Feature Engineering for a Symbolic Approach to Text Classification

by **Sam Scott**

a thesis submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Computer Science

under the auspices of the Ottawa-Carleton Institute for Computer Science.

Computer Science Department
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario
Canada K1N 6N5

Many thanks to Stan for always being a cheerful and reassuring supervisor. Thanks also to Peter Turney for his help in providing and explaining Extractor system, and to Martin Brooks, Alain Désilets, and Joel Martin for their support and advice. I am also grateful to my examiners, Rob Holte and Franz Oppacher, who offered some very helpful comments and suggestions, and to my parents for their constant support and encouragement.

Special thanks to Anne for her comments on an early draft and for smiling at me when I needed it.

Abstract

Most text classification research to date has used the standard “bag of words” model for text representation inherited from the word-based indexing techniques used in information retrieval research. There have been a number of past attempts to find better representations, but very few positive results have been found. Most of this previous work, however, has concentrated on retrieval rather than classification tasks, and none has involved symbolic learning algorithms.

This thesis investigates a number of feature engineering methods for text classification in the context of a symbolic rule-based learning algorithm. The focus is on changing the standard “bag of words” representation of text by incorporating some shallow linguistic processing techniques. Several new representations of text are explored in the hopes that they will allow the learner to find points of high information gain that were not present in the original set of words. Representations based on both semantic and syntactic linguistic knowledge are defined and evaluated using the RIPPER rule-learning system. Two major corpora are used for evaluation: a standard, widely-used corpus of news stories, and a new corpus of folk song lyrics.

The results of the experiments are mostly negative. Although in some cases the new representations are at least as good as the bag of words, the improvements in quantitative performance that were hoped for do not materialize. However, the results are not entirely discouraging. The syntactically defined representations may enable the learner to produce simpler and more comprehensible hypotheses, and the semantically defined representations do produce some real performance gains on smaller classification tasks that for various reasons fail to scale up to larger tasks. Some ideas are offered as to why the new representations fail to produce better results, and some suggestions are made for continuing the research in future.

TABLE OF CONTENTS

List of Figures

List of Tables

Glossary

BH0: Bag of Hypernyms with *h=0*.

BH1: Bag of Hypernyms with *h=1*.

BK: Bag of Keyphrases.

BN: Bag of Noun phrases.

BNW: Bag of Noun phrases and Words.

BP: See *Breakeven Point*.

Breakeven Point: Hypothetical point at which *precision* = *recall*. See section 4.2.

BW: Bag of Words.

BSK: Bag of Stemmed Keyphrases.

BSN: Bag of Stemmed Noun phrases.

BSW: Bag of Stemmed Words.

Categorization: See *Classification*.

Classification: The task of placing pre-defined labels on previously unseen objects.

DigiTrad: The Digital Tradition corpus of folk song lyrics. See section 2.2.

Extractor: System developed by Peter Turney [TUR98] for extracting *keyphrases* from a document. See section 6.2.

FLIPPER: First order logic version of *RIPPER*. See chapter 6.

F-measure: Statistic to measure the balance between *precision* and *recall*. See section 4.2.

FX: Feature eXtractor system built as part of the current work. See sections 5.2, 6.3, 7.2.3 and Appendix A. See also the "demos" section of the National Research Council's Interactive Information Group web site at: http://ai.iit.nrc.ca/II_public/index.html.

hypernym: Generalized synonym corresponding to the "*is a*" semantic relationship of *WordNet* (*vehicle* is a hypernym of *car*.) See *hyponym* and section 7.1.

hyponym: Specialized synonym corresponding to the "*instance of*" semantic relationship of *WordNet* (*car* is a hyponym of *vehicle*.) See *hypernym* and section 7.1.

ILP: Inductive Logic Programming. See chapter 6.

IR: Information Retrieval (often taken to include both *retrieval* and *classification* tasks.)

Macro-average: Macro-averaged statistics for text classification are computed by averaging similar statistics across a set of classes. See section 4.2.

Machine Learning: *Classification* technique in which a hypothesis is automatically learned from existing labeled objects and then applied to previously unseen objects.

Micro-average: Micro-averaged statistics for text classification are computed from a confusion matrix formed by component-wise addition of other confusion matrices. See section 4.2.

MDL: Minimum Description Length. See section 3.1.

ML: *See Machine Learning.*

NoPE: Noun Phrase Extractor. See section 6.1.

Precision: Proportion of positive classifications that are correct. See section 4.1.

Recall: Proportion of positive labeled examples that are correctly classified. See section 4.1.

Retrieval: The task of matching a short query statement to a set of documents to find the most relevant ones.

RIPPER: Repeated Incremental Pruning to Produce Error Reduction. Machine learning algorithm used in this study. See chapter 3.

Reuters-21578: Standard *text classification* corpus of news stories. See section 2.1.

SGML: Standard Generalized Markup Language. See the following document for more information: "http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG"

SynSet: Basic unit of *WordNet* hierarchy containing synonymous words. See section 7.1.

Text Categorization: See *Text Classification*.

Text Classification: *Classification* task in which the objects to be classified are electronic documents.

USENET: Internet discussion group system. See section 7.2.1.

WordNet: On-line lexical reference system containing semantic relations between words. See section 7.1.

1. Introduction

Text classification is a growing area of research at the intersection of information retrieval (IR) and machine learning. The task of text classification systems is to attach labels automatically to previously unseen electronic documents. These labels may indicate the topics discussed in the document, the relevance of that document to a given user, the mailbox or newsgroup into which the document should be filed, or even the style or authorship of the document. If a significant corpus of pre-labeled documents is already available, then the task is suitable for a machine learning system using this corpus as the training set.[1]

In machine learning for text classification, the system is provided with a function definition $f{:}T \rightarrow C^*$ which maps a document t_i from a corpus of training documents T to zero or more class labels from a pre-defined finite set C. This function is represented explicitly as a set of tuples in the form $<t_i, c_{i1}, c_{i2}, \ldots, c_{ik}>$ where t_i is the i^{th} document in the training set, c_{i1} to c_{ik} are the correct class labels for the document, and k is a non-negative integer. Given this set of tuples, the task of the learner is to form a hypothesis which can enable it to extend the mapping to previously unseen documents. Real-world text classification tasks typically involve thousands of documents, each of which may have any number of class labels attached to it. This differs slightly from a normal machine learning task in which each object is given one unique label, so the standard approach is to learn one hypothesis for each class, then apply these hypotheses to new documents to make binary "yes or no" decisions on whether to assign each class label.

This thesis reports on 16 major experiments in machine learning for text classification. In these experiments the focus is on *feature engineering* using existing sources of learned or engineered knowledge. This is an unusual approach in the text classification literature, as feature engineering is not considered by many researchers to be a fruitful line of inquiry. Instead, efforts tend to be concentrated on new learning algorithms or feature *selection* methods to try to win improvements in performance. However, recent publications concerning machine learning in other domains give reason to believe that feature engineering techniques can improve performance [FAP97, KHM98]. Furthermore, the recent successful application of symbolic machine learning techniques to text classification (notably with the RIPPER decision rule learning system [COS96, COH96a]) opens up new avenues for feature engineering techniques to improve performance.

[1] The terms *text classification* and *text categorization* will be treated as synonymous in this document.

1.1 The Effects of Feature Engineering

Before any objects can be classified by a machine learning system, they must first be represented as sets of features. Every object in the training set must be converted to a vector of feature values that defines a point in a feature space. A simple example is shown in Figure 1 below, in which the task is to distinguish between the classes of *ballet dancers* and *rugby players*. The programmer needs to make a decision at the outset about how these individuals should be represented to the system. (For the sake of presentation, it is assumed that the system can only handle two features at a time.)

In the first feature space, the individuals are represented by measurements of their *height* and *weight*. Since ballet dancers tend to be taller and lighter than rugby players, the individuals cluster nicely into their respective classes and the classification problem can be solved by drawing a single line between the clusters. In the second feature space, individuals are reduced to statistics measuring *% body fat* and *shoe size*. The classes are also separable in this space, although two lines must be drawn this time, and the *% body fat* measurement turns out to be irrelevant. In the final space, features based on *hair color* and *eye color* lead to a scattering of points which cannot easily be separated. In this case, even if the learner was able to find a hypothesis to fit the training data, it would be very unlikely to work well for new data.

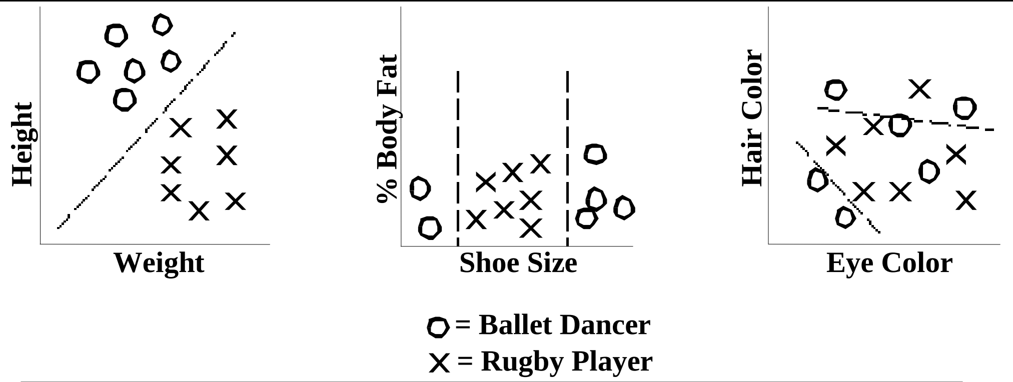

***Figure 1*: An example of feature engineering for machine learning using three different feature sets.**

The toy feature engineering problem above is trivial, but in real-world tasks it is usually not so obvious how to form a good set of features. In some domains an obvious set of features stands out because of the way the data is represented (as lists of medical test results, for example), but in the domain of text classification no such pre-defined features exist. Designers of a classification system must ask a series of questions that are very difficult to answer: What are the relevant features of natural language? More specifically, how can the information in a written document be transformed into a single point in n-space? If a good set of features is found for one type of document, will the same features will work well on others? Will a representation that works well for news stories also work for medical documents, short stories, personal correspondence and poetry? Does it even make sense to think in terms of a "feature space" for natural language?

To complicate matters further, the choice of feature space often affects different learning algorithms differently. For instance, the "*height* vs. *weight"* features in the example above would work well for *linear discriminant* learners such as the Perceptron algorithm. Algorithms of this type, which are popular for text classification, learn the parameters of a single hyperplane to separate the classes in the feature space. This approach will not work well in a feature space such as "*% body fat* vs. *shoe size*" in which the classes are not linearly separable.

Another family of algorithms, sometimes referred to as *symbolic* learning algorithms, would have no trouble with the second feature space. Most symbolic learners form hypotheses based on logical rules in disjunctive normal form, making orthogonal cuts in one dimension at a time. An example of such a rule is shown in Figure 2. Besides performing well, these algorithms have the advantage of forming hypotheses that are more easily comprehensible to a user – an important issue for a user who wants to be able to learn something about the problem in addition to solving it. This comprehensibility is also affected by the choice of feature space. For instance, a decision rule learner would probably be able to form a reasonable hypothesis for the "*height* vs. *weight*" feature space, but it would need many more clauses and rules to do the job properly, and would probably end up forming a less comprehensible hypothesis.

```
If    SHOE_SIZE >7    and    SHOE_SIZE < 10    then    RUGBY_PLAYER
else    BALLET_DANCER
```

***Figure 2*: An example of a hypothesis that might be formed by a symbolic decision rule learner to separate the classes in the feature space of Figure 1 (second diagram).**

1.2 Feature Engineering for Symbolic Text Classification

Most recent work on text classification has involved learning in a feature space known as the "bag of words". Features correspond to words in the corpus and take either binary values indicating presence or absence of a term in a document, or numeric values representing word frequency. This feature space was inherited from the widespread use of words as indexing terms in the information retrieval community. After many decades, researchers have been unable to find a consistently better feature space, leading many of them to pessimistic conclusions about feature engineering for classification [LES96].

It should be emphasized, however, that most of the former work was in the context of *retrieval* and is not necessarily directly applicable to text classification. There are at least three reasons for this. Firstly, the retrieval task is to match a short query statement (usually a simple list of words) to a list of existing documents. Although this can be viewed as a classification task of sorts, it differs in that the classes are not pre-defined and there is no training phase involved. Secondly, it has been pointed out elsewhere that most of the early work on representation for retrieval was based on manual transformations of the words, in which accuracy and completeness were necessarily sometimes compromised [LEW92c]. The speed and memory of modern computers enables such transformations to be done more quickly, efficiently and thoroughly than before. Finally most past work, even in classification, has involved statistical learning algorithms. Symbolic learners are still relatively new and have generally been slow and unable to handle the many thousands of features in the bag of words space. With recent advances in computing power and the development of faster learning algorithms, there has been more interest in using symbolic learning, but always in the context of the old bag of words model (for example [COH96a] and [JOA98]). It is time, therefore, to take a look at feature engineering for symbolic text classification.

1.3 Sneak Previews

As will be outlined in future chapters, there is reason to believe that symbolic learners will perform better with more powerful representations of text. Following on this observation, some new text representations are tested using real-world text classification tasks with a rule learning algorithm named RIPPER, leading to a rather mixed bag of conclusions. While changing the representation of text rarely leads to overall improvements in performance, there are sometimes subtle effects related to training set size or rule comprehensibility that may be relevant to some users.

The chapters that follow describe experiments using eight text representations – two variations of bag of words, two variations of syntactically defined noun phrases (using a method called NoPE based in part on Eric Brill's part of speech tagger [BRI92]), two variations of statistically defined keyphrases (using Peter Turney's Extractor [TUR98]), and two representations based on semantic relationships between words (using the WordNet linguistic reference system [MIL90]). Each of these eight representations is tested on two major corpora: the now-standard Reuters-21578 newswire corpus, and DigiTrad, a corpus of folk songs newly formatted for text classification. An integrated feature extraction tool (FX) is developed to perform the various changes of representation.

The document is laid out in three parts. Part one contains three chapters related purely to methodology: chapter 2 discusses the corpora used in the study, chapter 3 introduces the details of the RIPPER learning algorithm, and chapter 4 provides a discussion of evaluation techniques for text classification. Part two of the document contains complete reports of the experiments performed on the corpora: chapter 5 presents the baseline experiments using the bag of words model, chapter 6 presents the experiments using phrase-based representations, and chapter 7 presents the experiments using WordNet, including some preliminary work previously made public as [SCM98a] and [SCM98b]. Part three of the document contains the final conclusions and suggestions for future work followed by appendices, and references. Previous work is introduced as it becomes relevant to the discussion.

1.4 Summary of Contributions

The major contribution of this work is an examination of feature engineering techniques for text classification in the context of symbolic machine learning. Six new feature types are examined, corresponding to two variations each of noun phrases, keyphrases, and the semantic relationship of hypernymy defined in the WordNet system. Each of these representations is compared to two variations of a representation based on words alone. In the course of developing and testing these representations, a feature extraction tool FX and a companion tool WNPrepare were developed and are now available for other researchers. These tools in conjunction with a part of speech tagger can also be used to implement a simple system of noun phrase extraction referred to as NoPE. A new challenging corpus of folk songs was also introduced and formatted for text classification research. All the tools and the new corpus are available on the web site of the Interactive Information Group of the Institute for Information Technology at the National Research Council of Canada.[2]

[2] Go to http://ai.iit.nrc.ca/II_public/index.html and click on "demos".

Part 1: Methodology

2. The Corpora

The choice of corpora in this work reflects an attempt to balance evaluation between standard corpora that can be used to compare results with other researchers, and difficult or unusual corpora that seem likely to benefit most from alternative text representations. Two corpora were chosen: Reuters-21578, a well known corpus of news articles; and The Digital Tradition, a new corpus of folk song lyrics adapted to text classification for this study. The sections that follow describe these two corpora and outline some methodology related to them.

2.1 Reuters-21578

Reuters-21578 consists of 21578 economic news stories that originally appeared on the Reuters newswire in 1987. Each story has been manually assigned one or more indexing labels from a fixed list, making Reuters-21578 an excellent example of a real-world classification task. The labels are broken into categories such as TOPIC, PEOPLE, and PLACES, but the methodology of most researchers is to consider only the 135 TOPIC labels for classification. Like most text classification problems, these TOPIC labels form a set of binary classification tasks in which the choice is whether to label each example as a *positive* or *negative* example of each class. All of these tasks involve positive classes represented by a small minority of the total training data.

The Reuters corpus was originally developed in the context of a knowledge-engineering approach to classification. Topic categories were defined as part of the process of knowledge extraction, resulting in CONSTRUE - a system of rule-based classification which produced very high levels of accuracy on unseen examples [HAW90]. After the completion of the CONSTRUE work, the corpus was obtained by David Lewis who prepared it for use in machine learning systems as part of the work for his Ph.D. thesis. Lewis eventually oversaw a cleaning-up and formatting of Reuters that has resulted in its present form [LEW92b][3]. Reuters-21578 is freely available from Lewis' web site[4] and is widely used in text classification experiments, making it one of a small number of corpora which can be used as a point of comparison with other studies.

Table 1 contains some statistics relating to Reuters-21578. The corpus contains 21578 examples, just over half of which have been assigned at least one topic label. There are 30765 unique words and the average length of a document is 129 words. Many of the documents have no topic labels attached and are assumed to represent negative examples of every class. Fully one half of the topic labels are represented by fewer than 10 documents, and the 10 most frequent classes account for 75% of the total number of positive classifications (instances of assigned class labels) in the corpus.

In order to use a text corpus for machine learning research, it must be split into sets of training and testing examples. There are a number of standard ways to do this described in the notes distributed with the Reuters-21578 data. The "Modified Apte split"

[3] Many papers refer to Reuters-22173, a previous version of the corpus which contained a number of typos and duplicated examples. This version of the corpus is no longer available, and results from Reuters-21578 are not directly comparable.
[4] http://www.research.att.com/~lewis

(ModApte) first used in [ADW94] was chosen for this research because of the abundance of recent work using this configuration. The ModApte approach splits the corpus at April 7, 1987, placing all documents written on or before this date into the training set, and the rest into the test set. Some 7856 documents are excluded for various reasons, leading to a corpus of 9603 training examples and 3299 testing examples, some of which have no class labels attached. Only classes for which at least one training and test example exists are included, leading to a total of 90 classes as shown in Table 2.

All examples in Reuters-21578 are marked up with SGML tags that bracket the various parts of the document. An example of a document from Reuters-21578 is shown in Figure 3. For our purposes, the important fields are: <*TOPICS*> , which delimits the list of topic categories, <*D*> which delimits individual topics within this list, and <*TEXT*> which delimits the text of the document. The TEXT field is split into sub-fields such as <*TITLE*>, <*DATELINE*> and <*BODY*> but these tags are ignored and all text delimited by the <*TEXT*> tags is included.

It has been noted in previous classification studies that the topic labels in Reuters-21578 often consist of words that themselves serve as good clues for classification. For example, the presence of the word *corn* in a newswire story is very good evidence that the topic label *corn* should be assigned. This fact was exploited to improve classification accuracy in [RGD97]. Nevertheless, the Reuters-21578 corpus does pose some challenging problems for machine learning. Many categories do not correspond directly to words found in the text (*ipi, lei, cpi, bop* etc.), most classes are represented by a very small relative number of positive examples, and some news stories consist of titles only, or simply contain the words "blah blah blah" in the body. Despite these difficulties, however, researchers have obtained good results on the ModApte split of this corpus. The results of this previous work will be reviewed in chapter 5.

Category	**Statistic**	**Reuters-21578**	**ModApte**	
		Overall	Training	Testing
Basic counts	**documents**	21578	9603	3299
	class labels	135	90	90
	classified documents	*(53%)* 11367	*(81%)* 7775	*(92%)* 3019
	words	2783562	1257993	399179
	unique words	30765	27940	16907
	words per document	129	131	121
Classifications	**positive**	14302	9587	3745
	negative	2596636	864286	296464
Distribution of classes	**100+ documents**	*(17%)* 23	*(18%)* 16	*(8%)* 7
	10 - 99 documents	*(33%)* 44	*(49%)* 44	*(48%)* 43
	1 - 9 documents	*(39%)* 53	*(33%)* 30	*(44%)* 40
	0 documents	*(11%)* 15	0	0

***Table 1*: Some statistics for the Reuters-21578 corpus. The statistics in the *Basic counts* category show simple tallies of each item. The statistics in the *Classifications* category show the total number of positive and negative classifications summed for all the available classes. The statistics in the *Distribution of class labels* category indicate the number of classes (topics) that contain the given numbers of documents.**

Topic	Train	Test	Topic	Train	Test	Topic	Train	Test
acq	1650	719	**hog**	16	6	**platinum**	5	7
alum	35	23	**housing**	16	4	**potato**	3	3
barley	37	14	**income**	9	7	**propane**	3	3
bop	75	30	**instaldebt**	5	1	**rand**	2	1
carcass	50	18	**interest**	347	131	**rapeoil**	5	3
castoroil	1	1	**ipi**	41	12	**rapeseed**	18	9
cocoa	55	18	**ironsteel**	40	14	**reserves**	55	18
coconut	4	2	**jet**	4	1	**retail**	23	2
coconutoil	4	3	**jobs**	46	21	**rice**	35	24
coffee	111	28	**lcattle**	6	2	**rubber**	37	12
copper	47	18	**lead**	15	14	**rye**	1	1
copracake	2	1	**lei**	12	3	**ship**	197	89
corn	182	56	**linoil**	1	1	**silver**	21	8
cotton	39	20	**livestock**	75	24	**sorghum**	24	10
cottonoil	1	2	**lumber**	10	6	**soybean**	78	33
cpi	69	28	**mealfeed**	30	19	**soymeal**	13	13
cpu	3	1	**moneyfx**	538	179	**soyoil**	14	11
crude	389	189	**moneysupply**	140	34	**strategicmetal**	16	11
dfl	2	1	**naphtha**	2	4	**sugar**	126	36
dlr	131	44	**natgas**	75	30	**sunmeal**	1	1
dmk	10	4	**nickel**	8	1	**sunoil**	5	2
earn	2877	1087	**nkr**	1	2	**sunseed**	11	5
fuel	13	10	**nzdlr**	2	2	**tea**	9	4
gas	37	17	**oat**	8	6	**tin**	18	12
gnp	101	35	**oilseed**	124	47	**trade**	369	118
gold	94	30	**orange**	16	11	**vegoil**	87	37
grain	433	149	**palladium**	2	1	**wheat**	212	71
groundnut	5	4	**palmkernel**	2	1	**wpi**	19	10
groundnutoil	1	1	**palmoil**	30	10	**yen**	45	14
heat	14	5	**petchem**	20	12	**zinc**	21	13

***Table 2*: A list of the 90 Reuters classes (topics) in the ModApte split. Each class name is followed by the number of examples found in the training and testing sets.**

```
<TOPICS><D>vegoil</D><D>palmoil</D></TOPICS>
<TEXT>
&#2;
<TITLE>INDONESIA SEES CPO PRICE RISING SHARPLY</TITLE>
<DATELINE>    JAKARTA, April 8 - </DATELINE>
<BODY>
Indonesia expects crude palm oil (CPO)
prices to rise sharply to between 450 and 550 dlrs a tonne FOB
sometime this year because of better European demand and a fall
in Malaysian output, Hasrul Harahap, junior minister for tree
crops, told Indonesian reporters.
    Prices of Malaysian and Sumatran CPO are now around 332
dlrs a tonne CIF for delivery in Rotterdam, traders said.
    Harahap said Indonesia would maintain its exports, despite
making recent palm oil purchases from Malaysia, so that it
could possibly increase its international market share.
    Indonesia, the world's second largest producer of palm oil
after Malaysia, has been forced to import palm oil to ensure
supplies during the Moslem fasting month of Ramadan.
    Harahap said it was better to import to cover a temporary
shortage than to lose export markets.
    Indonesian exports of CPO in calendar 1986 were 530,500
tonnes, against 468,500 in 1985, according to central bank
figures.
 REUTER
&#3;</BODY></TEXT>
```

***Figure 3*: An example of a document from the Reuters-21578 corpus. This document should be categorized as both *vegoil* and *palmoil*. The class labels for the document appear delimited by the <TOPICS> and <D> tags at the top. For the classification experiments in this paper, the text of the document was assumed to be delimited by the <TEXT> tags. SGML tags within the <TEXT> field were ignored. In this example, only the relevant SGML fields are shown.**

2.2 The Digital Tradition

The Digital Tradition (DigiTrad) is a new corpus for text classification research introduced by Scott and Matwin [SCM98a]. It is a freely-available collection of folk song lyrics maintained by the folk song enthusiasts at the "MudCat Café" [GRE96].[5] In order to aid users, the owners of DigiTrad have bundled the files with a third-party search engine and assigned to most songs a set of keywords from a fixed list. These keywords often indicate the topical theme of the song (e.g. *marriage*, *murder*), but some of them also provide information about the geographic origin or genre of the song (*Irish*, *Canada*, *gospel* and *parody* are some good examples). Though DigiTrad is an unusual corpus for classification research, it is a typical example of the type of data that users of the world wide web deal with every day. A text classification system capable of handling this type of text could be of enormous help in automating the process of submitting and indexing new songs.

In order to make DigiTrad usable for machine learning, some preprocessing was performed on the raw text. After they were separated from the search engine, the documents each consisted of the following fields: the song title in upper case; the song lyrics; historical notes about the song (optional); the list of keywords (optional); and finally some information on where to find the appropriate sound files for the song (optional). Each song terminated with the ASCII character 0. To make the files more usable, the raw text was marked up with SGML tags in a style similar to Reuters-21578. First the NULL characters were removed and the songs, including titles and end notes, were delimited by the <*TEXT*> tags. Then the keywords were placed at the tops of the songs delimited by the <*TOPICS*> and <*D*> tags used in Reuters-21578. This formatting was done with a small program written for the purpose in C.

The type of texts typical of DigiTrad are very different from those in Reuters-21578. Sentence structure is loose and often non-existent; clever, flowery and rhyming language is generally preferred over clarity; songs are often written in dialect; and the writers of the songs tend to be indirect, often skirting around a topic without explicitly mentioning it. Figure 4 shows some typical examples of songs with these qualities. Some of the differences between DigiTrad and Reuters-21578 can also be measured statistically. Table 3 shows statistics for DigiTrad that can be compared to the equivalent Reuters-21578 statistics in Table 1. Note that DigiTrad contains fewer word instances but more unique words than Reuters. Measuring the proportion of unique words to total words for each corpus shows that the DigiTrad domain contains on average 2.71 times as many unique words as Reuters-21578. DigiTrad also has a much higher number of classes (672 versus 135), many more of which are represented by fewer than 10 documents (72% vs. 39%). All these quantitative and qualitative traits make DigiTrad an interesting challenge for text classification systems. The newly formatted DigiTrad corpus could become a new standard test collection for classification research.

[5] Available for download at http://www.deltablues.com/folksearch.html

```
<TOPICS><D>USSR</D><D>political</D><D>parody</D></TOPICS>
<TEXT>T-T-T-TROTSKY
   T-T-T-Trotsky, beautiful Trotsky,
   You're the B-B-B-Bolshevik that I adore
   And when the Red Flag flies over the Soviet
   I'll be waiting at the K-K-K-Kremlin door

- League for Socialist Action, Vancouver, 1975.
filename[ TTROTSKY
play.exe KKKKATY
JB
</TEXT>

<TOPICS><D>kids</D><D>Welsh</D></TOPICS>
<TEXT>TAFFY WAS A WELSHMAN
Taffy wiz a Welshman;
  Taffy wiz a thief;
Taffy cam' tae my hoose
  And stailt a bit beef.
I gaed tae Taffy's hoose,
  And there, upon my life,
When Taffy wisna lookin',
  I stole eez wife.
If he cam' back
  It wud be a great relief.
He cud keep his wife;
  I'd raither hae my beef.

Rodger Lang Strang (1948), 14.  Cf. ODNR 400 (no. 495),
first ref. 1780.
Note: It popped up in one of the earliest versions of Mother
    Goose (late 18th Century) RG

filename[ TFFYWLCH
MS
</TEXT>
```

***Figure 4*: Two examples of texts from the DigiTrad corpus, shown after conversion to SGML format. The keywords from the original database are delimited by the <TOPICS> and <D> tags at the top of each song. All text associated with the song, including title, lyrics and end notes, has been delimited by the <TEXT> tags. Note the differences in punctuation and use of language in these examples.**

In order to use the corpus for machine learning, it was necessary to decide how to split the data into training and testing sets and how to define a set of classes for the learning task. The first decision was to split DigiTrad into training and testing sets consisting of 2/3 and 1/3 of the documents respectively. The split was made by simply removing every third example from the original corpus (which was originally arranged in alphabetical order by song title.) The second decision was to train and test classifiers only for the classes with 100 or more examples in the total corpus. The training/testing split thus defined is referred to as "DT100". Statistics for DT100 are reported in Table 3, and the 33 classes are listed in alphabetical order in Table 4.

Category	Statistic	DigiTrad	DT100	
		Overall	Training	Testing
Basic counts	documents	6499	4333	2166
	class labels	672	33	33
	classified documents	(94%) 6084	(94%) 4090	(94%) 2036
	words	1748231	1161244	576156
	unique words	52343	43360	29415
	words per document	269	268	266
Classifications	positive	14366	5653	2892
	negative	4353015	137336	68586
Distribution of classes	100+ documents	(5%) 33	(61%) 20	(33%) 11
	10 - 99 documents	(23%) 157	(39%) 13	(67%) 22
	1 - 9 documents	(72%) 482	(0%) 0	(0%) 0
	0 documents	(0%) 0	(0%) 0	(0%) 0

***Table 3*: Some statistics for the DigiTrad corpus. The statistics in the *Basic counts* category show simple tallies of each item. The statistics in the *Classifications* category show the total number of positive and negative classifications summed for all the available classes. The statistics in the *Distribution of class labels* category indicate the number of classes (key words) that contain the given numbers of documents.**

Topic	Train	Test	Topic	Train	Test	Topic	Train	Test
America	177	77	**drink**	184	101	**outlaw**	115	69
animal	267	124	**English**	89	46	**parody**	259	121
army	80	50	**family**	96	57	**parting**	141	81
battle	75	30	**food**	82	42	**political**	128	69
bawdy	136	73	**home**	68	35	**religion**	177	63
Canada	73	34	**Irish**	249	124	**sailor**	375	189
Civil	78	30	**kids**	155	70	**Scots**	356	165
country	92	47	**love**	487	264	**seasonal**	88	30
courting	215	138	**marriage**	137	77	**soldier**	84	57
cowboy	77	39	**murder**	145	93	**war**	257	140
death	330	163	**music**	87	40	**work**	294	157

***Table 4*: A list of the 33 DigiTrad classes (keywords) used in this study followed by the number of examples found in the training and testing sets.**

3. The Learning Algorithm

Most text classification work to date has involved statistical algorithms such as Naive Bayes, Widrow-Hoff, or k-Nearest Neighbor. The high-dimensionality of most representations of text effectively prohibited the use of many other standard machine learning systems such as C4.5. However, recent developments have made possible the use of such systems: the ever increasing speed and capacity of modern computer workstations is helping to render these problems more solvable, and a new rule-based concept learning algorithm has been developed that is comparable in accuracy to C4.5 but considerably faster.

RIPPER was developed by William Cohen [COH95a] based on repeated application of Furnkranz and Widmer's [FUW94] IREP algorithm. (IREP stands for Incremental Reduced Error Pruning and RIPPER stands for Repeated Incremental Pruning to Produce Error Reduction.) The RIPPER algorithm represents a significant performance improvement over previous rule induction algorithms. For a training set of size *n*, RIPPER's performance scales as $O(n \log^2 n)$ whereas C4.5Rules scales as $O(n^3)$ [FUW94] and previously proposed algorithms such as REP and Grow scale as $O(n^4)$ [COH95a]. Further speedup on text classification problems is obtained through the use of set-valued features implemented as sparse binary vectors [COH96b]. This speedup comes with no loss of accuracy on noisy data. Cohen [COH95a] compared RIPPER and C4.5Rules and found that RIPPER produced rules of comparable or better accuracy on 22 of 37 benchmark data sets.

RIPPER has already been applied to a number of standard problems in text classification with quite promising results, and FLIPPER, a first-order logic version, was developed in an attempt to study the use of phrases for text classification [COH95b, COH95c]. It is important to emphasize that RIPPER is a rule-based machine learning system that has made its mark in a field dominated by purely statistical methods. This makes RIPPER interesting for this study since most conclusions about the effectiveness of various representations have been drawn in a context that may not apply to RIPPER. All the relevant previous work will be discussed fully in later chapters, but for now it is worth discussing in detail the operation of the algorithm on which RIPPER is based. These details are important in considering how to engineer features for better performance.

3.1 How RIPPER works

In order for objects to be classified by most learning algorithms, they must first be transformed into a representation suitable for concept learning. The discussion of how this transformation should be done for electronic texts is of course the major topic of this study and will be discussed in detail later. For now it is important to re-iterate that all representations must consist of a vector of *features*, each describing some aspect of the objects to be classified. In most machine learning systems, a feature may be either *nominal* (including binary) or *continuous*. Nominal features are those that take one of a finite number of pre-defined values, whereas continuous features are those that take on integer or real numeric values. For example, consider the task of categorizing types of vegetation. The nominal feature *leaves* could describe the type of leaves observed using values from the set {*round*, *oval, jagged, none*}, while the continuous feature *height* could describe the height of a plant in meters and take any positive real value.

In RIPPER, as with other rule induction systems, a decision rule is defined as a sequence of Boolean clauses linked by logical AND operators that together imply membership in a particular class. The clauses are of the form $A=x$ or $A\neq x$ for nominal attributes and $A\leq x$ or $A\geq x$ for continuous attributes. A classification hypothesis is a sequence of rules usually ending in a default rule with an empty set of clauses. During classification, the left hand sides of the rules are applied sequentially until one of them evaluates to true, and then the implied class label from the right hand side of the rule is offered as the class prediction. An example of an induction rule is shown in Figure 5.

To explain the operation of RIPPER, we consider the restricted case in which the examples fall into one of two classes: *positive* or *negative*. A high level view of the algorithm is presented in Figure 6, which shows the original steps in the IREP algorithm and the additions made in RIPPER. To the basic algorithm, RIPPER adds several *rule optimization* steps as well as the option to improve the rule set by repeating the entire process for a number of iterations. These additions will be discussed, but first an overview of the IREP algorithm is presented.

The original IREP algorithm forms rules through a process of repeated *growing* and *pruning*. During the growing phase the rules are made more restrictive in order to fit the training data as closely as possible. During the pruning phase, the rules are made less restrictive in order to avoid overfitting, which can cause poor performance on unseen examples. IREP splits the training examples into a *growing set* and a *pruning set*. Rules to predict the *positive* class are grown one at a time by starting with an empty rule and then adding clauses to the left hand side in a greedy fashion under the guidance of a *grow heuristic*. Growing of a single rule stops when it covers no negative examples from the growing set. Each rule is pruned immediately after it is grown by deleting clauses that cover too many negative clauses in the pruning set under the guidance of a *prune heuristic*. After a new rule is grown and pruned, the covered examples are removed from both the growing and pruning set. Then the remaining data is repartitioned and another rule is grown. This rule growing process continues until all the examples in the training set are covered or until some *stopping condition* is reached.

(Leaves = None) $\Rightarrow$ Grass
(Color = Green) and (Height $\geq$ 5) and (Fruit = Legume) $\Rightarrow$ Beanstalk
(Leaves = Oval) and (Height $\geq$ 50) $\Rightarrow$ OakTree
Default $\Rightarrow$ Clover

***Figure 5*: A hypothesis for classifying four types of vegetation. The nominal attributes are *Leaves, Color, Fruit* and *Bark*. The continuous attributes are *Height* and *StemWidth*. The classes are *Grass, BeanStalk, OakTree* and *Clover*. During classification, the rules will be considered one by one in the order they appear in the hypothesis. The first left hand side to evaluate to true will be assumed to indicate the correct class. Note that not all the features are used in the hypothesis.**

```
LOOP n TIMES
  Start with the empty rule (TRUE ⇒ positive).
  LOOP UNTIL the stopping condition is reached.
    Partition the training set into a growing set and a pruning set.
    Grow a rule by greedily adding a clause to the left hand side guided by the grow heuristic.
    Prune a rule by greedily deleting sequences of final clauses guided by the prune heuristic.
    Remove examples covered by the rule from the training set
  END LOOP.
  Perform rule optimization on the entire rule set.
END LOOP.
```

***Figure 6*: High-level description of the RIPPER algorithm. The bold-faced lines are new in RIPPER, while the rest of the algorithm is an implementation of IREP.**

During the rule growing phase, the goal is to add clauses greedily to an initially empty rule in such a way that the set of examples covered by the rule contains a maximum of positive examples and a minimum of negative examples. The grow heuristic used in RIPPER is the *information gain* function proposed by Quinlan [QUI90]. The algorithm starts with a set T_o consisting of all examples remaining in the growing set. At the i^{th} iteration of the rule growing algorithm, the learner is working with a set T_i consisting of t_i^+ positive examples and t_i^- negative examples. A measure of the information required to describe the class membership of all the examples is:

$$l(T_i) = -\log_2\left(\frac{t_i^+}{t_i^+ + t_i^-}\right)$$

The goal is to reduce the total amount of information. When a new clause A_i is added to a rule, a new set of examples T_{i+1} is formed consisting of all examples from T_i covered by the new rule with A_i added. Adding a clause can only restrict the coverage of a rule, therefore T_{i+1} must always be a subset of T_i, though the new set may still contain both positive and negative examples. The information required to describe the new state is:

$$l(T_{i+1}) = -\log_2\left(\frac{t_{i+1}^+}{t_{i+1}^+ + t_{i+1}^-}\right)$$

If the addition of A_i reduces the number of negative examples covered by the rule in relation to the number of positive examples, then the information required to describe the set will tend to decrease. A drop in the amount of information from T_i to T_{i+1} represents a *gain* in the amount of information contained in the rule under construction. But the goal is also to cover as many positive examples as possible, so the *gain heuristic* is defined as the number of remaining positive examples multiplied by the gain in information. So if t_i^{++} of the positive examples from T_i are still present in T_{i+1}, then:

$$Gain(A_i) = t_i^{++} \cdot (l(T_i) - l(T_{i+1})) .$$

After a rule is grown it will ideally cover many positive examples and no negative examples in the growing set. However, for noisy data it is not always desirable for the rules to fit the particular idiosyncrasies of the training data too closely. Rules that *overfit* the training data may perform poorly on unseen examples. The function of the pruning stage, therefore, is to relax the rule so that it is more general and less prone to overfitting. In order to do this, the rule is tested against the examples in the pruning set. If the rule overfit the growing set, it will cover negative examples from the pruning set. The prune heuristic measures this coverage. The ideal is for the rule to cover many positive examples and no negative examples. Clauses are deleted to maximize the function:

$$v = \frac{p-n}{p+n} ,$$

where p and n are the number of positive and negative examples in the pruning set that are covered by the rule. Note that this value is maximized when $n = 0$. In the original IREP implementations, clauses are deleted one by one in reverse order until no deletion can be found that increases the value of v. RIPPER extends this process to also consider dropping *sequences* of final clauses.

Finally, a stopping condition is used to decide when to stop adding rules to the hypothesis. Furnkranz and Widmer propose a heuristic of stopping when the error rate of the next rule is greater than that of the empty rule. In RIPPER this heuristic is replaced with one based on the *Minimum Description Length* (MDL) principle from information theory. This criterion, described in [QUR89], is an elegant formula that balances accuracy against complexity based on the number of bits required to communicate complete and correct class information for a set of examples. The *description length* is obtained by adding the number of bits required to describe the classification hypothesis to the number of bits required to enumerate the exceptions to this hypothesis. Attempting to minimize this measurement will bias the learner towards simple and accurate rules. RIPPER stops adding rules when the new description length is more than 64 bits larger than the best description length so far.

As a refinement to the process described so far, RIPPER also includes two *rule optimization* steps. In the first step, each rule is considered in turn and two new potential replacement rules are grown. The first rule is grown and pruned starting with the empty rule as before, and the second is grown starting with the original rule instead of the empty rule. The main difference is that both rules are grown and pruned so as to optimize the error rate of the *entire* rule set on the *entire* pruning set. After the new rules are formed, the decision on which of the three candidate rules to include in the hypothesis is guided by the MDL heuristic. Finally, RIPPER simply repeats the entire algorithm a number of times to try to cover any remaining positive examples. The number of iterations of this process is subject to a parameter set by the user. In the current experiments the default value of 2 is always used.

The description of RIPPER given above is for a two-class problem. RIPPER handles multiple classes by ordering them from least to most prevalent and then treating each in order as a distinct two-class problem. So if the classes are ordered C_1 to C_k, RIPPER first learns rules to distinguish the least prevalent class C_1 from classes C_2 to C_k. Then all examples covered by these rules are removed, and RIPPER learns rules to distinguish C_2 from C_3 to C_k. This continues until only the most prevalent class C_k remains and this is used as the default class [COH95a]. Note that this only applies to non-overlapping classes. When classes overlap, as they usually do in text classification, the only sensible choice is to train one binary classifier for each class, so the effect of RIPPER's class ordering means that rules will always be learned to cover the positive class first. The negative class will be left as the default.

3.2 RIPPER and Text Classification

The speed and accuracy of the RIPPER algorithm makes it a good candidate for the high dimensionality representations typical of text classification. Furthermore, RIPPER has the advantage of allowing set-valued features [COH96b] in which a set of atomic values can be defined as a single feature. The simplest way to use set-valued features for text classification is to define a single feature named *WORDS* and use the sets of all words that appear in each example as the feature values. RIPPER will consider the entire training set and assemble a binary feature vector in which each unique word in the corpus is represented by a single feature. For a given document, each feature has the value 1 if the word appeared in that document and 0 otherwise. This capability of RIPPER allows for simple and intuitive representation of text, a further speedup through the use of a sparse vector representation, and very easily comprehensible rules as shown in Figure 7. RIPPER has been used with good results in a number of text classification experiments such as those reported in [COH95a], [COS96], and [COH96b].

Hypothesis:
(beef $\in$ WORDS) $\Rightarrow$ carcass
(pork $\in$ WORDS) and (bellies $\notin$ WORDS) and (number $\notin$ WORDS) $\Rightarrow$ carcass
(meat $\in$ WORDS) and (told $\in$ WORDS) $\Rightarrow$ carcass
(feed $\in$ WORDS) and (main $\in$ WORDS) and (grain $\in$ WORDS) $\Rightarrow$ carcass
(poultry $\in$ WORDS) and (march $\notin$ WORDS) $\Rightarrow$ carcass
default $\Rightarrow$ not carcass

***Figure 7*: A RIPPER-style rule learned for the Reuters-21578 class *carcass* (formatted for ease of reading.).**

3.3 Rule Induction vs. Statistical Pattern Recognition

Statistical text classification systems such as Naive Bayes [MIT97], Rocchio [ROC71], and Widrow-Hoff [RMP86] have a number of important differences from rule learning text classification systems like RIPPER. Assuming the use of words as features, rule induction systems:

i) form hypotheses consisting of intensional (propositional) rules that are comprehensible to a human reader,

ii) use a "greedy" search heuristic to guide their search through the space of possible hypotheses,

iii) adhere to the principle of Occam's Razor, deliberately choosing a small subset of words on which to base classifications, and

iv) allow the context of a word to influence how that feature is used during classification [COS96]. (For example, one rule in Figure 7 assigns the category *carcass* to a document if it contains the word *poultry*. However, it will only do so if the document also does not contain the word *march*, thus allowing the context in which the word *poultry* appears to influence how it is used for classification.)

Statistical learners on the other hand:

i) form less comprehensible hypotheses based on the number of occurrences of each word in each document and optionally, in the corpus as a whole,

ii) always take all features into account, and

iii) pay no attention to the context in which a word occurs.

These differences are important enough that feature engineering methods can be expected to have different effects on the different learners. For example, the use of all available features in the statistical paradigm means that results can often be improved by feature selection strategies aimed at producing an optimal subset of the available words, and by feature engineering strategies such as stemming (discussed further in chapter 4) that combine similar words into common features [LEW92b, LEW98]. On the other hand, feature selection is built into a rule-based learner's search heuristic, implying that a priori reduction of the feature space should be avoided [COH96a].

It is appropriate, therefore, to carefully consider the characteristics of a learning system when trying to interpret its performance with a given feature engineering technique. While remaining aware of previous work, we should not assume that results arrived at in investigations of feature engineering for statistical systems will *necessarily* apply to rule-based systems.

4. Evaluation

This chapter looks at the various methods that will be used to compare different representations. The discussion begins with a review of some commonly used basic performance statistics, then goes on to discuss higher-order statistics such as the *F-measure* and the *breakeven point* and the use of *micro* and *macro-averaging*. Finally a method is proposed for evaluating rule comprehensibility.

4.1 Accuracy, precision, and recall

In most text classification experiments a single classifier is trained for each class using the available training data. Then each classifier is tested using the data in the testing set, and the classifications are counted in four categories: correct positives, correct negatives, false positives, and false negatives. In this context, *positive* means the example is (or was classified as) a member of the target class and *negative* means the example is not (or was not classified as) a member of the target class. These four statistics can be assembled into a simple contingency table, or confusion matrix, as shown in Figure 8 below.

		Hypothesis	
		+ve	**-ve**
Correct Class	**+ve**	**a**	**b**
	-ve	**c**	**d**

***Figure 8*: An example of a confusion matrix for a single class. The numbers *a* and *d* represent correct positive and negative classifications, and the numbers *b* and *c* represent the number of false negatives and false positives respectively.**

In the machine learning community, it is common to consider the *accuracy* of a classifier on the testing set as a good indication of its performance. Accuracy is defined simply as the number of correct classifications divided by the total number of classifications. Using the notation from Figure 8, this can be expressed as:

$$A = \frac{a + d}{a + b + c + d}$$

For text classification, there are usually many times more negative than positive examples in the testing set, meaning that even default classifiers that always classify as negative can yield accuracies of 95% or more. From Table 1 and Table 3 in chapter 2, we can deduce that a default classifier would achieve 98.6% accuracy for the testing set of Reuters-21578 and 96.0% accuracy for DigiTrad. To avoid this problem, the IR community usually considers two different statistics known as *precision* and *recall*. Roughly stated, precision (P) measures the proportion of examples classified as positive that are correctly classified, and recall (R) measures the proportion of examples which really are positive that are classified correctly. The formulae for precision and recall are:

$$P = \frac{a}{a+c}, \text{ and } R = \frac{a}{a+b}.$$

One problem with these statistics is the potential for *P* or *R* to take the value *0/0*. Usually this value is defined to be either *0* or *1*. Unfortunately, this choice of definition can sometimes have a large effect on averaged results. Other solutions have been proposed, such as defining *0/0* to be a very small positive number, but most of these seem at least as arbitrary as the obvious solutions (see review in [LEW92b].) In this study, statistics will always be presented for cases in which *0/0=0* and *0/0=1*.

In practice, increases in *P* often come at the expense of *R* and vice-versa. An extremely conservative classifier will produce *P >> R*, while an extremely eager classifier will produce *R >> P*. This creates problems in comparing classification methods. Consider the example results shown in Table 5, in which one method of classification yields 0.45 precision and 0.65 recall and another yields 0.60 precision and 0.50 recall. How does one compare these two methods? Lurking within this question is a second, more fundamental one: is high precision more or less important than high recall? The answer to this latter question depends entirely on the user's preference.

	Precision	Recall
Method 1	0.45	0.65
Method 2	0.60	0.50

***Table 5*: An example of two results that are difficult to compare simply using precision and recall.**

4.2 The F-measure and breakeven point

One popular evaluation method for balancing precision and recall is the F-measure proposed by van Rijsbergen [RIJ79] and reviewed recently by Lewis [LEW95]. The F-measure combines *P* and *R* with a new parameter β specifying the importance of *R* relative to *P*:

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R} = \frac{(\beta^2+1)a}{(\beta^2+1)a + b + \beta^2 c}.$$

If β*=1* (*P* and *R* are equally important), then this statistic is essentially the average of *P* and *R* with a penalty term that grows as the distance between *P* and *R* increases:

$$F_1 = avg - \frac{\Delta^2}{avg}, \text{ (where } \Delta = |avg - P| = |avg - R|\text{)} \quad \text{[TUR98].}$$

To deal with the issue of the relative importance of *P* and *R*, results are often reported for F-measures where β*=2* (recall twice as important as precision) and β*=0.5* (recall half as important as precision) in addition to $\beta=1$. These statistics will sometimes be denoted using a subscript for β (as in $F_{0.5}$-measure, or simply $F_{0.5}$). So to re-visit the example from Table 5, the first classification method would yield $F_2=0.59$, $F_1=0.53$ and $F_{0.5}=0.48$, while the second would yield $F_2=0.52$, $F_1=0.55$ and $F_{0.5}=0.58$. In other words, if recall is more important, the first method is better, but if precision is more important, the second is better. If they are equally important, the second method is again better since there is less spread between the values of *P* and *R*.

A second solution to the problem of comparing performance is to use the *hypothetical breakeven* point of precision and recall. Many learning systems contain parameters that can be manipulated with the effect of altering the ratio of precision to recall. (In RIPPER a parameter called the *loss ratio* varies the weight of false negative classifications relative to false positives and in naïve Bayes the probability threshold for a positive classification can be adjusted [LEW92b].) The breakeven point is found by varying this parameter to find the approximate value at which precision and recall are equal. Usually a fixed number of values are tried, the resulting precision and recall values are plotted.[6] A hypothetical breakeven value can then be interpolated by drawing a line through two points which bracket it or extrapolated by drawing a line through the two closest points. Three examples of this process are shown in Figure 9. Note that unlike the F-measure, the breakeven point always assumes that recall and precision are of equal importance – an assumption which may not be valid for a given application. Despite this bias, and the fact that breakeven points have been criticized in recent papers [JOA98], the statistic remains a popular evaluation method.

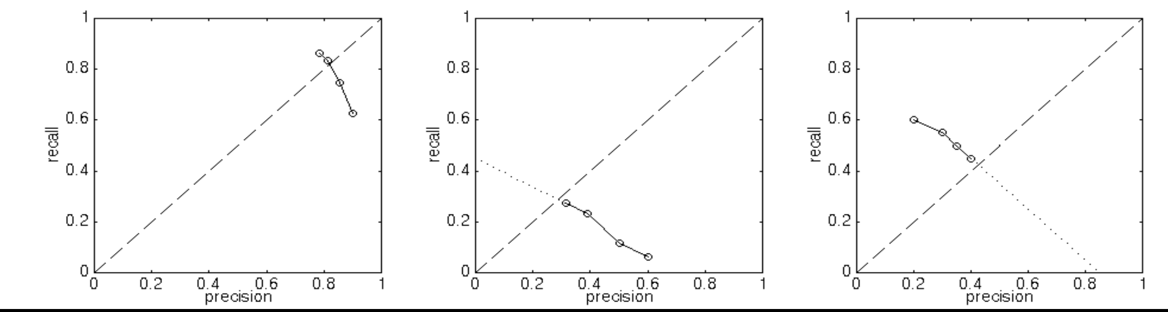

***Figure 9*: Three sample graphs of precision and recall. The first graph shows an interpolated breakeven point (where the solid line crosses the dashed line.) The second two graphs show extrapolated breakeven points.**

4.3 Averaging Techniques

So far the entire discussion has dealt with how to measure effectiveness on a single class. But most text classification tasks contain many dozens of classes, each of which will have their own F-measure or breakeven points. Typically, researchers deal with this in one of two ways: *macro-averaging* in which precision and recall are simply averaged over all classes; or *micro-averaging* in which a new confusion matrix is formed by component-wise addition of the confusion matrices for each class, and then precision and

[6] These graphs are similar to the ROC curves introduced by [PRF97]. ROC curves plot the *True Positive Rate* (recall) against *False Positive Rate* defined as $FP=b/(b+d)$.

recall are computed from this aggregate matrix. This process is illustrated in Figure 10, which shows the micro and macro-averaged F-measures for a three class categorization problem. Notice that the good performance on the frequent class B raises the micro-averaged $F_{1.0}$-measure, while the poor performance on the two infrequent classes lowers the macro-averaged $F_{1.0}$-measure. (The "average" $F_{1.0}$-measures are computed from the averaged precision and recall statistics. They are not the same figures one would get from averaging the three original $F_{1.0}$-measures.) Basically, the micro-average is a weighted average biased in the direction of more frequent classes (that are easier to learn) and the macro-average is biased in the opposite direction, towards more infrequent classes (that are more difficult to learn.) Neither of these biases may be desirable. Furthermore, macro-averaging is much more likely to run into problems with the *0/0* anomaly discussed in section 4.1, particularly when many classes are very infrequently represented.

Hypothesis

Correct Class

A	**+ve**	**-ve**
+ve	**10**	**23**
-ve	**54**	**3038**

Precision:	0.16
Recall:	0.30
$F_{1.0}$:	0.21

B	**+ve**	**-ve**
+ve	**1000**	**34**
-ve	**12**	**2079**

Precision:	0.99
Recall:	0.97
$F_{1.0}$:	0.98

C	**+ve**	**-ve**
+ve	**5**	**3**
-ve	**50**	**3067**

Precision:	0.09
Recall:	0.63
$F_{1.0}$:	0.16

Combined Matrix

A+B+C	**+ve**	**-ve**
+ve	**1015**	**60**
-ve	**116**	**8184**

Macro-Averaged

Precision:	0.41
Recall:	0.63
$F_{1.0}$:	0.50

Micro-Averaged

Precision:	0.94
Recall:	0.90
$F_{1.0}$:	0.92

***Figure 10*: Confusion matrices for three classes A, B and C. A and C have very few positive examples in the testing set, while B has many. Below the original confusion matrices is the combined matrix obtained by component-wise addition, and the macro- and micro-averaged precision and recall values.**

4.4 Comprehensibility

One of the claims that is often made about rule-based learners is that they produce more easily *comprehensible* hypotheses than statistical methods. Their output is easily readable and understandable to the user. In investigating the effects of different representations, it would be useful to have some objective measure of "comprehensibility" that could be used to evaluate the hypotheses produced using different representations. The question that will need answering is, "does a new representation result in the production of hypotheses which are generally easier to understand?"

This question is necessarily quite subjective, but the observation can be made that hypothesis *complexity* is related to comprehensibility – that is, a hypothesis with fewer rules and fewer clauses per rule is likely to be easier for a human to understand than one with many rules and clauses. An example of this effect is shown in Figure 11. So computing the average number of rules and the average number of clauses per rule for each representation can provide a rough estimate of comprehensibility. If one representation produces numbers which are lower, then that representation may be enabling RIPPER to produce more comprehensible rules.

Hypothesis 1 (words):
(retail $\in$ WORDS) and (sales $\in$ WORDS) and (mln $\notin$ WORDS) $\Rightarrow$ retail
(retail $\in$ WORDS) and (of $\notin$ WORDS) $\Rightarrow$ retail
(durable $\in$ WORDS) $\Rightarrow$ retail
default $\Rightarrow$ not retail

Hypothesis 2 (key phrases):
(“retail sales” $\in$ PHRASES) $\Rightarrow$ retail
(“german retail” $\in$ PHRASES) $\Rightarrow$ retail
(“durable goods” $\in$ PHRASES) $\Rightarrow$ retail
default $\Rightarrow$ not retail

Hypothesis 3 (noun phrases):
(“retail sales” $\in$ PHRASES) $\Rightarrow$ retail
(“goods orders” $\in$ PHRASES) $\Rightarrow$ retail
(“household consumption” $\in$ PHRASES) $\Rightarrow$ retail
default $\Rightarrow$ not retail

***Figure 11*: Three hypotheses formed by RIPPER for the *retail* class of *Reuters-21578*. The top rule, found using words as features, achieved 0% precision and recall on the testing set. The bottom two rules, found using phrases as features, each achieved 16.7% precision and 50.0% recall on the testing set. The more powerful rules are also more comprehensible in this case, partly due to the fact that they contain fewer clauses. (This example is from the experiments described in Part 2.)**

4.5 Summary: Evaluation methods used in this study

The chapters in part 2 of the thesis will report the overall micro- and macro-averaged breakeven points for each representation. Macro-averaged results will be presented for *0/0=0* and *0/0=1*. This will give some sense of overall performance, and enable comparison with other recent studies. In an attempt to find more subtle differences between representations, the best overall micro- and macro-averaged F-measure will be reported for β*=2, 1* and *0.5*. For further analysis the classes in each corpus will be grouped by number of positive training examples, and micro-averaged breakeven and F-measure statistics will be reported for each group. This will highlight the effects of the different representations on more or less frequent classes. Finally, the average complexity of the rules generated for each representation will be compared to get a rough measure of comprehensibility.

Part 2: Experiments

5. Word-Based Representations

The simplest and most common way to represent text for classification is to assemble a feature set in which each feature corresponds to a word found in the text. This approach (often referred to as the bag of words) forms the basis of most representations for text classification in both the machine learning community (see work by Lang [LAN95], Joachims [JOA97, JOA98], Koller and Sahami [KOS96], and Cohen [COH96a]) and the information retrieval community (see work recent work by Ng et al. [NGL97] or reviews by Lewis in [LEW92b], [LEW92c] and [LES96]). This chapter is devoted to exploring two variations on the bag of words model. The work will serve as a benchmark for comparing RIPPER's performance to other algorithms, and for comparison to the alternative representations explored later.

5.1 Methods and Representations: general considerations

For the bag of words representation, a word is defined to be any string of alphabetical characters delimited by spaces or punctuation. The words used as features are taken from a *local dictionary* consisting of words found in the training corpus with case information and punctuation removed. Often a list of *stop words* (functional or connective words that are assumed to have no information content) is removed from the original text. A typical list of stop words can be found in [LEW92b]. Sometimes the entire word set is used with no further processing (RIPPER is designed specifically with this in mind [COH96a]) but more typically, there is some attempt to make the features more general and thus reduce dependencies and redundancies.

The most common way to generalize words is to apply a *stemming* algorithm such as the one developed by Lovins [LOV68] to remove suffixes from words. Stemming has the effect of mapping several morphological forms of words to a common feature. For example the words *learner*, *learning*, and *learned* would all map to the common root *learn*, and this latter string would be placed in the feature set rather than the former three, thus reducing the overall number of features[7]. In a statistical learning paradigm, this has the advantage of removing some of the dependencies between the features. The effect of stemming in a symbolic learning paradigm is not so clear.

Stemming and stop-word removal are so widely regarded as useful that they are almost universally used in text classification experiments. However, some recent work has cast doubt on the universality of their effectiveness for retrieval [RIL95]. The chief advantage of stop-word removal is reduction of data set size. Since disk space is becoming less and less of a concern, and since it was observed in section 3.3 that feature selection is not necessarily desirable for RIPPER, this technique was not used. As for stemming the advantages and disadvantages are less clear, so classification was attempted using both unstemmed words and words stemmed using the Lovins stemmer.

[7] As a test, the Lovins stemmer was run on the entire Reuters-21578 and DigiTrad corpora. In both cases, the number of unique stems was 33% less than the number of unique words. (Reuters-21578 had 20534 stems vs. 30765 words, and DigiTrad had 35176 stems vs. 52343 words.)

In some experiments reported in the literature, the bag of words features are binary, indicating the presence or absence of a word. In others an attempt is made to use word frequency measurements. The naïve Bayes technique, for instance, was developed to work using word frequencies [MIT97], while RIPPER was designed with binary features in mind [COH96a]. It is also possible to assemble features based on frequency for RIPPER, but this greatly increases training time since RIPPER is no longer able to take advantage of its sparse vector representation. Because of this restriction, only binary features are used in these experiments.

Most work in text classification has employed a *feature selection* technique in which an attempt is made to reduce the number of features to the most "significant" set. Methods to do this include the selection of only the first *k* words of each document [COH96a], the use of Zipf's rule which states that infrequent and frequent words can be eliminated [KOS96], and more sophisticated statistical techniques such as selection based on *correlation coefficients* [NGL97] or information gain [JOA98]. Recent work by Yang [YAP97] found that some of these methods could reduce the feature space by up to 98% while at the same time slightly improving performance. In the current study, the focus is on the improvements that can be obtained by a change in representation. Feature selection is sometimes a part of this process, but it is not the part that is of primary interest. Therefore feature selection is performed only in cases where the number of features is too high for reasonably fast training using RIPPER.

5.2 The FX Integrated Feature Extraction System

A reliable toolkit was required to help engineer the various types of features used in the current study. The FX (Feature eXtractor) system was designed to read the SGML formats for both corpora, separate words from punctuation and white space, and produce either traditional feature vectors with nominal or continuous values, or the set-valued feature vectors for use with RIPPER. FX was developed purely as a research tool to meet the needs of this particular work, so the functionality of FX was expanded only as needed. The functions of FX that were necessary to perform the experiments in this chapter are listed below:

i) read the SGML tagging conventions;
ii) select the appropriate examples to form the training and testing sets;
iii) isolate the words in each document, filtering non-alphabetic characters and suppressing case information;
iv) stem words if required using the Lovins stemmer[8];
v) assemble files containing the training examples laid out as binary classification problems – each file containing a single classification problem.

Other functionality will be introduced in later chapters as it is needed. For more complete information on how to obtain and use FX, see Appendix A.

[8] Written by Linh Huynh (linh@kbs.citri.edu.au) in ANSII standard C and available as freeware.

5.3 Experiments

Based on the above discussion, experiments were run on the Reuters-21578 and DigiTrad data using the ModApte and DT100 splits described in chapter 2. Binary classifiers were trained for each class represented by 10 or more positive training examples. For classes with fewer than 10 training examples, a *default classifier* was used to always return a negative classification. This left 60 classes for training in Reuters-21578 and all 33 classes in DigiTrad. Classifiers were trained using both a plain bag of words representation (BW) and the stemmed version of the same representation (BSW). No feature selection strategies were used – all words appearing in the training set were included as features. For Reuters-21578, this resulted in 27940 features in the BW representation and 18590 in BSW. DigiTrad had 43301 features in the BW representation and 28996 in BSW.

Default options were used for RIPPER with two exceptions. The first was the enabling of so-called *negative tests* in RIPPER, which allows the algorithm to use both the $\in$ and $\notin$ operation on set valued features. This option was found to marginally improve performance in initial trials. Secondly, the *loss ratio* parameter (*L*) was varied to try to bracket the breakeven points on each task. The values {0.5, 1, 2, 4} for Reuters-21578 and {0.25, 0.5, 1, 2} for DigiTrad were used because they were found to bracket the micro-averaged breakeven points in most cases.

5.4 Results and discussion

The micro- and macro-averaged precision and recall for each value of *L* is shown for Reuters and DigiTrad in Table 6 and Table 7 respectively. "Macro$_0$" denotes macro-averaging in which 0/0 was defined as equal to 0, and "macro$_1$" means 0/0 = 1. These values represent averages across all classes, including those for which default classifiers were assigned. The most striking observation at first glance is that RIPPER was able to produce recall and precision for Reuters-21578 that was more than twice as high as for DigiTrad. This seems to confirm the observations from chapter 2 about the special difficulties posed by the latter domain. Comparing the two representations on both data sets shows very little difference between BW and BSW, although there was a slight *decrease* in performance using stemming on Reuters-21578, and an even slighter *increase* using stemming on DigiTrad. On Reuters-21578, there is also a noticeable difference between the macro$_0$ and macro$_1$ averages, due largely to the effect of the default classifiers.

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	**micro**	**.783**	**.865**	**.816**	**.834**	**.856**	**.746**	**.900**	**.627**
	$macro_0$	.435	.508	.435	.469	.432	.385	.444	.295
	$macro_1$	.776	.519	.819	.480	.839	.396	.872	.306
BSW	**micro**	**.774**	**.841**	**.812**	**.811**	**.858**	**.740**	**.894**	**.651**
	$macro_0$	.412	.463	.417	.433	.442	.377	.456	.302
	$macro_1$	.753	.474	.791	.444	.837	.388	.884	.313

***Table 6*: Micro- and Macro-averaged precision and recall obtained on the testing data of Reuters-21578 using the BW and BSW representations.**

		L=0.25		L=0.5		L=1.0		L=2.0	
		P	R	P	R	P	R	P	R
BW	**micro**	**.318**	**.394**	**.375**	**.344**	**.480**	**.221**	**.580**	**.098**
	$macro_0$	.273	.353	.318	.313	.387	.202	.371	.087
	$macro_1$	.273	.353	.318	.313	.418	.202	.674	.087
BSW	**micro**	**.309**	**.410**	**.373**	**.348**	**.484**	**.210**	**.573**	**.101**
	$macro_0$	.273	.365	.328	.316	.410	.204	.416	.102
	$macro_1$	.273	.365	.328	.316	.501	.204	.628	.102

***Table 7*: Micro- and Macro-averaged precision and recall obtained on the testing data of DigiTrad using the BW and BSW representations.**

The raw results from the previous tables are summarized in Table 8 and Table 9 using breakeven points and F-measures. F-measures are reported for the three standard values of β=0.5, 1.0 and 2.0. (*β=0.5* signifies recall is half as important as precision, *β=2.0* signifies recall is twice as important as precision, and *β=1.0* signifies both are equally important.) The numbers reported in each column are the maximum F-measure values obtained over the four runs. Figure 12 and Figure 13 show the breakeven points graphically. Note that the $macro_1$-averaged breakeven points for Reuters-21578 were obtained by extrapolation rather than interpolation. Extrapolated breakeven points will always be identified with a superscript "e" and should be viewed with caution

It is interesting to note that the $F_{1.0}$ statistic roughly agrees with the breakeven point in all cases. In fact, a standard test found 99.8% correlation between the two statistics, suggesting that either measurement is as good as the other for comparing representations. The F-measures for *β=2.0* and *0.5* again demonstrate slight differences in the effect of stemming on the two data sets. For Reuters, the micro-averaged statistics imply that stemming slightly improves precision. For DigiTrad, the same statistics imply that stemming slightly improves recall. Overall performance on BSW is worse for Reuters-21578 and better for DigiTrad. In both cases these differences are more exaggerated on the macro-averaged statistics, suggesting that the less represented classes were more affected. But again, all these effects are very slight and we should be cautious about drawing conclusions.

		Maximum F-measures			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.828**	**.825**	**.847**	**.821**
	macro_0	.448	.469	.492	.434
	macro_1	.718	.622	.556	$.641^e$
BSW	**micro**	**.832**	**.811**	**.826**	**.810**
	macro_0	.421	.436	.452	.422
	macro_1	.684	.582	.512	$.597^e$

***Table 8*: Maximum F-measures and breakeven points (BP) obtained on the testing data of Reuters-21578 using the BW and BSW representations. A Superscript "e" indicates an extrapolated point.**

		Maximum F-measures			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.389**	**.359**	**.433**	**.359**
	macro_0	.327	.315	.131	.315
	macro_1	.344	.315	.131	.315
BSW	**micro**	**.384**	**.360**	**.435**	**.360**
	macro_0	.341	.322	.136	.322
	macro_1	.388	.322	.136	.322

***Table 9*: Maximum F-measures and breakeven points (BP) obtained on the testing data of DigiTrad using the BW and BSW representations. A Superscript "e" indicates an extrapolated point.**

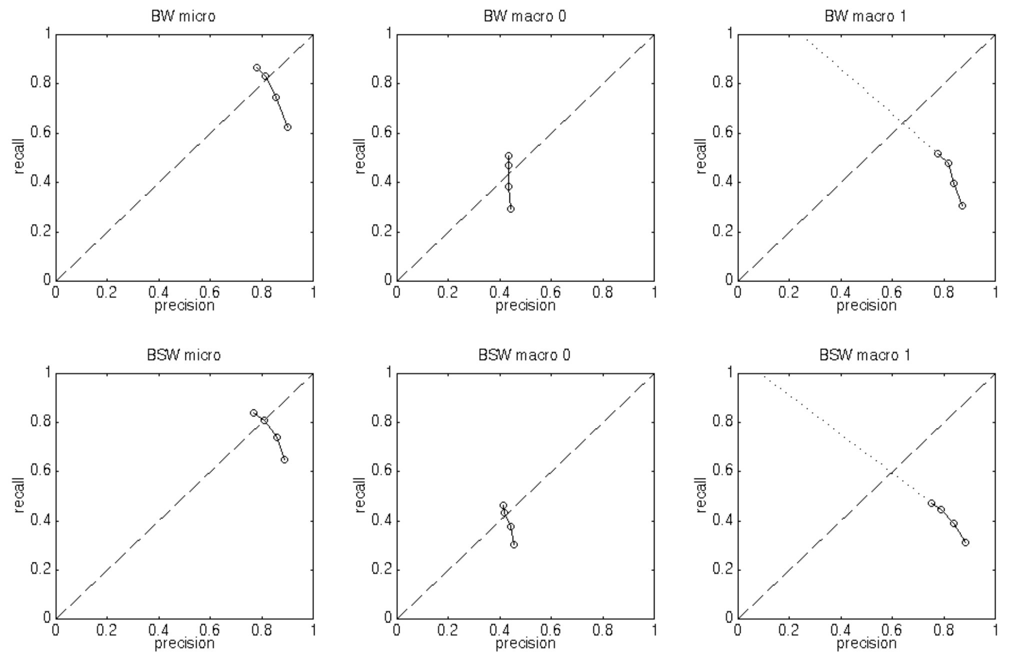

***Figure 12*: A graphical view of the Reuters-21578 breakeven points. The first row shows BW using micro, macro$_0$ and macro$_1$ averaging respectively, and the second row shows the same graphs for BSW. Note that the macro$_1$ values were extrapolated rather than interpolated.**

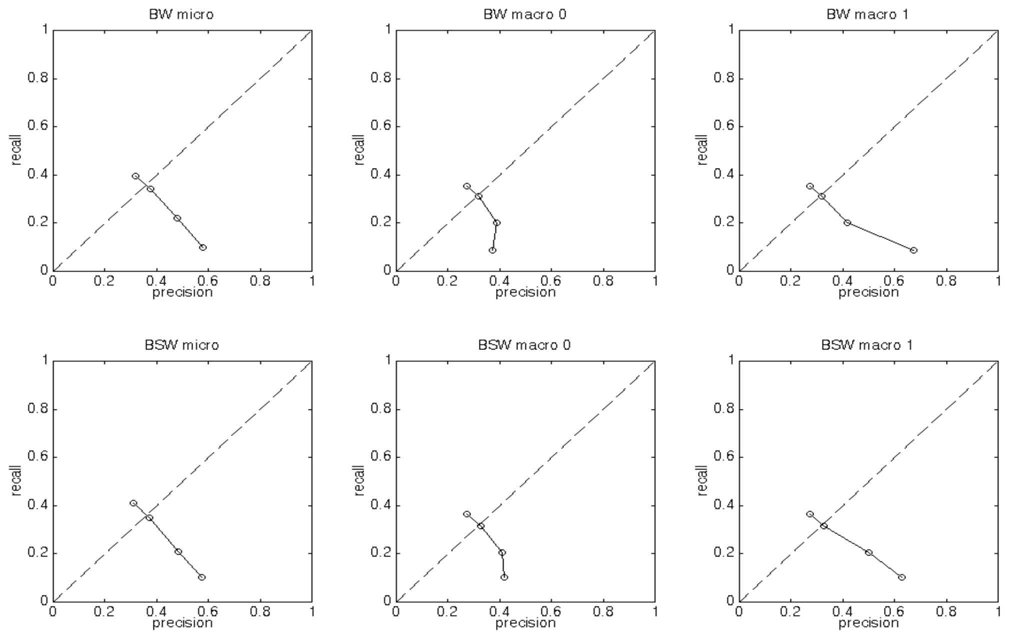

***Figure 13*: A graphical view of the DigiTrad breakeven points. The first row shows BW using micro, macro$_0$ and macro$_1$ averaging respectively, and the second row shows the same graphs for BSW.**

To try to find out if stemming had different effects in the presence of more or less training data, the results were broken down by grouping classes of a similar size together. Table 10 and Table 11 show the averaged results of each run broken down by number of training examples. Reuters-21578 was broken into four groups representing classes with 100 or more, 50 to 99, 25 to 49, and 10 to 24 positive training examples. (Classes with 0 to 9 positive training examples were classified with default classifiers.) DigiTrad was split into three groups representing 200 or more, 100 to 199, and 50 to 99 positive training examples. Table 12 and Table 13 show the maximum F-measures and breakeven points computed for each group. Not surprisingly, RIPPER's performance is usually better on both corpora when it has access to more positive training examples, and once again the effects of stemming seem to be more exaggerated among the less represented classes. Stemming seems to decrease performance slightly across all groups of Reuters classes, but for the DigiTrad classes, stemming did worse on the first group and better on the groups with fewer positive training examples.

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	100+	.811	.905	.839	.885	.878	.805	.919	.693
	50+	.647	.778	.657	.714	.717	.609	.797	.353
	25+	.714	.835	.765	.775	.798	.798	.800	.491
	10+	.616	.682	.717	.553	.689	.689	.698	.335
BSW	100+	.813	.890	.839	.870	.882	.808	.914	.721
	50+	.576	.680	.618	.662	.662	.492	.712	.372
	25+	.697	.812	.745	.752	.829	.647	.852	.500
	10+	.626	.696	.741	.556	.739	.506	.800	.451

***Table 10*: Micro-averaged precision and recall for Reuters-21578 classes grouped by amount of training data using the BW and BSW representations.**

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	200+	.381	.460	.433	.407	.518	.264	.602	.123
	100+	.251	.316	.327	.267	.449	.161	.597	.056
	50+	.239	.311	.273	.272	.394	.182	.489	.086
BSW	200+	.353	.476	.432	.401	.513	.244	.589	.111
	100+	.278	.346	.341	.280	.459	.146	.544	.080
	50+	.223	.309	.263	.291	.421	.205	.556	.102

***Table 11*: Micro-averaged precision and recall for DigiTrad classes grouped by amount of training data using the BW and BSW representations.**

		Maximum F-measures			**BP**
		β=0.5	β=1.0	β=2.0	
BW	100+	.863	.861	.885	**.854**
	50+	.692	.706	.748	**.678**
	25+	.767	.770	.807	**.767**
	10+	.677	.647	.667	**.645**
BSW	100+	.867	.853	.874	**.851**
	50+	.626	.639	.657	**.627**
	25+	.785	.750	.786	**.748**
	10+	.618	.571	.577	**.572**

***Table 12*: Maximum F-measures and breakeven points (BP) obtained for Reuters-21578 classes grouped by amount of training data using the BW and BSW representations.**

		Maximum F-measures			**BP**
		β=0.5	β=1.0	β=2.0	
BW	200+	.435	.419	.442	**.420**
	100+	.330	.294	.301	**.291**
	50+	.320	.272	.293	**.273**
BSW	200+	.425	.416	.445	**.416**
	100+	.326	.308	.330	**.311**
	50+	.348	.276	.287	**.281**

***Table 13*: Maximum F-measures and breakeven points (BP) obtained for Reuters-21578 classes grouped by amount of training data using the BW and BSW representations.**

To complete the quantitative analysis of the bag of words approach with RIPPER, the results are compared to those of previous studies. The DigiTrad domain is new and thus has no points of comparison, but Reuters-21578 and its predecessor Reuters-22173 have a long history of use in the text classification literature. Often direct comparison is hampered by the different ways in which researchers use the data, but the ModApte training split has been used recently by Joachims [JOA98] on Reuters-21578, and by Cohen and Singer [COS96] and Ng et al. [NGL97] on Reuters-22173. The latter work is not directly comparable, but the results are shown in Table 14 anyway for the sake of completeness. The work with RIPPER followed a very similar methodology to this study, using a bag of unstemmed words with punctuation and case information removed. The only difference is that stop words were removed before learning.

Learner	Options	Micro-averaged BP
Classi (Perceptron)	200 features	0.802
RIPPER	negative tests	0.796
RIPPER	no negative tests	0.793
SWAP-1	80-100 freq. features	0.789
SWAP-1	80-100 Boolean features	0.785
Experts	3-words	0.759
Experts	2-words	0.753
Rocchio		0.745
Experts	1-word	0.647

***Table 14*: Previous work using the ModApte split with Reuters-22173. Table adapted from [NGL97] and [COS96]. Classi is the Perceptron learner used in [NGL97], SWAP-1 is the decision tree learner used in [ADW94], and Experts is the "sleeping experts" system used in [COS96].**

Joachims [JOA98] has published some very complete data for the Reuters-21578 version of the ModApte split, in which he compares Naive Bayes, Rocchio, C4.5, k-Nearest Neighbor, and Support Vector Machines. These results are presented in the same style as Joachims in Table 15 below, with the RIPPER results added for both the BW and BSW representations. The representation used by Joachims was stemmed word frequency (using the Porter stemmer [POR80]) with stop words and words occurring in fewer than 3 documents removed. All 90 classes were used in training. For algorithms with no parameter comparable to RIPPER's loss ratio (Bayes and C4.5), the breakeven points were obtained by varying the number of features selected based on an information gain heuristic.

Using both representations, RIPPER comes in third place, well behind Joachims' Combined Support Vector Machines [HEA98], slightly ahead of Rocchio and C4.5, and very slightly behind k-Nearest Neighbor. These results show that even with the basic bag of words representation, RIPPER is competitive with other learning systems, beating even its close relative C4.5 overall and on 8 of the 10 most frequent classes. As noted previously, RIPPER is also one of the faster learners - Support Vector Machines are comparable in time complexity to C4.5 [JOA98].

	Bayes	**Rocchio**	**C4.5**	**k-NN**	**Combined SVM**[9]		**RIPPER**	
					poly	RBF	BW	BSW
earn	.959	.961	.961	.973	.985	.985	.953	.959
acq	.915	.921	.853	.920	.952	.954	.893	.861
moneyfx	.629	.676	.694	.782	.762	.763	.709	.643
grain	.725	.795	.891	.822	.924	.931	.911	.906
crude	.810	.815	.755	.857	.889	.889	.767	.802
trade	.500	.774	.592	.774	.771	.778	.701	.644
interest	.580	.725	.491	.740	.762	.762	.622[e]	.620
ship	.787	.831	.809	.792	.865	.854	.745	.777
wheat	.606	.794	.855	.766	.859	.852	.865[e]	.848[e]
corn	.473	.622	.877	.779	.857	.851	.908	.911
micro	**.720**	**.799**	**.794**	**.823**	**.860**	**.864**	**.821**	**.810**

***Table 15*: Previous work using the ModApte split on Reuters-21578. All results are from [JOA98] except the RIPPER results which are taken from this study. The first 10 rows show the breakeven points for the 10 most frequent Reuters classes, and the last row shows the micro-averaged breakeven point for all classes. SVM refers to Support Vector Machines (polynomial and radial basis function versions) as described in [JOA98]. Table adapted from [JOA98].**

Finally, the hypotheses learned by RIPPER were subjected to the analysis of complexity described in section 4.4. Table 16 shows the average number of rules per hypothesis, and the average number of clauses per rule for the two representations using a loss ratio of 1.0 in the case of Reuters and 0.5 in the case of DigiTrad (the loss ratios that returned the maximum micro-averaged $F_{1.0}$ statistic.) The figures shown are the average numbers of rules and clauses per hypothesis. These are combined to estimate the average number of clauses per rule. BSW formed slightly more complex hypotheses on both corpora (around 0.5 more rules per hypothesis), possibly because the stemmed features tended to over-generalize and cover too many negative examples.

		L.R.	$F_{1.0}$	Rules	Clauses	Clauses / Rule
Reuters-21578	BW	1.0	.825	5.1	13.0	2.5
	BSW	1.0	.811	5.6	13.5	2.4
DigiTrad	BW	0.5	.359	12.9	30.3	2.3
	BSW	0.5	.360	13.3	33.0	2.5

***Table 16*: Complexity statistics for each representation. The column *L.R.* shows the loss ratio setting for the rule set, and $F_{1.0}$ shows the micro-averaged $F_{1.0}$-measure achieved with the rules. *Rules* indicates the average number of rules per hypothesis. *Clauses* indicates the average number of clauses per hypothesis, and *Clauses/Rule* indicates the average number of clauses in each rule.**

[9] The values shown in the first 10 rows for SVM are the *best* values obtained by using a range of parameters and then selecting the parameter for each class which produces the classifier with the lowest VC Dimension. For complete details, see the original table in [Joachims 1998].

5.5 Summary

The experiments in this chapter form the baseline for comparison with the next two chapters. The impression that DigiTrad would be more difficult to classify than Reuters-21578 was confirmed. RIPPER using bag of words seems to compare quite favorably to other learning algorithms on the Reuters-21578 database, beating all other algorithms except Combined Support Vector Machines (which wins by a wide margin) and k-Nearest Neighbor (which wins by a slim margin). We experimented with stemmed and unstemmed versions of the bag of words representation, and found that stemming slightly hurt performance on the Reuters-21578 task, and slightly improved performance on DigiTrad. In both cases these effects were more pronounced on classes with fewer positive training examples. The maximum F-measure for $\beta=1.0$ seemed to be as good a statistic as the breakeven point for comparing representations. Some statistics were also gathered on hypothesis complexity in the two representations that will be used for comparison later. With the baseline experiments now complete, we can move on to experiments with other text representations.

6. Phrase-Based Representations

A very basic observation about using bag of words representations for text classification is that a great deal of the information from the original document is discarded. Paragraph, sentence and word order is disrupted, and syntactic structures are broken. The end result is that the text is rendered incoherent to humans in order to make it coherent to a machine learning algorithm. The aim of this section is to explore feature engineering techniques that preserve parts of the structure from the original document while still remaining compatible with a representation based on feature-vectors. The hope is that preserving some information about word ordering will provide RIPPER with some more informative features to increase its classification power.

Words alone do not always represent true atomic units of meaning. Units of meaning are often represented by *phrases* (defined as words occurring sequentially in the document) rather than individual words. For example, the phrase "machine learning" has a highly specific meaning that is separate and distinct from the words "machine" and "learning". This fact may be part of the difficulty with learning topic labels based on words alone. If so, then perhaps some automatic means of choosing phrases as features could help address this problem. In a phrase-based representation, a document would be represented as a "bag of phrases" rather than a bag of words.

The main problem to deal with in converting to a bag of phrases is the huge potential increase in the number of features. If all two-word phrases in a document are used as features, we will have the same number of phrase instances as word instances, but the number of distinct features will potentially grow from n to n^2 (where n is the number of distinct words.) If longer phrases are used, the dimensionality will get geometrically worse. In order to keep the problem tractable, some solution to this problem must be found. One such solution, attempted by William Cohen [COH95a], is to use the methods of *Inductive Logic Programming* (ILP) to define an implicit phrase-based feature space. A document is first converted to predicates of the form $w_i(d,p)$ which indicate that word w_i appears in document d at position p. These basic atoms are then combined with a background knowledge embodied in the following predicates:

- *near1(p1,p2)* is true when $|p_1-p_2| \leq 1$,
- *near2(p1,p2)* is true when $|p_1-p_2| \leq 2$,
- *near3(p1,p2)* is true when $|p_1-p_2| \leq 3$, and
- *after(p1,p2)* is true when $p_1 \leq p_2$.[10]

In order to make use of this representation, Cohen expanded RIPPER into a first-order logic version which retained the same search heuristics as the original. The new algorithm, FLIPPER was compared to the first-order logic learner FOIL using the same representation, as well as to the propositional algorithms of RIPPER and C4.5 using a plain bag of words. Cohen found little evidence to suggest that the ILP phrase-based

[10] Actually this representation defines a superset of phrases that includes words separated by one or two other words, but this does not alter the general line of argument presented here.

method was an improvement, except in the case where precision was more important than recall.

At first glance the above result seems to show that phrase-based representations will not be useful, but there is a problem with the approach. To see why, consider the example of a three class learning task in which the classes are EP (educational psychology), MT (machine tools), and AI (artificial intelligence). For the AI classifier, the phrase "machine learning" will probably come in quite handy. For the sake of argument let's assume that this phrase occurs in 50% of the AI documents and none of the EP or MT documents. Let's also assume that this makes it the phrase with the highest information gain for the AI class. So the learner needs to form the rule:

$$machine(d,p_1)\ \&\ learning(d,p_2)\ \&\ near1(p_1,p_2)\ \&\ after(p_1,p_2) \Rightarrow AI.$$

This turns out to be quite a tall order for a system like FLIPPER which grows rules one clause at a time in a greedy fashion. Starting with the blank rule, obviously *near1* and *after* will not produce any information gain, so either *machine* or *learning* needs to be added. But *machine* probably appears in almost all the MT documents, and *learning* will be used heavily in the EP documents. Therefore, it is unlikely that either word on its own will produce a high enough information gain to be selected, and the optimal rule will never be learned. For the same reason, neither FLIPPER nor RIPPER will even be capable of learning the lower quality rules:

$\text{machine}(d,p_1)\ \&\ learning(d,p_2) \Rightarrow AI$ (for FLIPPER), or
$(machine \in \text{WORDS})\ \&\ (learning \in \text{WORDS}) \Rightarrow AI$ (for RIPPER).

Suppose on the other hand, the feature space had been constructed to contain all the words plus all two-word phrases. In the new representation, many documents in the AI class would contain non-zero values for the features *machine*, *learning*, and *"machine learning"*. Now RIPPER should have no trouble applying its search heuristic to find the optimal rule:

$$(\text{"}machine\ learning\text{"} \in DOCUMENT) \Rightarrow AI.$$

Unfortunately however, we have come back to the problem of tractability. If documents are represented as all words plus all phrases up to a maximum length *m*, there will be $O(n^m)$ features in the representation. However, not all phrases are equally meaningful. For example the sentence *"The quick brown fox jumps over the lazy dogs,"* contains a number of potentially useful phrases such as "*brown fox*", "*quick brown fox*", and "*lazy dogs*" in addition to a larger number of phrases which are probably not useful for classification: "*The quick*", "*fox jumps*", "*over the*", and so on. So the new phrase-based representation should be amenable to an aggressive selection strategy aimed at weeding out the words and phrases which are least likely to be useful.

The next sections outline two possible strategies for selecting phrase-based representations of a reasonable size. The first focuses on a syntactic approach to extract *noun phrases* from texts, and the second looks at a statistical approach to extract *keyphrases* from a text. Both these methods will be evaluated in the sections that follow.

6.1 A Syntactic Approach: Noun Phrases

Lewis undertook a major study of the use of noun phrases for statistical classification as part of his Ph.D. thesis [LEW92b]. He reviewed the existing information retrieval literature on the use of syntactic phrases and found that most studies had not been able to demonstrate much improvement over word-based indexing. To try to explain these results, Lewis suggested that previous work fared poorly because of the following four factors:

i) **High Number of Terms**: Depending on the sophistication of the noun phrase grammar, there tend to be many more unique noun phrases than unique words in a document.

ii) **Uneven Distribution of Values**: Each noun phrase appears in very few documents.

iii) **High Redundancy of Terms**: The effect of synonymous words makes multi-word phrases highly redundant.

iv) **High Noise**: This redundancy also implies “noise” in the sense that all relevant phrases will not occur in every appropriate document.

All these characteristics pose theoretical problems for Bayesian classification (though it is important to note that they may not pose a problem for RIPPER.) Lewis attempted to correct these problems by using a statistical clustering algorithm to group phrase features into so-called *meta-features*. Unfortunately, the results of this phrasal clustering did not produce any improvement on the Reuters-22173 corpus [LEW92a, LEW92b] leading to somewhat pessimistic conclusions about the use of linguistic information in text classification until significant advances are made in natural language processing techniques [LES96].

Despite Lewis’ results, there is still room to work with noun phrases in a classification framework. Lewis, like most researchers prior to 1995, was using a statistical learning system based on Bayesian classification. There is no guarantee that results obtained in this context will apply to a decision rule learner such as RIPPER. As noted in the previous section, there are good reasons to hold out the hope that a carefully constructed phrasal representation could improve RIPPER’s performance. Furthermore, Lewis’ particular methodology differed quite radically from the general strategy outlined in the previous section. Lewis used a *bracketing* algorithm biased towards extracting the longest possible noun phrases. Single word phrases were excluded, resulting in a huge set of very sparsely distributed features which were then combined with statistical clustering.

The methodology used here is more closely based on Cohen's ILP approach which considered all phrases containing three or fewer words. Starting with all phrases of any length (including single word phrases), the approach is to build a Noun Phrase Extraction system to select noun phrases as potentially relevant features. For instance, if the system encounters the sentence *"The quick brown fox jumps over the lazy dogs,"* it should recognize that *quick brown fox* is a noun phrase while *The quick* and *over the* are not. Importantly, it should also recognize that *brown fox* and *fox* are valid noun phrases as well.

6.1.1 Extracting Noun Phrases with NoPE

Extracting noun phrases from a document requires two separate algorithms. The first is a tagging algorithm to assign part of speech tags (*noun*, *verb*, *preposition*, etc.) to the individual words, and the second is an algorithm to group the tagged words into noun phrases. Eric Brill's rule-based part of speech tagger [BRI92, BRI94] was used to assign the part of speech tags, and grouping was done with a simple regular expression based partly on the discussion in the previous section. The latter algorithm is implemented within the FX system. For simplicity, the combination of Brill's tagger with the new functionality in FX is referred to as the Noun Phrase Extractor system, abbreviated "NoPE".

Prior to 1992, most tagging algorithms were *stochastic* in nature. Given a tagged corpus of text, these algorithms tried to determine the most likely tag for a given word using probabilities estimated from a manually tagged corpus. Eric Brill developed a transformation-based part of speech tagger, based on a supervised machine learning model, that was comparable in accuracy to stochastic taggers [BRI92, BRI94]. The algorithm proceeds in two passes. On the first pass, the words are assigned their most likely tag, based on their frequency of appearance with that tag in a manually tagged training corpus. On the second pass, tags are transformed based on the application of a set of rules learned from the same corpus. These rules can change tags based on the context of surrounding words and tags in the sentence. For instance, one rule states that a tag should be changed "from *preposition* to *adverb* if the word two positions to the right is the word "*as*" [BRI94]. Rules such as these are also used to assign tags to words that did not appear in the original training corpus. Brill's tagger is freely available[11] in three different forms based on different training corpora: the Penn Treebank Wall Street Journal Corpus described in [MSM93], the Brown corpus [KUF67] or a corpus formed by combining both. NoPE uses the default configuration which uses the third option.

The texts in this study all required some preprocessing before they were given to the tagger. The tagger assumes that all punctuation is separated from the words by white space, and that each line contains a single sentence terminated with a period. The first requirement was easy, but the second posed some challenges. The first problem was how to decide when a period indicates the end of a sentence, and is not part of an abbreviation, decimal number, web site address, or some other object. In the context of Reuters-21578 it was assumed that a period always indicates the end of a sentence unless the next

[11] http://www.cs.jhu.edu/~brill

character is *not* white space. This still left some errors in delimiting sentences, and probably lead to more tagging errors, but this situation seemed unavoidable without some extremely sophisticated method of text analysis. The preprocessing steps for the Reuters-21578 documents can be summarized as follows:

i) separate punctuation from words,
ii) remove all carriage returns within the body of the texts, and
iii) insert carriage returns after every '.' character that was followed by a white space character in the original text.

An example of a preprocessed Reuters-21578 news story is shown in Figure 14.

Unfortunately, steps *ii* and *iii* of the pre-processing did not work well for the DigiTrad data. The basic problem is that the songs are laid out in the form of stanzas rather than sentences. In some cases, the punctuation is very close to regular prose, and the songs could actually be interpreted as collections of sentences terminated in the usual way. In other cases, there is no punctuation at all, or each line is terminated with a comma even though it might have been intended to be a sentence. Some examples of different punctuation styles are shown in Figure 4 of chapter 2. Because of the difficulty (and in some cases impossibility) of deciding where sentences began and ended, steps *ii* and *iii* of the preprocessing were skipped for the DigiTrad documents. Punctuation was separated from the words, but otherwise the documents were left in their original state. Whether this preprocessing decision produced more or less errors in tagging than the alternative is not known.

```
<TOPICS><D>gold</D></TOPICS>
<BODY>
 Thousands of black mineworkers roared support for a union proposal to seize control of South Africa '   s
gold ,  uranium ,  platinum and coal mines if the owners refuse to improve conditions for migrant black
workers .
    About  1  5  ,  0  0  0  miners attended a rally here to endorse moves proposed by last week '   s annual
meeting of the  2  0  0  ,  0  0  0  strong National Union of Mineworkers  ( NUM )  .
    They also supported a proposal for a national strike at the end of this month if the owners refused to
begin negotiations .
    Migrant workers from surrounding countries make up more than half of the labour force in the mines .
    It was not stated how the union would seize control .
    The miners '    leaders also demanded an end to the system of single sex hostels for migrant workers ,
to be replaced by housing schemes so that workers could live with their families .
    The crowd ,  one of the largest to attend a meeting since South Africa declared a state of emergency
last June ,  also shouted approval of a proposal to work closely with anti - apartheid movements such as
the United Democratic Front  ( UDF )  which claims two mln members .
 They also shouted their support for a demand that jailed black nationalist leader Nelson Mandela be
released .
  REUTER  &  #  3  ;
</BODY>
```

***Figure 14*: An example of a Reuters-21578 news story after preprocessing for the Brill tagger.**

The Brill tagger reproduces the original text but with part of speech tags attached to the ends of words as shown in Figure 15. The first character in each tag identifies the basic type of the word, such as noun (N), verb (V), or adjective (J). The second and third characters of the tag identify more detailed information such as plural versus singular, or proper versus common nouns. This latter information is not relevant to the current experiments so the tags will be treated as if they only contained a single character.

For the purposes of extraction, a *noun phrase* is defined to be a sequence of nouns or adjectives terminating in a noun. In regular expression form this is represented as:

Noun Phrase = [J,N]*N.

NoPE applies this definition at every possible point in a text to all phrases, even if they are subsets of another phrase. For example, the phrase *single/J sex/N hostels/N* from Figure 15 would give rise to the phrases *single sex hostels*, *single sex*, *sex*, *sex hostels* and *hostels*. This definition excludes more complex noun phrases such as *the system of single sex hostels*, identifies some phrases such as *sex hostels*, which are not intended to be noun phrases, and misses some other relatively simple noun phrases such as *miners' leaders* (because of the punctuation character). But despite its shortcomings, NoPE does accomplish its stated task of reducing the number of features in the bag of phrases by selecting many of those that represent the objects or subjects of discussion.

The/DT miners/NNS '/POS leaders/NNS also/RB demanded/VBD an/DT end/NN to/TO the/DT system/NN of/IN single/JJ sex/NN hostels/NNS for/IN migrant/JJ workers/NNS ,/, to/TO be/VB replaced/VBN by/IN housing/NN schemes/NNS so/RB that/IN workers/NNS could/MD live/VB with/IN their/PRP$ families/NNS ./.

***Figure 15*: A sentence tagged by the Brill tagger. The *tags* are the sequences of characters after the '/' character in each word. The original sentence is from Figure 14.**

6.1.2 The Change of Representation

NoPE was used to define a "bag of noun phrases" model using a multi-pass feature extraction method as follows:

i) First the documents are preprocessed to be usable by the Brill tagger.

ii) Next the tagger is used to assign parts of speech (which is itself a two pass process).

iii) Then during the first pass of feature extraction, the regular expression described in section 6.1.1 is used to assemble a list of all noun phrases in the corpus. As the list is assembled, the syntactic tags are stripped from the phrases to correct for any mistakes on the part of the tagger (e.g. we want the phrase *single/J sex/N hostels/N* to also match the phrase *single/N sex/N hostels/N* even though the tags don't match.)

iv) This global list then becomes the set of features in the document and the second pass determines a value for each feature in each document. For the purposes of feature extraction, a phrase is said to "occur" in a document each time the words in the phrase are found in the proper order separated only by white space.

Not surprisingly, NoPE tended to find a very large number of features – 177438 for Reuters-21578 and 115730 for DigiTrad. However, these numbers were greatly reduced by discarding phrases appearing fewer than 4 times in the training corpus. After this filtering step, 21427 phrases were left in Reuters-21578 and 12457 were left in DigiTrad.

6.2 A Statistical Approach: Keyphrases

The previous section described how to use the NoPE system for noun phrase extraction to construct a new "bag of noun phrases" representation. The basic goal of the method was to use syntactic information to select a useful subset of the all the available phrases in the text. This section describes a statistical method of achieving the same goal. Rather than focussing on a part of speech, the Extractor system developed by Peter Turney [TUR98] uses a statistical algorithm to try to extract the most *meaningful* phrases from a document.

The idea of a keyword (or "keyphrase") list for a document should be a familiar one. Many academic journals ask their authors to provide a list of such terms in addition to an abstract. The authors select the terms (usually fewer than 10) that they feel can best indicate the topical ground covered in their document. These keyphrases should be highly meaningful, unambiguous, and indicative of the content of the document – the very properties desired in the choice of phrase features for classification. Extractor has been trained to try and mimic the choices of keyphrases that a human would make.[12]

6.2.1 Extracting Keyphrases

The algorithm used by Extractor is described fully elsewhere [TUR98], however it is worth summarizing its operation quickly here. The learned component of Extractor is actually a set of parameters that govern its behavior within a general statistical framework. As with the Brill tagger, the user of Extractor does not have to perform any training – Extractor has already been trained by the author of the software. However both components of the system (the general framework of Extractor and the training process) are important and both will be discussed below.

Extractor considers all possible phrases of three or fewer words and assigns scores to them based on twelve parameters that are set during learning. The base score of any phrase is the number of occurrences of the stemmed, case-suppressed version of the phrase in the document. This score is subject to a number of multipliers based on the number of stems in the phrase (with multi-word phrases being scored higher) and on the relative position of the first occurrence of the phrase in the document (with a reward given for early occurrence and a penalty given for late occurrence.) Phrases containing stop words or punctuation marks are automatically ruled out. After scoring each stemmed

[12] See http://ai.iit.nrc.ca/II_public/extractor for a demonstration copy of the Extractor software and supporting documentation.

phrase, the case information and suffixes are restored and the phrases with the highest scores are ranked and considered for output. From this final list the following may be dropped: phrases that contain many fewer characters than average (unless they obtained a very high rank or appear to be abbreviations); proper nouns (depending on the value of a flag parameter); phrases that appear to be adjectives or contain words commonly used as verbs; and phrases that appear in a stop list of phrases. The final number of output phrases is also determined by a parameter. Repetitiveness is prevented by ensuring that no two phrases from a single document contain the same word stem.

The list of parameters governing Extractor's behavior is shown in Table 17 along with a brief explanation of the exact role each plays in the process described above. In the released version of Extractor, the twelve parameters are set to values determined using a commercially-available genetic algorithm named Genitor [WHI89]. A genetic algorithm is a learning system for numerical optimization that works with populations of individuals, usually represented as binary vectors. Initially the vectors are set randomly. On each cycle (or generation) individuals are scored using a fitness function and the highest scoring individuals are selected using a probabilistic algorithm to form the basis of the next generation of the population. In forming the next generation, a random mutation operator causes small random changes by flipping bits, and a crossover operation swaps sequences of bits between paired individuals. Once the next generation has been populated with the same number of individuals as the previous generation, the algorithm continues by applying the fitness function and repeating selection, mutation, and crossover again. Within this paradigm there is room for variation, but the basic principle is common to all of them - reproducing populations of individuals are subject to fitness-based selection, random mutation and crossover. The book by Melanie Mitchell [MIM96] is an excellent introduction to the field.

Genitor is a *steady-state* genetic algorithm rather than the more common *generational* algorithm described above. The main difference is that Genitor updates its population one individual at a time by generating a single new individual using selection, mutation, and crossover, scoring it with the fitness function, and then using this individual to replace the least fit individual in the original population. This variation apparently produces a slightly higher selection pressure on the population than the original algorithm, causing it to converge more quickly to a steady state [WHI89].

The fitness function for the Extractor parameters was defined by treating keyphrase extraction as a retrieval task on a fixed corpus of documents with keyphrases already assigned. Thus Extractor was trained to match human choices of keyphrases as closely as possible. Each document was viewed as containing a finite total number of phrases from which Extractor could choose. The decision of whether or not to include a particular keyphrase was treated as a binary classification decision. Seen in this way, performance could be measured in terms of precision and recall, and the fitness function defined using the F-measure.

Parameter Number	**Parameter Name**	**Description**
1	NUM_PHRASES	length of final phrase list
2	NUM_WORKING	length of a working list of single stems[13]
3	FACTOR_TWO_ONE	multiplier for two-word phrases
4	FACTOR_THREE_ONE	multiplier for three-word phrases
5	MIN_LENGTH_LOW_RANK	low ranking phrases in the final list must be longer than this value (expressed as a fraction of the average phrase length in the document)
6	MIN_RANK_LOW_LENGTH	short phrases may be kept if they have at least this rank in the final list
7	FIRST_LOW_THRESH	threshold for defining an "early" first occurrence of a phrase
8	FIRST_HIGH_THRESH	threshold for defining a "late" first occurrence of a phrase
9	FIRST_LOW_FACTOR	multiplier for "early" first occurrence of a phrase
10	FIRST_HIGH_FACTOR	multiplier for "late" first occurrence of a phrase
11	STEM_LENGTH	maximum number of characters in a stem[14]
12	SUPPRESS_PROPER	flag for suppressing proper nouns

***Table 17*: The twelve parameters used by Extractor. Adapted from [TUR98].**

Genitor was used to learn two sets of parameters based on this fitness function – one for short documents (less than 20 kilobytes long) and another for long documents. The final version of Extractor chooses the appropriate settings for each new document based on its length. The values of these parameters are shown in Table 18.[15] Notice that two and three word phrases are weighted approximately 3 times more heavily than single word phrases, and that the reward for early occurrence of a phrase is to increase its score by a factor of around 10. On the whole Extractor is biased towards multi-word phrases that occur early in the document.

Turney compared the performance of Extractor to the key word extraction algorithms contained in Microsoft Word 97 and Verity's Search 97, and also to a simple-minded approach using the Brill tagger to find the most frequent noun phrases. Extractor beat all other algorithms on each of the five test corpora [TUR98]. It is important to note that all that can be inferred from this is that Extractor was better at its stated task – matching the keyphrases chosen by humans. It does not necessarily mean that Extractor will choose

[13] Extractor first considers only single-stem phrases and assembles a working list of those with the highest rank. Then this list is used to filter the number of multi-word phrases considered and to ensure that no two phrases contain the same stem

[14] For efficiency Extractor stems words by simply truncating them to a fixed length

[15] Extractor has recently been extended to allow the user to specify roughly how productive the system should be in extracting phrases. This was done by training the system using the F-measure for various values of β and then allowing the user to specify a value for β to use during extraction. The β value is used to set each parameter based on a linear fitting of the original learned values. The new version of Extractor (2.0) was used in these experiments, but the β parameter was left at its default value of 1.0 and the values reported in Table 18 are for $\beta = 1.0$. [Personal communication with Peter Turney.]

good phrases for classification. Nevertheless, given the initial intuition that author-chosen keyphrases will be highly meaningful and unambiguous, and given that the Extractor is faster than the Brill tagger, it is well worth exploring the use of Extractor for this purpose. However, it should be noted that good performance is not a foregone conclusion. In particular, the folk songs are structured very differently from the journal and e-mail articles on which Extractor was trained. The heavy weighting of early occurring terms may not be appropriate for folk songs, in which the main point of the song is often introduced in the final stanza.

Parameter Number	**Parameter Name**	**Value for Short Documents**	**Value for Long Documents**
1	NUM_PHRASES	12	11
2	NUM_WORKING	12	32
3	FACTOR_TWO_ONE	2.45	2.73
4	FACTOR_THREE_ONE	3.51	2.94
5	MIN_LENGTH_LOW_RANK	1.66	0.92
6	MIN_RANK_LOW_LENGTH	7	5
7	FIRST_LOW_THRESH	438	439
8	FIRST_HIGH_THRESH	878	1983
9	FIRST_LOW_FACTOR	7.36	10.61
10	FIRST_HIGH_FACTOR	0.51	0.46
11	STEM_LENGTH	7	8
12	SUPPRESS_PROPER	0	1

***Table 18*: Learned settings for the twelve parameters in the released version of Extractor. Short documents are less than 20 kilobytes in length. [Personal communication with Peter Turney.]**

6.2.2 The Change of Representation

One obvious way to use Extractor would be to represent each document only using the keyphrases extracted from that document. The problem with this representation is that Extractor only produces a handful of keyphrases for each document, resulting in an extremely sparse representation in which the vast majority of the features have the value 0 for every document. Instead, a global list of keyphrases is assembled by using Extractor on each document in the collection. This results in a list of phrase features each of which was judged to be relevant or meaningful to at least one document in the corpus. The global list of keyphrases is then used in the same way that the global list of noun phrases was used - each document is searched for each keyphrase. This process is summarized below.

i) During the first pass, a global list is assembled comprising all keyphrases extracted from all documents using Extractor.

ii) This global list then becomes the set of features and the second pass determines a value for each feature in each document. For the purposes of feature extraction, a phrase is said to "occur" in a document each time the words in the phrase are found in the proper order separated only by white space.

The number of features in the new representations was manageable for RIPPER, so no filtering was performed. However, it is worth noting that the feature extraction method does provides a very natural method of limiting the number of features if necessary. Since Extractor returns keyphrases in decreasing order of relevance, a decision to only include the top *k* phrases from each document would limit the total number of features in the representation by cutting off some of those judged to be less relevant.

6.3 Extensions to the FX system

In order to perform the various steps involved in noun phrase and key phrase extraction, some extensions had to be made to the FX system. These extensions are itemized below:

i) Added the ability to perform the preprocessing steps for the Brill Tagger (separating punctuation and optionally rearranging the carriage returns in a corpus).

ii) Added the ability to read in data from a tagged corpus and apply the regular expression for noun phrases to create a list of phrases.

iii) Integrated with the Extractor API (made available by Peter Turney) in order to extract a list of key phrases.

iv) Added the ability to search a document for phrases of arbitrary length.

6.4 Experiments

Both “bag of noun phrases” (BN) and “bag of keyphrases” (BK) representations were evaluated in experiments identical to the bag of words experiments of chapter 5. Stemmed versions of both representations (denoted BSN and BSK) were also evaluated. In these representations, the Lovins stemming algorithm was applied to each word in each phrase. In the noun phrase representation, the stemming occurred immediately after the syntactic tags were removed. For instance, the phrase *single/J sex/N hostels/N* would become the phrase *singl sex hostel* after the stemming and removal of tags. As before, all punctuation and case information was removed from the words, and the BN and BSN representations had phrases appearing fewer than 4 times in the training set removed.

Table 19 and Table 20 show a summary of the number of features and average phrase lengths in each representation. Two observations can be made here. First, Extractor tends to find more multi-word phrases than NoPE. The average phrase length is approximately 25% higher in the BK and BSK representations than in the corresponding BN and BSN representations, despite the fact that Extractor has a maximum phrase length of 3. The second observation is that both algorithms have more difficulty finding phrases in the DigiTrad domain. The average phrase length found in Reuters-21578 is about 25% higher than DigiTrad for all four phrase-based representations. This latter observation can probably be explained by noting the differences between DigiTrad and the corpora on which Brill’s tagger and Turney’s Extractor were trained (discussed further in section 6.7).

	BW	BSW	BN	BSN	BK	BSK
Features	27940	18590	21427	20061	23110	22122
Average Phrase Length	1.0	1.0	1.8	1.9	2.3	2.4

***Table 19*: Number of features and average phrase length for each representation for the Reuters-21578 corpus.**

	BW	BSW	BN	BSN	BK	BSK
Features	43301	28996	14204	12457	15287	14264
Average Phrase Length	1.0	1.0	1.4	1.5	1.8	1.8

***Table 20*: Number of features and average phrase length for each representation for the DigiTrad corpus.**

As in chapter 5, default options were used for RIPPER except for the enabling of negative tests and the varying of the loss ratio parameter (*L*). For consistency, the same values of L were used: {0.5, 1, 2, 4} for Reuters-21578, and {0.25, 0.5, 1, 2} for DigiTrad. These runs bracketed the micro-averaged breakeven points in all cases except for the key phrase representations in DigiTrad.

6.5 Results and Discussion

The presentation of results mirrors that of section 5.4. In all tables, the BW and BSW results from section 5.4 are repeated for quick comparison. The micro- and macro-averaged precision and recall for each value of *L* is shown for Reuters and DigiTrad in Table 21 and Table 22 respectively. Table 23 and Table 24 show the breakeven points and F-measures for each representation. Figure 16 and Figure 17 show the breakeven points graphically. As before, all extrapolated (as opposed to interpolated) breakeven points are identified with a superscript “e”, and the reported F-measures in each case are the maximum values obtained for the four runs.

For Reuters-21578, the differences in classification power between the four representations are very slight. BN turns in the highest performance so far at .827 breakeven and .830 $F_{1.0}$-measure. BK performed a little worse than BW, and the stemmed phrase representations performed a little worse than their non-stemmed counterparts, though not to the same degree as the stemmed word representation. On DigiTrad, the performance was worse with both phrase-based representations. In the case of noun phrases, the performance decrease was not significant, but performance from the key phrase representations was dramatically worse. (This latter result should be treated with caution, however, since the data points do not bracket the true breakeven point.)

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	**micro**	**.783**	**.865**	**.816**	**.834**	**.856**	**.746**	**.900**	**.627**
	$macro_0$	.435	.508	.435	.469	.432	.385	.444	.295
	$macro_1$	.776	.519	.819	.480	.839	.396	.872	.306
BSW	**micro**	**.774**	**.841**	**.812**	**.811**	**.858**	**.740**	**.894**	**.651**
	$macro_0$	.412	.463	.417	.433	.442	.377	.456	.302
	$macro_1$	.753	.474	.791	.444	.837	.388	.884	.313
BN	**micro**	**.781**	**.870**	**.823**	**.838**	**.855**	**.753**	**.884**	**.624**
	$macro_0$	.438	.515	.461	.490	.460	.404	.419	.288
	$macro_1$	.779	.526	.823	.501	.855	.415	.881	.298
BSN	**micro**	**.783**	**.867**	**.813**	**.833**	**.848**	**.747**	**.892**	**.639**
	$macro_0$	.453	.510	.453	.472	.460	.409	.443	.311
	$macro_1$	.793	.521	.816	.483	.844	.420	.872	.322
BK	**micro**	**.798**	**.845**	**.827**	**.801**	**.866**	**.717**	**.911**	**.612**
	$macro_0$	.437	.472	.439	.428	.420	.355	.425	.274
	$macro_1$	.799	.483	.824	.439	.860	.366	.898	.285
BSK	**micro**	**.790**	**.843**	**.833**	**.798**	**.866**	**.736**	**.900**	**.612**
	$macro_0$	.432	.463	.454	.428	.441	.355	.432	.281
	$macro_1$	.784	.474	.827	.439	.869	.366	.839	.292

***Table 21*: Micro- and Macro-averaged precision and recall obtained on the testing data of Reuters-21578 using the phrase-based representations.**

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	**micro**	**.318**	**.394**	**.375**	**.344**	**.480**	**.221**	**.580**	**.098**
	macro_0	.273	.353	.318	.313	.387	.202	.371	.087
	macro_1	.273	.353	.318	.313	.418	.202	.674	.087
BSW	**micro**	**.309**	**.410**	**.373**	**.348**	**.484**	**.210**	**.573**	**.101**
	macro_0	.273	.365	.328	.316	.410	.204	.416	.102
	macro_1	.273	.365	.328	.316	.501	.204	.628	.102
BN	**micro**	**.316**	**.404**	**.368**	**.345**	**.491**	**.188**	**.569**	**.104**
	macro_0	.284	.359	.319	.309	.374	.178	.326	.092
	macro_1	.284	.359	.319	.309	.495	.178	.695	.092
BSN	**micro**	**.309**	**.394**	**.368**	**.346**	**.451**	**.210**	**.603**	**.113**
	macro_0	.271	.351	.322	.318	.362	.197	.400	.108
	macro_1	.271	.351	.322	.318	.392	.197	.642	.108
BK	**micro**	**.315**	**.273**	**.388**	**.232**	**.501**	**.118**	**.601**	**.062**
	macro_0	.259	.275	.310	.234	.356	.123	.241	.050
	macro_1	.319	.275	.371	.234	.508	.123	.695	.050
BSK	**micro**	**.357**	**.211**	**.388**	**.166**	**.491**	**.110**	**.540**	**.065**
	macro_0	.312	.175	.305	.140	.282	.097	.279	.048
	macro_1	.342	.175	.396	.140	.565	.097	.825	.048

***Table 22*: Micro- and Macro-averaged precision and recall obtained on the testing data of DigiTrad using the phrase-based representations.**

		Maximum F-measures			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.828**	**.825**	**.847**	**.821**
	macro_0	.448	.469	.492	.434
	macro_1	.718	.622	.556	$.641^e$
BSW	**micro**	**.832**	**.811**	**.826**	**.810**
	macro_0	.421	.436	.452	.422
	macro_1	.684	.582	.512	$.597^e$
BN	**micro**	**.826**	**.830**	**.851**	**.827**
	macro_0	.466	.475	.498	.461
	macro_1	.730	.628	.563	$.618^e$
BSN	**micro**	**.827**	**.823**	**.849**	**.819**
	macro_0	.480	.497	.463	.455
	macro_1	.718	.629	.559	$.690^e$
BK	**micro**	**.830**	**.821**	**.835**	**.817**
	macro_0	.454	.465	.443	.439
	macro_1	.707	.602	.525	$.685^e$
BSK	**micro**	**.826**	**.815**	**.832**	**.816**
	macro_0	.448	.447	.457	.444
	macro_1	.703	.591	.515	$.613^e$

***Table 23*: Maximum F-measures and breakeven points (BP) obtained on the testing data of Reuters-21578 using the phrase-based representations. A Superscript "e" indicates an extrapolated point.**

		Maximum F-measures			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.389**	**.359**	**.433**	**.359**
	macro_0	.327	.315	.131	.315
	macro_1	.344	.315	.131	.315
BSW	**micro**	**.384**	**.360**	**.435**	**.360**
	macro_0	.341	.322	.136	.322
	macro_1	.388	.322	.136	.322
BN	**micro**	**.371**	**.356**	**.427**	**.357**
	macro_0	.317	.317	.135	.315
	macro_1	.365	.317	.135	.315
BSN	**micro**	**.367**	**.356**	**.424**	**.356**
	macro_0	.321	.320	.135	.320
	macro_1	.327	.320	.135	.320
BK	**micro**	**.342**	**.293**	**.329**	**$.288^e$**
	macro_0	.291	.267	.095	.320
	macro_1	.332	.295	.116	$.294^e$
BSK	**micro**	**.314**	**.266**	**.265**	**$.297^e$**
	macro_0	.269	.224	.071	$.346^e$
	macro_1	.291	.231	.078	$.241^e$

***Table 24*: Maximum F-measures and breakeven points (BP) obtained on the testing data of DigiTrad for BN, BSN, BK and BSK. A Superscript "e" indicates an extrapolated point.**

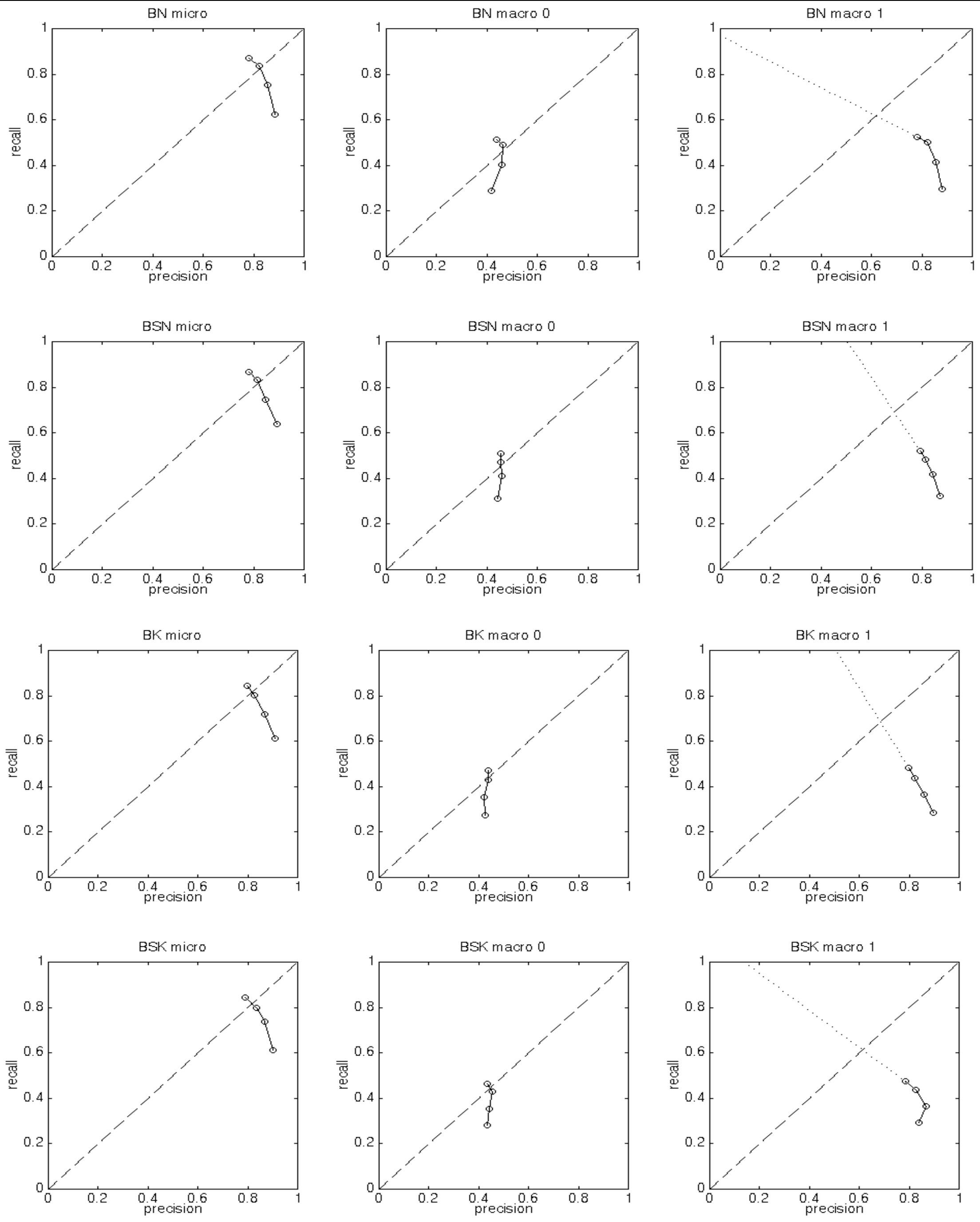

***Figure 16*: A graphical view of the Reuters-21578 breakeven points. The columns show the graphs for micro, macro$_0$ and macro$_1$ averaging respectively. From top to bottom the rows represent BN, BSN, BK and BSK.**

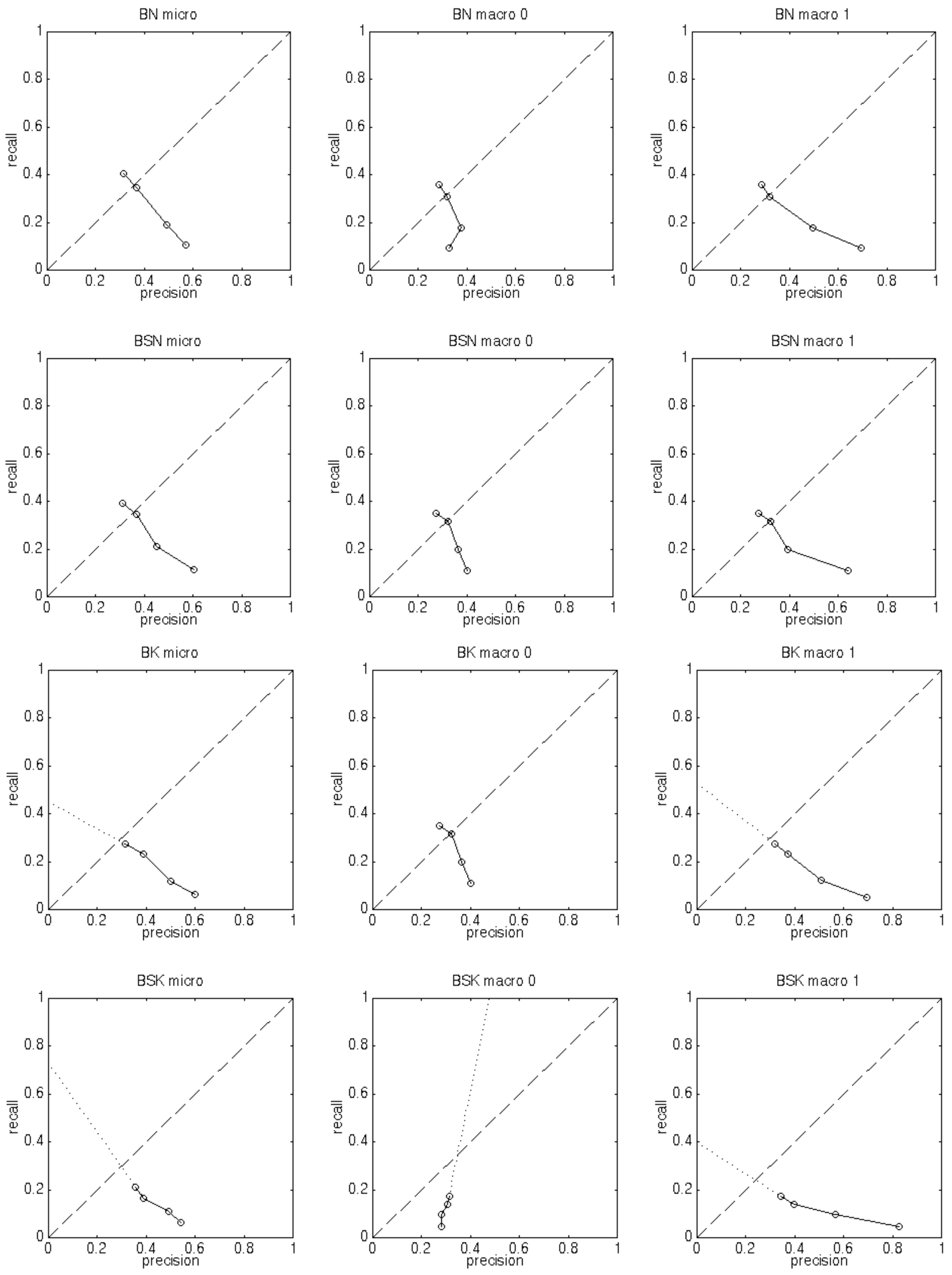

***Figure 17*: A graphical view of the DigiTrad breakeven points. The columns show the graphs for micro, macro$_0$ and macro$_1$ averaging respectively. From top to bottom the rows represent BN, BSN, BK and BSK.**

Table 25 and Table 26 show the micro-averaged results for each bag of phrases representation using the groupings defined in chapter 5. Table 27 and Table 28 show the F-measures and breakeven points computed for each group. Once again, all tables include the BW and BSW data for comparison. As before, the differences between representations are very slight, but one fact does stand out: on Reuters the noun phrase representations seem to derive their slightly greater classification power from the categories with fewer training examples. RIPPER's performance on the top two groups using BN is almost identical to BW, but BN scored higher on the bottom two. Similarly, BSN performed better than BSW on the bottom three groups. For the DigiTrad data, the phrase representations score uniformly lower on all three categories.

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	100+	.811	.905	.839	.885	.878	.805	.919	.693
	50+	.647	.778	.657	.714	.717	.609	.797	.353
	25+	.714	.835	.765	.775	.798	.798	.800	.491
	10+	.616	.682	.717	.553	.689	.689	.698	.335
BSW	100+	.813	.890	.839	.870	.882	.808	.914	.721
	50+	.576	.680	.618	.662	.662	.492	.712	.372
	25+	.697	.812	.745	.752	.829	.647	.852	.500
	10+	.626	.696	.741	.556	.739	.506	.800	.451
BN	100+	.806	.909	.846	.883	.873	.808	.899	.696
	50+	.623	.820	.662	.714	.731	.553	.763	.376
	25+	.750	.839	.785	.789	.787	.711	.813	.417
	10+	.661	.665	.714	.642	.739	.475	.710	.274
BSN	100+	.808	.910	.838	.880	.872	.797	.910	.706
	50+	.623	.752	.647	.737	.660	.620	.735	.406
	25+	.747	.826	.757	.757	.821	.693	.835	.440
	10+	.654	.698	.707	.592	.721	.447	.738	.346
BK	100+	.824	.890	.853	.855	.887	.775	.928	.677
	50+	.639	.805	.656	.736	.727	.590	.808	.395
	25+	.749	.794	.783	.647	.807	.596	.836	.468
	10+	.669	.542	.674	.508	.681	.346	.703	.251
BSK	100+	.822	.890	.864	.850	.891	.804	.922	.677
	50+	.597	.778	.647	.718	.660	.534	.688	.398
	25+	.723	.780	.753	.656	.781	.573	.822	.445
	10+	.642	.542	.674	.508	.773	.380	.794	.279

***Table 25*: Micro-averaged precision and recall for Reuters-21578 classes grouped by amount of training data.**

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	200+	.381	.460	.433	.407	.518	.264	.602	.123
	100+	.251	.316	.327	.267	.449	.161	.597	.056
	50+	.239	.311	.273	.272	.394	.182	.489	.086
BSW	200+	.353	.476	.432	.401	.513	.244	.589	.111
	100+	.278	.346	.341	.280	.459	.146	.544	.080
	50+	.223	.309	.263	.291	.421	.205	.556	.102
BN	200+	.371	.477	.418	.414	.542	.210	.612	.125
	100+	.255	.312	.328	.258	.461	.146	.564	.080
	50+	.240	.320	.269	.268	.395	.182	.432	.076
BSN	200+	.356	.457	.419	.400	.511	.241	.651	.130
	100+	.280	.330	.340	.276	.429	.163	.555	.085
	50+	.217	.302	.264	.285	.329	.188	.514	.102
BK	200+	.368	.281	.443	.248	.579	.121	.693	.078
	100+	.300	.280	.365	.211	.517	.117	.603	.057
	50+	.224	.238	.291	.216	.341	.114	.239	.020
BSK	200+	.390	.273	.438	.213	.579	.144	.557	.099
	100+	.323	.167	.341	.128	.302	.067	.458	.028
	50+	.244	.095	.247	.082	.463	.071	.500	.015

***Table 26*: Micro-averaged precision and recall for DigiTrad classes grouped by amount of training data.**

		Maximum F-measures			**BP**
		β=0.5	β=1.0	β=2.0	
BW	100+	.863	.861	.885	**.854**
	50+	.692	.706	.748	**.678**
	25+	.767	.770	.807	**.767**
	10+	.677	.647	.667	**.645**
BSW	100+	.867	.853	.874	**.851**
	50+	.626	.639	.657	**.627**
	25+	.785	.750	.786	**.748**
	10+	.618	.571	.577	**.572**
BN	100+	.859	.864	.886	**.856**
	50+	.687	.708	.771	**.678**
	25+	.786	.792	.820	**.785**
	10+	.699	.676	.664	**.664**
BSN	100+	.861	.858	.888	**.850**
	50+	.663	.689	.722	**.656**
	25+	.791	.784	.809	**.757**
	10+	.680	.676	.689	**.669**
BK	100+	.864	.855	.876	**.854**
	50+	.695	.712	.765	**.682**
	25+	.757	.771	.784	**.758**
	10+	.639	.599	.563	**.653**^{e}
BSK	100+	.872	.857	.876	**.857**
	50+	.660	.681	.734	**.651**
	25+	.734	.751	.768	**.734**
	10+	.640	.588	.559	**.594**^{e}

***Table 27*: Maximum F-measures and breakeven points (BP) obtained for Reuters-21578 classes grouped by amount of training data.**

		Maximum F-measures			**BP**
		β=0.5	β=1.0	β=2.0	
BW	200+	.435	.419	.442	**.420**
	100+	.330	.294	.301	**.291**
	50+	.320	.272	.293	**.273**
BSW	200+	.425	.416	.445	**.416**
	100+	.326	.308	.330	**.311**
	50+	.348	.276	.287	**.281**
BN	200+	.417	.417	.451	**.416**
	100+	.322	.289	.299	**.288**
	50+	.320	.274	.300	**.269**
BSN	200+	.417	.409	.433	**.409**
	100+	.325	.305	.318	**.306**
	50+	.286	.274	.281	**.272**
BK	200+	.383	.319	.295	**.308**^{e}
	100+	.318	.290	.284	**.290**^{e}
	50+	.272	.248	.235	**.235**
BSK	200+	.361	.321	.290	**.238**^{e}
	100+	.272	.220	.185	**.274**^{e}
	50+	.220	.137	.108	**.216**^{e}

***Table 28*: Maximum F-measures and breakeven points (BP) obtained for DigiTrad classes grouped by amount of training data.**

Table 29 shows the average number of rules per hypothesis, and the average number of clauses per rule for the grouped classes, and for all classes. The table also contains a statistic for "average phrase length". This was computed by averaging the number of words in each phrase occurring in each clause, and then dividing by the number of clauses. These numbers, representing the average length of the phrases used in the hypotheses, can be compared to the average length of the phrases that were available in the representations (shown in Table 19 and Table 20 above) to give some sense of the relative importance RIPPER attached to longer phrases. RIPPER still seems to prefer single words, although it was able to make some use of phrases, averaging 1 or 2 multi-word phrases per hypothesis. It is interesting to note that fewer phrases were used in the hypotheses formed using the DigiTrad representations, despite the greater abundance of phrases available in the training set.

Now average hypothesis complexity is considered. In the case of Reuters-21578, the phrase representations produced slightly less complex hypotheses than the words. This is quantified as approximately 0.3 fewer rules per hypothesis in the unstemmed representations and 1.0 fewer rules per hypotheses in the stemmed representations. Each rule contains about the same number of clauses in all representations. Separating the number of rules from number of clauses is important, because of the potential trivial result that fewer clauses can result from the combination of words (for example the two clauses "*machine* ∈ WORDS & *learning* ∈ WORDS" are replaced by the single clause "*machine_learning* ∈ PHRASES"). The results show that this is probably not happening: the rules contain the same average number of clauses, but the number of rules is reduced. Therefore since overall performance is approximately equal, the rules formed in the phrase representations must on average be more powerful. This complexity reduction does not guarantee more comprehensible rules overall, but rules like the ones shown in Figure 11 of section 4.4 give hope that, in some cases, the hypotheses will make more sense to the user.

The results are again not as encouraging on the DigiTrad corpus. Though hypotheses are less complex in the keyphrase case, this result is not interesting because of the overall poorer performance of keyphrases. It is likely that RIPPER simply had less to work with in the feature set. In the case of noun phrases, where performance was almost as good as words, the noun phrases use around 2.1 more rules per hypothesis, with about the same number of clauses per rule. This implies that the rules formed with noun phrases were on average *less* powerful than those formed using words. This possibility is discussed further in section 6.7.

		L.R.	$F_{1.0}$	Rules	Clauses	Clauses / Rule	Phrase Length
Reuters-21578	BW	1.0	.825	5.1	13.0	2.5	1.00
	BSW	1.0	.811	5.6	13.5	2.4	1.00
	BN	1.0	.830	4.8	11.7	2.4	1.09
	BSN	1.0	.823	4.6	12.1	2.6	1.12
	BK	1.0	.814	4.7	11.2	2.4	1.17
	BSK	1.0	.815	4.5	11.5	2.5	1.14
DigiTrad	BW	0.5	.359	12.9	30.3	2.3	1.00
	BSW	0.5	.360	13.3	33.0	2.5	1.00
	BN	0.5	.356	15.2	33.8	2.2	1.05
	BSN	0.5	.356	15.3	38.7	2.5	1.06
	BK	0.5	.290	11.7	23.1	2.0	1.06
	BSK	0.5	.232	12.9	31.8	2.8	1.04

***Table 29*: Complexity statistics for each representation. The column *L.R.* shows the loss ratio setting for the rule set, and *$F_{1.0}$* shows the micro-averaged $F_{1.0}$-measure achieved with the rules. *Rules* indicates the average number of rules per hypothesis. *Clauses* indicates the average number of clauses per hypothesis, and *Clauses/Rule* indicates the average number of clauses in each rule. *Phrase Length* shows the average number of words in each phrase used for classification.**

6.6 A Final Experiment

On the Reuters-21578 data, results from the BN representation were slightly better than BW, even though the feature sets were quite different, with an overlap of only around 50%. This lead to the theory that the BN representation had gained something from the introduction of new phrase features, but lost something in the screening out of some of the single word features. One final experiment was conducted to test this idea by combining noun phrases and words into a single representation. The original set of words from BW was screened to remove words appearing fewer than 4 times in the corpus, and then this set was combined with the multi-word phrases from BN to form a new representation (BNW) containing 24733 features with an average phrase length of 1.7. The experiments were repeated one more time for this representation, but the results according to almost every measuring technique fell between the BN and BW representation. Rather than present all the data for BNW, Table 30 summarizes the F-measures and breakeven points compared to BN and BW.

		Maximum F-measures			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.828**	**.825**	**.847**	**.821**
	macro_0	.448	.469	.492	.434
	macro_1	.718	.622	.556	$.641^e$
BNW	**micro**	**.821**	**.826**	**.850**	**.823**
	macro_0	.450	.471	.494	.451
	macro_1	.715	.625	.559	$.656^e$
BN	**micro**	**.826**	**.830**	**.851**	**.827**
	macro_0	.466	.475	.498	.461
	macro_1	.730	.628	.563	$.618^e$

***Table 30*: Maximum F-measures and breakeven points (BP) obtained on the testing data of Reuters-21578 using the BNW representation compared to BW and BN. A Superscript "e" indicates an extrapolated point.**

6.7 Why Did the Phrase-Based Representations Not Work?

The question that forms the title to this section is a rather provocative one, because by some measures the phrase-based representations did work in the way we hoped they would, at least for Reuters-21578. The original hypothesis presented at the beginning of this chapter was that the inclusion of multi-word phrases would give RIPPER access to word combinations with higher information content than individual words. This should have lead to more powerful and less complex rules and, hopefully to better overall performance on the testing data. Though this overall performance improvement did not materialize, there did seem to be evidence of less complex rules. Clearly the phrases were helping enough to enable fewer rules to do the work and improve performance on some classes, but overall they failed to make the classes any more separable.

In further defense of the phrase-based representations, it is worth pointing out that the BSN representation is a good method of feature reduction, producing higher performance than BSW with a similar number of features in the case of Reuters-21578, and producing approximately equal performance to BSW with a dramatic reduction in features in the case of DigiTrad. These numbers are reproduced in Table 31 below.

		Features	**BP**
Reuters-21578	**BW**	27940	.821
	BSW	18590 *(33% less)*	.810
	BSN	20061 *(28% less)*	.819
DigiTrad	**BW**	43301	.359
	BSW	28996 *(33% less)*	.360
	BSN	12457 *(71% less)*	.356

***Table 31*: Comparison of feature reduction versus performance as measured by micro-averaged breakeven point (BP).**

General performance on the DigiTrad domain, however, is a different story. The relatively bad performance of phrases could be due to the difficulties NoPE and Extractor had with the folk song format. If this is true, then it follows that since RIPPER's performance was worse with keyphrases, the performance of Extractor must have suffered more from the unusual text. The other possible interpretation is that using multi-word phrases always degrades performance, and this was worse for the keyphrase Extractor since it found longer phrases on average. But there are reasons to believe the first explanation and not the second.

An observation about the character of the folk songs in DigiTrad can explain both Extractor's low performance, and its ability to find more phrases. Folk songs often contain a lot of repetition of meaningless phrases, such as "*fol diddle*", "*fa la la*" or "*oh blah dee*". When these types of phrases occur in a song, they occur many times, making them seem more meaningful to Extractor. On the other hand, Brill's tagger probably does not know what to do with these phrases so NoPE may or may not have identified them as noun phrases. But the important point is that even if NoPE accepted these phrases, it would not consequently ignore other more legitimate phrases. Extractor on the other hand, is structured to ignore lower scoring phrases. The result is that Extractor would find many more long phrases, but the phrases would have very little classification power. NoPE on the other hand, would find relatively few long phrases, but form a better feature set overall. It is possible that retraining Extractor on a folk song corpus might help a little, but it is more likely that frequency measurements are simply not a good basis on which to extract meaningful phrases from song lyrics.

There is also subjective evidence that the phrases returned by Extractor are of poorer quality. On inspection, some examples stand out, such as the hypotheses for the class *love* shown in Figure 18 below. In the BW representation, the word *love* features prominently in all the rules, but in the BK representation, the word is not used at all, and the hypothesis suffers from it, returning an $F_{1.0}$-measure of 0.000 compared to 0.404 for the bag of words rule. The reason RIPPER did so poorly is that Extractor did not return the word *love* as a key phrase for any folk song in the collection (in fact the word *love* is included in Extractor's stop list of common verbs.)

The NoPE system was able to produce a feature set of almost the same quality as BW and BSW on the DigiTrad data. Unfortunately, this was at the price of *more complex* rules than before, implying that the feature set as a whole contained less informative features. The reason for this probably has to do again with the song lyric domain. The Reuters-21578 stories contain many stock phrases that are repeated over and over again. Phrases like *industrial production* and *current accounts* are standard terminology for the writers that are used frequently. On the other hand there is no standard terminology for folk songs, even songs about the same subject. In this context, single words are probably better as features simply because they are more likely to be repeated between songs than particular sequences of words. Indeed, this hypothesis is supported by the lower usage of phrases in the hypotheses formed for DigiTrad (shown in Table 29 above.)

love & heart & *not* who & *not* are & recorded & then ⇒ *love*
love & sf & heart ⇒ *love*
love & *not* old & *not* who & would & marry ⇒ *love*
love & *not* up & me & *not* come & *not* we & *not* was ⇒ *love*
love & my & *not* ve & exe & *not* rg & *not* at & by ⇒ *love*
love & recorded & tears & *not* amongst ⇒ *love*
love & heart & *not* away & *not* who & was ⇒ *love*
love & parents ⇒ *love*
love & alone & never ⇒ *love*
my & arb ⇒ *love*
me & sweetest & down ⇒ *love*
love & *not* for & were ⇒ *love*
default ⇒ *not love*.

heart & hair ⇒ *love*
true & lovers & *not* till & *not* court and *not* mind ⇒ *love*
heart & bosom ⇒ *love*
heart & kiss ⇒ *love*
robert_burns & sae ⇒ *love*
true & marry & *not* singing & *not* life & *not* maid ⇒ *love*
true & false ⇒ *love*
heart & dove ⇒ *love*
arms & rosy ⇒ *love*
heart & bob ⇒ *love*
fair & weel ⇒ *love*
default ⇒ *not love*

***Figure 18*: Example of two hypotheses formed for the DigiTrad class *love* (shown in a more compact form than usual due to the complexity of the rules.) The top hypothesis, formed using the BW representation, makes heavy use of the word *love*. The bottom hypothesis, formed using the BK representation, had to make do without this word, since it did not appear in the feature set. The BK rule failed to correctly classify any of the testing examples.**

6.8 Summary

This chapter looked at two methods of extracting phrase-based representations: statistically-defined phrases using Peter Turney's keyphrase Extractor, and syntactically-defined phrases using NoPE, a simple noun phrase extractor. The changes of representation produced almost identical performance to the bag of words representation. The one exception to this rule was the keyphrase representations for DigiTrad where performance was degraded due to the difficulties of statistically measuring phrase importance in song lyrics. On the Reuters-21578 corpus, both noun phrases and key phrases allowed RIPPER to form less complex, and therefore more comprehensible hypotheses. The noun phrase representations were also observed to perform slightly better than words when training data was limited. On the DigiTrad corpus, the noun phrase representations performed almost as well as the bag of words, but at the price of more complex rules. The "bag of stemmed noun phrases" representation was also shown to be a powerful feature reduction method that does not impact performance too badly.

6.9 Recommendations

On the basis of the discussion above, some recommendations can be offered to anyone thinking of using phrase-based representations with symbolic learning for text classification.

- In a technical domain such as Reuters, where comprehensibility is a concern and small increases in learning time are not an issue, noun phrases extracted using the NoPE system are a good choice of representation. The user can expect a very slight increase in performance with less complex rules. The use of a tagger on the training set will slow down the learning phase slightly (it took 12 hours to tag Reuters-21578 on a 200MHz Pentium with 128MB of RAM) but classification, if implemented efficiently, will not be any slower.

- In a non-technical domain such as DigiTrad, the phrase-based representations investigated are not recommended.

7. Classification with Synonyms and Hypernyms

The previous chapter began with the observation that the bag of words representation ignores much of the information contained in the original text. Focusing on word ordering led to two phrase-based representations as an alternative. This chapter looks at another type of information ignored by the bag of words model – semantic relations between words. Some methods are proposed and evaluated that incorporate semantic information from WordNet, a public domain on-line lexical reference system [MIL90, FEL98].[16] New text representations are engineered in which the features represent "word senses" and these representations are evaluated against the bag of words model.

This chapter proceeds in a slightly different way than the previous one. As in previous chapters, experiments are described on the ModApte and DT100 data sets, but first some preliminary work which originally appeared as [SCM98a] and [SCM98b] is reviewed in detail. In this preliminary work, the ModApte and DT100 data sets are replaced with smaller two-class problems drawn from Reuters-21578, DigiTrad, and USENET. The reason for including this work is that the change of representation as originally conceived was not compatible with RIPPER's set-valued features and did not scale up well to the ModApte and DT100 data sets. The results of the preliminary work were promising, but when the representation was converted to a form that would scale up, the results were quite disappointing. The task of finding a better way to scale up hypernym density is left as future work.

7.1 WordNet

WordNet is a large electronic thesaurus that captures a number of important conceptual relations in addition to the synonymy and antonymy found in most thesauri. These relations can be used to engineer a change of representation in text data by transforming vectors of words into vectors of word meanings. The synonymy relation can be used to map words with similar meanings together, and hypernymy (corresponding to the "is a" relation) can be used to generalize noun and verb meanings to a higher level of abstraction.

The nodes in WordNet's semantic network are referred to as "synsets". Each synset represents a particular semantic meaning shared by a group of words and phrases. Each word or phrase in the synset is a synonym of the others with respect to this particular meaning, and a word or phrase with more than one meaning will appear in more than one synset. The synset nodes therefore embody the concept of synonymy, while other semantic relations are modeled as arcs between the nodes. Synsets are also partitioned into part of speech (e.g. nouns and verbs) and the user of WordNet must specify ahead of time which partition should be used. Figure 19 shows an idealized piece of the WordNet hierarchy from the noun partition.

[16] WordNet is available for download at http://www.cogsci.princeton.edu:80/~wn/. Version 1.5 was used in this work.

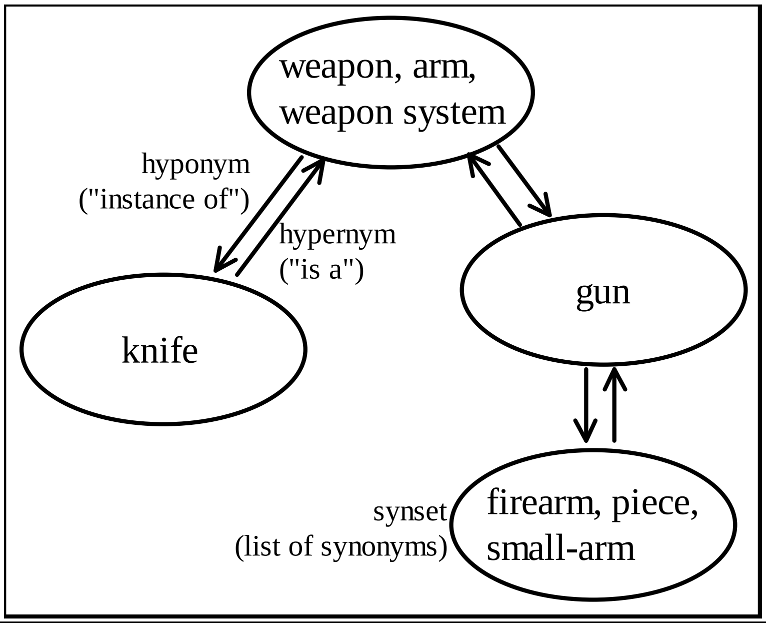
weapon, arm, weapon system
hyponym ("instance of")
hypernym ("is a")
knife
gun
synset (list of synonyms)
firearm, piece, small-arm

***Figure 19*: A piece of the WordNet semantic hierarchy (idealized model).**

When a word is "looked up" the user must specify the part of speech of the word as either noun, verb, adjective or adverb. WordNet then performs the following actions:

i) Convert the word to its root if possible. For example, the word "*was*" is converted to "*to be*" and "*cars*" is converted to "*car*". These roots are referred to as *lemmas* and the process is known as *lemmatization*.[17]

ii) Search the appropriate part of speech partition for all synsets containing the string from *i*. This list becomes the list of senses.

iii) If requested, traverse the semantic network along the links of the requested type and record all the synsets found.

iv) Display the list of synonymous and semantically related synsets from *ii* and *iii*.

Figure 20 shows the output from WordNet in response to a request for all the hypernyms of the noun *gun*. WordNet contains 3 noun senses for *gun*, and each corresponding synset is displayed, followed by all the synsets that appear above it in the hypernym hierarchy.

[17] Some words have their own senses, but are also a conjugation of other words. For example, the word "found" has its own verb senses (as in "to found an organization") but it is also a past tense conjugation of the verb "to find". In these cases, all the possible lemmas of the word will be looked up.

```
Synonyms/Hypernyms (Ordered by Frequency) of noun gun
3 senses of gun
Sense 1
gun
          ⇒ weapon, arm, weapon system
                    ⇒ weaponry, arms, implements of war, weapons system, munition
                              ⇒ instrumentality, instrumentation
                                        ⇒ artifact, artefact
Sense 2                                           ⇒ object, inanimate object, physical object
artillery, heavy weapon, gun, ordnance                      ⇒ entity
          ⇒ armament
                    ⇒ weaponry, arms, implements of war, weapons system, munition
                              ⇒ instrumentality, instrumentation
Sense 3                                 ⇒ artifact, artefact
gunman, gun                                       ⇒ object, inanimate object, physical object
          ⇒ shot, shooter                                   ⇒ entity
                    ⇒ expert
                              ⇒ person, individual, someone, somebody, mortal, human
                                        ⇒ life form, organism, being, living thing
                                                  ⇒ entity
                                        ⇒ causal agent, cause, causal agency
                                                  ⇒ entity
```

***Figure 20*: Some output typical of WordNet 1.5. The first line for each "sense" is a synset containing the word "gun". Each line preceded by "⇒" indicates a step upwards through the hypernym hierarchy of generalization. (The output has been formatted for presentation.)**

There has only been one previously published attempt to make use of WordNet for text classification. Rodríguez et al. [RGD97] used WordNet with linear discriminant learning algorithms to produce better classification performance on the Reuters-22173 data set. They were able to take advantage of the fact that Reuters topic headings often correspond to words that appear in the text of the documents. They compiled a list of all the topic headings and looked up these words in WordNet to find synonyms. This list of synonyms was then used to bias the learning algorithm for each topic by increasing the initial weights of the features corresponding to the synonyms of that topic. This entire process was done by hand and took advantage of the researchers' a priori knowledge of the Reuters-22173 domain. No change of representation was involved, and the use of WordNet was limited to the synonymy relation only.

Some researches in the IR community have also applied WordNet to retrieval tasks. Sharon Flank [FLA98] recently reported good results using many of the WordNet semantic relations, including hypernymy, to expand short descriptions of pictures for retrieval. Ellen Voorhees looked at the possibility of using WordNet for document indexing [VOO93] and query expansion [VOO94]. In these cases, the results were not particularly promising. The idea behind document indexing (which most closely resembles a change of representation) was to include synsets as indexing terms and then convert queries to synsets for retrieval. The query expansion idea was to replace words in a query with a list of synonyms from WordNet. When the queries were expanded or converted to synsets manually it was found that the method did improve some retrieval tasks significantly, but automatic expansion posed some real difficulties due to a problem known as *word sense disambiguation*.

Any attempt to use WordNet to look up synsets runs into the problem of how to find the correct senses for the words in the given context. This is the well-known problem of *word sense disambiguation*. WordNet's organization of synsets into parts of speech allows for some partial disambiguation, but each part of speech often still contains many different senses. The problem of automatic disambiguation has been and is currently being tackled by a number of researchers with some success (see [LSM94], [YAR95], and the plethora of papers in the *Proceedings of COLING-ACL'98* for some recent work), but the difficulty of this problem has hampered past attempts to use WordNet in the information retrieval community and the problem is often solved by manually selecting the correct word sense. The changes of representation discussed in this chapter proceed on the idea that in the presence of a large amount of training data, the problem of disambiguation can be partially sidestepped.

7.2 Preliminary Work

The general approach to feature engineering with WordNet is to map the original words into meta-features based on the semantic relationships between them. In this preliminary work, six binary classification tasks of varying difficulty are defined for RIPPER to learn. These tasks are drawn from three different corpora: Reuters-21578, DigiTrad, and USENET, the on-line discussion forum. In keeping with previous work, topic headings are the basis for the Reuters classification tasks and key words are the basis for DigiTrad. Since USENET was not used in the work discussed in previous chapters, and since the methodology for Reuters-21578 and DigiTrad differs slightly, some details will be provided now.

7.2.1 Notes on the Classification Tasks

USENET is an enormous, constantly changing corpus consisting of hundreds of thousands of news articles composed and posted by users of the USENET system. It is a public forum and anyone with an e-mail account can post a news story. The system is broken up into newsgroups, each with a unique name intended to denote the topic for discussion – a layout that lends itself naturally to a text classification task using newsgroup names as class labels. This task shares features with many other real-world tasks such as e-mail sorting, and web-site cataloging. For research purposes, USENET represents a huge pool of ever-changing training data. Users of USENET, on the other hand, could benefit from a system that suggested appropriate newsgroups based on the contents of an article they wished to post.

In addition to being a realistic task, USENET classification is also a difficult task. The users of a given newsgroup do not necessarily feel restricted to discussing the official newsgroup topic and often end up discussing tangential or completely unrelated themes. Furthermore, there is a widespread practice of cross-posting in which a single article is simultaneously submitted to two or more news groups. A number of researchers have used USENET newsgroups as a basis for their research. Joachims used Bayesian learning to achieve 89% accuracy on 20 selected USENET newsgroups [JOA96]. Lang's NewsWeeder system also used Bayesian learning to filter news stories according to a user's individual profile of preferences [LAN95]. In a different approach, Weiss et al. concentrated on classification within the *rec.** and *comp.** newsgroup hierarchies by forming models of typical documents for each group and comparing incoming documents to these models [WKB96]. Finally, Cohen studied the related problem of e-mail sorting using rule induction with RIPPER [COH96a].

Documents were extracted from USENET using the Netscape 3.0 news reader over a 6 week period in the Fall of 1997. USENET articles that were cross-posted or tagged as follow-ups were excluded so that the remaining articles reflected a wide variety of attempts to launch discussions within the given topics. Non-text objects such as uuencoded bitmaps were also removed from the postings. After extraction, the articles were concatenated into raw text files with the standard USENET header preceding each article. The documents were marked with SGML tags in a similar but slightly different format from Reuters-21578 and DigiTrad. The main difference stemmed from the importance of including the subject field in the text to be classified. The SGML style used was inspired by the Linguistic Data Consortium's Wall Street Journal corpus which contains a *Lead Paragraph* field in addition to the document body field.

As has been observed in previous chapters, not all types of text are equally difficult to classify. Reuters consists of articles written purely as a source of factual information. The writing style tends to be direct and to the point, and uses a restricted vocabulary to aid quick comprehension. DigiTrad and USENET are good examples of the opposite extreme. The texts in DigiTrad make heavy use of metaphoric, rhyming, unusual and archaic language and often the lyrics do not explicitly state what a song is about. Likewise, contributors to USENET often vary in their writing styles, stray from the topic, or use unusual language. All of these qualities tend to make subject-based classification tasks from DigiTrad and USENET more difficult than those of a comparable size from Reuters.

From the three corpora, six binary classification tasks were defined, as shown in Table 32. The tasks were chosen to be roughly the same size, and cover cases in which the classes seemed to be semantically related (REUTER2 and USENET2) as well as those in which the classes seemed unrelated (REUTER1 and USENET1). In all cases the classes were made completely disjoint by removing any overlapping examples. Since the tasks are binary two-class problems (*"A vs. B"* rather than *"A vs. notA"*) performance measures based on precision and recall could not be used. Instead, 10-fold cross-validated error replaced F-measure and breakeven points for evaluation.[18] Table 32 confirms once again that the intuitions about the difficulty of the six tasks for bag of words classification are valid. Error rates over 10-fold cross-validation for the Reuters

[18] In *n-fold cross-validation* the articles in the corpus are split into *n* partitions. Then the learning algorithm is executed *n* times. On the k^{th} run, partition *k* is used as a *testing set* and all the other partitions make up the *training set*. The mean error-rate (percentage of the *testing set* wrongly classified) on the *n* runs is taken as an approximate measure of the real error-rate of the system on the given corpus.

tasks were under 5%, while error rates for the other tasks ranged from approximately 19% to 38%.

It is worth noting that difficult tasks for RIPPER are not necessarily difficult for human classifiers. 200 examples from each of the SONG1 and SONG2 tasks were classified by hand (with no special training phase) and the results were compared to the classes indicated by the DigiTrad labels. The error rates were approximately 1% for SONG1 and 4% for SONG2. Clearly the background knowledge a human brings to a classification task can be used to overcome the difficulties posed by the text itself.

Task Name	**Source**	**Classes**	**Size**	**Balance**	**Words**	**Error**
REUTER1	Reuters-21578	*livestock* / *gold*	224	98/126	154	1.75
REUTER2	Reuters-21578	*corn* / *wheat*	313	130/183	173	3.87
SONG1	DigiTrad	*murder* / *marriage*	424	200/224	331	30.23
SONG2	DigiTrad	*political* / *religion*	432	194/238	241	32.64
USENET1	USENET	*soc.history* / *misc.taxes.moderated*	249	79/170	166	19.92
USENET2	USENET	*bionet.microbiology* / *bionet.neuroscience*	280	117/163	152	37.86

***Table 32*: Information on the classification tasks discussed in this section. "Size" refers to total number of examples in each task. "Balance" shows number of examples in each class. "Words" shows the average length of the documents in each task. "Error" shows the average percentage error rates for each task using RIPPER with bag of words and 10-fold cross-validation.**

7.2.2 The **Hypernym Density** *Representation*

If every word could be correctly disambiguated, a function similar to stemming could be performed in which synonymous words were mapped to the same meta-feature. The difference is that the mapping would be semantically rather than morphologically based. To extend this concept, the “is a” links could also be followed through the semantic network to generalize meanings by mapping to the hypernym senses of each word. To see why this might help, we can consider the songs in the *murder* category of SONG1. These songs often refer to weaponry of various sorts, whereas the songs in the *marriage* category almost never do. Yet words like *knife* or *gun* never appear in the bag of words rules learned by RIPPER. Most likely this is because there are enough different words for weapons that the information gain is never enough for RIPPER to decide to use a single one of them. However, if every occurrence of a word like *knife* or *gun* was mapped to a feature representing the hypernym *weapon*, then this new “meta-feature” might become usable. (In fact this exactly what happens, as discussed in section 7.2.5.)

The previous example assumed the use of automatic disambiguation – a difficult problem, especially on a large corpus. The key observation that allows the change of representation to proceed is that simply including all synsets without disambiguation may still produce a feature space with highly informative features. RIPPER may still be able to find the useful synsets among all the noise. So even if *knife* has meanings other than *weapon*, the *weapon* hypernym will still be included and available for classification. The fact that *knife* is also a hyponym of *edged tool* should not pose a problem. Following on these observations, no disambiguation is attempted during the change of representation. Instead all senses returned by WordNet are judged equally likely to be correct, and all of them are included in the feature set. To help RIPPER pick the correct ones, we define a *density* measurement representing the normalized frequency of appearance of a synset. This is attempt to capture some measure of relevancy of the synsets. Hopefully, the many different but synonymous or hyponymous words will map to common synsets, thus raising the densities of the "more relevant" synsets.

The algorithm for computing hypernym density requires three passes through the corpus.

a) During the first pass, the Brill tagger assigns a part of speech tag to each word in the corpus.

b) During the second pass, all nouns and verbs are looked up in WordNet and a global list of all synonym and hypernym synsets is assembled. Infrequently occurring synsets are discarded, and those that remain form the feature set. (A synset is defined as infrequent if its frequency of occurrence over the entire corpus is less than 0.05N, where N is the number of documents in the corpus.)

c) During the third pass, the density of each synset (defined as the number of occurrences of a synset in the WordNet output divided by the number of words in the document) is computed for each example resulting in a set of numerical feature vectors.

The calculations of frequency and density are influenced by the value of a parameter *h* that controls the *height of generalization*. This parameter can be used to limit the number of steps upward through the hypernym hierarchy for each word. At height $h=0$ only the synsets that contain the words in the corpus will be counted. At height $h>0$ the same synsets will be counted as well as all the hypernym synsets that appear up to *h* steps above them in the hypernym hierarchy. A special value of $h=max$ is defined as the level in which *all* hypernym synsets are counted, no matter how far up in the hierarchy they appear.

In the new representation, each feature represents a *set* of either nouns or verbs. At $h=max$, features corresponding to synsets higher up in the hypernym hierarchy represent supersets of the nouns or verbs represented by the less general features. At lower values of *h*, the nouns and verbs represented by a feature (synset) will be those that map to synsets up to *h* steps below it in the hypernym hierarchy. The best value of *h* for a given text classification task will depend on characteristics of the text such as use of terminology, similarity of topics, and breadth of topics. It will also depend on the characteristics of WordNet itself.

The fact that each word potentially maps to dozens of features in the new representation could lead to an explosion of dimensionality far worse than the bag of words approach. Fortunately, the overlapping nature of the synsets along with the part of speech and frequency filtering described above counteracts this tendency and reduces the total number of features in the new representation significantly. A comparison of the dimensionality of the two representations is shown in Table 33.

Clearly the change of representation process leaves a lot of room for inaccuracies to be introduced to the feature set. Some sources of potential error are: a) the tagger, b) the lack of true word sense disambiguation, c) words missing from WordNet, and d) the shallowness of WordNet's semantic hierarchy in some knowledge domains.

Data Set	Bag of Words	Hypernym Density			
		h=0	h=1	h=6	h=max
REUTER1	4227	1205	2122	2573	2573
REUTER2	4738	1127	2012	2446	2449
SONG1	9902	1894	3268	3842	3844
SONG2	10550	1539	2787	3346	3347
USENET1	6295	1213	2185	2627	2627
USENET2	6852	1104	2010	2451	2451

***Table 33*: Comparison of the number of features in the bag of words representation to the hypernym density representations at various values of h.**

7.2.3 Extensions to FX

Because WordNet's API is written in LISP, it was not easy to integrate the database directly with the FX system. Instead the strategy was to assemble batch files to look up words through WordNet's command-line interface and store the output in a text file which was then converted to a form readable by FX. This latter step was accomplished with a new utility WNPrepare, included as a companion utility to FX. In addition to reformatting, WNPrepare also implements the *height of generalization* parameter by screening out hypernyms more than *h* steps up in the hierarchy.

The implementation of the change of representation therefore had 4 steps:

i. run FX to generate a batch file,
ii. run the batch file,
iii. run WNPrepare on the output of the batch file, and
iv. run FX again using this output to change the representation to bag of hypernyms.

The following functionality was added to FX in order to implement this process:

i. the ability to read a tagged text and generate a batch file to look up all words tagged as nouns and verbs in WordNet,
ii. the ability to assemble a word-to-synset translation table from the output of WNPrepare, and
iii. the ability to form features by looking up words in the translation table.

7.2.4 Experiments and results – Accuracy

The new hypernym density representation differs in three important ways from the bag-of-words: a) all words are discarded except nouns and verbs, b) filtered normalized density vectors replace binary vectors, and c) hypernym synsets replace words. To show convincingly that improvements in accuracy are derived solely from the use of synsets, two normalizing experiments were performed using the following representations:

a) bag-of-words using only nouns and verbs, and
b) filtered normalized density vectors for nouns and verbs.

The results of these two runs were compared to the bag of words approach using 10-fold cross-validation (see Table 34) and in no case was any statistically significant difference found.

For the main experiments, average error rates over 10-fold cross-validation were compared for all six classification tasks using hypernym density representations with values of *h* ranging from *0* to *9* and *h=max*. For each classification task the same 10 partitions were used on every run so the results could be tested for significance using a paired t-test. Table 35 shows a comparison of three error rates: bag of words, hypernym density with *h=max*, and finally hypernym density using the best value for *h*. (For completeness, the full set of results is shown in Table 36.)

In the case of the Reuters tasks, no improvements over bag of words were expected and none were observed. On the other hand, a dramatic reduction in the error rate was seen for SONG1 (47% drop in number of errors for *h=9*) and USENET1 (34% drop for *h=9*). For the SONG2 and USENET2 data sets, the use of hypernyms produced error rates comparable to bag of words. The discussion of these results is left to section 7.2.6.

Task	Bag of Words	Bag of Nouns and Verbs	Noun and Verb Dens.
REUTER1	1.75	1.75	1.75
REUTER2	3.87	4.19	5.48
SONG1	30.23	27.35	27.67
SONG2	32.64	28.23	29.56
USENET1	19.92	18.56	20.45
USENET2	37.86	37.86	35.00

***Table 34*: Comparison of error rates over 10-fold cross-validation for the normalizing experiments. No statistically significant benefit or harm is derived from any of these changes of representation.**

Task	**Bag of Words**	**Hypernym Density**			
		h	**error**	***h***	**error**
REUTER1	1.75	max	2.38	0	1.75
REUTER2	3.87	max	6.13	0	4.84
SONG1	30.23	max	*22.04*	9	*16.00*
SONG2	32.64	max	34.45	4	31.04
USENET1	19.92	max	*14.36*	9	*13.11*
USENET2	37.86	max	40.00	2	36.43

***Table 35*: Quick comparison of percentage error rates over 10-fold cross-validation for the six data sets. Statistically significant improvements over bag of words are shown in Italics.**

Data Set	h=0	h=1	h=2	h=3	h=4	h=5	h=6	h=7	h=8	h=9	h=max
REUTER1	1.8	3.1	2.2	2.2	2.2	2.7	2.7	2.7	2.7	2.7	2.4
REUTER2	4.8	6.8	6.8	6.1	5.8	7.1	7.1	6.8	7.7	7.7	6.1
SONG1	27.7	20.8	23.8	18.8	23.6	19.4	25.6	17.7	21.4	16.0	22.0
SONG2	34.7	35.0	34.7	35.0	31.0	35.7	34.3	37.5	36.6	37.4	34.5
USENET1	22.5	21.5	16.1	17.3	18.8	15.2	15.5	14.7	13.6	13.1	14.4
USENET2	38.2	44.5	36.4	41.4	38.2	45.4	45.0	37.9	38.9	42.5	40.0

***Table 36*: Complete results for the Hypernym Density representation at various values of *h*. All results are error rates over 10-fold cross-validation. The best result obtained for each task is emphasized in boldface.**

7.2.5 Experiments and Results – Comprehensibility

A systematic investigation of the comprehensibility of rules produced using hypernym density versus bag of words was beyond the scope of the preliminary work. However, some evidence was noted of the better comprehensibility of the rules produced from the hypernym density representation.

Figure 21 shows a comparison of the rules learned by RIPPER on the USENET1 data set. The results for both bag of words and hypernym density are shown for the same fold of data. In the case of hypernyms, RIPPER has learned a simple rule saying that if the synset *possession* has a low density, the document probably belongs in the history category. Over the 10 folds of the data, seven folds produced a rule almost identical to the one shown. For the remaining three folds, the *possession* hypernym appeared along with other synsets in slightly different rules. The hyponyms of *possession* include words such as *ownership*, *asset*, and *liability* - the sorts of words used often during discussions about taxes, but rarely during discussions about history. On the other hand, the rules learned on the bag of words data seem less comprehensible: they are more elaborate and less semantically clear. Furthermore, the rules tended to vary widely across the 10 folds, suggesting that they were less robust and more dependent on the specifics of the training data.

possession $\leq 2.9 \Rightarrow$ soc.history
default $\Rightarrow$ misc.taxes.moderated

Rule learned using hypernym density

for a document D,
("tax" $\notin$ D & "history" $\in$ D) OR
("tax" $\notin$ D & "s" $\in$ D & "any" $\in$ D) OR
("tax" $\notin$ D & "is" $\notin$ D & "and" $\notin$ D & "if" $\notin$ D & "roth" $\notin$ D) OR
("century" $\in$ D) OR
("great" $\in$ D) OR
("survey" $\in$ D) OR
("war" $\in$ D) $\Rightarrow$ soc.history
default $\Rightarrow$ misc.taxes.moderated

Rule learned using bag of words

***Figure 21*: A comparison of the rules learned by RIPPER using hypernym density with *h=max* (top) and bag of words (bottom) on a single fold of the USENET1 data. The bottom rule produced twice as many errors on the holdout data.**

A similar situation is depicted in Figure 22 for the SONG1 data set. In this case it is interesting to note the use of the hypernym *weapon_arm_weapon_system* which appeared in the rules across all 10 folds. The use of this hypernym is significant because its hyponyms, such as *knife* and *gun* were never used in the bag of words approach.

(marry ≥ 0.8 & termination ≤ 0) OR
(kill ≤ 0.45 & weapon_arm_weapon_system $\leq$ 1.09) OR
(abstraction $\leq$ 12.15) $\Rightarrow$ marriage
default $\Rightarrow$ murder

Rule learned using hypernym density

for a document D,
("married" $\in$ D) OR
("dt" $\notin$ D & "was" $\notin$ D & "child" $\notin$ D & "came" $\notin$ D) OR
("wife" $\in$ D & "murder" $\notin$ D & "no" $\in$ D & "killed" $\notin$ D) OR
("wed" $\in$ D) OR
("tailor" $\in$ D) OR $\Rightarrow$ marriage
default $\Rightarrow$ murder

Rule learned using bag of words

***Figure 22*: A comparison of the rules learned by RIPPER using hypernym density with *h=max* (top) and bag of words (bottom) on a single fold of the SONG1 data. Some hypernym names have been shortened to save space. The bottom rule produced 60% more errors on the holdout data.**

7.2.6 Discussion

Hypernym density was observed to greatly improve classification accuracy in some cases, while in others the improvements were not particularly spectacular. In the case of the Reuters tasks, this lack of improvement is not a particular worry. It was very unlikely that any change of representation could have improved on the accuracy of bag of words for these tasks. The other cases, however, need a little more explanation.

In the case of the SONG2 task, the main problem seemed to be that the classes (*political* and *religion*) were more closely semantically related than their class labels suggested. Visual inspection of these songs revealed that many of the political songs contain statements about religion, make references to religious concepts, or frame their messages in religious terminology. This was the source of the higher error rate reported in section 7.2.1 when these songs were classified by hand. Inspection of RIPPER's output revealed that bag of words rules made heavy use of religious words such as *Jesus*, *lord*, and *soul*, while the hypernym rules mostly contained highly abstract political synsets such as *social group* and *political unit*. It is possible that subtle differences in religious terminology (for instance between gospel hymns and political parodies of religion) were mapped to common synsets in WordNet.

In the case of USENET2 the problem was two-fold. The classes were semantically closely related (*microbiology* and *neuroscience*) and the writers tended to use highly technical terms that are not found in WordNet. Some examples of missing words include *neuroscientist*, *haemocytometer*, *HIV*, *kinase*, *neurobiology*, and *retrovirus*[19]. An attempt was made to add the missing words manually into the WordNet hierarchy, but even then the extended semantic hierarchy was not fine-grained enough to allow meaningful generalizations. Because of the shallowness of the hierarchy, *overgeneralization* quickly became a problem as the height of generalization increases. This is probably why the best error rate for USENET2 using hypernym density was found at $h=2$.

7.2.7 Conclusions from the preliminary experiments.

These preliminary experiments gave some reason to believe that incorporating WordNet knowledge into text representation could lead to significant reductions in error rates on certain types of text classification tasks. An observed side benefit of the hypernym density representation was that RIPPER often formed simpler and more comprehensible hypotheses. The preliminary experience indicated that the hypernym density representation could work well for texts that use an extended or unusual vocabulary, or are written by multiple authors employing different terminologies. It is not likely to work well for text that is guaranteed to be written concisely and efficiently, such as the text in Reuters-21578. In particular, hypernym density was more likely to perform well on classification tasks involving narrowly defined and/or semantically distant classes such as those in SONG1 and USENET1.

7.3 Further Experiments

This section describes the ways in which the preliminary work was extended for the current study, and presents experimental results that can be compared to those from chapters 5 and 6. The presentation and discussion in this section closely mirrors the previous chapters.

[19] Some of these terms do appear in WordNet 1.6

7.3.1 From Hypernym Density to a Bag of Hypernyms

The main difficulty in using hypernym density for the larger experiments was the necessary use of continuous valued features. In moving to the larger classification problems, the work became much less tractable for RIPPER. In order to counter this, the density representation was abandoned in favor of a more generic “bag of hypernyms” in which the features took binary values, exactly as in the bag of words and bag of phrases representations. Unfortunately, this meant some loss of information, and therefore some potential loss of classification power. RIPPER could still select the hypernyms at any level of generality, but was likely to be more susceptible to “noise” in which inappropriate synsets are triggered by the wrong senses of some words. The change of representation process was almost exactly the same as described in section 7.2.2. The only difference was that the third pass no longer involves a density calculation. Instead, any synset found was simply inserted into the representation as a unique token, allowing RIPPER to use a set-valued feature to describe the documents.

7.3.2 Experiments

The bag of hypernyms representation was evaluated using experiments similar in methodology to the experiments of chapters 5 and 6. Two representations were attempted, corresponding to $h=0$ and $h=1$. These representations are denoted BH0 and BH1 in the discussion which follows. Low values for h were chosen for two reasons:

i) despite the positive results obtained for some classes in the preliminary work, the results with stemming reported in chapters 5 and 6 cautioned against using overly general meta-features in the larger tasks, and

ii) more importantly the BH representations were much less tractable for RIPPER (140 hours for BH1 on Reuters-21578 versus less than 24 hours each for the word and phrase-based representations.)

The reason for the latter observation is that at a higher h, more synsets will be triggered for each document and the feature vectors become much less sparse. Since much of RIPPER’s speed derives from its sparse vector representation, it is understandable that this would cause a significant slowdown in performance.

Some minor changes in methodology were required due to the fact the BH representations differ in a number of ways from the word and phrase representations. Stemming does not apply to bag of hypernyms in any meaningful way, and since WordNet contains information on proper nouns and abbreviations the case information in the original words and in the resulting synsets was preserved. Note that the new representations produced a dramatic reduction in the number of features, as shown in Table 37.

	Words		**Features**		**BSW**
	looked up	**found**	**BH0**	**BH1**	
Reuters	27639	13875	13385	15393	27940
DigiTrad	43406	20664	18646	21044	43301

***Table 37*: Number of features in each representation for each corpus. Table shows the total number of nouns and verbs looked up in WordNet, the number that had an entry, and the number of features extracted for h=0 and h=1. The number of features in BSW is included as a reference point.**

7.3.3 Results and Discussion

The presentation in this section mirrors that of sections 5.4 and 6.5. All tables have the BW and BSW results reproduced for quick comparison. The micro- and macro-averaged precision and recall for each value of L is shown for Reuters and DigiTrad in Table 38 and Table 39 respectively. Table 40 and Table 41 show the breakeven points and F-measures for each representation. Figure 23 and Figure 24 show the breakeven points graphically. As before, all extrapolated (as opposed to interpolated) results are identified with a superscript "e", and the reported F-measures in each case are the maximum values obtained for the four runs. Results across the board are lower than for the bag of words – around 7 points lower for DigiTrad and 8 points lower for Reuters. Following on preliminary work, the degradation was expected for Reuters-21578, but the poor performance on the DigiTrad domain was a surprise. Possible reasons are discussed in section 7.3.4.

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	**micro**	**.783**	**.865**	**.816**	**.834**	**.856**	**.746**	**.900**	**.627**
	macro_0	.435	.508	.435	.469	.432	.385	.444	.295
	macro_1	.776	.519	.819	.480	.839	.396	.872	.306
BSW	**micro**	**.774**	**.841**	**.812**	**.811**	**.858**	**.740**	**.894**	**.651**
	macro_0	.412	.463	.417	.433	.442	.377	.456	.302
	macro_1	.753	.474	.791	.444	.837	.388	.884	.313
BH0	**micro**	**.757**	**.721**	**.785**	**.685**	**.823**	**.656**	**.858**	**.559**
	macro_0	.376	.381	.372	.341	.388	.306	.375	.245
	macro_1	.761	.392	.779	.352	.839	.317	.902	.256
BH1	**micro**	**.737**	**.732**	**.772**	**.700**	**.810**	**.647**	**.850**	**.596**
	macro_0	.374	.380	.389	.354	.410	.313	.419	.261
	macro_1	.748	.391	.774	.365	.828	.324	.880	.272

***Table 38*: Micro- and Macro-averaged precision and recall obtained on the testing data of Reuters-21578 using the BH representations.**

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	**micro**	**.318**	**.394**	**.375**	**.344**	**.480**	**.221**	**.580**	**.098**
	macro_0	.273	.353	.318	.313	.387	.202	.371	.087
	macro_1	.273	.353	.318	.313	.418	.202	.674	.087
BSW	**micro**	**.309**	**.410**	**.373**	**.348**	**.484**	**.210**	**.573**	**.101**
	macro_0	.273	.365	.328	.316	.410	.204	.416	.102
	macro_1	.273	.365	.328	.316	.501	.204	.628	.102
BH0	**micro**	**.271**	**.303**	**.319**	**.227**	**.415**	**.142**	**.498**	**.047**
	macro_0	.232	.268	.266	.204	.288	.119	.230	.046
	macro_1	.232	.268	.266	.204	.379	.119	.715	.046
BH1	**micro**	**.247**	**.311**	**.302**	**.263**	**.427**	**.127**	**.478**	**.060**
	macro_0	.206	.268	.243	.228	.312	.111	.222	.051
	macro_1	.206	.268	.273	.228	.433	.111	.646	.051

***Table 39*: Micro- and Macro-averaged precision and recall obtained on the testing data of DigiTrad using the BH representations.**

		Maximum F-measure			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.828**	**.825**	**.847**	**.821**
	macro$_0$	.448	.469	.492	.434
	macro$_1$	.718	.622	.556	.641^e
BSW	**micro**	**.832**	**.811**	**.826**	**.810**
	macro$_0$	.421	.436	.452	.422
	macro$_1$	.684	.582	.512	.597^e
BH0	**micro**	**.775**	**.739**	**.728**	**.741**^{e}
	macro$_0$	.377	.379	.380	.375
	macro$_1$	.640	.518	.434	.647^e
BH1	**micro**	**.783**	**.735**	**.733**	**.734**^{e}
	macro$_0$	.382	.377	.379	.376
	macro$_1$	.632	.513	.432	.570^e

***Table 40*: Maximum F-measures and breakeven points (BP) obtained on the testing data of Reuters-21578 using word and hypernym representations. A Superscript "e" indicates an extrapolated point.**

		Maximum F-measure			BP
		β=0.5	β=1.0	β=2.0	
BW	**micro**	**.389**	**.359**	**.433**	**.359**
	macro$_0$	.327	.315	.131	.315
	macro$_1$	.344	.315	.131	.315
BSW	**micro**	**.384**	**.360**	**.435**	**.360**
	macro$_0$	.341	.322	.136	.322
	macro$_1$	.388	.322	.136	.322
BH0	**micro**	**.299**	**.286**	**.344**	**.283**
	macro$_0$	.250	.248	.083	.245
	macro$_1$	.263	.248	.083	.245
BH1	**micro**	**.293**	**.281**	**.351**	**.281**
	macro$_0$	.240	.235	.074	.236
	macro$_1$	.275	.249	.082	.245

***Table 41*: Maximum F-measures and breakeven points (BP) obtained on the testing data of DigiTrad using word and hypernym representations. A Superscript "e" indicates an extrapolated point.**

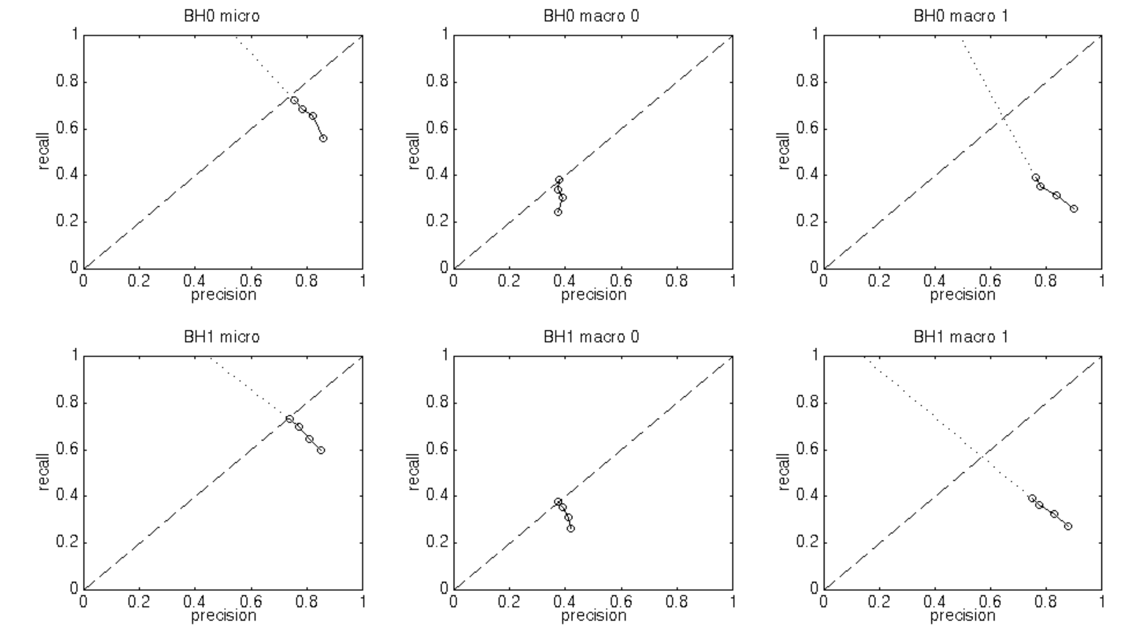

***Figure 23*: A graphical view of the Reuters-21578 breakeven points. Each row shows the graphs for micro, macro$_0$ and macro$_1$ averaging respectively. From top to bottom the rows represent BH0 and BH1.**

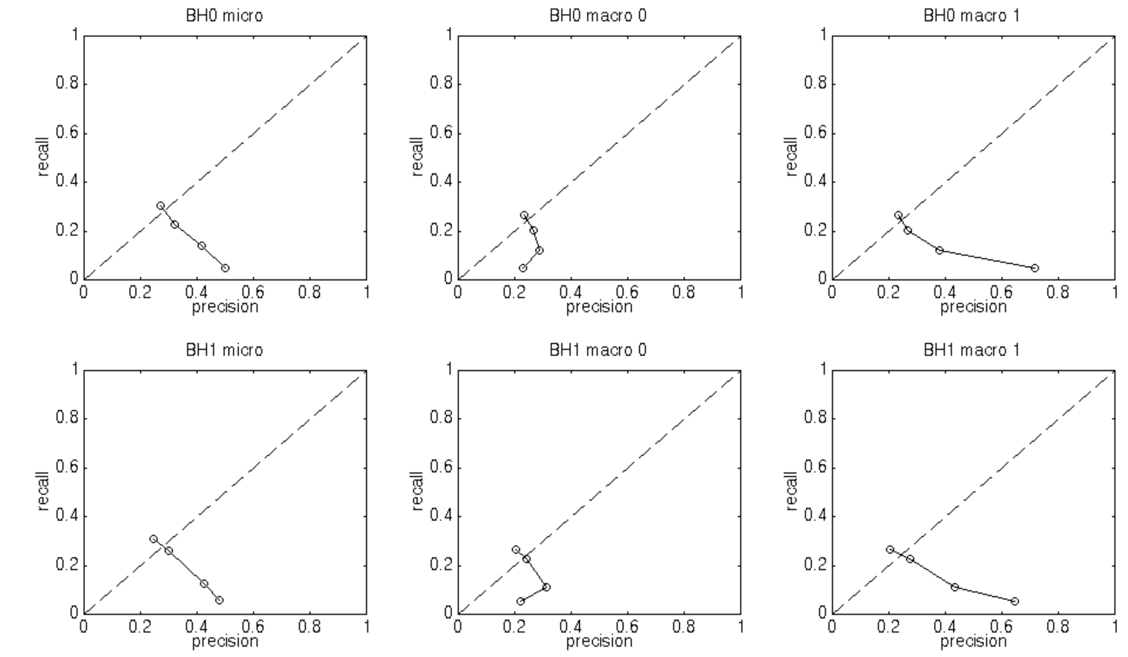

***Figure 24*: A graphical view of the DigiTrad breakeven points. Each row shows the graphs for micro, macro$_0$ and macro$_1$ averaging respectively. From top to bottom the rows represent BH0 and BH1.**

Table 42 and Table 43 show the results of each bag of hypernyms run broken down by number of positive training examples. The groups are defined in the same way as sections 5.4 and 0. Table 44 and Table 45 show the F-measures and breakeven points computed for each group. Results again were lower across the board.

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BW	100+	.811	.905	.839	.885	.878	.805	.919	.693
	50+	.647	.778	.657	.714	.717	.609	.797	.353
	25+	.714	.835	.765	.775	.798	.798	.800	.491
	10+	.616	.682	.717	.553	.689	.689	.698	.335
BSW	100+	.813	.890	.839	.870	.882	.808	.914	.721
	50+	.576	.680	.618	.662	.662	.492	.712	.372
	25+	.697	.812	.745	.752	.829	.647	.852	.500
	10+	.626	.696	.741	.556	.739	.506	.800	.451
BH0	100+	.793	.768	.814	.735	.843	.716	.873	.620
	50+	.603	.669	.635	.662	.690	.553	.748	.402
	25+	.668	.693	.684	.606	.777	.528	.811	.394
	10+	.457	.330	.563	.251	.580	.223	.642	.190
BH1	100+	.773	.782	.803	.750	.829	.714	.861	.657
	50+	.536	.643	.569	.632	.654	.447	.737	.432
	25+	.661	.706	.721	.628	.760	.450	.836	.422
	10+	.504	.330	.589	.313	.652	.324	.770	.263

***Table 42*: Micro-averaged precision and recall for Reuters-21578 classes grouped by amount of training data for the BH0 and BH1 representations.**

		L=0.5		L=1.0		L=2.0		L=4.0	
		P	R	P	R	P	R	P	R
BH0	200+	.381	.460	.433	.407	.518	.264	.602	.123
	100+	.251	.316	.327	.267	.449	.161	.597	.056
	50+	.239	.311	.273	.272	.394	.182	.489	.086
BH1	200+	.353	.476	.432	.401	.513	.244	.589	.111
	100+	.278	.346	.341	.280	.459	.146	.544	.080
	50+	.223	.309	.263	.291	.421	.205	.556	.102
BH0	200+	.325	.350	.378	.259	.447	.177	.506	.054
	100+	.229	.251	.301	.194	.377	.089	.526	.026
	50+	.190	.242	.203	.179	.341	.112	.462	.056
BH1	200+	.310	.367	.364	.307	.492	.154	.516	.082
	100+	.211	.280	.253	.224	.376	.083	.349	.019
	50+	.137	.190	.204	.188	.304	.108	.418	.052

***Table 43*: Micro-averaged precision and recall for DigiTrad classes grouped by amount of training data for the BH0 and BH1 representations.**

		Maximum F-measures			**BP**
		β=0.5	β=1.0	β=2.0	
BW	100+	.863	.861	.885	**.854**
	50+	.692	.706	.748	**.678**
	25+	.767	.770	.807	**.767**
	10+	.677	.647	.667	**.645**
BSW	100+	.867	.853	.874	**.851**
	50+	.626	.639	.657	**.627**
	25+	.785	.750	.786	**.748**
	10+	.618	.571	.577	**.572**
BH0	100+	.814	.780	.773	**.783**^{e}
	50+	.657	.648	.656	**.644**
	25+	.710	.680	.688	**.672**
	10+	.451	.383	.349	**.384**^{e}
BH1	100+	.810	.778	.781	**.777**
	50+	.646	.599	.618	**.589**
	25+	.700	.683	.697	**.681**
	10+	.556	.433	.360	**.359**^{e}

***Table 44*: Maximum F-measures and breakeven points (BP) obtained for Reuters-21578 classes grouped by amount of training data.**

		Maximum F-measures			**BP**
		β=0.5	β=1.0	β=2.0	
BW	200+	.435	.419	.442	**.420**
	100+	.330	.294	.301	**.291**
	50+	.320	.272	.293	**.273**
BSW	200+	.425	.416	.445	**.416**
	100+	.326	.308	.330	**.311**
	50+	.348	.276	.287	**.281**
BH0	200+	.346	.337	.345	**.334**
	100+	.271	.239	.246	**.241**
	50+	.242	.213	.229	**.199**
BH1	200+	.351	.336	.354	**.337**
	100+	.247	.241	.263	**.241**
	50+	.223	.196	.191	**.189**

***Table 45*: Maximum F-measures and breakeven points (BP) obtained for DigiTrad classes grouped by amount of training data.**

To complete the analysis of results, complexity measurements for the hypotheses are presented below. Table 46 shows the average number of rules per hypothesis, and the average number of clauses per rule for each hypothesis. As with the phrase-based representations, the hypotheses were again less complex in the new representations, but like the keyphrase hypotheses formed for DigiTrad, the result is not particularly interesting since the lower complexity rules also produce lower classification power.

		L.R.	$F_{1.0}$	Rules	Clauses	Clauses / Rule
Reuters-21578	BW	1.0	.825	5.1	13.0	2.5
	BSW	1.0	.811	5.6	13.5	2.4
	BH0	1.0	.739	4.0	11.0	2.8
	BH1	1.0	.735	3.7	11.4	3.1
DigiTrad	BW	0.5	.359	12.9	30.3	2.3
	BSW	0.5	.360	13.3	33.0	2.5
	BH0	0.5	.286	10.1	26.3	2.6
	BH1	0.5	.281	11.6	34.9	3.0

***Table 46*: Complexity statistics for each representation. The column *L.R.* shows the loss ratio setting for the rule set, and *$F_{1.0}$* shows the micro-averaged $F_{1.0}$-measure achieved with the rules. *Rules* indicates the average number of rules per hypothesis. *Clauses* indicates the average number of clauses per hypothesis, and *Clauses/Rule* indicates the average number of clauses in each rule.**

7.3.4 Why didn't it work?

Reasons for the poor performance of hypernyms with Reuters-21578 have already been suggested in previous sections: bag of words does so well that it is very difficult to improve on the representation, and the domain generally seems to be more amenable to highly specific rather than general features. Moving to the noun phrase representation, which included some more specific features, produced a slight improvement, but moving to stemmed representations with more general features produced a slight degradation in performance. In this context, moving to an even more general representation using hypernyms should not have been expected to work well.

The poor result requires some more explaining for DigiTrad, a domain with a wide vocabulary which should have benefited from the imported knowledge of synonymy and hypernymy. More specifically, why did the representation do well in the preliminary work and then poorly in the extended experiments? One possible reason is that we simply got lucky in the smaller experiments and happened to pick classes for which the method worked well. A second possibility is that the task of separating one class from another was easier than the task of separating one class from all the rest. To test this idea, two sets of 50 folk songs were categorized by hand using the 10 most frequent classes, and the result was lower than expected – with an $F_{1.0}$-measure of approximately 0.60 on each run. Comparing this to the result reported earlier of 1% and 4% error on the SONG1 and SONG2 data, it seems that the latter is a much easier task. Finally, it is possible that the use of the density measurement rather than binary features, added a significant amount of information. If so this suggests some future work in finding ways to make hypernym density tractable for larger classification tasks.

7.4 Summary

This chapter discussed a method of exploiting semantic knowledge to form hypernym-based representations. In the preliminary work the hypernym density representation was tested on some small tasks with good results. Then the bag of hypernyms representation was applied to the larger tasks of the ModApte and DT100 splits of Reuters-21578 and DigiTrad, with considerably poorer results. This was probably due to a combination of factors, but the possibility cannot be ruled out that the main culprit was the loss of information in going from hypernym density to bag of hypernyms. Evidence of better rule comprehensibility was seen in both the preliminary and extended work, but the poorer quantitative performance on the larger tasks overshadows this result.

7.5 Recommendations

Use of the bag of hypernyms representation is not recommended for symbolic learning of text classification tasks, although hypernym density may be worth trying for non-technical texts if issues of tractability can be overcome.

Part 3: Conclusions and Future Work

8. Final Conclusions and Future Directions

In an article on the current state of information retrieval research for the January 1996 issue of C*ommunications of the ACM,* David Lewis and Karen Sparck-Jones write that statistical techniques have “apparently picked some of the low-hanging fruit off the tree” [LES96]. What they mean is that despite recent advances in Natural Language Processing systems, statistical word-based indexing techniques still produce a higher level of performance on retrieval tasks. Linguistic processing techniques should eventually improve performance, but for now the higher fruit is out of reach. The hope in this thesis was that symbolic learning combined with some shallow linguistic processing could provide a stepping stone to reach the next piece of fruit. Where phrase-based representations would confuse a statistical learner, a symbolic learner would find the information it needed. The problems of word sense disambiguation that had prevented the effectiveness of WordNet for information retrieval would be sidestepped in the presence of a large corpus of training data. Unfortunately, most of the evidence suggests that this is not the case. The higher fruit remains out of reach.

This is not to say that nothing of value came out of this study. We found evidence that in technical domains it is possible to use noun phrase representations to produce equally powerful, yet more comprehensible, or at least less complex, hypotheses. In non-technical domains, it seems that semantic knowledge can sometimes be exploited to improve classification and also produce more comprehensible rules although this has yet to be demonstrated on a large corpus. Perhaps most importantly, there is still room for future research. One door is now closed: it is probably not worth pursuing phrase-based representations on their own any further. Another is still partly open: it may be worth the effort, for non-technical domains, to find a way to make hypernym density more tractable for RIPPER. In confirmation of Lewis and Sparck-Jones’ sentiments, this latter opportunity will probably require more sophisticated use, and perhaps even further development of existing techniques for Natural Language Processing.

On that note, here are a number of possible directions for future research. On the issue of how to make hypernym density representations tractable for RIPPER, one possibility is to use feature selection techniques to reduce the number of hypernym density features. Another, perhaps more promising possibility is to continue to use bag of hypernyms but perform some fast partial word sense disambiguation as part of the pre-processing. This latter approach may be able to make up some of the deficiencies of the bag of hypernyms approach and at the same time speed up processing by making the feature vectors more sparse.

Another area of interest is the investigation of the proposed feature engineering techniques with other learning algorithms. Most of the techniques described in this paper were conceived in the context of symbolic learning and may not work well for statistical learners, but for completeness it is worth trying other algorithms such as naïve Bayes to see if any benefit can be found. This is particularly true for the hypernym density representation which, while intractable for RIPPER, should not pose a problem for naïve Bayes’ simple counting algorithm.

Future work could also be done on evaluation methods. One of the challenging parts of this work was trying to see through the smoothing effects of the F-measure and breakeven point to find out what kinds of subtle effects the changes of representation might be producing. Varying the loss ratio parameter, for instance, had a slightly different effect on each class in each representation, but these effects tended to be glossed over in the aggregate statistics. Is there a way to reliably measure performance that will reveal some of the more subtle differences between representations?

Finally there is work to be done in applying some new developments in machine learning to text classification. Combinations or ensembles of classifiers have been getting more attention recently. The idea behind Boosting is to combine classifiers trained on different subsets of the training data [SCH90]. After the first classifier is trained, a new training set is formed consisting of a higher proportion of misclassified examples, and a new classifier is trained. This is repeated for a number of iterations and then the classifiers are combined, often with very good results.

An even simpler method of combining classifiers takes advantage of the following basic observation: if a number of different high-quality classifiers can be formed which make uncorrelated errors, then performance can be improved by combining them with a simple majority vote [DIE97]. As an example imagine a set of 3 uncorrelated classifiers, each with a 80% accuracy. If they are combined with a simple majority voting scheme, the expected accuracy is $(0.8^2 * 0.2) * 3 + (0.8^3) \approx \underline{0.90}$. As the number and quality of classifiers increases, the expected performance rises.

While the issue of how to train different classifiers for text classification requires some thought, a small last-minute experiment demonstrates the possibilities of combining classifiers trained in different feature spaces. RIPPER's predictions for three representations (BN, BK, and BH0) were combined with a simple majority vote and the micro-averaged breakeven point was computed. The results were very encouraging. The new breakeven for DigiTrad beat all the other individual methods, and the new breakeven for Reuters-21578 beat all previously published results except for Joachims' combined support vector machines (compare the results in Table 47 below to the previously published results in Table 15.) Indeed Joachims' result was also obtained using a combination of support vector machines in which the classifier with the lowest VC-dimension was selected for each class. At the present time, research into techniques for combining feature engineering with multiple classifiers probably holds the most promise for the future.

	Previous Best	Combined Classifier
Reuters-21578	0.827 (BN)	0.840
DigiTrad	0.360 (BSW)	0.369[e]

***Table 47*: Results of a preliminary experiment in combining classifiers.**

Appendices and References

Appendix A: Documentation for Users of the FX system

(Produced as part of contract CS7-247145-00 for the National Research Council. FX is freely available for research purposes along with the most recent version of this help file at http://ai.iit.nrc.ca/II_public/index.html in the "demos" section.)

FX.EXE help file (Release 1.0)

- Written by Sam Scott, IMRL, IIG, IIT, NRC, May 29, 1998.

Overview: FX.EXE is a command-line Win95 application that takes a text corpus marked up with SGML tags and extracts from it documents that match a certain criteria. These documents are then processed and turned into feature vectors for machine learning using one of a number of different methods. The output files are in the format required by the RIPPER learning system found at:

http://www.research.att.com/~wcohen/ripperd.html

Usage: fx <text corpus> <output file stem>

Example:

The command "fx reuters.corp output" will read the "reuters.corp" file and produce "output.names", "output.data", and possibly an intermediate file "output.feat".

Detailed Instructions:

FX requires the user to answer a series of questions. These questions are dealt with in the order that they appear.

```
What format is the input data in?
1. Reuters-21578,
2. Wall Street Journal,
3. Usenet (WSJ Format).
```

1. The Reuters-21578 SGML format can be found at: http://www.research.att.com/~lewis/reuters21578.html (The TOPICS tags will be used as class names. All non-SGML text found between the <TEXT> and </TEXT> tags is taken as the body of the document.)

2. The Wall Street Journal format can be found at: http://www.ldc.upenn.edu/ and requires a license.

3. Usenet simply assumes the Wall Street Journal SGML format.
   - The newsgroup names (classes) should have non-alpha characters removed and be delimited like this: <IN>(sochistory)</IN>

- The subject line should be delimited with <LP> </LP>
- The body should be delimited with <TEXT> </TEXT>
- HTML and other tags can be left in the documents, but </TEXT> tags should be removed
- Bitmaps and other non-text objects should be removed

```
Search by (1)name or (2)number of topics or (3) both:
```

1. When searching by name, the user is prompted for a list of class names to be included in the output files.
   - The user also has the option of using an "other" class. This will randomly select documents that do not match the criteria to be included with the class label "_other". The user may specify what proportion of these documents will be included.
2. When searching by number of topics (classes), the user specifies a maximum number of topics (classes) a document can have to be included in the output. Entering 0 causes all documents to be included regardless of the number of topics.
3. When searching by both parameters, the user may enter a list of desired class names and then also specify the maximum number of class names. For instance, entering "cocoa" and "coffee" with maximum = 1 will cause all "cocoa" and "coffee" documents to be included except those which are tagged with both class names.

- NOTE: since RIPPER requires a single class, if a document contains more than one class label, these labels will be concatenated to form a new class label in the output file (e.g. "cocoa_coffee")

```
What kind of features do you want?
 1. Just the words,
 2. Just key phrases (from Extractor),
 3. Words and key phrases,
 4. Word presence (binary vector),
 5. Absolute word frequency,
 6. Relative word frequency,
 7. Key phrase presence,
 8. Absolute key phrase frequency,
 9. Relative key phrase frequency,
10. No features - select to new SGML files,
11. Prepare data for the Brill tagger,
12. Produce WordNet batch file from tagged input,
13. Just the SynSets,
14. Absolute SynSet frequency,
15. Relative SynSet frequency,
16. Noun phrase presence (binary vector),
17. Absolute noun phrase frequency,
18. Relative noun phrase frequency,
19. Counts.
```

1. **"Just the words"** produces RIPPER-style set valued features containing all words in the document. All punctuation, and non-alpha characters are removed from the words first. All words are forced to lower-case.
   - Option: screen out all words except nouns and verbs. (The corpus must be tagged using the Brill Tagger found at: http://www.cs.jhu.edu/~brill/)
   - Option: stem the words with the Lovins stemming algorithm.

2. **"Just key phrases"** produces RIPPER-style set valued features containing all the "key phrases" returned by Extractor with default settings. All case information is removed from the phrases. For more info see http://ai.iit.nrc.ca/II_public/extractor.html

3. **"Words and key phrases"** produces RIPPER-style set valued features containing the original set of words plus the Extractor key phrases (see 2).
   - Option: use only nouns and verbs (see 1)
   - Option: stem the words (see 1)

4. **"Word presence"** produces a binary feature for every word instead of using set-valued features.
   - Option: use only nouns and verbs (see 1)
   - Option: stem the words (see 1)
   - Option: use Zipf's law to screen out infrequent and frequent words. (See http://www.dcs.gla.ac.uk/Keith/Chapter.2/Ch.2.html)

5. **"Absolute word frequency"** produces integer features for each word. The integer represents the number of times the word appears in the document.
   - Option: use only nouns and verbs (see 1)
   - Option: stem the words (see 1)
   - Option: use Zipf's law (see 4)

6. **"Relative word frequency"** produces real-valued features for each word. The value represents the number of times the word appears in the document divided by the number of words in the document. (The final number is also multiplied by a constant equal to the average number of words per document in the corpus. This avoids the "very small number" problem.)
   - Option: use only nouns and verbs (see 1)
   - Option: stem the words (see 1)
   - Option: use Zipf's law (see 4)

7. **"Key phrase presence"** produces a binary features representing key phrases from the Extractor (see 2). However there is an important difference from 2 above. First a global list of key phrases is assembled by running Extractor on every document, then this total list is used to form the feature set and each document is searched for every key phrase.
   - Option: stem the words (see 1)

- Option: maximum number of key phrases. This parameter limits the number of key phrases taken from each document. The key phrases are taken in order of confidence. If all key phrases are desired, use a high number like 1000 (no document ever has more than 10).

8. **"Absolute key phrase frequency"** produces integer features instead of the binary features produced in 7. See 5 for more details.
   - Option: stem the words (see 1)
   - Option: maximum number of key phrases (see 7)

9. **"Relative key phrase frequency"** produces real-valued features instead of the binary features produced in 7. See 6 for more details.
   - Option: stem the words (see 1)
   - Option: maximum number of key phrases (see 7)

10. **"No features"** can be used if you just want to filter the documents from the original corpus to keep only those with certain topics. The documents are put in new files with no change except that superfluous SGML headings are removed.
    - Option: number of output files. Up to 5 files can be generated that will contain the extracted documents in a "round-robin" fashion. The files are given the extension "sgm0" to "sgm4".
    - Option: use Lewis' split (Reuters-21578 only). This allows you to select only testing or training documents using the criteria described in http://www.research.att.com/~lewis/reuters21578/README.txt

11. **"Prepare data for the Brill Tagger"** separates punctuation from words and puts the documents in a file with the "corp" extension.
    - Option: preserve line feeds. If you answer "no", it will remove existing line breaks and insert new line breaks after every '.' Character as described in the Brill Tagger documentation (see 1).

12. **"Produce WordNet batch file"** produces a batch file that can be used to look up every word in the documents in WordNet. For information on WordNet see http://www.cogsci.princeton.edu/~wn/
    - Options: use synonyms, hypernyms, or hyponyms.
    - NOTE: corpus must be tagged with Brill tagger (see 1)

13. **"Just the synsets"** produces a set-valued feature containing every synset that appears in the documents.
    - Options: use synonyms, hypernyms, or hyponyms.
    - Options: use Zipf's law (see 4)
    - NOTE: In order to use this, you must first run with option 13, execute the batch file produced, then run WNPREPARE.EXE to produce a ".wnf" or ".wef" file

14. **"Absolute synset frequency"** produces integer features as in 5.
    - Options: use synonyms, hypernyms, or hyponyms

- Options: use Zipf's law (see 4)
- NOTE: (see 13)

15. **"Relative synset frequency"** produces real-valued features as in 6.
    - Options: use synonyms, hypernyms, or hyponyms
    - Options: use Zipf's law (see 4)
    - NOTE: (see 13)

16. **"Noun phrase presence"** produce binary features as in 4, but each feature represents a noun phrase. The definition of a noun phrase is a non-empty sequence of adjectives and nouns terminating in a noun. All possible noun phrases are generated, so that the phrase "quick brown fox" results in the noun phrases "fox", "quick fox", and "quick brown fox".
    - Option: stem the words (see 1)
    - Option: use Zipf's law (see 4)
    - NOTE: corpus must be tagged with Brill tagger (see 1)

17. **"Absolute noun phrase frequency"** produces integer features as in 5, but using noun phrases as in 16.
    - Option: stem the words (see 1)
    - Option: use Zipf's law (see 4)
    - NOTE: corpus must be tagged with Brill tagger (see 1)

18. **"Relative noun phrase frequency"** produces real-valued features as in 6, but using noun phrases as in 16.
    - Option: stem the words (see 1)
    - Option: use Zipf's law (see 4)
    - NOTE: corpus must be tagged with Brill tagger (see 1)

19. **"Counts"** produces no output but counts all occurrences of every class label and prints a report at the end.

References

[ADW94] Chidanand Apte, Fred Damerau and Sholom M. Weiss. Toward Language Independent Automated Learning of Text Categorization. In *Proc. SIGIR-94*. 1994. 23-30.

[BRI92] Eric Brill. A simple rule-based part of speech tagger. In *Proc. of the Third Conference on Applied Natural Language Processing*. ACL. 1992. 152-155.

[BRI94] Eric Brill. Some Advances in Rule-Based Part of Speech Tagging. In *Proc. AAAI-94*. 1994. 722-727.

[COH95a] William W. Cohen. Fast Effective Rule Induction. In *Proc. ICML-95*. 1995. 115-123.

[COH95b] William W. Cohen. Learning to Classify English Text with ILP Methods. In *Proc. 5th Int. Workshop on Inductive Logic Programming.* July 31, 1995. 3-24.

[COH95c] William W. Cohen. Text Categorization and Relational Learning. In *Proc. ICML-95*. 1995. 124-132.

[COH96a] William W. Cohen. Learning Rules that Classify E-Mail. In *Proc. of the 1996 AAAI Spring Symposium on Machine Learning in Information Access*. 1996. (http://www.parc.xerox.com/istl/projects/mlia/)

[COH96b] William W. Cohen. Learning Trees and Rules with Set-valued Features. In *Proc. AAAI-96*. 1996. 709-716.

[COS96] William W. Cohen and Yoram Singer. Context-Sensitive Learning Methods for Text Categorization. In *Proc. SIGIR-96.* 1996. 307-316.

[DIE97] Thomas G. Dietterich. Machine Learning Research: Four Current Directions. *Artificial Intelligence Magazine*. Winter 1997. 97-136.

[FAP97] Tom Fawcett and Foster Provost. Adaptive Fraud Detection. In *Data Mining and Knowledge Discovery* 1-28. 1997.

[FEL98] Christiane Fellbaum (ed.). *WordNet: An Electronic Lexical Database*. MIT Press. 1998.

[FUW94] Johannes Furnkranz and Gerhard Widmer. Incremental Reduced Error Pruning. *Proc. ICML-94.* 1994. 70-77.

[FLA98] Sharon Flank. A Layered Approach to NLP-Based Information Retrieval. In *Proc. COLING-ACL'98*. August, 1998. 397-403.

[GRE96] Dick Greenhaus. *About the Digital Tradition*. www.mudcat.org/DigiTrad-blurb.html. 1996.

[HAW90] Philip J. Hayes and Steven P. Weinstein. CONSTRUE: A System for Content-Based Indexing of a Database of News Stories. In *Proc. Second Annual Conference on Innovative Applications of Artificial Intelligence*. 1990. 1-5.

[HEA98] Marti A. Hearst. Support Vector Machines. In *IEEE Intelligent Systems*. July/August 1998. 18-28.

[JOA96] Thorsten Joachims. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*. Technical Report CMU-CS-96-118. Carnegie Mellon University. 1996. (Summarized in [MIT97].)

[JOA97] Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *Proc. ICML-97*. 1997. 143-146.

[JOA98] Thorsten Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. In *Proc. ECML-98*. 1998. 137-142.

[KHM98] Miroslav Kubat, Robert C. Holte and Stan Matwin. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. In *Machine Learning* vol. 30. February 1998. 195-216.

[KOS96] D. Koller and M. Sahami. Hierarchically Classifying Documents Using Very Few Words. In *Proc. ICML-97*. 1997. 170-176.

[KUF67] Henry Kucera and W. Nelson Francis. *Computational Analysis of Present-day American English*. Brown University Press. 1967

[LAN95] K. Lang. *NewsWeeder: Learning to Filter Netnews*. In *Proc. ICML-95*. 1995. 331-339.

[LES96]	David D. Lewis and Karen Sparck Jones. Natural Language Processing for Information Retrieval. In *Communications of the ACM* 39(1). January, 1996. 92-101.
[LEW92a]	David D. Lewis. An Evaluation of Phrasal and Clustered Representations on a Text Categorization Task. In *Proc. SIGIR-92*. 37-50.
[LEW92b]	David D. Lewis. *Representation and Learning in Information Retrieval*. Ph.D. thesis, University of Massachusetts at Amherst. Technical Report 91-93. February, 1992.
[LEW92c]	David D. Lewis. Text Representation for Intelligent Text Retrieval: A Classification-Oriented View. In Paul S. Jacobs (Ed). *Text-Based Intelligent Systems*. Lawrence Erlbaum Associates. 1992. 179-198.
[LEW95]	David D. Lewis. Evaluating and Optimizing Autonomous Text Classification Systems. In *Proc. SIGIR-95*. 1995. 246-254.
[LEW98]	David D. Lewis. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In *Proc. ECML-98*. 1998. 4-15.
[LSM94]	Xiaobin Li, Stan Szpakowicz and Stan Matwin. A WordNet-based Algorithm for Word Sense Disambiguation. In *Machine Learning*. 1994. 1368-1374.
[LOV68]	J.B. Lovins. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics* 11. 1968. 22-31.
[MIL90]	George A. Miller. WordNet: an On-line Lexical Database. In *International Journal of Lexicography* 3(4). 1990. 235-244.
[MIM96]	Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press. 1996.
[MIT97]	Tom Mitchell. *Machine Learning*. McGraw Hill. 1997

[MSM93] M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics*. 1993. (Go to http://www.cis.upenn.edu/~treebank/ and click on "overview".)

[NGL97] Hwee Tou Ng, Wei Booh Goh and Kok Leong Low. Feature Selection, Perceptron Learning, and a Usability Case Study for Text Categorization. In *Proc. SIGIR-97*. 1997. 67-73.

[POR80] M. Porter. An Algorithm for Suffix Stripping. *Program (Automated Library and Information Systems)*, 14(3). 1980. 130-137.

[PRF97] Foster Provost and Tom Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *Proc. of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*. 1997. (http://www.croftj.net/~fawcett/ROCCH/)

[QUI90] J. R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning 5:3*. August, 1990. 236-266.

[QUR89] J. Ross Quinlan and Ronald L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation 80:3*. March, 1989. 227-248.

[RIJ79] C. J. van Rijsbergen. *Information Retrieval*, second edition. London: Butterworths. 1979.

[RIL95] Ellen Riloff. Little Words Can Make a Big Difference for Text Classification. In *Proc. SIGIR-95*. 1995. 130-136.

[RGD97] Manuel de Buenaga Rodríguez, José María Gômez-Hidalgo and Belén Díaz-Agudo. Using WordNet to Complement Training Information in Text Categorization. In *Proc. RANLP-97*. March 25-27, 1997. 150-157.

[RMP86] David E. Rumelhart, James L. McLelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1. MIT Press. 1986.

[ROC71] J. Rocchio. Relevance Feedback in Information Retrieval. In: G. Salton (ed.). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall Inc. 1971. 313-323.

[SCH90] Robert E. Schapire. The Strength of Weak Learnability. In *Machine Learning* 5(2). June 1990. 197-227.

[SCM98a] Sam Scott and Stan Matwin. Text Classification Using WordNet Hypernyms. In *Usage of WordNet in Natural Language Processing Systems: Proceedings of the Workshop* (COLING-ACL'98). August, 1998. 45-51.

[SCM98b] Sam Scott and Stan Matwin. Using Lexical Knowledge for Text Classification. Available as *University of Ottawa Department of Computer Science Technical Report TR-98-03*. 1998.

[TUR98] Peter Turney. Learning to Extract Keyphrases from Text: Empirical Evaluation of a Hybrid Genetic Algorithm. Submitted to *Journal of Artificial Intelligence Research*. 1998. (E-mail the author at Peter.Turney@iit.nrc.ca for a copy.)

[VOO93] Ellen M. Voorhees. Using WordNet to Disambiguate Word Senses for Text Retrieval. In *Proc. SIGIR-93*. 1993. 171-180.

[VOO94] Ellen M. Voorhees. Query Expansion Using Lexical-Semantic Relations. In *Proc. SIGIR-94*. 1994. 61-69.

[WKB96] Scott A. Weiss, Simon Kasif, and Eric Brill. Text Classification in USENET Newsgroups: A Progress Report. In *Proc. of the 1996 AAAI Spring Symposium on Machine Learning in Information Access*. 1996. (http://www.parc.xerox.com/istl/projects/mlia/)

[WHI89] D. Whitley. The GENITOR Algorithm and Selective Pressure. In *Proc. ICGA-89*. 1989. 116-121.

[YAP97] Yiming Yang and Jan O. Pederson. A Comparative Study on Feature Selection in Text Categorization. In *Proc. ICML-97*. 1997. 412-420.

[YAR95] David Yarowski. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proc. of the 33rd Meeting of the ACL*. June 26-30, 1995. 189-196.