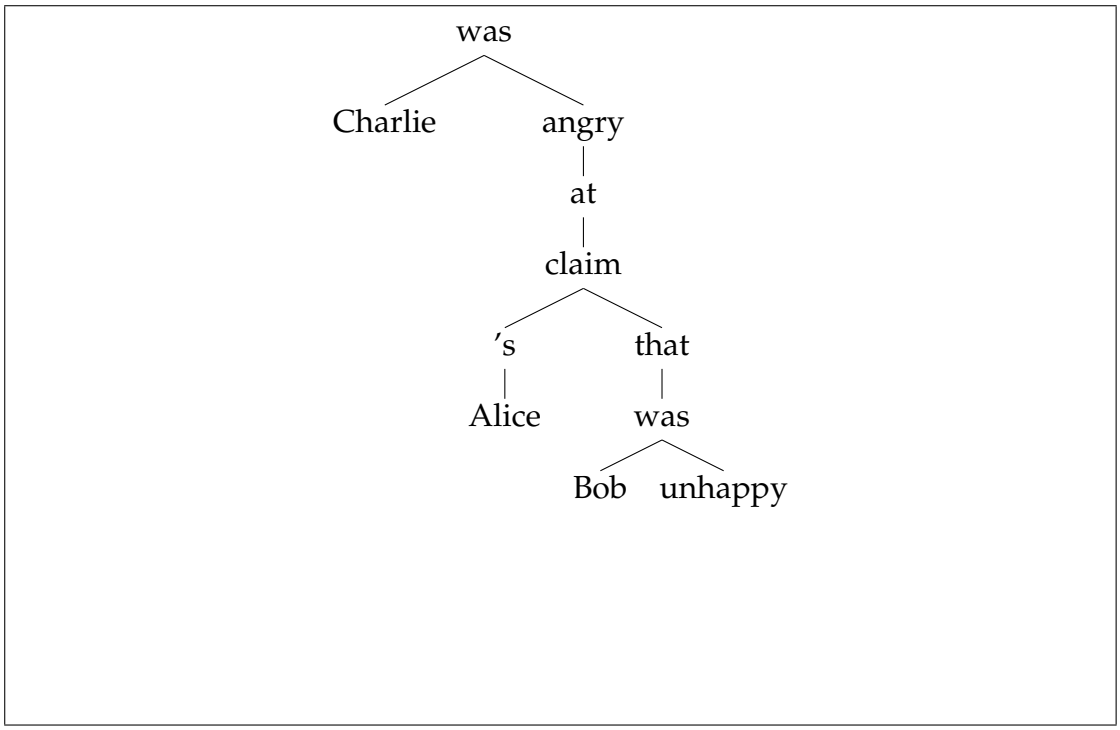

Figure 4.2: Dependency parse of sentence 1 according to the Collins parser.

ing its object be nearby. Bergler argues such collocations form part of the lexical semantics of reporting verbs, and supports this with a corpus study of TIME magazine. Bergler (1993) discusses the "semantic dimensions" of the semantic field of reporting verbs. These dimensions, such as polarity of the complement or official vs formal content, are what distinguish members of the field from one a nother.

Gerard (2000) proposes a computational model of the reader of a news article. Her model provides for multiple levels of hierarchical beliefs, such as the nesting of a primary source's belief within that of a reporter.

4.2 The Approach

Our task is to find the hierarchical structure among the DSESEs in individual sentences. One's first impression might be that this structure should be obvious from the syntax: one DSESE should filter another roughly when it dominates the other in a dependency parse. This heuristic, for example, would succeed for "claim" and "unhappy" in sentence 1, whose DSESE structure is given in Figure 4.1 and parse structure (as produced by the Collins parser) in Figure 4.2.[3]

Even in sentence 1, though, we can see that the problem is more complex: "angry" dominates "claim" in the parse tree, but does not filter it. An analysis of the `syntax` heuristic on our training data uncovered numerous additional sources of error. Therefore, rather than trying to handcraft a more complex collection of heuristics, we chose to adopt a supervised machine learning approach that relies on features identified in this analysis. Our approach has two steps:

1. Train a binary classifier to make pairwise decisions as to whether a given DSESE is the immediate parent of another.
2. Combine the decisions of the classifier decisions to find the hierarchical information-filtering structure of all DSESEs in a sentence. This is done using a simple approach described below.

We assume that we have a training corpus of sentences, annotated with DSESEs and their hierarchical DSESE structure (Section 4.3 describes the corpus). Training instances for the binary classifier are all pairs of DSESEs from the

[3]For this heuristic and the features that follow, we will speak of the DSESEs as if they had a position in the parse tree. However, since DSESEs are often multiple words, and do not necessarily form a constituent, this is not entirely accurate. The parse node corresponding to a DSESE will be the highest node in the dependency parse corresponding to a word in the DSESE. We consider the writer's implicit DSESE to correspond to the root of the parse.

Table 4.1: Training instances generated from sentence 1

target t	parent p	class
claim	writer	1
angry	writer	1
unhappy	claim	1
claim	angry	0
claim	unhappy	0
angry	claim	0
angry	unhappy	0
unhappy	writer	0
unhappy	angry	0

1: **for** Each DSESE d_0 in the sentence **do**
2: **for** Each other DSESE d_1 in the sentence **do**
3: confidence(d_0, d_1) $\leftarrow$ binary_classifier(d_0, d_1).
4: **end for**
5: parent(d_0) $\leftarrow$ argmax$_{d_1} confidence(d_0, d_1)$
6: **end for**

Figure 4.3: Algorithm for identifying hierarchical DSESE structure

same sentence, $\langle t, p \rangle$[4]. We assign a class value of 1 to a training instance if p is the immediate parent of t in the manually annotated hierarchical structure for the sentence, and 0 otherwise. For sentence 1, the training instances generated are listed in Table 4.1. The features used to describe each training instance are explained below.

During testing, we construct the hierarchical DSESE structure of an entire sentence as described in Figure 4.3. For each DSESE in the sentence, ask the binary classifier to judge each other DSESE as a potential parent, and choose the DSESE with the highest confidence. There is an ambiguity if the classifier assigns the same confidence to two potential parents. For evaluation purposes, we consider the classifier's response incorrect if any of the highest-scoring po-

[4]We skip sentences where there is no decision to make (sentences with zero or one non-writer DSESE). Since the writer DSESE is the root of every structure, we do not generate instances with the writer DSESE in the t position.

tential parents are incorrect. Finally, join these immediate-parent links to form a tree. The directed graph resulting from flawed automatic predictions might not be a tree (i.e. it might be cyclic and disconnected). Since this occurs very rarely (5 out of 9808 sentences on the test data), we do not attempt to correct any non-tree graphs.

We considered other variations of this approach. In particular, we found that assigning the class value of a training case $\langle t, p \rangle$ based on whether p was an ancestor (rather than immediate parent) of t was an easier task for our model to learn (given the features below). However, reconstructing the hierarchical structure from this information is more complex than doing so from the immediate-parent decisions.

One might also try comparing pairs of potential parents for a given DSESE, or other more direct means of ranking potential parents. We chose what seemed to be the simplest method for this first attempt at the problem.

Features

Here we motivate and describe the 21 features used in our model. The values of these features for three instances generated from sentence 1 are given in Table 4.2. Unless otherwise stated, all features are binary (1 if the described condition is true, 0 otherwise).

Parse-based features (6). Based on the performance of the parse-based heuristic, we include a p-`dominates`-t feature in our feature set. To compensate for parse errors, however, we also include a variant of this that is 1 if the parent of

p dominates t (p's-parent-dominates-t).

Many filtering expressions filter DSESEs that occur in their complements, but not in adjuncts. Therefore, we add variants of the previous two syntax-based features that denote whether the parent node dominates t, but only if the first dependency relation is an object relation (p's-object-dominates-t, p's-parent's-object-dominates-t).

For similar reasons, we include a feature calculating the domination relation based on a partial parse. Consider the following sentence:

3. He was **criticized** more than **recognized** for his policy.

One of "criticized" or "recognized" will be the root of this dependency parse, thus dominating the other, and suggesting (incorrectly) that it filters the other DSESE. Because a partial parse does not attach all constituents, such spurious dominations are eliminated. The partial parse feature p-dominates-t-partial-parse is 1 for fewer instances than p-dominates-t, but it is more indicative of a positive instance when it is 1.

So that the model can adjust when the parse is not present, we include a feature parse-failed that is 1 for all instances generated from sentences on which the parser failed.

Positional features (4). Forcing the model to decide whether p is the parent of t without knowledge of the other DSESEs in the sentence is somewhat artificial. We therefore include several features that encode the relative position of p and t in the sentence. Specifically, we add a feature p-is-root that is 1 if p is the root

of the parse (and similarly for t, t-is-root). We also include a feature p-pos giving the ordinal position of p among the DSESEs in the sentence, relative to t (-1 means p is the DSESE that immediately precedes t, 1 means immediately following, and so forth). To allow the model to vary when there are more potential parents to choose from, we include a feature giving the total number of DSESEs in the sentence (number-DSESEs-in-sentence).

Special parents and lexical features (5). Some particular DSESEs are special, so we specify indicator features for four types of parents: the writer DSESE (p-is-writer) , and the lexical items "said" (the most common non-writer DSESE) and "according to" (p-is-``said'', $p$-is-``according to''). "According to" is special because it is generally not very high in the parse, but semantically tends to filter everything else in the sentence.

In addition, we include as features the part of speech of p and t (reduced to noun, verb, adjective, adverb, or other), since intuitively we expected distinct parts of speech to behave differently in their filtering (p-part-of-speech, t-part-of-speech).

Genre-specific features (6). Finally, journalistic writing contains a few special forms that are not always parsed accurately. Examples are:

4. "Alice **disagrees** with me," Bob **argued**.
5. Charlie, she **noted**, **dislikes** Chinese food.

The parser may not recognize that "noted" and "argued" should dominate all other DSESEs in sentences 4 and 5, so we attempt to recognize when a sen-

Table 4.2: Features for three training instances from sentence 1. The journalistic features are not included because the sentence does not match the patterns (see Section 4.2).

feature for $\langle t, p \rangle$	$\langle$claim, writer$\rangle$	$\langle$unhappy, claim$\rangle$	$\langle$unhappy, angered$\rangle$
class	1	1	0
p-dominates-t	1	1	1
p's-parent-dominates-t	1	1	1
p's-object-dominates-t	1	1	0
p's-parent's-object-dominates-t	1	1	1
p-dominates-t-partial-parse	1	0	0
parse-failed	0	0	0
p-is-root	1	0	1
t-is-root	0	0	0
p-pos	-1	-1	1
number-DSESEs-in-sentence	4	4	4
p-is-writer	1	0	0
p-is-``said''	0	0	0
p-is-``according to''	0	0	0
p-part-of-speech	-	N	V
t-part-of-speech	N	J	J
p-is-last	0	0	0

tence falls into one of these two patterns. For $\langle$disagrees, argued$\rangle$ generated from sentence 4, features p-pattern-1 and t-pattern-1 would be 1, while for $\langle$dislikes, noted$\rangle$ generated from sentence 5, feature p-pattern-2 would be 1. We also add features (p-in-quote, t-in-quote) that denote whether the DSESE in question falls between matching quote marks. Finally, a simple feature (p-is-last) indicates whether p is the last word in the sentence.

Resources

We rely on a variety of resources to generate our features. The corpus (see Section 4.3) is distributed with annotations for sentence breaks, tokenization, and

part of speech information automatically generated by the GATE toolkit (Cunningham et al., 2002). GATE's sentences sometimes extend across paragraph boundaries, which seems never to be warranted. Inaccurately joining sentences has the effect of adding more noise to our problem, so we split GATE's sentences at paragraph boundaries, and introduce writer DSESEs for the newly created sentences. For parsing we use the Collins (1999) parser. We convert the parse to a dependency format that makes some of our features simpler using a method similar to the one described in Xia and Palmer (2001). We also employ a method from Adam Lopez at the University of Maryland to find grammatical relationships between words (subject, object, etc.). For partial parses, we employ CASS (Abney, 1997). Finally, we use a simple finite-state recognizer to identify (possibly nested) quoted phrases.

For classifier construction, we use the IND package (Buntine, 1993) to train decision trees (we use the `mml` tree style, a minimum message length criterion with Bayesian smoothing).

4.3 Data Description

Again, the data for these experiments come from version 1.1 of the MPQA corpus described in Section 3.2. The corpus consists of 535 newswire documents, of which we used 66 (1375 sentences) for developing the heuristics and features, while keeping the remaining 469 (9808 sentences) blind (used for 10-fold cross-validation). These numbers differ from those in the previous chapter because the experiments reported here were conducted earlier, and at the time, only 66 documents were designated as "development."

Although the MPQA corpus provides annotations for all DSESEs, it does not provide annotations to denote directly their hierarchical structure within a sentence. This structure must be extracted from an attribute of each DSESE annotation, which lists the DSESEs' direct and indirect sources. For example, the "source chain" for "unhappy" in sentence 1, would be (*writer*, *Alice*, *Bob*), where *writer*, *Alice*, and *Bob* are arbitrary identifiers assigned by the human annotator to denote each source. The source chains allow us to automatically recover the hierarchical structure of the DSESEs: the parent of a DSESE with source chain $(s_0, s_1, \ldots s_{n-1}, s_n)$ is the DSESE with source chain $(s_0, s_1, \ldots s_{n-1})$. Unfortunately, ambiguities can arise. Consider the following sentence:

6. *Bob* **said**, "Welcome!" and then *he* **told** us that *Mary* **was happy**.

Because the annotators also performed coreference resolution on sources, "said" and "told" have the source chain (*writer*, *Bob*), while "was happy" has the source chain (*writer*, *Bob*, *Mary*). It is therefore not clear from the manual annotations whether "was happy" should have "told" or "said" as its parent. 5.82% of the DSESEs have ambiguous parentage (i.e. the recovery step finds a set of parents $P(DSESE)$ with $|P(DSESE)| > 1$). For training, we assign a class value of 1 to all instances $\langle t, p \rangle, p \in P(t)$. For testing, if an algorithm attaches $DSESE$ to any element of $P(DSESE)$, we score the link as correct (see Section 4.3.1). Since ultimately our goal is to find the sources through which information is filtered (rather than the DSESEs through which it is filtered), we believe this is justified.

For training and testing, we used only those sentences that contain at least two non-writer DSESEs – for all other sentences, there is only one way to construct the hierarchical structure. Under certain circumstances, such as

Table 4.3: Breakdown of number of direct subjective and speech expressions per sentence

number of DSESEs	number of sentences
1	3612
2	3256
3	1810
4	778
5	239
>5	113

paragraph-long quotes, the writer of a sentence will not be the same as the writer of a document. In such sentences, the MPQA corpus contains additional DSESEs for any other sources besides the writer of the document. Since we are concerned in this work only with one sentence at a time, we discard all such implicit DSESEs besides the writer of the sentence. Also, in a few cases, more than one DSESE in a sentence was marked as having the writer as its source. We believe this to be an error and so discarded all but one writer DSESE. Again, Table 4.3 presents a breakdown (for the test set) of the number of DSESEs per sentence – thus we only use approximately one-third of all the sentences in the corpus.

4.3.1 Evaluation

How do we evaluate the performance of an automatic method of determining the hierarchical structure of DSESEs? Lin (1995) proposes a method for evaluating dependency parses: the score for a sentence is the fraction of correct parent links identified; the score for the corpus is the average sentence score. Formally, the score for a method evaluated on the entire corpus (`Lin`) is

Table 4.4: Performance on test data. `Lin` is Lin's dependency score, `perfect` is the fraction of sentences whose structure was identified perfectly, and `binary` is the performance of the binary classifier (broken down for positive and negative instances). "Size" is the number of sentences or DSESE pairs.

metric	size	`writer`	`syntax`	our method
`Lin`	2940	65.57%	71.64%	78.30%
`perfect`	2940	36.02%	45.37%	54.52%
`binary`	21933	73.20%	77.73%	82.12%
`binary` +	7882	60.63%	66.94%	70.35%
`binary` −	14051	80.24%	83.78%	88.72%

$$\texttt{Lin}(m) = \frac{\sum_{s \in S} \frac{|\{d | d \in N(s) \wedge m(d) = c(d)\}|}{|N(s)|}}{|S|}$$

where S is the set of all sentences in the corpus, $N(s)$ is the set of non-writer DSESEs in sentence s, $c(d)$ is the correct parent of d, and $m(d)$ is the automatically identified parent of d, according to method m.

We also present results using two other (related) metrics. The `perfect` metric measures the fraction of sentences whose structure is determined entirely correctly. The `binary` is the accuracy of the binary classifier (with a 0.5 threshold) on the instances created from the test corpus. We also report the performance on positive and negative instances.

4.4 Results

We compare the learning-based approach (denoted "our method" in the tables below) to the heuristic-based approaches introduced in Section 4.2 — the `writer` baseline assumes that all DSESEs are attached to the writer's implicit DSESE; the `syntax` baseline is the parse-based heuristic that relies solely on

Table 4.5: Performance by number of DSESEs per sentence

#DSESEs	# sentences	writer	syntax	our method
3	1810	70.88%	75.41%	81.82%
4	778	59.17%	67.82%	74.38%
5	239	53.87%	61.92%	68.93%
>5	113	49.31%	58.03%	68.68%

the dominance relation – that is, the syntax baseline attaches a DSESE to the DSESE most immediately dominating it in the dependency tree. If no other DSESE dominates it, a DSESE is attached to the writer's DSESE.

We use 10-fold cross-validation on the evaluation data to generate training and test data (although the heuristics, of course, do not require training). The results of the decision tree method and the two heuristics are presented in Table 4.4.

4.5 Discussion

Encouragingly, our machine learning method uniformly and significantly[5] outperforms the two heuristic methods, on all metrics and in sentences with any number of DSESEs. The difference is most striking in the perfect metric, which is perhaps the most intuitive. Also, the syntax baseline significantly[6] outperforms the writer baseline, confirming our intuitions that syntax is important in this task.

As the binary classifer sees many more negative instances than positive, it

[5] $p < 0.01$, using an approximate randomization test with 9,999 trials. See (Eisner, 1996, page 17) and (Chinchor et al., 1993, pages 430-433) for descriptions of this method.

[6] Using the same test as above, $p < 0.01$, except for the performance on sentences with more than 5 DSESEs, because of the small amount of data, where $p < 0.02$.

is unsurprising that its performance is much better on negative instances. This suggests that we might benefit from machine learning methods for dealing with unbalanced datasets.

Examining the errors of the machine learning system on the development set, we see that for half of the DSESEs with erroneously identified parents, the predicted parent is either the writer's DSESE, or a DSESE like "said" in sentences 4 and 5 having scope over the entire sentence. For example,

7. "Our **concern** is whether persons used to the role of policy implementors can objectively **assess and critique** executive policies which impinge on human rights," **said** Ramdas.

Our model chose the parent of "assess and critique" to be "said" rather than "concern." We also see from Table 4.5 that the model performs more poorly on sentences with more DSESEs. We believe that this reflects a weakness in our choice to combine binary decisions, because the model has learned that in general, a "said" or writer's DSESE (near the root of the structure) is likely to be the parent, while it sees many fewer examples of DSESEs such as "concern" that lie in the middle of the tree.

Although we have ignored the distinction throughout this chapter, error analysis suggests speech event DSESEs behave differently than private state DSESEs with respect to how closely syntax reflects their hierarchical structure. It may behoove us to add features to allow the model to take this into account. Other sources of error include erroneous sentence boundary detection, parenthetical statements (which the parser does not treat correctly for our purposes), other parse errors, partial quotations, and some errors in the annotation.

Examining the learned trees is difficult because of their size, but looking at one tree to depth three reveals a fairly intuitive model. Ignoring the probabilities, the tree decides p is the parent of t if and only if p is the writer's DSESE (and t is not in quotation marks), or if p is the word "said." For all the trees learned, the root feature was either the writer DSESE test or the partial-parse-based domination feature.

CHAPTER 5

A LIGHTWEIGHT SYSTEM FOR NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING WORKFLOWS[1]

Running experiments in natural language processing and machine learning typically involves a complicated network of programs. One program might extract data from a raw corpus, others might pre-process it with various linguistic tools before finally the main program being tested is run. Further programs must evaluate the output, and produce graphs and tables for inclusion in papers and presentations. All of these steps can be run by hand, but a more typical approach is to automate them using tools such as UNIX shell scripts. We argue that any approach should satisfy a number of basic criteria.

Reproducibility At some future time, the original researcher or other researchers ought to be able to re-run the set of experiments and produce identical results[2]. Such reproducibility is a cornerstone of scientific research, and ought in principle to be easier in our discipline than in a field requiring physical measurements such as physics or chemistry.

Simplicity We want to create a system that we and other researchers will find easy to use. A system that requires significant overhead before any experiment can be run can limit a researcher's ability to quickly and easily try out new ideas.

Support for a realistic life-cycle of experiments A typical experiment evolves in structure as it goes along - the researcher may choose partway through to add

[1] This chapter is adapted from Breck (2008).

[2] User input presents difficulties that we will not discuss.

new datasets, new ranges of parameters, or new sets of models to test. Moreover, a computational experiment rarely works correctly the first time. Components break for various reasons, a tool may not perform as expected, and so forth. A usable tool must be simple to use in the face of such repeated re-execution.

Support for good software engineering Whether writing shell scripts, makefiles, or Java, one is writing code, and software engineering concerns apply. One key principle is modularity, that different parts of a program should be cleanly separated. Another is generality, creating solutions that are re-usable in different specific cases. A usable tool must encourage good software engineering.

Support for the combinatorial nature of our experiments Experiments in natural language processing and machine learning typically compare different datasets, different models, different feature sets, different training regimes, and train and test on a number of cross-validation folds. This produces a very large number of files that any system must handle in a clean way.

In this chapter, we present `zymake`[3], and argue that it is superior to several alternatives for the task of automating the steps in running an experiment in natural language processing or machine learning.

[3]Any name consisting of a single letter followed by `make` already refers to an existing software project. `zymake` is the first pronounceable name consisting of a two letter prefix to `make`, starting from the end of the alphabet. I pronounce "zy-" as in "zydeco."

Table 5.1: Training regimes

training regime	classes
two-way distinction	DSE vs ESE+O
two-way distinction	ESE vs DSE+O
three-way distinction	DSE vs ESE vs O
baseline comparison	DSE+ESE vs O

5.1 A Typical NLP Experiment

As a running example, we will use the experiments described in Section 3.5. To recap, the task is one of entity identification — we have a large dataset in which two different types of opinion entities are tagged, DSEs, and ESEs. We will use a sequence-based learning algorithm to model the entities, but we want to investigate the relationship between the two types. In particular, will it be preferable to learn a single model that predicts both DSEs and ESEs, or two separate models, one predicting DSEs, and one predicting ESEs? The former case makes a three-way distinction between DSEs, ESEs, and entities of type O, all other words. The latter two models make a distinction between DSEs and both other types or between ESEs and both other types. Furthermore, the list-based baselines to which we wish to compare do not distinguish between DSEs and ESEs, so we also need a model that just predicts entities to be either DSEs or ESEs, versus the background O. These four training regimes are summarized in Table 5.1.

Given one of these training regimes, the model is trained and tested using 10-fold cross-validation, and the result is evaluated using precision and recall. The evaluation is conducted separately for DSEs, for ESEs, and for predicting the union of both classes.

```
for fold in `seq 0 9`; do
  extract-test-data $fold raw-data $fold.test
  for class in DSE ESE DSE+ESE; do
    extract-2way-training $fold raw-data $class \
       > $fold.$class.train
    train $fold.$class.train > $fold.$class.model
    predict $fold.$class.model $fold.test \
       > $fold.$class.out
    prep-eval-2way $fold.$class.out > $fold.eval-in
    eval $class $fold.$class.eval-in > $fold.$class.eval
  done
  extract-3way-training $fold raw-data > $fold.3way.train
  train $fold.3way.train > $fold.3way.model
  predict $fold.3way.model $fold.test > $fold.3way.out
  for class in DSE ESE DSE+ESE; do
    prep-eval-3way $class $fold.3way.out \
       > $fold.3way.$class.eval-in
    eval $class $fold.3way.$class.eval-in \
       > $fold.3way.$class.eval
  done
done
```

Figure 5.1: A shell script

5.1.1 Approach 1: A UNIX Shell Script

Many researchers use UNIX shell scripts to co-ordinate experiments[4]. Figure 5.1 presents a potential shell script for the experiments discussed in Section 5.1. Shell scripting is familiar and widely used for co-ordinating the execution of programs. However, there are three difficulties with this approach - it is difficult to partially re-run, the specification of the filenames is error-prone, and the script is badly modularized.

[4]Some researchers use more general programming languages, such as Perl, Python, or Java to co-ordinate their experiments. While such languages may make some aspects of co-ordination easier – for example, such languages would not have to call out to an external program to produce a range of integers as does the script in Figure 5.1 – the arguments that follow apply equally to these other approaches.

Re-running the experiment The largest difficulty with this script is how it handles errors - namely, it does not. If some early processes succeed, but later ones fail, the researcher can only re-run the entire script, wasting the time spent on the previous run. There are two common solutions to this problem. The simplest is to comment out the parts of the script that have succeeded, and re-run the script. This is highly brittle and error-prone. More reliable but much more complicated is to write a wrapper around each command that checks whether the outputs from the command already exist before running it. Neither of these is desirable. It is also worth noting that this problem can arise not just through error, but when an input file changes, an experiment is extended with further processing, additional graphs are added, further statistics are calculated, or if another model is added to the comparison.

Problematic filenames In this example, a filename is a concatenation of several variable names - e.g. `$(fold).$(class).train`. This is also error-prone - the writer of the script has to keep track, for each filename, of which attributes need to be specified for a given file, and the order in which they must be specified. Either of these can change as an experiment's design evolves, and subtle design changes can require changes throughout the script of the references to many filenames.

Bad modularization In this example, the `eval` program is called twice, even though the input and output files in each case are of the same format. The problem is that the filenames are such that the line in the script that calls `eval` needs to be include information about precisely which files (in one case `$fold.3way.$class`, and in the other `$fold.$class`) are being evaluated.

This is irrelevant – a more modular specification for the `eval` program would simply say that it operates on a `.eval-in` file and produces an `.eval` file. We will see ways below of achieving exactly this.[5]

5.1.2 Approach 2: A `makefile`

One solution to the problems detailed above is to use a makefile instead of a shell script. The `make` program (Feldman, 1979) bills itself as a "utility to maintain groups of programs"[6], but from our perspective, `make` is a declarative language for specifying dependencies. This seems to be exactly what we want, and indeed it does solve some of the problems detailed above. `make` has several new problems, though, which result in its being not an ideal solution to our problem.

Figure 5.2 presents a portion of a makefile for this task. When a makefile rule is executed, `$^` is replaced with all input files (dependencies) of the current rule, and `$@` is replace the output filename. For this part, the makefile ideally matches what we want. It will pick up where it left off, avoiding the re-running problem above. The question of filenames is sidestepped, as we only need to deal with the extensions here. And each command is neatly partitioned into its own section, which specifies its dependencies, the files created by each command, and the shell command to run to create them. However, there are three serious problems with this approach.

[5]One way of achieving this modularization with shell scripts could involve defining functions. While this could be effective, this greatly increases the complexity of the scripts.

[6]GNU `make` manpage.

```
# a .model file is built from a .train file
%.model: %.train
  train $^ > $@

# a .out file is built from a .model file and a .test file
%.out: %.model %.test
  predict $^ > $@
```

Figure 5.2: A partial makefile

Files are represented by strings The first problem can be seen by trying to write a similar line for the `eval` command. It would look something like this:

```
%.eval: %.eval-in
  eval get-class $^ > $@
```

However, it is hard to write the code represented here as `get-class`. This code needs to examine the filename string of `$^` or `$@`, and extract the class from that. This is certainly possible using standard UNIX shell tools or make extensions, but it is ugly, and has to be written once for every time such a field needs to be accessed. For example, one way of writing `get-class` using GNU make extensions[7] would be:

```
GETCLASS = $(filter DSE ESE DSE+ESE,$(subst ., ,$(1)))

%.eval: %.eval-in
  eval $(call GETCLASS,$@) $^ > $@
```

[7]For an explanation of how this works, see `http://www.gnu.org/software/make/manual/make.html#Text-Functions`. However, the reader need not follow the details to see that this method is rather arcane.

A related problem is that the above `predict` command is slightly different from the predict command given in the shell script, in that the test file passed to it as an argument must include the class variable in its name, since the model and output file do. This requires duplicating the test file for each class variable, even though the files are the same. The basic problem here is that to `make`, a file is represented by a string, its filename. For machine learning and natural language processing experiments, it is much more natural to represent a file as a set of key-value pairs. For example, the file `0.B.model` might be represented as `{ fold = 0, class = B, filetype = model }`.

Combinatorial dependencies The second problem with `make` is that it is very difficult to specify combinatorial dependencies. If one continued to write the makefile above, one would eventually need to write a final `all` target to specify all the files that would need to be built. There are 60 such files: one for each fold of the following set

```
$fold.3way.DSE.eval
$fold.3way.ESE.eval
$fold.3way.DSE+ESE.eval
$fold.DSE.eval
$fold.ESE.eval
$fold.DSE+ESE.eval
```

There is no easy way in `make` of listing these 60 files in a natural manner. One can escape to a shell script, or use GNU `make`'s `foreach` function, but both ways are messy.

Non-representable dependency structures The final problem with `make` also relates to dependencies. It is more subtle, but it turns out that there are some sorts of dependency structures that cannot be represented in `make`. Suppose one wants to compare the effect of using one of three parsers, one of three part-of-speech-taggers and one of three chunkers for a summarization experiment. This involves three separate three-way distinctions in the makefile, where for each, there are three different commands that might be run. A non-working example is in Figure 5.3. The problem is that `make` pattern rules (rules using the `%` character) can only match the suffix or prefix of a filename[8]. This makefile does not work because it requires the parser, chunker, and tagger to all be the last part of the filename before the type suffix.

5.1.3 Approach 3: `zymake`

`zymake` is designed to address the problems outlined above. The key principles of its design are as follows:

- Like `make`, `zymake`files can be re-run multiple times, each time picking up where the last left off.
- Files are specified by key-value sets, not by strings
- `zymake` includes a straightforward way of handling combinatorial sets of files.
- `zymake` syntax is minimally different from shell syntax.

[8]Thus, if we were only comparing two sets of items – e.g. parsers and taggers but not chunkers – we could write this set of dependencies by using a prefix to distinguish one set and a suffix to distinguish the other. This is hardly pretty, though, and does not extend to more than two sets.

```
%.taggerA.pos: %.txt
  tagger_A $^ > $@

%.taggerB.pos: %.txt
  tagger_B $^ > $@

%.taggerC.pos: %.txt
  tagger_C $^ > $@

%.chunkerA.chk: %.pos
  chunker_A $^ > $@

%.chunkerB.chk: %.pos
  chunker_B $^ > $@

%.chunkerC.chk: %.pos
  chunker_C $^ > $@

%.parserA.prs: %.chk
  parser_A $^ > $@

%.parserB.prs: %.chk
  parser_B $^ > $@

%.parserC.prs: %.chk
  parser_C $^ > $@
```

Figure 5.3: A non-functional makefile for testing three independent decisions

Figure 5.4 presents the simplest possible zymakefile, consisting of one *rule*, which describes how to create a `$().test` file, and one *goal*, which lists what files should be created by this file. A rule is simply a shell command[9], with some number of *interpolations*[10]. An *interpolation* is anything between the characters `$(` and the matching `)`. This is the only form of interpolation done by `zymake`, so as to minimally conflict with other interpolations done by the shell, scripting

[9]Users who are familiar with UNIX shells will find it useful to be able to use input/output redirection and pipelines in zymakefiles. Knowledge of advanced shell programming is not necessary to use `zymake`, however.

[10]This term is used in Perl; it is sometimes referred to in other languages as "substitution" or "expansion."

languages such as Perl, etc. The two interpolations in this example are file interpolations, which are replaced by `zymake` with a generated filename. Files in `zymake` are identified not by a filename string but by a set of key-value pairs, along with a suffix. In this case, the two interpolations have no key-value pairs, and so are only represented by a suffix. Finally, there are two kinds of file interpolations - *inputs*, which are files that are required to exist before a command can be run, and *outputs*, which are files created by a command[11]. In this case, the interpolation `$(>).test` is marked as an output by the > character[12], while `$().test` is an input, since it is unmarked.

```
extract-test-data raw-data $(>).test

: $().test
```

Figure 5.4: Simple zymakefile #1

The goal of the program in Figure 5.4 is to create a file matching the interpolation `$().test`. The single rule does create a file matching that interpolation, and so this program will result in the execution of the following single command (note that the interpolation `$()` expands to the empty string here because `zymake` does not need to distinguish different kinds of `.test` files):

```
extract-test-data raw-data .test
```

Figure 5.5 presents a slightly more complex zymakefile. In this case, there are two goals - to create a `.test` file with the key `fold` having the value `0`, and

[11] Unlike `make`, `zymake` requires that each command explicitly mention an interpolation corresponding to each input or output file. This restriction is caused by the merging of the command part of the rule with the dependency part of the rule, which are separate in `make`. We felt that this reduced redundancy and clutter in the zymakefiles, but this may occasionally require writing a wrapper around a program that does not behave in this manner.

[12] `zymake` will also infer that any file interpolation following the > character, representing standard output redirection in the shell, is an output.

another `.test` file with `fold` equal to `1`. We also see that the rule has become slightly more complex – there is now another interpolation. This, however, is not a file interpolation, but a variable interpolation. `$(fold)` will be replaced by the value of `fold`.

```
extract-test-data $(fold) raw-data $(>).test

: $(fold=0).test $(fold=1).test
```

Figure 5.5: Simple zymakefile #2

Executing this zymakefile results in the execution of two commands:

```
extract-test-data 0 raw-data 0.test
extract-test-data 1 raw-data 1.test
```

Note that the output files are now not just `.test` but include the fold number in their name. This is because `zymake` infers that the fold key, mentioned in the extract rule, is needed to distinguish the two test files. In general the user should specify as few keys as possible for each file interpolation, and allow `zymake` to infer the exact set of keys necessary to distinguish each file from the rest[13].

Figure 5.6 presents a small refinement to the zymakefile in Figure 5.5. The commands that will be run are the same, but instead of separately listing the two test files to be created, we create a variable `folds` that is a list of all the folds we want, and use a *splat* to create multiple goals. A splat is indicated

[13]Each file will be distinguished by all and only the keys needed for the execution of the command that created it, and the commands that created its inputs. A unique, global ordering of keys is used along with a unique, global mapping of filename components to (key, value) pairs so that the generated filename for each file uniquely maps to the appropriate set of (key, value) pairs.

by the asterisk character, and creates one copy of the file interpolation for each value in the variable's list.

```
extract-test-data $(fold) raw-data
$(>).test

folds = 0 1

: $(fold=*folds).test
```

Figure 5.6: Simple zymakefile #3

Figure 5.7 is now a straightforward extension of the example we have seen so far. It uses a few more features of `zymake` that we will not discuss, such as string-valued keys, and the `range` function, but further documentation is available on the `zymake` website[14]. `zymake` wants to create the goals at the end, so it examines all the rules and constructs a *directed acyclic graph*, or *DAG*, representing the dependencies among the files. It then executes the commands in some order based on this DAG – see Section 5.4 for discussion of execution order.

5.2 Benefits of `zymake`

`zymake` satisfies the criteria set out above, and handles the problems discussed with other systems.

- *Reproducibility*. By providing a single file that can be re-executed many times, `zymake` encourages a development style that encodes all informa-

[14]Binaries for Linux, Mac OS X, and Windows, as well as full source code, are available at `http://www.cs.cornell.edu/~ebreck/zymake/`.

```
extract-test-data $(fold) raw-data $(>).test

extract-2way-training $(fold) raw-data
 $(class) > $(train="2way").train

extract-3way-training $(fold) raw-data
  > $(train="3way").train

train $().train > $().model

predict $().model $().test > $().out

prep-eval-3way $(class) $().out > $(train="3way").eval-in

prep-eval-2way $().out > $(train="2way").eval-in

eval $(class) $().eval-in > $().eval

classes = DSE ESE DSE+ESE
ways = 2way 3way

: $(fold = *(range 0 9)
   class = *classes
   train = *ways).eval
```

Figure 5.7: An example zymakefile. The exact commands run by this makefile are presented in Figure 5.8.

tion about a workflow in a single file. This also serves as documentation of the complete workflow.

- *Simplicity*. `zymake` only requires writing a set of shell commands, annotated with interpolations. This allows researchers to quickly and easily construct new and more complex experiments, or to modify existing ones.
- *Support for experimental life-cycle*. `zymake` can re-execute the same file many times when components fail, inputs change, or the workflow is extended.

```
extract-2way-training 0 raw-data DSE > DSE.0.2way.train
train DSE.0.2way.train > DSE.0.2way.model
extract-2way-training 0 raw-data ESE > ESE.0.2way.train
train ESE.0.2way.train > ESE.0.2way.model
extract-2way-training 0 raw-data DSE+ESE > DSEESE.0.2way.train
train DSEESE.0.2way.train > DSEESE.0.2way.model
extract-3way-training 0 raw-data > 0.3way.train
train 0.3way.train > 0.3way.model
extract-test-data 0 raw-data 0.test
predict DSE.0.2way.model 0.test > DSE.0.2way.out
prep-eval-2way DSE.0.2way.out > DSE.0.2way.eval-in
eval DSE DSE.0.2way.eval-in > DSE.0.2way.eval
predict ESE.0.2way.model 0.test > ESE.0.2way.out
prep-eval-2way ESE.0.2way.out > ESE.0.2way.eval-in
eval ESE ESE.0.2way.eval-in > ESE.0.2way.eval
predict DSEESE.0.2way.model 0.test > DSEESE.0.2way.out
prep-eval-2way DSEESE.0.2way.out > DSEESE.0.2way.eval-in
eval DSE+ESE DSEESE.0.2way.eval-in > DSEESE.0.2way.eval
predict 0.3way.model 0.test > 0.3way.out
prep-eval-3way DSE 0.3way.out > DSE.0.3way.eval-in
eval DSE DSE.0.3way.eval-in > DSE.0.3way.eval
prep-eval-3way ESE 0.3way.out > ESE.0.3way.eval-in
eval ESE ESE.0.3way.eval-in > ESE.0.3way.eval
prep-eval-3way DSE+ESE 0.3way.out > DSEESE.0.3way.eval-in
eval DSE+ESE DSEESE.0.3way.eval-in > DSEESE.0.3way.eval
```

Figure 5.8: Output of the zymakefile in Figure 5.7 Only the commands run for fold 0 are presented, not for all 10 folds. In actual execution `zymake` adds a prefix to each filename based on the name of the zymakefile, so as to separate different experiments. This is only one possible order that the commands could be run in.

- *Support for software engineering*. Each command in a `zymake`file only needs to describe the inputs and outputs relevant for that command, making the separate parts of the file quite modular.
- *Support for combinatorial experiments*. `zymake` includes a built-in method for specifying that a particular variable needs to range over several possibilities, such as a set of models, parameter values, or datasets.

5.3 Using `zymake`

Beginning to use `zymake` is as simple as downloading a single binary from the website. Just as with a shell script or makefile, the user then writes a single textual zymakefile, and passes it to `zymake` for execution. Typical usage of `zymake` will be in an edit-run development cycle.

5.4 Parallel Execution

For execution of very large experiments, efficient use of parallelism is necessary. `zymake` offers a natural way of executing the experiment in a maximally parallel manner. The default serial execution does a topological sort of the DAG, and executes the components in that order. To execute in parallel, `zymake` steps through the DAG starting at the roots, starting any command that does not depend on a command that has not yet executed.

To make this practical, of course, remote execution must be combined with parallel execution. The current implementation provides a simple means of executing a remote job using `ssh`, combined with a simple `/proc`-based measure of remote cpu utilization to find the least-used remote cpu from a provided set.

We are currently looking at extending `zymake` to interface it with the Condor system (Litzkow et al., 1988). Condor's DAGMan is designed to execute a DAG in parallel on a set of remote machines, so it should naturally fit with `zymake`. Interfaces to other cluster software are possible as well. Another important extension will be to allow the system to throttle the number of con-

current jobs produced and/or collect smaller jobs together, to better match the available computational resources.

5.5 Other approaches

Deelman et al. (2004) and Gil et al. (2007) describe the Pegasus and Wings systems, which together have a quite similar goal to `zymake`. This system is designed to manage large scientific workflows, with both data and computation distributed across many machines. A user describes their available data and resources in a semantic language, along with an abstract specification of a workflow, which Wings then renders into a complete workflow DAG. This is passed to Pegasus, which instantiates the DAG with instances of the described resources and passes it to Condor for actual execution. The system has been used for large-scale scientific experiments, such as earthquake simulation. However, we believe that the added complexity of the input that a user has to provide over `zymake`'s simple shell-like syntax will mean a typical machine learning or natural language processing researcher will find `zymake` easier to use.

The GATE and UIMA architectures focus specifically on the management of components for language processing (Cunningham et al., 2002; Ferrucci and Lally, 2004). While `zymake` knows nothing about the structure of the files it manages, these systems provide a common format for textual annotations that all components must use. GATE provides a graphical user interface for running components and for viewing and producing annotations. UIMA provides a framework not just for running experiments but for data analysis and application deployment. Compared to writing a `zymake` script, however, the re-

quirements for using these systems to manage an experiment are greater. In addition, both these architectures most naturally support components written in Java (and in the case of UIMA, C++). `zymake` is agnostic as to the source language of each component, making it easier to include programs written by third parties or by researchers who prefer different languages.

`make`, despite dating from 1979, has proved its usefulness over time, and is still widely used. Many other systems have been developed to replace it, including `ant`[15], `SCons`[16], `maven`[17], and others. However, so far as we are aware, none of these systems solves the problems we have described with `make`. As with `make` and shell scripts, running experiments is certainly possible using these other tools, but we believe they are far more complex and cumbersome than `zymake`.

5.6 Future Extensions

There are a number of extensions to `zymake` that could make it even more useful. One is to allow the dependency DAG to vary during the running of the experiment. At the moment, `zymake` requires that the entire DAG be known before any processes can run. As an example of when this is less than ideal, consider early-stopping an artificial neural network. One way of doing this is train the network to full convergence, and output predictions from the intermediate networks at some fixed interval of epochs. We would like then to evaluate all these predictions on held-out data (running one process for each of them) and

[15] `http://ant.apache.org/`.
[16] `http://www.scons.org/`.
[17] `http://maven.apache.org/`.

then to choose the point at which this score is maximized (running one process for the whole set). Since the number of iterations to convergence is not known ahead of time, at the moment we cannot support this structure in `zymake`. We plan, however, to allow the structure of the DAG to vary at run-time, allowing such experiments.

We are also interested in other extensions, including an optional textual or graphical progress bar, providing a way for the user to have more control over the string filename produced from a key-value set[18], and keeping track of previous versions of created files, to provide a sort of version control of the output files.

5.7 Conclusion

Most experiments in machine learning and natural language processing involve running a complex, interdependent set of processes. We have argued that there are serious difficulties with common approaches to automating these experiments. In their place, we offer `zymake`, a new scripting language with shell-like syntax but `make`-like semantics. We hope our community will find it as useful as we have.

[18]This will better allow `zymake` to interact with other workflows.

CHAPTER 6

TOWARDS IMPROVED FEATURE ENGINEERING FOR NATURAL LANGUAGE PROCESSING

In previous chapters, we have described the empirical, machine learning methods used to solve our natural language problems. In this section, we will discuss one of those methods, feature engineering. In machine learning, problem encoding or feature engineering is a critical part of developing an effective classifier for a learning task. However, the machine learning literature has little to say about this, typically leaving it up to an unspecified domain expert.

```
1: training_data, development_data ← collect raw data
2: learner ← choose learner
3: feature_set ← choose initial features
4: while (error is too high and time remains) do
5:    predictor ← learner(training_data, feature_set)
6:    predictions ← predictor (development_data, feature_set)
7:    diagnosis ← error analysis(predictions, development data)
8:    feature_set ← add/remove features(feature_set, diagnosis)
9: end while
```

Figure 6.1: The steps of feature engineering. Italics indicate steps involving per-task researcher intervention.

Figure 6.1 presents pseudocode of the process of feature engineering, based on our experience in applying machine learning to problems in natural language processing. We believe these steps to be typical for most researchers in natural language processing. The researcher begins a new project by collecting "raw" data, typically the original text of documents along with any existing annotations, and splitting it into training data and development data (the latter is sometimes called the "devtest", in that it is used for testing during development of the model). The researcher also chooses a learning model, such as support vector machines or maximum entropy models. The researcher next chooses

an initial set of features. The loop of feature engineering starts by training the model on the training data, and using the learned model to make predictions on the development data. The researcher then examines the output, scrutinizing the errors, trying to produce a diagnosis of what's causing the errors. Based on this diagnosis, the researcher may remove features that appear to be unhelpful and/or add new features that may improve the model, and then iterates this process.

In this chapter, we explore two ways in which this process can be improved. First, many machine learning algorithms, such as support vector machines, decision trees, and others, expect that feature values for the data set be pre-calculated before learning or prediction take place. Many features are complex, and so this pre-calculation can be a very computationally expensive step. Moreover, feature calculation is typically a two-step process. A knowledge resource is collected and perhaps applied to the data. For a natural language processing task, a knowledge resource might be a part-of-speech tagger, parser, or an information source such as the WordNet hierarchy (Miller, 1995). Then, feature values are extracted for each data instance based upon the knowledge resource. In this chapter, we introduce a simple infrastructure based on databases that supports this two-step process and makes the step of feature pre-calculation more efficient.

A natural question that then arises is whether the steps in the feature engineering process that require per-task researcher intervention (those in italics in Figure 6.1) can be conducted in a semi-automatic, or entirely automatic manner. To this end, we report experiments that intend to produce *automatic feature engi-*

neering. We further introduce the idea of *active feature engineering*, a process that leaves the human in the loop, but minimizes the effort required.

The rest of this chapter proceeds as follows. In Section 6.1, we discuss related work in feature engineering. In Section 6.2, we discuss our database-based infrastructure for pre-calculation of feature values during feature engineering. Finally, in Section 6.3, we discuss performing feature engineering in an entirely automatic manner.

6.1 Related Work

Here we discuss feature engineering research as it relates to our work. It is well known that the formal representation with which an automatic learning procedure is presented matters greatly in the ability of the learner to successfully generalize from training data to novel examples. Hence, there is a huge literature in automatically transforming data from one representation to another. That is, the data is assumed to exist in an input formal representation M (having some structure), and the learning procedure needs instead a set of feature vectors in K^n, where K can be $\mathcal{R}, \{0, 1\}$ or some finite set of values, and n is a positive integer denoting the dimension of the output space. Research in automatically changing representation, then, searches the space of functions $f : M \rightarrow K^n$. We further discuss three subareas: feature selection, constructive induction and user-assisted constructive induction.

Feature Selection Guyon and Elisseef (2003) provide an overview of recent research in feature selection. Most of this research assumes that the researcher has

a (large) set of initial features, but only a small number are relevant, so a learner can produce a better predictor if the relevant subset can be identified. Methods used include ranking features by a scoring function, analyzing the correlation among features, and greedily choosing a subset that directly maximizes the performance of a chosen learning algorithm. Another subarea involves mappings that do not simply choose features individually but project the input features onto a different space, using functions from PCA to wavelets. Either way, research in this area begins with the assumption that instances are represented by feature vectors, while one of our goals is to start with a more basic representation.

Constructive Induction Fawcett (1993) provides an overview of work in constructive induction (albeit older work). The input representation assumed in this work is potentially richer than that assumed in feature selection - some tractable variant of first-order logic is typically used. To cope with this, researchers make a variety of assumptions. One type of assumption is that a formal domain theory is available to guide the search. Such domain theories are rarely available for NLP tasks. Another type of (implicit) assumption assumes that the search will proceed by combining existing features using a small number of combinators (such as logical AND, OR or arithmetic operations). While this may succeed if useful features are a simple combination of a small number of existing features, such methods cannot efficiently explore a very large space of potential features. We hypothesize that for many NLP tasks, the space of potential features is so large that existing methods for automatic exploration of the space will be much less effective than they have been in their original prob-

lem domains (we are not aware of constructive induction work that specifically targets a natural language processing task).

User-assisted constructive induction We are aware of a small amount of research which explores a similar problem to constructive induction but involves the user in the feature creation process. Lo and Famili (1997) develop a system in which a user can, through a simple graphical interface, introduce novel features that are simple arithmetic combinations of existing features. They also explore a number of metrics and methods for evaluating the utility of the new features in the context of data mining. Ngai and Yarowsky (2000) compare an active learning approach to humans writing regular-expression rules for the task of noun phrase chunking. The authors find that active learning appears to be more efficient in terms of human effort for this task.

Active Learning In this chapter, we discuss feature engineering, and several variants from a fully manual process to a fully automatic one. In between would be a process involving back-and-forth interaction between a human and a machine, and a related area of work is called active learning. Active learning is an area of research in which the learning algorithm "has some control over the inputs it trains on" (Cohn et al., 1994, page 201). One typical scenario called "sample selection" involves a large initial pool of unlabeled instances, from which the learner selects a subset for a human to annotate. For example, the learner may select instances on which its prediction has a low confidence. Given this manual annotation, the learner learns a new model, and selects a new subset, and the human and learner iterate back-and-forth until a desirable level of performance is reached. Typically this results in the human needing to annotate far fewer

instances than without active learning.

6.2 A reusable, efficient, and lightweight infrastructure for pre-calculating feature values during feature engineering in natural language processing

In this section, we introduce a reusable, efficient, and lightweight infrastructure for pre-calulating feature values during feature engineering in natural language processing tasks. The infrastructure is motivated by two key observations about the process of Figure 6.1.

- *Specifying the format of the raw data will allow more reusability of the infrastructure code* A researcher might argue that a fixed format for data would be insufficient to capture any desired annotation. However, without a fixed formal format for storing data, it is difficult to produce an infrastructure that can be re-used across tasks.
- *Calculating feature values can be very inefficient* At every iteration, the values for all features for all data instances must be calculated before the rest of the loop can proceed, and yet this calculation can require substantial engineering to operate efficiently. While this is true of other steps in the while loop, training and predicting with machine learning have received considerable attention in the literature. If a reusable infrastructure can improve this efficiency while allowing the researcher to write cleaner code, the researcher's time can be spent more effectively.

Data format What is a reasonable format to use for natural language processing annotations? Many NLP tools natively produce annotations in an "in-band" or "inline" format that intersperses the annotation with the text it modifies. Several examples of inline formats in use in NLP are presented in Figure 6.2. However, in-line annotation has problems. In a typical NLP pipeline, where a succession of programs adds more and more annotations, each program must be able to fully parse the output from all prior annotations, and faithfully reproduce all prior annotations as well as its own on output. Especially when using tools developed by different researchers, this can be a problem, for reasons such as character sets and varied annotation formats. Many in-line annotation formats, such as those derived from XML or SGML, expect tree-structured annotation, meaning that crossing brackets (e.g. `<a> <b> </a> </b>`) cause problems.

A more robust strategy is to produce "out-of-band" or "standoff" annotations, which simply list pairs of an index into the text with the desired annotation information. The GATE system (Cunningham et al., 2002), for example, stores annotations logically as attached to start and end locations in text, and physically in an XML representation. A more abstract structure for annotations is described by Bird and Liberman (1999). There are many ways of specifying annotations, but one possible set of standoff versions of the annotations in Figure 6.2 is presented in Figure 6.3.

There are potentially many physical ways of storing such standoff annotations, but we note that relational databases, a proven and well-understood technology, seem like a very close fit to the annotations in Figure 6.3. We will describe here an implementation that stores each annotation in a separate relational database table, and leave for future work implementations involving

Named entity

```
<ENAMEX TYPE="PERSON">President Bush</ENAMEX> is in
<ENAMEX TYPE="LOCATION">Texas</ENAMEX>
```

Part-of-speech

```
President/NNP Bush/NNP is/VBZ in/IN Texas/NNP
```

Constituent parse

```
(S (NP President Bush) (VP is (PP in (NP Texas))))
```

Figure 6.2: Typical natural language processing annotations

Constituent parse		
start	end	nonterminal
1	5	S
3	5	VP
1	2	NP
4	5	PP

Part-of-speech	
position	part-of-speech
1	NNP
2	NNP
3	VBZ
4	IN
5	NNP

Named entity		
start	end	TYPE
1	2	PERSON
5	5	LOCATION

Figure 6.3: Typical natural language processing annotations – standoff

XML databases or other formats.

Efficient data generation Figure 6.4 depicts an obvious but flawed approach to generating training and test data during feature engineering. With each new

```
1: ...
2: while (error is too high and time remains) do
3:    for instance i in data do
4:       for feature f do
5:          calculate f(i), store in output file
6:       end for
7:    end for
8:    ...
9: end while
```

Figure 6.4: Naive data generation

iteration of feature generation, for every instance, for every feature, we calculate the value of the feature for that instance, and store it in the training or test data file. There are two problems with this approach. The first is that since feature calculation can be very inefficient, it is expensive to re-calculate feature values at every iteration of data generation. Instead, we should cache them once, when the features are invented. The second problem is that multiple features may share common computation, which again is wastefully duplicated. For example, the **prev, next**, and **cur** features from Section 3.2 are all based on running the CASS partial parser, but clearly we do not want to re-run the parser for each feature calculation. Instead, we want to cache shared computation. However, the shared computation is simply another sort of annotation, which we store, as with other annotations, in the annotation database.

Once all annotations and shared computations are stored in the database, the computational specification of a feature becomes simply a database query. Section 6.2.2 below provides a complete example of this encoding. One immediate advantage of defining a feature in this way is that a complicated information source can be stored in the database without specifying exactly how to encode it as a feature, and then the process of feature engineering can be simply exploring alternate queries. For example, one could store all of WordNet as a database, and then explore variant encodings of WordNet-based features by simply writing different queries. Tenenbaum's WordNet-based features for opinion expression identification (see Table 3.5) and our alternate WordNet-based features (see Table 3.9) could be described by writing different queries to the same database. The example in the next section gives a number of specific queries for features for the opinion expression identification task.

6.2.1 Query encoding

One subtlety to be discussed is the exact format in which queries representing features should return their results. Consider the features and feature values in Table 3.4. How should the queries for these features return their result? One way would be to return a table with one record per instance, one column per feature value. For tokens t_1, t_2 and features w_0, **pos**, this would result in Table 6.1.

Table 6.1: A possible result of a feature query

w_0	**pos**	$\cdots$
The	DT	$\cdots$
speaker	NN	$\cdots$
$\vdots$	$\vdots$	$\ddots$

There are two problems with this sort of encoding. One is that it does not accommodate features with multiple values. For example, the **wordnet** feature in Table 3.9 would be difficult to include in this sort of encoding. The other problem is that in practice, it is useful to be able to issue feature queries independently and collect the results later, but since this format does not identify the instances, it requires the construction of a single database query that encodes the entire current feature set. While not impossible, making this query efficient can be problematic. Therefore, we assume that our result tables have a rotated structure, with one record per feature-value, and with every record having three columns - a globally unique instance identifier, a string denoting the name of each feature, and another string denoting the value of the feature. Table 6.2 gives an example of this format. It is easy to issue each query separately and concatenate (union) the results. It is straightforward to convert from this format to a binarized feature-value format for input to SVM^{light} or other

machine learning programs.

Table 6.2: Results of feature queries in our infrastructure

instance identifier	feature name	feature value
t_1	w_0	The
t_1	**pos**	DT
t_2	w_0	speaker
t_2	**pos**	NN
⋮	⋮	⋮

One aspect of problem encoding that is not always discussed is the question of what exactly is an instance. From a high-level description of a research task, it may not always be clear whether an instance for machine learning is, for example, a word or a phrase, a sentence or a paragraph, and so forth. Obviously a computational encoding of the problem must address this, and this infrastructure provides a convenient language for answering this question. One specifies the set of instances by providing a query that returns a set of instance identifiers.

Putting it all together, Figure 6.5 describes our infrastructure for feature engineering. Note that the code that the researcher writes is code that adds tables to databases, and database queries. This remains language agnostic as bindings to popular databases exist in most popular languages.

6.2.2 An example: identifying DSESEs (from Section 3.2)

We will use relational databases and SQL[1] in this example, although we could have used XML databases and xquery (The World Wide Web Consortium, 2005) or other databases. We construct SQL queries corresponding to the features

[1]SQL is the Structured Query Language, the standard for relational databases such as Oracle or MySQL.

```
1: (training_data, development_data) ← collect raw data
2: learner ← choose learner
3: database ← choose initial knowledge sources
4: feature_queries ← choose initial feature set
5: feature_cache ← issue feature_queries to database
6: while (error is too high and time remains) do
7:    (training_vectors, development_vectors) ← combine values from feature_cache {Only use features in current feature_queries set}
8:    predictor ← learner(training_vectors)
9:    predictions ← predictor (development_vectors)
10:   diagnosis ← error analysis(predictions, development_vectors)
11:   database ← if needed, add new knowledge source
12:   feature_queries ← add/remove features(diagnosis, feature_queries)
13:   feature_cache ← issue new feature_queries to database
14: end while
```

Figure 6.5: An infrastructure for feature engineering. Italics indicate processes not handled by the infrastructure.

used in DSESE identification (described in Section 3.2), from which an SQL engine can produce feature vectors. We do not intend these to be necessarily the most efficient queries for the problem, but merely to make clear how a query language can be used for the purpose of defining features.

The problem of identifying DSESEs uses the MPQA corpus introduced in Section 3.2. This corpus is the "raw data" of Figures 6.1 and 6.5. We represent all the data in the corpus – the raw text and manual annotations – as well as knowledge sources like part of speech and parse information as tables in a relational database. Schema design is a difficult problem, but here we simply assume that we have one table per type of annotation. Thus we have four tables in our database: `token`, `cass`, `dsese`, and `lev_fn`.

Consider Table 6.3, depicting the `token` table. The table has one row per token in all documents in the corpus. The fields are, in order, the character index of the start of the token in the raw text, the index of the end of the token,

the ordinal index of the token in the corpus, the part-of-speech of the current token, and the string itself.

Table 6.3: The `token` table

start	end	ind	pos	string
1	4	1	DT	The
5	8	2	NN	speaker
10	13	3	VBD	argued
⋮	⋮	⋮	⋮	⋮

The `cass` table, in Table 6.4, contains information extracted from the partial parse provided by the CASS parser. Besides the information common to all tables, this table gives the previous, current, and following node in the partial parse tree. The `lev_fn` table, in Table 6.5, lists categories of words based on Beth Levin's verb categorization and the Framenet resource. Finally, the `dsese` table, in Table 6.6, lists all the DSESEs, including for each one, its source and whether it is explicit in the text or implicit (e.g. the writer's DSESE is implicit).

Table 6.4: The `cass` table

start	end	prev	cur	next
1	4	empty	nx	vx
5	8	empty	nx	vx
10	13	nx	vx	vx
⋮	⋮	⋮	⋮	⋮

Table 6.5: The `lev_fn` table

start	end	type
10	13	levin_strong_verb
10	13	framenet_speech_event_verb
⋮	⋮	⋮

Table 6.6: The `dsese` table

start	end	is_implicit	source
1	1	true	writer
10	13	false	speaker
⋮	⋮	⋮	⋮

6.2.3 Queries

With the raw data for the problem stored according to this schema, we can express the original features for DSESE-identification with the SQL queries depicted in Table 6.7. Each query is required to return a table of records with three fields - a numerical instance identifier, a string naming which feature this is a value of, and a third string that denotes the value of that feature. The infrastructure handles translating these categorically-valued features into binary features.

Table 6.7: SQL queries to generate the original features for DSESE identification from Section 3.2

feature	query
instance	`select ind from token`
class	`select token.ind,'class',1 from token join dsese using (start,end) where not dsese.is_implicit`
$w_{-4} \ldots w_4$	`select t0.ind,'w'||(t1.ind-t0.ind),t1.string from token as t0,token as t1 where t1.ind >= t0.ind-4 and t1.ind <= t0.ind+4`
lev, **fn**	`select token.ind,'lev_fn',type from token join lev_fn using (start,end)`
cur	`select token.ind,'cur',cur from token join cass using (start,end)`
next	`select token.ind,'next',next from token join cass using (start,end)`
prev	`select token.ind,'prev',prev from token join cass using (start,end)`
pos	`select ind,'pos' as name,pos as value from token`

Table 6.8 depicts the result of executing this query on the tables shown. A simple wrapper script can transform this table into feature vectors for input to

the base learner.

Evaluation A system like this is difficult to evaluate. Anecdotally, our work described in Chapter 3 was done using a preliminary implementation of this framework. It allowed us to easily add new features by writing a short SQL query for each new feature, and to quickly test various combinations of features. We have also done feature engineering using a procedural encoding of features, and found that the effort involved in making the pre-calculation efficient and in adding new features was substantially more than in using this infrastructure. Finally, note that while this discussion has used a simple stand-off annotation scheme and relational databases, a similar infrastructure could be built using other annotation schemes and other databases, and we are interested in exploring this in the future.

Table 6.8: The result of the queries in Table 6.7

ind	name	value
3	class	1
⋮	⋮	⋮
1	cur	nx
2	cur	nx
3	cur	vx
⋮	⋮	⋮
1	next	vx
2	next	vx
3	next	vx
⋮	⋮	⋮
1	pos	DT
2	pos	NN
3	pos	VBD
⋮	⋮	⋮
1	prev	empty
2	prev	empty
3	prev	nx
⋮	⋮	⋮
3	w-2	The
⋮	⋮	⋮
2	w-1	The
3	w-1	speaker
⋮	⋮	⋮
1	w0	The
2	w0	speaker
3	w0	argued
⋮	⋮	⋮
1	w1	speaker
2	w1	argued
⋮	⋮	⋮
1	w2	argued
⋮	⋮	⋮
3	lev_fn	levin_strong_verb
3	lev_fn	framenet_speech_event_verb
⋮	⋮	⋮

6.3 Automatic feature engineering

A natural question that follows from the previous section is whether we can fully or partially automate those steps of the feature engineering process that required human intervention. We introduce *automatic feature engineering*, a process that attempts to take raw data and a learning algorithm, and determine automatically the ideal feature set. This is distinct from constructive induction as no base set of features is provided, only knowledge sources, encoded in relational databases.

Researcher intervention during feature engineering in the infrastructure described in Figure 6.5 comes in three parts: diagnosing the errors; based on diagnosis, adding new knowledge sources; and based on diagnosis, adding or removing feature queries. In this section, we investigate the hypothesis that each of these steps can be done fully automatically.

We automate each of these three pieces in a simple way as follows.

- *Diagnosing the errors*. Ideally, we would have a system that could automatically examine the predictions of the learned model, identify errors, determine the causes of errors and propose solutions. In this simple experiment, however, we simply identify errors, i.e. we will identify whether or not a test instance from the development corpus was assigned the correct class.
- *Adding new knowledge sources*. It is not clear what a reasonable way of automatically adding new knowledge sources would be. One potential solution, discussed below, is to build up a very large stable of knowledge sources useful across NLP, so that a new task would not be likely to need

additional sources. In this simple experiment, we simply begin with a small number of initial knowledge sources and not add any more during feature engineering.

- *Adding or removing new features*. Since we do not have diagnosis, we have no guide for what features to add. Instead, we simply choose a large set of potentially useful features before feature engineering begins, and add one feature from the set at each iteration. We only consider removing the most recently added feature, based on the results of automatic evaluation.

In the next section, we describe an experiment run using this setup. In following sections, we analyze these automatic steps, and suggest potential avenues for alleviating the difficulties encountered.

6.3.1 Automatic feature engineering for identifying direct subjective expressions

Given the automatic steps above, we will run experiments to test the efficacy of automatic feature engineering on the task of identifying direct subjective expressions defined in Chapter 3. We will compare the automatically learned features to hand-engineered features. We hope to be able to substantially outperform the hand-engineered features, as well as to discover one or more "killer features" that we did not think of when doing manual feature engineering.

Adding and removing features

For this experiment, rather than exploring the full complexity of SQL queries, we will explore a simple space of potential features. The space will consist of features b_i, where b is a basis feature, and i is the offset from the target word - i.e. b_0 represents evaluating b on the current word, while b_2 represents evaluating b on the word two to the right of the current word.

The basis features we use are based on the features described in Chapter 3. From Section 3.2 we include the word itself w, the part-of-speech **pos**, the partial-parse features **prev, cur, next**, and the Levin and FrameNet features **lev** and **fn**. From Section 3.3 we include the **lemma**, hypernyms of the lemma **wordnet**, and the original Levin category **levinorig**. We also include a version of **levinorig** that is only active for words tagged as verbs, called **levinorig-verb**. From Section 3.5, we include a feature based on the wordlist from Wilson et al. (2005b) **wilson**, as well as analogous features based on the wordlist from Wiebe and Riloff (2005) **wiebe**, another wordlist used by Wiebe[2] **wiebe1**, and on the original baseline dictionary **oldbase**.

There are many potential ways of exploring this space of features. One typical method, called greedy forward stepwise feature selection, involves considering every feature at each iteration, and choosing the best one. We were intrigued by the work of Ungar et al. (2005) on Streamwise Feature Selection. The idea of this paradigm is that a space of features may be so large that traditional forward stepwise selection is not feasible, as it is too expensive to consider every potential feature at each iteration. Instead, they suggest ordering the features, and considering each feature only once, in order, accepting or rejecting it. Evaluat-

[2]Personal communication, the list transmitted from Wiebe in August 2005.

ing on a synthetic dataset, a bankruptcy prediction dataset, and a link prediction task based on CiteSeer, they find that Streamwise Feature Selection is able to successfully cope with extremely large potential feature sets and is robust to large numbers of noise features. We adopt this paradigm for these experiments, and compare to a model using the features manually selected in Chapter 3 and to a model using all features within a window of six words.

The next decision to make is how to decide to accept or reject each feature. Ungar et al. (2005) suggest using statistical model selection criteria to make this decision, and we follow their suggestion.

Model selection criteria The statistics and information theory literature have produced a number of model selection criteria, among them the Akaike Information Criterion (AIC) (Akaike, 1974), Bayes Information Criterion (BIC) (Schwarz, 1978), the Risk Inflation Criterion (RIC) (Foster and George, 1994), the Information Investment Criterion (IIC) (Ungar et al., 2005). Each criterion computes a score for a model such that, when choosing between two potential models, the one having the lower score is to be selected. The formulas for each score are presented below.

$$AIC(model) = 2k - 2\ln L$$

$$BIC(model) = k\ln n - 2\ln L$$

$$RIC(model) = 2k\ln p - 2\ln L$$

Here k is the number of parameters in the trained model, L is the likelihood for the model, n is the number of instances in the data set, and p is the total number of available features – that is, the number of features that will ever be considered. We assume that the learned model has one parameter per feature, so k will be equal to the current number of features. In our case, we are interested in whether to choose a model with a particular feature, or without that feature. At the ith iteration, we compute the likelihood L_{i-1} for the model with k_{i-1} features, add the new feature, compute the likelihood L_i for the model with $k_{i-1} + 1$ features. All criteria are of the form $C = 2kx - 2\ln L$, and suggest that we choose the model with the smaller criterion value, i.e. accept the new feature if $C_i < C_{i-1}$. This works out to $2(k_{i-1} + 1)x - 2\ln L_i < 2k_{i-1}x - 2\ln L_{i-1}$, or $x < \ln L_i - \ln L_{i-1}$ — in other words, each criterion type sets a threshold that the increase in log-likelihood must be above for the feature to be accepted by the model. From the definitions above, the threshold x is 1 for AIC, $\frac{1}{2}\ln n$ for BIC, and $\ln p$ for RIC.

The IIC criterion works slightly differently from the rest, in that its value changes over time, becoming less restrictive as valuable features are added to the model and more restrictive as the selection process moves later into the stream of features. The threshold is set to $w_\Delta + b - \log \frac{w}{2i}$, where w_Δ is a constant, specifying the amount by which the "wealth" is increased each time a valuable feature is added to the model (here set to 0.5, following Ungar et al. (2005)), b is the coding cost for the coefficient (here set to 3, following Ungar et al. (2005)), w is the current wealth, adjusted as the process proceeds, and i is the iteration number.

Experiments

Our experiment in automatic feature engineering is based on the experiments to identify opinion expressions described in Section 3.5. We choose in particular to identify direct subjective expressions, and based on the results in Table 3.12, we use order-0 conditional random fields trained to predict both direct subjective expressions and expressive-subjective elements. We will report the $SF^{overlap}$ statistic of Section 3.2 . We diverge from the experimental setup in Section 3.5 in just two ways - first, we train and test solely on the 135-document development data[3] (due to the computationally-intensive nature of these experiments), and, of course, we use a different set of features. At each iteration, we consider a single feature from the ordered set described above, and accept or reject it according to the IIC criterion. We begin by using the IIC criterion because it was found to be superior by Ungar et al. (2005), and because, as we will see below, all the criteria turn out to be equally poor.

Results Figures 6.6 and 6.7 present the results of the experiment described above and including performance on test and training data. Results are also presented for a model using the manually selected feature set described in Section 3.5 and a model using all potential features above within a six-word window on either side of the target word.

Discussion The results are disappointing, showing that the algorithm heavily overtrains, with training set performance continually increasing while test set performance peaks early (at iteration 27) and then declines. The automatic fea-

[3]We use the 66 documents of development data referred to in Chapter 4 for (dev)testing, and the remaining 69 documents of development data for training.

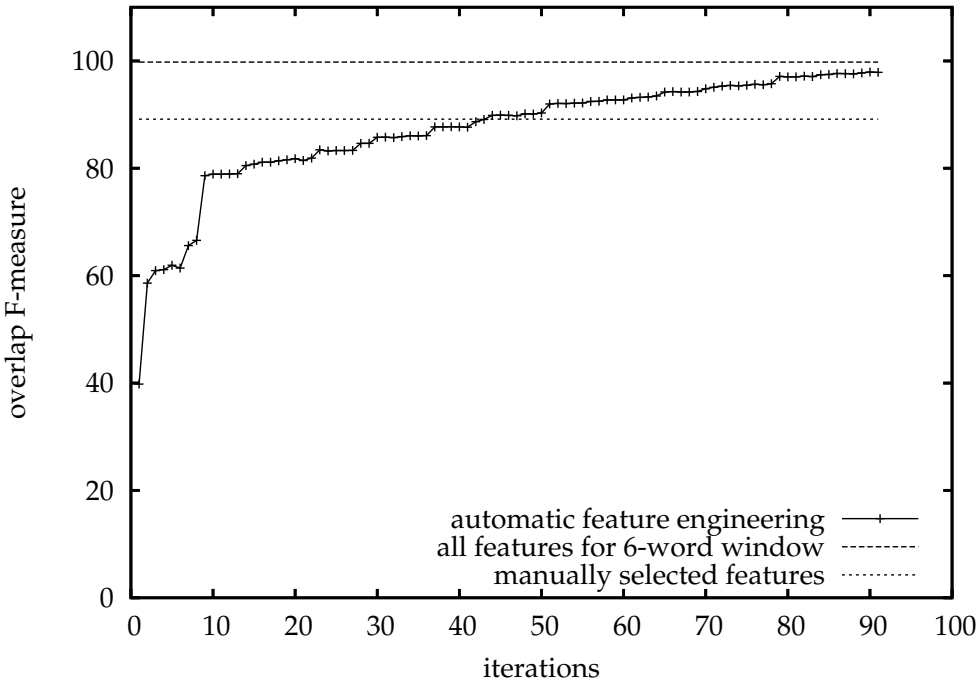

Figure 6.6: Results on training set of automatic feature engineering for identifying direct subjective expressions.

ture engineering results for the test set performance do slightly outperform the hand-selected features (the peak value at iteration 27 is 64.94 versus 63.87 for the hand-selected features). The hand-selected feature set has greater test set performance than using all features (again, 64.94 at its peak versus 60.13 for all features).

The key problem turns out to be that the model selection selection criteria are a poor match for this task. When we examine which features were selected, we see a difficulty with this algorithm - nearly all features are being selected. Recall that the information criteria select any feature if adding it produces a gain in log-likelihood bigger than some decision threshold. Figure 6.8 plots the gain in

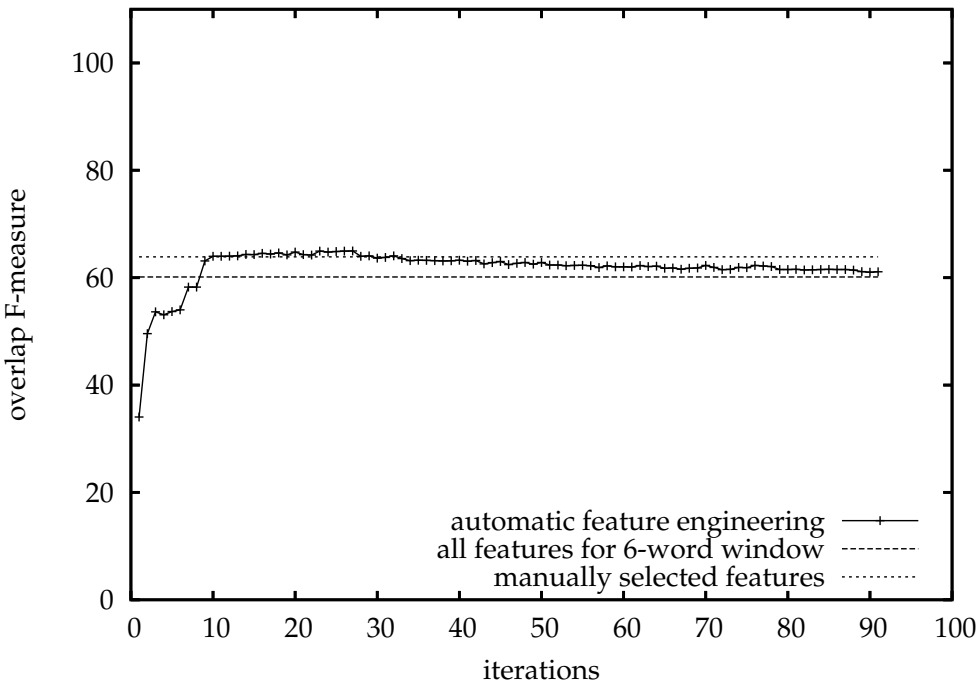

Figure 6.7: Results on devtest set of automatic feature engineering for identifying direct subjective expressions

log-likelihood at each iteration of the experiment, and also the decision threshold corresponding to each of the four information criteria (again, the criterion actually used for choosing features was IIC). The figure shows that the various information criteria's thresholds are quite similar, and all are well below the required change in log-likelihood for most features. Moreover, the log-likelihood increase for features remains quite high for later iterations, even as the features are less and less helpful to F-measure performance, so that all criteria will continue to select features throughout the process. It is not clear why the model selection criteria relate so poorly to F-measure test set performance. One reason may be that in the calculations above, we assumed each feature adds one

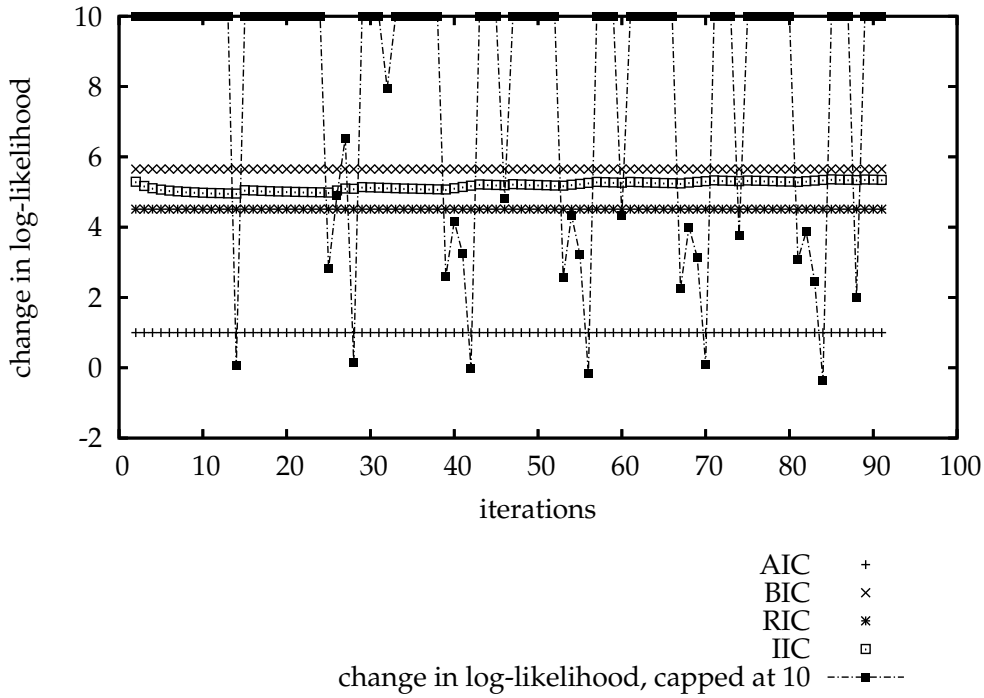

Figure 6.8: Selecting features according to several information criteria. This graph plots change in log-likelihood during the experiment described in the text, using IIC to select features. It also plots the decision threshold corresponding to each of four information criteria.

model parameter, while in fact many of our features (such as w_0, the current word, or **wordnet**, the wordnet hypernyms of the current word), correspond to tens or hundreds or thousands of features when binarized for use in the conditional random field, and thus correspond to equally large numbers of weights. There are a number of potential ways of coping with this, but all have difficulties. Dividing the log-likelihood gain at each step by the corresponding number of features would downweight the usefulness of a feature like WordNet that is extremely helpful but corresponds to tens of thousands of binary features. Considering each binary feature separately would be computationally infeasible.

6.3.2 Examining automatic feature engineering

Our simple fully automatic feature engineering experiment was not successful. That experiment was based on simple automation of several steps of the feature engineering process. Here we examine these automations and their impact on the success of the experiment.

Diagnosing the errors The above feature generation method does not include diagnosis of the errors. Without diagnosis, the method must be "generate-and-test" - that is, it generates potential features (by generating successively more complex queries, or expressions in some language), and then either generates several of them and does feature selection, or chooses, one-by-one, whether to include each feature. Neither of these is really ideal; it would be preferable to learn, given the problem at hand, what the feature is that can "solve it". Existing rule-learning systems, however, are only guided search at the level of how to combine atomic features, and here, atomic features are the problem.

Adding and removing features The space of features considered in the experiments above in is not particularly large. In this case, existing machine learning methods (such as support vector machines or maximum entropy models) can often cope with reasonably large feature sets. Even if they have relatively poor performance, the problem then reduces to feature *selection*, which is well-studied. On the other hand, if the space of features is something like "SQL queries less than a page long", we believe that (a) no system could possibly cope with the ridiculous number of potential features, but (b) nearly all features of interest (not involving significant external knowledge sources) for many problems

could be encoded. We have yet to find a problem domain between these two extremes - too large to reduce to "just feature selection", but too small to actually contain any nonobvious "killer features" worth generating/searching for in the first place.

Adding new knowledge sources The method above requires that a set of knowledge sources be computed before automatic feature generation begins. Knowing in advance of a given project what information will be useful is a daunting expectation. However, it may be possible to canvass the literature on natural language processing to come up with a moderately large set of standard knowledge resources, beyond which a particular problem will require little additional new information.

6.3.3 Active feature engineering

One potential point between fully automatic and fully manual feature engineering would be a semi-automatic process that keeps the human in the loop but tries, similar to active learning, to carefully target their efforts to the most useful point. We call this process *active feature engineering*, and discuss two problems to be solved so that this process can be used.

A better query language Relational databases and SQL are a reasonable approach and we have used them successfully in many experiments. However, there are difficulties with these tools. Portable SQL is not able (although extensions may exist) to handle notions such as "ancestor" that are important in

tree-based knowledge bases like WordNet or in computing features from parses. It is also difficult to compute `argmax`, a central notion in many natural language processing models. The XQuery (The World Wide Web Consortium, 2005) language, a Turing-complete language for querying XML databases, may present a solution to some of these problems, or it may be worthwhile developing a domain-specific language for feature queries.

Automatic error analysis and error views Error analysis typically involves a tedious process of looking at system outputs, trying to figure out what's causing the errors, and hypothesizing how they might be eliminated. Our goal is to bring as much systematicity to this process as possible. In an ideal world, a fully automatic process could analyze a system's outputs and report back to a human with a causal account of the system's mistakes. We feel that this is unlikely to be achieved in the near term, so our goal is simpler: to provide a system that will canvass the system's output, find patterns of error, and provide a coherent report to the researcher in the form of error views. These error views come in two classes: views of the data as a whole, and views of individual errors. Views of the data as a whole might include confusion matrices, learning curves, or bias/variance decompositions. Views of individual errors might sort the instances by some criterion, or group instances by a common trait. Since we have not yet implemented any of these ideas, we leave further details for Section 7, but the general idea is to provide the researcher with tools for looking at the data in such a way that the causes of error will be immediately apparent.

Figure 6.9 presents a new architecture, which we call active feature engineering. Rather than the researcher coming to a conclusion about the sources of error unaided, we propose a step of "automatic error analysis", that provides a set of

```
1: training_data, development_data ← collect raw data
2: learner ← choose learner
3: database ← choose initial knowledge sources
4: feature_queries ← choose initial feature set
5: feature_cache ← issue feature_queries to database
6: while (error is too high and time remains) do
7:    (training_data, development_data) ← combine values from feature_cache {Only use features in current feature_queries set}
8:    predictor ← learner(training_data)
9:    predictions ← predictor (development_data)
10:   error_views ← automatic error analysis(system output, development data)
11:   diagnosis ← manual error analysis(error_views)
12:   database ← if needed, add new knowledge source, based on diagnosis
13:   feature_queries ← add/remove features (diagnosis, feature_queries)
14:   feature_cache ← issue new feature_queries to database
15: end while
```

Figure 6.9: Active feature engineering. Italics indicate manual steps

"error views" from which the researcher can more easily deduce the causes of error[4]. Once the researcher has (manually) inspected these views, rather than having to write new program code to modify the system, he or she simply proposes one or more new features, written in a feature language. Note that the training step might include automatic feature selection, so the researcher's new features might or might not actually be utilized by the system.

In this chapter, we have discussed the problem of feature engineering for natural language processing, and introduced two improvements to the process. We introduced a simple framework based on databases for improving the efficiency of pre-calculating feature values. In the future we hope to more effectively evaluate this framework, and to consider incorporating other annotation schemes. We also set up an experiment in automatically conducting the process

[4]We are interested in investigating whether commercial data mining tools may already provide useful functionality of this type.

of feature engineering. While not successful, this experiment leaves a number of interesting questions for future research.

CHAPTER 7

CONCLUSION

This thesis has introduced several problems in empirical natural language processing and demonstrated solutions. Here we recap the contributions of this work and discuss some promising directions for future work.

Identifying expressions of opinion We have created and evaluated a system that identifies several types of opinion expressions with high accuracy. This sets up a basic building block for opinion-oriented information extraction systems.

Identifying opinion hierarchies We have presented work that identifies with high accuracy the hierarchical structure of opinion expressions. This structure reflects the manner in which information is passed from one source to another and will be crucial in analyzing the reliability of the information available to a reader.

A lightweight system for machine learning workflows We have introduced a tool for coordinating the execution of programs for machine learning and natural language processing experiments. We have argued that this tool is superior to alternatives such as shell scripts and makefiles for coordinating experiments. We hope to extend this tool to make it even more useful, for example to allow it to integrate with grid computing resources. Moreover, we are interested in exploring other ways of encouraging reproducibility in computational linguistics research.

Feature engineering for natural language processing We have analyzed the process of feature engineering and introduced a simple infrastructure for improving the efficiency of the process. We also conducted experiments to test the difficulty of automating feature engineering, discovering that a simple approach does not work. We outlined a number of future approaches, including building a library of reusable knowledge resources to make feature engineering easier on future tasks.

BIBLIOGRAPHY

[Abney1996] Steven Abney. 1996. Partial parsing via finite-state cascades. *Journal of Natural Language Engineering*, 2(4):337–344.

[Abney1997] Steven Abney. 1997. The SCOL manual. cass is available from http://www.vinartus.net/spa/scol1h.tar.gz.

[Akaike1974] Hirotugu Akaike. 1974. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

[Anick and Bergler1991] Peter Anick and Sabine Bergler. 1991. Lexical structures for linguistic inference. In *Workshop in Lexical Semantics and Knowledge Representation*.

[Bergler1991] Sabine Bergler. 1991. The semantics of collocational patterns for reporting verbs. In *EACL91*.

[Bergler1993] Sabine Bergler. 1993. Semantic dimensions in the field of reporting verbs. In *Proceedings of the Ninth Annual Conference of the University of Waterloo Centre for the New Oxford English Dictionary and Text Research*, Oxford, England, September.

[Bethard et al.2004] Steven Bethard, Hong Yu, Ashley Thornton, Vasileios Hatzivassiloglou, and Dan Jurafsky. 2004. Automatic extraction of opinion propositions and their holders. In *Working Notes of the AAAI Spring Symposium on Exploring Attitude and Affect in Text: Theories and Applications*. March 22-24, 2004, Stanford.

[Bird and Liberman1999] Stephen Bird and Mark Liberman. 1999. A formal framework for linguistic annotation. Technical Report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania.

[Breck and Cardie2004] Eric Breck and Claire Cardie. 2004. Playing the telephone game: Determining the hierarchical structure of perspective and speech expressions. In *The 20th International Conference on Computational Linguistics (COLING 2004)*, pages 120–126.

[Breck et al.2007] Eric Breck, Yejin Choi, and Claire Cardie. 2007. Identifying expressions of opinion in context. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-2007)*, Hyderabad, India, January.

[Breck2008] Eric Breck. 2008. zymake: A computational workflow system for machine learning and natural language processing. In *Working Notes of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*. A workshop of ACL-2008.

[Buntine1993] Wray Buntine. 1993. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall,London. Available at http://ic.arc.nasa.gov/projects/bayes-group/ind/IND-program.html.

[Chinchor et al.1993] Nancy Chinchor, Lynette Hirschman, and David Lewis. 1993. Evaluating message understanding systems: An analysis of the third message understanding conference (MUC-3). *Computational Linguistics*, 19(3):409–450.

[Choi et al.2005] Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. 2005. Identifying sources of opinions with conditional random fields and extraction patterns. In *Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing*, Vancouver, Canada, October.

[Choi et al.2006] Yejin Choi, Eric Breck, and Claire Cardie. 2006. Joint extraction of entities and relations for opinion recognition. In *Proceedings of the 2006 Conference on Empirical Methods in Natura l Language Processing (EMNLP-2006)*.

[Church and Mercer1993] Kenneth Ward Church and Robert L. Mercer. 1993. Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1), March. Special Issue on Using Large Corpora.

[Cohen1995] William W. Cohen. 1995. Fast effective rule induction. In A. Prieditis and Stuart Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA. Morgan Kaufmann.

[Cohn et al.1994] David A. Cohn, Les Atlas, and Richard E. Ladner. 1994. Improving generalization with active learning. *Machine Learning*, 15(2):201–221.

[Collins1999] Michael John Collins. 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

[Cunningham et al.2002] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings*

of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL '02), Philadelphia, July.

[Daelemans et al.2000] W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. 2000. TiMBL: Tilburg memory based learner, version 3.0, reference guide. Technical report, Tilburg University.

[Das and Chen2001] Sanjiv Das and Mike Chen. 2001. Yahoo! for amazon: Sentiment parsing from small talk on the web. In *Proceedings of the Asia Pacific Finance Association Annual Conference*, Bangkok, Thailand, July.

[Dave et al.2003] Kushal Dave, Steve Lawrence, and David M Pennock. 2003. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *WWW2003*.

[Deelman et al.2004] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. 2004. Pegasus : Mapping scientific workflows onto the grid. In *Across Grids Conference*, Nicosia, Cyprus.

[DUC2007] 2007. *Proceedings of the Document Understanding Conference*, Rochester, NY, April. Presented at the NAACL-HLT 2007.

[Eisner1996] Jason Eisner. 1996. An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, IRCS, University of Pennsylvania.

[Fawcett1993] Tom E. Fawcett. 1993. *Feature Discovery for Problem Solving Systems*. Ph.D. thesis, University of Massachusetts, Amherst, May. CMPSCI Technical Report 93-49.

[Feldman1979] Stuart I. Feldman. 1979. Make-a program for maintaining computer programs. *Software - Practice and Experience*, 9(4):255–65.

[Ferrucci and Lally2004] David Ferrucci and Adam Lally. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348.

[Foster and George1994] Dean P. Foster and Edward I. George. 1994. *Annals of Statistics*, 22(4):1947–1975.

[Gerard2000] Christine Gerard. 2000. Modelling readers of news articles using nested beliefs. Master's thesis, Concordia University, Montréal, Québec, Canada.

[Gil et al.2007] Yolanda Gil, Varun Ratnakar, Ewa Deelman, Gaurang Mehta, and Jihie Kim. 2007. Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI)*, Vancouver, British Columbia, Canada, July.

[Guyon and Elisseef2003] Isabelle Guyon and André Elisseef. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

[Hatzivassiloglou and McKeown1997] Vasileios Hatzivassiloglou and Kathleen R. McKeown. 1997. Predicting the semantic orientation of adjectives. In *ACL97*.

[Hobbs et al.1988] Jerry Hobbs, Mark Stickel, Paul Martin, and Douglas Edwards. 1988. Interpretation as abduction. In *Proceedings of the 26th annual meeting of the Association for Computational Linguistics*.

[Jindal and Liu2006] Nitin Jindal and Bing Liu. 2006. Mining comparative sentences and relations. In *Proceedings of AAAI*.

[Kim and Hovy2006] Soo-Min Kim and Eduard Hovy. 2006. Extracting opinions, opinion holders, and topics expressed in online news media text. In *Proceedings of ACL/COLING Workshop on Sentiment and Subjectivity in Text*, Sydney, Australia.

[Kobayashi et al.2004] Nozomi Kobayashi, Kentaro Inui, Yuji Matsumoto, Kenji Tateishi (NEC), and Toshikazu Fukushima (NEC). 2004. Collecting evaluative expressions for opinion extraction. In *Proceedings of the First International Joint Conference on Natural Language Processing*, pages 584–589.

[Lafferty et al.2001] John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labe ling sequence data. In *ICML*.

[Levin1993] Beth Levin. 1993. *English Verb Classes and Alternations*. University of Chicago Press.

[Lin1995] Dekang Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *IJCAI*, pages 1420–1427.

[Litzkow et al.1988] Michael Litzkow, Miron Livny, and Matthew Mutka. 1988. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June.

[Lo and Famili1997] Suzanne Lo and Abolfazl Famili. 1997. Development of a knowledge-driven constructive induction mechanism. In *Proceedings of the Second International Symposium on Intelligent Data Analysis (IDA-97)*, London, England, August.

[Marsh2005] Bill Marsh. 2005. The blame game, October 1. The New York Times.

[McCallum2002] Andrew Kachites McCallum. 2002. MALLET: A machine learning for language toolkit. http://mallet.cs.umass.edu.

[Miller1995] George A. Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, November.

[Morinaga et al.2002] Satoshi Morinaga, Kenji Yamanishi, Kenji Tateishi, and Toshikazu Fukushima. 2002. Mining product reputations on the web. In *KDD 2002*.

[MTX2007] 2007. *Proceedings of Machine Translation Summit XI*, Copenhagen, Denmark, September.

[Munson et al.2005] M Arthur Munson, Claire Cardie, and Rich Caruana. 2005. Optimizing to arbitrary NLP metrics using ensemble selection. In *HLT-EMNLP05*.

[Munson2004] M Arthur Munson. 2004. Automatic annotation of speech events and explicit private state in newswire. Unpublished class project.

[Nasukawa and Yi2003] Tetsuya Nasukawa and Jeonghee Yi. 2003. Sentiment analysis: capturing favorability using natural language processing. In *Proceedings of the international conference on Knowledge capture*.

[Ngai and Yarowsky2000] Grace Ngai and David Yarowsky. 2000. Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking.

In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*.

[Norvig1987] Peter Norvig. 1987. Inference in text understanding. In *Proceedings of the National Conference on Artificial Intelligence*.

[Pang and Lee2004] Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of ACL 2004*.

[Pang and Lee2008] Bo Pang and Lillian Lee. 2008. *Opinion mining and sentiment analysis*. Now publishers, July.

[Pang et al.2002] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *EMNLP*.

[Quinlan1986] J.R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Riloff and Wiebe2003] Ellen Riloff and Janyce M Wiebe. 2003. Learning extraction patterns for subjective expressions. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*.

[Riloff et al.2003] Ellen Riloff, Janyce Wiebe, and Theresa Wilson. 2003. Learning subjective nouns using extraction pattern bootstrapping. In *CoNLL-2003*.

[Schwarz1978] Gideon E. Schwarz. 1978. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464.

[Stoyanov et al.2005] Ves Stoyanov, Claire Cardie, and Janyce Wiebe. 2005. Multi-perspective question answering using the opqa corpus. In *HLT-EMNLP05*.

[Subasic and Huettner2001] Pero Subasic and Alison Huettner. 2001. Affect analysis of text using fuzzy semantic typing. In *IEEE-FS*, volume 9, pages 483–496, August.

[Tenenbaum2004] Sofya Tenenbaum. 2004. Applying wordnet to automatic annotation of speech events and private state. CS790 report (Master's of Engineering project).

[The World Wide Web Consortium2005] The World Wide Web Consortium. 2005. XQuery 1.0: An XML query language. W3C working draft. http://www.w3c.org/TR/xquery.

[Tong2001] Richard M Tong. 2001. An operational system for detecting and tracking opinions in on-line discussions. In *SIGIR 2001 Workshop on Operational Text Classification Systems*.

[Turney and Littman2002] Peter D. Turney and Michael L. Littman. 2002. Unsupervised learning of semantic orientation from a hundred-billion-word corpus. Technical report, National Research Council Canada. NRC Technical Report ERB-1094.

[Turney and Littman2003] Peter D Turney and Michael L Littman. 2003. Measuring praise and criticism: Inference of semantic orientation from association. *ACM Transactions on Information Systems*, 21(4):315–346.

[Ungar et al.2005] Lyle Ungar, Jing Zhou, Dean Foster, and Bob Stine. 2005. Streaming feature selection using IIC. In *AI and Statistics*.

[Voorhees1999] Ellen Voorhees. 1999. The trec-8 question answering track report. In *Proceedings of TREC-8*.

[Wiebe and Riloff2005] Janyce Wiebe and Ellen Riloff. 2005. Creating subjective and objective sentence classifiers from unannotated texts. In *Proceedings of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2005)*. Springer-Verlag. Invited paper.

[Wiebe and Wilson2002] Jaynce Wiebe and Theresa Wilson. 2002. Learning to disambiguate potentially subjective expressions. In *Sixth Conference on Natural Language Learning*, Taipei, Taiwan, August.

[Wiebe et al.1999] Janyce M. Wiebe, Rebecca F. Bruce, and Thomas P O'Hara. 1999. Development and use of a gold standard data set for subjectivity classifications. In *Proc. 37th Annual Meeting of the Assoc. for Computational Linguistics*, pages 246–253.

[Wiebe et al.2002] Janyce Wiebe, Eric Breck, Chris Buckley, Claire Cardie, Paul Davis, Bruce Fraser, Diane Litman, David Pierce, Ellen Riloff, and Theresa Wilson. 2002. NRRC Summer Workshop on Multiple-Perspective Question Answering Final Report. Tech report, Northeast Regional Research Center, Bedford, MA.

[Wiebe et al.2003] Janyce Wiebe, Eric Breck, Chris Buckley, Claire Cardie, Paul Davis, Bruce Fraser, Diane Litman, David Pierce, Ellen Riloff, Theresa Wilson, David Day, and Mark Maybury. 2003. Recognizing and Organizing Opinions Expressed in the World Press. In *Papers from the AAAI Spring Symposium on New Directions in Question Answering (AAAI tech report SS-03-07)*. March 24-26, 2003. Stanford University, Palo Alto, California.

[Wiebe et al.2004] Janyce M. Wiebe, Theresa Wilson, Rebecca F. Bruce, Matthew Bell, and Melanie Martin. 2004. Learning subjective language. *Computational Linguistics*, 30(3):277 – 308, September.

[Wiebe et al.2005] Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation (formerly Computers and the Humanities)*, 39(2-3):165–210.

[Wiebe1994] Janyce M Wiebe. 1994. Tracking point of view in narrative. *Computational Linguistics*, 20(2):233–287.

[Wilson et al.2004] Theresa Wilson, Janyce Wiebe, and Rebecca Hwa. 2004. Just how mad are you? finding strong and weak opinion clauses. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*.

[Wilson et al.2005a] Theresa Wilson, Paul Hoffmann, Swapna Somasundaran, Jason Kessler, Janyce Wiebe, Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. 2005a. OpinionFinder: A system for subjectivity analysis. In *Human Language Technology Conference*, Vancouver, Canada, October. Demo abstract.

[Wilson et al.2005b] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005b. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of Human Language Technologies Conference/Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*. Vancouver, Canada.

[Witten and Frank1999] Ian Witten and E. Frank. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman.

[Xia and Palmer2001] Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the HLT Conference*, March.

[Yi et al.2003] Jeonghee Yi, Tetsuya Nasukawa, Razvan C. Bunescu, and Wayne Niblack. 2003. Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, Melbourne, Florida, December.

[Yu and Hatzivassiloglou2003] Hong Yu and Vasileios Hatzivassiloglou. 2003. Towards answering opinion questons: Separating facts from opinions and identifying the polarity of opinion sentences. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*.