AN ABSTRACT OF THE THESIS OF

Ray Liere for the degree of Doctor of Philosophy in Computer Science presented on 7 December 1999 . Title: Active Learning with Committees: An Approach To Efficient Learning in Text Categorization Using Linear Threshold Algorithms

Abstract approved: ____

Prasad Tadepalli

We developed and investigated machine learning methods that require minimal preprocessing of the input data, use few training examples, run fast, and still obtain high levels of accuracy.

Most approaches to designing machine learning programs are based on the supervised learning paradigm – training examples are chosen randomly and given to the learner. We explore the "active learning" paradigm – the learner automatically selects the more informative training examples. Our domain of interest is text categorization, but most of the methods developed are quite general.

The purpose of text categorization is to assign each document in a collection to appropriate categories. Most existing text categorization methods require large amounts of time to prepare the documents for learning and large numbers of examples for training. Humans must assign correct categories to documents before they can be used for training; this costs time and money. Our goal is to develop machine learning methods that, when compared to other methods currently

available, are more efficient in time and space, use fewer training documents, and are as accurate.

We developed the Active Learning with Committees (ALC) framework – inspired by the Query by Committee approach of Freund, Seung, et al. A "committee" is a group of learners that jointly participate in learning and in predicting the classes of new examples. We perform minimal preprocessing of the documents and thus the domain is noisy, high dimensional, and has large numbers of irrelevant attributes. We use linear threshold learning algorithms to obtain computational efficiency with respect to these large numbers of attributes, with specific algorithms being chosen because they also generalize well when large numbers of attributes are irrelevant.

We developed and analyzed several ALC systems. Our results show that it is possible to design active learning systems that scale up to large numbers of features and obtain accuracies comparable to the supervised learning methods while using an order of magnitude fewer examples and an order of magnitude less time. The ALC methods developed have run times on the order of seconds, typically use only 5 - 7% of the training documents, and are as accurate as their supervised counterparts.

Active Learning with Committees: An Approach To
Efficient Learning in Text Categorization
Using Linear Threshold Algorithms

by

Ray Liere

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented 7 December 1999
Commencement June 2000

Doctor of Philosophy thesis of Ray Liere presented on 7 December 1999

APPROVED:

____

Major Professor, representing Computer Science

____

Head of Department of Computer Science

____

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

____

Ray Liere, Author

Acknowledgements

My thanks . . .

To the members of my doctoral committee for serving on my committee – Prof. Prasad Tadepalli, Prof. Curtis Cook, Prof. Toshimi Minoura, Dr. William Hersh, Prof. Gerald Smith, and Prof. Robert Kiekel. All have shown special patience as I requested permission to do things in non-standard ways, and as I emailed them repeated optimistic schedules.

To Prasad Tadepalli, my major professor. Many professors are not interested in guiding doctoral students unless those students' interests mesh with the specialization areas of the professor. However, Prof. Tadepalli gave me the intellectual freedom to pursue my own particular area of interest, and enthusiastically supported my desire to do research in an area that is not one of the specialization areas at OSU. He looked upon it as an opportunity to learn new things rather than as an obligation that was going to cost him a lot of time. But I know that it did cost him a lot of time, and that he had to make many sacrifices as a result. I appreciate his patience, his good counsel, his honesty, and his friendship. I also thank him for the financial support that I received through him in the form of research assistantships and scholarships.

To Curt Cook. A great deal of my initial impression of OSU was due to Prof. Cook, one of my first professors here. A very caring person, and genuinely interested that I learned the material and thoroughly understood it. His clear manner of teaching helped me a great deal, not only to learn the materials in his course, but to build a firm foundation for what was to follow.

To Toshi Minoura, for allowing me in his database course to investigate an area that was of interest to me, even though it was "a bit out of the ordinary". This opportunity and others to follow are what led me to do research in the area that I have chosen. May the fish always be biting.

To Bill Hersh, for his words of encouragement, his practical knowledge of information retrieval and text categorization and his willingness to share it so freely, and for personally introducing me to so many other members of the information retrieval community. It made such a big difference to me to actually meet so many of the big names in the field – it not only allowed me to get to know them personally, but also made it much easier to later exchange ideas with them.

To Bob Kiekel, for his unbounded enthusiasm for crossing over so many "academic discipline" lines (linguistics, computer science, statistics). Even though not in my department, he always made me feel welcome, encouraging me to "drop in and chat" any time. He provided me with a great deal of guidance in the areas of natural language and linguistics, and many pointers to good books and papers.

To the administrators of Oregon State University and the Department of Computer Science, for the way that I have been treated, even though I am certainly not the typical graduate student in terms of age or personal responsibilities. I was never made to feel that I did not belong at OSU, and I am very fortunate to have enrolled here for my doctoral program. When my wife Helen was very seriously injured in an automobile accident, the University and the Department showed great compassion. So many people at OSU certainly did not act like the stereotyped "big university bureaucrats" when Helen was hurt. People were very sympathetic to the situation and went the extra mile and then some to minimize the impact my sudden departure had on my education. My special thanks to Prof. Walter Rudd for the many memos and phone calls to other departments at OSU to cut red tape, thereby allowing me instant emergency leave, and then a repeat of those efforts to smooth the process of my return.

To Prof. Thomas Dietterich, Prof. Bruce D'Ambrosio, and Prof. Prasad Tadepalli, for all that I learned in your AI classes. They are all well-known and respected scholars in their fields, and I am very fortunate to have been given the opportunity to learn from them.

To my fellow computer science graduate students at OSU – they were wonderful at helping an "older than average student" blend in to student life. We had many good talks together, helping each other through the difficulties encountered by anyone optimistic enough to think that he/she can persuade an assemblage of silicon and wires to perform complex tasks. I never felt excluded from discussions, problem sessions, impromptu gatherings, or any other events, even though I have different experiences and perceptions than most students. Being "away from home" was made much easier by your accepting attitude.

To OSU, for creating a very diverse and accepting atmosphere where you can grow and learn and, within reason, be yourself. My backhanded thanks also to the schools that I spoke to that were not interested in my attending due to concerns about my "not fitting in" at their institutions because of my age, or that wondered "why I was not acting my age". I appreciate your having made your attitudes so very clear from the very beginning – I am sure that I would not have been nurtured in your waters.

To all of the other people in information retrieval and related fields that were also so generous with their time, ideas, resources, and expertise. I was helped a great deal by so many of these people – they did not treat knowledge as a treasure to be hoarded, but rather gave freely of what they had and were happy to do so. I especially want to thank David Lewis, Yiming Yang, Chid Apté, Dan Roth, Naoki Abe, Yoav Freund, Fred Damerau, Tom Mitchell, Thorsten Joachims, Mehran Sahami, Kamal Nigam, Andrew McCallum, Tom Bylander, Ellen Voorhees, Marti Hearst, Efthimis Efthimiadis, Susan Dumais, Michael Kearns, David Evans, David Heckerman, Doug Fisher, Will Taylor, and Greg Cooper.

To NSF. For part of the time that I was doing research on this project, I was supported by the National Science Foundation under grant number IRI-9520243.

To David Lewis. The availability of the Reuters-22173 and Reuters-21578 corpora has greatly assisted in our research.

To Gary Perlman. The |STAT Data Manipulation and Analysis Programs have been very useful to us, and we appreciate them having been made available for our research.

Table of Contents

List of Figures

List of Tables

Dedication

I dedicate this thesis . . .

To Helen, my wife, who has been patient, understanding, and actively supportive of my desire to return to school to earn a doctorate. This support did not waiver, even after all that she personally went through as a result of being in a very very serious automobile accident. She has never ever doubted that I would be successful in completing my doctoral degree.

To our children, Aaron and Karyn, who shouldered more than the normal share of family burdens due to my being away at school during the week. Money was tight with 3 of 4 family members in college, but no complaints. I consider myself to be very lucky. Aaron and Karyn were always there when I needed them, and always shared in my hopes for completing my doctoral degree.

To my parents Agnes, Oscar, and Muriel Liere, for their kind thoughts and kind words, and for bringing me up to be stubborn enough to see this through.

To Helen – Can I Have This Dance?

I'll always remember the song they were playing
The first time we danced and I knew
As we swayed to the music and came to know each other
I fell deeply in love with you

I'll always remember that magic moment
When I first held you close to me
As we moved together I knew right away
That you were all I would ever need

Can I have this dance for the rest of my life
Will you be my partner every single day
When we're together we can conquer come what may
Can I have this dance for the rest of my life

(Based on the song "Could I Have This Dance" – Holyfield/House, Polygram International Publishing, Tree Publishing, Brian Paul Sullivan)

Active Learning with Committees: An Approach To
Efficient Learning in Text Categorization
Using Linear Threshold Algorithms

1. Introduction

In this project, we developed and investigated machine learning methods that require minimal preprocessing of the input data, use few training examples, run fast, and still obtain high levels of accuracy. Our domain of interest is text categorization, but most of the methods developed are quite general and so are applicable to other domains having large collections of data that change often, that are composed of very large numbers of examples, that have high dimensionality, and that contain attribute and class noise.

The purpose of text categorization is to assign each document in a collection to appropriate categories based on the content of the document. Most existing methods of text categorization need large amounts of time to prepare the documents for learning and also need large numbers of examples to analyze in order to train the system. Many collections of documents are now stored digitally in computer files. These collections are often very large, and they also frequently change as documents are added, removed, and updated. It is therefore important to develop methods that automate the text categorization process, and perform it quickly and accurately.

The two general approaches to problem solving are "knowledge engineering" and "machine learning". In the knowledge engineering approach, one designs a program that directly solves the problem of interest. In machine learning, one uses a somewhat indirect approach – one designs a system that learns how to solve the problem of interest. The machine learning approach to text categorization generalizes information contained in example documents in order to build a knowledge base that can then be used to categorize additional documents. Machine learning is a suitable approach for this task since it allows us to construct a system that will be able to adjust itself to handle different categories, different types of documents, and different languages. The use of machine learning in this domain, however, is difficult due to certain characteristics of the domain — the very large number of input features, noise, and the large percentage of features that are irrelevant. As a result, supervised learning often requires extensive preprocessing of the textual data, long run times for large document collections, and a large number of training documents. Training documents need to be categorized by a human, and so needing large numbers of training documents translates into spending large amounts of time and money.

The goal of this research project was to develop machine learning methods for text categorization that are efficient in time and space, that use fewer training documents, and that are accurate. To achieve these goals, we developed the Active Learning with Committees (ALC) framework. "Active learning" refers to any form of learning that controls the examples on which the system is trained [Cohn et al. 1994]. A "committee" is a group of several learners that jointly participate in learning and also in predicting the classes of new examples. The ALC approach is an overall system structure that is used to guide the problem-solving process. An ALC system consists of 3 parts: [1]determining which training documents to use for learning, [2]the actual learning process, and [3]forming the prediction of the committee.

We will now briefly examine the 3 parts of an ALC system:

1. determining which training documents to use for learning

   ALC needs a way of deciding whether or not to use each available training document. This is the "active learning" aspect of the system. There is a stream of candidate training documents coming into the ALC system, but we want the system to use only a few of them for learning. We have developed several methods for deciding which documents to use for learning. They are all based on the notion that if each member of the committee of learners predicts on a document, and if these predictions are evenly or almost evenly split, then that document is probably a good one for the committee to use for learning (since ≈½ of the learners are getting it wrong – those learners do not "understand" that example and so need to learn it).

This aspect of ALC was inspired by the Query by Committee approach [Freund et al. 1992, Seung et al. 1992, Freund et al. 1997]. The Query by Committee (QBC) approach uses a committee to decide which documents to use for training. Briefly (details later) QBC works as follows. We start with a category of interest and a collection of documents. A random subset of the documents is chosen to be used for training. QBC creates a committee that contains all possible hypotheses for categorizing the documents. Each training document is presented to QBC, 2 hypotheses in the committee are chosen at random, and each predicts whether or not the document presented is in the category of interest. If the 2 hypotheses disagree, then the document is assumed to be worthwhile for learning, and QBC requests the actual status of that document – whether it is or is not in the category of interest. A human must provide this information. Once provided, all hypotheses in the committee that are in error are removed from the committee. Thus QBC provides a mechanism whereby the learner automatically selects the most informative training documents. Since the learning process does not use all available training documents, less work needs to be done by the human whose task it is to categorize the training documents.

ALC adapts QBC to domains which contain noise and which also have so many possible candidate hypotheses that it is not practical to represent each of them explicitly.

2. the actual learning process

For learning algorithms that form the committee of learners, we needed to choose methods that are able to handle the very high dimensionality of this domain. We chose 3 algorithms – perceptron [Weiss and Kulikowski 1990], winnow [Littlestone 1989], and naive Bayes [Duda and Hart 1973]. These algorithms all use the same hypothesis space – each algorithm represents the concept being learned as a hyperplane. The object of the learning process is to position this hyperplane in the document space so that it separates the documents that are in the category of interest from those that are not. Once this has been accomplished, prediction on future documents is done by simply determining on which side of the hyperplane the new document is located and then making the corresponding prediction. It turns out that both the learning and the predicting when using these algorithms is quite fast, which was another consideration in choosing them.

Like QBC, ALC does use a committee of several learners. However, in ALC the size of the committee does not change during learning.

3. forming the prediction of the committee.

ALC systems need a way of combining the predictions of the individual committee members into a single prediction of the committee. One common method of prediction is majority voting, with ties being broken by a coin toss.

An example of an ALC system follows. The system consists of a committee of 10 winnow learners. A document is used for training if 5 of the learners predict that the document is in the category of interest and the other 5 predict that it is not. Committee prediction is by majority vote. In a typical use of this ALC system, the user will specify which category is of interest, the system will learn using only a few of the available examples, and then the system will provide predictions to the user on previously unseen documents.

In our ALC systems, we do not preprocess the documents except to divide them up into separate tokens at punctuation and whitespace. More extensive types of preprocessing are normally used by text categorization systems. Methods frequently include techniques such as removing common words ("the", "a", "is"), stemming ("cars" $\rightarrow$ "car"), using only the more informative words in each document, and attribute creation. Such preprocessing usually has as one of its main purposes the reduction of the domain dimensionality so that the learning algorithm of choice can better function in a manner that is acceptable in terms of computational resources and/or in terms of accuracy. Such preprocessing is not only domain-specific to text categorization, but often is also very specific to the types of documents in the corpus. There is not any general theory to support these preprocessing techniques, and in fact each of the various methods seems to be championed by some and denigrated by others. And finally, such preprocessing can take quite large amounts of time. We instead have designed the ALC system so that the choice of which document tokens are in fact relevant is done solely by the learning algorithm itself. This minimal preprocessing approach makes the overall system more general and also as an added benefit saves preprocessing execution time. It also requires that our systems deal directly with a very noisy high dimensional domain having large numbers of irrelevant attributes.

In our ALC systems, we use a document representation based on the vector space model [Salton 1971]. In this model, documents are treated as vectors in an *n* dimensional document space. Each dimension measures some characteristic of the document thought by the system designer to be related to document content. We create a dictionary containing all tokens used in the entire collection of documents. We use as document characteristics the existence or non-existence of tokens in each particular document. Each document is therefore represented by a boolean vector, where the value of each element in the vector indicates whether (1) or not (0) the corresponding token appears in that document. The vectors contain as many elements as there are tokens in the dictionary, with values of 10^4 to 10^5 being typical. The vector is unusually long because of our having done minimal preprocessing.

Since supervised learning systems have access to all of the training documents that are available, they are used as baseline systems – systems against which we compare our efforts. In other words, we would compare the performance of the system described above with a system also composed of 10 learners, also all winnows, but using supervised learning.

This project developed ALC methods for text categorization that:

1. are computationally efficient in both time and space – typical run times for the faster systems are on the order of seconds
2. use only 5 - 7% of the training examples used by corresponding supervised learning methods
3. obtain accuracy and other measures similar to other systems using similar learning algorithms

4. scale up well to full text

In this thesis, we not only developed systems that met our goals, but also analyzed them as to why they behave as they do and investigated several other related areas:

1. We examine why ALC systems can result in faster execution time, the use of fewer examples, and still obtain similar results when compared to their supervised counterparts. ALC takes advantage of two effects – active learning and committee prediction. In active learning, one attempts to select the "more informative" examples and to use only them for learning. If one considers all of the examples available for training, then certainly some are more informative than others. By selecting only a few examples, but ones that are more informative, we place less of a burden on the human that must provide the correct category information for training documents and we also get a system that runs faster. Committee prediction takes advantage of the fact that, if the members make errors independently of one another, then the committee as a whole can be more accurate that its most accurate member. We find that the main contribution to ALC is in the active learning, but that committee prediction is also of benefit.
2. The winnow and perceptron learning algorithms are both amenable to efficient active learning and are also very similar conceptually and structurally. There are some key differences, however, so we compared their performance in our domain. We found that winnow performed better, and we discuss why this may or may not be what one would expect, and what specific characteristics of the two algorithms result in winnow being better in our domain.

3. Many text categorization algorithms are probability-based, in the sense that they categorize documents by computing a probability and comparing it to some threshold value. In such systems, one can essentially get ranked output of documents for free. We investigated ways to obtain ranked output from committees of winnow and perceptron learners and present results that show that they are able to do a good job of ranking.

4. We examine the impact of making several modifications to the learning algorithms we are using.

    a. The standard winnow and perceptron algorithms only learn if their prediction on the training example is in error. For example, if the document actually is in the category but was predicted not to be, then winnow (or perceptron) will learn. We investigate how the behavior of these algorithms changes if we alter the definition of what it means to make an error.

    b. We also investigate the effects of completely removing the restriction of learning only when in error. We allow winnow (or perceptron) to learn from all examples that are selected as informative.

    c. We also examine the impact on learning of trying to use some of the readily-available information on training examples that are not labeled by a human. This is attractive since such examples are in a sense "free", in that no labeling by a human is required.

We find that the standard definition of mistake seems to work best. We also find that our approach did not result in our benefitting from the use of certain information in unlabeled examples. However, we also found that winnow does better if it is allowed to learn from all selected examples rather than learning only when it is in error. We discuss why these conclusions seem to make sense in this domain.

5. We have included the naive Bayes algorithm since it is also a linear threshold learning algorithm. However, the results using this algorithm are mixed. Advantages of the algorithm include fast run times and good performance in spite of the obvious violations of the assumptions on which it is based. Disadvantages of naive Bayes are that it is supervised and so uses all training examples. One is tantalized by its good performance and wants therefore to adapt it to the ALC framework. However, we found that the structure of the algorithm makes it difficult to achieve active learning behavior in a manner that is both general and computationally efficient. We look at some initial results of experiments with a learning algorithm that uses a committee of naive Bayes learners in the ALC framework and which therefore uses fewer examples for training than the standard (supervised) naive Bayes algorithm.
6. We have a host of methods that we tried that did not work out very well. We briefly discuss a few of these that are perhaps of interest to other researchers. Some of the methods did perform well, but were abandoned because others worked slightly better. Other methods have recognizable names and so are ones that one might think ought to do well, but in this domain they do not, so we discuss the reasons why.

2. Text Categorization

In this chapter, we discuss what we mean by text categorization, why one wants to do it, why it is a difficult task, some possible software approaches, why we are using machine learning, and characteristics of the domain that make text categorization difficult for machine learning. We also give a general description of the supervised learning model and discuss several evaluation criteria for supervised learning methods.

2.1 What is it?

The purpose of text categorization is to assign each document to the appropriate categories, based on the content of the document. The list of categories is predetermined. Each document can be in any number of categories, or in none of them.

A stream of documents in a collection enters the text categorizer. When each document exits, it will have been assigned categories that represent the content of the document. The process by which different text categorization systems determine the meaning of the document is a characteristic that is often of interest. The meaning of a document is a result of a great many aspects of natural language – the words used, how they are structured with respect to each other, punctuation, etc. Surprisingly, many text categorization systems (ours included) are able to do quite well without using very much of the information available in natural language text. Our system, for example, uses only the presence or absence of words themselves and does not use any information on word frequency or order.

We will use the terms "text categorizer" and "indexer" synonymously. Both refer to a human or to a mechanized process that assigns content descriptors to documents.

2.2 Why do it?

Predefined categories are usually topic areas of long term interest (as opposed to information retrieval, where the queries are more ad hoc). Text categorization can therefore be used for:

1. routing/filtering documents of interest (such as technical reports, news, email, newsgroup/mailing list postings) based on user interest profiles.
2. indexing documents – assigning controlled vocabulary category names to documents for subsequent text retrieval and/or for library organization; automated indexing is also used as an aid to human text categorizers, especially if the automated method is able to explain/justify its categorizations.
3. natural language processing – reducing an infinite set of possible natural language inputs to a small set of categories is one strategy in common use.

The task of text categorization is difficult for many reasons:

1. Category assignments are based on the meaning of the text. So a highly accurate categorizer has to, at least at some level, understand the document.
2. Natural language is ambiguous – the same sentence can have multiple meanings. For example, "A rabbi married my sister" can mean that [1]my sister was married (to x) at a service performed by a rabbi, or [2]my sister married a rabbi (at a service performed by y).

3. There are lots of synonyms – different words with same/similar meanings. It is good writing style not to repeatedly use the same word for a particular concept, but instead to use synonyms. But this means that the categorizer has to deal with a lot of words having similar meanings, since it is meaning that is used to determine the assigned categories.

4. There are words that are spelled the same but that have different meanings – homonyms. The *bank* of a river, *bank* as a financial institution, a pile (snow *bank*), to *bank* (turn) an aircraft, to *bank* a fire ... The categorizer must, in spite of these all being the "same word", be able to distinguish the different meanings, since meaning is what categorization is based upon.

2.3 Why Use Machine Learning?

The two general approaches to problem solving are "knowledge engineering" and "machine learning". In the knowledge engineering approach, one creates a program which solves the problem of interest directly. A knowledge engineering approach to text categorization would therefore require one to determine a set of rules that correctly categorizes documents. Determining a specific solution method would require the development of a significant new theory in order to design a highly accurate direct solution, and given the effort that has been spent on other already-existing knowledge engineered systems, this would have been a very daunting task. We also wanted to develop a system that would be capable of adjusting itself to handle different topics, different types of documents (newspaper articles, newsgroup postings, technical reports, etc.) and perhaps even be able to perform text categorization of documents written in languages other than English.

The machine learning approach is an indirect approach, in that the system itself learns how to solve the problem of interest. Example documents with category information ("labels") are provided to a learning program. This program then analyses these examples, extracting and generalizing the knowledge they contain and storing this knowledge in a knowledge base. This knowledge base can then be used to solve previously-unseen problems. Often the knowledge base is viewed as a collection of hypotheses, with some ordering imposed on how they are to be utilized for prediction.

We chose to use the machine learning approach. This allows us to avoid the knowledge engineering bottleneck of having to acquire, organize, and resolve large amounts of incomplete and conflicting expert knowledge. Instead we will design a machine learning system that will compute a solution method that is very accurate. Also, using machine learning makes the system very flexible. One can potentially retrain the system with documents on other topics, other types of documents, documents in other languages, or even on problems that deal with the categorization of objects that are not documents (such as performing medical diagnoses, classifying flowers, determining winning chess moves, finding boolean expressions to fit data).

2.4 Supervised Learning

When most people use the term "machine learning", they are often referring to a specific type of machine learning called "supervised learning". In this section we will discuss supervised learning in general and also explain some of the terminology used in machine learning and how specifically we will be applying those concepts to text categorization.

Figure 1 shows the basic model used for supervised learning and how it is applied to the specific domain of text categorization. Raw documents are first preprocessed, to put them into a form which can be used by the machine learning algorithm. Text preprocessing includes breaking the text into tokens, which often

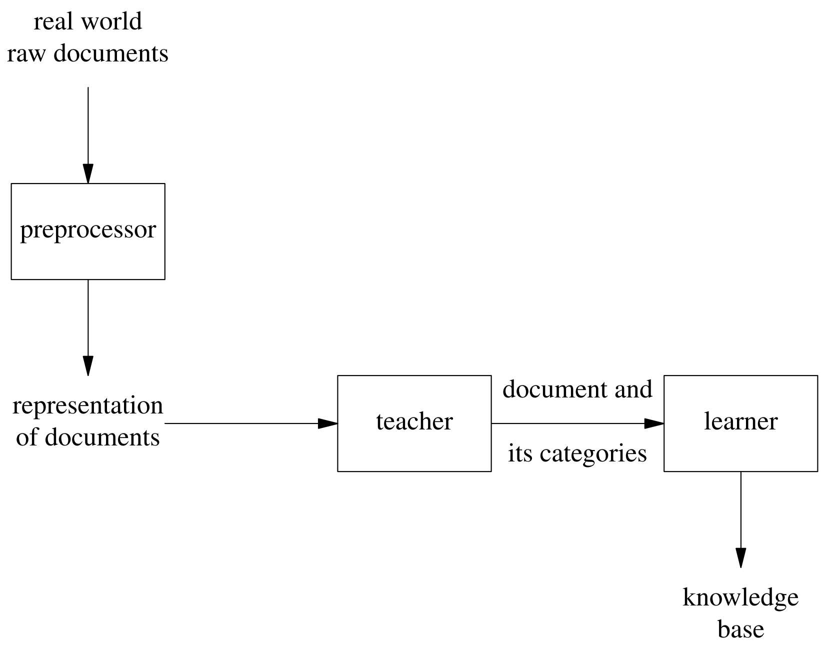

Figure 1. Supervised Learning Model

are loosely call "words". Preprocessing can also include other actions, such as removal or replacement of some words.

Most machine learning systems represent each document as a feature vector, where each position (or "feature" or "attribute") in the vector corresponds to a particular word. For example, the value stored may be the number of times that the corresponding word appears in the document. We store a 0 or 1 in each position in the vector – indicating whether or not that word occurs at all in the document. We do not store any indication of how many times each word was used.

In training mode, the teacher (a human) assigns categories to each training document. A document that is in a particular category is termed a positive example for that category, and a document not in that category is referred to as a negative example. Alternatively, one speaks of a document that is in the category as being "in the class" or "in the yes class", and a document not in the category as "not in the class" or as being "in the no class". This labeling by the teacher is in fact where the term "supervised learning" comes from. The "supervision" is the provision of the label by the teacher for all training examples.

The labeled examples are then presented to the learner. The learner is a program that analyzes the training examples, extracts and generalizes the knowledge in them, and stores the knowledge in a knowledge base.

Some learners actually learn incrementally from each example. That is, they may update their knowledge base each time a labeled example is received. An advantage of incremental learning is that one can stop the learner at any time and it will have developed at least a portion of a knowledge base. Other learners read through all of the examples before doing any computations ("batch learners"). Often this approach is more efficient in time and space.

Some learners may go through the set of examples more than once. The teacher only has to label each example once, but the learner can make as many passes as it wants through the set of training examples. Each pass is referred to as an "epoch". Learners that require several epochs often take more time than single-pass learners, but there can be offsetting advantages to this approach, as we will see later, such as being better able to choose and utilize informative examples.

Once training has been completed, the resulting knowledge base can be used to categorize previously unseen documents – documents that have not been previously labeled by a human or seen by the learner. Figure 2 diagrams this process. The documents first need to be preprocessed to put them into feature vector form. Then the predictor program uses the knowledge base to determine the categories to be assigned to each of these new documents.

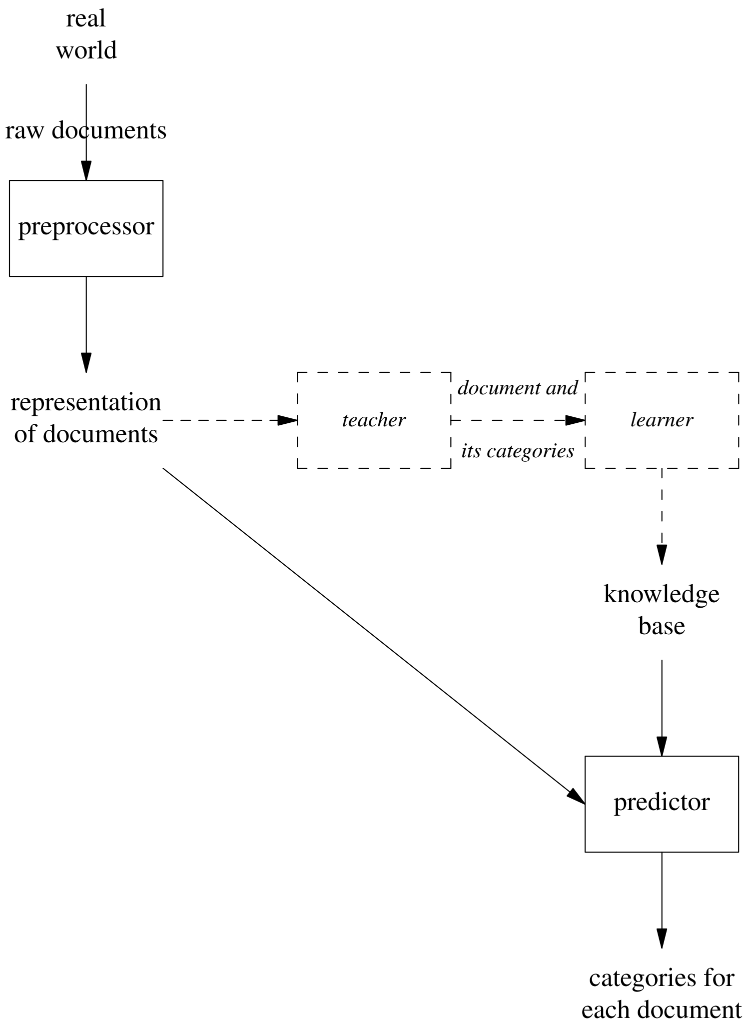

Figure 2. Supervised Prediction Model

Note that the model shown in Figure 2 can also be used to test the ability of the learner to construct an accurate and generalized knowledge base. After learning has been completed, additional documents are given to both the predictor and the teacher. A record is kept of the ability of the predictor to accurately predict – thus giving us an indication of how well the learner learned.

Regardless of the learning algorithm used, since the target concepts are complex, large numbers of labeled examples (usually several thousand) are typically needed in order to obtain good text categorization results with supervised learning. Obtaining large training sets for supervised learning is often not practical or possible. Rarely are documents produced in such a manner as to be already labeled for free. The labeling of examples takes significant human expertise and time, and often those resources are not readily available. This is especially true as collections of documents become larger, more distributed, and more dynamic. This is our motivation for wanting to decrease the number of training documents needed for learning.

2.5 Why Text Categorization is Difficult for Machine Learning

Text categorization is difficult for machine learning methods for several reasons:

1. large number of input features

   Most machine learning methods do not scale up well, meaning that as one increases the number of input features (which in our case are tokens), performance often degrades. This occurs because:

   a. For many learning algorithms, increasing the number of features requires that the algorithm use more training examples in order to obtain the same level of accuracy (sample complexity). If a sufficiently large number of examples are not available, then the algorithm will be unable to extract the knowledge needed in order to be able to predict accurately.

b. Having to process a large amount of training data (large number of features and large number of training examples) results in many algorithms needing much more time and/or space in order to extract the knowledge from those examples (time and space complexity). The resulting performance degradation can result in unacceptably large run times and/or memory demands.

2. many features are irrelevant

Many words are irrelevant, in the sense that a human categorizer does not use them in order to decide what a document is about. However, most machine learning algorithms have difficulties if large numbers of input features are irrelevant. Most algorithms have difficulty determining that these features are in fact irrelevant, and instead attempt to include them in some fashion in the knowledge that they accumulate.

3. attribute and class noise

Attribute noise refers to the fact that some words in a document may not be what they are actually supposed to be. Words may be misspelled, or the wrong words may have been used (for example, "there" instead of "their").

Class noise refers to the fact that the categorization done by humans to label the training examples may contain errors. In other words, some of the training documents may be accompanied by incorrect labels. The learning algorithm assumes that the labels provided by the teacher are correct, and the knowledge base is constructed accordingly. Some learning algorithms are better able to handle labeling errors than others, but a high level of incorrect label values will defeat most learning algorithms.

4. complex target concepts

   The target concepts are difficult to learn since they are complex – i.e., the concepts often require large and/or complicated descriptions to obtain reasonably accurate approximations. In fact, we do not even know the general form of the target concepts and so in general we are trying to learn a good approximation to the target concept as opposed to learning the target concept itself. This distinction is important to note, since we often could benefit by knowing the target concept's form or distribution, in that we could take advantage of theoretical results that have been developed for specific families of functions and/or certain types of distributions. Instead, however, we must choose a reasonable approximating family of functions and a reasonable approximating distribution and use those, relying on empirical results to indicate whether or not our assumptions were in fact reasonable.

2.6 Performance Measures

We need ways to compare various methods of text categorization. How do we tell which method is best? There are several commonly-used performance measures that allow us to compare different methods. It is, however, very difficult to specify one single performance measure for use in all situations, since which performance measure is most important often depends on the characteristics of the document collection and on the needs of the user [Hersh 1996].

Elapsed processor time and the number of training examples used are for us the main performance measures of interest in our research. We want to have a system that runs very fast and that uses very few training examples. However, one obviously needs a fast-running, example-efficient system that provides useful information to the user. We therefore also need to use performance measures that are based on the text categorization results actually provided by each system. Elapsed processor time and number of training examples used are already well-defined concepts. In the remainder of this chapter, we will examine other performance measures that we will use – the performance measures that measure the text categorization capabilities of the various systems. Since the design of our experiments typically involves running large numbers of trials, we will examine various methods of computing average behavior for a system as a function of its performance on a series of a large number of trials. And finally, we will recap which performance measures we will use and their relative importance to us.

2.6.1 Contingency Table Based Performance Measures

Given a collection of documents and a category of interest, some of the documents actually are in the category and some are not. Similarly, a particular text categorization system will predict that some documents are in the category and some are not. We may therefore summarize the performance of a text categorization system using a 2x2 contingency table structured as shown in Table 1. In the text of this thesis, we will use "YES" to mean "document is in the category of interest" and "NO" to mean "document is not in the category of interest". Context will be used to indicate whether we are talking about predicted or actual categorization values.

A comment is in order regarding our use of the word "predict". We use it to mean the process a system goes through in order to classify or categorize a

TABLE 1. 2x2 Contingency Table

		actual label value NO	*actual label value* YES
predicted label value	NO	a	b
	YES	c	d

document that it has in general not previously seen. The idea behind our choice of the word "predict" is that the predicting is done before the system is given the correct label. Unfortunately, "predict" is also used in computational learning theory to refer to a particular type of representation-independent learning. Our use of the word "predict" does not include learning and is synonymous with "classify" or "categorize".

The *a*, *b*, *c*, and *d* values in the contingency table are document counts. For example, *c* is the number of documents that the system predicted were in the category but actually were not. One wants the number of correct predictions (*a* and *d*) to be large, and the number of errors (*b* and *c*) to be small.

Due to terminology used in medical diagnoses [Weiss and Kulikowski 1990], some refer to *b* as the number of false negatives and to *c* as the number of false positives. Similarly, *a* and *d* are referred to as the number of true negatives and true positives, respectively.

Since comparing such tables is difficult, several performance measures based on the contingency table have been developed. These performance measures compute a single value from the 4 values in the table. They are usually structured so that they range over [0.0, 1.0] and so that "bigger is better". The process of transforming 4 values into a single value causes some loss of information, and so there are situations where certain performance measures may be preferred over others.

We use the following performance measures based on the contingency table:

1. **Accuracy**

   Accuracy measures the ability of the system to correctly categorize documents. Accuracy answers the question: are the documents predicted as being YES actually YES and are the documents predicted as being NO actually NO.

   Accuracy is defined as the fraction of documents that are correctly categorized. In terms of contingency table entries, it is computed as:

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

$$\text{if } a+b+c+d=0, \text{ then } accuracy = 0$$

2. **Precision**

   Rather than giving the user a list of documents that are predicted to be in the category of interest and another list of documents that are predicted not to be in the category of interest, most text categorization systems provide the user with only the first list – the list of documents for which it predicts YES. Precision measures the ability of the system to categorize as YES *only* those documents that are actually in the category.

Precision is defined thusly: of the documents that are predicted to be YES, what fraction are actually in the category. It is computed as:

$$precision = \frac{d}{c + d}$$

$$\text{if } c + d = 0, \text{ then } precision = 0$$

3. **Recall**

When the user is interested in the documents in a certain category, the user usually wants most of the documents that are actually in the category to be predicted YES by the categorizer. Recall measures the ability of the system to categorize as YES all documents that are actually in the category.

Recall is defined thusly: of the documents that are actually in the category, what fraction are predicted to be YES. Recall is computed as:

$$recall = \frac{d}{b + d}$$

$$\text{if } b + d = 0, \text{ then } recall = 0$$

4. **$\mathbf{F}_{\beta}$**

In actual practice, text categorization systems exhibit precision-recall tradeoff. That is, for a system that has been tuned for optimal performance, if an adjustment of parameters causes precision to rise, then recall will fall (and visa versa). To see why this occurs, consider a document for which the system is not sure whether to predict YES or NO. If the goal of the system is to get high precision, then it will predict NO since the computation of precision does not include the number of NO predictions, whether correct or not (neither a nor b appears in the equation for precision). The system seeking a high precision rating should only predict as YES those documents that it is very sure of, so that it has a better chance of having a high percentage of its YES predictions being correct.

However, if this same system in the same situation wants to get a high recall value, it should predict YES since the computation of recall does not include the number of actual NO documents, whether correctly predicted or not (neither a nor c appears in the recall computation). The system seeking a high recall rating should predict YES if there is any uncertainty at all about the correct categorization of the document, since it then has a better chance of predicting YES for all documents that are actually YES.

When comparing different systems with different values of precision and recall, one usually has to trade off between the two in order to decide which is the better system. In comparing 2 systems, one will always favor the one having higher precision *and* higher recall. But usually the decision is more difficult – one will normally be faced with having to choose between two systems where one has higher precision and the other has higher recall. This in turn has led to performance measures that compute a single value that incorporates both precision and recall. Usually such measures also include some indication of the relative importance of precision and recall to the user.

We use one such measure, which is called F_β ("eff sub beta") [van Rijsbergen 1979, Lewis and Gale 1994]. F_β is a function of precision, recall, and a value called β. β is defined as the ratio of the importance of recall to the importance of precision. It is determined by the needs of the particular user. β can be any value between 0.0 (ignore recall – only precision is of concern) to ∞ (ignore precision – only recall is of concern).

A user with a low β value is interested in high precision ... in having most of the documents that are on the list of YES predictions actually being YES. This user wants a list that is mostly relevant documents and does not want to wade through lots of documents that are actually not relevant. Unfortunately, this means that some relevant documents are not on the list. An example: a user just wants to answer a question, so a few good documents is all that are needed.

A user with high β is interested in high recall ... in finding most of the relevant documents. This will often mean also having lots of irrelevant documents included in the list. An example: a user doing a literature survey probably wants most if not all documents in the category of interest.

F_β is a generalized measure combining the concepts of precision and recall, with a specific mechanism (β) included that incorporates the desires of the user.

Let p = precision and r = recall. F_β is computed as:

$$F_\beta = \frac{(\beta^2 + 1)pr}{\beta^2 p + r}$$

$$\text{if } p = 0 \text{ and } r = 0, \text{ then } F_\beta = 0$$

β is fixed, given the characteristics of the user. We will usually use β values of 0.5 (precision is twice as important as recall), 1.0 (precision and recall are equally important), and 2.0 (recall is twice as important as precision).

As various systems operate, each generates a value of precision and a value of recall. Each such point (p, r) is then mapped to a single value by the equation for F_β. The value of F_β lies inclusively between the values of p (precision) and r (recall).

One can certainly think of simpler equations that combine precision and recall. Although the form of the equation seems counter-intuitive, this equation was in fact developed so as to contain the following characteristics, which are representative of how actual users value the combined effects of precision and recall. In other words, once β has been determined and is fixed, F_β reflects how most users will value the tradeoff that occurs between precision and recall when evaluating several systems:

a. For a given β, F_β is largest when $p = 1$ and $r = 1$.

b. Increasing either precision or recall or both always increases F_β.

c. Different users have different interests in terms of the relative weighting of precision and recall. This relative weighting is expressed in the value of β. We therefore want the following to be true: when $\frac{\partial F_\beta}{\partial r} = \frac{\partial F_\beta}{\partial p}$, we want to be at a point (p, r) where $\frac{r}{p} = \beta$. The above equation for F_β has this property.

d. The form of the equation for F_β incorporates the fact that, while the user wants $\frac{r}{p} = \beta$, the user also does not want to have the value of either precision or recall too extreme. There is in most systems a conflict here, since it is often easier for a system to provide (for example) very high precision *and* very low recall. The willingness to trade precision for recall (or visa versa) lessens as one engages in larger trades. That is, while a $\beta = 1$ user near the point (0.6, 0.6) may be willing to give up 0.01 of precision in order to gain 0.01 in recall, this same user is probably not willing to give up 0.20 in precision in order to gain 0.20 in recall. This decreasing marginal worth is based on the notion that what most users really want is both high precision *and* high recall. For most users, sacrificing too much of one for the other is undesirable.

e. While F_β does contain some assumptions about user behavior, these assumptions are based on actual observations of user preferences. Also, the equation for F_β was validated against other measures that had been proposed, and it was shown to be a generalized form of those other measures. And finally, F_β has become a standard performance measure for those wanting to use one measure that incorporates both precision and recall.

Figure 3 is a plot of $F_{1.0}$. Note that $F_{1.0}$ is a maximum when p = r = 1 and is a minimum when p = 0 or r = 0 (or both). There is an upwards sloping fold along the line $\beta = \frac{r}{p} = 1 \Longrightarrow r = p$. Thus, for a given value of (say) p, the highest value of $F_{1.0}$ occurs when r = p.

Figure 4 shows F_β for two rather extreme values of β. Note that for $F_{4.0}$, the upwards-sloping fold is now along the line $\beta = \frac{r}{p} = 4 \Longrightarrow r = 4p$ and for $F_{0.25}$, the fold is along the line r = 0.25p.

A numeric example will help clarify the use of F_β. Say we have 4 sets of text categorization results that we want to evaluate. These could be from 4 different systems, or from 4 different operating points of the same system. We will use the F_β performance measure to determine which set of results is best. First we will examine the results for a β = 1 user (precision

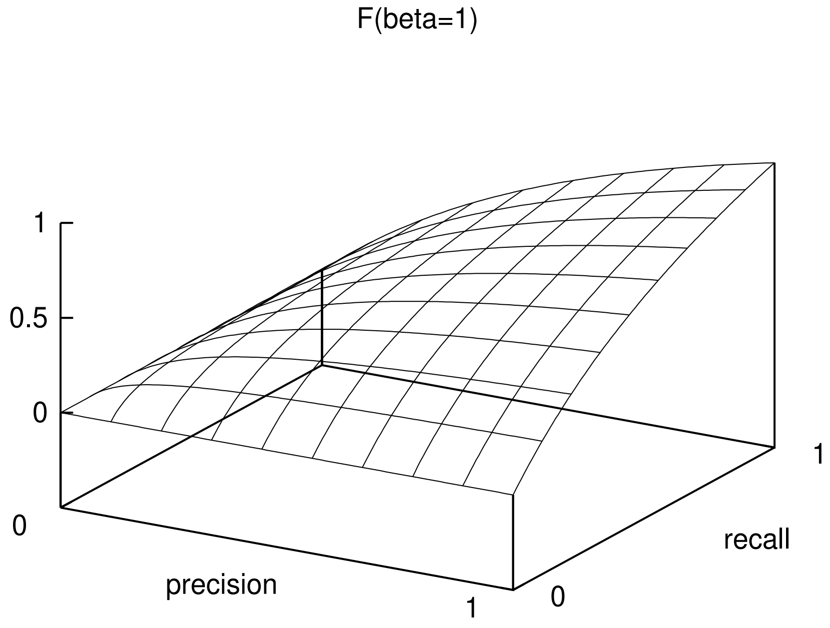

Figure 3. $F_{1.0}$

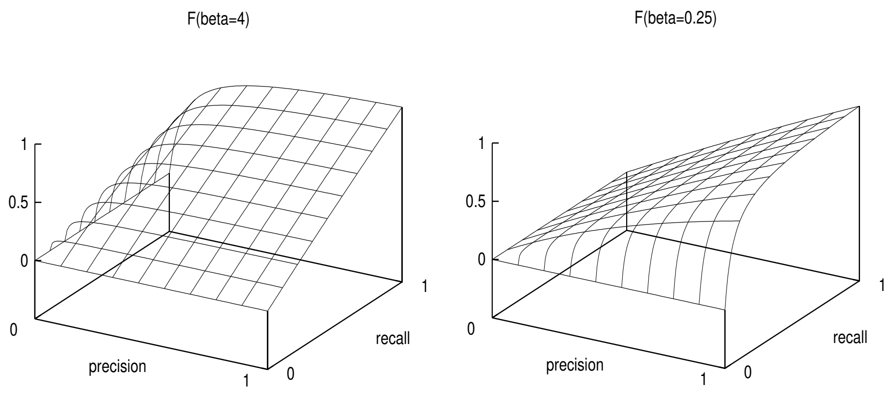

Figure 4. $F_{4.0}$ (left) and $F_{0.25}$ (right)

and recall are equally important) and then for a $\beta = 0.5$ user (precision is twice as important as recall).

Tests are run and yield the following results:

set of results	*precision*	*recall*	$F_{1.0}$	$F_{0.5}$
A	0.60	0.50	0.545	0.577
B	0.64	0.46	0.535	0.594
C	0.70	0.40	0.509	0.609
D	0.72	0.36	0.480	0.600

For the $\beta = 1$ user, A is the best since it has the largest value of $F_{1.0}$. A has slightly more precision and/or slightly less recall than the user would ideally like to have, but it is the best of the 4 sets of results that are available. B, as compared to A, trades 0.04 recall for a 0.04 increase in precision. This moves the user further away from having equal precision and recall, but causes only a slight decrease (0.010) in $F_{1.0}$. However, when a larger but still equal trade of 0.10 occurs (comparing A and C), $F_{1.0}$ drops by a larger relative amount (0.036) in spite of this user still viewing precision and recall as being equally important. This is because both comparisons A→B and A→C are moving in the same direction (increasing precision and decreasing recall with respect to the desired ratio), but the value of recall is already lower than what the user would like to have. This is an example of decreased marginal worth, which was discussed earlier.

The $\beta = 0.5$ user will prefer C since it has the highest $F_{0.5}$ value. Note that D does have better precision than B, and the $\beta = 0.5$ user weights precision as being twice as important as recall. In fact, D on the surface appears to give the user "exactly what is wanted" – a precision to recall ratio of 2. However, the assumption incorporated into the F_β calculation is that the user wants the specified ratio *and* high values of both precision and recall, so there is a tradeoff factored into the comparison of C and D. For example, D obtained additional precision, but at the cost of a decrease in an already low value of recall, and so the tradeoff included in the F_β equation favored keeping the precision value in C rather than trading it for an increase in recall.

5. **precision-recall breakeven point**

The final contingency table based performance measure that we will consider is the precision-recall breakeven point [Lewis and Ringuette 1994]. This performance measure is based on the precision-recall tradeoff that occurs in most systems. The use of this measure, in fact, tacitly assumes that there is some relatively straightforward way to get the system to tradeoff precision for recall. As discussed earlier, most users want both high precision and high recall, but if they do exhibit a preference, then they usually still do not want to use a system that operates too far from the point where precision and recall are about equal. This performance measure simply looks at the system when it has been adjusted so as to operate with precision and recall being equal; that value of precision (and recall) is the system's precision-recall breakeven point. This is a reasonable performance measure since this is often the point at which the user would like to actually use the system. It is also often a measure for which it is a challenge for most systems to achieve a high score. Many systems can operate at high precision or at high recall, but having them operate at reasonably high values of both is often much more difficult.

For many systems, the precision-recall breakeven point is a relatively easy performance measure to compute, since often the method used to categorize documents involves computing some measure of certainty as to whether each document is or is not in the category of interest. Converting this certainty figure into a prediction of NO or YES usually involves simply comparing it to some fixed value which is a parameter of the system. So, for example, a system may compute a probability p (in [0,1]) for each document it examines. To form the NO or YES prediction, it might simply compare p to a parameter h. If $p > h$ then predict YES else predict NO. If h is also a real in [0,1], then by varying h one can often tune the precision-recall tradeoff to a quite fine degree of granularity. For example, increasing h will require that the system be more certain that a document is YES before making that prediction, so the system will only predict YES for documents that are highly likely to actually be YES. Thus precision will increase. However, because the system is now predicting NO for documents that it is only moderately sure are YES, it will predict NO for more documents that really are YES, so recall will decrease.

In such a text categorization system with a parameter that is easy to adjust to get fine gradations in precision-recall tradeoff, one can adjust the system to operate at a point where precision and recall are equal. This value of precision (and recall) is the value of the precision-recall breakeven point. As a practical matter, rather than finding the settings that achieve the exact precision-recall breakeven point, one often will obtain readings on either side of the precision-recall breakeven point and then use linear interpolation to compute the value.

There is a relationship of sorts between the precision-recall breakeven point and F_β. If one looks at the equation for $F_{1.0}$ when p = r and $p \neq 0$, one finds that $F_{1.0} = \frac{(1^2+1)pp}{1^2 p + p} = \frac{2p^2}{2p} = p = r$. That is, at the precision-recall breakeven point, $F_{1.0}$ and the precision-recall breakeven point are equal. However, in general $F_{1.0}$ may be larger or smaller elsewhere (for other values of p and r). The precision-recall breakeven point is a particular value of $F_{1.0}$, so about all that one can conclude is that:

$$min(F_{1.0}) \leq \text{precision-recall breakeven point} \leq max(F_{1.0})$$

At this point one might wonder which of these is "the best" measure. In text categorization, that question does not have one answer that works for everyone. The reason this is the case, and some cautions, follow:

1. One can not look only at accuracy, since in situations where most documents are NO, a system can obtain a high accuracy score by always responding to the user's inquiry with the (very unhelpful) response "no relevant documents found".
2. One can not look at only precision or only recall, since often a system that has a high value for one has a low value for the other. One needs therefore to look at both, or at some measure that includes the effects of both (such as F_β or the precision-recall breakeven point).

An overall example will be help clarify the use of the contingency table based performance measures that we will be using. Out of 200,000 documents in a collection of medical papers, 20,000 are on the topic of cancer. A text categorization system examines the entire collection and predicts that 15,000 documents are on cancer. Of these, 12,000 actually are on cancer. So we have:

The resulting contingency table is:

	act=NO	*act*=YES
pred=NO	177,000	8,000
pred=YES	3,000	12,000

Note that in this and in subsequent examples, we will use the above row and column labels as memory aids on at least one of the contingency tables in each example.

The performance measure values for this contingency table are therefore:

1. accuracy:

$$\frac{177,000 + 12,000}{200,000} = 0.945$$

2. precision:

$$\frac{12,000}{15,000} = 0.8$$

3. recall:

$$\frac{12,000}{20,000} = 0.6$$

4. F_β:

$$F_{0.5} = \frac{(0.25 + 1) \times 0.8 \times 0.6}{0.25 \times 0.8 + 0.6} = 0.75$$

$$F_{1.0} = \frac{(1.0 + 1) \times 0.8 \times 0.6}{1.0 \times 0.8 + 0.6} = 0.69$$

$$F_{2.0} = \frac{(4.0 + 1) \times 0.8 \times 0.6}{4.0 \times 0.8 + 0.6} = 0.63$$

5. precision-recall breakeven point:

Assume that we adjust the parameters in the system (to cause precision-recall tradeoff to occur), and the same text categorization request now gives the following results:

	act=NO	*act*=YES
pred=NO	171,000	6,000
pred=YES	9,000	14,000

Note that the total number of documents that are actually NO and that are actually YES (180,000 and 20,000 respectively) have not changed, but the system is now doing a better job at correctly predicting YES documents (14,000 now versus 12,000 earlier). Unfortunately, the system is also doing worse at predicting NO documents (171,000 now versus 177,000 earlier). If we compute precision and recall for this second test, we get:

$$p = \frac{14,000}{23,000} \approx 0.61$$

$$r = \frac{14,000}{20,000} = 0.8$$

To compute the precision-recall breakeven point using linear interpolation, We do the following. We have a line p = m r + b defined by 2 points (r, p) = (0.6,0.8) and (0.8,0.61). Therefore, $p = -0.95r + 1.37$ and so the precision-recall breakeven point is at $p = \frac{b}{1-m} \approx 0.70$. (In a real evaluation, this is probably a larger interpolation interval than one would like to use.)

Note that any system could have achieved a respectable accuracy of 0.90 by always predicting NO. However, both its precision and recall would be 0.0. This is in fact a problem we encountered in some of the active learning methods that we developed (and discarded); under certain conditions, the learner would learn to "just say NO" and still get a good accuracy score. This is in fact a common problem with some types of learning algorithms in certain domains.

2.6.2 Averages

Often one performs a series of many tests on the same system, varying some aspect of the problem that the system is solving (such as the category). An issue that arises in such situations is how to compute one average value that is representative of the system's performance on the entire series of tests. For performance measures such as elapsed processor time and number of training examples used, the concept of average is well-defined. However, for performance measures based on the 4 values in the contingency table, what one means by "average" is perhaps not so clear. The problem is that each test will have a resulting contingency table, and there are several ways in which one can combine the data in these tables to form one single performance measure value. There are two main approaches to solving this problem [Lewis 1991a]. Basically, one needs to decide between two ways of computing an average performance measure value:

1. **Macroaverage**

   Compute the performance measure for each test's contingency table and then average these performance measure values to get a single value for the series of tests. This is referred to as the "macroaverage" approach.

This approach assumes that each text categorization request is equally important and so deserves equal weighting. Or, in the case of information retrieval, macroaveraging assumes that each query is equally important.

Macroaveraging is usually used in information retrieval experiments because it gives equal weight to each user query regardless of how many documents are retrieved by each query.

2. **Microaverage**

Combine (add) all of the contingency tables to obtain one "table of totals" for the series of tests. This table is referred to as the aggregate table – each entry is the sum of the corresponding entries in all of the trial contingency tables. Then one computes the performance measure using the aggregate table. This is referred to as the "microaverage" approach.

Each document-category pair requires that the system make a decision (a prediction of YES or NO). This approach assumes that each of these decisions should be weighted equally.

If one looks at a series of tests, in which one submits several categorization requests to a system, then *micro*averaging treats each prediction decision on each document for each requested category as the case of interest – so each case is a small part of the entire series of tests. On the other hand, *macro*averaging treats the entire response of the system to each of the categorization requests as the case of interest, so each case is a much larger portion of the entire series of tests [Kantor 1999].

Since microaveraging is usually used in text categorization experiments, we used microaveraging in this thesis.

As an example of macroaverage and microaverage calculations, consider the following two contingency tables. They are from two runs of the same system (each run was made with different parameter settings) on a collection containing 10,000 documents.

A:

	act=NO	*act*=YES
pred=NO	5,000	0
pred=YES	3,000	2,000

B:

	act=NO	*act*=YES
pred=NO	3,000	4,000
pred=YES	2,000	1,000

To demonstrate the differences that can occur with these two approaches for computing averages, we compute the macroaveraged and microaveraged recall for the above 2 trials.

$$r_{macro} = \frac{\left(\frac{2,000}{2,000} + \frac{1,000}{5,000}\right)}{2} = 0.60$$

$$r_{micro} = \frac{3,000}{7,000} = 0.43$$

Even in this example with only two tests, the averaging method used makes a big difference. Macroaveraged recall is assuming that the recall values for each of the two tests are equally important. Macroaveraged recall does not take into account the fact that test A contains only 40% as many documents that are actually YES as test B (A has 2,000; B has 5,000). Microaveraged recall is assuming that each of the 20,000 YES-NO predictions being made by the system is of equal importance, and in fact microaveraged performance measures look upon the above series of tests with a focus on the fact that it is *a* series. Microaveraging views the series as the following aggregate table:

	act=NO	*act*=YES
pred=NO	8,000	4,000
pred=YES	5,000	3,000

Before introducing the final system evaluation concepts that we will be using, we need to provide an overview of how the various systems are tested and also introduce some additional terminology. The various corpora used for testing typically contain several categories. That is, each document in the collection has been labeled as being in any number (or none) of a large number of possible categories. Rather than test using all possible categories, most corpora have an established set of categories that are used in testing text categorization systems. We use these categories for our tests so as to allow comparison of our results to those of others. Each category has a name, but is usually referred to by a unique category number which we assign to it.

The procedure we use for testing a text categorization system is to first subdivide all of the documents in the corpus into a training and test set. These two sets are disjoint. The system learns from the training set and is tested using the test set. However, during learning one is not allowed to use any aspect of the label values for documents in the test set.

Testing (using the test set) can occur at any time. We do testing *during* learning so as to be able to plot learning curves. These plots show how performance measure values change as learning is occurring, and are of interest in comparing learners for situations where learning may be interrupted at some point (perhaps by an impatient user demanding to have the system "use what you know

now"). And of course testing is done at the end of learning so that we can measure the final performance of the systems.

The assumption is that both the training set and the test set are, on the average, representative of the entire corpus. Most of our experiments create the training and test sets by randomly splitting the corpus. We use every document in the corpus in either the training set or the test set. We refer to a particular division of the corpus into training and test sets as a "split". Some splits are difficult to accurately learn, because the training set may happen to be not very representative of the test set. One therefore needs to run tests on many splits and then compute an average of the results in order to get an accurate picture of how the system actually behaves.

We therefore want to train and test each system on several categories using several splits. Each execution of the system on a particular category using a particular split is termed a "trial". Figure 5 outlines the steps in a typical run (series of trials) using the so-called train-and-test approach.

Once the series of trials shown in Figure 5 has completed, we have a 2x2 contingency table for each trial. We in effect have a 2 dimensional matrix whose elements are contingency tables. An example will help to demonstrate the situation that results from a series of trials. Assume that we have a collection containing 500 documents, and we decide to use a training set size of 400 and a test set size of 100. We are going to run tests on two of the categories in the corpus – categories 1 and 2. For each category, we will use two different training-test set configurations – we will use two splits, splits 1 and 2. The series of tests therefore contains 4 trials. In each trial, the system will learn from the training set and then be tested using a test set of 100 documents. The results of each trial will be a 2x2 contingency table, and the results of the 4 trials can be thought of as a matrix containing the 4 contingency tables, with the rows of the matrix representing different splits and the columns representing different categories.

for split in $split_1$, $split_2$, $split_3$, . . .
    for category in $category_1$, $category_2$, $category_3$, . . .
        (start of this trial)
        system learns using the training set
        system is tested using the test set
        results (2x2 contingency table) are saved
        (end of this trial)
    end
end
compute and output the results for the series of trials

Figure 5. Steps in Typical Train-and-Test Run (Series of Trials)

Consider the following example:

		category 1: act=NO	category 1: act=YES	category 2: act=NO	category 2: act=YES
split 1	pred=NO	7	72	21	4
split 1	pred=YES	16	5	53	22
split 2	pred=NO	5	1	48	2
split 2	pred=YES	90	4	33	17

Note that there is no requirement that the column totals (the number of documents for which the actual label is NO, YES) for any trial be equal to those for any other trial. Each split results in a different test set. Each category within a particular split has in general a different distribution of NO and YES labels. These two facts combine to result in there being no requirement on column totals matching between any two contingency tables resulting from a series of trials. It is however true that each contingency table has its entries sum to the same value (100 in the above example), since there are the same number of documents in each test set.

Continuing with the above example, we will use recall as the performance measure that we are computing. The approach is similar for other performance measures. Let $r_{s,c}$ stand for the recall value for the trial of split s and category c. Then we have:

$$r_{1,1} = \frac{5}{77} = 0.064$$

$$r_{1,2} = \frac{22}{26} = 0.846$$

$$r_{2,1} = \frac{4}{5} = 0.800$$

$$r_{2,2} = \frac{17}{19} = 0.895$$

therefore:

$$r_{macro} = \frac{0.064 + 0.846 + 0.800 + 0.895}{4} = 0.651$$

Microaveraged recall is computed from the aggregate table formed by summing the contingency tables from all of the trials, which is:

	act=NO	*act*=YES
pred=NO	81	79
pred=YES	192	48

Therefore:

$$r_{micro} = \frac{48}{79 + 48} = 0.378$$

It turns out that two other types of averages are useful to us in analyzing system behavior from a series of trials:

1. **Categoryaverage**

   Each document-category pair requires that the system make a decision (a prediction of YES or NO). Recall that microaveraging is based on the assumption that each of these decisions should be weighted equally over all trials. It is also useful to apply this assumption to all trials for a particular category. This results in an aggregate contingency table for each category. One computes the performance measure for each such aggregate table and then computes the average of those values. We refer to this as the "categoryaverage".

   In the matrix of trial contingency tables (rows are splits, columns are categories), this results in first forming an aggregate contingency table for each column and using those tables to compute the performance measure value for each column, and then taking the average. Essentially one is microaveraging each column and then macroaveraging the column performance measure values. The real value of this concept is in being able to determine how system behavior varies by category (over all splits), which one can do by looking at the microaveraged values for each column (category). Some categories may be easier to learn than others, and this computation gives us that information.

2. **Splitaverage**

Correspondingly, one can do the same thing for the splits. That is, compute an aggregate contingency table for each split (each row of the matrix of trial contingency tables), compute the performance measure for each of those tables, and then compute the average. We refer to this as the "splitaverage". This concept is useful in that it allows one to see how system behavior varies by split (over all categories). Some splits may result in the training set being especially poor at representing the test set, while for other splits the learner may do very well. If there is a large amount of variation in system behavior for different splits, then we have a situation in which there is a large amount of variation among different splits in the knowledge the learner is gaining from the training set as compared to the knowledge needed to correctly categorize the test set. In such a situation, we need to run more splits so as to be more confident that our results are representative of overall system behavior.

We will finish our discussion of performance measures by computing the categoryaverage and splitaverage for the above example of a 4 trial test series (2 splits x 2 categories). Let $r_{catAve,c}$ stand for the categoryaveraged recall for category c only. Then:

$$r_{catAve} = \frac{r_{catAve,1} + r_{catAve,2}}{2}$$

$$r_{catAve,1} = \frac{5+4}{(72+1)+(5+4)} = 0.110$$

$$r_{catAve,2} = \frac{22+17}{(4+2)+(22+17)} = 0.866$$

therefore:

$$r_{catAve} = \frac{0.110 + 0.866}{2} = 0.488$$

Note that the intermediate calculations for categoryaveraged recall tell us that, for this system, recall for category 1 is much worse than for category 2.

Let $r_{splitAve,s}$ stand for the splitaveraged recall for split s only. Then:

$$r_{splitAve} = \frac{r_{splitAve,1} + r_{splitAve,2}}{2}$$

$$r_{splitAve,1} = \frac{5 + 22}{(72 + 4) + (5 + 22)} = 0.262$$

$$r_{splitAve,2} = \frac{4 + 17}{(1 + 2) + (4 + 17)} = 0.875$$

therefore:

$$r_{splitAve} = \frac{0.262 + 0.875}{2} = 0.569$$

The intermediate calculations for splitaveraged recall tell us that, for this system, the splitting process is producing training and test sets with large variations in the knowledge contained in each. This would be an indication that one would need to run a large number of splits in order to get average performance measure values that were representative of the system's overall behavior.

2.6.3 Splitting

A couple of brief final comments on splitting. We use random splitting and run several train-and-test trials, each on a different split, and then compute averages of performance measures. Another approach is to use one of the various standard resampling techniques such as cross-validation or bootstrapping. However, it is felt by many that, if one has enough data (which we do), then a train-and-test approach on a random split is easiest to analyze and has very clear theoretical support [Weiss and Kulikowski 1990]. One such random split does, however, have the disadvantage of ignoring some examples for training. This is why we use multiple random splits.

Many if not most text categorization researchers use the train-and-test approach on a single split, which they usually define in their published results. Such predefined splits are usually not constructed in a random manner – they often are purposely time-based to simulate real-world data flow (earlier documents are for training and later ones are for testing), and they often include only documents that have certain characteristics (such as which categories they belong to, how many categories they belong to, etc.). While the use of such predefined splits makes experiments go much faster and makes it easier for different researchers to compare results, we are concerned that such methods may not give performance measure estimates that are unbiased, of low variance, and that cover the entire corpus. We will, when necessary for comparison purposes, use such predefined splits. While they make comparisons amongst systems easier (or at least, should), we have a genuine concern as to whether they give results indicative of overall system performance that would be experienced by a general user [Weiss and Kulikowski 1990].

Another issue that comes up when one uses predefined splits is due to most predefined splits not using all of the documents in the corpus. Some documents in the corpus are omitted completely, appearing in neither the training nor the test set, but rather being placed in an unavailable set. When this occurs, the question arises as to whether one should shorten the feature vector so that there are only slots for those tokens that appear in either the training or test set being used, or should one continue to use the feature vector format that applies to the entire corpus. Most researchers do in fact shorten the feature vector when using a predefined split, effectively treating the training and test sets as a new and smaller corpus. We view this action as a form of dimensionality reduction, which we do not do. In particular, our view is that having certain documents flagged as unavailable for a particular test is a temporary condition. We do not modify the feature vector to take advantage of this, but instead always use the feature vector that applies to the entire corpus.

2.6.4 Performance Measures We Will Use

As mentioned before, elapsed processor time and the number of training examples used are for us the performance measures of greatest interest. We will generally give the number of training examples used as a percentage of the total number of examples that are available for training. The elapsed processor time is given in seconds. The system on which all experiments were performed is an IBM-compatible 486-DX4-133 PC, 80 Mb of 70ns access time main memory, 5Gb of 11ms seek time hard disk (on 2 disk drives), operating system is Linux version 1.2.13. The programs are written in C++, and the gnu C++ compiler version 2.7.0 was used to compile and link them.

To make sure that our systems are doing a good job of categorizing text, we also use various contingency table based performance measures. We will use macroaveraged accuracy, since it is probably the most-often used performance measure in the field of machine learning. We also want to include microaveraged recall and microaveraged precision in our evaluations, since these are the performance measures most often used in the field of text categorization. We will avoid the dilemma of predicting whether the user favors recall or precision by examining both, or as is more often the case by examining microaveraged F_β for β = 1.0. This means that the user wants both good precision and good recall, and does not want to sacrifice very much of one to get more of the other. We use these values of β since, from our experience, this makes the resulting systems more valuable to the user. At least in our research, many systems tested were able to give good precision *or* good recall, but few do well at both. And few users desire systems that operate at either extreme, since having high precision at all costs means that one will miss many relevant documents, and having high recall at all costs means that one will have long lists of documents predicted to be YES that must be manually filtered to separate out the many documents that are actually irrelevant.

3. Previous Research

In this chapter we give an overview of previous relevant research in the areas of the vector space model, dimensionality reduction methods, supervised learning algorithms, active learning, bagging, and boosting.

3.1 Vector Space Model

The vector space model for document representation was developed by Gerard Salton in the 1960's [Salton 1971, Salton and McGill 1983, Salton 1989]. This model is similar to the nearest neighbor model, which is widely used in machine learning and in pattern recognition [Aha et al. 1991]. The vector space model is simple to envision conceptually, and it obtains quite good results. It is the basis of the majority of existing text categorization and information retrieval systems, and concepts related to the development of the model are used in most other systems that are not purely vector space. The development of the model has spurred many of the advances that have been made over the past 20 years in the fields of text categorization and information retrieval.

Basically, the vector space model envisions documents as vectors existing in an n dimensional document space. Each dimension corresponds to some characteristic that can be attributed to documents. These characteristics are usually some encoding of the words present in each document, such as the number of occurrences of each word. One can also encode language structure, document structure, etc. as characteristics. Typically the number of characteristics (n) is very large (by machine learning standards) – anywhere from 10^3 to 10^7, depending on how the characteristics are chosen. There are many possibilities as regards choices of dimensions, and in fact this is an area that differentiates many of the text categorization systems that are based on the vector space model.

The vector space model measures *similarity* between two documents by the cosine of the angle between their vectors. Recall $cos(0°) = 1$ and $cos(90°) = 0$. Therefore, vectors that are collinear or nearly so will have a high cosine similarity measure, whereas those that have an angle between them that is close to 90° will be very dissimilar.

One (simple) way that this model can be used for text categorization is by remembering the vectors corresponding to the documents whose categories are known, and then determining whether or not the similarity is sufficiently high for a previously unseen document so as to include it in that category.

While the vector space model makes many assumptions that are known to be violated by natural languages, it nevertheless performs well. A great number of tests run on a great number of varying implementations of this model have shown that it gives quite accurate results [Salton 1971].

3.2 Dimensionality Reduction Methods

Most methods of text categorization that have been applied to collections with a large vocabulary use one or more methods to reduce the number of words that are actually "stored" in the representation of each document. This dimensionality reduction is done to reduce the number of attributes and often to also attempt to remove irrelevant features. This is typically done in a preprocessing step that is executed only once for each document in the collection.

We next present a brief discussion of some of the more common dimensionality reduction methods. Some of these methods are specific to text categorization and information retrieval, others are generally applicable to a variety of domains. Note that many systems use a combination of these approaches rather than just one. We have also included comments on how each method can encounter problems when used to categorize text. The use of dimensionality reduction methods is quite common, however, and in fact one of the aspects of our research that is unusual is that our preprocessing is minimal – we do not do any dimensionality reduction.

Some of the more common dimensionality reduction methods are:

1. removal of common words ("stoplisting")

   Words that appear often are thought by many to not be of any use in text categorization since they presumably have little differentiating value. One creates a "stop list" of common words. The stop list is a list of words that will be ignored – stoplisted words are effectively removed from the document during the preprocessing step. Examples of words often found on stop lists: a, an, the, they, to, of, and, . . . A problem that can occur using stoplists is that one may in fact remove words that would have been helpful or even necessary in categorizing the text. For example, removal of certain prepositions and auxiliary verbs with a stoplist can produce quite different results in text categorization system behavior [Riloff 1995]. ("Auxiliary verbs" are verbs accompanying other verb forms in order to indicate tense or mood – she *could* walk to work, she *will* walk to work.)

   As another example, many systems doing stoplisting remove any "word" that consists only of a single letter. This makes it very difficult to obtain documents about the C programming language.

2. stemming ("lemmatizing")

   Many different words may have a common stem. One can therefore reduce dimensionality greatly by replacing each word by its stem. The assumption is that the stem contains the important part of the meaning of the word, so stemming should do little to change the meaning of the document as a whole. For example, the words "banked", "banking", "bank", "banks", and "banker" all have "bank" as their stem. If we used stemming, then all occurrences of any of these 5 words would be replaced by "bank". Stemming is usually done using a stemming algorithm (versus a table lookup), and there are a great number of stemming algorithms in use [Hull 1996]. However, a potential problem is that in fact some information *is* being lost by the stemming process. For example, if a document contains the word "banker", then the document may be about finance ... it is probably not about flying airplanes or plowing snow. But if the document is stemmed, then this information is not available to the learner. There has been some research in the area of specifically examining the impact of stemming [Hull 1996]. It has been found that loss of plural noun form by stemming can greatly impact text categorization [Riloff 1995] and in general most stemming needs to be much more tailored to the words being stemmed and to the collection of documents if one does not want to modify the text so much as to significantly impact text categorization [Church 1995].

Especially in languages that are morphologically more complex than English (such as Russian), one often finds the term "lemmatize" used to describe the concept of determining the "base form" of a word. To many people in linguistics, "stemming" refers only to that aspect of lemmatization that is accomplished by removing word endings to find the stem. For example, jumping→jump and loves→love can be accomplished by stemming, but went→go and thought→think require a lemmatizer.

3. token "condensation"

   Many systems condense several tokens into one token by replacing all tokens of a particular type by one "reserved" token. For example, one often finds that tokenizers replace any and all numbers by a reserved token such as "num" [McCallum and Nigam 1998]. Other systems might replace all date-like constructs with "date", time-like constructs with "time", etc.

4. attribute/feature selection

   There is a host of methods that attempt to use some reasonable criterion for selecting which attributes will be used in the feature vectors and which will be omitted. These approaches often tend to be applicable to other domains (versus stoplisting and stemming, which are specific to text categorization and information retrieval). Most of these methods perform one of several mathematical analyses on the training data to determine which features are of greatest discriminatory value. For example, removing features that have very high frequencies of occurrence [Apté et al. 1994], removing features that have very low frequencies of occurrence [Lewis and Ringuette 1994], concave minimization [Bradley and Mangasarian 1998], category utility [Devaney and Ram 1997], rough set data reduction [Jelonek et al. 1995], information gain [Apté et al. 1994, Schütze et al. 1996], various forms of hill-climbing [Doak 1992], genetic programming [Sherrah 1998], and breadth first search [Almuallim and Dietterich 1991]. The problem that can occur when these methods are used with text categorization is: would in fact the words that were removed have been of little or no use in categorizing the text? This is a difficult question to answer decisively since often the results of tests depend greatly on the specific document collection being used.

5. attribute creation

Several methods reduce dimensionality by creating a new set of features that are mathematically related to the original set, but that contain many fewer features. Some of the methods in use are boolean feature generation [Kudenko and Hirsh 1998], singular value decomposition [Yang and Chute 1994], latent semantic indexing [Berry et al. 1995, Schütze et al. 1996], connectionist networks [Saund 1989], and computation of eigenvectors or characteristic vectors [Franklin 1968, Bellman 1970]. Each method employs its own procedures and has its own corresponding metrics as regards any restrictions on the new dimensions (such as independence), how the new dimensions are computed, and how the error introduced by the dimensionality reduction is measured and minimized. The idea behind all of these attribute creation methods is that one hopes to find a much smaller set of new dimensions that can express almost all of the information contained in the original data. These new dimensions will in general be some weighted combination of the original dimensions. For example, instead of having the tokens "computer" and "science", one might have a single dimension that is "$0.3 \times computer + 0.2 \times science$". Indeed, one of the disadvantages of this approach is that it is difficult to meaningfully (to a human) tie the results back to the original documents. Another problem is that these methods have very high time complexities, usually $O(n^3)$ or worse, where n is a measure of the size of the original corpus.

Many dimensionality reduction methods are also extremely time consuming and so are rarely done online. Instead, the computations are performed in batch mode on the entire document collection. Methods that use this approach must therefore partition their text categorization activities so that this time consuming portion of the task has already been done and the results saved when the time arrives for a user to actually interact with the system. Such a restriction on system use also makes it more difficult for such systems to respond quickly to changes in the document collection, such as can occur when individual documents are updated, removed, or added.

Yang and Chute have developed an approach that combines knowledge engineering and machine learning technologies by using labeled training data to obtain associations between document words and the words used to describe the desired categories [Yang and Chute 1994]. The set of words in the training documents form a "source vocabulary". The set of words in the category descriptions form a "target vocabulary". Recall that one form of attribute creation is to compute a new set of dimensions that is far fewer in number but is able to express (most of) the information in the original data. They carry this idea a step further by specifying that the new set of dimensions be the words in the category descriptions themselves. They employ singular value decomposition to obtain a mapping from the unrestricted document vocabulary to the restricted document indexing vocabulary. Categorization can then be performed by using the weights in the mapping matrix to compute a score for each test document for each category, with the highest score indicating the category to be predicted for that document.

3.3 Supervised Learning Algorithms

In supervised learning, categorization knowledge is learned from documents that have already been categorized by a human and stored in a knowledge base.

One representation for this knowledge base that is especially easy for a human to understand is a set of if-then rules. Typically the set of if-then rules is given as an ordered list. Given a document D to categorize, one starts at the top of the list of if-then rules and evaluates each if-clause. The first if-clause that is true gives one the predicted label (in the then-clause). One of the if-then rules in the list might be:

> If D contains the word 'storm' and the word 'desert',
> but not the word 'meteorology' then category is 'war'.

One of the most impressive results in applying machine learning to text categorization has been obtained by Apté, Damerau, and Weiss, using optimized rule-based induction. They reported a 0.805 precision-recall breakeven point using the Reuters-22173 collection [Apté et al. 1994]. However, to achieve this result, over 10,000 labeled examples were used. Both general and category-specific feature selection were used. For general feature selection, they sorted all words and word pairs by frequency and kept the 10,000 most frequent terms. This list was further reduced by removing function words (high frequency "contentless" words). Category-specific feature selection was then performed using statistical methods (such as entropy) to remove additional terms deemed irrelevant to that category.

The decision tree method of constructing a knowledge base examines the attributes one value at a time, leading one to an eventual decision. For example, to decide if an animal A is a mammal, one might first ask "Does A lay eggs?". If the answer is no, then predict mammal. If the answer is yes, then ask "Is A a platypus?". If the answer is yes, then predict mammal, otherwise predict not mammal.

Decision trees are the machine learning algorithm of choice for many, and this method has also been applied to text categorization [Lewis and Ringuette 1994, Sakakibara et al. 1996]. Lewis and Ringuette did an extensive comparison of decision tree methods with a Bayesian approach. They concluded that each approach had its strengths and weaknesses, but that both were accurate enough for operational use if feature selection was used as a pre-filter [Lewis and Ringuette 1994]. They experimented with several feature selection methods, including category-specific information gain, stoplisting, and removal of infrequently-occurring words.

Both rule learning and decision tree learning *need* to do some feature selection because they perform poorly when there is a large number of irrelevant features. On the other hand, certain linear threshold learning algorithms seem to scale better that others when large numbers of attributes are irrelevant.

The perceptron learning algorithm is a widely-used linear threshold learning algorithm [Rosenblatt 1962, Weiss and Kulikowski 1990]. For examples represented by a boolean feature vector of length n, the perceptron maintains its hypothesis as a vector of n weights and a threshold value θ. One can envision the threshold and weights as defining the location of a hyperplane. Initial values are assigned to the weights and to θ. As examples are seen, the weights and θ are adjusted in an effort to move the hyperplane so that it separates the YES and NO examples. The weights and θ are adjusted by adding a small constant to each of them so that the hyperplane moves in the correct direction. The rate at which the hyperplane moves is determined by parameters of the algorithm. The perceptron is a mistake-driven learning algorithm, meaning that it learns only when it has predicted incorrectly.

"Winnow" refers to a quite large family of algorithms [Littlestone 1989]. These algorithms are similar conceptually to the perceptron algorithm, in that they also attempt to position a hyperplane so as to separate the YES and NO points in the n dimensional document space, and they are also mistake-driven. The winnow algorithm, however, uses a different method for computing the amount of the shift. It uses multiplication by a small constant rather than addition to change the weights, thereby shifting the hyperplane. Also, winnow algorithms typically adjust only the weights and not the threshold value as learning occurs.

The naive Bayes learning algorithm is based on the work done by Duda and Hart [Duda and Hart 1973, Dietterich 1997, Mitchell 1997]. This method estimates the probability that each feature is present given the class. Then Bayes rule is used to compute the probability of each class given the feature values present. When an example is presented for classification, the algorithm multiplies the prior probability for each class value times the associated conditional probabilities for each feature value given that class value. One such product is formed for each possible class value. Whichever product is largest indicates the class value to predict.

The "naive" aspect of the algorithm is that the algorithm makes no attempt to ascertain relationships between the features. For example, in a collection of newspaper articles, the presence of the word "dog" in an article about food might increase the probability of the presence of the word "hot", as in "hot dog". The naive Bayes algorithm would not use such information in categorizing an article, but rather would look at the words "hot" and "dog" as if the occurrence of one were independent of the presence or absence of the other.

While the naive Bayes algorithm makes seemingly major simplifying assumptions in calculating the probabilities used to determine its predictions, it very often achieves quite good results when used for classification in many different domains [Kononenko 1990, Langley 1993, Singh and Provan 1995, Domingos and Pazzani 1996, Domingos and Pazzani 1997, McCallum et al. 1998].

Most of our experiments used the perceptron, winnow, and naive Bayes algorithms, and so these 3 algorithms will be discussed in more detail later.

3.4 Active Learning

"Active learning" in its most general sense refers to any form of learning wherein the learning algorithm has some degree of control over the examples on which it is trained [Cohn et al. 1994]. One active learning approach is the membership query paradigm, in which the learner can construct new sets of inputs and request that the teacher provide their labels [Angluin 1988]. The normal strategy used by the learner is to generate examples that are close to examples for which it already knows the labels. The idea is to try to map the boundaries of the solution space regions that define each class. The use of membership queries for active learning in text categorization would utilize a learner that generates documents which are slight variations of what a human would write. However, these artificial documents would be very difficult for a human to meaningfully and reliably classify.

The other type of active learning is meticulous sampling. In this paradigm, the teacher has available to it all of the examples. However, the learner carefully ("meticulously") chooses which examples it will use for learning. The idea is that a series of randomly chosen examples often contains repetitive information, so one could perhaps use fewer examples if they were selected carefully. Typically the cycle proceeds as follows. The teacher presents the learner with the feature vector portion of an example (i.e., the example without the label). The learner examines the feature vector and then decides whether or not to ask for the label. If the label is requested, then the example is considered as having been used as a training example by the learner. The idea behind this way of counting the number of examples that are actually used is that each label must ultimately be provided by a human, whereas the feature vector is generated by the action of the preprocessor program on a pool of already-gathered raw data. Once the data has been gathered and the preprocessor written, providing feature vectors is relatively inexpensive in terms of human involvement. However, human effort is required to provide each label. Since the label is the most expensive portion of the example to provide incrementally, the example is not counted as having been used by the learner unless the learner also sees the label.

Most methods of active learning employing meticulous sampling to decide whether or not to ask for the label (and thereby use the example) form their decision by computing some measure of how informative the example might be. This measure is usually computed based on the current knowledge state of the learner/s.

Recently there have been some promising results in the active learning area. Query by Committee (QBC) is a learning method that uses a committee of hypotheses to decide for which examples the labels will be requested. It also uses the committee to predict the value of the label. Since QBC exerts some control over the examples on which it learns, it is one form of active learning. QBC maintains a committee of all hypotheses consistent with the labeled examples it has seen so far – a representation of the version space. (Given a hypothesis space and a set of training examples that have been used, the "version space" is the subset of all candidate hypotheses that are consistent with those training examples.) Each unlabeled training example is presented to the algorithm. An even number of hypotheses (usually 2) is chosen at random, given the attribute values of the example, and asked to predict the label. If their predictions form a tie, then the example is assumed to be maximally informative, the algorithm requests the actual label from the teacher, and the version space is updated [Freund et al. 1992, Seung et al. 1992, Freund et al. 1997]. Freund, Seung, Shamir, and Tishby analyzed QBC in detail and showed that the number of examples required in this learning situation is proportional to the logarithm of the number of examples required for random example selection learning [Freund et al. 1992].

Cohn, Atlas, and Ladner developed the theory for an active learning method called *selective sampling*, and then applied it to some small to moderate sized problems, as a demonstration of the viability of this new approach [Cohn et al. 1994]. In selective sampling, the learner computes a region of uncertainty based on the examples it has seen so far, and then draws the next example from that region of uncertainty (by querying an oracle). Their results indicate a decrease by a factor of 2 - 3 in the number of labeled examples needed by the learner for problems suited to this method. A key issue in determining the applicability of this approach for any domain is whether or not the learner can generate legal examples, and whether or not one can efficiently and effectively represent the regions of uncertainty.

Lewis and Gale developed a method that is conceptually similar to selective sampling, but it is specifically meant for use with an input stream of candidate training examples. Their method is called *uncertainty sampling* – the learning algorithm selects for labeling those examples whose membership is most unclear [Lewis and Gale 1994]. They developed the approximations necessary in order to determine which examples are most ambiguous to classify. Since the method was developed for text categorization, it is able to handle noise as well as a large numbers of features. Their approach greatly reduced the amount of training data that was needed, often by 2 - 3 orders of magnitude when compared to random sampling.

Dagan and Engelson proposed a general method, termed *committee-based sampling*, for selecting examples to be labeled [Dagan and Engelson 1995, Argamon-Engelson and Dagan 1999]. Their method employs a set of models (the "committee"). The informativeness of an example (and so the desirability of having it labeled) is indicated by the entropy of the predictions of the various committee members. They found that committee-based sampling improved learning efficiency by significant amounts as long as the desired accuracy was reasonably high, and that the amount of efficiency gained increased as desired accuracy increased.

While approaches and results vary, all of these studies and others conclude that active learning improves learning efficiency (decreasing the number of training examples used) by significant amounts.

3.5 Bagging and Boosting

A problem often encountered in machine learning is that, for a given set of training and test examples, one may get very different results if one, for example, removes just one of the training examples. In general, a learning method is termed "unstable" if small changes in the training-test set split can result in large changes in the resulting predictor [Breiman 1996b]. A method termed "bagging" (***b***ootstrap ***agg***regat***ing***) will in many such cases result in the ability to predict more accurately [Breiman 1996a]. Bagging is a method for reusing the training data. If one has a training set of size t, then one draws t random examples from it with replacement (using a uniform distribution), uses those t examples to learn, and then repeats this process several times. Since the draw is with replacement, usually the examples drawn will contain some duplicates and some omissions as compared to the original training set. Each cycle through the process results in one knowledge base. After the construction of several knowledge bases, learning ceases. Prediction is then performed by taking a vote of the predictions of the individual knowledge bases.

Another method used in machine learning to attempt to get better accuracy from a training set is called boosting [Schapire 1990]. AdaBoost ("***ada***ptive ***boost***ing") is a practical version of the boosting approach [Freund and Schapire 1995, Freund and Schapire 1996]. Boosting is similar in overall structure to bagging, except that one keeps track of the performance of the learning algorithm and forces it to concentrate its efforts on examples that have not been correctly learned. In AdaBoost, one keeps track of the accuracy of the knowledge base resulting from each learning cycle. Instead of choosing the t training examples randomly using a uniform distribution, one chooses the training examples randomly but in such a manner as to favor the examples that have *not* been accurately learned in the past. The idea is that successive learners will benefit from the experience of the earlier learners by focusing on portions of the example space that those earlier learners were not able to accurately generalize. After several cycles, one has (as in the case of bagging) several knowledge bases. Learning ceases, and prediction is then performed by taking a weighted vote of the predictions of the individual knowledge bases, with the weights being proportional to each learner's accuracy on its training set.

4. Active Learning with Committees (ALC)

In this chapter, we discuss the goals of our research project, our Active Learning with Committees (ALC) approach to text categorization, and the main techniques used.

4.1 Goals and Motivation

In developing text categorization methods, we have the following goals:

1. use fewer training examples

   We want our methods to make efficient use of examples, and this is our motivation for using active learning. The notion is that, by choosing training examples *carefully*, we may be able to learn as much without using so many examples. The reason for wanting to reduce the number of examples used is that the labeling of examples must typically be done by a human, so we are trying to reduce human costs, both in time and money.

2. scale up to large numbers of features (minimal preprocessing)

   Any text categorization method that looks at individual tokens in text must in some way address the problem of the very large number of different tokens that exist in most natural languages. It is not unusual for the attribute vectors that represent documents to contain 50,000 – 200,000 elements. There is, at least initially, a "slot" in the feature vector for every token that appears in any document in the collection.

Many machine learning approaches do not scale up well – they do not perform acceptably when the number of dimensions is increased to such large values. This problem is usually handled in one of two ways:

a. Use a preprocessing step that will reduce the number of tokens that one has in the document representation. Examples: stop lists, stemming, singular value decomposition, latent semantic indexing, statistical attribute selection, and attribute replacement/creation.

b. Use a machine learning approach that scales up well (and therefore presumably does some form of attribute filtering itself).

Our goal is to use the latter approach – to use a learning algorithm that allows one to use the textual data pretty much "as is", with the learning algorithm itself coping with the high dimensionality directly. We will not engage in preprocessing that changes the tokens that are in each of the documents. This minimal preprocessing approach makes the overall system more general and also as an added benefit saves preprocessing execution time.

Preprocessing can in general be somewhat tricky anyway, in that one often forgets that the preprocessing has in a sense changed the problem being solved. Or, another way to look at it: dramatic preprocessing is putting a "knowledge engineering approach" front end on the machine learning system. How can one be sure that this preprocessing is not altering the original problem in such a manner as to make its solution more difficult or less robust?

Moreover, most dimensionality reduction methods (with the exception of stemming and stoplisting) are usually extremely time consuming. Also, many forms of preprocessing (especially attribute creation) often result in a knowledge base that can no longer be tied back to meaningful textual features (such as specific words), thereby making it difficult if not impossible to obtain human-comprehensible rules from the knowledge base. Quite a few preprocessing methods are domain-specific, so one is possibly adding domain restrictions by including such methods in the overall system. However, preprocessing that to some degree reduces dimensionality is certainly the rule rather than the exception in current approaches to text categorization.

The methods we use to tokenize do not use stop lists or stemming or even a table lookup. The documents are viewed as a stream of characters which are typically broken into tokens at white space and punctuation. This is fast, takes little memory, and results in a representation that can easily be tied back to the original document.

3. efficient space and time complexity

We want to develop a method that is both space and time efficient, by which we mean typically $O(n)$, where n is a measure of the input size. This applies to both the learning and the prediction processes. Naturally we want the system to "run fast" and "not use much memory". But efficient use of time and space are also, as a practical matter, partially dictated by our doing minimal preprocessing. We must deal directly with a very high dimensional domain. For example, some text categorization methods, operating on the Reuters-22173 corpus, have only 300 - 500 tokens in the feature vector once preprocessing has occurred. Our system uses all tokens (approximately 50,000).

A comment is in order since many methods (ours included) use a 2 step approach to solving the text categorization problem. The first step does "preprocessing" and the second step does the "learning". It is often this second step which is analyzed in detail and which is usually the most interesting. But to evaluate the overall system, one must include an examination of what the preprocessing did to the original data and the resources it took to do it. The modifications made to the documents by the preprocessor should be detailed so that one can better judge aspects of the overall system such as [1]the degree to which the preprocessor modified the original problem, and [2]how specific to the particular corpus being used the preprocessing method is. Also, it is helpful to know the computational complexity of the 2 steps. Having $O(n^3)$ preprocessing followed by $O(n)$ for learning is not the same as having the entire process $O(n)$. Preprocessing in most systems (ours included) is only done once for a particular document collection, so in some ways one is justified in not dwelling too much on its complexity. However, in situations where the document collection is both large and changing often, having efficient preprocessing becomes more important.

4. general method

Our goal was to develop methods that are general and that will work on a variety of types and sizes of documents. In this regard, we have tried to avoid methods that seem a bit too "contrived". Most machine learning algorithms have various options as regards the choice of parameter values, methods used for computing various values, etc. One must be very careful not to use methods in which one can inadvertently make the method specific to just one problem (or to a small group of problems) simply by the choice of parameter values. We have avoided methods that would be hard to justify in terms of answering "why's", in an effort to develop methods that are more defensible and we hope better able to solve large classes of problems.

5. obtain same accuracy as supervised methods

   We want to develop text categorization methods that do all of the above, but without sacrificing accuracy.

Our goals may appear overly-optimistic. We want to find text categorization methods that are as good as or better than existing methods when comparisons are made using run time, the number of training examples used, and contingency table based performance measures. Elapsed processor time and the number of training examples used are for us the main driving forces behind our research. We want to have a system that runs very fast and that uses fewer training examples. It is important to keep in mind that there are conflicts encountered while developing any system. Decisions need to be made as to the relative importance of various performance measures. One would like to do well in all of the performance measures. However, if that is not possible, one must then make difficult choices. In order to get fast execution speed and low training example use, we may have to make sacrifices in terms of other performance measures. Not too much, though – obviously a system that runs very fast and uses few training examples but provides little useful information is not desired. Most published results on text categorization research do not report computational complexity or the elapsed processor time for the preprocessing or for the test runs. This in a sense places us at a disadvantage in terms of making comparisons, because we are emphasizing performance measures that are not considered as important by other researchers and therefore usually not reported by them. But yet we may have to make some sacrifices in the measures that are reported by others in order to reach our goals.

4.2 Overall Approach

From the vector space model we borrow the idea of representing documents as vectors. Each document is represented as a boolean vector, with each component indicating whether or not the corresponding token (word) is in that document. The length of the vector is the number of unique tokens in the entire document collection. Conceptually, think of each document as being represented by a data point in an n dimensional space. This is similar to the document representation used in the vector space model – we are using a "booleanized" version of the vector space model's data representation.

The Active Learning with Committees (ALC) approach is defined as follows. One uses a finite committee of learners, each learner representing a different hypothesis. Since the learners represent different hypotheses, they will at times disagree on how a particular example should be classified. Active learning is achieved by using this disagreement amongst the members to decide whether or not to see training example labels. Learning occurs by changing some of the hypotheses. Classification is performed by combining the individual member predictions to form a prediction of the committee as a whole. Therefore, ALC consists of three basic parts:

1. deciding whether or not to see the label
2. learning
3. forming the committee prediction

As there are several methods available for each of these three parts, one can create a very large number of different ALC systems. We examined a great many of these systems in our research. We will first give an overview of the general approach used for each of these three aspects of ALC, and then go into more detail about each of the techniques used.

4.2.1 Deciding Whether or Not to See the Label

Like the Query by Committee (QBC) approach, we use a committee of learners. The committee is composed of a finite number of members, each of which uses the same learning algorithm but which in general is initialized differently, thus insuring that the members at least initially behave differently from one another. The learning methods we have developed scale up well with the number of members since execution time and memory demands are linear in the number of members. There is a tradeoff between having a small committee (fast execution, low memory requirements, but not many "different opinions") and a very large committee.

When an unlabeled example is presented to the committee, the committee as a whole needs to decide whether or not to ask to see the label. There are several ways that the committee can do this. In general, some or all of the individual members of the committee make a prediction on the example – that is, they predict the label based on the feature vector and on their current knowledge. Those predictions are then in some manner converted into a decision by the committee as a whole on whether or not to see the label. If the label is seen ("example is used"), then the example with its label is presented to each member for learning. However, depending on the characteristics of the learning algorithm used by the individual members, the state of each member, and the example, in general some learners will learn from the example and some will not.

4.2.2 Learning

We chose perceptron, winnow, and naive Bayes as the main learning algorithms used with ALC. Conceptually, each of these methods consists of a set of weights which defines a hyperplane through the space of documents. The weights are adjusted during learning in an effort (by each committee member) to position the hyperplane so as to separate the documents that are in the category from those that are not. As the weights change, the hypothesis represented by each learner changes. All three of these algorithms are in a class of algorithms called "linear threshold algorithms". This term comes from the fact that the algorithm is used to predict one of two values (in our case YES or NO) based on comparing a linear function of the attribute values to a threshold value.

Perceptron and winnow learners are, as machine learning algorithms go, better able than most to handle large numbers of attributes, noise, and large percentages of irrelevant attributes. As an added benefit, they are simple algorithms (in time complexity, space complexity, and conceptually). Both are also mistake-driven.

The naive Bayes learning algorithm, in spite of its somewhat pejorative name and the seemingly quite drastic nature of the assumptions it makes, does quite well in classification, and in many different domains. The standard naive Bayes learning algorithm is a supervised learning algorithm and is normally used in batch mode (more on this later). We have developed from this standard naive Bayes algorithm a learning algorithm that uses a committee of naive Bayes learners, is incremental, and can be used in active learning.

A committee of learners (perceptron, winnow, or naive Bayes) has its members initialized randomly – meaning that the weights for each member are set to some random value. As this committee of learners is presented with unlabeled examples, it decides which examples to use, labels are provided for those examples that are to be used, and the weights are changed as learning occurs. The main differences among the 3 learning algorithms we use is the conditions under which an example whose label has been requested actually causes an adjustment in the weights and how the weights are actually modified. These differences (and others) are characteristic of the standard supervised form of each algorithm, and so will be covered in detail in the following section where we cover the specifics of each algorithm. Briefly, perceptron and winnow are mistake-driven learning algorithms – they only adjust their weights if their prediction on the training example was incorrect. Naive Bayes always adjusts its weights, whether its prediction on the training example was correct or incorrect. Perceptron modifies its weights additively, winnow modifies its weights multiplicatively, and naive Bayes modifies its probabilities by additively modifying the numerator and denominator of a ratio (weights are a function of these probabilities).

4.2.3 Forming the Committee Prediction

At any time, the committee may be used to predict categories for previously-unseen documents. To form the prediction of the committee, one in general has some or all of the individual members of the committee make a prediction, and then those individual predictions are combined in some manner to form a single prediction for the entire committee. Generally we use majority voting to determine the prediction of the committee as a whole.

4.2.4 ALC Example

There are many methods that can be used for each of the 3 basic components of an ALC system. A concrete example describing one specific choice for each of the 3 components may be helpful in clarifying the ALC concept. Say we have a committee composed of 8 learners:

1. *deciding whether or not to see the label*:

   When an unlabeled example is presented to the committee, each member predicts the value of the label. If there is an even split (4 YES, 4 NO), then we assume that this example is very informative, since half of the committee is getting it wrong. The label is requested (and so the example is used) whenever the vote is an even split.

2. *learning*:

   The committee members learn from the complete example (i.e., the example with the label) using the winnow algorithm.

3. *forming the committee prediction*:

   The individual committee members predict, and majority vote determines the prediction of the committee. In case of a tie vote, randomly predict YES or NO.

4.2.5 Terminology

In an ALC system, a candidate training example goes through several steps before it actually impacts learning. We have developed a specific terminology to make this process clear:

1. A training example is "selected" if the system determines that it is an informative example. The code that decides whether or not to see the label is deciding which training examples are to be selected.

2. A selected training example is "used" when the system requests its actual label. The number of training examples used is one of the main performance measures we are interested in, since it is an indication of how much work the teacher has to do. Usually (but not always), the label is requested so that the complete training example (attributes and label) can be passed to the learner. In such cases, we might say "used for learning" to make the description flow more smoothly.

3. A selected and used training example is "learned from" when actual changes in the knowledge base of the learner are made in response to that example. It is very important to note the distinction between an example that is used and one that is learned from. As we shall see, some learning algorithms may ask for the label (and thus use the example) but then decide not to learn from it.

4. We also make a distinction between examples in the corpus that are "unused" versus "unavailable". An unused example is a training example that was not used by the learner – that is, its label was not requested. An unavailable example is an example in the corpus that was excluded from both the training and test sets.

One final comment on the above terminology: for most methods used to decide whether or not to see the label, selection (deciding that the example is informative) and use (requesting the label) occur at the same time, and so in those cases we refer to the example being used (with selection being implied). For the bagging and boosting methods, however, selection and use occur at different times.

4.3 Main Techniques Used

4.3.1 Supervised Learning

We used the following three learning algorithms in our research: perceptron, winnow, and naive Bayes. All 3 algorithms were originally developed as supervised learning algorithms – meaning that each algorithm uses all examples in the training set. Supervised learning, since it uses all of the training data, provides us with our "baselines" – the methods to which we will compare the results of other methods. One of our main goals is to be able to categorize text as well as supervised learning methods while using far fewer training examples. If the method we are testing uses multiple epochs, then we will allow the supervised learning algorithm to use the same number of epochs. Similarly, the committee of supervised learners has the same number of members and the members are initialized in the same manner as is the case for the ALC method we are testing.

In an n dimensional space, the equation of a hyperplane is $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$, equivalently written as $\vec{a} \cdot \vec{x} = b$. A 1-dimensional hyperplane is a point, a 2-dimensional hyperplane is a line, a 3-dimensional hyperplane is a plane. We will refer to $\vec{a}$ as a vector of weights, and the b value as the threshold. When one "learns" a hyperplane, one is determining the values of $\vec{a}$ and b so that the hyperplane separates the $\vec{x}$ that are YES from the $\vec{x}$ that are NO. A useful fact about hyperplanes is that $\vec{a}$ is orthogonal to the hyperplane. This in fact comes from the usual definition of orthogonality. Two vectors $\vec{a}$ and $\vec{x}$ are orthogonal to each other if $\vec{a} \cdot \vec{x} = 0$ [Lay 1993]. An easy way to see that this means that $\vec{a}$ is also orthogonal to the hyperplane is to consider a hyperplane passing through the origin, say $\vec{a} \cdot \vec{x} = 0$. Then by the definition of orthogonality, $\vec{a}$ and $\vec{x}$ are orthogonal to each other. But the $\vec{x}$ are vectors drawn from the origin to any point in the hyperplane. Since $\vec{a}$ is orthogonal to all such $\vec{x}$ and all such $\vec{x}$ are what define the hyperplane, $\vec{a}$ is orthogonal to the hyperplane. While the hyperplanes in our domain do not pass through the origin, they can be thought of as such a hyperplane that has had its coordinates translated, and we will see later that this does not affect vector size or direction and so does not affect this result.

A hyperplane is a mathematically simple structure (no cross products of coordinates, no exponents) and so is attractive since it is computationally fast to manipulate. All of the learning algorithms we are using represent their hypotheses using a hyperplane. Each algorithm attempts to position this hyperplane in the space of document data points so that the points representing all of the documents that *are* in the specified category lie on one side of the hyperplane, and the points representing all of the documents that are *not* in the category lie on the other side of the hyperplane. The learning algorithm in effect tries to separate the points into two clouds, one containing only YES documents and one containing only NO documents. Often the clouds of YES and NO points are not separable with a hyperplane, in which case the algorithm tries to find a hyperplane that does the best job it can of separating the clouds.

Two of the learning algorithms that we use, perceptron and winnow, attempt to determine this "best position" for the hyperplane by starting with a hyperplane that cuts through the points in a random way and then adjusting its location so that the hyperplane separates the two clouds. This effort will usually be easier if the two clouds are some distance apart. The distance between the clouds is proportional to a "spacing factor" called δ (delta) which is in (0,1) [Littlestone 1991]. If the clouds happen to overlap, then δ is a measure of the distance between the best non-overlapping portions of the clouds. The rate at which one moves the hyperplane when trying to find its best position (the "learning rate") usually varies as δ varies. For example, if the two clouds are quite close together, then one needs to move the hyperplane a very small amount each time, and so a low learning rate is employed. Otherwise one will repeatedly overshoot the area between the clouds and essentially end up with the hyperplane oscillating between positions *within* the main bodies of the two clouds. Since both algorithms allow the use of multiple epochs, one might adopt the strategy of making each movement of the hyperplane very small. However, if the learning rate is small but δ is in fact quite large, then such a strategy will result in learning that is unacceptably slow.

One final comment on δ. While it is a measure of the distance between the 2 clouds, we also use it as an input parameter to our text categorization system. Different algorithms then use this δ value, usually in different ways, to compute a learning rate that is proportional to δ. This approach avoids our having to give as parameters the learning rate for each different algorithm in use. Instead we give one parameter based on our understanding of the corpus as a whole, a relative measure of how far apart we think the 2 clouds are, and then each learning algorithm is responsible for using that information to compute any internal learning rate parameters it needs. The value of δ is a characteristic of the corpus and how it was preprocessed, and is usually found through trial and error. Determining the best value of δ is referred to as tuning and is an accepted fact of life for algorithms that have parameterized learning rates. One does not, however, want to use δ to dial in the correct solution to specific problems. The value of δ that is used must be the same for all uses of a particular corpus. Also, one wants systems for which the best value of δ is actually a range of values. Since it is not practical to actually compute δ, one would like to have the system be robust to variations in δ. In this way, one is more confident that the specific value of δ is not playing too major of a role in the quality of the system's performance, and one is also more confident that the system will behave reasonably well on other corpora without modifying δ.

Note that the supervised naive Bayes learning algorithm is different in this regard. As we will see, the method used to compute the weights that define the separating hyperplane in the naive Bayes algorithm does not directly involve any "learning rate" parameter. This in fact is one of the strengths of the supervised naive Bayes method – one does not need to specify a "learning rate". This is an advantage since in general one does not know δ and it is usually not possible to compute it in a reasonable amount of time.

A few comments are probably in order as regards methods which we can not use, as a practical matter, because of the nature of the domain and/or our performing minimal preprocessing. Most mathematical optimization or approximation methods that one often thinks of (linear programming, multivariate linear regression, etc.) take time $O(n^3)$ or worse. Many also require that there in fact be a solution or feasible region. Many are also intolerant of noisy data. The document collections we deal usually have both attribute and class noise, and they often have inconsistent categories and so there is often no feasible region. And with the number of attributes we are dealing with, complexity much greater than $O(n \log n)$ is (as we shall see) barely tolerable, and then only if the constant of proportionality is very small. Finding the best linear separator (one that minimizes the number of incorrect classifications) is known in general to be NP-hard [Höffgen and Simon 1992], so computing the best solution for all cases is not practical in our domain, and probably not desirable either (due to noise).

4.3.1.1 Winnow

We use a standard version of winnow – WINNOW2 in [Littlestone 1988], with some modifications from [Littlestone 1991]. We will hereafter refer to our

1. Obtain the next example, which consists of attributes $\vec{x}$ and the value of the actual label.
2. If $\vec{w} \cdot \vec{x} > \theta$ then predict YES, otherwise predict NO
3. If the actual label and the predicted label are different, then learning occurs.
   a. if the actual label is YES and the predicted label is NO, a promotion is performed:

   $$\text{for } i = 1, n \quad w_i = w_i \times \alpha^{x_i}$$

   where $\alpha > 1$

   b. if the actual label is NO and the predicted label is YES, a demotion is performed:

   $$\text{for } i = 1, n \quad w_i = w_i \times \beta^{x_i}$$

   where $0 < \beta < 1$

Figure 6. Winnow Learning Algorithm

algorithm as "winnow". The winnow algorithm assigns initial weight values and then adjusts those weights during learning, at a rate determined by two parameters α and β. In this version of winnow, the threshold value (θ) is not changed as learning occurs.

Figure 6 outlines the steps used in winnow for learning. Each document is represented by $\vec{x}$, a boolean vector of length n, where n is the number of unique tokens in the entire document collection. $\vec{w}$ is a vector of n real-valued weights $(w_{1,} w_{2,} \cdots, w_n)$. It can also, in conjunction with the constant θ, be viewed as defining a hyperplane whose equation is $\vec{w} \cdot \vec{x} = \theta$. We discuss later how the value of θ and the initial values of $\vec{w}$ were chosen in our experiments.

Prediction is done by computing whether the point defined by $\vec{x}$ is above or below the current location of the hyperplane. If it is above the hyperplane, then predict YES, otherwise predict NO. To form the prediction, $\vec{w} \cdot \vec{x}$ is computed and compared to the threshold value θ. If the dot product is greater than θ, then the document is predicted to be in the category, otherwise it is predicted not to be in the category.

The winnow algorithm learns by adjusting the weights so that the hyperplane separates the documents that are in the category from those that are not. The weight adjustment is done using the multiply operation, so this algorithm uses multiplicative updating. Note that the weights are adjusted only if the actual and predicted labels are different. Since learning occurs only if the learner is in error, this algorithm is mistake-driven. Also note (since $\alpha^0 = 1$ and $\beta^0 = 1$) that only weights corresponding to attributes whose values are 1 are actually changed.

Weight increases are called promotions and tend to move the hyperplane closer to the origin. Promotions occur when the actual label is YES but the predicted label is NO. Therefore to get that point above the hyperplane where it presumably belongs, the hyperplane needs to be moved towards the origin. This is done by increasing the weights along those dimensions for which $x_i = 1$. Recall that $\vec{x}$, the feature vector, is a boolean vector, so each element is either 0 or 1. Thus changes in w_i only occur when x_i is 1. Similarly, demotions tend to move the hyperplane away from the origin by decreasing the weights along those dimensions for which $x_i = 1$.

The initial values of the weights and the threshold θ must be greater than 0. The weight values will remain greater than 0 due to the updating being multiplicative. This therefore limits the patterns that can be learned by winnow to those that can be represented using a separating hyperplane defined by positive weights. That is, the learning is based on those attributes that *are* in the document. This seems intuitively how a human classifies documents – by aggregating the impact of the words that are in the document (versus penalizing by those words that are not in the document).

4.3.1.2 Perceptron

We use a standard form of the perceptron learning algorithm. There are many similarities between the winnow and perceptron learners. Common features are:

1. Both algorithms maintain a hypothesis as a set of weights.
2. Each document class is learned separately.
3. A document is classified as being in the category of interest (YES) if the dot product of the weight vector and the feature vector that represents the document is greater than a threshold value (θ).
4. Both algorithms update ("learn") only when a document is misclassified.
5. During learning, only weights corresponding to attributes whose value is 1 are actually changed.

The main difference between perceptron and winnow lies in the way the weights are updated. The winnow learner uses multiplicative updating (see Figure 6). However, the perceptron learner adds (subtracts) a small constant to the weight corresponding to each $x_i = 1$ if a YES (NO) document is classified incorrectly. Thus the perceptron learner uses additive updating of weights. Additional differences are [1]the perceptron also updates θ when learning occurs, and [2]perceptron weights can be negative. Figure 7 outlines the steps in the perceptron learning algorithm.

1. Obtain the next example, which consists of attributes $\vec{x}$ and the value of the actual label.
2. If $\vec{w} \cdot \vec{x} > \theta$ then predict YES, otherwise predict NO
3. If the actual label and the predicted label are different, then learning occurs.
    a. if the actual label is YES and the predicted label is NO, a promotion is performed:

    $$\text{for } i = 1, n \quad w_i = w_i + \alpha^{x_i}$$

    $$\theta = \theta - \alpha$$

    where $\alpha > 0$

    b. if the actual label is NO and the predicted label is YES, a demotion is performed:

    $$\text{for } i = 1, n \quad w_i = w_i - \alpha^{x_i}$$

    $$\theta = \theta + \alpha$$

    where $\alpha > 0$

Figure 7. Perceptron Learning Algorithm

Because the weights in the perceptron learner are changed additively, they can be either negative or positive or zero. Thus the weights and θ can be initialized to any value (positive, negative, or zero). This in turn means that there are no restrictions on the type of hyperplane that can be represented using perceptron weights. (Recall that all winnow weights are positive.) Therefore, any linearly separable data set can be perfectly learned using a perceptron learner, and it turns out that this can be accomplished in a finite number of steps. This result is usually referred to as the Perceptron Learning Theorem and is due to [Rosenblatt 1962]. This sounds much more wonderful than it is in practice, since in text categorization we have significant amounts of noise, we do not know the value of δ and so can not compute the value of α, and often the documents are not linearly separable. Because winnow adjusts its weights by an amount proportional to the current weight values, whereas the perceptron always adjusts the weights by the same amount, one can see that there will be significant granularity differences between the two algorithms as learning progresses, and this in turn may impact their comparative behaviors.

4.3.1.3 Naive Bayes

The naive Bayes algorithm works by looking at how the probability of each possible x_i value correlates with each possible class value. In our research, the x_i are boolean and so are either 0 or 1. Also, the class value is boolean – its value is either YES or NO.

Figure 8 outlines the steps used in the naive Bayes learning algorithm, as we have implemented it. As an example, consider how the algorithm works assuming that it makes one pass through all of the training data. In step #1, it initializes all of the counts and totals to 0 and then goes through the training examples, one at a time. For each training example, it is given the feature vector $\vec{x}$ and the value of the label for that document, k. The algorithm goes through the feature vector and increments $c_{i,k}$ for each x_i that is 1. It then increments t_k. In step #2, these counts and totals are converted to probabilities $p(x_i = j|k)$ by dividing each count by the number of training examples in class k. The final step (#3) computes the prior probabilities $p(k)$ as the fraction of all training examples that are in class k. The resulting conditional and prior probabilities are the knowledge base for the naive Bayes algorithm – the result of the learning.

Note that the naive Bayes learning algorithm is not mistake-driven. It learns by adjusting the counts $c_{i,k}$ and totals t_k, which in turn allows one to compute the probabilities $p(x_i = j|k)$ and $p(k)$. Thus each and every attribute of each and every

1. From the training examples, compute the values of $c_{i,k}$ ("count" of the number of examples in class k that have $x_i = 1$) and t_k ("total" number of training examples labeled k).
2. Compute the $p(x_i = j \mid k)$ values; $p(x_i = j \mid k)$ is the conditional probability that $x_i = j$ when the class label is k:

$$p(x_i = 1 \mid k) = \frac{c_{i,k}}{t_k}$$

$$p(x_i = 0 \mid k) = 1 - p(x_i = 1 \mid k)$$

3. Compute the p(k) values; p(k) is the prior probability that the label is k, based on seeing only the training examples:

$$p(\text{NO}) = \frac{t_{\text{NO}}}{t_{\text{NO}} + t_{\text{YES}}}$$

$$p(\text{YES}) = \frac{t_{\text{YES}}}{t_{\text{NO}} + t_{\text{YES}}}$$

Figure 8. Naive Bayes Learning Algorithm

training example will impact at least some of the $p(x_i = j|k)$ and $p(k)$ values, and thus the algorithm learns from all attributes of all training examples. This is in contrast to winnow and perceptron, which do not learn (adjust weights) unless their prediction on the training example was in error, and *even then*, they do not adjust weights except for attributes whose value is 1.

The naive Bayes algorithm is traditionally used in "batch mode", meaning that the algorithm does not perform the majority of its computations after seeing each training example, but rather accumulates certain information on all of the training examples and then performs the final computations on the entire group or "batch" of examples. The algorithm typically performs step #1 for all training examples, and *then* performs the remaining steps only *once*. However, note that there is nothing inherent in the algorithm that prevents one from using it to learn incrementally. In other words, one can initialize the counts $c_{i,k}$ and the totals t_k, and then execute all of the steps in Figure 8 on each training example, updating the counts and totals, and then computing new values of the conditional and prior probabilities. Obviously computing the various probabilities after each training example instead of doing it just once after having seen all training examples will take more time. Doing it in this manner does make it possible to compare the performance of naive Bayes to other algorithms *during* the learning process, and this can in turn be of interest if one is investigating a situation in which the user can halt learning at any time (before the entire training set has been used) and request that the system proceed with prediction. However, it is important to realize that such an incremental mode of operation does *not* affect the results that are obtained. In other words, the knowledge base that results from the incremental mode after n training examples have been used is exactly the same as one would have if the algorithm were used in its more traditional single-batch mode on those same n training examples. The fact that the incremental version learns as it goes does not affect what is learned or not learned from each example. This is decidedly not the case for the winnow and perceptron algorithms. Since they are mistake-driven, the order in which the training examples are used affects the learned weights.

We next consider how the naive Bayes method uses the probabilities it has computed in order to classify previously unseen examples. Given a particular document $\vec{x}$, we would like to compute $p(\text{YES}|\vec{x})$ and $p(\text{NO}|\vec{x})$. We could then compare these two probabilities, with the one that is larger indicating which class label value is more likely to be the actual label. We will use $\Re$ to stand for the probability ratio that we want to compute. Then:

$$\Re = \frac{p(\text{YES}|\vec{x})}{p(\text{NO}|\vec{x})} \begin{array}{l} > 1:\ \textit{predict}\ \text{YES} \\ \\ \leq 1:\ \textit{predict}\ \text{NO} \end{array} \tag{1}$$

We have, from the learning phase of the naive Bayes algorithm, the values of the $p(x_i = 0|k)$ and $p(x_i = 1|k)$, so we might expect that we can compute $p(\vec{x}|k)$. However, for our classification task we need to compute $p(k|\vec{x})$. Bayes Theorem allows us to perform the necessary inversion:

$$p(k|\vec{x}) = \frac{p(\vec{x}|k)\ p(k)}{p(\vec{x})}$$

Therefore:

$$\Re = \frac{p(\text{YES}|\vec{x})}{p(\text{NO}|\vec{x})} = \frac{p(\vec{x}|\text{YES})\ p(\text{YES})}{p(\vec{x}|\text{NO})\ p(\text{NO})}$$

We have the values of p(NO) and p(YES) from the learning phase. To see how we can obtain the $p(\vec{x}|k)$ values, first consider that from the definition of conditional probability, we can express the conditional probabilities that we want to compute in terms of joint events [Pearl 1988]:

$$p(\vec{x}|k) = \frac{p(\vec{x}, k)}{p(k)}$$

Also note that p($\vec{x}$, k) can be written in component form as $p(x_{1,} x_{2,} \cdots, x_{n,} k)$.

Using the chain rule for probabilities [Pearl 1988]:

$$p(x_{1,} x_2, \cdots, x_n, k) =$$

$$p(x_1|x_2, \cdots, x_n, k)\ p(x_2|x_3, \cdots, x_n, k) \cdots p(x_{n-2}|x_{n-1}, x_n, k)\ p(x_{n-1}|x_n, k)\ p(x_n|k)\ p(k)$$

To simplify this expression, consider [Pearl 1988] that the event A is conditionally independent of event B given event C if:

$$p(A|B, C) = p(A|C)$$

If we assume that, given k, each of the x_i is conditionally independent of the rest of the attributes, then we can simplify the above chain rule equation considerably. This in fact is the "naive" assumption that gives this approach its name. If, given k, each x_i is conditionally independent of the rest of the attributes, then each of the $p(x_i|x_{i+1}, x_{i+2}, \cdots, x_n, k)$ terms in the above chain rule expansion is equal to $p(x_i|k)$. Therefore the above chain rule result becomes:

$$p(x_{1,}x_2, \cdots, x_n, k) = p(x_1|k)\, p(x_2|k) \cdots p(x_{n-2}|k)\, p(x_{n-1}|k)\, p(x_n|k)\, p(k)$$

Therefore, the ratio $\Re$ that we want to compute can be calculated thusly:

$$\begin{aligned} \Re &= \frac{p(\vec{x}|\text{YES})\, p(\text{YES})}{p(\vec{x}|\text{NO})\, p(\text{NO})} \\ &= \frac{p(\vec{x}, \text{YES})}{p(\vec{x}, \text{NO})} \\ &= \frac{p(x_{1,}x_2, \cdots, x_n, \text{YES})}{p(x_{1,}x_2, \cdots, x_n, \text{NO})} \end{aligned} \tag{2}$$

$$= \frac{p(x_1|\text{YES})\, p(x_2|\text{YES}) \cdots p(x_{n-2}|\text{YES})\, p(x_{n-1}|\text{YES})\, p(x_n|\text{YES})\, p(\text{YES})}{p(x_1|\text{NO})\, p(x_2|\text{NO}) \cdots p(x_{n-2}|\text{NO})\, p(x_{n-1}|\text{NO})\, p(x_n|\text{NO})\, p(\text{NO})}$$

$$\Re = \frac{p(\text{YES}) \prod_{i=1}^{i=n} p(x_i|\text{YES})}{p(\text{NO}) \prod_{i=1}^{i=n} p(x_i|\text{NO})} \tag{3}$$

Note that $p(x_i|k)$ in equation (3) represents $p(x_i = 1|k)$ if the i^{th} attribute of the test example (x_i) is 1 and $p(x_i = 0|k)$ if it is 0.

We have, from the learning phase of the naive Bayes algorithm, the values of $p(k)$, $p(x_i = 0|k)$, and $p(x_i = 1|k)$, for $i = 1,n$ and for k = NO or YES, so we can perform the above computation and thereby compute the above ratio $\Re$. We then classify $\vec{x}$ as YES if $\Re > 1$ and NO if $\Re \leq 1$.

Also note that since we really only need to know the relative sizes of the numerator and the denominator in the right hand side of equation (3), we do not have to actually compute the specific value of $\Re$.

The naive Bayes classification method is quite granular, in the sense that relatively large changes in the underlying training data can result in one obtaining the same prediction on the test examples. One essentially computes the values of $p(\vec{x}, k)$ for the various values of k, finds the largest $p(\vec{x}, k)$, and then chooses that value of k as the predicted class value. Thus, while the "naive" assumption made by this method may be quite drastic for some domains, the method will still classify correctly as long as the net error in the various $p(\vec{x}, k)$ values does not result in choosing a k different from the actual class label [Domingos and Pazzani 1996]. This also means that the naive Bayes classifier is quite robust and stable, in that departures from the naive assumption can be quite large and one can still obtain the correct classification.

In the actual implementation of the naive Bayes algorithm, we encountered difficulties in two areas. First, since the Bayes classification algorithm uses a product operation to compute the probabilities $p(\vec{x}, k)$, it is especially prone to being unduly impacted by probabilities of 0. This can occur both for $p(k)$ and for $p(x_i = j|k)$ values. As an example of the former, one might not have any examples in the training set that are categorized as YES, so $p(\text{YES})$ will be 0, and so $p(\vec{x}, \text{YES})$ will be 0. This can occur when one is dealing with categories that have very few positive examples in the corpus – the training set may well contain only negative examples.

Similarly, it is not unusual for some tokens to not appear at all in training documents that are in a particular category. For example, the word "pickle" might never occur in training documents that are about skiing. This would result in the count ($c_{<pickle>,<skiing=yes>}$) being 0, and so the corresponding probability ($p(x_{<pickle>} = 1 \mid < skiing = yes >)$) would also be 0. During classification, this will cause the probability $p(\vec{x}, < skiing = yes >)$ to be 0, regardless of the other tokens in the document. In other words, because there were not any training documents containing pickle that were about skiing, the classifier will always predict NO when asked about skiing if the document does contain the word "pickle" – no matter what other words (like "skiing"!) are in the document.

Either of these situations [$p(k) = 0$ or $p(x_i = j|k) = 0$] can occur as a result of the fact that our training sets are necessarily of limited size and certainly not all-encompassing. To avoid this problem, we use two methods. First, we use the m-estimate method for computing conditional probabilities. And secondly, we employ certain limit checks.

An approach that partially addresses this problem of zero probabilities is to use the m-estimate method [Mitchell 1997], also referred to as the beta distribution method [Cestnik 1990, Neapolitan 1990]. In step #1 of Figure 8, one is computing counts $c_{i,k}$ and totals t_k. The "straightforward" approach is to initialize these counts and totals to 0 before any training documents are seen. Then, as training documents are examined, the appropriate counts and totals are incremented. This initialization of $c_{i,k}$ and t_k to 0 is in fact the main source of these zero probability difficulties, since any $c_{i,k}$ that remains 0 after all of the training documents have been used will result (step #2 of Figure 8) in the corresponding $p(x_i = 1|k)$ being 0. Similarly, any t_k that remains 0 will result in the corresponding $p(k)$ being 0 (in step #3). In the m-estimate method, one in effect initializes the counts and totals to non-zero values, based on some prior assumptions and on ones confidence in those assumptions. The idea behind these methods is that initializing the $c_{i,k}$ and t_k to 0 makes no assumptions about the data. If one does in fact have some prior domain knowledge, then that knowledge can be included in the probability computations if one uses the beta distribution. The m-estimate method has two parameters $\hat{m}$ and $\hat{p}$. $c_{i,k}$ is initialized to $\hat{m} \times \hat{p}$ (instead of 0), and t_k is initialized to $\hat{m}$ (instead of 0).

For the value of $\hat{p}$, we want to use an estimate of the expected value of $p(x_i = 1|k)$. The list of all tokens that are used in the corpus is termed a "dictionary". Therefore $p(x_i = 1|k)$ is the probability that the i^{th} token in the dictionary appears in a document that is (k = YES) or that is not (k = NO) in the category of interest. Actually, since we have not seen any training examples yet, we have no way of distinguishing between an estimate for YES and NO, regardless of category, so we compute just one estimate, for $p(x_i = 1|k)$ for either k value and for any category. Say we have a corpus of documents and the average number of tokens in each document is £ (the average document "length"). The dictionary for the entire corpus contains Ð tokens. Note that the values of £ and Ð are easily obtained during preprocessing of the corpus and their computation does not involve examining the actual values of attributes or labels of any of the documents. Observe that if one has Ð tokens and is going to draw a token for filling the first position in a document, then the chance of drawing a particular token is $\frac{1}{Ð}$, and so the chance of *not* drawing that particular token is:

$$1 - \frac{1}{Ð} = \frac{Ð - 1}{Ð}$$

Then for a particular document (independent of category and independent of k), assuming that the occurrences of the tokens are independent of the other tokens in the document, the probability that a particular token (the i^{th} token) is *not* anywhere in the document is the probability that it is not in the first position times the probability that it is not in the second position times . . . etc. In other words, it is:

$$\left(\frac{Ð - 1}{Ð}\right)^{£}$$

Therefore, the probability that the i^{th} token *is* in the document is:

$$1 - \left(\frac{Đ - 1}{Đ}\right)^{£}$$

This value is the estimated value of $p(x_i = 1|k)$, and so is the value of $\hat{p}$ for the m-estimate method. For example, $\hat{p}$ computed in this manner for the Reuters-22173 corpus for titles only is 0.00047 (Đ = 16,600 and £ = 7.78424).

$\hat{m}$ corresponds to how certain one is that the chosen value of $\hat{p}$ is actually correct. If one were not very confident, then one would choose a small value for $\hat{m}$, thereby not greatly influencing the true counts or the true totals. If one were extremely confident, one might choose an $\hat{m}$ value that is quite large. Let h = the number of different values of k; in our case, $h = 2$. Then initializing $c_{i,k}$ to $\hat{m} \times \hat{p}$ and t_k to $\hat{m}$ is *exactly* what would have occurred on the average if, before seeing any training documents, one examined a "pre-training" set of $h \times \hat{m}$ documents that contained $\hat{m}$ documents in each class k and for which each $x_i = 1$ with probability $\hat{p}$ (thus resulting in a count of $\hat{m} \times \hat{p}$). For this reason, $\hat{m}$ is often referred to as an equivalent sample size.

This is a very general initialization method. Note that: [1]if $\hat{m} = 0$, then we get the aforementioned "straightforward" way of initializing $c_{i,k}$ and t_k, and [2]if $\hat{m} = 2$ and $\hat{p} = 0.5$, we get a method that is often used and which is usually referred to as "Laplace's Law of Succession". In our experiments, we generally used values of $\hat{m}$ in the range 0 - 10.

A final comment: the use of the m-estimate method can still result in very small or even 0 probabilities. Even if $\hat{m} > 0$, one can still have values of $p(x_i = 1|k)$ that are very small. Also, any of the $p(x_i = 1|k)$ can be 1 or close to it, in which case the corresponding $p(x_i = 0|k)$ will be 0 or very small. Therefore, one still needs to *also* take some action to handle small probabilities when computations are performed that involve multiplying large numbers of these probabilities together.

The method we use is as follows: if the computed value of a probability is 0, we set it instead to a small positive value – the idea being that the corresponding situation *could* occur ("nothing is impossible") ... it just happened not to occur in the training data that was used.

The other problem encountered during the actual implementation of the naive Bayes algorithm is due to the fact that the feature vector is very long, but usually not many of the attribute values are 1. This results in a great many 0 values for x_i in a typical $\vec{x}$. For example, in the Reuters-22173 corpus, the feature vectors are of length 48,446 (the number of unique tokens in the entire corpus), but the average document contains only about 82 different tokens. If the corresponding $p(x_i = 0 \mid k)$ are very small, then multiplying large numbers of them together can cause exponent underflow. Our solution to this second problem is to use the logarithms of the probabilities instead of the probabilities themselves. This also means that the multiplication done during classification (equation (3) on page 86) is replaced by addition. Note that this now looks very similar to the equation used by both winnow and perceptron to compute classification values. Each of those algorithms predicts by computing whether the point defined by the new document $\vec{x}$ is above or below the current location of the hyperplane. To do this, $\vec{w} \cdot \vec{x}$ is computed and compared to the threshold value θ. If the dot product is greater than θ, then the document is predicted to be in the category, otherwise it is predicted not to be in the category. By transforming equation (3) to use log probabilities instead of the probabilities themselves, we will obtain an evaluation equation for the naive Bayes classifier that looks very much like computing $\vec{w} \cdot \vec{x}$ and comparing it to θ.

To see this, let us modify equation (3) so that we perform the naive Bayes classification in a different but equivalent manner. We will instead compute a log difference. First, we define the binary operator $\gtrless$ ("compare") to indicate comparison and prediction. $a \gtrless b$ means that one compares the numeric values of a and b and predicts YES if $a > b$ and NO otherwise. Note that for any expression e,

$(a \gtrless b)$ and $(a+e) \gtrless (b+e)$ give the same result. As an example of the use of the $\gtrless$ operator, recall how both the winnow and perceptron algorithms compute the predicted label. They compute $\vec{w} \cdot \vec{x}$ and compare it to a threshold value θ. If the dot product is greater than θ, then the document is predicted to be in the category, otherwise it is predicted not to be in the category. Since $\vec{w} \cdot \vec{x}$ can be written as $\sum w_i x_i$, the prediction method for both the winnow and perceptron algorithms can be expressed as $\sum w_i x_i \gtrless \theta$.

Recall equation (1):

$$\begin{aligned} \Re &> 1: \textit{predict } \text{YES} \\ &\leq 1: \textit{predict } \text{NO} \end{aligned}$$

Since log is a strictly increasing function, $a > b \Leftrightarrow log(a) > log(b)$. Also, log(1) = 0. Therefore:

$$\begin{aligned} log(\Re) &> 0: \textit{predict } \text{YES} \\ &\leq 0: \textit{predict } \text{NO} \end{aligned}$$

Which, using the $\gtrless$ operator, becomes:

$$log(\Re) \gtrless 0 \tag{4}$$

Recall from equation (2):

$$\Re = \frac{p(\vec{x}, \text{YES})}{p(\vec{x}, \text{NO})}$$

Therefore, equation (4) becomes:

$$log(p(\vec{x}, \text{YES})) - log(p(\vec{x}, \text{NO})) \;\gtrless\; 0$$

This expression computes the difference between the 2 log probabilities, compares the result to 0, and forms a prediction which is exactly the same as would be predicted using equation (3).

We can compute $log(p(\vec{x}, \text{YES}))$ with the following (the numerators of equations (2) and (3) are equal; take log of both sides):

$$log(p(\vec{x}, \text{YES})) =$$

$$log(p(\text{YES})) + \sum_{i=1}^{i=n} \{(1 - x_i)\, log(p(x_i = 0 | \text{YES})) + (x_i)\, log(p(x_i = 1 | \text{YES}))\}$$

The (1-x_i) and (x_i) terms are used so that the appropriate log( ) term will be selected based on the actual value of x_i in the example being classified. Note that since x_i is 0 or 1, (1-x_i) is correspondingly 1 or 0.

Continuing . . .

$$log(p(\vec{x}, \text{YES})) = log(p(\text{YES})) + \sum_{i=1}^{i=n} \{log(p(x_i = 0 | \text{YES})) - log(p(x_i = 0 | \text{YES}))\, x_i + log(p(x_i = 1 | \text{YES}))\, x_i\}$$

$$= log(p(\text{YES})) + \sum_{i=1}^{i=n} log(p(x_i = 0 | \text{YES})) - \sum_{i=1}^{i=n} log(p(x_i = 0 | \text{YES}))\, x_i + \sum_{i=1}^{i=n} log(p(x_i = 1 | \text{YES}))\, x_i$$

Similarly for log($p(\vec{x}, \text{NO})$).

Therefore:

$$log(p(\vec{x}, \text{YES})) - log(p(\vec{x}, \text{NO})) \gtrless 0$$

becomes:

$$log(p(\text{YES})) + \sum_{i=1}^{i=n} log(p(x_i = 0 | \text{YES})) - \sum_{i=1}^{i=n} log(p(x_i = 0 | \text{YES}))\, x_i + \sum_{i=1}^{i=n} log(p(x_i = 1 | \text{YES}))\, x_i - log(p(\text{NO})) - \sum_{i=1}^{i=n} log(p(x_i = 0 | \text{NO})) + \sum_{i=1}^{i=n} log(p(x_i = 0 | \text{NO}))\, x_i - \sum_{i=1}^{i=n} log(p(x_i = 1 | \text{NO}))\, x_i \gtrless 0$$

This is an admittedly ugly-looking expression. The thing to note is that we basically have only two kinds of terms. One kind (the first, second, fifth, and sixth terms) contains only log probabilities, where these probabilities have been computed by the naive Bayes learning algorithm *and do not depend at all* on the particular example $\vec{x}$ for which we are trying to predict the label. The remaining terms contain these same types of log probabilities multiplied by x_i – these terms *do* depend on the particular example for which we are predicting. Gathering the terms with x_i factors on the left and those without on the right, we get the following expression:

$$\sum a_i x_i \gtrless b \tag{5}$$

where:

$$a_i = -log(p(x_i = 0|\text{YES})) + log(p(x_i = 1|\text{YES})) + log(p(x_i = 0|\text{NO})) - log(p(x_i = 1|\text{NO}))$$

$$b = -log(p(\text{YES})) - \sum_{i=1}^{i=n} log(p(x_i = 0|\text{YES})) + log(p(\text{NO})) + \sum_{i=1}^{i=n} log(p(x_i = 0|\text{NO}))$$

Since the naive Bayes classification expression $\sum a_i x_i \gtrless b$ is of the same form as the classification expression used by the winnow and perceptron algorithms ($\sum w_i x_i \gtrless \theta$), we have shown that the naive Bayes classifier, like winnow and the perceptron, can be thought of as attempting to position a hyperplane in such a manner as to separate the NO and YES documents [Duda and Hart 1973].

We discussed earlier why the perceptron can learn any linearly separable pattern and why winnow can not. How does naive Bayes perform in this regard? It turns out that there are linearly separable patterns that the naive Bayes algorithm can not learn perfectly. For example, consider the family of "at least m of n" concepts, in which the yes class is any m or more bits out of n being 1. Such

```
==== starting system # = 1 ===(perceptron, supervised)======

Committee contingency table:
                actual label value
                     0       1
predicted   0        29       0
  label
  value     1         0      99

committee accuracy = 1
committee precision = 1
committee recall = 1
weights: 0.18163 0.238773 0.295916 0.295916 0.324487 0.295916 0.324487
theta: 0.657143

==== starting system # = 2 ===(incremental naive Bayes, supervised) ======

Committee contingency table:
                actual label value
                     0       1
predicted   0         8       0
  label
  value     1        21      99

committee accuracy = 0.835938
committee precision = 0.825
committee recall = 1

j  k      . . . . . . . . . . . log(p(x[i]=j|k)) . . . . . . . . . . . .
         i=1       i=2       i=3       i=4       i=5       i=6       i=7
1  0  -0.6173   -0.6173   -0.6173   -0.6173   -0.6173   -0.6173   -0.6173
0  0  -0.119975 -0.119975 -0.119975 -0.119975 -0.119975 -0.119975 -0.119975
1  1  -0.23976  -0.23976  -0.23976  -0.23976  -0.23976  -0.23976  -0.23976
0  1  -0.372386 -0.372386 -0.372386 -0.372386 -0.372386 -0.372386 -0.372386
```

Figure 9. Solution for At Least 3 of 7 Problem

counting concepts are a concise way of describing rules used in medical diagnoses, where a patient having "at least *m* of *n*" symptoms is likely to have a certain disease [Spackman 1988]. The "at least 3 of 7" concept can not be learned by the naive Bayes algorithm, although it is a linearly separable pattern and in fact can be perfectly learned by a perceptron [Kohavi 1995].

Figure 9 shows excerpts from the output of our system on the "at least 3 of 7 problem". The first system in Figure 9 is a supervised perceptron learner. As can be seen from the resulting contingency table, the perceptron has perfectly learned the concept. The naive Bayes algorithm, however, obtained an accuracy of only 83.6%, since it incorrectly predicts YES ("1") for 21 examples which are actually NO ("0").

One final comment on the naive Bayes algorithm. Equation (5) is not generally the one used when the algorithm is actually in use as a classifier. Equation (5) is used when we want to compute a single weight for each token, but the more standard method of computing the classification using log probabilities is done as follows. Taking the log of both sides (recall that log is a strictly increasing function), equation (3) becomes:

$$log(\Re) = \left(log(p(\text{YES})) + \sum_{i=1}^{i=n} log(p(x_i|\text{YES})) \right) - \left(log(p(\text{NO})) + \sum_{i=1}^{i=n} log(p(x_i|\text{NO})) \right)$$

Actually, we do not really need to compute the value of log($\Re$). We just need to know which is larger, the first term on the right hand side or the second. If the first term is larger, then we predict YES, otherwise we predict NO. In other words, we want to compute:

$$\left(log(p(\text{YES})) + \sum_{i=1}^{i=n} log(p(x_i|\text{YES})) \right) \not\lessgtr \left(log(p(\text{NO})) + \sum_{i=1}^{i=n} log(p(x_i|\text{NO})) \right) \qquad (6)$$

This will give us the same classification decision as equation (3) on page 86, but using log probabilities.

A small example will help demonstrate how the naive Bayes algorithm works as represented both in equation (3) and (6), and also show when one might want to use the weighting equation (5). The example will also serve to demonstrate the specifics of how documents are converted to feature vectors during preprocessing. We will use a very small corpus and we will use only the document titles. We are interested in whether or not each document is "about pets". Five training documents have been submitted to a human indexer and labeled. Here are the document titles and the label given to each by the indexer:

doc#	*title*	*label*
1	Mean Dog Cat Lion	YES
2	Ugly Big Dog	YES
3	Big Yellow Cat	YES
4	Big Ugly Lion	NO
5	Big Yellow Mean Mean Mean Lion	NO

The dictionary for this corpus therefore contains the following 7 tokens (listed in no particular order): mean, ugly, dog, big, yellow, cat, and lion. If we use this ordering for creating our feature vectors, we will then have the following feature vectors for the corpus:

doc#	- - - - - - - - - - - - - - - - - *feature vector* - - - - - - - - - - - - - - - - -							
	. . . . *attributes (tokens)*							*label*
	mean	*ugly*	*dog*	*big*	*yellow*	*cat*	*lion*	
1	1	0	1	0	0	1	1	YES
2	0	1	1	1	0	0	0	YES
3	0	0	0	1	1	1	0	YES
4	0	1	0	1	0	0	1	NO
5	1	0	0	1	1	0	1	NO

The above 5 documents are used as input to the naive Bayes learning algorithm (ref. Figure 8), with counts and totals initialized to 0 (i.e., we are not using the m-estimate method). Naive Bayes will compute the following probability values:

probability	*mean*	*ugly*	*dog*	*big*	*yellow*	*cat*	*lion*	*label*
$p(x_i=1\|0)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{0}{2}$	$\frac{2}{2}$	$\frac{1}{2}$	$\frac{0}{2}$	$\frac{2}{2}$	
$p(x_i=0\|0)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{2}{2}$	$\frac{0}{2}$	$\frac{1}{2}$	$\frac{2}{2}$	$\frac{0}{2}$	
$p(x_i=1\|1)$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	
$p(x_i=0\|1)$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	
p(NO)								$\frac{2}{5}$
p(YES)								$\frac{3}{5}$

Recall that we are replacing probability values of 0 with "a small positive value". In this example, we will use for that value $\frac{0.01}{t_{\text{NO}} + t_{\text{YES}}} = \frac{0.01}{5} = 0.002$. Therefore, the above probability values become:

probability	*mean*	*ugly*	*dog*	*big*	*yellow*	*cat*	*lion*	*label*
$p(x_i=1\|0)$	0.5	0.5	0.002	1.0	0.5	0.002	1.0	
$p(x_i=0\|0)$	0.5	0.5	1.0	0.002	0.5	1.0	0.002	
$p(x_i=1\|1)$	0.33	0.33	0.67	0.67	0.33	0.67	0.33	
$p(x_i=0\|1)$	0.67	0.67	0.33	0.33	0.67	0.33	0.67	
p(NO)								0.4
p(YES)								0.6

Since we are using log probabilities instead of the probabilities themselves, we then have as the final results of the naive Bayes learner:

log probability	*mean*	*ugly*	*dog*	*big*	*yellow*	*cat*	*lion*	*label*
	. . . . *attributes*							
$\log(p(x_i=1\|0))$	-0.301	-0.301	-2.699	0.0	-0.301	-2.699	0.0	
$\log(p(x_i=0\|0))$	-0.301	-0.301	0.0	-2.699	-0.301	0.0	-2.699	
$\log(p(x_i=1\|1))$	-0.477	-0.477	-0.176	-0.176	-0.477	-0.176	-0.477	
$\log(p(x_i=0\|1))$	-0.176	-0.176	-0.477	-0.477	-0.176	-0.477	-0.176	
log(p(NO))								-0.398
log(p(YES))								-0.222

Once the naive Bayes learner has been trained using the above 5 documents, it needs to be tested. We might be tempted to show the degree to which it learned the 5 training examples by giving each of them to the system to see if the predicted label is correct. However, this is not a very good "test" in the usual sense, as we would be using the training data for testing. This is usually a bad idea, because the purpose of testing is to see how well the learner has generalized the knowledge in the training examples so that it can accurately predict on previously-unseen examples. If we use training examples for testing, we may only be testing the learner's ability to memorize. So we will test the above learner with the following 2 test documents:

doc#	*title*	*label*
6	Mean Ugly Cat	YES
7	Big Yellow Lion	NO

To decide, for example, the prediction for "Mean Ugly Cat" using equation (3), we would compute:

$$p(\text{YES})\prod_{i=1}^{i=n} p(x_i|\text{YES}) =$$

$$0.6 \times 0.33 \times 0.33 \times 0.33 \times 0.33 \times 0.67 \times 0.67 \times 0.67 = 0.00219$$

$$p(\text{NO})\prod_{i=1}^{i=n} p(x_i|\text{NO}) =$$

$$0.4 \times 0.5 \times 0.5 \times 1 \times 0.002 \times 0.5 \times 0.002 \times 0.002 = 4 \times 10^{-10}$$

Since 0.00219 is larger than 4×10^{-10}, the naive Bayes algorithm would predict YES for the document entitled "Mean Ugly Cat". We of course get the same prediction if we use equation (6):

$$log(p(\text{YES})) + \sum_{i=1}^{i=n} log(p(x_i|\text{YES})) =$$

$$-0.222 + (-0.477) + (-0.477) + (-0.477) + (-0.477) +$$

$$(-0.176) + (-0.176) + (-0.176) =$$

$$-2.658$$

$$log(p(\text{NO})) + \sum_{i=1}^{i=n} log(p(x_i|\text{NO})) =$$

$$-0.398 + (-0.301) + (-0.301) + 0 + (-2.699) +$$

$$(-0.301) + (-2.699) + (-2.699) =$$

$$-9.398$$

-2.658 is larger than -9.398, so naive Bayes would predict YES.

```
==== starting system # = 1 ===(incremental naive Bayes, supervised)=======

Number of training examples used = 5 (2-; 3+)

j  k     . . . . . . . . . . . log(p(x[i]=j|k)) . . . . . . . . . . . .
         i=1       i=2       i=3       i=4       i=5       i=6       i=7
1  0  -0.30103  -0.30103  -2.69897     0      -0.30103  -2.69897     0
0  0  -0.30103  -0.30103   0        -2.69897  -0.30103   0        -2.69897
1  1  -0.477121 -0.477121 -0.176091 -0.176091 -0.477121 -0.176091 -0.477121
0  1  -0.176091 -0.176091 -0.477121 -0.477121 -0.176091 -0.477121 -0.176091

Committee contingency table (majority (w)):
                actual label value
                     0       1
predicted   0        1       0
  label
  value     1        0       1

committee accuracy = 1
committee precision = 1
committee recall = 1
committee F(1) = 1

a values:
-0.30103=mean
-0.30103=ugly
       3=dog
-2.39794=big
-0.30103=yellow
       3=cat
      -3=lion

b=-4.34139

high a, in |a| order: dog cat -lion -big
low a, in |a| order: -mean -yellow -ugly

Overfitting test -- testing with only the training
examples that were used:
Committee contingency table (majority (w)):
                actual label value
                     0       1
predicted   0        2       0
  label
  value     1        0       3

committee accuracy = 1
committee precision = 1
committee recall = 1
committee F(1) = 1
```

Figure 10. Solution for Pets Problem

Figure 10 shows excerpts from the output of our system for the "pets problem". We see that the learner used 5 training examples and that 2 were negative (meaning the label was NO) and 3 were positive (label is YES). Then we see the results of the learning – the log probabilities that were computed. Next is a contingency table showing the performance of the trained system on the 2 test documents ("Mean Ugly Cat" and "Big Yellow Lion"). The table indicates that the system got both of these test cases correct. The values of some performance measures based on the contingency table follow. There are then listed values of a (weights) and b (threshold). These were computed according to equation (5), with the a values being listed in the same order as the tokens are represented in the feature vector. Following the b value is a list of the tokens in order of decreasing $|a|$. We use absolute value because we are interested in the magnitude of the effect that each token has on the prediction. If the token is preceded by a "-", then the effect is negative. A human solving this problem would most likely quickly decide that a document is "about pets" if it contains the word "cat" or "dog". We can see from the tokens sorted by $|a|$ that this is also what the naive Bayes learner computed. However, the learner also noticed that additional good indicators of a document being about pets are (in order): no mention of "lions" and not using the word "big" (all negative training examples contained "big"). The words "mean", "yellow", and "ugly" were not found to be very useful in classifying documents (each word occurs once in a YES document and once in a NO document). At the end of the system output, we see another contingency table, labeled "overfitting test, testing with only the training examples that were used". As mentioned before, testing with training data is analogous to only testing students with problems that have already been solved for them by the teacher. Such an approach may in fact test their knowledge, but it may also be simply testing their ability to memorize. We, however, often perform such an "overfitting test" after the completion of the real testing that uses the test set. This was originally added to the code as a way to attempt to monitor overfitting, which in a sense is what happens when a machine learning algorithm stops generalizing information in the examples and starts learning aspects of specific training examples (which is not of much use when predicting on unseen examples) ... in a sense, memorizing. The idea behind this overfitting test is that if the testing done using the training set produces markedly better results that the testing done using the test set, then it is possibly an indication that the learner is overfitting. Note that in this case, the overfitting test does indicate that after learning, the naive Bayes algorithm was also 100% accurate on the training set.

4.3.1.4 Learning Algorithms Used in ALC

The winnow, perceptron, and naive Bayes algorithms, as discussed earlier, are all used in supervised learning systems, thus providing "baseline" systems against which we will compare ALC methods. Recall that the 3 parts of ALC are deciding whether or not to see the label, learning, and forming the committee prediction. We will also use each of these 3 algorithms as learners in ALC systems.

4.3.2 Deciding Whether or Not to See the Label

In our experiments, we used several methods to decide whether or not to see the label. In this section we discuss the main methods that we developed. These methods were used in several experiments and so we explain them in some detail here. Other methods will be covered in the discussion of the experiment in which each was used.

Recall that the 3 parts of ALC are deciding whether or not to see the label, learning, and forming the committee prediction. The methods discussed in the rest of this section are all methods for deciding whether or not to see the label.

4.3.2.1 QBC-REP

We term this method "QBC-REP" to distinguish it from QBC methods developed by others. The original form of QBC is not, as will be explained, directly usable in certain real-world domains (including text categorization), so various researchers have developed different adaptations of QBC to their particular domain of interest. We call our QBC-based method QBC-REP.

The original QBC (Query By Committee) approach maintains a committee that initially contains all possible candidate hypotheses, i.e., a representation of the version space [Freund et al. 1992, Seung et al. 1992, Freund et al. 1997]. k is an integer-valued parameter for the QBC method. When presented with an unlabeled example, $2 \times k$ members are chosen from the committee at random and each member is queried and gives its prediction for that example. If their predictions form a tie, then the example is assumed to be maximally informative, and the label is requested. Once the label is received, *all* hypotheses inconsistent with the label are removed from the committee – not just the k errant voters. Learning occurs by removing hypotheses from the committee (hypotheses are removed from the version space). As incorrect hypotheses are removed, the committee decreases in size.

Once learning has completed, the committee consists of all possible hypotheses that are consistent with all of the training examples that have been used. Prediction on previously unseen examples is done by picking any one member of the committee at random and using its prediction. All members of the committee are at this point presumed to be equally good at predicting, since each member is consistent with all of the training examples that were used.

Most researches use a value of $k = 1$, even though the committee may be quite large (especially initially). That is, whether or not to ask to see the label is determined by the predictions of only 2 committee members. One may wonder at the use of such a small value for k. Of course, these member predictions take time, and especially if the prediction process is fairly complex, one will want to use a small value of k. But one might also think that having k large would make the voting more representative of the distribution of predictions of the entire committee, on the average. However, it has been found that making k larger than 1 does not actually result in very much improvement [Freund et al. 1997].

The motivation for using QBC is that it offers a significant reduction in the number of examples used as compared to the number used by supervised learning or by randomly choosing examples. When comparing to the supervised learning case (which is the comparison case of interest to us), the number of examples used by QBC is proportional to the logarithm of the total number of training examples [Freund et al. 1992]. We first offer an intuitive explanation of why QBC behaves as it does "on the average". Intuitive explanations are by their nature not mathematically rigorous but instead are meant to give one a feel for why the observed behavior is occurring. We will then examine a situation where the performance of QBC will perhaps not be so great.

The following discussion is based on the behavior of the committee and its members (the hypotheses) *on the average*. Specific cases may in fact not behave in exactly this manner, depending on the nature and distribution of incorrect hypotheses and on the order in which hypotheses are actually randomly selected for querying. This behavior does occur in the manner described, on the average, if one considers a large number of cases and has, on the average, incorrect hypotheses that exhibit a range of incorrect behaviors.

Consider a committee consisting of all possible hypotheses. We will first examine the behavior for $k = 1$. Correct hypotheses will always predict correctly. Incorrect hypotheses will (unfortunately) vary greatly in their degree of incorrectness. Generally, an incorrect hypothesis will in fact predict correctly for some or even a great many training examples, and only be incorrect for certain types of training examples, perhaps for very few. One can think of incorrect hypotheses as having expertise in some areas (predicts correctly) and weaknesses in other areas (predicts incorrectly). A correct hypothesis therefore has no areas of weakness.

A training example is presented to 2 randomly chosen members of the committee as a query – each member is asked to predict the label for that example. If the 2 members' predictions agree, it means that both members are correct or both are incorrect on that example. If the 2 members disagree in their predictions, it means that, on that example, one of the members is correct and one is incorrect. The example is used by QBC only when the predictions of the 2 randomly chosen committee members disagree. Therefore, when disagreement occurs and the label is requested, on the average half of the hypotheses in the committee are incorrect *on that example* and thus are incorrect hypotheses. All of these incorrect hypotheses are then removed from the committee. Therefore the committee size will decrease by a factor of 2, on the average, each time a training example is used. The assumption is that the choice of the 2 members that were queried was by a random process and so on the average their predictions are representative proportionally of the predictions of all members of the committee. The fact that half of the members queried predicted incorrectly on that example therefore means that half of the members in the committee as a whole are incorrect on that example (on the average). After each such reduction in committee size, two things are true. First, the percentage of remaining members that are correct has increased (and the percentage of remaining members that are incorrect has decreased). Secondly, the incorrect hypotheses that remain, since they have survived thus far, are on the average incorrect less often than the incorrect hypotheses that have been removed. The remaining incorrect hypotheses have more areas of expertise/fewer areas of weakness than the incorrect hypotheses that have already been removed, at least as such areas are measured by the training examples. This is true since each member is equally likely to be chosen to be queried, and the incorrect hypotheses that remain on the committee have survived all of the preceding rounds of querying and incorrect hypothesis removal, so they must have been incorrect less often than the incorrect hypotheses that have not survived. As these query-removal cycles progress, the committee gets smaller, and it will be increasingly difficult for subsequent training examples presented to cause a disagreement amongst 2 randomly chosen members and so be used.

To see this, consider that for each query of 2 members (member A and member B), there are four possible outcomes. Let H_M stand for the hypothesis represented by member M. The four possible outcomes are:

1. H_A and H_B are both correct hypotheses

   In this case, their predictions will always agree (both will be YES or both will be NO).

   The 2 hypotheses will never "disagree".

2. H_A is correct and H_B is incorrect

   In this case, their predictions may agree or disagree. H_A will always give a correct prediction. H_B may give the correct prediction or may give an incorrect prediction, depending on whether the training example presented is in an area of H_B expertise or weakness.

The 2 hypotheses will "disagree" only if the training example is in an area of weakness of H_B.

3. H_A is incorrect and H_B is correct

   (same as above for case #2, but switch roles of H_A and H_B).

4. H_A and H_B are both incorrect hypotheses

   In this case, their predictions may agree or disagree. If the training example is in the area of expertise of both H_A and H_B, then their predictions will agree (and both will be correct). If the example is in an area of weakness of H_A and in an area of weakness of H_B, then their predictions will also agree (but both will be incorrect). Otherwise, one hypothesis will give a correct prediction and one will give an incorrect prediction.

   The 2 hypotheses will "disagree" only if the training example is in an area of weakness of exactly one of H_A or H_B.

As the committee size decreases, incorrect hypotheses are being removed from the committee, but the number of correct hypotheses remains constant. Therefore, as learning occurs, the proportion of hypotheses in the committee that are correct increases. All of the hypotheses on the committee that are correct will always predict correctly, when chosen to predict. To get the tie vote needed in order to ask for the label, 2 randomly chosen members of the committee have to disagree. From the above list of four cases, one can see that disagreement only occurs when a training example is in an area of weakness of exactly one of the 2 hypotheses being queried. In order for a tie vote to occur, one of the randomly chosen hypotheses will have to be correct on the example being presented; this will become increasingly easy to have occur, since the proportion of committee members that are correct is rising, and the remaining incorrect committee members have only a few weak areas. The other randomly chosen hypothesis will have to be incorrect on the example being presented; this will become increasingly difficult to have occur, since there are fewer incorrect hypotheses and those that remain are quite often correct in their predictions. It therefore becomes increasingly difficult for training examples presented to QBC to obtain the tie vote needed to have the label asked for.

We have shown that as examples are used and the committee size decreases, it becomes increasingly difficult to use an example in the random stream of candidate training examples that are being presented to the committee. But what about the examples that *are* used – are they of benefit? Those examples are of benefit since (on the average) each time one is used, half of the remaining committee members are incorrect on the example and so those incorrect hypotheses are removed from the committee. Thus even though the rate at which examples are used drops logarithmically, each example used is (on the average) beneficial in that it significantly improves the accuracy of the committee by removing additional incorrect hypotheses from the committee.

The above analysis was for $k = 1$. By increasing the value of k, one increases the number of members that are asked to predict. This takes more time, especially for learning methods for which prediction is costly, but it does give one more certainty that any tie vote is representative of the predictions of all of the committee members.

QBC will not work very well in situations in which the committee is initially (or after several query-remove cycles) in a state where the (remaining) incorrect hypotheses are severely skewed towards having very large areas of expertise and having small and differing areas in which incorrect predictions are made. In such a situation, it will be very difficult for QBC to randomly find 2 hypotheses that disagree, and so QBC may end up not requesting any (more) labels. While this is a situation in which QBC will not perform well, it is not clear, once a committee reaches this state, that this is an unacceptable situation. While the committee has not been reduced to one containing only correct hypotheses, it has been reduced to one which contains only correct hypotheses and some very accurate and diverse incorrect hypotheses. Even selecting a hypothesis at random and using that hypothesis for predicting on unseen examples (such as QBC does) will produce highly accurate albeit not perfect results. Some aspects of committee voting (to be discussed later) can also help to overcome this difficulty of having accurate but not perfect hypotheses in the committee. In fact, we will see that in noisy domains, this situation is the rule rather than the exception.

Note that QBC is a "complete" method that handles all aspects of the learning and predicting processes. It specifies how it is to be decided whether or not to see the label in a training example ($2 \times k$ members vote, see label if vote is a tie), how to learn (incorrect hypotheses are removed from the committee), and how to predict (a random remaining member is used).

There are, however, some difficulties in using the original QBC for certain domains (including text categorization):

1. QBC needs to maintain, in some form, a representation of all possible hypotheses consistent with the training data seen so far – some representation of the version space. Since one needs to be able to remove individual hypotheses when found to be incorrect, this usually means that one must in some manner represent each individual hypothesis. However, for many real-world problems, there is a large (effectively infinite) number of candidate hypotheses, and so representing the individual hypotheses will not be practical and perhaps not even possible.

2. QBC is not able to handle noisy data. Consider the case of a training example being used that has class noise – which means that the label given with the example is incorrect. This will cause QBC to remove *all* correct hypotheses from the committee.

Text categorization data is noisy, and the large number of attributes typically results in large numbers of possible hypotheses. So we have developed a method which we call QBC-REP which addresses these problem areas. It is based on QBC, adapted to work using a finite committee to **rep**resent the hypotheses. QBC-REP is a general method developed specifically for domains which have a very large version space and/or which have class or attribute noise. While we did develop this method for use in our text categorization research, it is not specific to that domain and so can be used in other domains.

The QBC-REP method is a not a complete learning method, but rather is a method used for deciding whether or not to see the label. Two members of the committee are chosen at random, and if their predictions disagree, then we ask to see the label. QBC-REP does not specify how to maintain the hypotheses, how to learn, or how to predict. We can therefore combine QBC-REP with other methods that handle hypothesis representation, learning, and forming the committee prediction.

4.3.2.2 AllMargin

AllMargin is also a method for deciding whether or not to ask for the label and can be combined with various learning and prediction methods to form many different ALC systems. In the AllMargin method, we have modified and also generalized our QBC-REP in several ways. First, all members of the committee predict. This is done because the prediction methods we are using are quite fast, so we lose little time by having the entire committee predict. It appears that this does give one a voting margin that is more representative of the entire committee. We define margin to be the number of NO predictions less the number of YES predictions. The margin magnitude is the absolute value of the margin. In this method, we look at the margin magnitude of the prediction votes to determine whether or not to ask for the label. Each committee member is given the unlabeled example and predicts NO or YES depending on its current hypothesis. The magnitude of the difference in the number of votes for each class is the margin magnitude for that training example. This value can be as large as the size of committee (if they, for example, all vote NO). The smallest possible margin magnitude value is 0 if the committee contains an even number of members and 1 if the committee contains an odd number of members. This smallest possible value is called *spumm* (smallest possible unweighted margin magnitude). It is unweighted in the sense that each committee member gets exactly one vote. It is a magnitude since we always take the absolute value of the difference between the votes for NO and the votes for YES. As an example, a committee of 8 learners has $spumm = 0$ (since the smallest possible margin magnitude occurs when there is a tie vote); if the voting is 3 NO and 5 YES, then the margin magnitude for that training example is 2.

When using the AllMargin method, one also sets two integer parameters – *lowEnd* and *highEnd*. The value of *spumm* is then used in conjunction with these parameters to determine whether or not to ask for the label. Margin values that fall within the inclusive range bounded by these parameters with respect to *spumm* cause those examples to be used – their labels are requested and learning occurs. That is, the method asks to see the label if:

$$spumm + lowEnd \leq margin\ magnitude \leq spumm + highEnd$$

This method is based on the notion that the degree of disagreement amongst *all* of the committee members is a good measure of how informative the example is, where an example is informative in the sense that it contains information different from the current hypotheses of several of the learners. For example, if *lowEnd* = 0 and *highEnd* = 2, and the committee contains 11 members, then *spumm* = 1 and only examples with a margin magnitude of 1 or 3 will be used. The assumption is that the knowledge that is in examples on which the committee members already pretty much agree is assumed to already be contained in several of the learners' knowledge bases – that is, several learners "already know" the knowledge that is in those examples.

This may sound like a very safe method to use. There are, however, some problems that can arise. First, if the committee members all agree on the predicted label (and so do not ask for the label and do not learn from the example), it may be that in fact the committee already possesses the knowledge in that example. But unfortunately it may also be that the committee members are all/mostly stupid and their predictions are incorrect. Using this method, they will never find that out.

Another potential problem has to do with noise. Document collections often have large amounts of noise. It may be that these noisy examples are the ones which cause maximal disagreement in the committee member's predictions and so are the ones that AllMargin picks as "informative". But by their nature, noisy examples contain errors, and the learners will not fare well if they learn from erroneous examples instead of from correct ones. To allow experiments to investigate this is in fact why the *lowEnd* parameter was included – so that one can not use examples that have a small margin magnitude if those in fact are quite noisy.

In spite of these potential problems, this method works very well at selecting informative examples.

4.4 Computational Complexity

Computational complexity is a measure of the resources needed to solve a problem, expressed as a function of the size of the input. There are several different measures of computational complexity, basically differing in whether or not the function bounds the actual resource demand from above, from below, or both, and whether or not the bound/s are asymptotically tight. In addition, one usually computes such behaviors based on either worst-case or average-case conditions.

The resources we are interested in examining are elapsed processor time ("time") and memory usage ("space"). We will see that, due to the way we store the documents, space complexity will be very manageable, and so our main concern will be time complexity. We want an upper bound on time complexity that is asymptotically tight. Normally we strive for worst-case behavior in our analyses. In some situations it is possible to compute a clearly representative average-case value of a component in an expression, and so we will use that average, usually assigning it to a specific variable name. We thus as a practical matter end up with an analysis that is almost completely worst-case. We will refer to it as "worst-case", with the understanding that when we are able to compute a clearly defined and clearly representative average-case value of a component in an expression, we will use that.

Resource demand is usually represented by $O(b(n))$, where n is a measure of the size of the input. We start with an intuitive definition of $O(\,)$ and also introduce the notation that we will be using for $O(\,)$.

We want to compute a function $b(n)$ which forms an upper bound on the actual behavior. In other words, we want to determine a function $b(n)$ that is greater than or equal to the actual behavior of the system. The "or equal to" gives us a tight upper bound, which is of more practical use in predicting system behavior. Let $a(n)$ be the function for which we want to compute $O(\,)$. $a(n)$

describes the **a**ctual behavior of the system in terms of its worst-case use of the resource of interest (for example, time or space). We want to compute $O(\,)$ for $a(n)$. Usually for $a(n)$ we have only a series of data values measured from various runs of the system of interest – a set of $(n,\ a(n))$ values – rather than having a closed form expression for $a(n)$. We say that $O(\,)$ for $a(n)$ is $b(n)$ if some constant multiple times $b(n)$ is greater than or equal to $a(n)$ for all sufficiently large n. The "some constant multiple" is due to the fact that we are interested only in the behavior as a function of input size and not interested in proportionality constants. That is, $O(2.5n^3)$ and $O(n^3)$ gives us the same information, that the behavior behaves as the cube of the input size. There is nothing technically wrong with $O(2.5n^3)$, but $O(n^3)$ is clearer, conveys the same information, and so is the preferred form. The "for all n sufficiently large" is to handle the often occurring situation where, for smaller values of n, system behavior may be of a different form. We are interested in predicting behavior of the system only for arbitrarily large values of n. More formally, $O(\,)$ is defined as follows (from [Cormen et al. 1996]): $O(\,)$ for $a(n)$ is $b(n)$ if there exist positive constants c and n_0 such that $0 \le a(n) \le cb(n)$ for all $n \ge n_0$.

The input size n for $O(\,)$ has to be for a "reasonably efficient" representation of the input. Otherwise one could, through a poor representation of the input, make an algorithm seem more efficient than it truly is by simply measuring its performance with respect to a distorted yardstick.

There is a potential notational ambiguity in $O(\,)$ notation. When one sees "$O(n^4)$, is the n^4 the bounding function $b(n)$ or the actual behavior $a(n)$? It is the bounding function $b(n)$, and in fact as mentioned above one rarely has an expression in terms of n for the actual behavior. Therefore, "$O(n^4)$" means that we are discussing a system whose time (for example) complexity is bounded above by some constant times n^4 for n sufficiently large.

Recall that one of our goals is to develop methods that are both space and time efficient. We will usually term a process "efficient" if its complexity is $O(n)$.

When we derive $O()$ expressions for the various modules of an ALC system, it will be helpful to simplify them. We can simplify $O()$ expressions by doing 3 things:

1. remove lower order terms

   Our main interest is in computational complexity as n gets very large. For very large n, the highest order term dominates.

   Example: $O(n^2 + 3n) \rightarrow O(n^2)$

2. remove constants of proportionality

   We can do this because the definition of $O()$ already includes a constant of proportionality, so we need not explicitly provide one, and it is clearer if we do not.

   Example: $O(3n^2 + 5n) \rightarrow O(n^2)$

3. decide which parameters are actually constants, or at least not themselves functions of the input size

   We would like to simplify our $O()$ as much as possible. Parameters that do not vary with input size are constants and so should be treated as such.

We are interested in the computational complexity of the system as a whole, and also of certain major components of the system. First we will develop expressions for input size, and then look at $O()$ for several systems/components. We will as mentioned above use worst-case analysis, and we will generally simplify the expressions using the rules listed above. In simplifying $O()$ expressions, whether and to what degree one applies the above rules, and if so, when and in what order, can sometimes be an issue. We have purposely erred on the side of keeping too many terms rather than too few. We do this so that the expressions derived are fairly general and not overly specific to our domain. When we apply the expression in a specific experiment, then we will incorporate the knowledge we have about our domain.

The system as a whole takes as input a corpus of documents and a text categorization request. Each document in the corpus is represented as a feature vector whose length is the number of tokens in the dictionary. If we have d documents and t tokens, then the corpus is of size $Kd(t+1)$, where K is a constant of proportionality. In other words, the input size is proportional to $d(t+1)$ (versus being exactly equal to it) because we have several bookkeeping arrays in addition to the space used for storing the data itself. The "+ 1" is for the label portion of each feature vector. A text categorization request is simply the name of the category that we are interested in, and so can be represented by a single integer or string. Thus we have that the input size is $O(d(t+1)+1)$. Dropping low order terms, we have that the input size is $O(dt)$.

However, recall that there is a requirement that the input be represented in a reasonably compact manner. We will see later that we are dealing with very sparse data (statistics are provided later for the Reuters-22173 and Reuters-21578 corpora). Very few positions in each feature vector contain "1" values, with all of the rest of the positions containing 0's. On the average, less than 0.2% of the feature vector positions contain 1's. We therefore store the feature vectors using a sparse data representation that basically only stores the feature vector indices for elements whose value is 1. Let t_a = the average number of tokens in each document. The size of the input is then:

$$O(dt_a) \tag{7}$$

In other words, from now on when we speak of the input size, we are referring to $O(dt_a)$.

A brief discussion is appropriate on the relationship between t and t_a since we will often (such as in the next paragraph) want to compare $O()$ expressions containing t with the input size, which is $O(dt_a)$. In general, we would *like* to determine the function $f()$ where $t = f(t_a)$. As one increases the average number of unique tokens used in a document (t_a), one would expect the number of tokens in the dictionary (t) to also increase. For example, if one had a corpus of documents each containing 5,000 different tokens on the average and compared it to a corpus in which each document contained an average of 10,000 different tokens, one would expect that the latter corpus would usually require a larger dictionary. In other words, as the number of different tokens used in documents increases, it is likely that some of them will not yet have already been used in other documents in the corpus and so will need to be added to the dictionary. Thus we can say that $t_a\uparrow \Rightarrow t\uparrow$. This tells us that $f()$ is an increasing function of t_a. But we would also like to know in general the exact form of $f()$ – or at least whether the variation in t caused by changes in t_a is sublinear, linear, or superlinear. In an effort to find out more about $f()$, we do the following. Let l_a = average document length expressed in number of actual tokens. l_a is based on the total number of tokens used in each document, whereas t_a is based on the number of different tokens used in each document. In other words, l_a tallies each occurrence of each token whereas t_a is the number of different/unique tokens. If we compute t_a, t, and l_a for the corpora that we used in our research (Reuters-22173 and Reuters-21578), and also for another large document collection (the 20 Newsgroups corpus), we obtain the values given in Table 2. The entries are in order of increasing t_a.

We are interested in determining more about $f()$, where $t = f(t_a)$. There is a large body of empirical results by several researchers suggesting that there is a relationship between l_a and t [Zörnig and Altmann 1995]. Using Zipf's Law and work done by Halstead, Prather [Prather 1988] derived estimates for the bounds on l_a and found that the value of l_a is lower bounded by $K_1 t(\gamma + \ln t)$ and is upper bounded by $K_2 t \log_2 t$, where $\gamma \approx 0.577$ (Euler's constant) and the K_i are

TABLE 2. Corpus Statistics

corpus	t_a	t	l_a
20 Newsgroups titles only	5.83247	9,634	5.94274
Reuters-21578 titles only	7.74377	15,842	7.86514
Reuters-22173 titles only	7.78424	16,600	8.19551
Reuters-21578 full text	81.8377	48,428	139.617
Reuters-22173 full text	81.8523	48,446	139.985
20 Newsgroups full text	202.48	203,921	388.842

constants. In our domain, $\ln t \gg \gamma$, so we can say that l_a varies as $t \log t$. This means that l_a is an increasing superlinear function of t, and therefore t is an increasing sublinear function of l_a. We are, however, interested in the relationship between t_a and t. We know that $t_a \leq l_a$. But determining a general form for $f()$ from so few data points as are in Table 2 would be difficult.

Back to the basics. Do we really need to know $f()$? Note from Table 2 that $t \gg t_a$ and also that $l_a \approx t_a$. These are characteristics of the data used in our experiments. This is sufficient to at least allow us to compare various $O()$ expressions that contain t, t_a, and l_a as regards our experiments.

Since we do not know the function $f()$ and we want our derived $O()$ expressions to be general, when we develop $O()$ expressions that contain t, we will retain t in the expressions. When we are examining a $O()$ expression for a system being used in our experiments, then we will use the above characteristics ($t \gg t_a$ and $l_a \approx t_a$) to simplify the expression.

It is important to point out that the sparse data representation that we are using is the main reason we are not very worried about space complexity – the entire corpus easily fits into memory. If one has c committee members, d documents in the corpus, t tokens in the dictionary, and if each document contains on the average t_a unique tokens, then (ref. Equation (7)) the space needed to store the corpus is $O(dt_a)$ (for the indices of the 1-valued attributes). The space needed by the learners is $O(ct)$ (for the weights). Thus space complexity of the entire system is low.

We now turn to the time complexity of certain specific algorithms that we are using. Recall that the 3 parts of ALC are deciding whether or not to see the label, learning, and forming the committee prediction. We will next analyze $O(\,)$ behavior for some of the more common methods used for each of these 3 parts. The analysis of other ALC methods will be done when we discuss the experiments in which they are used.

We will examine 3 methods for deciding whether or not to see the label: always (i.e., supervised learning), QBC-REP, and AllMargin. For each method, we will develop expressions for $O(\,)$ for a complete trial. The input to the module that decides whether or not to see the label is the feature vector for an example (containing t_a values), so we include in this computation the effort required to read the input data. We do not assume that all of the input data can be stored in high speed memory at one time.

For supervised learning, the module simply responds "yes", regardless of the size or contents of the feature vector. Thus the time complexity for simply indicating that the committee wants to see the label is $O(1)$. That is, the input data need not even be examined. It does, however, still need to be read (for use by the learning algorithm). Let e be the number of epochs. Since there are d documents and since the training set size is generally proportional to d (but recall that we remove constants of proportionality from $O(\,)$ notation), this results in $O(ed)$ behavior for the supervised method of deciding whether or not to see the label.

However, if the learning algorithm is mistake-driven and if the decision is made to see the label, then we have the module also compute the predicted labels for all of the committee members. We use this approach since, in general, some or most of the predictions will have been computed anyway (see for example QBC-REP and AllMargin), and so it is more efficient for all of the remaining members to form predictions and have all of those predictions passed on to the mistake-driven learner. Let c be the number of members in the committee. If we (for now) let $O_p()$ be the time complexity for one learner predicting on one training example, then this results in $O(edcO_p())$ behavior if the learning algorithm is mistake-driven.

For QBC-REP, one picks 2 committee members at random and has each of them predict the label. If they disagree, then the label is requested, otherwise not. Then $O(ed2O_p())$ is the time spent deciding whether or not to see the label. If the learner is mistake-driven and QBC-REP returns "yes", then it needs to predict for all committee members, and so the worst-case behavior will be $O(edcO_p())$. This is worst-case since it applies only if the decision as to whether or not to see the label is "yes".

For AllMargin, one has all committee members predict and then, based on the voting margin, decides whether or not to ask for the label. The time spent in deciding whether or not to see the label is therefore $O(edcO_p())$. The AllMargin method is smart enough to avoid unnecessary predicting, so for any one training example, there may be fewer than c predictions required. For instance, if we have a committee of size 8 and we are taking only examples that have a margin of 0, (and the learner is not mistake-driven), then once 5 members have predicted YES, no further predicting is required. However, we will use the worst-case behavior.

Next we look at the second part of ALC – learning. We will analyze the time complexity for the winnow, perceptron, and naive Bayes algorithms, also for a complete trial. For winnow, we consider the case where the winnow algorithm is using an example. That is, it has been decided to see the label for the example, and now the example is passed to the committee of winnow learners. (Ref. Figure 6 on page 78). First, each member needs to predict on this example. But that computation has already been performed by the module that decides whether or not to see the label and thus the effort required was included in the $O(\,)$ computations for the various methods for deciding whether or not to see the label. For worst-case, all of the members will have to learn – that is, they were all incorrect in their predictions. Only weights corresponding to attributes whose value is 1 are actually modified, so the time complexity for one member learning from one training example is $O(t_a)$. For e epochs, d documents, c committee members, it is therefore $O(edct_a)$. This is worst-case since we are assuming not only that all of the members learn each time, but that each training example is learned from during each and every epoch.

The analysis for perceptron (ref. Figure 7 on page 81) is very similar. Recall that the only differences are, first, the operator used in the updating of weights (+ instead of ×), and, second, the fact that θ is also updated in the perceptron. Therefore worst-case behavior is $O(edc(t_a + 1)) = O(edct_a)$.

The analysis for standard naive Bayes proceeds as follows (ref. Figure 8 on page 83). In step #1, for each training example, we increment counts corresponding to $x_i = 1$ and we increment (once) the corresponding t_k. Therefore we have $O(t_a + 1)$ for one example, and $O(d(t_a + 1))$ for all training examples. Recall that standard naive Bayes is supervised and uses only one learner (c = 1) and one epoch (e = 1). After all examples have been processed by step #1, steps #2 and #3 are each performed once. Step #2 computes 4 conditional probabilities for each attribute – $p(x_i = j|k)$ for j = 0, 1 and k = YES, NO. Thus we have $O(4t)$. Recall that t is the number of tokens in the dictionary and also the length of the attribute portion of the feature vector. Step #3 computes 2 values, p(YES) and p(NO), and so is $O(2)$. Thus standard naive Bayes time complexity for learning is $O(d(t_a + 1) + 4t + 2) = O(dt_a + 4t)$.

We next look at computational complexity for the 3rd part of ALC – forming the committee prediction. Actually, it turns out to be easier if what we first determine is the computational complexity for the predicting done by an individual learner on one example. This is because predictions of individual learners are used in several ways – they are used by some methods in order to determine whether or not to see the label, they are used by mistake-driven learners to decide whether or not to learn from the example, and they are used to form the committee prediction. By developing an expression for the behavior of a single learner on a single example, we can then use that expression in the analyses of all of the above situations. For the above analyses of computational complexity for deciding whether or not to see the label and for learning, we have determined $O(\,)$ for that module for an entire trial – that is, for e epochs, c committee members, d documents, etc. The analyses of prediction computational complexity for one learner on one example will give us expressions for what we have previously referred to as $O_p(\,)$.

Winnow and perceptron both predict in the same manner. Given an example $\vec{x}$, the cross-product $\vec{w} \cdot \vec{x}$ is computed and compared to θ (predicting YES if >, NO otherwise). Thus prediction on one example is $O(t_a)$. This is $O_p(\,)$ for winnow and also for perceptron.

For standard naive Bayes, using (as we do in our system) Equation (6) (on page 96), we obtain the following expression for $O(\,)$ for prediction. For each side of the $\gtrless$ operator, we need to compute an expression which is $O(1 + t)$. One might reasonably argue that we are also taking the log( ) and so should include that, which will give us $O(2 + 2t)$. Doing this for both sides of Equation (6) gives us $O(2(2 + 2t)) = O(t)$. This then is $O_p(\,)$ for the standard naive Bayes learning algorithm.

We have thus far derived $O(\,)$ for several of the main modules of an ALC system. We can simplify these $O(\,)$ expressions further by deciding which input size parameters are actually constants, or at least not themselves functions of the input size. In particular, we do not vary the number of committee members for inputs of different sizes. We usually use 7. Sometimes we use 1 if we want a single standard supervised learner. Thus c is a constant, so for example $O(edct_a) \rightarrow O(edt_a)$. Also the number of epochs varies only slightly if at all (and we want worst case anyway), and therefore e can be treated as a constant, and $O(edt_a) \rightarrow O(dt_a)$. Our $O(\,)$ computations, after removing the remaining constants, are summarized in Table 3. Since input size is $O(dt_a)$, all of the $O(\,)$ expressions in Table 3 are linear in the input size except for standard naive Bayes learning and prediction, which vary as t. Recall that in our experiments $t \gg t_a$.

Using the results summarized in Table 3, we will next work through several $O(\,)$ examples (whose results we will refer back to later):

1. An ALC system using AllMargin and winnow. Its time complexity will therefore be the sum of that for AllMargin, winnow, and the final test predictions, and so will be $O(dO_p(\,) + dt_a + dO_p(\,))$. Substituting in for $O_p(\,) = t_a$, we get the following, which is linear in the input size:

$$O(dt_a) \tag{8}$$

2. An ALC system that is supervised and using winnow. Since winnow is mistake-driven, we get $O(dO_p(\,) + dt_a + dO_p(\,))$. Since $O_p(\,) = t_a$, this becomes:

$$O(dt_a) \tag{9}$$

3. A system using the standard (supervised) naive Bayes learning algorithm. Recall that naive Bayes is not a mistake-driven algorithm. Therefore, we have $O(d + dt_a + 4t + dt)$, which (remove lower order terms) becomes:

$$O(d(t_a + t)) \tag{10}$$

TABLE 3. Computational Complexity Summary

d = number of documents in the corpus
t = number of tokens in the dictionary
= length of the attribute portion of the feature vector
t_a = average number of unique tokens in each document

The size of the input to the system as a whole is: $O(dt_a)$

$O(\,)$ for Deciding Whether or Not to See the Label [for a complete trial]:

method	mistake-driven	not mistake-driven
supervised	$dO_p(\,)$	d
QBC-REP	$dO_p(\,)$	$dO_p(\,)$
AllMargin	$dO_p(\,)$	$dO_p(\,)$

$O(\,)$ for Learning [for a complete trial]:

method	$O(\,)$
winnow	dt_a
perceptron	dt_a
standard naive Bayes	$dt_a + 4t$

$O(\,)$ for Prediction (= $O_p(\,)$) [one learner predicting on one example]:

method	$O(\,)$
winnow	t_a
perceptron	t_a
standard naive Bayes	t

At times it is difficult to perfectly reconcile the reported elapsed processor time with the $O(\,)$ expressions derived above. Our purpose in examining $O(\,)$ behavior was to obtain relatively conservative upper bounds on time complexity. To make the math reasonable, broad but conservative assumptions were made – assumptions that do not always hold for specific actual runs or even on the average.

The main assumptions made (in terms of impacting comparison of $O()$ to actual run times) are:

1. Our system does many other things besides deciding whether or not to see the label, learning, and forming the committee predictions.
2. In the $O()$ derivations, we always chose worst-case when no obvious and defensible average was available. This it turns out will be the assumption having the most impact.
3. In the $O()$ derivations, we tacitly assumed that each of the basic operations analyzed takes the same amount of time each time it is executed ... for example, the time spent in the multiplication of 2 numbers does not vary with the values of those 2 numbers, the time spent computing $log(x)$ does not vary with the value of x.

These assumptions do not of course hold in reality for specific runs, and may not even hold on the average. This does not mean that the above analyses are without value – we have obtained conservative expressions for time complexity that are linear in t_a (or, for naive Bayes, linear in t), which indicates that the systems will be reasonably efficient users of time. While our system does have a moderate to large overhead, that overhead is linear in input size. We do "turn the clock off and back on" for major consumers of time that ought not be included in the reported total run time. For many computations, however, it is not practical to turn the clock off and on, as the code is very modular and the timing subroutine invocation overhead can become significant in terms of its impact on reported time. Basically, our overall approach was to time the code that we thought one would probably want in an operational system.

To give an idea of what the reported running time does and does not include, following are lists of tasks that are or are not included in the time that is reported for each run. Basically, we do not include time that is for a process whose results are subsequently used by more than one trial, or for a process that is gathering data for us to analyze and is not a part of the actual text categorization process.

1. included
    a. deciding whether or not to see the label, learning, and any predicting associated with either
    b. input error checking, various internal consistency checks
    c. bookkeeping (for example, keeping track of which examples are utilized, for what purpose, and when)
    d. computation of statistics
    e. dynamic memory allocation, initialization, management, and 2d$\rightarrow$1d mapping
    f. initialization of data structures (such as the learning algorithm weights)
    g. calculations and output
    h. the final test for each system (included because we decided that, in terms of time, this was analogous to the reporting of the text categorization results to the user)

2. not included
    a. loading program in to memory
    b. loading data into memory
    c. getting parameter values from the user
    d. splitting corpus into training, test, and unavailable sets

e. testing performed for the sole purpose of generating learning trace points

f. computations and disk I/O for several output files created from each run – a typical run generates some 15 files (some optional) that contain information such as is used for ANOVA, plot labels, ranking, training document distribution plots, weighted tokens listings, learning trace plots, and learned weight distribution plots

4.5 Sample Complexity

Recall that the performance measures of main concern to us are time and space computational complexity and the number of training examples used during learning. The above discussion of computational complexity analyzes various ALC systems for space and time complexity and develops upper bounds ($O(\,)$ expressions) for their need for time and space resources. One might wonder if a similar analysis is possible for the number of training examples needed. Sample complexity is a measure of how the number of training examples needed grows with the size of the problem being solved [Mitchell 1997]. We will not go into this in great detail, but a brief intuitive discussion will be helpful in pointing out where we stand in the area of sample complexity and also why computing it for our situation would be very difficult and perhaps not of much practical benefit.

One would like to have an expression analogous to $O(\,)$ that would give us a lower bound on the number of training examples that we need in order to learn a target concept. Such expressions have been derived for certain cases, but because few assumptions are made about the distribution from which the training examples are drawn, these bounds tend to be quite large when compared to empirical results. We do not know the form of the distribution of documents in the document space, so certainly assuming little about the form of the distribution is safe, but again such an analysis will yield very loose bounds that in and of themselves are not very helpful since we do not have any where near the number of examples specified in these bounds. Perhaps of more interest in our situation is not so much the exact value of the computed lower bound, but how it might vary with changes in certain parameters. We will use standard sample complexity notation. We will need to briefly get into an area of computational learning theory called the probably approximately correct (PAC) model of machine learning [Valiant 1984]. This area is a complex field in its own right, so we will provide a very general overview, with two specific goals: [1]to show what the relationship is between sample complexity and the number of attributes in our domain, and [2]to discuss aspects of our domain that make this analysis very approximate. We will omit other details of PAC learning theory.

We will use standard PAC model notation, which unfortunately means some reuse of symbols we have already introduced but with different meanings. In particular, δ is used in PAC learning theory as a confidence parameter. This bears no relationship to our use of δ in other parts of this thesis.

Let ε be an error rate and let δ be the probability of the learner not having an error rate $\leq \varepsilon$. Also let n be the number of attributes and let m be the number of training examples needed. The idea behind the PAC model is that a learning algorithm is given the task of learning a close approximation to an unknown boolean-valued target function B. The learner gains information by processing examples from B. The distribution of the examples provided to the learner is unknown. We want to determine a lower bound on m such that, regardless of the distribution of training examples used for learning, we will with probability $\geq$ (1 - δ), obtain a learner whose error rate is $\leq \varepsilon$. That is, one wants a learner that is probably (1 - δ) approximately correct (ε). One usually wants a learner for which the probability (1 - δ) is high and for which the error rate (ε) is low. This corresponds to having small values for both δ and ε. In typical analyses, one often places reasonable upper limits on δ and ε in order to obtain meaningful results.

We now look at our specific situation. The learning algorithm is a linear threshold learner, and we have very large values of n. It can be shown that in this case m is lower bounded by [Blumer et al. 1989, Freund et al. 1997]:

$$\max\left\{\frac{4}{\varepsilon}\log_2\left(\frac{2}{\delta}\right), \frac{8n}{\varepsilon}\log_2\left(\frac{13}{\varepsilon}\right)\right\}$$

In our domain and for realistic choices of δ, the right hand term in the max dominates. Therefore, if we keep ε and δ constant and double the number of attributes, we should expect to have to double the number of training examples used. If we do not or can not use twice as many training examples, then we should expect the performance of the learner (accuracy, $F_{1.0}$, etc.) to decrease.

The assumption that we know absolutely nothing about the distribution of the B examples provided to the learner results in this bound being quite large and usually much larger than empirical results that are obtained from actual systems. The pessimism inherent in not knowing anything about the distribution of examples forces the PAC model to compute sample complexities that will hold for any possible distribution. For example, if we specify that we want to be 95% confident that the learner we obtain is at least 98% accurate, then $\delta = 0.05$ and $\varepsilon = 0.02$, and so the above equation indicates that the number of training examples needed is approximately $4000n$. A typical value of n is 16,600 (Reuters-22173 corpus, titles only), so the PAC model would specify that a minimum of approximately 66 million examples are needed in order to be 95% certain of obtaining a learner that is 98% accurate. The entire Reuters-22173 corpus contains only 22,173 documents, and ALC systems often need only 5 - 7% of those.

The other aspect of computing sample complexity for our domain that poses difficulties is that our domain contains attribute and class noise. The standard PAC model assumes that the examples provided to the learner are noise-free, in that they are faithfully drawn from the the function B. When one has attribute or class noise, the examples provided to the learner are no longer true examples of B. Work in this area is being done by several researchers, including [Aslam and Decatur 1996, Kearns and Li 1988]. As one might expect, the bounds on sample complexity become even higher when noise is present, especially if one makes no assumption about the nature of the noise.

In conclusion, yes there is a concept called sample complexity which provides theoretical lower bounds on the number of training examples needed, but the bounds are extremely loose and we empirically obtain good results using far fewer examples than predicted by this theory. A value of looking at sample complexity, however, is that one would expect, as the number of attributes increase, to need to use proportionally more training examples in order to maintain the same level of performance.

5. Experimental Methodology

In this chapter we discuss the details of how we performed the various experiments and provide information on the corpora that were used in those experiments.

5.1 Approach

Our experiments were conducted using mainly the Reuters-22173 corpus, but we also used another text corpus – Reuters-21578. See the section entitled "Corpora Used" for descriptions of the corpora used. However, the two text corpora used have several characteristics in common:

1. The documents in the corpus are real documents.

   The documents were written by humans for some purpose other than to create a corpus. The documents are "real-world" in the sense that they were not machine-generated, nor were they written for the purpose of testing text categorization systems.

2. Each corpus contains a large number of documents (about 22,000).

3. A document can be included in any number of categories (including none).

4. Each document has been assigned to zero or more categories by a professional indexer.

5. Each document has a title and a body.

   Many text categorization systems (and many of our experiments) use only the document titles. This typically makes the problem easier to solve because usually the dictionary is smaller, so the feature vectors have fewer attributes, and thus the processing is faster. This would not in and of itself be a sufficient reason, but quite a few text categorization systems do perform extremely well just using the titles. The idea is that in the title, the author has provided a very short statement of what the document is about, and "what the document is about" is the basis for text categorization.

6. documents have missing or misspelled words (attribute noise)

7. the category labels are sometimes in error (class noise)

   Recall that the learning algorithms are relying on the "actual labels" provided by the teacher being correct. Using the label provided by the teacher, in conjunction with the information in the feature vector, is how the learning algorithm modifies its knowledge base so as to "learn" from the example. Most large collections of documents are not all indexed by the same person – several indexers work on the documents, and the task is usually performed over a span of time (as the documents are gathered for the collection). It is an unfortunate fact of life that even trained human indexers are not as correct as one might think or at least hope. Many studies have been done, and the consistency within and amongst trained human indexers can range *at the high end* anywhere from 50% to 75%. [Cooper 1969, Saracevic 1991, Hersh 1996, Sievert and Andrews 1991].

The standard equation for consistency between two indexers *A* and *B* is [Hersh 1996]:

$$\frac{both}{both + onlyA + onlyB}$$

where *both* is the number of categories assigned by both *A* and *B*, *onlyA* is the number of categories assigned only by *A*, and *onlyB* is the number of categories assigned only by *B*. Thus, for example, if *A* assigns 5 categories and *B* assigns 4, and 3 are in agreement, then the consistency would be 0.50.

8. the documents in each category are often inconsistent

Several different documents in the corpus may, after preprocessing, have identical feature vectors. Therefore, from the perspective of the system, these documents are "the same". When this occurs, it can also happen that the human text categorizer has assigned different categories to these documents. In our case, for example, there can be several documents that contain the same tokens but that were categorized differently by the indexer. Such groups of documents are termed "inconsistent", in that they have the same feature vector but have different labels. Many learning algorithms are especially troubled by inconsistent groups of documents, since it appears to the learner that the teacher is changing his/her mind as to what the correct label is for particular examples. And perhaps even worse, if one is unlucky, one class of such documents (YES) will be in the training set and the other class (NO) will be in the test set. Having learned YES for certain types of examples, the learner finds that the answer on test day has been changed to NO. Categories that contain inconsistent documents are, for us, a definite concern since it means that it is impossible for any one committee member to learn to classify documents perfectly.

Given a category, one can have a group of inconsistent documents for one or more of several reasons:

a. The human indexer made errors.

   See above discussion of class noise.

b. Approximations made during preprocessing.

   Whether or not all of a group of documents have the same feature vector is often a function of the method used to convert the original documents into feature vectors. Even though the feature vectors are the same, the documents themselves may not be identical. Given that most preprocessing (ours certainly included) does not take full advantage of the richness of expression in natural languages, one would in fact expect this situation to arise, since the preprocessed document is an approximate representation of the original document. Our preprocessing is quite drastic in terms of how much information in the original document is not used. For example, we do not save information about language structure, word order, or word frequency (we just save information on whether each word was or was not used in each document).

c. The original documents are in fact identical.

   In our system, for example, having multiple documents "with the same tokens" is not too unusual when processing only document titles. The name of a weekly column might appear repeatedly in a newspaper corpus, but its contents (and therefore its categories) may well differ from week to week.

9. portions of documents may be garbled, missing, or reordered

10. there may be formatting errors

    For example, special symbols used to denote the end of a document body may be missing.

While these characteristics certainly make it more of a challenge to work with these documents, the problems inherent in processing such documents are typical of most real-world data and so need to be acknowledged and solved.

We were very conservative in how we preprocessed the data. Our preprocessing is fast, extracting a minimum of information from the documents and not making any modifications to the information that is extracted. We specifically did *not* correct any misspellings, either in the text of the articles, or in the names of the categories assigned to each article. Neither did we remove any tokens. The preprocessing step converts the raw data in the corpus into labeled examples. To do this, the preprocessor does the following:

1. uncompresses and unpacks the corpus
2. performs a rough structural parse

   This determines where the major structural aspects of each document are located – where the document identification information is at, where the title is at, where the body is at, etc.

3. separates document text and category information
4. constructs a table of categories that were used in one or more documents
5. tokenizes the text

6. constructs a table of tokens that were used in one or more documents

   This table is often referred to as the "dictionary", since in many systems it also contains allowed parts of speech, allowed stems, and other information. In our case, it is just a list of tokens that are present in the corpus.

7. prepares the labeled examples

   The documents are converted to feature vector format, with attributes representing those tokens present in the particular document and labels representing the categories to which this document was assigned by a human indexer.

In these experiments, each token in a document is a feature. Each feature is boolean-valued; either the token does or does not appear in the document. The labels are also boolean-valued – the document either is or is not in each category.

There are several possible tokenizing methods. The experiments reported in this paper tokenize text by breaking up the text stream into tokens at whitespace or punctuation. Punctuation is removed, and all letters are downshifted. For example, these two sentences:

> I'm taking the 8:45 train to Dublin on
> 4/5/97. It doesn't stop in Osco-bosco.

will be tokenized thusly:

> i m taking the 8 45 train to dublin on
> 4 5 97 it doesn t stop in osco bosco

Our goal was to perform very simple preprocessing that quickly and in a straightforward manner converts the documents into feature vectors. There is no natural language processing going on of any kind. All of the intelligence for determining a correct categorization method will rest in the way the examples are used and in the learning algorithms that are employed. Note also that this preprocessing method also allows us to more easily utilize ALC systems in other domains. In other words, since the preprocessing has "only" converted the input text to feature vectors (and has *not* done "some of" the text categorization), we can more easily perform classification experiments using other data sets that are already in feature vector form, such as those available from the UCI Machine Learning Database Repository [Blake et al. 1999] or the UCI Knowledge Discovery in Databases Archive [Bay 1999]. These data sets could be text or non-text data sets.

Recall from our earlier discussion (ref. Equation (7) on page 117) that the size of the input to the text categorization system is $O(dt_a)$, where d = number of documents in the corpus and t_a = average number of unique tokens in each document. Recall that l_a is the average number of tokens in each document. Then the input to the preprocessor is $O(dl_a)$. One can see by examining the list of tasks done during preprocessing (uncompress and unpack corpus, perform a rough structural parse, separate text and category information, construct list of category names, tokenize the text, construct the dictionary, prepare labeled examples) that each step is of time complexity $O(dl_a)$, and so preprocessing overall is $O(dl_a)$. (We realize that "parsing" is not usually linear in the size of the input – our parser is linear in the preprocessor input size – it is only breaking up the input into documents and then delineating the various major parts of each document). Recall that in our experiments $l_a \approx t_a$, so preprocessing is also linear in t_a. Typical execution speeds for the parts of preprocessing that are unusual in our system (due to the minimal preprocessing) are: [1]uncompress, unpack, parse, separate: ≈14 documents per second, and [2]tokenize and create dictionary: ≈19 documents per second. This is using code written mainly in Bourne Shell script, sed, and awk, and so would certainly run faster if coded in C or C++.

We used statistical tests to help us evaluate the results of experiments. The statistical software we used was |STAT [Perlman]. Our main statistical tool was the repeated measures analysis of variance test, hereafter "ANOVA". The null hypothesis is that all of the systems actually perform the same on the average, and ANOVA tells us whether or not there is a significant difference among systems for various performance measures. We performed single factor tests, with the factor being the system. We generally require results significant at the 0.01 level before we will claim that a relationship exists, but we will usually state the computed probability or the significance level if we are claiming that a relationship exists.

As discussed earlier, we form training and test sets using random splitting. Given a corpus, one specifies the number of training examples, the number of test examples, and the seed for the random number generator. The generated sequence of numbers is used to randomly divide the corpus into training and test sets. For a given corpus and a given category, there are good and bad splits. A split is good if the training and test sets are each representative of the corpus (and of each other). If the training and test sets are not representative of the corpus as a whole (or of each other), then the learner will not be able to completely "learn the corpus" from the training set, and neither will the test set actually test for all of the knowledge in the corpus. Hence we use repeated trials and then compute averages of various performance measures. By repeating our tests for several different splits, we make it more likely that our performance measure values are on the average actually indicative of the system's overall behavior.

One might ask how many times one needs to repeat the cycle of splitting, training, and testing. One of the results of the Central Limit Theorem in statistics is that, regardless of the distribution of the population, the distribution of sample means will approach a normal distribution as we perform more and more cycles [Gravetter and Wallnau 1985]. As a practical matter, one needs to know how many cycles are needed so that the resulting distribution will be "close enough" to normal. This depends on the underlying distribution of the population being sampled, with few cycles being required if that distribution is itself normal, and more being required if it is quite different from normal. It is surprising how few cycles are actually needed in order to get very close to a normal distribution of the means. Empirically it is usually the case that the number of repetitions falls between 4 and 100 [Snedecor and Cochran 1989], and most of the time it is around 25 or 30 [Gravetter and Wallnau 1985].

While different splits have different examples in the training sets (and different examples in the test sets), in our experimental design all trials in a series have the same *number* of examples in the training set and the same *number* of examples in the test set. Therefore, in terms of contingency table entries, the total of all of the entries $a + b + c + d$ is equal to the number of examples in the test set and will be the same for each of the trials in a series. Since $(a + b + c + d)$ forms the denominator of the equation for accuracy, this also means that macroaveraged accuracy, categoryaveraged accuracy, splitaveraged accuracy, and microaveraged accuracy are all equal for a series of trials. To see why, assume that we have a performance measure Q that is computed as the ratio of two values $\frac{N}{D}$, where N = numerator, D = denominator. (Of the performance measures we have examined, this is the case for accuracy, precision, recall, and F_β).

Let k be the number of trials. Then:

$$Q_{macro} = \frac{\frac{N_1}{D_1} + \frac{N_2}{D_2} + \cdots \frac{N_k}{D_k}}{k}$$

$$= \frac{N_1}{D_1 k} + \frac{N_2}{D_2 k} + \cdots \frac{N_k}{D_k k}$$

If all of the D_i are equal (to, say, D), which in our case is true for the accuracy performance measure, then:

$$\begin{aligned} Q_{macro} &= \frac{N_1}{Dk} + \frac{N_2}{Dk} + \cdots \frac{N_k}{Dk} \\ &= \frac{N_1 + N_2 + \cdots + N_k}{Dk} \\ &= \frac{N_1 + N_2 + \cdots + N_k}{D_1 + D_2 + \cdots + D_k} \\ &= Q_{micro} \end{aligned}$$

We will refer to the average accuracy values that we compute as macroaveraged since that is how most people think of them.

Deciding how large to make the test set is often problematic. Usually one has a limited number of examples. On the one hand, one wants a lot of training examples, since this usually means that one can more thoroughly train the learner – it will be more accurate since it has seen more examples. But since the training and test sets are disjoint, this will mean having a small test set. Small test sets are more likely to be unrepresentative of the corpus, so the fantastic learner that was trained using all of those training examples may get very poor performance measurement scores because the test set is too small to thoroughly test what the learner actually knows. A thumb rule often used when the corpus size is small is to use ⅔ of the examples for training and ⅓ for testing.

The Reuters-22173 corpus is large by machine learning standards – it contains 22,173 examples. And the other text corpora we used also generally had large numbers of examples. So having to worry about training set size versus test set size for the reasons discussed above was not a major issue. However, since one of our goals was to develop methods that use fewer examples during learning without sacrificing accuracy, we did want to have the training set as large as possible – so that we could clearly see the effects of any reduction in number of training examples used that we were able to make.

If one repeatedly splits a corpus into training and test sets in a random manner (so that each is on the average representative of the corpus as a whole) and performs a sequence of training and testing, then one has the following results [Weiss and Kulikowski 1990, Fayyad and Simoudis 1996]:

1. with 1,000 test cases, the difference between true and predicted error is below 1% (with 95% confidence)
2. with 5,000 test cases, predicted error is virtually identical to the true error … 5,000 test cases will unquestionably give one enough accuracy to distinguish true differences between two classifiers

In most cases, we elected to use slightly more than 1,000 test examples. The additional 4,000 examples needed to get to the next plateau was too big of an expense for us. We decided that those examples would be better utilized in the training set.

How one initializes certain variables is important in several learning algorithms, including winnow (initialize weights) and perceptron (initialize weights and θ). One wants to initialize the weights in such a manner that the hyperplane does not have to move too far in order to be in a highly accurate position, since the process of moving the hyperplane (learning) takes time. Moving the hyperplane also uses training examples, and one of our goals is to decrease the number of examples used. However, one can not use any information about the distribution of label values in the corpus when initializing the learners, since that would be tantamount to peeking at the answers to the test questions.

The equation of a hyperplane is:

$$\sum_{i=1}^{i=n} w_i x_i = \theta$$

Recall that n is the number of attributes, the x_i are elements of the feature vector, and each x_i is either 0 or 1.

Let w_{ave} be the value of weight such that if all $w_i = w_{ave}$, then the hyperplane would bisect the space of all possible data points (on the average). We are at this point looking at all possible data points (there are 2^n of them), not at actual data points in the corpus. Then the hyperplane equation becomes:

$$\sum_{i=1}^{i=n} w_{ave} x_i = \theta$$

$$w_{ave} \sum_{i=1}^{i=n} x_i = \theta$$

Next, let k = the average number of x_i that are 1 in each $\vec{x}$; the remaining x_i are 0. We can compute k easily. As we go through the feature vectors, we tally how many 1 bits there are (in the attributes) and divide this total by the number of documents in the corpus. This need only be done once for each corpus, and in fact can be done during the preprocessing phase when the feature vectors are being constructed. Note that we are not looking at the label values, just at the attribute values.

The idea behind our using k is that we would like to take into account the distribution of actual corpus points in the space of possible points. We do not want to initialize the hyperplane to the "center" of the space of all possible points if (as is usually the case) the actual data points are not distributed uniformly over the space. If the actual data points were uniformly distributed over the space, then on the average each document would contain $\frac{n}{2}$ tokens and so k would equal $\frac{n}{2}$.

Empirically, $k \ll \frac{n}{2}$, so we would like to factor this in to how we initialize the hyperplane position.

Using k, we have:

$$\sum_{i=1}^{i=n} x_i = k$$

Therefore:

$$w_{ave} k = \theta$$

$$w_{ave} = \frac{\theta}{k}$$

Note also that since $w_{ave} = \frac{\theta}{k}$ and since each axis intercept is at $\frac{\theta}{w_{ave}}$, each axis intercept is at $x_i = k$.

Using $w_{ave} = \frac{\theta}{k}$ will cause us to initially position the hyperplane so that it on the average bisects the space of all possible data points, biased by the fact that we also know that on the average each of the actual documents contains k attributes whose value is 1.

Figure 11 shows two examples of the results of this equation. Both examples are for 2 dimensions, so the hyperplane is a line, and there are $2^2 = 4$ possible data points in the corpus. The symbol ◊ represents the location of the actual data points in the corpus – the data points that were used to compute k. Assume $\theta = 2$. In Figure 11, left, we have all 4 possible points actually in the corpus, so $k = \frac{4}{4} = 1$ and so the axis intercepts are at 1. In Figure 11, right, we have only 3 data points actually in the corpus. $k = \frac{4}{3} = 1.33$. Note that we did not need to know the label values of any of the points in order to position the hyperplane. The claim is that on the average the hyperplane bisects the space of all possible points, biased by the

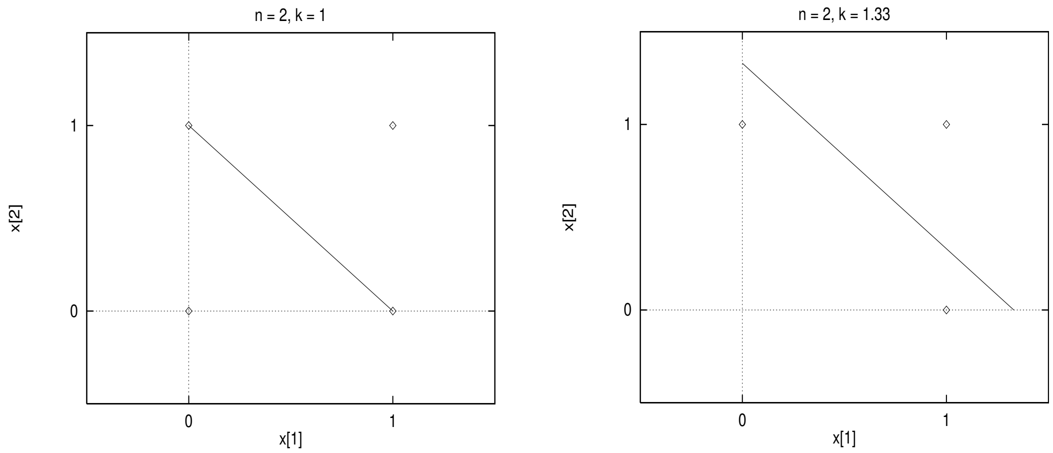

Figure 11. Examples of Average Hyperplane Location

fact that we also know the value of k. No claim is made that the hyperplane thus positioned does very well at classifying.

If we have a hyperplane with each $w_i = \frac{\theta}{k}$, then we will have a hyperplane that on the average will bisect the space of all possible data points for n dimensions and for the given value of k. This seems to be a good initial position for the hyperplane, in that it is a position that minimizes (on the average) the amount of shifting that the hyperplane has to undergo in order to relocate to a final position that is highly accurate. The hyperplane shifts only a certain amount in response to each example that it learns from, so it is important to have the learner start the learning process at a reasonable location. Note that this positioning of the initial hyperplane does not use any information about actual label values. In fact, it does not even make any assumption about which class (NO or YES) is larger.

Note that this initialization method for $\vec{w}$ automatically scales to the initial value used for θ. We use an initial value of 1.0 for θ (for both the winnow and perceptron learners).

Individual committee members are randomly initialized to slightly different hyperplanes so that they represent different initial hypotheses. A difficulty encountered in initializing a committee of learners is that one wants the learners to be different (otherwise they will always agree), but not too different (or they will too often disagree). Once w_{ave} has been computed, we use it to define a range of values that extends from slightly above 0 to slightly below $2 \times w_{ave}$, where w_{ave} is at the center of the range. We avoid initial weights equal to 0 because winnow uses multiplicative updating and so a weight of 0 will remain 0. We avoid weights < 0 because winnow requires the use of positive weights.

Initial weight values are then generated from a uniform random distribution over this range. If there are M committee members and W weights for each member, then one method of initialization generates a random value for each of the $M \times W$ weights. Another method uses the same value to initialize all weights for a particular member (and so this method generates M different weight values). We used both of these methods to see if the differing degree of randomness in the initial weight values made any difference. Overall, which of these two initialization methods were used did not seem to make a significant difference in the final committee accuracy.

5.2 Corpora Used

5.2.1 Reuters-22173

The Reuters-22173 corpus is a collection of 22,173 Reuters newswire articles ("documents") from 1987 [Reuters-22173]. It is a 25Mb full text corpus. Each article has been assigned to categories by human indexers. Typical categories are "topic=grain", "topic=gold", "place=Canada", and "topic=trade". An article may be assigned to any number of categories, including none.

Full corpus statistics for Reuters-22173 (after our preprocessing):

- 22,173 documents
- number of unique tokens:
  - 16,600 (titles only)
  - 48,446 (full text)
- average number of unique tokens per document:
  - 7.78424 (titles only)
  - 81.8523 (full text)
- 679 categories

The 679 categories are in turn divided into 6 subcategories: topics, places, people, organizations, exchanges, companies. Most research has been done using the topics subset of categories, which contains 184 categories.

There are 3 standardized predefined splits, named after the researcher who defined and first used each of them: Hayes, Lewis, and Apté.

We used this corpus because it is very real-world, and it was readily available when our research started. Also, several others have used it, which we felt would facilitate comparison of our systems with the results of others. See especially

[Hayes et al. 1990, Hayes and Weinstein 1991, Lewis 1991b, Lewis 1992, Apté et al. 1994, Lewis and Ringuette 1994, Cohen and Singer 1996, Cohen and Singer 1999, Yang 1999].

While many have used the Reuters-22173 corpus in tests of their systems, there has unfortunately been no standard way of using the corpus, and so meaningful comparisons are somewhat difficult to obtain [Yang 1996, Reuters-22173]. One can be faced with problems such as there not being enough information given in the paper to allow one to replicate the experiment, or (due to ambiguity in the corpus format) different researchers may differ in how the title or body text is extracted, which documents are used and for what (training or test), which categories are used, etc.

As regards how we extracted information from the corpus: we used as the document title the text following ˆB up to <newline><space>, and we used as category labels any string not containing whitespace that fell in one of the category fields. In particular, we did not compare the labels in the corpus to any list of legal label names and thus discard typographical errors.

We used the 10 most frequently occurring topic categories, as listed in [Lewis 1991b], for our experiments. These categories and their characteristics over the entire corpus are given in Table 4. Note that on the average, there are only 5.04% positive examples in each category.

Table 4 is typical of the information we will report on each of the categories used in the various corpora. The rows of the table are ordered based on percentage of positive examples, going from highest to the lowest. By way of explanation of the columns reporting the number of inconsistent examples: as discussed earlier, a group of documents is inconsistent if they all have the same attribute vector but have differing label values. Inconsistent examples are of particular interest to us, since for the learning algorithms that we are using, each committee member is trying to position a hyperplane so that it separates NO and YES examples for the

TABLE 4. Reuters-22173 Categories

category name	*fraction* YES	*number of inconsistent examples titles only*	*full text*
topic=earn	0.1925	10	2
topic=acq	0.1184	5	4
topic=money-fx	0.0391	41-51	15-17
topic=grain	0.0305	16	5
topic=crude	0.0302	8	2
topic=trade	0.0269	21-22	16-17
topic=interest	0.0250	41-68	10-11
topic=wheat	0.0149	15	4
topic=ship	0.0145	6-8	1-2
topic=corn	0.0121	9-10	2

category of interest. This will be impossible if there are inconsistent examples in that category.

When inconsistencies occur, the preprocessor is not able (and is not allowed to try) to determine which labels are actually correct. One can therefore have uncertainty in the number of inconsistent examples. For example, assume that we are looking at the category "place=denmark" for titles only. There is one group of six documents that have the same title, but one is labeled NO and five are labeled YES. We would then say that the number of consistent examples in this group of documents is either 1 or 5 (and thus the number of inconsistent examples is either 5 or 1). In general there are several such "inconsistent groups of examples" for the same category. Say there are h such groups. Rather than actually keeping track of all possible values of the number of inconsistent examples for that category [out of the 2^h (duplicated) possibilities], we list in Table 4 only the minimum and maximum values of the number of inconsistent examples (totaled over all h groups). This especially seems justified since the values given are for the entire corpus, and so one can most likely obtain most values in the range depending on the size of the training and test sets and how the inconsistent documents end up being split. For the above "place=denmark" example, the entry in Table 4 would therefore be "1-5".

A single number can appear in the "number of inconsistent examples" columns. See for example "8" for the category "topic=crude", titles only, in Table 4. This means that it happens that, over the entire corpus, exactly 8 documents will be inconsistent. In this particular case, there are 8 groups of inconsistent examples, each group containing 2 documents, one of which is labeled NO and one of which is labeled YES. The same table entry of "8" would occur if there were 4 groups of 4 documents (each group split 2:2), or one group of 16 inconsistent documents split 8:8, etc.

In most experiments using this corpus, we used several different random splits – typically 20. This results in a large number of trials (20 splits $\times$ 10 categories = 200 trials). We verified that the number of splits being used in each experiment was enough by running some experiments with far greater numbers (as many as 200), but the results we obtained with those increased numbers were not significantly different. For each split, we divided the 22,173 documents into 21,000 training documents and 1,173 test documents. We usually used titles only, since this is faster and gives quite good results.

Figure 12 shows two sample documents from the Reuters-22173 corpus.

```
^D PATTERN-ID 302 TRAINING-SET
 1-APR-1987 12:15:26.41
TOPICS: corp-news loan cbond    END-TOPICS
PLACES: usa    END-PLACES
PEOPLE:     END-PEOPLE
ORGS:     END-ORGS
EXCHANGES:     END-EXCHANGES
COMPANIES:     END-COMPANIES
^E^E^EA RM^M
^V^V^Af1611^_reute
r f BC-BROWN-GROUP-<BG>-DEBT   04-01 0110^M
^B^M
BROWN GROUP <BG> DEBT DOWNGRADED BY S/P^M
    NEW YORK, April 1 - Standard and Poor's Corp said it^M
lowered 128 mln dlrs of Brown Group Inc's senior debt to^M
A-minus from A and commercial paper to A-2 from A-1.^M
    The agency said a decline in sales and profits were little^M
offset from the company's retail business. Also, foreign^M
producers eroded Brown's assets by gaining market share.^M
    Brown's debt ratio increased to 49.8 pct by year-end 1986^M
as higher inventories, and a four-year stock repurchase program^M
has required additional financing, S and P said.^M
    Restructuring efforts should improve productivity but not^M
in the forseeable future, the agency added.^M
 Reuter^M
^C
```

```
^D PATTERN-ID 5644 TRAINING-SET
26-FEB-1987 16:35:24.57
TOPICS: money-supply END-TOPICS
PLACES: usa END-PLACES
PEOPLE:  END-PEOPLE
ORGS:  END-ORGS
EXCHANGES:  END-EXCHANGES
COMPANIES:  END-COMPANIES
^E^E^ERM A^M
^V^V^Af0036^_reute
b f BC--FEDERAL-RESERVE-MONE   02-26 0092^M
^B^M
FEDERAL RESERVE MONEY SUPPLY REPORT - FEB 26^M
    One Week Ended Feb 16^M
 M-1........................736.7 up.......2.1^M
 Previous week revised to....734.6 From...734.2^M
 Avge  4 Weeks (Vs Week Ago).735.0 Vs.....733.5^M
 Avge 13 Weeks (Vs week Ago).731.8 Vs.....729.8^M
 Monthly aggregates (Adjusted avgs in billions)^M
 M-1 (Jan vs Dec)............737.6 Vs.....730.5^M
 M-2 (Jan vs Dec)..........2,820.1 Vs...2,798.4^M
 M-3 (Jan vs Dec)..........3,513.6 Vs...3,488.1^M
 L...(Dec vs Nov)..........4,141.5 Vs...4,110.5^M
 Domestic Debt(Dec vs Nov).7,604.4 Vs...7,519.8^M
^M
 Reuter^M
^C
```

Figure 12. Reuters-22173 Examples (#302 and #5644)

5.2.2 Reuters-21578

The Reuters-21578 corpus is a collection of 21,578 Reuters newswire articles [Reuters-21578]. It was developed to provide a more standardized corpus for testing text categorization systems and is based on the Reuters-22173 corpus. The formatting was modified and clearly documented, 595 duplicate documents were removed (22,173 - 595 = 21,578), documents were reordered (so that the articles are in chronological order), and typographical errors in the category label fields were corrected.

Full corpus statistics for Reuters-21578 (after our preprocessing):

- 21,578 documents
- number of unique tokens:
  - 15,842 (titles only)
  - 48,428 (full text)
- average number of unique tokens per document:
  - 7.74377 (titles only)
  - 81.8377 (full text)
- 451 categories

The 451 categories are in turn divided into the same 6 subcategories as for Reuters-22173: topics, places, people, organizations, exchanges, companies, but in the Reuters-21578 corpus the companies field is always empty. Most research has been done using the topics subset of categories.

There are 3 standardized predefined splits, analogous (insofar as is possible) to the similarly-named splits in Reuters-22173: ModHayes, ModLewis, and ModApté.

This corpus is relatively recent (made available in December 1996). Many researchers (ourselves included) had started projects using Reuters-22173 and

TABLE 5. Reuters-21578 Categories

category name	*fraction* YES	*number of inconsistent examples* *titles only*	*full text*
topic=earn	0.1848	29-724	0
topic=acq	0.1134	3-736	0
topic=money-fx	0.0371	33-782	8-12
topic=crude	0.0294	15-745	2
topic=grain	0.0291	18-745	2
topic=trade	0.0256	15-747	8
topic=interest	0.0238	37-803	9-10
topic=wheat	0.0167	12-747	1
topic=ship	0.0141	13-735	1
topic=corn	0.0118	9-745	0

continued using that corpus, but some results using Reuters-21578 are starting to appear in the published literature [Dagan et al. 1997, Dumais et al. 1998, Joachims 1998, Cohen and Singer 1999, Scott and Matwin 1999, Weiss et al. 1999, Yang and Liu 1999].

Basically, we view this corpus as a full text only corpus, in that a quite large number of the articles (737 or 3.4%) do not have a title indicated. One could admittedly create one's own titles by having an unbiased human read such articles and create titles, or by taking as the title the first sentence in the article. Either approach, however, would make it extremely difficult to compare results to those of other researchers unless they happened to use the same title generation method.

Since we typically do run "titles only" tests, we chose to treat articles that did not have a title indicated in the corpus as having no title. Since there are such a large number of such articles, this results in quite large numbers of "inconsistent examples" for the titles only case. This is because the 737 titleless articles all have the same title (the empty title), but have widely differing categorizations based on the content of the article body.

We used the same 10 topic categories for our experiments using Reuters-21578 as were used in Reuters-22173. These categories and their characteristics over the entire corpus are given in Table 5. Note that on the average, there are only 4.83% positive examples in each category.

Figure 13 shows two sample documents from the Reuters-21578 corpus. These are the same documents as were shown in Figure 12 for the Reuters-22173 corpus. (→ indicates where I have inserted a line break so that long lines would fit on the paper.)

A comparison of Figures 12 and 13 shows some of the substantive (to us) differences between these two corpora. Notice that the first document in each figure has had all of its topics category labels removed in going from Reuters-22173 to Reuters-21578. All 3 of the topics listed in the Reuters-22173 version (corp-news, loan, and cbond) were illegal topic names. For the second document in Figure 12, in Reuters-22173 we extracted the title of "FEDERAL RESERVE MONEY SUPPLY REPORT - FEB 26", but in Reuters-21578 this document has no title.

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"→
OLDID="302" NEWID="12074">
<DATE> 1-APR-1987 12:15:26.41</DATE>
<TOPICS></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;A RM
&#22;&#22;&#1;f1611&#31;reute
r f BC-BROWN-GROUP-&lt;BG>-DEBT   04-01 0110</UNKNOWN>
<TEXT>&#2;
<TITLE>BROWN GROUP &lt;BG> DEBT DOWNGRADED BY S/P</TITLE>
<DATELINE>    NEW YORK, April 1 - </DATELINE><BODY>Standard and Poor's Corp said it
lowered 128 mln dlrs of Brown Group Inc's senior debt to
A-minus from A and commercial paper to A-2 from A-1.
    The agency said a decline in sales and profits were little
offset from the company's retail business. Also, foreign
producers eroded Brown's assets by gaining market share.
    Brown's debt ratio increased to 49.8 pct by year-end 1986
as higher inventories, and a four-year stock repurchase program
has required additional financing, S and P said.
    Restructuring efforts should improve productivity but not
in the forseeable future, the agency added.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
```

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"→
OLDID="5644" NEWID="101">
<DATE>26-FEB-1987 16:35:24.57</DATE>
<TOPICS><D>money-supply</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;RM A
&#22;&#22;&#1;f0036&#31;reute
b f BC--FEDERAL-RESERVE-MONE   02-26 0092</UNKNOWN>
<TEXT TYPE="UNPROC">&#2;
FEDERAL RESERVE MONEY SUPPLY REPORT - FEB 26
    One Week Ended Feb 16
 M-1.......................736.7 up.......2.1
 Previous week revised to....734.6 From...734.2
 Avge  4 Weeks (Vs Week Ago).735.0 Vs.....733.5
 Avge 13 Weeks (Vs week Ago).731.8 Vs.....729.8
 Monthly aggregates (Adjusted avgs in billions)
 M-1 (Jan vs Dec)...........737.6 Vs.....730.5
 M-2 (Jan vs Dec).........2,820.1 Vs...2,798.4
 M-3 (Jan vs Dec).........3,513.6 Vs...3,488.1
 L...(Dec vs Nov).........4,141.5 Vs...4,110.5
 Domestic Debt(Dec vs Nov).7,604.4 Vs...7,519.8
 Reuter
&#3;
</TEXT>
</REUTERS>
```


Figure 13. Reuters-21578 Examples (#12074 [was 302] and #101 [was 5644])

6. Experiments Performed and Results

This chapter describes the main experiments that we performed and the results obtained. The results of some of these experiments have been published previously [Liere and Tadepalli 1996, Liere and Tadepalli 1997, Liere and Tadepalli 1998a, Liere and Tadepalli 1998b]. In this document, we also update these previously reported results.

6.1 Where Are We Going?

Preliminary experiments had shown that ALC, as compared to supervised learning with a single learner, can result in learning methods that use far fewer examples and also take less execution time, but still achieve almost the same accuracy [Liere 1997]. We demonstrate here this effect, as it is the main motivation for our research. This experiment not only demonstrates the main effect that we investigated, but also will serve to introduce and explain the types of graphs and tables we will typically use to report results of many of our experiments.

We ran a series of trials on 3 systems. We typically identify the different systems in an experiment using a system number as well as a descriptive name. On graphs, the system number is used. In the related discussion, when we use the descriptive name and the system number, we will put the system number in { } if we need to clarify that it is the system number. The 3 systems in this experiment were:

{1} ALC

Active Learning with Committees, with a 7 member committee, using the AllMargin method to determine whether or not to see the label, winnow as the learning algorithm, and majority voting as the prediction method. In the majority voting prediction method, if there is a tie vote, then the prediction is the majority class of the training examples that have been used so far. If that is also a tie, then a prediction that alternates between NO and YES is given.

{2} SL-WI

supervised learning system (uses all training examples), 1 committee member, using the winnow learning algorithm

{3} SL-NB

supervised learning using the naive Bayes learning algorithm

Thus {2} and {3} are our baseline systems.

We used the Reuters-22173 corpus, full text (as opposed to titles only). This results in there being 48,446 tokens in the dictionary (and so 48,446 attributes in the feature vector representing each document).

Figure 14 shows elapsed processor time (recall, this is always given in seconds) as a function of the number of training examples used for each of the 3 systems. There is one dot on the graph for each trial of each system. We have added boxes around the cluster of dots for each system to make it easier to visually identify the operating region for each system. Each box is labeled with the system number. Since systems 2 and 3 are supervised learners, their boxes collapse into vertical lines. The symbol ◊ ("diamond-dot") denotes the location of the average elapsed processor time and average number of training examples used for each system.

Note that the clusters are some distance apart. This dramatically illustrates that, in this experiment, the differences among the systems in terms of both the number of labeled training examples used and the elapsed execution time were

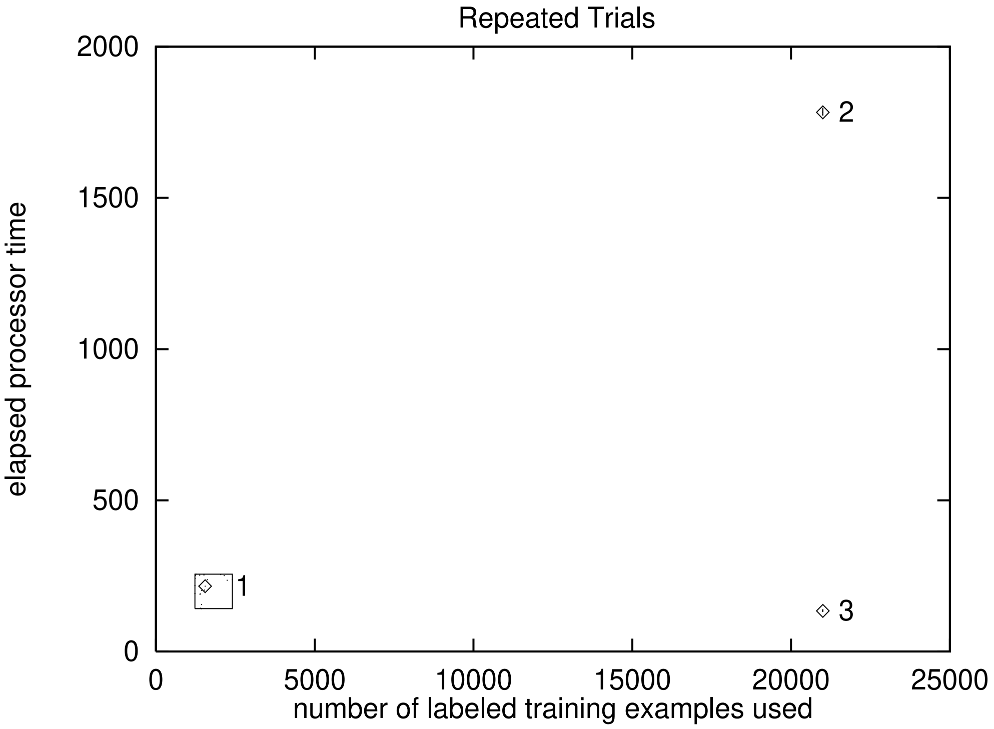

Figure 14. Elapsed Time versus Number of Training Examples Used [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 1 member, winnow; 3 = supervised learning, naive Bayes]

quite large. Observe also that, for each system, the dots form very tight clusters. This indicates that the variation in behavior within each system, for both the number of training examples used and elapsed processor time, was quite small.

These effects are typical of many of our experiments and are important points to note. If the elapsed processor time and number of training examples used for a particular system do not vary much, then that indicates that the system is quite consistent in its use of time and number of training examples, and so our confidence that the average behavior is not too much different than any one trial's actual behavior is high, independent of split or category. When clusters for different systems do not overlap, it means that those systems are quite distinct in their performance as regards time and number of training examples used.

Figure 15 is an accuracy learning trace for the 3 systems. We will often look at learning traces – a learning trace shows how a particular performance measure varies during the learning process – as training examples are used. One can therefore graph learning traces for any performance measure. We use a log scale on the horizontal axis. Note that the accuracy measure used in Figure 15 is "Macroaveraged Accuracy", since that is the normal accuracy measure used in machine learning tests. However, as we discussed earlier, a side effect of how our experiments are designed is that microaveraged and macroaveraged accuracy are equal.

Two cautions are in order as regards examining learning traces. First, one might have a learning trace that rises and then approximately levels off, such as is the case for system 3 of Figure 15. Does this mean that system 3, which is a supervised learner, has essentially learned all it needs to learn after using (say) 800 examples, and so is actually a very good active learner? Could we stop system 3 after using 800 examples and get accuracies as good as the other two systems, but using fewer training examples? Yes, we could. However … such a decision assumes that the other performance measures are of no concern. In this case, if we peek ahead to Figure 16, we see that system 3 has obtained a high accuracy early in the learning process at the expense of $F_{1.0}$, and that the learner then uses the subsequent training examples to develop a better $F_{1.0}$ performance. So, at least in our domain, we would not want to stop system 3 after using 800 examples and use it for text categorization.

The second caution has to do with the general shape of the learning trace. For the contingency table based performance measures of interest to us (accuracy, precision, recall, $F_{1.0}$), learning traces will generally start at a low to moderate initial values (based on the initial random positions of the hyperplanes) and then rise as training examples are used and learning occurs. One might ask what it means if a learning trace rises to a high point and then, as more training examples are used, starts to drop. This can occur for one of two reasons. The learner may be trading off one performance measure for another – for example, it might be losing recall but gaining precision. As long as this tradeoff is not too extreme and the system that finally results is an overall good performer, such tradeoffs are not a major concern and in fact are to be expected. However, what if this rising and then dropping in a learning trace is seen in several of the performance measures, and there appears to be no trade off occurring – we appear to not be gaining something in return for these performance loses? This can be caused by overfitting, especially if we are looking at the results of one or very few trials. Or it can be due to the learning algorithm being very poorly tuned. Recall the role that δ plays in our systems. It is a relative measure of the distance between the NO and YES clouds, or at least between the main bodies of the clouds. If we, for example, tell the system that the clouds are very far apart when in fact they are very close together, the resulting large values of the learning rate parameters used by winnow and perceptron will cause the hyperplanes to be within the clouds instead of between them, and the learned hypotheses will be poor approximations to the target concept. In cases such as this, we need to change the value of δ. Note that the naive Bayes algorithm has as one of its characteristics that it does not use any learning rate parameter, so at least for standard naive Bayes, if it exhibits this performance, there is nothing we can do to change it.

We can see from Figure 15 that system 1, which is employing active learning, uses many fewer examples than is used by the supervised systems 2 and 3. This aspect of the graph is consistent with the results given in Figure 14. However, Figure 15 also shows that the 3 systems end up with very similar final accuracies, but that the path that each takes to get there is quite different. The final accuracy values are given in Table 6.

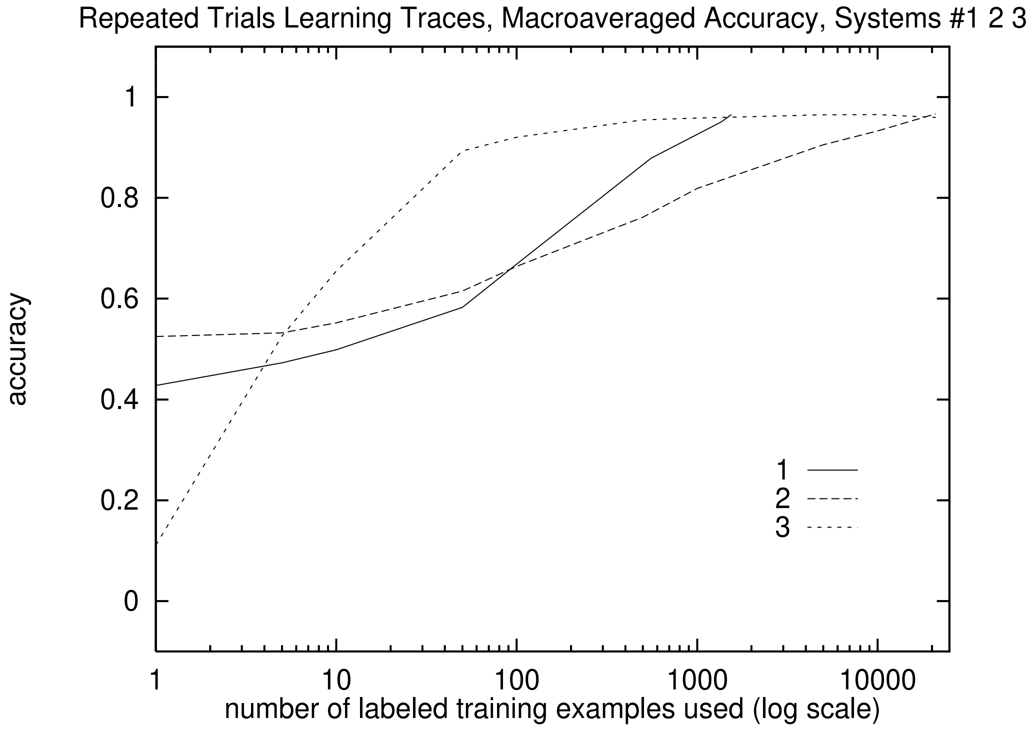

Figure 15. Average Accuracy by System [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 1 member, winnow; 3 = supervised learning, naive Bayes]

TABLE 6. Final Accuracy Values

	- - - *accuracy* - - -	
system	*mean*	*std dev*
ALC {1}	0.965	0.0196
SL-WI {2}	0.967	0.0267
SL-NB {3}	0.960	0.0444

ANOVA does not indicate that the system used has a significant effect on final accuracy. It appears that the 3 systems have essentially the same accuracies. This is good from our standpoint, since (from Figure 14) we see that the system using active learning (ALC {1}) runs about as fast as the fastest supervised learner (SL-NB {3}) but uses only 8.5% as many training examples as a supervised learning algorithm, and with no significant decrease in accuracy.

As discussed earlier, a performance measure that is more often of interest in text categorization is $F_{1.0}$, so in Figure 16 we look at $F_{1.0}$ versus number of training examples used for the 3 systems. This is also a learning trace, showing how $F_{1.0}$ varies for each system as it learns. One can also see that in this figure, while the paths differ, the final $F_{1.0}$ values of the 3 systems appear to be about the same. The final $F_{1.0}$ values are given in Table 7. This shows that the ALC system also performs about as well as the supervised learners when judging performance using the $F_{1.0}$ measure.

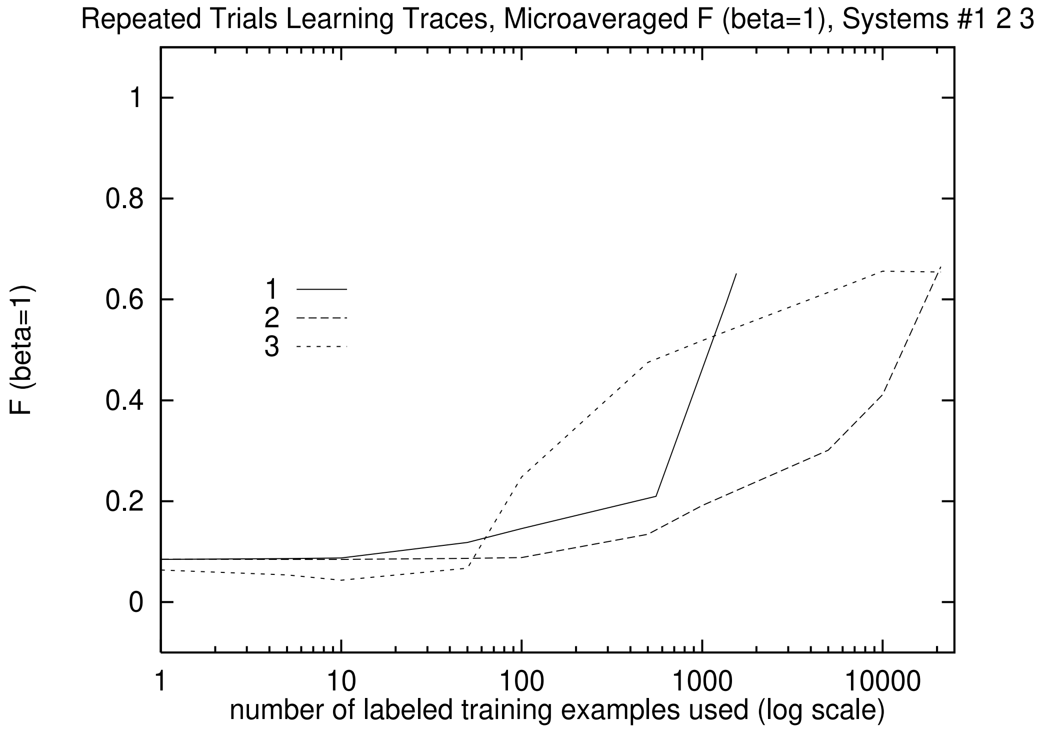

Figure 16. Average $F_{1.0}$ by System [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 1 member, winnow; 3 = supervised learning, naive Bayes]

TABLE 7. Final $F_{1.0}$ Values

system	*mean*
ALC {1}	0.652
SL-WI {2}	0.665
SL-NB {3}	0.654

The remaining experiments focus on investigating how ALC is able to obtain accuracy and $F_{1.0}$ values that are similar to those for a supervised learning system while using far fewer training examples and (usually) much less time. We also look at implications of the differences in the learning path trajectories, especially for systems whose final performance measures are the same. And of course we will also look at ways of improving the performance of ALC. The systems we are dealing with are quite complex, so we perform several experiments, each examining a particular aspect of the problem.

Most of the remaining experiments will be run using the Reuters-22173 corpus with titles only. We use the Reuters-22173 corpus because it has been available to us since this research project began. Although other corpora have since become available, continuing to use Reuters-22173 has allowed us to take advantage of our own historical data in terms of determining whether or not various approaches are better or worse than ones tried in the past. We will, in one of the final experiments, run our system on a different corpus (Reuters-21578) to show that the methods we have developed generalize to this newer corpus.

Using only titles allows us to run our tests faster, and we actually get better results. The tests run faster because the number of tokens in the dictionary (and therefore the number of attributes in each feature vector) is less – 16,600 versus 48,446. We think that the reason for the results being better is that the title of each document is in a sense a summarization of the contents of the entire document, created by the author. It thus distills the meaning of the document into a form that is more concise and that probably also contains less attribute noise. We will, in one of the final experiments, run our system on full text, to show that our system does scale up well, although one could argue that we have already shown that it scales up reasonably well by being able to handle 16,600 attributes. Most machine learning problems involve far fewer attributes. For example, most data sets in the UCI Machine Learning Database Repository [Blake et al. 1999] have fewer than 100 attributes.

6.2 The Number Of Members Needed in the Committee

The question arises as to how large we should make our committees of learners. Both the time and space requirements of most of the methods we are using vary linearly with the number of committee members, so we would like to keep the committees as small as possible. However, since we are relying on the committee members to represent a variety of possible hypotheses, we intuitively might think that the bigger the committee, the better. In this experiment, we investigate how many members we actually need in a committee of active learners.

First, we ran a series of trials on 4 systems. We used the Reuters-22173 corpus, titles only. Each system used QBC-REP to determine whether or not to see the label, winnow as the learning algorithm, and majority voting as the prediction method.

The 4 systems differed only in how many members were in the committee. The 4 systems in this experiment were:

{1} 3 members

{2} 7 members

{3} 100 members

{4} 1,000 members

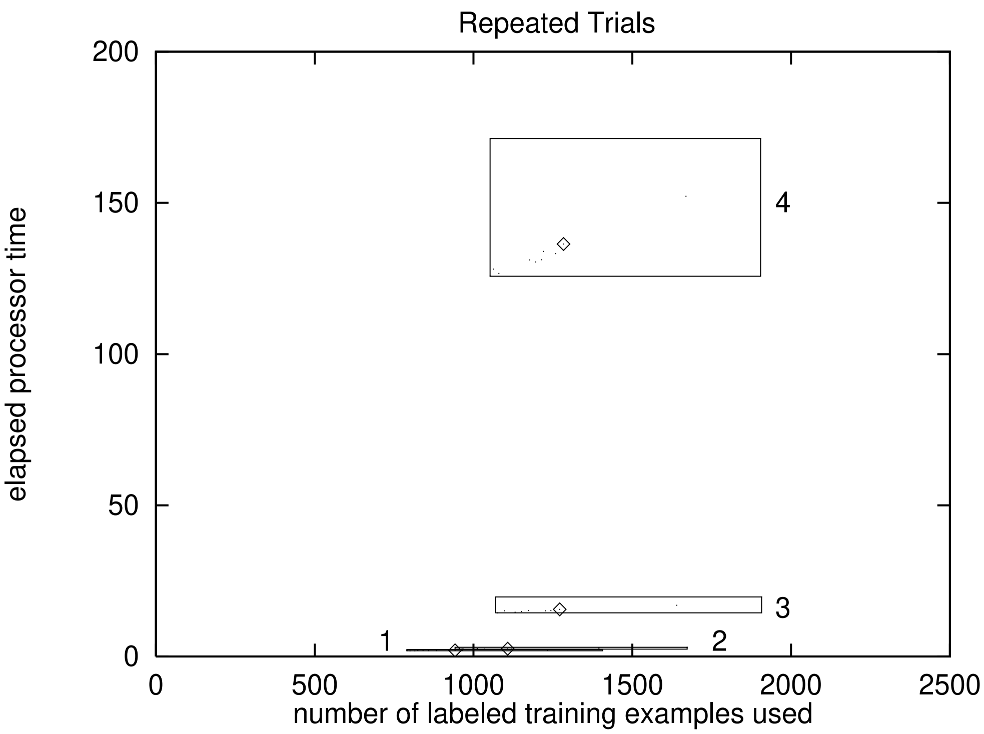

Figure 17. Elapsed Time versus Number of Training Examples Used [1 = 3 members; 2 = 7 members; 3 = 100 members; 4 = 1,000 members]

Figure 17 shows elapsed processor time as a function of the number of training examples used for each of the 4 systems. One can see that, as the number of members in the committee goes up, so does elapsed processor time and also the number of training examples used. Also observe that, for each system, the dots form fairly tight clusters. This indicates that there is not much variation in the behavior within each system, especially for elapsed processor time. Figure 17 also illustrates that the differences among the systems in terms of the elapsed execution time was significant, in that the boxes do not overlap in time. (Systems 1 and 2 do not overlap in time, but that is difficult to see on the graph). The graph also shows that the 4 systems used very similar ranges of numbers of training examples. Also note that elapsed processor time does appear to be approximately linear in the

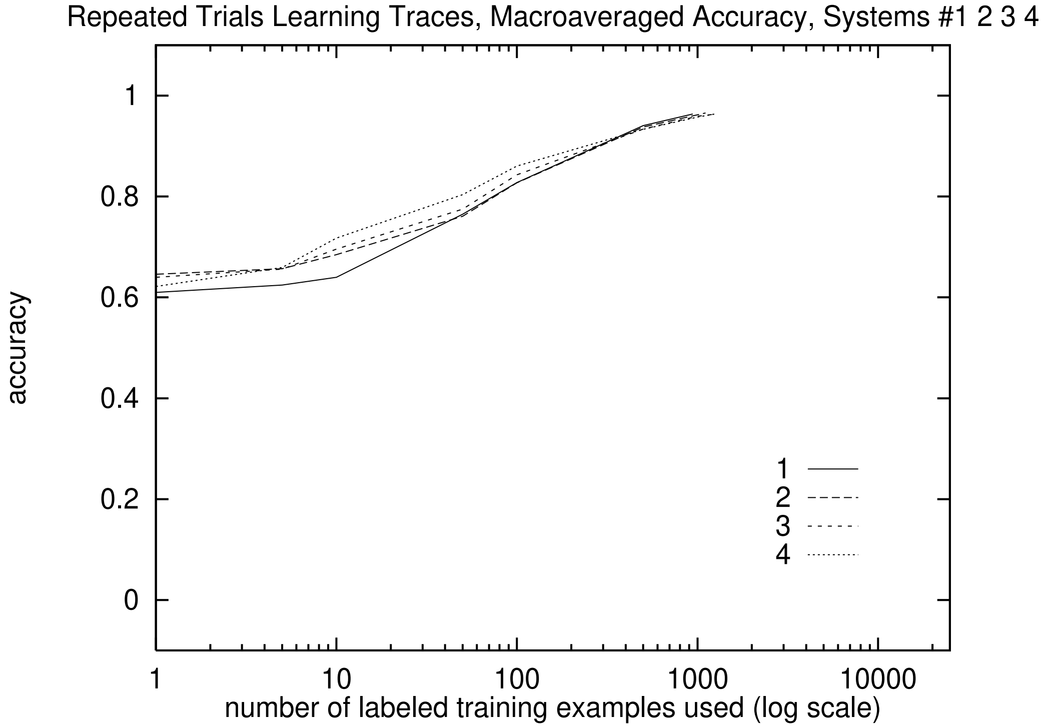

Figure 18. Average Accuracy by System [1 = 3 members; 2 = 7 members; 3 = 100 members; 4 = 1,000 members]

number of committee members. System 3 has 100 members and an average elapsed processor time ≈ 15, and system 4 with 1,000 members has an average elapsed processor time ≈ 137.

It would be difficult to choose which system is best based only on Figure 17. One might be tempted to pick system #1 since it runs the fastest on the average, and also on the average it uses fewer training examples. We know this because of the position of its ◊. We will therefore also look at some contingency table based performance measures. Figure 18 shows the average accuracy for each of the 4 systems as a function of the number of training examples used. We can see that the

TABLE 8. Final Accuracy Values

		- - - *accuracy* - - -	
system	*committee size*	*mean*	*std dev*
{1}	3	0.963	0.0205
{2}	7	0.966	0.0191
{3}	100	0.964	0.0192
{4}	1,000	0.964	0.0205

systems are employing active learning since they each use many fewer examples than would be used by a supervised learner (21,000 for the Reuters-22173 corpus). Note that this aspect of the graph is consistent with the results given in Figure 17. However, Figure 18 also shows that the 4 systems end up with very similar final accuracies, and that the path that each takes to get there is very similar. The final accuracy values are given in Table 8.

ANOVA also does not indicate that the system used has a significant effect on final accuracy. It appears that the 4 systems have essentially the same accuracies. One might therefore still be tempted to pick system #1, since it runs the fastest. We will next look at $F_{1.0}$, since precision and recall are generally of more interest to the text categorization user than accuracy.

Figure 19 shows the average value of $F_{1.0}$ for each of the 4 systems as a function of the number of training examples used. This is also a learning trace, showing how $F_{1.0}$ varies for each system as it learns. One can see that the paths are very similar, but that there does appear to be some differences in final $F_{1.0}$ amongst the 4 systems. The final $F_{1.0}$ values are given in Table 9.

The conclusion that we draw from this experiment and several others similar to it is that a committee size of 3 is too small, 100 is larger than necessary,

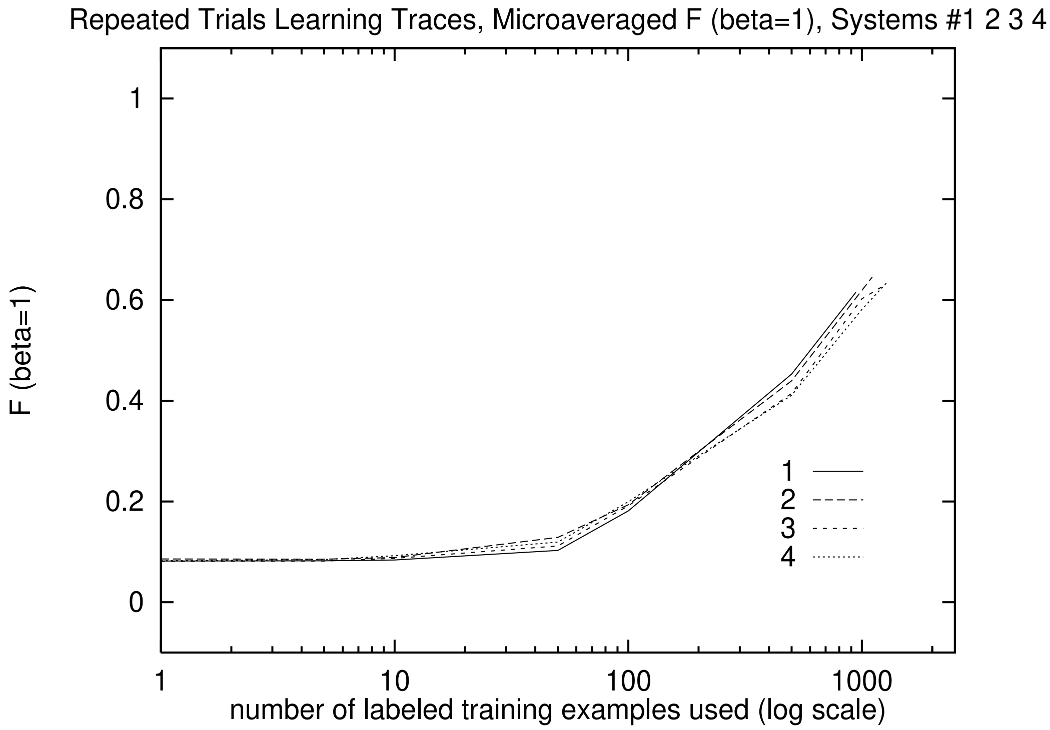

Figure 19. Average $F_{1.0}$ by System [1 = 3 members; 2 = 7 members; 3 = 100 members; 4 = 1,000 members]

TABLE 9. Final $F_{1.0}$ Values

system	*committee size*	$F_{1.0}$
{1}	3	0.615
{2}	7	0.646
{3}	100	0.631
{4}	1,000	0.635

and 7 is about right. We reach this conclusion because $F_{1.0}$ at 7 members is definitely better than at 3 members. Then $F_{1.0}$ drops slightly as we increase the number of members to 100 and 1,000. ANOVA indicates that there is a significant difference in $F_{1.0}$ due to the system being used (p = 0.044), but it also indicates that this difference is mostly due to system 1. ANOVA does not indicate a significant difference in the $F_{1.0}$ values for systems 2, 3, and 4. However, in a sense whether or not the difference in $F_{1.0}$ is significant or not as we go from 7 to 100 to 1,000 members is not relevant, since either the $F_{1.0}$ is most likely unchanged or it is dropping slightly. In either case, we will prefer the 7 member committee because its $F_{1.0}$ is highest, and its use of training examples and time is lower than for 100 and 1,000 member committees.

This type of experiment was repeated often, for differing numbers of members, for different ALC systems, and also for committees of supervised learners. We obtained similar results, in that committees smaller than about 5 or larger than about 25 did not perform as well as committees in the 5 – 25 range. We therefore conducted most of our experiments with committees having 7 – 20 members.

We feel that this effect is probably common in all ALC systems, although the range numbers of members that provide the best results will likely vary with the domain dimensionality and the amount of noise.

It is, however, not obvious why this effect occurs. So we did additional experiments. In the following experiment, we gave systems with 3, 7, 100, and 1,000 members (but otherwise identical) a set of documents and had the committees employ active learning. We looked at the learning traces for the committee as a whole (as we did above) *and* for each of the members in the committee. For these results, individual member learning traces are shown in solid lines, and the location of the learning trace for the committee as a whole is shown by a path of ◊'s. In this particular experiment, winnow is the learning algorithm and prediction was by majority vote.

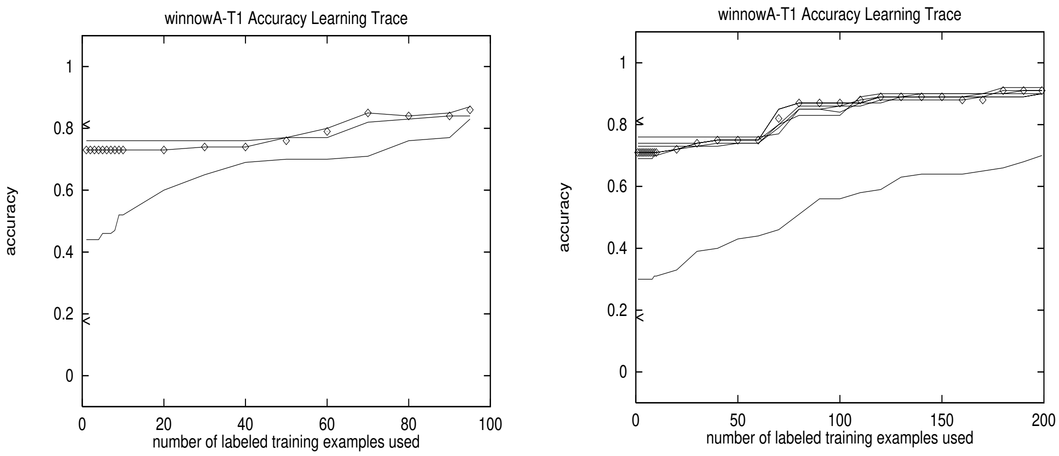

Figure 20. Member and Committee Accuracy: 3 members (left) and 7 members (right)

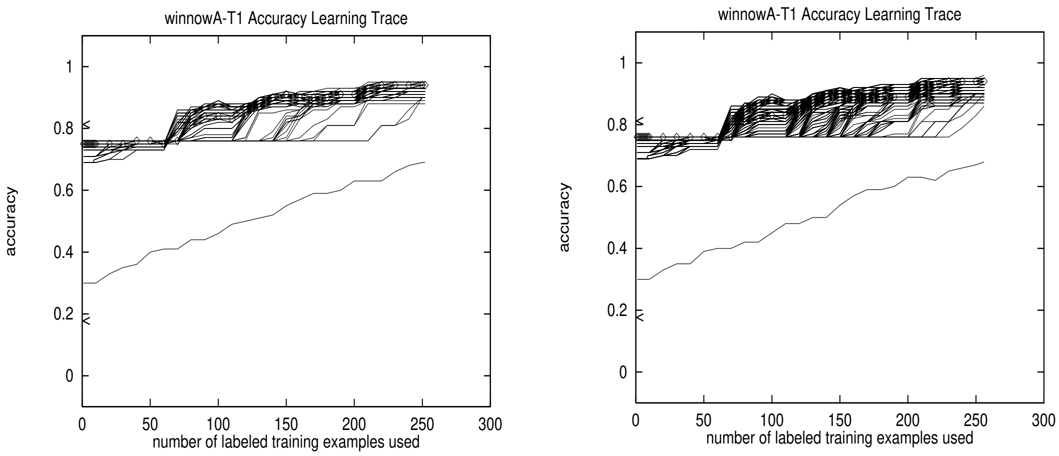

Figure 21. Member and Committee Accuracy: 100 members (left) and 1,000 members (right)

For very small committees, one often finds that the committee is dominated by one or two powerful individuals, and in a sense the committee is not really functioning as a mixture of individual members. See for example Figure 20, left, where the committee's behavior is essentially that of one member (since the $\Diamond$'s fall on or very close to that one member's learning trace). For very large committee sizes, members start duplicating one another and one does not, for a committee of size n, actually have anywhere near n distinguishable hypotheses represented. See for example Figure 21, right, which is the n = 1,000 case. But note that there are far fewer than 1,000 individual traces. In a sense, a form of weighting is occurring, where the weight given to each distinguishable hypothesis is based on the number of members voting in that manner. This may in fact be a worthwhile approach to use, but this experiment does not indicate that such is the case, since the larger committee used more training examples but yet performed the same as or worse than a smaller committee. We will comment further on this idea of weighting the votes later.

6.3 Separating the Effects of Active Learning and Learning by Committee

Preliminary experiments had shown that ALC can, as compared to supervised learning with a single learner, result in learning methods that use far fewer examples and take less time, but still achieve almost the same accuracy [Liere 1997]. The purpose of this experiment was to determine which aspect of ALC accounts for most of the performance improvement. Is meticulous sampling necessary, or is it the committee itself that gives most if not all of the improvement? In other words, as long as we use a committee, does it matter *which* X% of the examples we use for training? This is of interest since active learning expends some effort in determining which examples to actually use for learning. Can we get similar accuracy results using similar numbers of examples chosen randomly along with a committee of *supervised* learners? Is the reduction in the number of examples used by ALC due mainly to the active learning component or to the use of the committee for learning and prediction?

Many learning methods include using a committee to select examples according to some measure of their "usefulness" [Freund et al. 1992, Cohn et al.

1994, Lewis and Gale 1994, Quinlan 1996]. On the other hand, evidence exists in the literature that the committee itself is a good idea. There is for example a quite extensive literature on neural network committee machines [Takiyama 1978, Schwartze and Hertz 1993, O'Kane and Winther 1994] and on bagging [Breiman 1996a] and boosting [Schapire 1990]. There are several reasons why committees are thought to be a good idea [Hansen and Salamon 1990, Dietterich 1997]. Having a committee of learners allows each member to focus on learning different portions of the solution space. Therefore members will also make errors on different portions of the input space, thereby on the average having their errors "cancelled out" by other members that are more adept at classifying that particular example type. Also, using a committee allows one to learn hypotheses that are more complex than the hypothesis space of any one member. For example, a committee of linear threshold learners can learn concepts that are more complex than those that can be learned by a single linear threshold learner.

Having the members of a committee make uncorrelated errors can result in a committee actually outperforming its most accurate member. This is certainly not intuitively obvious. Consider the results given in Figure 22, which show the learning traces for individual members (lines) and for the committee (◊) for the accuracy and $F_{1.0}$ performance measures. These results are from an experiment using active learning with a committee size of 7, with winnow as the learning algorithm and prediction being by majority vote. Observe that there are places where the committee outperforms every one of its members. Examine the traces at "number of labeled training examples used" = 140, 180, and 190. The ◊ at each of those locations is above *all* of the individual learning traces.

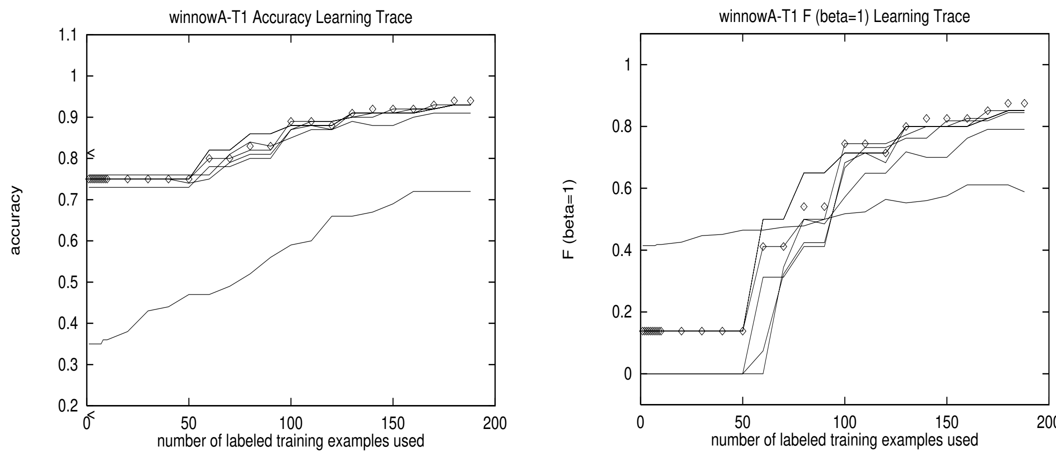

Figure 22. Member and Committee Performance: Accuracy (left) and $F_{1.0}$ (right) [lines = individual members; ◊ = committee]

An example of how this is possible is given in Table 10. This committee is composed of 3 learners, and prediction is by majority vote. Note that each of the members gets 3 out of 4 examples correct, and so each member has an accuracy of 0.75. Yet the committee predicts correctly on all 4 examples, and so has an accuracy of 1.0.

TABLE 10. Individual versus Committee Accuracy

test example	*actual*	- - - - - - - - - - - - - *predicted label* - - - - - - - - - - - - -			
number	*label*	*member #1*	*member #2*	*member #3*	*committee*
1	NO	NO	NO	NO	NO
2	NO	NO	NO	YES	NO
3	NO	NO	YES	NO	NO
4	YES	NO	YES	YES	YES

To determine whether it is the meticulous sampling or the committee learning and predicting itself that gives most of the improvement, we performed an experiment that analyzes the improvement obtained using ALC. We broke apart the ALC method, in an effort to determine if the improvement is mainly due to active learning, mainly due to the use of committees in prediction, or due to an interaction of the two mechanisms.

For this experiment, we used winnow as the learning algorithm. For all active learning systems, we used the QBC-REP method for deciding which training examples to use. We examined the performance of 4 different learning systems. We not only compared them to see which system is "the best", but we also looked into why. In order to meet this second goal, we designed a factorial experiment [Snedecor and Cochran 1989]. That is, we constructed one experiment in which each of the systems does/does not have particular features. This experiment was conducted using the titles of newspaper articles from the Reuters-22173 corpus. The systems are:

{1} *active-majority*, the learner is a committee of 7 winnows which uses QBC-REP to determine which labels to obtain from the teacher. Prediction is made by that same committee, using majority vote.

{2} *passive-majority*, the learner is a committee of 7 winnows which passively accepts all labels from the teacher. Prediction is made by that same committee, using majority vote.

{3} *active-single*, the learner is a committee of 7 winnows which uses QBC-REP to determine which labels to obtain from the teacher. However, in the prediction phase, only a single member of the committee is used. A committee member chosen at random makes all of the predictions in a particular trial.

{4} *passive-single*, the learner is a single winnow which passively accepts all labels from the teacher. Prediction is by that same winnow. This can be thought of as the "baseline system" – a single supervised learner and predictor.

Figure 23 shows elapsed processor time as a function of the number of training examples used, for each of the 4 systems. Recall that we add a box around the cluster of dots for each system and label the box with the system number. Since passive-majority {2} and passive-single {4} are supervised learners, their boxes collapse into vertical lines. Observe that, for each system, the dots form very tight clusters. This dramatically illustrates that, in this experiment, the differences among the systems in terms of both the number of labeled examples used and the elapsed execution time were quite large. Figure 23 also shows that the variation in behavior within each system, for both the number of training examples used and elapsed processor time, was quite small.

ANOVA indicates that the system one uses has a significant effect on both the number of training examples used ($p < 0.0001$) and on the elapsed execution time ($p < 0.0001$).

One usually does not have as many labeled examples as one wants. In those situations, active learning (active-majority {1} and active-single {3}) is beneficial. (There is also the question of whether or not the active learning systems are losing any accuracy. This will be discussed in Figure 24). From Figure 23, it would appear that the 2 active learning systems are very similar, in that they have similar average values both of number of training examples used and elapsed processor time. How would one choose between these 2 systems?

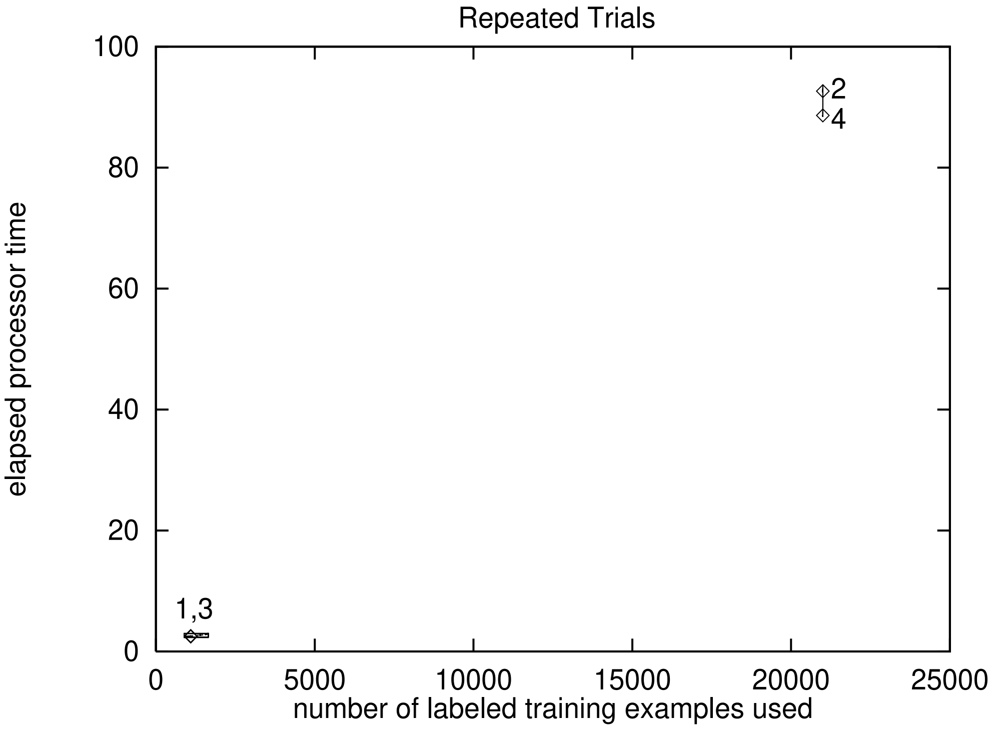

Figure 23. Elapsed Time versus Number of Training Examples Used [1 = active-majority, 2 = passive-majority, 3 = active-single, 4 = passive-single]

Figure 24 shows the average accuracy for each of the 4 systems as a function of the number of training examples used. This is a learning trace, showing how accuracy varies for each system as it learns. We can see that the systems employing active learning use many fewer examples than those using supervised learning, which is consistent with the results in Figure 23. However, Figure 24 also shows that the 4 systems end up with very similar final accuracies, while the path that each takes to get there is different. The fact that the systems are all about the same in terms of final accuracy justifies our having looked at other system characteristics (such as number of training examples used and elapsed processor time) as metrics on which to base our comparison of the systems. As regards the previously posed question as to whether active-majority {1} or active-single {3} is

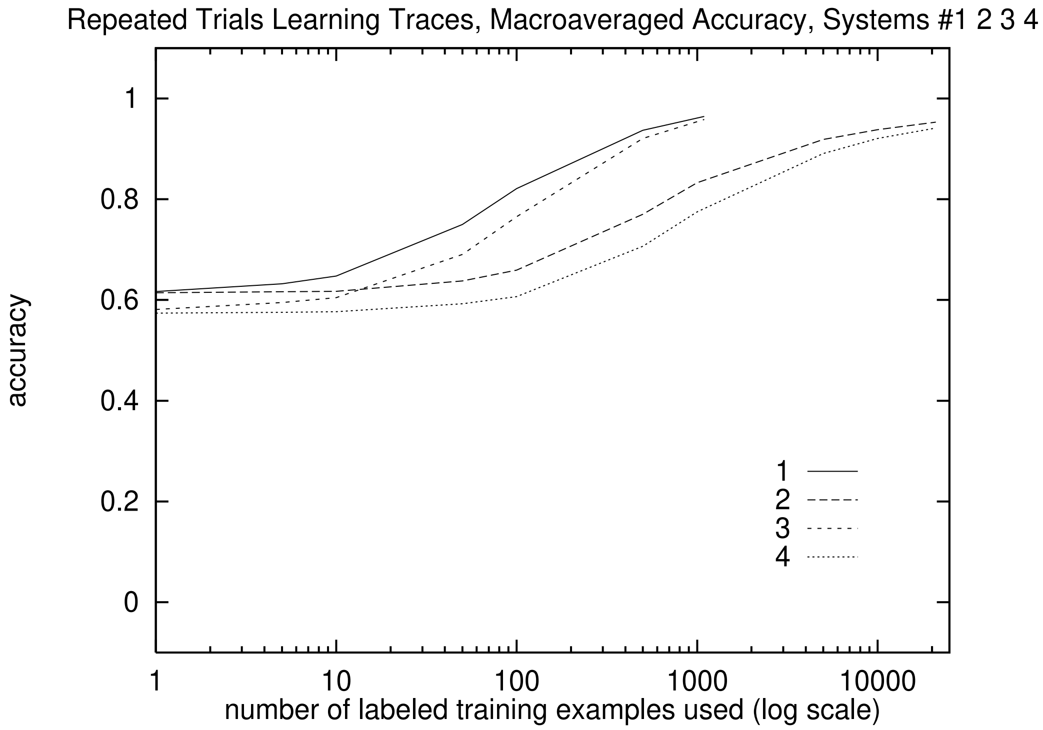

Figure 24. Average Accuracy by System [1 = active-majority, 2 = passive-majority, 3 = active-single, 4 = passive-single]

better, one can see in Figure 24 that active-majority {1} is, during the learning process, more accurate than active-single {3}, with convergence to a common value occurring only towards the end of the learning process. This difference is a consideration in situations where one has a very limited number of training examples available or in situations where the user can at any time ask the system to stop learning and to use what it now knows. Figure 24 indicates that active-majority {1} would be the better system in such situations, since its accuracy is, on the average, always greater than that of the other systems.

Finally, it is appropriate to examine final accuracy in detail, since one of our goals is to develop systems that reduce the number of training examples used without reducing accuracy. The above arguments as to which system is better in various situations would certainly need to be modified if the systems' final accuracies were quite different. ANOVA indicates that the system used has a significant effect on final accuracy ($p < 0.0001$). Table 11 recaps the accuracies of the 4 systems.

TABLE 11. Final Accuracies

	- - - *accuracy* - - -	
system	*mean*	*std dev*
active-majority {1}	0.965	0.0172
passive-majority {2}	0.953	0.0240
active-single {3}	0.958	0.0179
passive-single {4}	0.941	0.0256

We can see that the differences in accuracy are relatively small. That is, all 4 of the systems gave similar levels of accuracy, even though ANOVA does indicate that there is in fact a significant difference in accuracy among the 4 systems. Active-majority {1} has the highest accuracy.

We should also look at $F_{1.0}$, to see how the 4 systems do in precision and recall. Figure 25 shows the average $F_{1.0}$ for each of the 4 systems. It is clear that the best performer when using the $F_{1.0}$ metric is still the active-majority {1} system. ANOVA indicates that the system used has a significant effect on final $F_{1.0}$ ($p < 0.0001$). Table 12 recaps the $F_{1.0}$ values for the 4 systems.

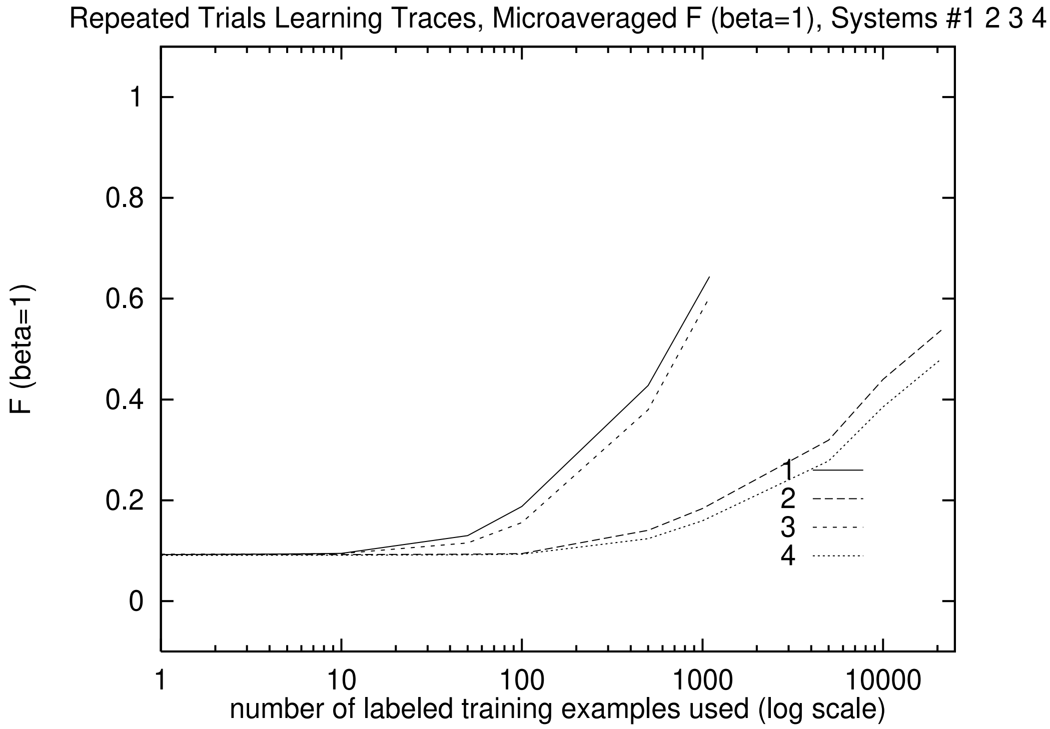

Figure 25. Average $F_{1.0}$ by System [1 = active-majority, 2 = passive-majority, 3 = active-single, 4 = passive-single]

TABLE 12. Final $F_{1.0}$

system	$F_{1.0}$
active-majority {1}	0.644
passive-majority {2}	0.537
active-single {3}	0.603
passive-single {4}	0.481

One can conclude that, of the four systems tested, active learning with committees (active-majority {1}) is the best approach. It achieves accuracies and $F_{1.0}$ values that are the same as or slightly better than those obtained by the other systems, but uses only 5.2% as many training examples as the supervised learners. It also requires less execution time than either the single supervised learner or the committee of supervised learners. Because it has the best average accuracy and average $F_{1.0}$ as learning progresses, active-majority {1} is also the best one for applications in which learning can be halted (and prediction commence) after a certain period of elapsed time, such as when interactive processing is occurring with a human being.

Active-majority {1} performed better than any of the other 3 systems, and it uses both active learning and committee prediction. As regards our original question, as to whether the reduction in the number of examples used by ALC without loss in accuracy is due mainly to the active learning component or to the use of a committee for learning and predicting. Both active-single {3} and passive-majority {2} have better accuracies than passive-single {4}. There is a synergistic effect due to these two components which resulted in the active-majority system outperforming both active-single and passive-majority in accuracy and $F_{1.0}$.

TABLE 13. Features in Each System

		active learning	
		yes	no
prediction by committee	yes	{1}	{2}
	no	{3}	{4}

Table 13 gives a summary of which features each of the 4 systems does/does not have. Comparing whether or not the system used active learning, one can see from Figures 24 and 25 that the systems that are not using active learning (systems 2 and 4) have not achieved the same accuracy or $F_{1.0}$ by the time they reached the

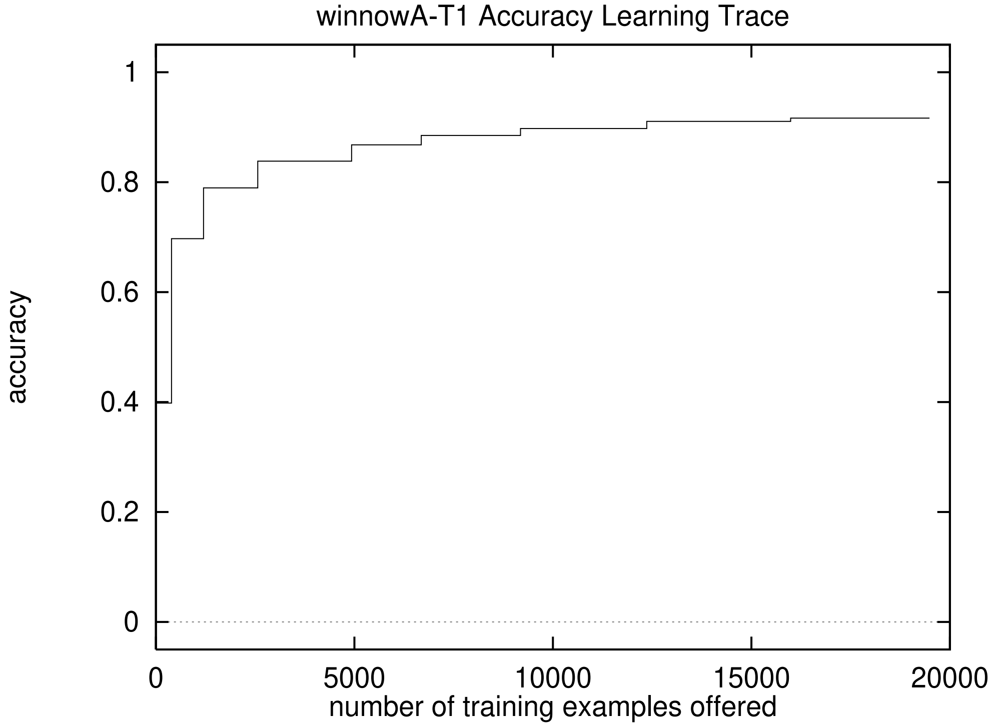

Figure 26. Accuracy versus Training Examples Offered (for one trial)

number of labeled training examples used by the active learning systems. Accuracy at this point for both {2} and {4} appears to be about 75 - 80%, with both methods eventually reaching 94 - 95%. Progress in $F_{1.0}$ is even worse at this point – $F_{1.0}$ for {2} and {4} is at about 15 - 20%, with most of the progress towards the final values of 48 - 54% being made after using many more examples.

Given that we have concluded that active learning is beneficial, comparing whether or not the method used a committee for prediction amounts to comparing {1} and {3}. There is definitely less of a difference here than when {1} was compared to {2} and {4}, so we conclude that the committee prediction is also helpful, but does not contribute as much benefit as is obtained from active learning.

We present here a brief intuitive argument as to why the active-majority method works so well. Please see [Freund et al. 1992] and [Freund et al. 1997] for a more detailed discussion of this aspect of active learning, as it relates to QBC. Figure 26 shows accuracy as a function of the number of training examples *offered* to the learner (instead of the number of training used by the learner). Accuracy is calculated at each hundredth example used (to smooth out the effects of noise and to make the steps more visible). Initially, assuming that the hypotheses in the committee are sufficiently diverse, two randomly chosen hypotheses disagree on an example with a significantly high probability. Hence labels are requested for a significant fraction (about ½) of the examples. As the learning progresses, each hypothesis approaches the optimal target hypothesis, and hence the diversity between the different hypotheses decreases. As a result, the informativeness of an example as measured by the probability of disagreement between two randomly chosen hypotheses decreases, and the distance between two successive label requests increases. This effect is demonstrated by the horizontal portions of the steps in Figure 26 becoming longer as learning progresses. In fact, in the noiseless case, the number of examples offered between successive uses of examples must increase exponentially fast [Freund et al. 1992].

6.4 AllMargin

Since ALC is composed of 3 parts (deciding whether or not to see the label, learning, and forming the committee prediction), one can presumably improve ALC by improving any one of those parts. The learning algorithms we are using seem to perform well, in that they are able to handle large numbers of attributes (necessitated by our performing minimal preprocessing). Winnow has also been shown to be especially efficient in terms of the time needed for training when one has irrelevant and noisy attributes present [Littlestone 1988, Littlestone 1991]. The prediction method we use almost exclusively is majority vote. We examined several others, but none of them gave as good results. Prediction is, for our knowledge base structure, quite fast, so it does not seem reasonable to spend considerable amounts of effort having committee members learn individually and then to not use the predictions of all individuals to form the prediction of the committee. During our research, our initial efforts at improving ALC were therefore aimed at new methods for deciding whether or not to see the label. Being clever/more clever in this area could reap one huge benefits, since doing a better job of deciding which training examples are more useful would mean that one could use even fewer examples *and* also take less execution time. We spent a lot of time developing many many different methods for deciding on whether or not to see the label. Sadly, most of them did not perform very well. In the previous experiment, we used QBC-REP, which is our version of the QBC method, adapted to work using a finite committee to represent the hypotheses. Recall that QBC-REP is a method developed specifically for domains with very large version spaces and/or which have class or attribute noise. The other method that we developed which gives good performance is AllMargin, which we examine in more detail in this experiment.

This experiment uses the Reuters-22173 corpus, titles only. The 3 systems examined are:

{1} AL-WI

7 member committee, AllMargin for deciding when to see labels, winnow learners, prediction by majority vote

{2} SL-WI

7 member committee, supervised learning, winnow learners, prediction by majority vote

{3} SL-NB

supervised learning using naive Bayes

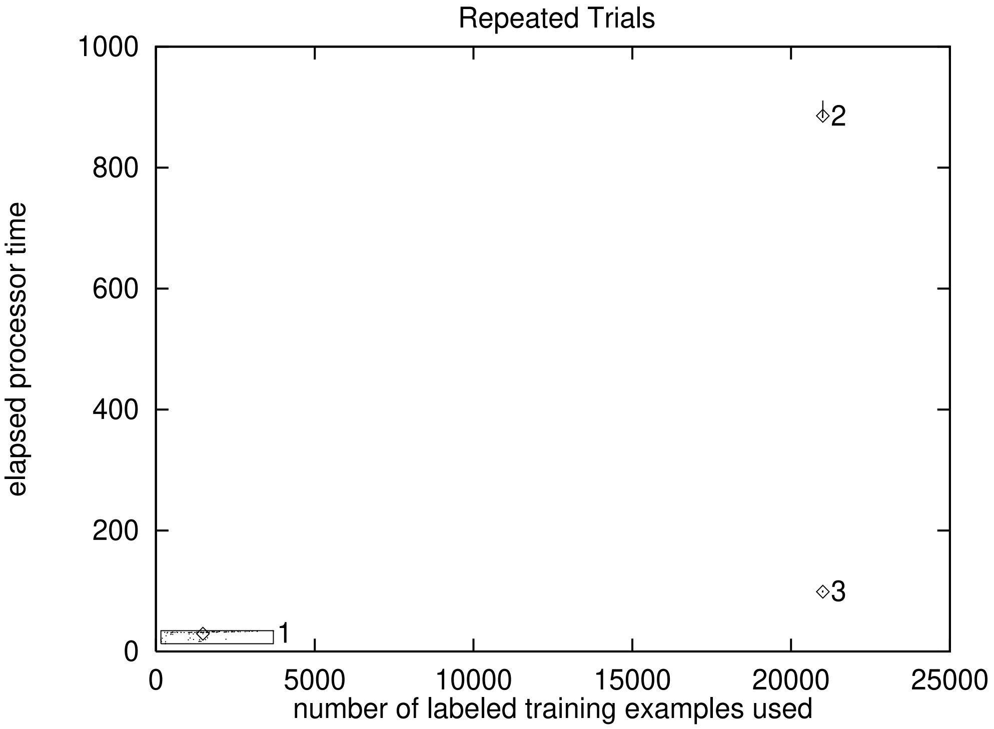

Figure 27. Elapsed Time versus Number of Training Examples Used [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 7 members, winnow, majority voting; 3 = supervised learning, naive Bayes]

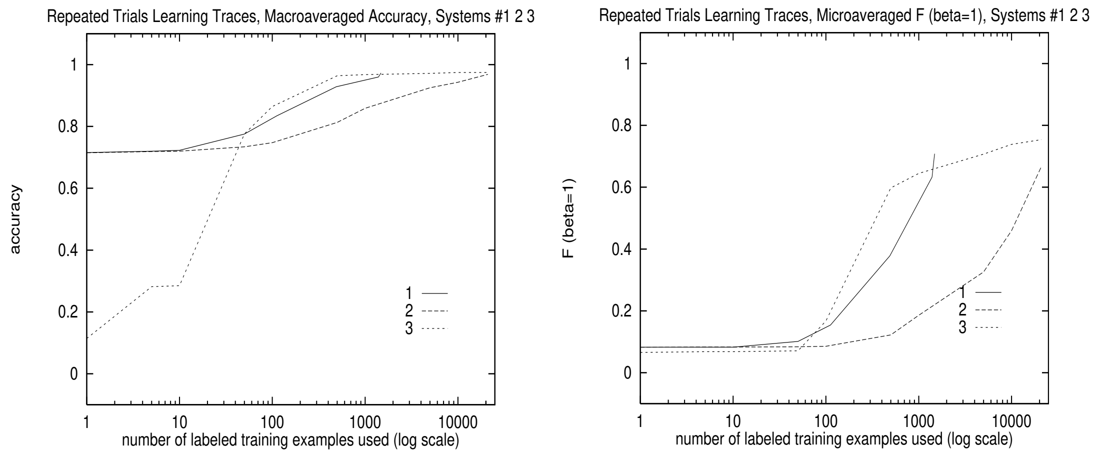

Figure 28. Learning Traces: Accuracy (left) and $F_{1.0}$ (right) [1,2,3=see above]

TABLE 14. Final Performance Measure Values

system	*accuracy*	$F_{1.0}$
AL-WI {1}	0.973	0.708
SL-WI {2}	0.969	0.667
SL-NB {3}	0.974	0.754

Examining elapsed processor time versus number of labeled training examples used, we find (see Figure 27) that AL-WI {1} takes less time than either supervised method, and uses about 7.05% as many examples as the supervised learners. Note that SL-WI {2} is especially slow, due to its consisting of a committee of 7 supervised learners (and so all 7 use each training example).

In Figure 28, we see the learning traces for accuracy and $F_{1.0}$. Table 14 gives the final values for accuracy and $F_{1.0}$. Once again, the systems tested seem to obtain similar values of final accuracy. This in fact is quite typical, due in large part to there being so few positive examples in each category (see Table 4 on page 149). The final values of $F_{1.0}$ are interesting. AL-WI {1} is doing better than the committee of winnow supervised learners, even though it is using fewer training examples and running in less time. However, note that supervised naive Bayes does very well in terms of its $F_{1.0}$ performance. This is troublesome to people who like a nice predictable world tied up with a ribbon and a bow. Naive Bayes by all rights should not do well at all, as its assumptions do not hold for natural language, and its approach is (in spite of the seemingly ugly math) so simplistic that it "does not deserve to win". The consistently good showing made by naive Bayes is what prompted us (in a later experiment) to develop a committee-based naive Bayes algorithm. However, in this experiment, while it is true that naive Bayes has the best $F_{1.0}$ measure, it is also a supervised learning algorithm and so is using all available training examples. Also, its run time (ref. Figure 27) is more than 3 times greater than that of AL-WI {1} (average elapsed processor time values: AL-WI {1} = 29.5, SL-NB {3} = 98.8).

How does AllMargin compare to QBC-REP? Table 15 summarizes the results of the ALC systems in this experiment and the one immediately previous. Recall that both are using the same committee size (7), the same prediction method (majority vote), and the same learning algorithm (winnow). And both experiments were run on the same corpus – Reuters-22173 titles only.

TABLE 15. AllMargin vs. QBC-REP Average Performance Measure Values

system	*time*	*% tng ex used*	*accuracy*	$F_{1.0}$
AllMargin	29.6	7.05	0.973	0.708
QBC-REP	2.6	5.21	0.965	0.644

As is often the case, the answer as to "which is better?" is ... it depends. AllMargin is the better method in terms of $F_{1.0}$ and (slightly) accuracy, but it is on the average using more training examples and taking more time. The additional time required is due to the fact that AllMargin requires a prediction from *all* committee members on each candidate training example, whereas QBC-REP only requires the prediction of 2 committee members.

6.5 Comparison of Winnow and Perceptron

So far, we have used winnow in our ALC system experiments. The perceptron algorithm is very similar to winnow conceptually, and it differs in actual implementation only in the operator used for updating (multiplication for winnow, addition for perceptron) and the fact that in most implementations of winnow (ours included) the threshold θ is fixed, whereas in most implementations of the perceptron (ours also included), θ is adjusted during learning (along with the weights w_i). There are, nevertheless, many papers in the literature, both theoretical and applications-oriented, that find significant differences in the behaviors of these 2 algorithms. Experiments in text classification have found that sometimes additive updating is better and at other times multiplicative updating is better in linear classifier algorithms – it depends on corpus, document representation, and topic [Lewis et al. 1996]. In a series of tests that examined many different variations on winnow and perceptron algorithms for supervised text categorization, winnow appeared to in general be the better algorithm [Dagan et al. 1997]. There are many other papers examining winnow and/or the perceptron in various domains. For example, there are winnow-based applications for calendar scheduling [Blum 1995], natural language disambiguation [Roth 1998], context-sensitive text categorization [Cohen and Singer 1996], online investment portfolio management [Helmbold et al. 1998], and spelling correction [Golding and Roth 1999] to name only a few. We felt that it would be interesting to compare winnow and perceptron in this domain using ALC systems.

There has been a great deal of theoretical work done in analyzing the winnow and perceptron algorithms and how many mistakes they make. Recall that both of these algorithms are mistake-driven, so one is interested in the number of mistakes made since that is an indication of how much time will be spent learning. In [Kivinen and Warmuth 1995], a comparison of winnow and perceptron mistake bounds is derived for tuned learning algorithm performance on k-literal disjunctions. These are boolean functions of the form $x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k}$. If n = the total number of attributes and k = the number of attributes that are relevant, then the perceptron algorithm will make a number of mistakes that is $O(kn)$, whereas winnow will make a number of mistakes that is $O(k\ log(n))$. We do not claim that the target concept in text categorization is a k-literal disjunctive function, but this is a simple subclass of boolean linear threshold functions and is a reasonable approximation to the approach used by some text categorization systems to categorize text. Their results indicate that, when learning a k-literal disjunction, winnow can, in domains where there is a large number of irrelevant attributes (k is small), make a number of mistakes only logarithmic in the number of attributes. There is also evidence that winnow performs better in terms of accuracy when there are large numbers of attributes with many of them irrelevant [Littlestone 1995]. These results are of special interest to us, since due to the simple nature of our tokenizing method and the fact that we perform minimal preprocessing, we have large numbers of attributes, and we expect that many of them will be irrelevant.

The purpose of this experiment is to compare the winnow and perceptron algorithms and to determine which is better as the learning algorithm in ALC when categorizing text. This set of experiments also brought us squarely up against the need to pick *a* specific contingency table based performance measure that is "most important" in order to make a firm decision as regards which learning algorithm is better. This series of experiments was conducted using the Reuters-22173 corpus, titles only. All systems used committees of 7 members, and all systems predicted by majority vote. The ALC systems used AllMargin to decide when to see the label.

We had initially "preliminarily concluded" that the perceptron was the better learner, because it ran significantly faster than winnow, used fewer training examples, and achieved essentially the same accuracy [Liere and Tadepalli 1998a, Liere and Tadepalli 1998b]. These statements remain true in our current experiments. However, we investigated further and found that perceptron's behavior when evaluated using other performance measures, in particular $F_{1.0}$, is often worse than for winnow. Essentially what was often happening was that the perceptron was learning to "just predict NO". Since there are very few positive examples in many of the categories, it is possible to get a quite good accuracy rating in this manner. However, using such a system for information retrieval or text categorization would not be acceptable to most users. (Recall that a system always predicting NO has a 0.0 score for both precision and recall).

Below we give the results of one of a great many different series of trials. Having erred once, we did not want to compound the mistake. We tried a great many different settings of δ, and even tried some other weight initialization procedures that we thought might be more applicable to the perceptron. We first discuss the results of a typical series of trials, and then discuss the implications of these results and possible reasons for them.

The typical series of trials for which we will present results consisted of 8 systems. Several are (it turns out) similar in behavior, so we will not need to distinguish between all 8 of them. The systems are:

{1} SL-WI
supervised learning using winnows

{2} AL-WI
ALC using winnows

{3} SL-PI
supervised learning using perceptrons

{4} -{8} AL-PI
ALC using perceptrons; different systems used different settings of δ, in an effort to give perceptron every chance

Figure 29 shows elapsed processor time versus number of training examples used. The supervised learning systems, both SL-WI {1} and SL-PI {3}, take relatively large amounts of time (since they are using all 21,000 examples) – an average time of ≈886. The AL-WI {2} system has an average time of 29.6 and uses an average of 7.05% of the training examples. All of the perceptron ALC systems ({4}-{8}) are clustered tightly together with average times of 20 - 23 and with average number of training examples used of 1 - 2%. So at this point perceptron is

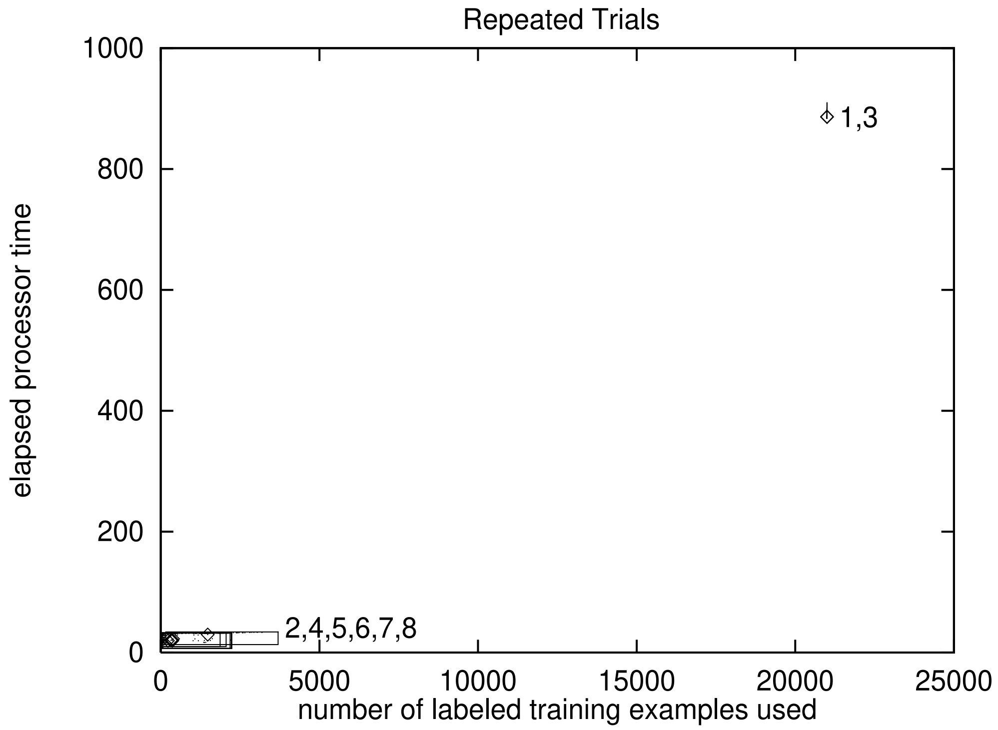

Figure 29. Elapsed Time versus Number of Training Examples Used [1 = supervised learning, 7 members, winnow, majority voting; 2 = ALC, 7 members, AllMargin, winnow, majority voting; 3 = supervised learning, 7 members, perceptron, majority voting; 4-8 = ALC, 7 members, AllMargin, perceptron, majority voting, various settings of δ]

clearly doing better than the other systems (having examined only the time and examples used performance measures); it is faster and using very few of the available training examples.

Figure 30 shows the learning traces for these 8 systems for the accuracy performance measure. This if anything makes one even more certain that the perceptron is the better learning algorithm, since it appears that the final accuracies are all about the same. In fact, they are all in the range 0.961 - 0.973. However,

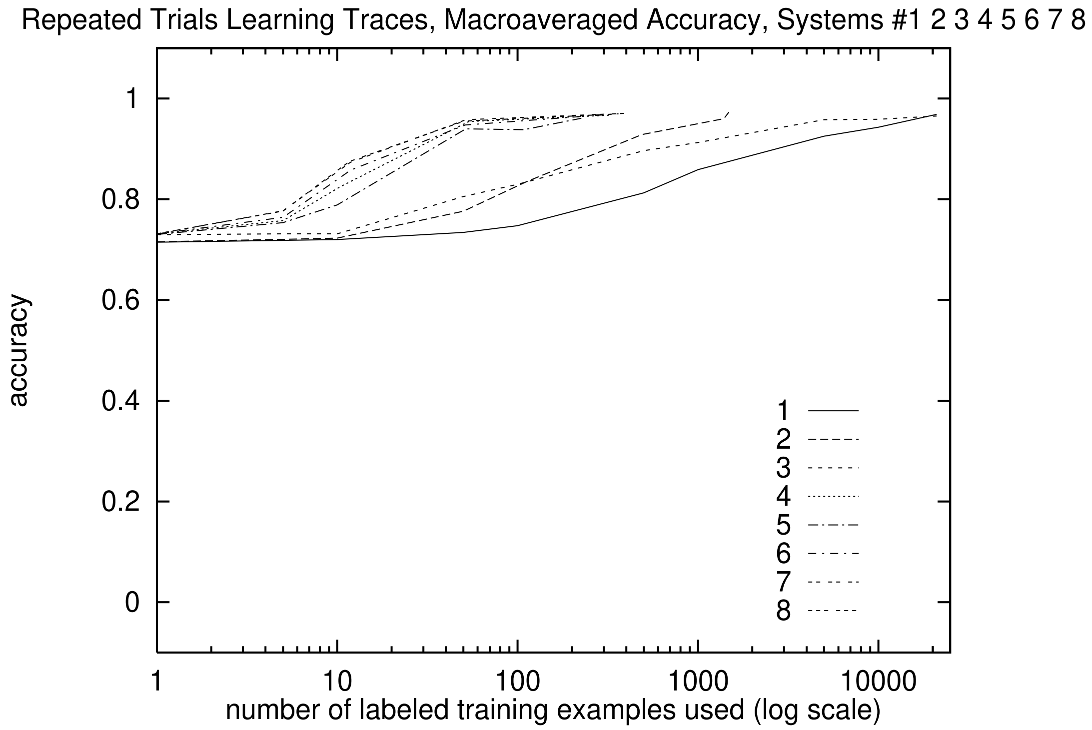

Figure 30. Accuracy Learning Traces [1 = supervised learning, 7 members, winnow, majority voting; 2 = ALC, 7 members, AllMargin, winnow, majority voting; 3 = supervised learning, 7 members, perceptron, majority voting; 4-8 = ALC, 7 members, AllMargin, perceptron, majority voting, various settings of δ]

examination of the $F_{1.0}$ learning trace, shown in Figure 31, changes our outlook on the results. Clearly system AL-WI {2} is the better active learning system. And it appears that SL-WI {1} is also better than SL-PI {3}. This is confirmed by Table 16, which gives the final values for accuracy and $F_{1.0}$ for each system for the series of trials.

It appears from this experiment (and several others that we performed) that winnow is a better learning algorithm for our purposes than perceptron. Some questions come to mind. Is this result a surprise? Do we have a strong desire that a specific algorithm be the best?

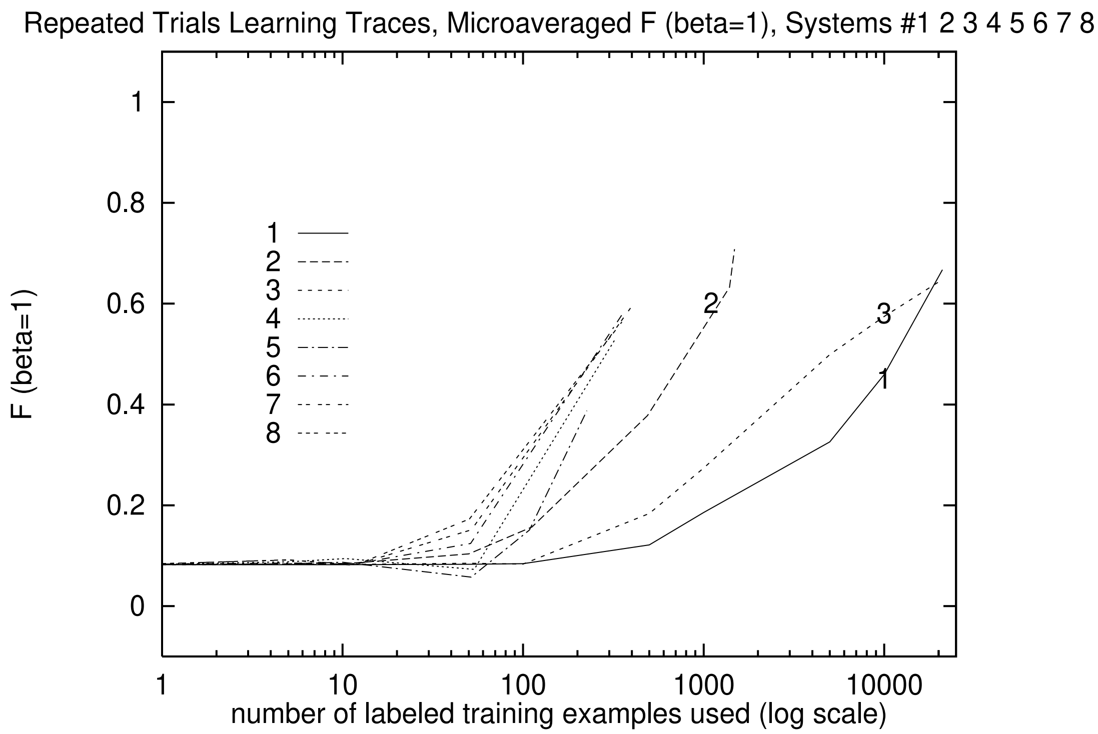

Figure 31. $F_{1.0}$ Learning Traces [1 = supervised learning, 7 members, winnow, majority voting; 2 = ALC, 7 members, AllMargin, winnow, majority voting; 3 = supervised learning, 7 members, perceptron, majority voting; 4-8 = ALC, 7 members, AllMargin, perceptron, majority voting, various settings of δ]

TABLE 16. Final Performance Measure Values

system	*accuracy*	$F_{1.0}$
SL-WI {1}	0.969	0.667
AL-WI {2}	0.973	0.708
SL-PI {3}	0.965	0.648
best of AL-PI {4}-{8}	0.970	0.591

As discussed above, the "only difference" computationally between the two algorithms is that one (winnow) uses floating point multiplication to perform the updates, the other uses floating point addition. The common notion is that floating point multiplication is a much more expensive operation, so perhaps we would like perceptron ("addition") to be better. While multiplication used to be quite a bit more expensive than addition, modern cpu architectures have made the penalty for multiplication as compared to addition quite a bit less. And besides, there are so many other things going on, whether the update operation is multiplication or addition probably makes little difference overall. This in fact is verified by our timing runs.

Recall that perceptron, due to the additive updating of its weights, can learn any linearly separable pattern. This would seem to be a very major advantage. The version of winnow that we are using is unable to learn any pattern that requires learning a negative weight. This means that all of the axis intercepts of the learned hyperplane must be ≥ 0 for winnow. This would seem to be a major disadvantage of the winnow algorithm. So why did it win?

We did an experiment to test the hypothesis that the capability of learning negative weights is not actually needed. We ran several trials of the following test. A perceptron learner (which *can* learn negative weights) was initialized and started learning with a large value of δ (recall the discussion on "separation" of the YES and NO clouds). It was allowed to make as many passes as it wanted through the data, using all examples during each epoch. Computational efficiency was not a consideration. Basically the learner adjusted its learning rate in a greedy hill-climbing manner as learning progressed, in an effort to obtain a very good separating hyperplane for the given data. Learning did not cease until one of the following conditions occurred: [1]specified performance measure goals were reached, [2]limit on number of epochs was reached, [3]learning ceased, or [4]there was not sufficient floating point precision to continue. The Reuters-22173 titles only corpus was used. Once this very good (albeit time-consuming) solution was found, the weights were examined as regards their sign and magnitude. Then all weights whose value were < 0 were set to 0, and the modified learner was retested to see how the loss of the negative weights affected performance. We repeated this test for a large number of trials. Here we present the results for a typical trial. This particular trial happens to be for the category "topic=acq". Its final accuracy was 0.984, its final $F_{1.0}$ was 0.93 ... certainly a very good solution in terms of contingency table based performance measures. Recall (ref. Table 4 on page 149) that the categories in the Reuters-22173 corpus are typically inconsistent and so can not be perfectly learned by a single linear threshold learner. The distribution of weights learned for this very good (especially in terms of $F_{1.0}$) solution is shown in Figure 32.

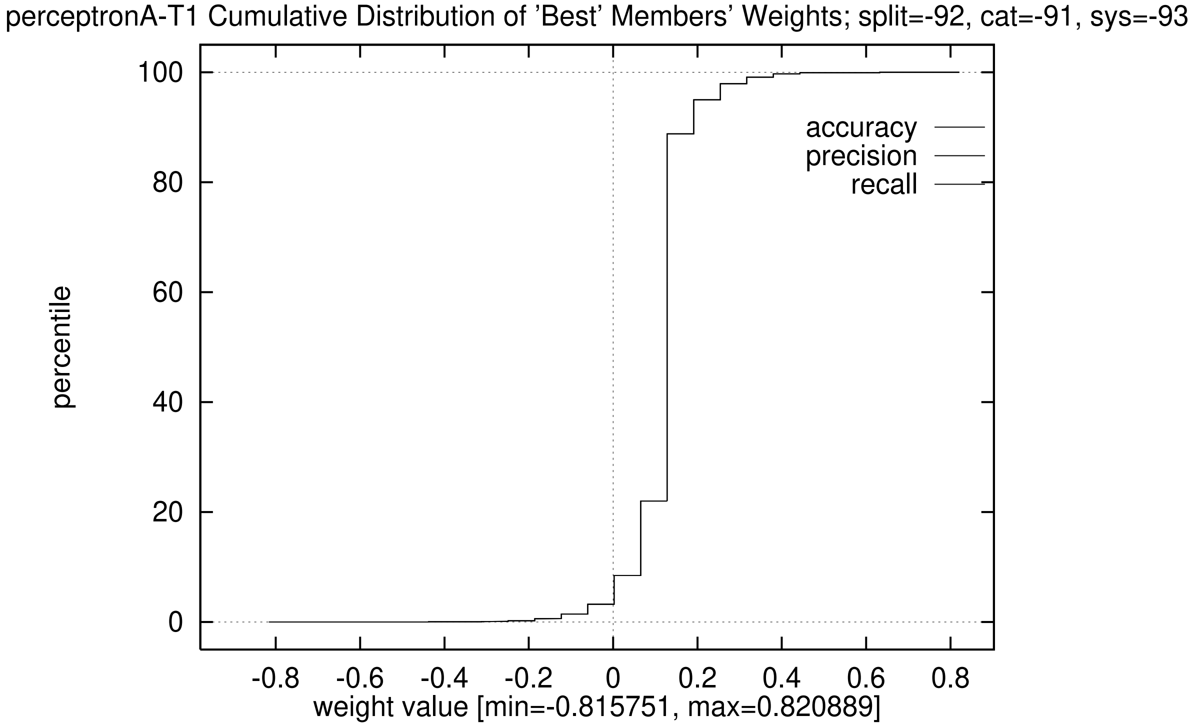

Figure 32. Cumulative Distribution of Weights

As can been seen from Figure 32, the vast majority of the weights are ≥ 0. And the ones that are < 0 are of small magnitude. Over a large number of such trials, we found that typically less than 3% of the weights were negative, and most of those were only slightly negative – normally with magnitudes of less than 5% of the positive weight range.

Interestingly enough, setting the negative weights to 0 rarely made any difference whatsoever in the contingency table of the learner. In other words, while negative weights had been learned by the perceptron, negative weights were not needed to represent a solution that was equally good in terms of *all* contingency table based performance measures.

Given that the ability to learn negative weights is not needed, we can explain the comparative performance of winnow and perceptron in our domain by returning to the operator used for updating the weights. Perceptron always updates by a fixed amount, α (ref. Figure 7 on page 81). α is a learning rate that is computed at the beginning of learning and that remains unchanged throughout the learning process. The fact that updating uses the addition operator means that the same *magnitude* of adjustment occurs to all weights that are adjusted, regardless of their size. Therefore a weight that is quite small can, in one adjustment, be made several times larger. For example, consider perceptron $\alpha = 0.5$ and a weight of value 0.01 being updated. $0.01 + 0.5 = 0.51$, so the weight has increased by a factor of 51. But note that a weight that is quite large may, in one adjustment, have hardly any relative change made to its value. Consider a weight of value 100.0. $100.0 + 0.5 = 100.5$, which is ≈ 100.

Compare this to the winnow algorithm, which also has learning rate parameters – it happens to have 2, α and β (ref. Figure 6 on page 78). These parameters also do not change once they have been assigned values, but note that since the updating is multiplicative, the amount of adjustment made to each weight is proportional to its current magnitude. That is, each update of a weight will result in the weight changing by the same *relative* amount, regardless of its value. For example (using winnow $\alpha = 1.5$), updating a weight of value 0.01: $0.01 \times 1.5 = 0.015$. And updating the weight of value 100.0: $100.0 \times 1.5 = 150.0$. In other words, in a sense the updating mechanism used by winnow adapts to the sizes of the various weights, changing small weights by small amounts and large weights by large amounts. This self-adjusting nature of the algorithm seems in our domain to be causing the algorithm to accurately position the hyperplane faster (i.e., in fewer shifts) as learning progresses.

Returning to the 2 cloud discussion, once the hyperplane has been initialized to some location in the document space, it will usually be the case that it is at least in some of its dimensions some distance from a location that will separate the 2 clouds. That is, if one compares the initial weight values to those of an optimal separating hyperplane, one will find that some large weights need to become small and some small weights need to become large. And of course, some weights will not need to change much. In dimensions where large changes in weights are needed, one would like to accomplish that large change in as few weight adjustments as possible (so that fewer training examples are used and also so that less time is required for learning). This is an argument for making the learning rate parameters large. But, since the data in our domain contains noise and contains large numbers of irrelevant attributes, we need to also be concerned about an example E causing a large change in weights that are *already* optimal. In other words, once a weight that is optimal is "improperly" adjusted by E (due to noise in E and/or due to the presence in E of irrelevant attributes), we have to allow for the fact that we may not get any examples that will "undo" the harm done by example E, and so we want the impact of the harm caused by E minimized. This is an argument for making the learning parameters small. As a compromise, one must therefore set the learning algorithm parameters so that a series of (net) like type weight adjustments (promotion or demotion) causes a reasonable amount of change in weight values in dimensions for which the weight value must be changed by a large amount (large weight → small weight or small weight → large weight), but so that the changes made as a result of one example to weights that are already near their optimal value will have minimal impact. Winnow seems best able to perform under this compromise, since it adjusts weights multiplicatively and therefore each weight is adjusted by the same relative amount each update. Needed large changes in weight values (small → large or large → small) can be accomplished by winnow in fewer updates since the size of each update made is relative, so as the weight values change, the adjustment amounts change automatically, and in the correct direction.

6.6 Ranked Output

So far we have discussed a text categorization system that gives the user a YES or NO prediction when presented with a document. Additional information is made available by many text categorization systems in the form of ranked output. A ranking algorithm defines an ordering on the documents according to their degree of similarity to the category of interest [McGill 1979]. Any method that computes some similarity measure can therefore be used to rank documents. For example, many text categorization systems determine their prediction by first computing a probability p that the document is in the category of interest. Therefore the value of p will be in [0,1]. To obtain the prediction of NO or YES, the system then simply compares the computed probability to some threshold parameter h. If $p > h$ then it predicts YES otherwise it predicts NO. Such systems can therefore easily output the documents ordered by decreasing value of p. In fact, many text categorization systems routinely report their predictions in ranked order.

There is a bit of a terminology problem here. Technically, the rank of each document is an integer starting at 0 or 1, and increasing consecutively to indicate decreasing similarity to the category of interest. Therefore the document ranked 1 is (in the opinion of the system) more relevant to the category of interest than, say, the document that is ranked 10. However, in order to compute the rank, the text categorization system first computed some measure of similarity between each document and the category of interest. This computation does not (generally) directly produce the desired series of unique and consecutive integer-valued ranks. Instead, it likely computes a value that is a measure of the similarity between each document and the category of interest. We will call this value on which the ranking is based the rank raw score. The rank raw score is likely real and is likely to be large when the similarity is high and small when the similarity is low. Sorting the documents by the rank raw score, from high to low, allows us to compute the ranks (going from low to high). For example (from [Harman 1996]):

```
030 Q0 ZF08-175-870  0 4238 prise1
030 Q0 ZF08-306-044  1 4223 prise1
030 Q0 ZF09-477-757  2 4207 prise1
030 Q0 ZF08-312-422  3 4194 prise1
030 Q0 ZF08-013-262  4 4189 prise1

... The fourth column is the rank ...
and the fifth column shows the score
(integer or floating point) that
generated the ranking.
```

The fifth column is what we are calling the rank raw score.

Systems that use such a similarity calculation process to perform ranking can therefore also easily report to the user the rank raw score of each document as well as the rank itself. This is useful additional information, as a group of 3 YES documents returned to the user with ranks and rank raw scores of 1=0.9, 2=0.85, 3=0.25 not only tells the user the ranking, but also gives the user an idea of the degree of similarity for each document. In this case, for example, the first 2 documents are almost tied, and the 3rd document is much less similar than the first two. The rank information allows the user to examine the "most likely" documents first. The rank raw score information allows the user to more intelligently determine his/her own cutoff value for where to stop detailed examination of the actual documents.

Winnow and perceptron do not compute a probability in order to form a prediction. They simply determine on which side of the hyperplane the document lies, and that indicates whether the prediction is YES or NO. Since rank and rank raw score values are quite useful to the end user, we investigated ways to compute these in ALC. Our goal was to find some way of computing ranking information that worked reasonably well and that did not require much extra in the way of computations. In other words, one could always perform a second complete set of computations to obtain rank raw scores, but we hoped to be able to find a way to essentially use what information we already had available.

Before we report the results of our experiments, we need to discuss how we will present ranking information. While text corpora do contain labels indicating the categories into which each document belongs, they contain no "list of right answers" for the ranking process, so computation of some kind of "ranking contingency table" is not possible. The method we use for presenting ranking results is a standard method that is virtually identical to the one used in many research comparisons including the TREC (Text REtrieval) Conference evaluations. It is described in detail in [Hersh 1996] and in [Voorhees and Harman 1997]. We will go over this standard method briefly, and then comment on a difference in our use of the method. Basically the standard method converts a ranked list of documents into a graph that one then compares to the ideal case in order to judge how well the ranking is working. The approach used is to first sort the output of the text categorization system into increasing rank order (by a decreasing sort on the rank raw scores) and then to compute precision and recall for successive YES documents (actual label) on the sorted list and to plot those values. Let us examine the use of the method in more detail, and specific to the ideal case since that is the case to which results are compared. The ideal case occurs when the text categorization system's predictions and ranks are 100% accurate. This means that [1]all of the predicted labels are correct and therefore are equal to the actual labels, and [2]all of the YES documents are ranked above all of the NO documents. The output of the text categorization system is a list, in rank order, of: rank, some kind of document identification number, the YES or NO prediction, and perhaps the rank raw score. For the ideal case, we will have all of the YES documents at the top of the list and all of the NO documents following. Presumably in between the lowest-ranked YES document and the highest-ranked NO document is where the threshold value of h discussed above lies. To apply the standard method of plotting precision versus recall using the ranks, one begins at the top of the list and progresses down the list of documents, computing recall and precision after each YES document (actual label) on the list. These computations depend on the following 4 values:

- y = the number of documents encountered so far that are actually YES
- t = the number of documents encountered so far
- Y = the total number of documents that are actually YES
- T = the total number of documents

Recall at each point is computed as $\frac{y}{Y}$. Precision at each point is $\frac{y}{t}$. Say there is a total of 10 YES documents. Then, in the perfect situation described, the first document on the list will have a recall of $\frac{1}{10} = 0.1$ and a precision of $\frac{1}{1} = 1.0$. The second document will have recall = $\frac{2}{10} = 0.2$ and precision = $\frac{2}{2} = 1.0$, etc. until we reach the tenth document on the list, whose recall will be $\frac{10}{10} = 1.0$ and whose precision will be $\frac{10}{10} = 1.0$. Thus the ideal system will have a precision-recall curve that is a horizontal line at precision = 1.0 and extending from a recall of $\frac{1}{Y}$ to 1.0.

The method that we used for plotting ranked data is identical to the above method, except that we plot a point for each document on the list, regardless of its actual label. This was done in an effort to have the detailed plots give us more information about problem areas in the ranking. For the ideal case, therefore, our system will plot the same horizontal line as for the standard method, and then precision will drop to $\frac{Y}{T}$.

Since there are typically lots of small fluctuations in the standard precision versus recall rank plot (and even more so in our modified version), the normal approach is to use recall intervals of 0.1, thereby obtaining a smoother plot. "Intervalizing" the data also involves several rules regarding how the interval boundaries are handled and which value within each interval is chosen for plotting. We will not cover those details here, but they are covered in [Hersh 1996, Voorhees and Harman 1997]. We use the standard 0.1 interval method (i.e., no modifications to the standard method).

We investigated many approaches to ranking. The one that seemed to work best overall uses the following method. To compute the rank raw score, we first compute the signed distance to the document from each of the hyperplanes in the committee (+ for above the hyperplane and − for below the hyperplane). Those values are then averaged and then the magnitude is taken, and this is our rank raw score. Our rank raw score values are therefore real numbers ≥ 0. The value of the predicted label is then used if later the sign is needed.

One might ask why we threw away the sign, only to later (when it is needed) obtain it by looking at the committee predicted value of YES or NO. The answer is: the two signs are not necessarily the same. There is no guarantee that the sign of the computed rank raw score and the committee prediction will agree. Depending on the method used for computing the committee prediction, it is possible for the prediction of the committee to be, for example YES, but the above-described average distance to be negative. Recall that the predictions by the individual members are very granular – YES or NO, based only on where the document is with respect to the hyperplane and not at all on the distance of the example from the hyperplane. So for example, if there were 5 learners in the committee and the distances to an example were -5, -5, +1, +3, +2, then the committee would (assuming majority voting) predict YES since the vote is 3:2, but the average distance computation will give $\frac{-4}{5} = -0.8$. The method we use for ranking assumes that, on the average, the committee prediction is correct and that rank raw score is proportional to the magnitude of the average distance to the document from the individual hyperplanes.

The following experiment was performed on the Reuters-22173 corpus, titles only. We used the following 3 systems:

{1} AL-WI
ALC using AllMargin, committee of 7 winnows, and prediction by majority vote

{2} SL-WI
supervised learning using committee of 7 members, prediction by majority vote

{3} SL-NB
supervised naive Bayes learner

These 3 systems were chosen so that we could see how our ranking method compared for supervised winnow as well as for ALC using winnow. We included the naive Bayes learner because it is also a supervised linear threshold learner and often does surprisingly well. Also, naive Bayes actually predicts using probabilities, so perhaps it will be a better system to compare to. Figure 33 shows representative precision versus recall curves based on ranks for AL-WI {1}. The "∗" symbol represents the location of the operating point of the text categorizer in terms of final precision and recall. Some of the "∗" symbols are not exactly on one

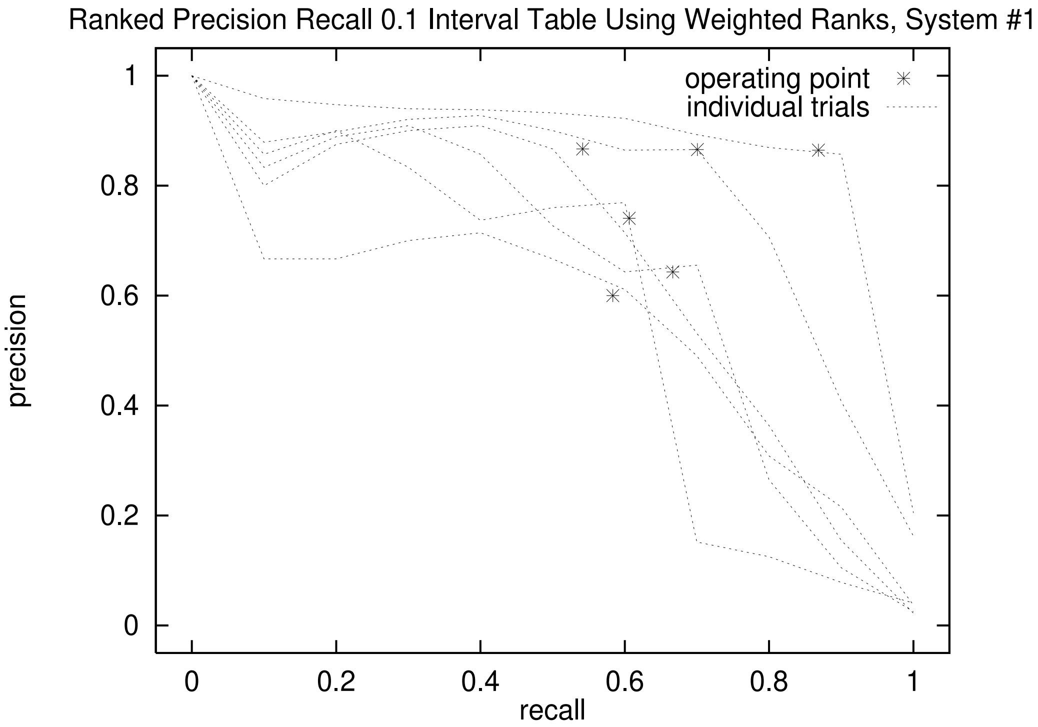

Figure 33. Ranked Precision versus Recall Using 0.1 Interval: AL-WI{1}

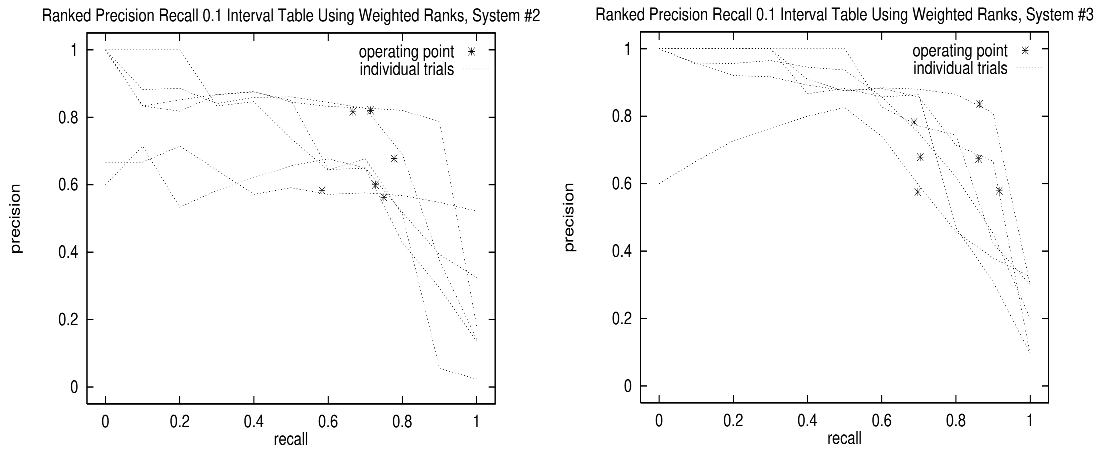

Figure 34. Ranked Precision versus Recall Using 0.1 Interval: SL-WI {2} (left), SL-NB {3} (right)

of the lines for a trial – this is because the process of intervalizing the data causes the plot to not go through each and every precision-recall point. Figure 34 shows the same trials for SL-WI {2} and SL-NB {3}.

Recall that a system that does perfect ranking will have a ranked precision versus recall curve that is a horizontal line at $p = 1$ extending from $r = 0$ to $r = 1$, and then the line will drop to $p \approx 0$. To judge how well the various systems are ranking the documents, we compare to the perfect system as well as comparing amongst the systems. AL-WI {1} does a credible job of ranking. Keep in mind that these ranks are computed "after the fact" using already-available information. AL-WI {1} does a better job than its supervised counterpart SL-WI {2}, since its plots for all categories start at $p = 1$ (for $r = 0$) and generally stay higher than the plots for SL-WI {2}. Apparently its choice of the better examples to learn from has also allowed it to perform better ranking. The supervised naive Bayes learning algorithm, with the exception of the lower curve, seems to do the best job of ranking since its curves are the closest to the ideal of a horizontal line that then drops to ≈0 on the right of the plot. Perhaps this is not too surprising; the naive Bayes algorithm does well in categorizing text and is also in a sense more appropriate for computing ranks because it computes probabilities in order to make its predictions. Recall that winnow does not compute probabilities in order to predict.

6.7 Full Text

As discussed earlier, we performed most of our experiments using Reuters-22173 titles only. This allowed us to run our tests faster, and results were actually better in most cases. Tests ran faster because of there being fewer attributes. We think that performance was better due to the title being relatively free of attribute noise and also being a concise abstract of the document's meaning.

However, we promised to later run our system on full text, to show that our system does handle full text well. That time has come.

First, we will do an experiment on Reuters-22173 full text. This corpus has 48,446 tokens in the dictionary, so the feature vector representing each document contains 48,446 elements versus 16,600 for titles only. We will show that our method scales up in terms of handling this increased number of attributes. While the number of attributes (t) has gone up by a factor of almost 3, the average number of unique tokens per document (t_a) has gone up by a factor of more than 10. We will compare performance on full text to performance on titles only in this same corpus. A very important point to note is that no effort was made to tune the system for full text. We instead used the same δ that we had used for titles only.

We use the same 3 systems as were used earlier in the experiments that investigated the performance of AllMargin. Those 3 systems are:

{1} AL-WI

7 member committee, AllMargin for deciding when to see labels, winnow learners, prediction by majority vote

{2} SL-WI

7 member committee, supervised learning, winnow learners, prediction by majority vote

{3} SL-NB

supervised learning using naive Bayes

Figure 35 shows time versus number of training examples used for the 3 systems. As can be seen, AL-WI {1} uses far fewer examples (8.08%) than either of the supervised learners. While it is not quite as fast as SL-NB {3}, it certainly is much faster than the supervised learning committee SL-WI {2}. Figure 36 shows accuracy and $F_{1.0}$ learning traces for the same 3 systems. It appears that the 3

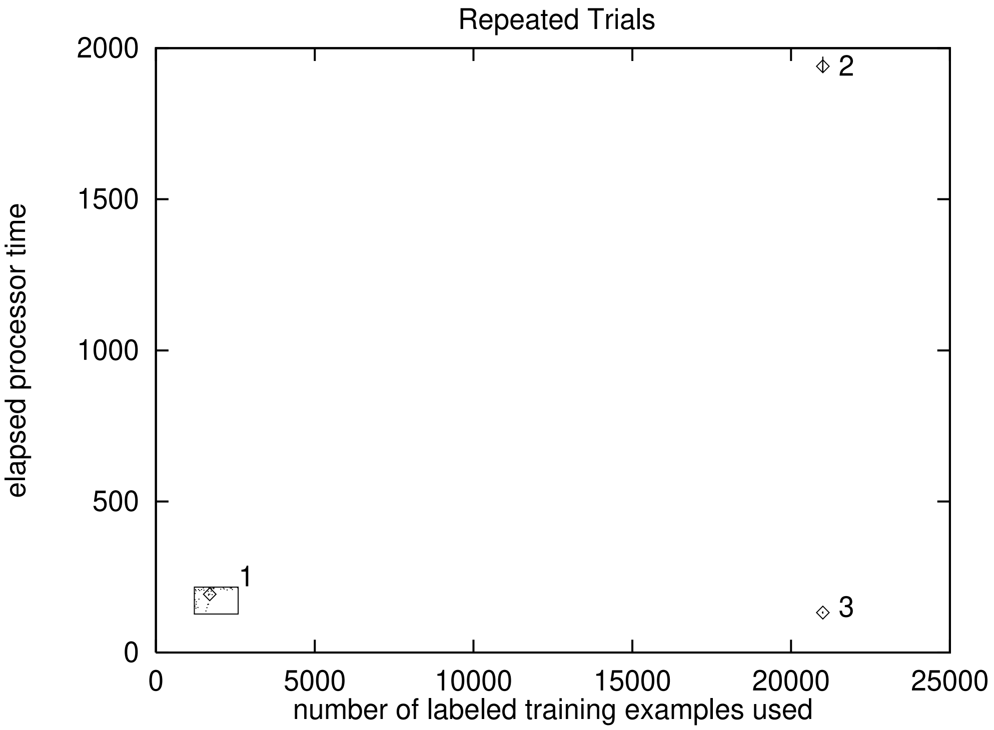

Figure 35. Elapsed Time versus Number of Training Examples Used (Reuters-22173) [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 7 members, winnow, majority voting; 3 = supervised learning, naive Bayes]

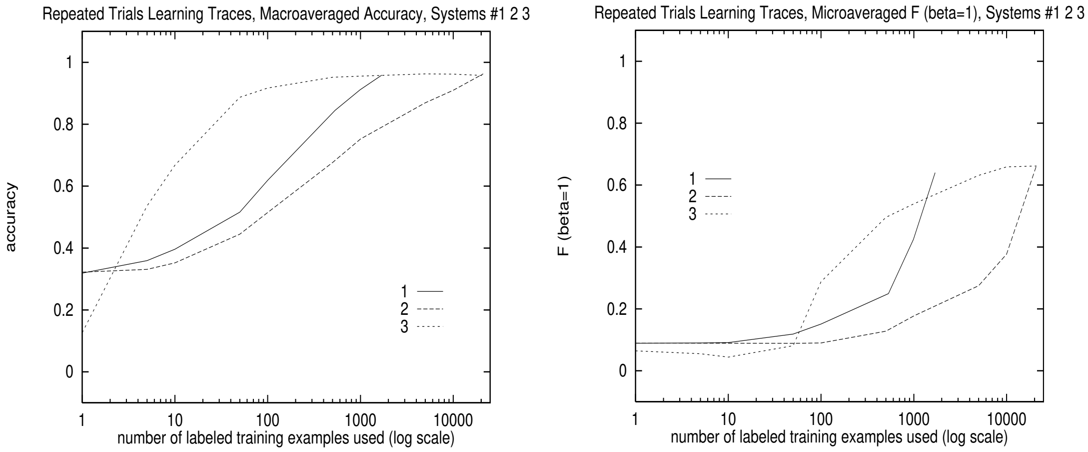

Figure 36. Learning Traces: Accuracy (left) and $F_{1.0}$ (right) for Reuters-22173

TABLE 17. Reuters-22173 Performance: Titles Only ("titles") vs. Untuned Full Text ("u. full")

system	*time*		*% tng ex used*		*accuracy*		$F_{1.0}$	
	titles	*u. full*	*titles*	*u. full*	*titles*	*u. full*	*titles*	*u. full*
AL-WI {1}	29.5	193.0	7.05	8.08	0.973	0.959	0.708	0.640
SL-WI {2}	885.6	1939.8	100	100	0.969	0.963	0.667	0.663
SL-NB {3}	98.8	132.5	100	100	0.974	0.957	0.754	0.662

systems have essentially identical final accuracies and that their final $F_{1.0}$ values are also quite close. Table 17 compares these 3 systems for several performance measures, both for titles only (data from previously presented experiments) and for full text (data from this experiment). In going from titles only to full text, the number of attributes has increased by a factor of almost 3. Recall that earlier when we discussed sample complexity, we obtained the result that the lower bound on the number of examples needed for training in order to obtain a particular accuracy is proportional to the number of attributes. Since in full text we still have essentially the same number of training examples being used as we did for titles only (AL-WI {1} does use ≈1% more), one would expect performance to drop. One might also expect a performance decrease because presumably more care goes into the crafting of the document title than into the document body, and so there will likely be more attribute noise when we process full text.

SL-NB {3} has scaled up very well in time performance. From Equation (10) (on page #124) and since $t \gg t_a$, SL-NB {3} has time complexity $O(dt)$. In this case, d is unchanged so we have $O(t)$. Since t increased by a factor of 3, $O(t)$ certainly upper bounds the actual run times. Accuracy dropped, but only by about 0.02. However, $F_{1.0}$ dropped by almost 0.1. This is not good, since for the text categorization domain $F_{1.0}$ is one of the more important contingency table based performance measures. Other experimenters have found that naive Bayes, when compared to other learning algorithms, has difficulties handling large numbers of attributes [Littlestone 1995, King et al. 1995], especially when many of them are irrelevant [Littlestone 1995]. Naive Bayes also has difficulties with very large sample sizes [Domingos and Pazzani 1997]. Although it performs well in spite of the "naive" assumption, one can not totally ignore the implications of assuming conditional independence of attributes given the class, and in domains where there is a large amount of dependency between many pairs of attributes, performance will degrade [King et al. 1995, Singh and Provan 1995]. It appears that having a very large number of attributes with many of them irrelevant is what has caused the full text case to impact SL-NB {3} so severely in its $F_{1.0}$ performance.

AL-WI {1} has scaled up well, in that it uses only an additional 1% of the available training examples but loses only 0.014 in accuracy. Its time complexity (ref. Equation (8) on page #124) is $O(dt_a)$, which with d unchanged becomes $O(t_a)$. Since t_a increased by a factor of 10, $O(t_a)$ certainly upper bounds the observed increase in run time. However, AL-WI {1} lost almost 0.07 in $F_{1.0}$. Not as much of a decrease as for SL-NB {3}, but more than we would like.

SL-WI {2} seems to scale up the best, increasing execution time by a factor of a little over 2 (it is, from Equation (9) on page #124, with d unchanged, $O(t_a)$). It lost only .006 accuracy and .004 $F_{1.0}$. And this is all with no tuning for full text.

We next did experiments using the Reuters-21578 corpus, full text. The Reuters-21578 corpus is a modified version of the Reuters-22173 corpus, but has several different characteristics (discussed earlier). We ran the same 3 systems as above on this corpus, and obtained results that were similar to the above (for Reuters-22173 full text) in time, fraction of available training examples used, and accuracy. Interestingly, the SL-NB {3} system also obtained essentially the same level of $F_{1.0}$ as it did for the Reuters-22173 corpus. However, the two winnow systems scored almost 0.09 lower on $F_{1.0}$. We purposely did not tune δ for this different corpus, so this probably had a lot to do with the decrease in $F_{1.0}$ performance. When one examines the learning traces for the 2 components of $F_{1.0}$ for the 2 corpora, something very interesting is revealed. Precision learning traces

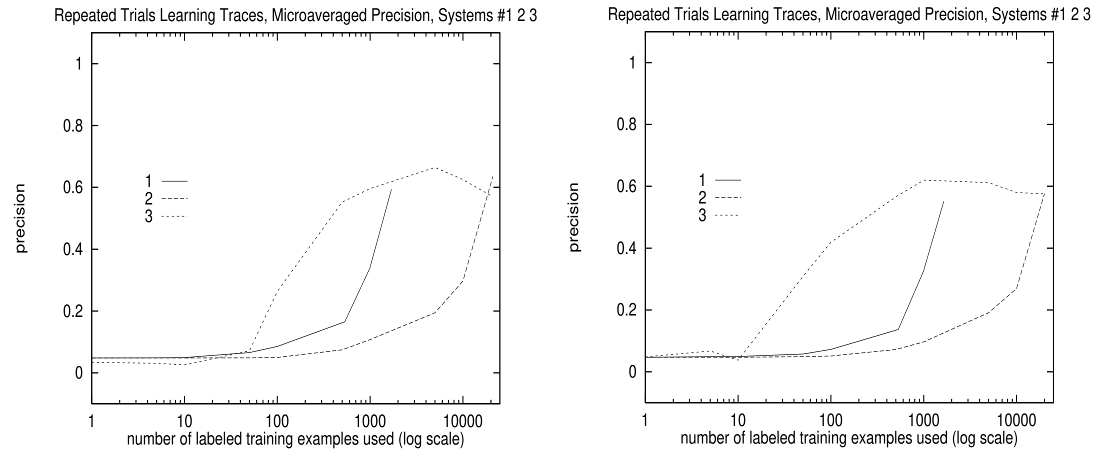

Figure 37. Precision Learning Traces: Reuters-22173 (left) and Reuters-21578 (right) [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 7 members, winnow, majority voting; 3 = supervised learning, naive Bayes]

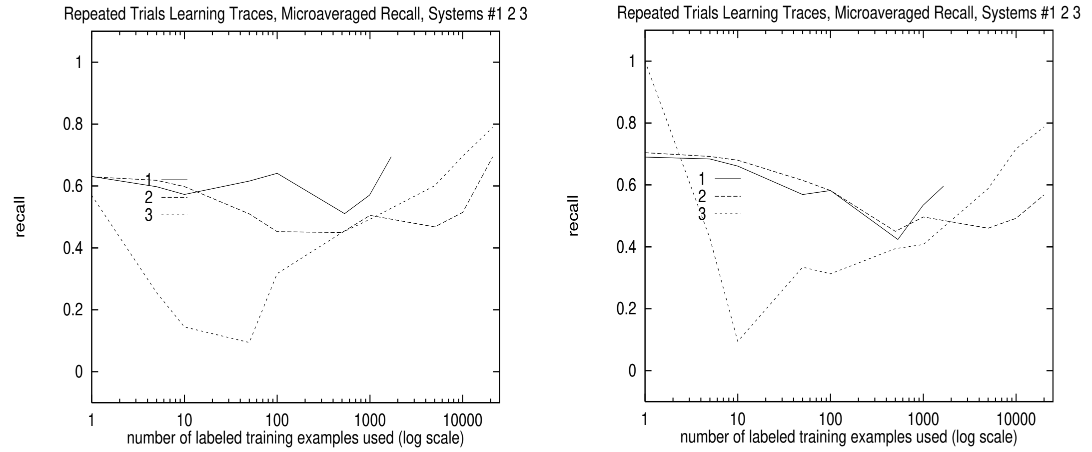

Figure 38. Recall Learning Traces: Reuters-22173 (left) and Reuters-21578 (right) [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 7 members, winnow, majority voting; 3 = supervised learning, naive Bayes]

for the 2 corpora are shown in Figure 37. Note that when comparing the two corpora, the curves look very similar for each of the 3 systems. The learning traces for recall are shown in Figure 38, and obviously here is where the difference lies.

In Figure 38, we see that the general shape of the SL-NB {3} curves are similar across the 2 corpora. Recall drops but then, as more examples are used, it recovers and reaches 0.79 for both corpora. The curves for AL-WI {1} and SL-WI {2} and are also very similar in shape across the 2 corpora, but for Reuters-21578, the variations in the curves are less pronounced and, as compared to the curves for Reuters-22173, appear to be drooping. It appears that this is a direct result of not having tuned the system for Reuters-21578 by adjusting δ. This hypothesis is supported by the fact that SL-NB {3} did not exhibit this compression in the variations, and it does not use the δ parameter. This hypothesis is also supported by the fact that precision was not affected – of the documents predicted to be YES, the fraction of documents that were actually YES did not change. But recall dropped – of the documents that are actually YES, fewer were predicted to be YES. These events could occur if the hyperplanes had not moved far enough during learning, which again points to δ needing to be tuned for Reuters-21578. To see this, envision that the hyperplanes, when randomly initialized, are each intersecting the NO and/or YES clouds. Recall that winnow is a mistake-driven learner. Therefore, hyperplanes which use a negative example will move away from the origin and towards the YES cloud if they predicted YES and will not move at all of they predicted NO. Similarly, hyperplanes which use a positive example will move towards the origin and towards the NO cloud if they predicted NO and will not move at all of they predicted YES. Recall that the categories in these corpora have far more negative examples than positive ones, so on the average most hyperplane motion will be away from the origin. Thus hyperplanes initially located in the YES cloud are more likely to remain there (since they already predict NO on most examples used and since they are correct they do not move during learning). Thus these hyperplanes will, if the learning rate is too low, end up well within the YES cloud, and so will correctly predict YES for examples located above them, thus getting a good precision score, but they will incorrectly predict NO for examples in the YES cloud that are below them, thus getting a poor recall score.

Conclusions: for titles only, other experiments that we performed indicated that the document space as seen by the learning algorithms is fairly consistently structured between Reuters-22173 and Reuters-21578. However, for full text, it would appear from the above analysis that using a larger δ would give better results. This is also consistent with the fact that Reuters-21578 was created from (a portion of) Reuters-22173, and the creation process included correction of "a variety of typographical and other errors in the categorization and formatting of the collection" [Reuters-21578]. Thus Reuters-21578 is in a sense a "cleaner" corpus, which in turn would warrant using a larger value of δ (and a correspondingly higher learning rate).

Time allowed us to try out this idea. We increased δ by approximately 15% and did obtain better results on the Reuters-21578 corpus. We do not claim that this was a "thorough tuning", but we did get much better results. Since modifying δ does not affect naive Bayes operation, we did not need to include system 3 in this final series of trials. Figures 39 and 40 show the learning traces for accuracy, $F_{1.0}$, precision, and recall. Notice that adjusting δ has cured the droop problem that was occurring in recall.

To recap this series of experiments: we had been doing all experiments using Reuters-22173 titles only, and wanted to show that our method scaled up to the processing of full text documents with no parameter changes being made. So we ran tests using Reuters-22173 full text. Compared to Reuters-22173 titles only, performance remained good with the possible exception of AL-WI {1}, for which $F_{1.0}$ decreased by 0.07 (which is about 10% of its titles only value). We then ran the same systems on Reuters-21578, still without any tuning. The winnow systems

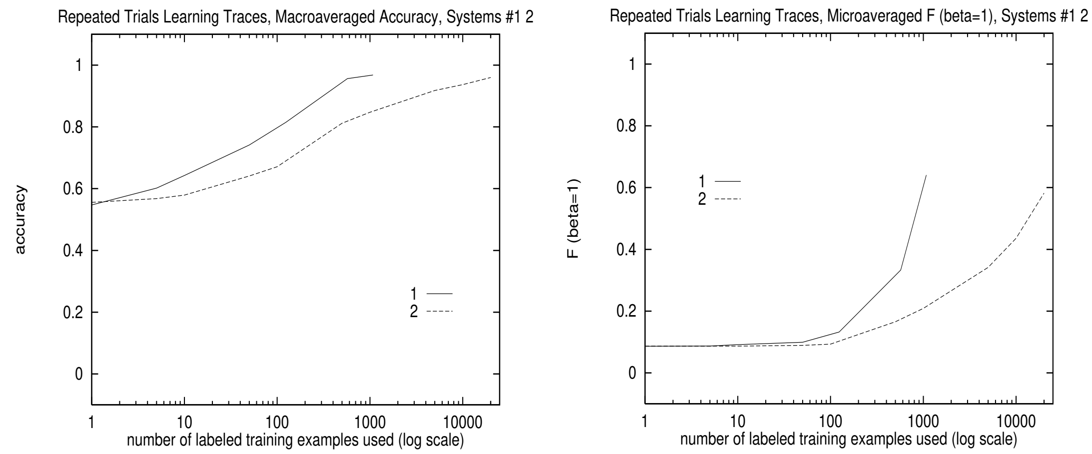

Figure 39. Learning Traces: Accuracy (left) and $F_{1.0}$ (right) for Reuters-21578 [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 7 members, winnow, majority voting]

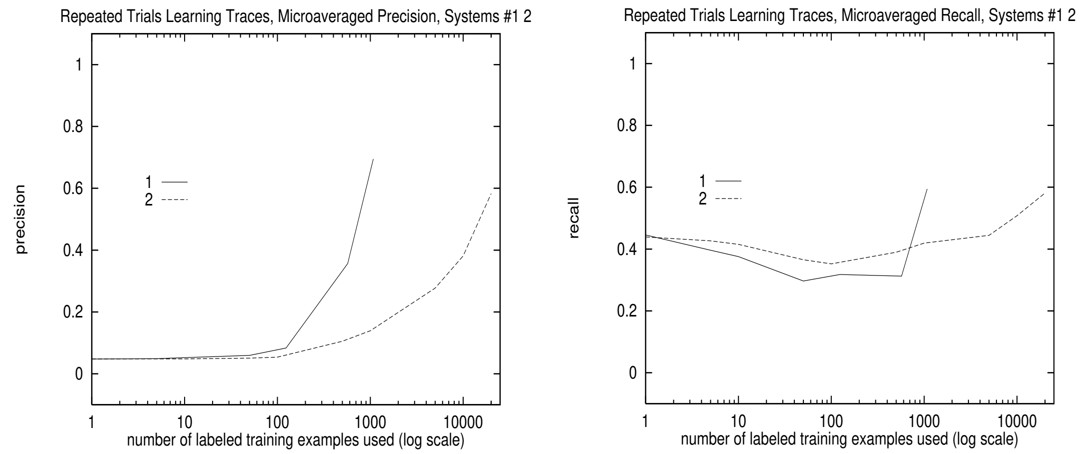

Figure 40. Learning Traces: Precision (left) and recall (right) for Reuters-21578 [1 = ALC, 7 members, AllMargin, winnow, majority voting; 2 = supervised learning, 7 members, winnow, majority voting]

dropped in $F_{1.0}$ by about 0.09 below Reuters-22173 full text. So we decided to see if "minor tuning" for Reuters-21578 would help. By making only an adjustment in δ consistent with our analysis of the Reuters-21578 corpus, we were able to obtain performance similar to what we obtained on Reuters-22173 full text. Table 18 summarizes the Reuters-21578 full text results.

TABLE 18. Average Performance of Reuters-21578 Full Text

system	*time*	*% tng ex used*	*accuracy*	$F_{1.0}$
AL-WI {1}	223.6	5.4%	0.968	0.641
SL-WI {2}	3605.5	100%	0.960	0.582
SL-NB {3}	138.5	100%	0.962	0.665

6.8 Different Notions of 'Mistake'

When we introduced the notion of "mistake", we casually defined it to mean "predicting incorrectly". Looking at the contingency table we are using (see Table 19), we see that making a mistake on a training example corresponds to the learning algorithm making a prediction on that example which is different from the actual label that is provided by the teacher, and therefore we end up at location *b* or *c* for that example. Thus when we say that an algorithm (such as winnow and perceptron) is "mistake-driven", we mean that the algorithm learns if and only if its prediction is different from the actual label. In fact, the definition of mistake ("predicting incorrectly") and the definition of accuracy ("fraction of documents that are *correctly* categorized") are obviously related. The notion that occurred to us is: if the implementation of the common definition of mistake is meant to cause the learning algorithm to increase in accuracy, then perhaps some other definition

TABLE 19. 2x2 Contingency Table

		actual label value NO	*actual label value* YES
predicted label value	NO	a (true negatives)	b (false negatives)
	YES	c (false positives)	d (true positives)

of mistake will cause the same algorithm to increase in the performance measures that are of more interest to us in text categorization, namely precision and recall.

We decided to do experiments to see if changing the definition of "mistake" that is used by winnow (a mistake-driven learning algorithm) would produce increases in the recall and precision performance measures, and if so, with what side effects. We looked at the following 3 definitions for mistake, as regards the location/s in the contingency table corresponding to the state of the learner after making a prediction on an example and then seeing the actual label:

1. *b* or *c* (the standard definition of "mistake", learning from it is meant to and does increase accuracy)

2. *b* / false negatives only

3. *c* / false positives only

The normal definition of mistake causes accuracy to increase because the learning algorithm will learn from an example on which the current hypothesis of the learning algorithm is not accurate, the idea being that after learning, the hypothesis will be more accurate overall. In examining the contingency table

(Table 19), note that the real world establishes a correct value both for $b + d$ (i.e., the total number of test documents that are actually YES) and for $a + c$ (i.e., the total number of test documents that are actually NO). The purpose of a mistake-driven learner that is using the normal definition of mistake is to learn training examples that are now b occurrences so that later when that part of the document space is tested, the learner will correctly predict YES ... i.e., obtain an entry in d. And similarly for c occurrences $\rightarrow$ a entries.

If we define a mistake as occurring only when the b / false negative situation arises, then the learning algorithm will perhaps learn from that example and later testing in that part of the document space will instead result in a d / true negative entry. This will increase recall since recall = $\frac{d}{b+d}$ and if b decreases by 1 and d increases by 1, then the denominator remains constant but the numerator increases, so recall will increase. Similarly, perhaps if we define a mistake as occurring only when the c / false positive situation arises, then the learning algorithm will perhaps learn from that example and later testing in that part of the document space will instead result in an a / true positive entry. This will cause precision = $\frac{d}{c+d}$ to go up (because of decreasing c and not changing d).

To test this hypothesis, we ran an experiment using the Reuters-22173 titles only corpus, using both supervised and ALC systems. All of the systems use a committee of 7 winnow learners and predict using majority vote. All of the ALC systems use AllMargin to decide when to see the label. We ran the experiment using the following 6 systems:

{1} AL-bc

ALC, mistake is b or c (standard definition of mistake, causes accuracy to increase)

{2} SL-bc

supervised learning, mistake is *b* or *c* (standard definition of mistake, causes accuracy to increase)

{3} AL-b

ALC, mistake is *b* only (we think this will cause recall to increase)

{4} SL-b

supervised learning, mistake is *b* only (we think this will cause recall to increase)

{5} AL-c

ALC, mistake is *c* only (we think this will cause precision to increase)

{6} SL-c

supervised learning, mistake is *c* only (we think this will cause precision to increase)

We will also refer to the above systems as "accuracy prone", "recall prone", or "precision prone", depending on the definition of mistake being used.

Figure 41 shows elapsed processor time versus number of training examples used. The supervised learners all take essentially the same amount of time. They are using all examples (but learning only from the ones on which they make mistakes). Since most categories have few positive examples, it is probably true that when an error is made in predicting the label, it is more often a *c* / false positive entry than a *b* / false negative entry. So why are all supervised learning systems taking essentially the same amount of time? The time complexity for all 3 systems is the same – namely $O(dt_a)$ (ref. Equation (9) on page #124), but recall that this is an upper bound. The actual learning time is more like $O(Lt_a)$, where L =

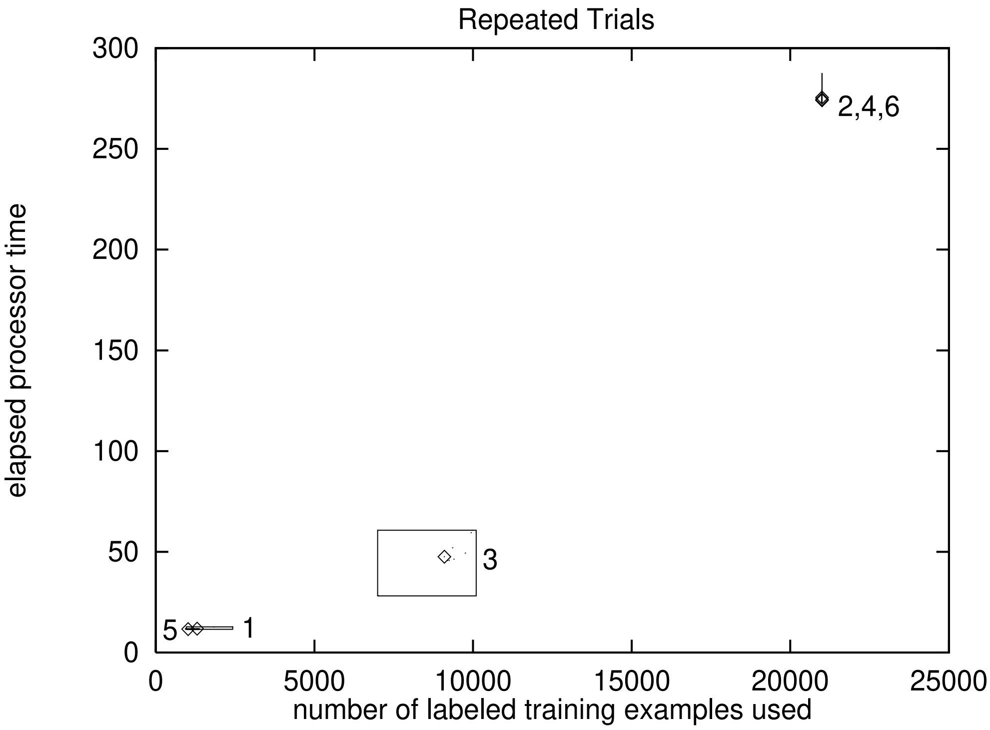

Figure 41. Elapsed Time versus Number of Training Examples Used [1 = mistake is b or c, ALC, 7 members, AllMargin, winnow, majority voting; 2 = mistake is b or c, supervised learning, 7 members, winnow, majority voting; 3 = mistake is b, ALC, 7 members, AllMargin, winnow, majority voting; 4 = mistake is b, supervised learning, 7 members, winnow, majority voting; 5 = mistake is c, ALC, 7 members, AllMargin, winnow, majority voting; 6 = mistake is c, supervised learning, 7 members, winnow, majority voting]

number of documents learned from. (In our worst-case analysis, we had set $L = d$ = number of documents). The point being that the time spent in learning in these systems is small compared to the time spent predicting (where the d is not worst-case but is actual), so the predicting time (and system overhead) dominate. Before one can determine if the training example represents a mistake, the actual label must be obtained, and it is the number of training examples *used* that we are looking at on the plot.

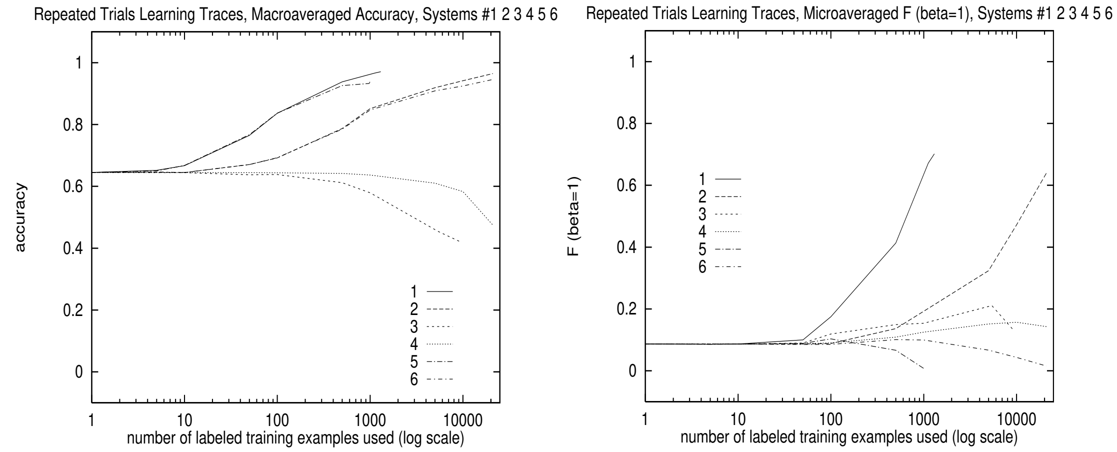

Figure 42. Learning Traces: Accuracy (left) and $F_{1.0}$ (right) [1 = mistake is b or c, ALC, 7 members, AllMargin, winnow, majority voting; 2 = mistake is b or c, supervised learning, 7 members, winnow, majority voting; 3 = mistake is b, ALC, 7 members, AllMargin, winnow, majority voting; 4 = mistake is b, supervised learning, 7 members, winnow, majority voting; 5 = mistake is c, ALC, 7 members, AllMargin, winnow, majority voting; 6 = mistake is c, supervised learning, 7 members, winnow, majority voting]

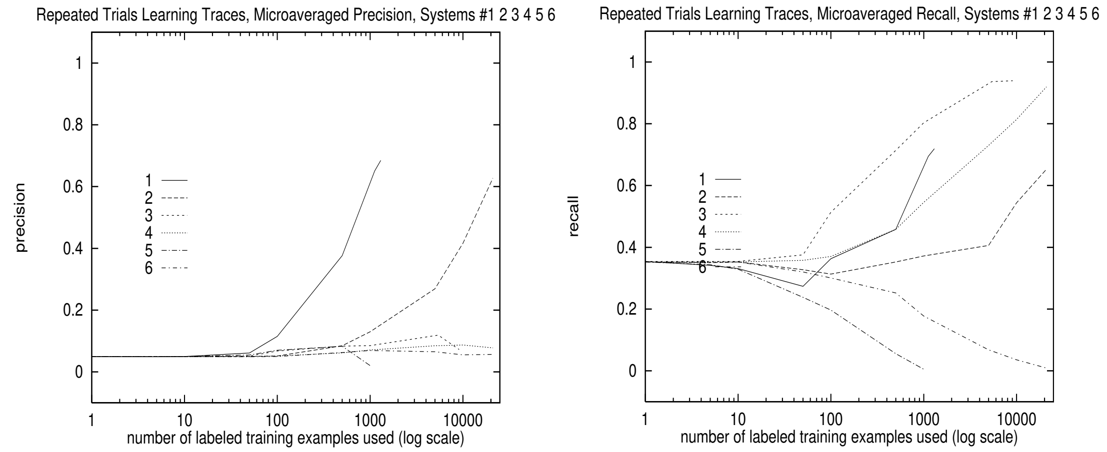

Figure 43. Learning Traces: Precision (left) and Recall (right) [1,2,3,4,5,6=see above]

Figure 41 also shows that the ALC systems use fewer examples and require less time than the supervised learners. AL-c {5} uses the fewest examples (4.8%) followed by AL-bc {1} with 6.03% and AL-b {3} with 43.3%. AL-bc {1} and AL-c {5} take about the same amount of time, AL-b {3} takes more time because it is using more training examples. Figures 42 and 43 show the accuracy, $F_{1.0}$, precision, and recall learning traces for the 6 systems.

First, let us examine whether or not the different definitions of mistake had the anticipated effect on their corresponding performance measures. The accuracy prone systems (AL-bc {1} and SL-bc {2}) were supposed to provide high accuracy, and according to Figure 42 (left), these systems did indeed obtain the highest accuracies (0.971 and 0.965 respectively). The recall prone systems (AL-b {3} and SL-b {4}) were supposed to provide high recall, and according to Figure 43 (right), they did (0.939 and 0.919 respectively). The precision prone systems (AL-c {5} and SL-c {6}) were supposed to provide high precision, but according to Figure 43 (left), they definitely did not.

Looking in more detail at the hoped-for main effects and also at the side effects of our modifications to the definition of mistake, we see that:

1. The accuracy prone systems achieved by far the best precision and did a decent job on recall. The recall prone systems did best at recall [with 0.939 and 0.919], but the accuracy prone systems came in at relatively good values of recall of 0.719 and 0.653. This resulted in the accuracy prone systems being the only ones to obtain good $F_{1.0}$ scores.

2. The recall prone systems did do very well on recall, but were abysmal in precision, and accuracy actually *decreased* as they learned. Examination of the behavior of these systems during learning reveals that the learners, since they learn only from examples whose actual label is YES, eventually learn to "just say YES". This is borne out by examination of the contingency tables as

learning progresses. An example of a typical final contingency table follows:

		actual label value NO	YES
predicted label value	NO	257	8
	YES	769	139

The system has in fact obtained excellent recall (0.95), but by having learned to just say YES, the system does poorly on documents whose actual label is NO, and so has low accuracy (0.34) and poor precision (0.15).

3. Interestingly enough, the precision prone systems, while doing very poorly on their "assigned" performance measure of precision, obtained quite good accuracy scores – 0.944 and 0.946, only slightly below the scores obtained by the accuracy prone systems (0.971 and 0.965). The precision prone systems did terribly on recall, actually losing recall as learning progressed. One could assign this latter effect to precision-recall tradeoff if these systems had been gaining in precision, but they were not. Examination of the behavior of these systems during learning reveals that the learners, since they learn only from examples whose actual label is NO, eventually learn to "just say NO". This is supported by examination of the contingency tables as learning progresses. An example of a typical final contingency table follows:

		actual label value NO	YES
predicted label value	NO	1135	26
	YES	11	1

> Since most categories have few positive examples, just saying NO yields a highly accurate learner – in this case accuracy is 0.97. However, precision (0.08) and recall (0.04) are very low.

One remaining observation. Why did AL-b {3}, the recall prone active learning system, use so many more examples than the other active learning systems (see Figure 41)? The reason is that, by learning only from positive examples, the learners tend to learn to just say YES, which means that the hyperplanes move closer to the origin. However, most training examples are in fact NO and so occupy this same region of document space close to the origin, so this makes it more likely that the committee members will disagree amongst themselves on the predicted label for a candidate training example, and so makes it more likely that the example will be selected as informative and used.

6.9 Eager vs. Mistake-Driven Learning and Skeptical vs. Optimistic Learning

The purpose of this experiment is to explore some other approaches to learning that may provide benefit in our domain of text categorization with minimal preprocessing. First, we need to introduce some additional terminology and then we will discuss the motivation for this experiment. We then briefly discuss the normal rationale for using learning algorithms with certain characteristics. Then we will discuss the experiment itself and the results we obtained.

6.9.1 Terminology

Let us first define some additional terms. Recall that "supervised learning" refers to machine learning in which a teacher provides the label for all training examples, and "active learning" refers to machine learning in which the learning algorithm has some control over the examples from which it learns. If the learning algorithm learns from unlabeled examples (using just the attribute portion of the feature vector), then one is using "unsupervised learning". In unsupervised learning, the labels are not used at all and thus no teacher is needed.

The "normal" situation is that learning can occur only after seeing the label. This is how all of the experiments we have discussed so far have worked, and in fact how most machine learning systems operate. We define "unlabeled training example" as a training example for which the learner does not know the actual label because the learner did not ask the teacher for the label. (The distinction we are making here is that we are not considering the case where the learner does ask for the label, but then for some reason simply does not use it in any way.) In this set of experiments, the learning algorithm may or may not, on a per system basis, *also* learn from unlabeled training examples. A machine learning system will be said to be "skeptical" if it must see the actual label before learning from a training example. A machine learning system that will *in addition* learn from training examples whose labels are predicted by the system and not obtained from the teacher will be referred to as "optimistic". Thus an optimistic learner learns from both labeled and unlabeled examples.

Recall from earlier definitions that a training example is "selected" if the system determines that it is an informative example, a selected training example is "used" when the system requests its actual label, and a selected and used training example is "learned from" when actual changes in the knowledge base of the learner are made in response to that example. If one has a stream of candidate training examples entering a skeptical system, then only after a particular example is selected and used might learning occur. (We say "might" because even then, learning may not occur – it depends on whether or not the learning algorithm actually changes its knowledge base in response to the example). All of the systems we have discussed thus far are skeptical, and this is also by far the normal situation in machine learning. On the other hand, if one has a stream of candidate training examples entering an optimistic system, then learning might occur if a particular example is selected and used, but learning might also occur if the example were not selected (and therefore not used).

We need to introduce one additional new term. We have made a distinction as to whether or not learning algorithms are mistake-driven. For example, standard winnow and perceptron are mistake-driven, but standard naive Bayes is not mistake-driven. We also included this distinction when deriving $O(\,)$ behavior for the code that decides whether or not to see the label. The term "not mistake-driven" is a bit awkward, so we will use the term "eager" to describe learning algorithms that are not mistake-driven. In other words, standard naive Bayes is an eager learning algorithm since it learns from any example presented to it.

6.9.2 Motivation for This Experiment

Our decision to perform this experiment comes from 2 papers, one by Servedio [Servedio 1999] and one by Nigam, McCallum, Thrun, and Mitchell [Nigam et al. 1998]. Each paper discussed some very interesting concepts that we wanted to apply to our text categorization domain to see if we could use them to improve our systems.

Servedio derives some interesting theoretical results regarding the mistake-driven nature of the perceptron algorithm for situations in which the data follows a certain type of uniform distribution. He shows that the perceptron is actually better when used in an eager manner in situations in which there is no attribute noise but which do have classification noise of certain types present. He finds that the eager algorithm is faster and also is able to learn patterns that the mistake-driven perceptron can not learn. We realize that our domain does not exactly match the one for which Servedio derived his results, but we were interested in determining if our domain was perhaps "close enough" to obtain some benefit from his ideas. We have already shown that, in our experiments, winnow is superior to perceptron, but we decided to examine the behavior of both the winnow and perceptron algorithms when used in an eager manner. We know that our domain has both attribute noise and classification noise, and we strongly suspect that the classification noise present in our domain does not exactly follow the model used by Servedio. We also strongly suspect that our data does not follow his uniform distribution requirements, but we are not really sure. We therefore also included some computationally simple calculations to determine if and by how much our data violates the uniform distribution conditions. These calculations are discussed in detail later.

Nigam, McCallum, Thrun, and Mitchell [Nigam et al. 1998] use unlabeled examples to augment the learning accomplished using labeled examples by having the EM algorithm compute labels for the unlabeled data assuming that the labeled and unlabeled documents come from the same overall distribution. The EM algorithm has a quite high computational complexity, so using it is not practical in our case, since we have very large numbers of attributes. However, we did want to see if we could somehow, in an efficient manner, obtain performance gains from the use of the unlabeled examples.

Before we discuss the experiment, we need to take a couple of brief asides – one regarding the motivation for using unsupervised learning and one regarding the motivation for using mistake-driven learning algorithms.

6.9.3 Unsupervised Learning

The motivation for wanting to use unsupervised learning is that it is much cheaper than supervised learning or active learning – cheaper in terms of human time and money costs. Unsupervised learning occurs when the learner is not provided with any labels by the teacher, it uses only the documents themselves. The unsupervised learner attempts to detect patterns in the documents so that the learner can impose structure on the input data and form clusters of data points. The points in each cluster form a class. In several methods, a description of each class is also generated. The characteristics of the clusters are then used by the unsupervised learner to categorize new documents.

Castelli and Cover used Bayesian analysis to determine the relative worth of labeled versus unlabeled examples [Castelli and Cover 1995]. They concluded that labeled examples are exponentially more valuable than unlabeled examples. This indicates that one will save a great deal of (computational) learning effort by using labeled rather than unlabeled examples. A learning algorithm will have to work much harder (in time or space or both) to learn the same thing from unlabeled examples as from labeled ones.

Ratsaby and Venkatesh analyzed the effects of labeled versus unlabeled examples when one also knows some additional information [Ratsaby and Venkatesh 1995]. In their analyses, this additional information is the parametric form of the class conditional densities. They also analyzed the nonparametric case. They found large differences in the worth of labeled examples as compared to unlabeled ones – a polynomial factor in the parametric case, and an exponential factor in the nonparametric case.

Unfortunately, examples do not label themselves – a human must do that. For some situations, this is not practical and perhaps not even possible. The "exponentially more valuable" aspect of labeled examples can be viewed as the result of partially solving the learning problem by using a preprocessor that utilizes resources, such as time and money, and in this case happens to be a human being. There is thus a tradeoff between the human doing the labeling so that the computer can do less work learning (supervised learning) and the human doing "no" work and the computer spending more time learning (unsupervised learning).

A natural conclusion to draw is that one ought to use only unsupervised learning. This would allow the use of unlabeled examples, which are certainly more plentiful than labeled ones. The dream is to feed an entire corpus into an unsupervised learning algorithm, and it will perform text categorization. Of course, the question is: will the categorization determined by the unsupervised learning algorithm be a categorization that is of some use to a human? There are many reasonable ways to categorize a particular collection of documents. There is no reason to think or even hope that categorization by general topic content is how an unsupervised learning algorithm will do it, in the absence of additional information, suggestions, proddings, etc.

We in fact thought that an unsupervised classification approach was worth investigating [Liere and Tadepalli 1996]. We conducted some experiments using AutoClass C, an unsupervised Bayesian classifier [Cook et al. 1996, Cheeseman and Stutz 1995, Hanson et al. 1991]. Results indicated that there were simply too many dimensions in the problem to obtain useful results using a reasonable amount of computing resources (time and space). The current state of the art in inferencing using an existing general Bayesian network and performing the inference in a reasonable amount of time (i.e., while a user waits) is around 100 - 300 nodes with few arcs [D'Ambrosio 1996]. Our networks had more like 10,000 - 30,000 nodes, and arcs were often not very sparse. We concluded that, in our very complex domain (text categorization with minimal preprocessing), fully unsupervised learning is too unconstrained and ill-understood at this time to yield useful results.

6.9.4 Mistake-Driven Learning

When one first encounters mistake-driven learning, one may be a bit perplexed as to why, once the learning algorithm has asked to see the label, it does not then use the example for learning. The price has already been paid in that the cost of providing the label by a human has already occurred, so it seems "wasteful" to not actually adjust the learner's knowledge base in response to the information in the example.

Recall that when we discussed the naive Bayes learning algorithm, we used a pets example (ref. page 97). We had a small collection of documents and the task was to categorize them as to whether or not they were "about pets". When we discussed the pets example, we touched slightly on overfitting and informally described it as what happens when a machine learning algorithm stops generalizing information in the examples and starts learning specific training examples. This can result in the learning algorithm performing poorly when asked to predict on unseen examples, as it has in a sense, memorized examples instead of acquiring general knowledge. More formally, overfitting is defined as [Mitchell 1997]:

> a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution (i.e., including instances beyond the training set).

In other words, it is possible to glean too much information from specific examples and thereby lose some of the ability to generalize knowledge to the broad class of problems of interest. Mistake-driven learning algorithms are able to avoid overfitting, or at least seem better able to avoid it, because they do not learn from examples on which they already predict correctly [Dietterich 1997]. The notion is that the learner already knows that example and so there is no need to learn. Value has in fact been obtained from the actual label – the knowledge that the learner already knows the example.

Besides overfitting, mistake-driven learning algorithms are attractive because they can run faster than eager ones. Mistake-driven algorithms update their knowledge bases less often (only when they make a prediction mistake). However, recall that a mistake-driven algorithm must always predict on the candidate training example before learning, as a comparison of the predicted with the actual label is what determines whether or not a mistake has been made. If prediction is inexpensive as compared to learning, then mistake-driven learning can overall be much faster than eager learning.

6.9.5 The Actual Experiment

The standard use of the winnow and perceptron learning algorithms is mistake-driven and skeptical, and that is how all of our previous experiments have used them. The purpose of this experiment is to investigate their use as eager (as compared to mistake-driven) and/or optimistic (as compared to skeptical).

To obtain eager learning, we simply remove the restriction in winnow and perceptron that learning occur only if the predicted and actual labels differ. This also means that we do not, at least insofar as learning is concerned, need to have each committee member predict on each example that is going to be used.

To obtain optimistic learning, we do the following. Since on the average the committee of learners is right more often than it is wrong (i.e., accuracy > 0.50), we will use the label predicted by the committee as the actual label value for training examples for which we do not ask the teacher for the actual label. In other words, the stream of candidate training examples enters the module that decides whether or not to see the label. Some examples (all of the examples if we are using supervised learning) are selected as especially informative and are then used for learning. So far, both skeptical and optimistic learners behave the same. The difference is in how the examples that are not selected are treated. A skeptical learner discards them. An optimistic learner assumes that the committee predicted label is the actual label, and they can then be learned from.

These experiments were conducted using the Reuters-22173 titles only corpus. We again used a factorial experiment design. We wanted to include systems that have one characteristic in each of the following 4 categories:

1. method for deciding whether or not to see the label: always (supervised ["SL"]) or AllMargin (active learning ["AL"])

   (We chose AllMargin since it is the best active learning method that we have developed so far for deciding whether or not to see the label).

2. learning algorithm: winnow ["W"] or perceptron ["P"]

3. can unlabeled examples be learned from? yes (optimistic ["O"]) or no (skeptical ["S"])

4. when training examples are used, which ones are actually learned from? all (eager ["E"]) or only ones for which the individual member prediction is in error (mistake-driven ["M"])

It would appear that we will need $2^4 = 16$ systems in our experiment. However, systems that are optimistic supervised learners in a sense "do not exist" because in a supervised learning system, labels for all of the training examples are by definition selected and used, and so none of the training examples are unlabeled. This means that we need $2^2 + 2^3 = 12$ systems. All 12 systems use a 7 member committee and predict by majority vote. The 12 systems (using the characteristics ["X"] listed above) are described in Table 20. We will refer to each of the 12 systems using these characteristics in a left-to-right fashion. For example, system #1 is SL-W-S-M {1}.

TABLE 20. Characteristics of the 12 Systems

system number	*see label method: SL or AL*	*learning algorithm: W or P*	*unlabeled ex. available for learning? yes (O) or no (S)*	*examples learned from: all (E) or mistakes (M)*
1	SL	W	S	M
2	SL	W	S	E
3	AL	W	O	M
4	AL	W	O	E
5	AL	W	S	M
6	AL	W	S	E
7	SL	P	S	M
8	SL	P	S	E
9	AL	P	O	M
10	AL	P	O	E
11	AL	P	S	M
12	AL	P	S	E

Some observations about the individual learners in the above systems:

1. Systems #1 and #7 are the standard skeptical mistake-driven committees of supervised learners. All training examples are labeled. Each learner learns from all examples on which its prediction is incorrect.

2. Systems #5 and #11 are the standard skeptical mistake-driven committees of active learners. They select training examples for use in learning, and of those selected and used, each learner learns only from those on which its prediction was incorrect.

3. Systems #2 and #8 are supervised committees of skeptical eager learners. They learn from all examples, and all examples are labeled.

4. Systems #3 and #9 are optimistic mistake-driven active learners. They learn from selected examples on which prediction was incorrect, and they also learn from unlabeled examples on which prediction was incorrect.

5. Systems #4 and #10 are optimistic eager active learners. They learn from all examples – from all selected examples and from all unlabeled examples.
6. Systems #6 and #12 are skeptical eager active learners. They learn from all selected labeled examples.

Figure 44 (left) shows elapsed processor time versus number of labeled examples used for all 12 systems. This information is also shown on Table 21. The supervised systems take much more time than the active learning systems. The supervised eager systems SL-W-S-E {2} and SL-P-S-E {8} take a little more time than their mistake-driven counterparts SL-W-S-M {1} and SL-P-S-M {7}, which is as expected since the eager supervised systems are learning from all examples. All of the active learning systems use far fewer examples and far less time. Figure 44 (right) focuses on only the active learning systems. The purpose in showing this detail is to show the locations of the average points (◊). On the average, it appears that the winnow systems ({3}-{6}) use more training examples than the perceptron systems ({9}-{12}).

Figures 45 through 48 give the learning traces that resulted from the tests that we ran on these 12 systems. Each figure contains 2 plots. We plotted systems 1 - 6 and systems 7 - 12 separately so that the separate learning traces are easier to see. Note that systems 1 - 6 are winnow systems and systems 7 - 12 are perceptron systems. Also note that systems s and $s+6$ have the same characteristics except for the learning algorithm, and their learning traces are made using the same line style. Therefore systems 1 and 7 are the same (except for learning algorithm), as are 2 and 8, 3 and 9, etc. Accompanying each pair of figures is a table giving the final values of those same performance measures for each of the 12 systems.

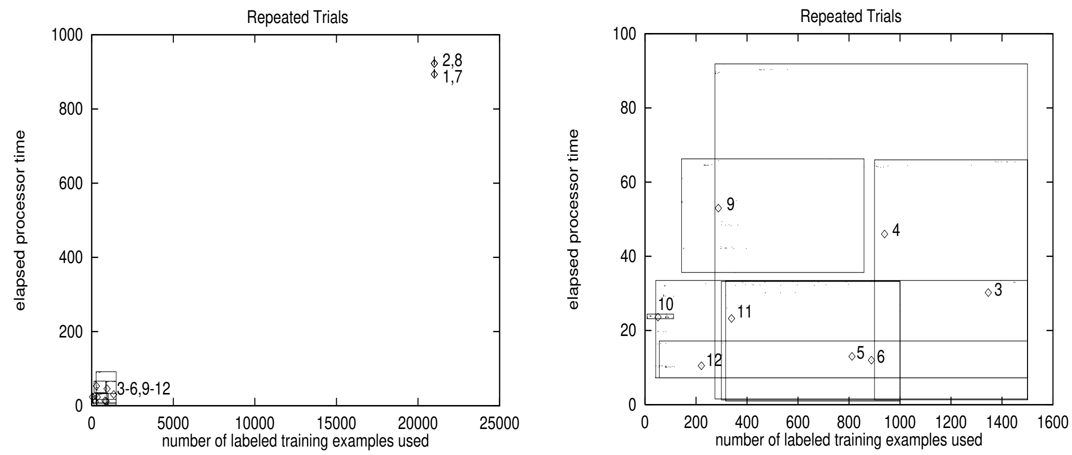

Figure 44. Elapsed Time versus Number of Training Examples Used: all systems (left), AL systems only (right)

TABLE 21. Final Performance Measure Values: Time and Training Examples Used

system	*time*	*% tng ex used*
SL-W-S-M {1}	893.9	100
SL-W-S-E {2}	922.2	100
AL-W-O-M {3}	30.2	6.4
AL-W-O-E {4}	46.0	4.5
AL-W-S-M {5}	13.0	3.7
AL-W-S-E {6}	12.0	4.2
SL-P-S-M {7}	893.3	100
SL-P-S-E {8}	923.3	100
AL-P-O-M {9}	53.0	2.4
AL-P-O-E {10}	23.6	2.4
AL-P-S-M {11}	23.3	1.6
AL-P-S-E {12}	10.5	1.1

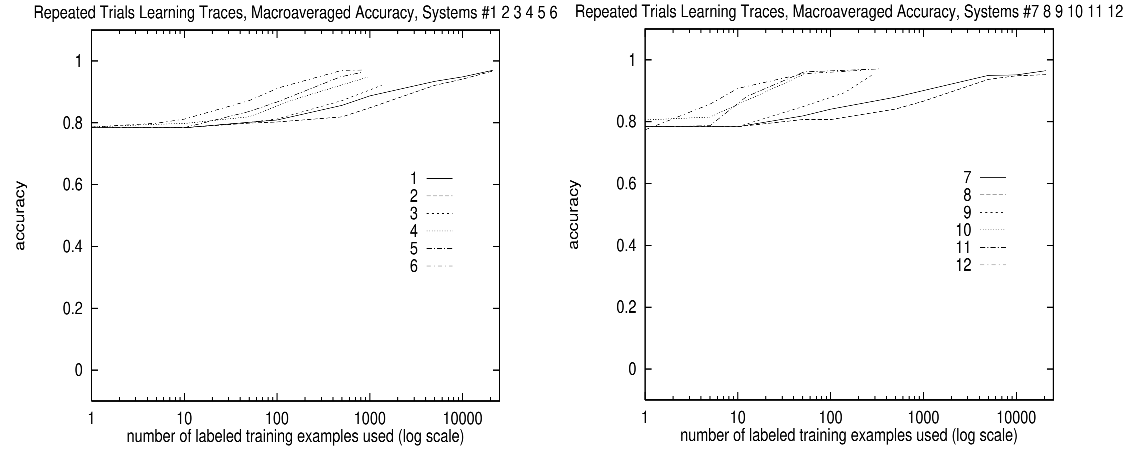

Figure 45. Accuracy Learning Traces: systems 1-6 (left) and 7-12 (right)

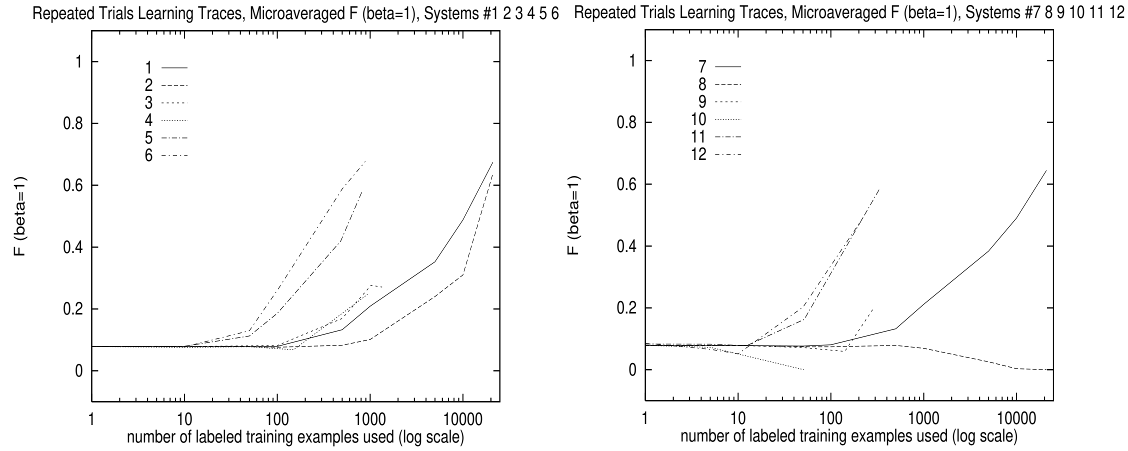

Figure 46. $F_{1.0}$ Learning Traces: systems 1-6 (left) and 7-12 (right)

TABLE 22. Final Performance Measure Values: Accuracy and $F_{1.0}$

system	*accuracy*	$F_{1.0}$
SL-W-S-M {1}	0.970	0.674
SL-W-S-E {2}	0.967	0.637
AL-W-O-M {3}	0.922	0.271
AL-W-O-E {4}	0.947	0.247
AL-W-S-M {5}	0.963	0.577
AL-W-S-E {6}	0.971	0.677
SL-P-S-M {7}	0.965	0.645
SL-P-S-E {8}	0.952	0.0
AL-P-O-M {9}	0.954	0.198
AL-P-O-E {10}	0.952	0.0
AL-P-S-M {11}	0.971	0.588
AL-P-S-E {12}	0.967	0.491

Accuracy (Figure 45) is about the same for all 12 systems, ranging from a low of 0.922 to a high of 0.971. While there is some difference in accuracy, we will see (as we have in the past) that it is not a very good performance measure for this domain since we have so few positive examples. Systems can obtain a high accuracy score with behavior that is not what most users want in a text categorization system. Examining $F_{1.0}$ (Figure 46), we see that 7 of the systems (1, 2, 5, 6, 7, 11, and 12) obtained adequate or better $F_{1.0}$ scores, ≈ 0.50 or above. The 7 systems seem to subdivide into two groups – systems 6, 1, 7, and 2 (in order of

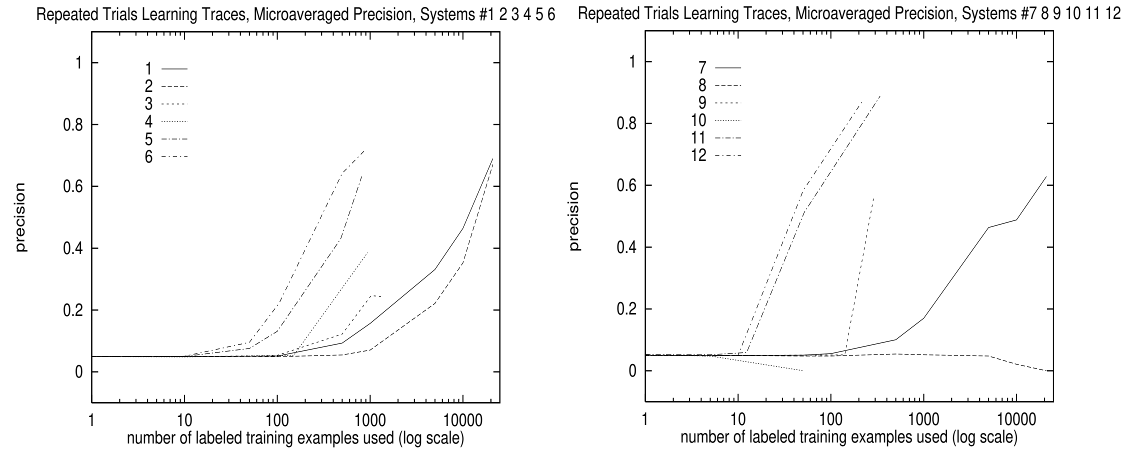

Figure 47. Precision Learning Traces: systems 1-6 (left) and 7-12 (right)

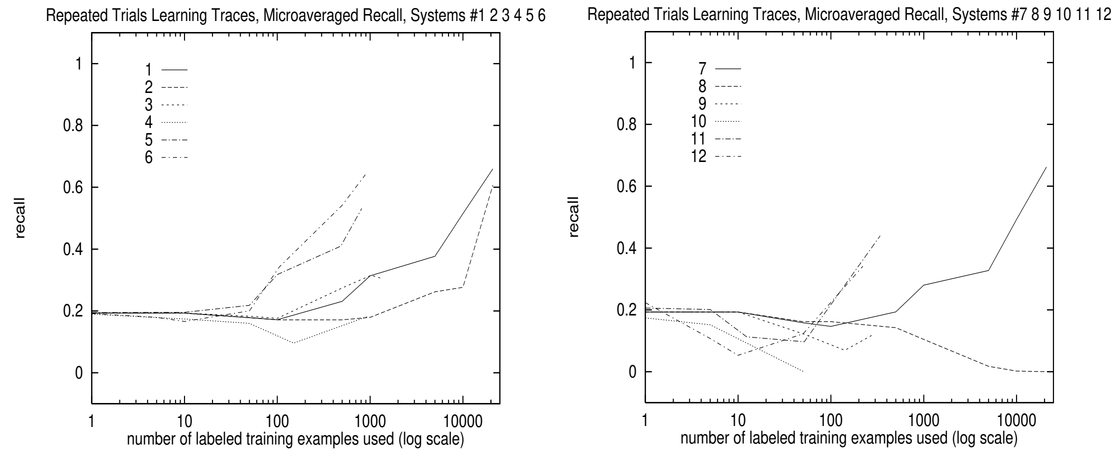

Figure 48. Recall Learning Traces: systems 1-6 (left) and 7-12 (right)

TABLE 23. Final Performance Measure Values: Precision and Recall

system	*precision*	*recall*
SL-W-S-M {1}	0.690	0.659
SL-W-S-E {2}	0.671	0.607
AL-W-O-M {3}	0.244	0.305
AL-W-O-E {4}	0.386	0.182
AL-W-S-M {5}	0.632	0.531
AL-W-S-E {6}	0.718	0.641
SL-P-S-M {7}	0.628	0.662
SL-P-S-E {8}	0.0	0.0
AL-P-O-M {9}	0.557	0.121
AL-P-O-E {10}	0.0	0.0
AL-P-S-M {11}	0.889	0.440
AL-P-S-E {12}	0.875	0.341

decreasing $F_{1.0}$) which we will refer to as the "very good" systems, and systems 11, 5, and 12 which we will refer to as the "satisfactory" systems. Note that the very good systems are all skeptical eager systems, differing in want-to-see-the-label-method (active/supervised) and learner (winnow/perceptron). From Figure 47, we see that all 7 of these systems obtained quite good precision scores (in [0.628, 0.889]). Note that the variation in $F_{1.0}$ is often more due to the variation in recall, which (Figure 48) varied in these 7 systems from 0.341 to 0.662. The better performers were the systems that were better able to balance precision and recall.

In order to be able to reach some conclusions as to what this all means, we take the contents of Table 20 and draw boxes around the above-identified satisfactory (dashed boxes) and very good (solid boxes) systems. We get the following:

```
 {1} SL-W-S-M
 {2} SL-W-S-E
 {3} AL-W-O-M
 {4} AL-W-O-E
 {5} AL-W-S-M
 {6} AL-W-S-E
 {7} SL-P-S-M
 {8} SL-P-S-E
 {9} AL-P-O-M
{10} AL-P-O-E
{11} AL-P-S-M
{12} AL-P-S-E
```

This makes it clear that all of the satisfactory and very good systems are skeptical rather than optimistic learners. In other words, learning from unlabeled examples (in the manner described above) caused a drop in system performance. According to Figures 45 through 48, all 4 of the optimistic systems (3, 4, 9, 10) had poor recall and moderate to poor precision. This suggests that the use of unlabeled examples in this domain by using the committee prediction as the actual label is not a good idea. When the committee prediction on an unlabeled example is in error and we are using an optimistic learner, the effect is exactly the same as if that example had been selected and used but had been assigned the incorrect actual label by the teacher. One possibility is that the additional class noise that is thus injected into the data using the unlabeled examples in this manner makes it too difficult for the learners to obtain an accurate approximation to the actual target concept.

Perhaps another question to examine is "why didn't SL-P-S-E {8} do better". It is the only skeptical system that did not do well. SL-P-S-E {8} took the most time (Table 21), used all 21,000 of the training examples, and over the series of trials achieved scores of 0.0 for both precision and recall (Figures 48 and 47). And those scores of 0.0 are exact – not a result of rounding off to 3 significant digits. This means that the SL-P-S-E {8} system is learning to just say NO, and doing so very consistently – even for "easy categories" (ones with a relatively high percentage of positive examples). And yet, SL-P-S-E {8} is the closest of the 12 systems to the AVERAGE algorithm analyzed by Servedio [Servedio 1999]. We therefore need to analyze SL-P-S-E {8} in more detail. This system is supervised, so it selects and uses all training examples. All training example labels are available to the learner, and so there are no unlabeled training examples. It is an eager learner, so all selected and used examples are actually learned from. In other words, each and every training example is used to adjust weights. Basically, what is happening is that the learner, in learning from all examples, ≈95% of which are negative, quickly learns to say NO, and there are not enough positive examples to "unlearn" this. Consider the following thought experiment. One has a situation where the data is in fact linearly separable. The 2 clouds do not overlap, but are quite close together. Most training examples are negative, and there is a large number of irrelevant attributes. By extreme good luck, one starts with the hyperplane between the 2 clouds (but of course the learner has no way of knowing this). The perceptron learns from 950 negative examples and 50 positive ones. Regardless of the order in which the examples are learned from, it is very unlikely that the effects of the 950 negative examples ⇒ 950 demotions (weight decreases) will be undone by the effects of the 50 positive examples ⇒ 50 promotions (weight increases). Recall that for the perceptron each promotion/demotion is by the same amount (α). Thus the learner most likely will end up in a state where future predictions result in it just saying NO.

When the problems with SL-P-S-E {8} arose, we did try a very wide variety of learning rate values to see if the problem could be solved by modifying δ. The only values of δ that resulted in any YES predictions were so small that we had loss of floating point precision problems. And even then, we were getting values of $F_{1.0}$ on the order of 0.0008. Far from acceptable.

Does this conflict with Servedio's analyses? No. We violated the conditions he established for his algorithm to work, and the violations were too severe to be overcome by a good algorithm. There are some significant differences between our situation and the one analyzed by Servedio. Recall that we knew this going into this experiment, but hoped we would still be able to obtain some beneficial results. Actually, we have obtained several good results, so the fact that this one system did not perform as hoped is a minor setback. Servedio sets several preconditions for his analysis. We violate most of them to some degree or another, but the main ones we violate are:

[1] the example space is origin-centered

[2] no attribute noise

[3] monotonic class noise, which means that the probability of an example being labeled incorrectly by the teacher decreases monotonically with the distance between the example and the concept hyperplane; this noise model was developed by Bylander [Bylander 1998] and generalizes certain other classification noise models (such as the noise-free and random classification noise models)

[4] the examples are uniformly distributed over the surface of an n dimensional sphere

Servedio's analysis decomposes each example vector into a component in the direction of the target concept vector and a component perpendicular to the target concept vector. The above preconditions mean that, on the average, the perpendicular components of the example vectors cancel out, leaving the average vector aligned or at least almost aligned with the target concept vector. Thus it would seem that the normal definition of "uniform distribution" can be relaxed somewhat – one needs to have the distribution uniformly banded with the bands oriented perpendicular to the direction of the target concept vector. Think of the earth with the lines of latitude being the bands and the direction of the target concept vector being through the poles. As long as the distribution of examples is uniform along a particular latitude, the perpendicular components will on the average cancel out. Each latitude can have a different uniform distribution mean than the other latitudes, but how much variation there is in the distribution mean as one progresses from one latitude to the next will affect how difficult it will be (and how many examples will be required) in order to get the components perpendicular to the target concept vector to almost cancel out, and the size of the bound on "almost". The normal (more strict) definition of uniform distribution would then be the case where there is only one band.

One would always like to find a theory whose preconditions exactly matched the problem being solved, but often one instead finds a theory that "maybe almost" matches and one needs then to analyze whether it is reasonable that it will still provide at least some benefit and so is worth the effort of implementing and testing. Let us return to the 3 preconditions of Servedio and discuss how they compare to our situation.

[1] Our document space is definitely not origin-centered, but the value of θ in $\vec{w} \cdot \vec{x} = \theta$ is how we take care of this. That is, a translation of coordinate systems will map between our coordinate system and an origin-centered one. Translation is a rigid-body transformation [Hearn and Baker 1997] and

so does not change the magnitude or direction of vectors. To see the role of θ, consider the hyperplane defined by $\vec{w} \cdot \vec{x} = 0$. This hyperplane passes through the origin since $\vec{x} = \vec{0}$ satisfies the equation for the hyperplane. Now translate each x_i to x_i' where $x_i' = x_i + d_i$. That is, we shift all points $\vec{x}$ to $\vec{x'}$ by adding a constant $\vec{d}$. Therefore $\vec{x'} = \vec{x} + \vec{d} \;\Rightarrow\; \vec{x} = \vec{x'} - \vec{d}$. Therefore $\vec{w} \cdot \vec{x} = 0 \;\Rightarrow\; \vec{w} \cdot (\vec{x'} - \vec{d}) = 0$. Since dot product is distributive [Lay 1993], we get $\vec{w} \cdot \vec{x'} = \vec{w} \cdot \vec{d}$. We refer to $\vec{w} \cdot \vec{d}$ as θ. In our case, the translated origin is usually in the first quadrant, so the d_i happen to be positive.

[2] We know we definitely have attribute noise. However, since other learning methods, especially the "varying δ perceptron" system discussed in section 6.5 ("Comparison of Winnow and Perceptron"), do quite well in this domain (albeit not very efficiently), we can assume that the attribute noise usually does not prevent learning a concept that is a close approximation to the target concept.

[3] We know we have class noise, and suspect that it is not very well behaved – in the sense of being uniformly random or varying in any consistent manner as a function only of the distance between the examples and the target concept hyperplane. However, it appears (for the same reasons as for attribute noise) that the class noise is not of such a form as to prevent learning a good approximation to the target concept.

[4] The examples in our domain are not on an n dimensional sphere, but rather they occupy the vertices of an n dimensional boolean cube. It seems conceptually that the results of Servedio should still apply, although the discreteness of the cube might make the "perpendicular components cancelling out" a bit more problematic in terms of obtaining an actual bound on the size of the (net) perpendicular component of the average vector (which Servedio does for the n dimensional sphere case). However,

while the basic approach still applies, we have no reason to suspect that the distribution of documents in typical corpora are even approximately banded uniform with respect to the direction of the target concept vector. This is probably the main precondition whose degree of violation we could not, based on the results of other experiments, argue was not being violated too badly. It is for this reason that we added code to attempt to measure the degree of this particular violation, so that we might be better able to explain any poor performance that was encountered and perhaps lend support to some other ideas that we wanted to investigate.

Next we discuss how we attempted to test the degree to which our domain violates the banded uniform distribution precondition. We are dealing with a very high dimensional and very sparse space, so most standard statistical tests for goodness of fit for discrete multivariate data (such as Pearson's χ^2 and the Loglikelihood Ratio Statistic G^2) [Read and Cressie 1988] are not computationally feasible. And besides, we wanted to test for a relaxed version of the definition of uniform and allow for banded uniformity. We are mainly interested in the directions of certain vectors in the document space, so we chose an efficient method – the cosine measure of similarity that is often used in the vector space model for document representation. Recall that the vector space model measures similarity between two documents using the cosine of the angle between their vectors. Since $cos(0°) = 1$ and $cos(90°) = 0$, vectors that are collinear or nearly so will have a high cosine similarity measure, whereas those that have an angle between them that is close to 90° will be very dissimilar. The dot product of two vectors $\vec{a} \cdot \vec{b}$ is defined as $|\vec{a}|\,|\vec{b}|\cos\theta$ when $\vec{a} \neq \vec{0}$ and $\vec{b} \neq \vec{0}$, and 0 otherwise [Kreyszig 1962]. Therefore we can compute the cosine similarity measure (hereafter "CSM") as $\dfrac{\vec{a} \cdot \vec{b}}{|\vec{a}|\,|\vec{b}|}$ when $\vec{a} \neq \vec{0}$ and $\vec{b} \neq \vec{0}$, and 0 otherwise [Salton and McGill 1983].

We are interested in how the directions of the following pairs of vectors compare using the CSM – that is, we will examine 5 different types of CSM values. All vectors are averages. Also, recall that a training example is "used" when the system requests its actual label:

1. all training examples versus the entire document space (CSM_1)
2. training examples used versus the entire document space (CSM_2)
3. training examples used versus all training examples (CSM_3)
4. (by member) learned weights versus all training examples (CSM_{all})
5. (by member) learned weights versus training examples used (CSM_{used})

Note that [1]we do not know how to compute the actual target concept vector and can not even efficiently compute the best approximation to it, so we are approximating it with the learned weights; we believe that this approximation is a reasonable one if the learner is accurate, [2]the CSM measures CSM_1, CSM_2, and CSM_3 are always ≥ 0 in our domain, [3]CSM_{all} and CSM_{used} are > 0 for learners having all learned weights > 0 [such as winnow], and [4]CSM_{all} and CSM_{used} can be positive or negative or 0 for learners that allow positive, negative, or 0 values for learned weights [such as perceptron].

Typical values of CSM_1 (all training examples versus the entire document space) depend on the split, but typically are in the range [0.11,0.16] for titles only, and smaller ([0.10,0.11]) for full text. This is as one would expect. Given a dictionary of tokens, most combinations of them will not result in meaningful documents, and given that language has many syntax rules and non-uniform use of words, one would not expect actual documents to be uniformly distributed with respect to the entire document space.

The remaining CSM values vary by system and trial. Note that the remaining types of CSM only *vary* between categories for active learning systems that are skeptical, since both supervised learners and optimistic learners use the (attribute portion) of all training examples. For skeptical active learning systems, we compared these remaining CSM measures to various contingency table based performance measures using the Spearman Rank Correlation Coefficient (r_s) [Snedecor and Cochran 1989]. We focus on comparisons for the "very good" systems (6, 1, 7, 2), since it is those that we want to investigate and emulate. We next briefly examine these CSM results.

1. The values of CSM_2 (training examples used versus the entire document space) were generally small [0.11,0.18], as one might expect. For AL-W-S-E {6}, CSM_2 and accuracy correlate positively at the 0.05 level. We also observed that CSM_2 for AL-W-S-E {6} was consistently greater than for the second place system SL-W-S-M {1} (which is supervised) by ≈20%.

2. The values of CSM_3 (training examples used versus all training examples) were generally large [0.6,1.0], as one might expect since the training examples used are drawn from the pool of all training examples. Note that for supervised and/or optimistic learners, CSM_3 is 1.0. For AL-W-S-E {6}, CSM_3 and accuracy correlate negatively at < 0.001 level.

3. CSM_{all} (learned weights versus all training documents) and CSM_{used} (learned weights versus training documents used) are usually in the range [-0.3,0.3]. We did not find any interesting patterns in their values per se, but how they compared within systems was interesting. For those systems for which these measures can vary, the comparison of CSM_{all} and CSM_{used} was generally mixed within each trial, with some members having $CSM_{all} < CSM_{used}$ and some having $CSM_{all} > CSM_{used}$. Notable exceptions were AL-W-S-E {6}, which almost always had all members having $CSM_{all} < CSM_{used}$, and AL-P-S-E {12}, which almost always had all members having $CSM_{all} > CSM_{used}$.

Note that AL-W-S-E {6}, which achieved the highest $F_{1.0}$ scores, also had interesting results in certain comparisons of CSM values with contingency table based performance measures and also in certain comparisons of CSM values themselves. The fact that the CSM between the training examples used and all training examples correlates negatively with accuracy indicates that the learner is more accurate when it chooses training examples that are not distributed in the same manner as all training examples. Does this makes sense, given that the main difference between active and supervised learning is in the training examples that are used? What does one expect? Probably one expects that in active learning one will generally find $CSM_{all} < CSM_{used}$ since the active learner would presumably learn a concept closer to the training examples used than to all training examples. However, one might also think that the actual performance of a very good active learner will occur when $CSM_{all} \geq CSM_{used}$. This would indicate that, in spite of using only some training examples, the learned concept was actually more representative of all training examples. However, $CSM_{all} < CSM_{used}$ was consistently true only for AL-W-S-E {6}. These results suggest that very good active learners actually do better if the concept they learn is consistently more representative of the training examples used rather than of all training examples. In other words, it would appear that the purpose of active learning is *not* to somehow select a few training examples that are representative of all training examples, but rather to somehow select a few training examples that are representative of the target concept. AL-W-S-E {6} is choosing examples in a significantly different manner than if it chose examples randomly or in a supervised manner, and the method used for choosing examples is providing benefit (in the form of high accuracy). The more the learner is able to select very good examples, the better the accuracy becomes and the less the training examples used are like the typical training example.

One interesting hypothesis that we did not find support for is the idea that active learning could in some sense make a non-uniform training example distribution more uniform (thus allowing one to use algorithms specifically designed for uniform distributions after active selection of good examples). At least in this domain, while the filtering done by active learning does result in training examples used being more similar to the document space as a whole than all training examples, both are still quite dissimilar to the document space. Typical CSM values for the former are in [0.11,0.16] and for the later are in [0.11,0.18]. Perhaps in other domains (in particular, those with a higher ratio of number of training examples to number of dimensions), this effect might be more pronounced and so be explicitly used beneficially.

A few final comments on this experiment. While SL-W-S-E {2} did well (it placed 4^{th} in $F_{1.0}$ score), it is very difficult to tune its learning rate so as to avoid exponent overflow. Winnow learns using multiplicative updating of the weights, and SL-W-S-E {2} is using ("SL") and learning from ("E") all 21,000 examples. Consider for example a token that appears in a large number of positive examples. And say the weight for that token was initialized to 1.0. Each time that token is encountered in a positive example (including the effects of multiple epochs), that weight will be made larger ($w_i \rightarrow w_i \times \alpha$). Typical values of α that we use for supervised winnow learners are ≈1.04, but $1.04^{2000} \approx 10^{34}$, which is about where exponent overflow occurs on our system (for single precision floating point). Thus the use of SL-W-S-E {2} requires using a much smaller δ and/or fewer epochs, thus making it more problematic to compare it to other systems with respect to our original promise of using the same number of epochs for all systems in each experiment.

And finally, we discuss which system was "the best". It appears that AL-W-S-E {6} is best. It has the highest $F_{1.0}$ score (0.6770), ties for highest accuracy (0.971), and yet is an active learner using relatively few examples (4.2%). It runs quite fast (12.0 in a range of [10.6,923.3]), and in fact only one other system was faster. Also AL-W-S-E {6} obtains its high $F_{1.0}$ by having both good precision (0.718) and good recall (0.662).

It would appear from this experiment that in our domain (text categorization with minimal preprocessing), active learning can perform better than supervised learning and can do so using significantly fewer examples and running faster. Results of this experiment reinforce our earlier conclusion, discussed in section 6.5 ("Comparison of Winnow and Perceptron"), that winnow performs better than perceptron. This experiment also indicates that, in our domain, optimistic learning is not beneficial, but eager learning (even for learning algorithms that are normally mistake-driven) does provide benefit.

6.10 Committee of Incremental Naive Bayes Learners

First we will discuss some of the aspects of the standard naive Bayes algorithm that make it different from winnow and perceptron, especially as regards its use in ALC. These comments apply to many other existing supervised algorithms as well. In the case of winnow and perceptron, the hyperplane is initialized to some random location and then, as individual examples are used, the hyperplane shifts location, where the amount and direction of the shift is determined by the current state of the learner (current position of the hyperplane) and the characteristics of the one training example currently being processed. Since the initial location of the hyperplane is random, one can generate a committee of different learners by initializing each member to a different hyperplane location, in the manner we discussed earlier. However, in naive Bayes, there is no ready-made random element involved in determining the location of the hyperplane. The naive Bayes algorithm does not initialize a hyperplane and then relocate it while learning. Even if one uses the naive Bayes algorithm in an incremental fashion, the algorithm is each time computing the hyperplane location based on probabilities computed from all training examples used to date. It is not in any sense starting with a current location of the hyperplane and shifting it in response to information gleaned from the one training example currently being examined.

Another difference to note as to how the naive Bayes learning algorithm differs structurally from the winnow and perceptron learning algorithms is in the use of multiple epochs. For both the winnow and perceptron learning algorithms, one specifies a learning rate and may choose to have the algorithm make multiple passes through the training data. As long as some of the training examples are incorrectly predicted, these learners will continue to learn in subsequent passes through the data. However, observe from the steps involved in the naive Bayes learning algorithm (Figure 8 on page 83) that the standard algorithm obtains no benefit from making multiple complete passes through all of the training data. After the algorithm has seen all of the training data once and has determined the various totals and counts, additional complete passes will not change the probabilities that the algorithm has learned.

We decided that whatever mechanism was used to introduce randomness into the members of a committee of naive Bayes learners, it needed to be supportable and not contrived. One can always introduce variation by, at the end of the prediction phase, adding a postprocessing step that, with a certain probability, modifies the prediction. However, our intuition is that the method that one uses to introduce variation in the learners must be a bit more substantive, in that one would prefer to have a committee of learners each of whose hypothesis is defensible in its own right. Indeed, it appears that a major contributor to the robustness of the winnow ALC algorithms we have discussed so far is that any one of the members of the committee is in its own right designed to be a good and independent learning algorithm.

Recall also that one of the reasons committees are thought to be a good idea is that the errors they make are independent.

The naive Bayes algorithm did quite well in supervised learning in the above experiments, so we decided to try to develop a naive Bayes active learning algorithm. Our goal was to form a committee of naive Bayes learners that could be used for active learning. Recall that the standard naive Bayes algorithms used so far have been supervised (and also eager). In order to form a committee whose differences in individual predictions can be used to determine which candidate training examples are the most informative, one needs to somehow make the learners on the committee represent different hypotheses, at least initially. We chose to use the previously-discussed m-estimate parameter method for computing probabilities as a way to introduce variation into incremental naive Bayes learners and thereby have a committee of learners that each represents a different hypothesis. Referring back to Figure 8 on page 83, normally the counts $c_{i,k}$ and totals t_k are initialized to 0 prior to step #1. To introduce variation amongst the committee members, we instead generate random values for the two parameters $\hat{m}$ and $\hat{p}$, and then initialize the $c_{i,k}$ to $\hat{m} \times \hat{p}$ (instead of 0) and initialize the t_k to $\hat{m}$ (instead of 0). This introduces variation into the committee because later (in steps #2 and #3) when conditional and prior probabilities are computed, they are computed from the $c_{i,k}$ and t_k. Even though each member has (for the eager case) learned from exactly the same training examples, the computed probabilities will differ because of the different initializations of $c_{i,k}$ and t_k.

Initial results on the Reuters-22173 titles only corpus looked very promising, but what one good split giveth ($F_{1.0}$ = 0.778), multiple splits taketh away ($F_{1.0}$ = 0.696). The method *does* give improved performance if one allows it to use more training examples, and certainly even $F_{1.0}$ = 0.696 is an excellent score. Figure 49 gives learning trace results using this algorithm.

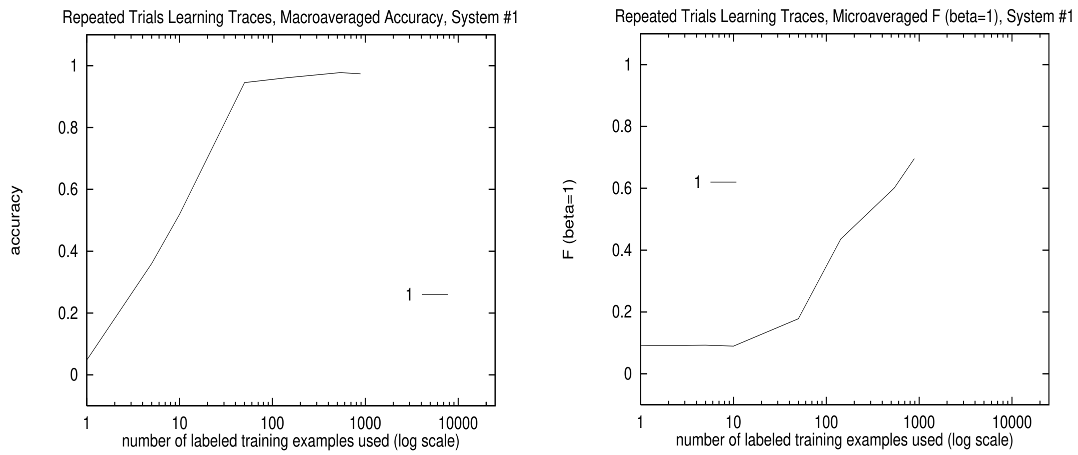

Figure 49. Learning Traces: Accuracy (left) and $F_{1.0}$ (right)

Final performance measures are: accuracy = 0.974, $F_{1.0}$ = 0.696, training examples used = 4.2%, precision = 0.778, recall = 0.629. However, average time was 5994.5. The algorithm seems to work well in this domain in terms of contingency table based performance measures and percentage of training examples used, but its elapsed run time is high and so its use is hard to justify in our domain.

To compute the computational complexity of the committee of naive Bayes learners, we proceed as follows. The time required is the time spent deciding whether or not to see the label plus the time spent learning plus the time spent predicting. Using the AllMargin method, complexity for deciding whether or not to see the label, for one trial, will be (ref. Table 3 on page 125) $O(dO_p) = O(dt)$. In the committee of incremental naive Bayes learners algorithm, learning is incremental and occurs after every example that is selected for training. Consider the time spent learning from one example (ref. Figure 8 on page 83). We increment counts corresponding to $x_i = 1$ and we increment (once) the corresponding t_k (step #1). This gives us $O(t_a + 1)$ for this one example. Next, we perform step #2, which computes 4 conditional probabilities for each attribute and so we have $O(4t)$.

Step #3 computes 2 values and so is $O(2)$. Thus time complexity for learning from one example is $O(t_a + 1 + 4t + 2) = O(t_a + 4t)$. Therefore, for one trial (and making the standard worst-case assumption, that all examples are used for learning) we get $O(dt_a + 4dt)$. Prediction (ref. Table 3) is $O(t)$ for one example, and so is $O(dt)$ for one trial. Therefore, total complexity for one trial is $O(dt + dt_a + d4t + dt)$ = $O(dt_a + d6t) = O(d(t_a + 6t))$.

Recall (Equation (10) on page #124) that the computational complexity for the standard naive Bayes algorithm is $O(d(t_a + t))$. In our experiments, $t \gg t_a$, so one could further simplify both this expression and the above committee expression to $O(dt)$. One might therefore expect that the committee of incremental naive Bayes learners will perform about as fast as the standard naive Bayes learning algorithm. Empirical results however are that the committee of incremental naive Bayes learners algorithm is much slower than the standard naive Bayes algorithm, at least for the domain in which we are dealing. This is because the learning that the standard naive Bayes algorithm does once (mainly the computation of the 4 $p(x_i = j|k)$ values) is being done by each committee member for each example used. The computational complexity analyses are correct, but what is happening is that the constant of proportionality for the committee version of the algorithm is very large, and for the input sizes we are dealing with, this is a significant factor.

7. Other Experiments and Ideas

In this chapter we discuss other areas that we investigated in our research. Some of these methods did give improvements at the time, but we later found more general ALC methods (presented above) which were simpler and which gave equal or better performance. Some of the methods discussed gave interesting but disappointing results, but possibly provide good starting points for future work. We report on these experiments here so that others wanting to investigate these areas may benefit from our experiences. Some of these are existing approaches that we adapted to our domain. Others are we believe new ideas that may even work well in their current form in other domains.

7.1 Basting

With a large number of dimensions, the initial location of the learner's hyperplane may be a very long distance from the optimal location. An initial solution to this problem was to allow what we call "basting". Basting refers to repeatedly applying the learning algorithm to the entire committee for the training example currently being used until some specified condition is met. The distinction here is that one does not repeatedly apply the algorithm to one member and then to the next, etc., but rather one basting pass involves applying the example once to each of the committee members. A variety of basting termination conditions were examined. We also allowed one to baste on positive examples only, negative only, or both.

The name originally came from the notion of basting being related to ***ba***gging and boo***sting***. In basting, one has to make the decision whether or not to baste with that example at the time that example is being used for learning. The fact that baste means both "to coat with liquid periodically while cooking" and also "to thrash" (as in grain ... as in winnow ...) seemed psychologically to confirm that this was a good approach to at least investigate further.

There is support for this type of action in the literature. Of course, a great many researchers employ multiple epochs in their machine learning algorithms, especially when using artificial neural networks [Mitchell 1997]. As regards deciding whether or not to weight positive examples and/or negative examples in the current epoch, one version of the Rocchio algorithm for incorporating relevance feedback has parameters for weighting the terms obtained from positive and negative documents [Rocchio 1971, Buckley et al. 1994], and weighting of documents according to a confidence score is done in [Yang et al. 1998]. Researchers reporting specific numbers seem to prefer weighting positive over negative examples and in ratios typically in the 2:1 to 4:1 range. While we did not normally employ basting decision methods that simply basted with each example for a specified number of times, on the average we usually had numbers of basting passes in the 2 - 5 range.

We obtained good results using basting, but other methods developed later produced slightly better results and seemed less contrived. With basting, one must specify which kinds of examples to baste (positive, negative, or both) and specify a decision method to be used to determine when to stop basting with each qualifying example. In defense of basting, it runs very fast (since one is in a sense folding the effects of many epochs into one or at least very few epochs), and typical $F_{1.0}$ scores were about 0.03 - 0.05 below those for the methods described in this thesis. Since the methods we are using are already quite fast, we decided that the time gained by using basting was not worth the lower performance scores and the fact that one had additional parameters to tune. It appears that the slightly lower scores are due to the fact that, in basting, the decision as to whether or not to reuse an example must be made when that training example is being learned from. When one instead utilizes multiple epochs, it allows the learning algorithm to make a more informed decision about each potential reuse of an example, since a learner with more generalized experience is making the decision.

7.2 Two Phase Learning

One difficulty we had for a large portion of the early part of the project was introducing diversity into the committee so that the diversity was not contrived and so that individual members would disagree in their predictions sometimes (and at the right times) but not disagree too often. This problem worsened as we progressed into tests involving ever larger numbers of dimensions. If members disagree often, then one can end up with essentially a committee of supervised learners; their frequent disagreement results in all candidate training examples looking informative and so being used. The opposite can also occur (and this was our usual problem); the committee members can agree almost all of the time, and so few if any candidate training examples appear to the committee to be informative, and so few examples are used. This would be fine if in fact the individual committee predictions were correct, as it would mean that the committee had learned a good approximation to the target concept. However, what was usually the case in these situations was the learners had learned an extremely poor approximation to the target concept but did not have any way of realizing it and correcting the situation.

Note that this problem is caused by the fact that we use absolutely no information about category label distribution when initializing the learners. Thus the learners are on the average initialized to the central region of the document space. However, since all of the categories of interest (in both Reuters-22173 and Reuters-21578) have less than 20% positive examples, and since all but 2 categories have less than 4% positive examples, the initialized state of the learners is on the average in the "just say NO" region. One solution therefore would be to have the user enter an estimated percentage of positive examples and initialize accordingly. We thought that this approach would place an inappropriate burden on the user, and also would result in one more knob to tweak.

We decided to try what we call "2 phase learning". This idea was obtained from [Freund et al. 1997]. In their proof that QBC is efficient for the hyperplane concept class when the prior distribution and the distribution of examples are both approximately uniform, a starting vector with certain characteristics is needed. This is obtained by using an initial supervised learning phase.

We therefore implemented and tested 2 phase learning, where the first phase is supervised and the second phase is active. The basic idea is that one uses a small amount of supervised learning to get the individual committee members to be moderately accurate, so that their subsequent individual predictions which are used to determine the informativeness of future candidate training examples in active learning mode are somewhat based on the category of interest rather than on their random initial state.

The method works reasonably well. One unfortunately does have to tell the method how to end the supervised phase, for example by providing a number of positive examples after which it will then switch into active learning mode. So this method did result in adding a parameter to adjust. Typical run times are quite fast, it being only slightly slower than basting. It gave similar results to basting in that $F_{1.0}$ was about 0.03 - 0.05 below scores for the methods described earlier in this thesis. It did tend to use more examples than the methods we now use, which is not surprising since it uses supervised learning for the initial portion of the learning process.

7.3 Comparison to the Abe and Mamitsuka Experiments

Abe and Mamitsuka developed committee-based learning methods based on QBC, bagging, and boosting, and they used these methods in several experiments [Abe and Mamitsuka 1998]. Each committee member was a C4.5 learner. A focus of their research was to examine different structured ways of introducing randomness into the committee, since C4.5 (like standard naive Bayes) is deterministic. They developed a QBC-based method (which we will refer to as QBC-AM) and also methods based on bagging (QBag, query by bagging) and on boosting (QBoost, query by boosting). They then ran several experiments using 8 databases from the UCI Repository of Machine Learning Databases [Blake et al. 1999]. These databases were of moderate size, both in numbers of examples (≈ 300 - 1000) and in numbers of attributes (≈ 5 - 40). The majority class usually comprised ≈ 55 - 65% of the examples. Having a large majority class is good since one has larger numbers of positive examples, and most learning algorithms do better learning from positive examples than from negative ones. Having a large majority class, however, also means that a learner has to actually learn a reasonable approximation to the target concept, in that "just saying no" will not result in a very high accuracy score. QBC-AM was mainly used by them as a basis for QBag and QBoost because QBC-AM does not itself introduce variation into the learners and so can not be used successfully for active learning when the learner itself, such as is the case for C4.5, does not have a randomizing component. Their main experiments involved QBag and QBoost, and they reported reductions in the number of training examples used of 25 - 50%. We decided to use their methods with different learning algorithms (winnow and perceptron) and to investigate whether their methods would work in our domain. Since our method of initializing winnow and perceptron involves a randomizing element, we would also be able to experiment with QBC-AM itself as well as with QBag and QBoost. We have larger numbers of examples and much larger numbers of attributes, especially since we are doing minimal preprocessing. Thus whether or not the methods would scale up was a concern. Also, our training data usually has less than 4% positive examples. Their experiments dealt with moderate-sized databases, so the fact that these methods are of higher computational complexity than $O(n)$ was not a major concern to them in their research. We realized that this higher time complexity might present practical problems in our domain. But we were interested in seeing if those problems would in fact arise and if they did, whether or not we could solve them, the hope being to still obtain benefit from the methods.

For each of their methods (QBC-AM, QBag, and QBoost), we will first give a description of the method, including a brief discussion as to its computational complexity, and then we will describe our results when using the method in our domain. We will describe the method as we adapted it to the winnow and perceptron learners and to our overall approach. We will strive to use the same or at least similar notation to what is used in [Abe and Mamitsuka 1998], but since we are summarizing our adaptations of their algorithms, there may be some notational differences. Also note that since the learning algorithms we are using are different, there are some differences or at least some additional information that we need to provide since we must be clear about if and when we are reinitializing our learners. Another difference is the distinction we make between an example being selected (system judges it to be informative) and used (system actually requests its label). Since examples being used is one of our main performance measures, we do not count an example as having been used until the label is actually requested by the learners. As a practical matter, this difference is slight in terms of the counts obtained, but does affect the terminology we use in describing the methods. We end this section with some conclusions based on our overall experiences with these methods.

All 3 methods are methods for selecting examples for members of a committee of learners. QBag and QBoost do this in a manner that makes it unnecessary for the underlying learning algorithm to itself have a random nature to it. Recall that due to the way we initialize the winnow and perceptron learners, the individual members of the committee represent different hypotheses. Such a mechanism, however, does not readily exist for many learning algorithms, so Abe and Mamitsuka's research had as one goal the development of methods that could be used in conjunction with such learning algorithms in order to obtain the ability to have the individual learners differ. Their QBag and QBoost methods do this by having the different learners on the committee use different examples, thereby making the hypotheses that each learns different. The successive methods are in a sense increasingly successful in making the individual learners better, but this improvement comes at an increasing computational cost. One aspect of the methods we therefore will need to evaluate is if, for our domain, this increased cost is offset by some other benefit.

As we will see (and as is detailed in [Abe and Mamitsuka 1998]), each method builds on the preceding one. We will ourselves rely on this fact when describing each successive method.

Let S be a set of training examples, initialized to contain one randomly chosen example. Then all of the methods follow this basic pattern *for each epoch*: learning occurs using some examples from S and then one additional training example is selected as being "very informative" and is put into S. The various methods differ mainly in how the learning is done and which examples in S are actually used for that learning. Note that one training example is added to S during each epoch, and that it is added to S after the learning for that epoch has been completed (and thus it can not be used for learning during that epoch). Since these methods select only one additional training example during each epoch, trials typically involve using large numbers of epochs (we use 300 - 1,500).

A reminder on the definitions we used in analyzing $O(\,)$ behavior:

c = the number of members in the committee
e = the number of epochs
d = number of documents in the corpus
t = number of tokens in the dictionary
 = length of the attribute portion of the feature vector
t_a = average number of unique tokens in each document

7.3.1 QBC-AM

The QBC-AM method differs from our QBC-based method QBC-REP in that QBC-REP decides whether or not to use each training example at the time that one example is being examined by the system. In other words, with QBC-REP one training example at a time is offered to the system, and the system decides whether or not to use it. QBC-AM, on the other hand, uses a more sophisticated procedure to determine which training examples to actually use. A set of labeled training examples, called S, is constructed. The set S consists of training examples that have in previous epochs been selected as informative. Each epoch results in the use of one example by all members for learning and also in one more (different) labeled example being added to S. The main functions of QBC-AM are to incrementally educate a committee of learners and to also decide which one candidate training example to add to S during the current epoch. Let T = the number of committee members and i be the epoch number. We will use S_i to denote the contents of S at the beginning of epoch i. The initialization of the learners is done in the normal manner (hyperplanes randomly perturbed about w_{ave}, ref. page 142), and S_1 is initialized to contain one randomly selected labeled training example. Let us consider what happens in QBC-AM during a typical epoch. At the end of epoch $i-1$ (as we shall see), one more labeled training example was added to S. Call this example e_{i-1} since it is the example added at the end of epoch $i-1$. Epoch i then begins with S containing i labeled training examples. The first thing to occur in epoch i is that each of the T learners uses example e_{i-1} for learning. This completes the learning for this epoch. Next we decide which one training example out of all of those that are not in S would be best to add to S – that is, we choose e_i. We do this as follows. We choose R training examples that are not in S. R is a parameter of QBC-AM. The R training examples are chosen at random using a uniform distribution without replacement. Once this group of R examples has been chosen, each example is given to the committee for prediction by its members and the example for which the margin magnitude is smallest is selected as e_i and is added to S. If several of the R examples have the smallest margin magnitude value, then one of those examples is randomly chosen as e_i. This completes the processing of epoch i.

QBC-AM has itself not introduced randomness into the individual committee members since all members have used the same examples. Thus only if the learner itself has a randomizing element will the resulting committee members represent different hypotheses. However, what QBC-AM has done, regardless of the type of learner used, is to have all learners use examples that are informative. This should make each member of the committee quite accurate. The final committee (i.e., the one used for predicting on previously unseen examples) is the committee that results from the last epoch. Prediction is by majority vote.

An item to note. A training example is selected during each epoch (and used in the next epoch), so the structure of the QBC-AM algorithm has solved the problem of the committee members always agreeing and so not asking to see any labels. One is guaranteed that one additional training example will be used during each epoch and that, in the judgement of the learners, it is a relatively informative one.

The computational complexity of the method largely depends on whether or not the value of R is considered to be some function of the input size. One would often like to dismiss such parameters as "constants" so that one can obtain a lower ordered $O(\,)$ expression, but obviously this is illegal since we want $O(\,)$ to reflect computational complexity for any changes in input size. Since the purpose of R is to allow the method to select its next training example from a group of several representative examples, it seems reasonable to conclude that R will probably increase in some fashion as the input size increases, and so should not be treated as a constant. In the experiments performed in [Abe and Mamitsuka 1998], R = 100 was used, which is quite large considering the databases contained only 300 - 1000 examples and 5 - 40 attributes. In our experiments, we used values for R ranging from 100 to 400. We tried larger values but did not find much benefit from using the larger values. Since we do not know the functional form of R, and in fact it may well depend on the domain, we will simply include it explicitly in our expression for $O()$. Because of the existence of the inner loop which has the committee predicting on R candidate training examples in each epoch in order to pick a most informative one, this algorithm's behavior will likely be worse than for QBC-REP or AllMargin.

We determine $O()$ for QBC-AM as follows. During each epoch, in QBC-AM the committee uses one example for learning and then predicts on R examples. We typically use the winnow learner, so for one epoch we have $O(ct_a + cRO_p())$ = $O(ct_a + cRt_a)$. For an entire trial of e epochs we have $O(ect_a + ecRt_a)$ = $O(ect_a(1 + R))$. Usually $R \gg 1$, so we have $O(ect_a R)$. Recall that c is a constant. In the case of QBC-AM and in our domain, since e is the number of training examples used, as a practical matter it is not a constant, but rather is likely some increasing function of t_a and probably also of d. Similarly, R (as discussed above) is some increasing function of d and probably also of t_a. That is, since input size is $O(dt_a)$, each of e and R are some increasing function of the input size. We therefore end up with the computational time complexity of QBC-AM when using the winnow (or perceptron) learner as being $O(eRt_a)$. Thus $O()$ for QBC-AM is worse than linear in the input size.

It is very easy to specify the number of training examples used with this method, as it is simply the number of epochs. Our tests were run with 1500 epochs and thus used 7.14% of the training examples. $F_{1.0}$ scores for QBC-AM were about 0.01 - 0.02 below those obtained by AllMargin, and QBC-AM used more examples.

7.3.2 QBag

The QBag method is similar structurally to QBC-AM except that at the beginning of each epoch i, each learner is completely reinitialized and then chooses $i = |S|$ examples from S to use for learning. Recall that each learner in QBC-AM only used 1 example for learning, and it was always e_{i-1} (so no choice of training examples to be used was at that point involved). In QBag, the choice of i examples is made randomly by each learner using a uniform distribution, with replacement. Thus the computational complexity of QBag is higher than for QBC-AM, because now the learning done by each learner at the beginning of each epoch is done with i examples, whereas in QBC-AM we carried forward the previous epoch's learners and each learner incrementally used only one new example in that epoch. Thus in QBC-AM, the value gleaned from the processing effort as successive epochs are completed is reflected mainly in a presumably increasingly accurate committee of learners. In QBag, this epoch to epoch transfer of the value of prior processing effort is only in the set S. QBag completely retrains the learners during each epoch, which is expensive in terms of time but does introduce differences amongst the learners (they each learn on different examples). It also presumably results in learners that are more accurate, since the learners are being totally retrained on the most current set S (whereas the QBC-AM learners are each only allowed to use one additional example per epoch and so carry with them from epoch to epoch ever increasing amounts of old baggage).

QBag has thus introduced randomness into the individual committee members by having each member (on the average) learn from different examples. All of the examples used are informative, so this should make each member of the committee quite accurate and also perhaps (as compared to random perturbation) result in learners who although different from one another are all reasonably accurate. The final committee (i.e., the one used for predicting on previously unseen examples) is the committee that results from the last epoch. Prediction is by majority vote.

It is important to note that in QBag there is a strong distinction between selecting an example (occurs when the training example is put into S) and using an example (occurs when one of the learners actually uses it for learning). In previous methods we have discussed (such as QBC-REP and AllMargin), selection of an example was immediately followed by its use. In QBag, however, it is possible for examples to be selected but not ever used since examples that are selected are put into S and then examples to be used are chosen from S using random sampling with replacement. As a practical matter, however, except for the training example selected during the last epoch, it was very unusual in the experiments we performed with QBag to have examples selected but never used. The degree to which this holds obviously depends on system parameters being used, such as the number of members in the committee. Note also that this same effect occurs in QBC-AM, in that an example e_{i-1} that is selected in epoch $i-1$ is not used until epoch i.

Using the same general approach as was used for QBC-AM, we compute $O()$ for QBag (using the winnow learner) as follows. During epoch i, the committee uses i examples for learning and then predicts on R examples. Thus for one epoch we have $O(ict_a + cRt_a)$. $\sum_{i=1}^{i=e} i = \frac{e(e+1)}{2}$ which varies as e^2, so for an entire trial we have $O(e^2ct_a + ecRt_a) = O(ct_a(e^2 + eR))$. For c constant, we therefore get $O(t_a(e^2 + eR))$. Recall that e and R are some increasing function of the input size. Thus $O(\,)$ for QBag is worse than linear and also worse than QBC-AM in computational complexity.

QBag is not very practical in our domain because of its time complexity. As with QBC-AM, the number of examples used is equal to the number of epochs. We used 1500 epochs (7.14% of the training examples). These trials lasted 15 - 100 times longer than trials using AllMargin and also used more examples. $F_{1.0}$ results were lower by 0.02 - 0.03 than for the AllMargin method.

7.3.3 QBoost (with AdaBoost)

QBoost further modifies the learning process of the committee members by using the AdaBoost algorithm [Freund and Schapire 1995] to control the learning process. We say "control" because AdaBoost is not itself a learning algorithm, but rather a method for deciding which training examples will be used by each of the individual members of a committee of learners and how to weight the resulting hypotheses for the committee prediction process. Recall that QBag, at the beginning of epoch i, retrains each learner completely by using i examples chosen from S. The choice of examples is made randomly from a uniform distribution, with replacement. QBoost, instead of directly training the committee members, invokes AdaBoost to oversee the training that occurs at the beginning of each epoch. The idea behind AdaBoost is that the choice of examples is still random, but not over a uniform distribution. Instead, AdaBoost adjusts the distribution over S so that examples which the previous learners (in the current epoch) have not learned very well are favored. In this way, subsequently trained committee members will provide a good complement to the earlier trained learners since they will be more likely to be trained on examples that the earlier learners did not learn. To see how this is accomplished, consider a typical invocation of AdaBoost by QBoost. Recall that there are T learners and that we are at the beginning of epoch

i. The first learner chooses i examples from S and uses them for learning. The choice of i examples is made randomly from a uniform distribution with replacement (the same as was done by QBag as the first step in each epoch). This learner is now examined on how well it has learned by comparing its predictions to the actual labels of the training examples in S. (Note that this comparison is made over all examples in S, not just on those examples in S that this particular learner used). For training examples that the learner did learn correctly (i.e., its prediction matches the actual label), the distribution over S is adjusted so that those examples are less likely to be chosen by the next learner. Now the second learner chooses i examples from S, the examples still being chosen randomly and with replacement. However, the distribution governing the random sampling is no longer uniform. Instead, it favors those examples that the first learner did not correctly learn. This process continues until all of the learners have learned. And thus each learner is on the average given examples to use for learning that the previous learners did not learn, or at least did not learn very well. Each learner is in a sense specializing in a different part of the total problem, and then the knowledge of all of the learners is combined via the committee voting mechanism. Once AdaBoost has trained all T members, it passes back to QBoost the trained committee and also a set of T voting weights $\mathcal{V}$ which contains one voting weight $\mathcal{V}_t$ for each member. Some learners maintain their hypotheses as sets of weights. These voting weights are in addition to any such knowledge base weights. The $\mathcal{V}_t$ are an inverse function of the error rate of learner t (as was measured by AdaBoost using S). Larger $\mathcal{V}_t$ indicate a more accurate learner (again, as measured using S). To form the committee prediction, individual members predict and then their predictions are weighted by their respective $\mathcal{V}_t$. The committee uses as its prediction the prediction that received the largest weighted vote. After the learning (controlled by AdaBoost), the actions taken in QBoost are the same as for QBag. QBoost chooses R training examples that are not in S (randomly chosen from a uniform distribution without replacement), the committee predicts on each of the R examples, and the example with the smallest margin magnitude (or one randomly chosen if there are several) is put into the set S.

Like QBag, QBoost has introduced randomness into the individual committee members by having each member (on the average) learn from different examples. QBoost (via AdaBoost) expends more effort in this regard, by making an attempt to have the members complement each other, and also by determining a relative weighting of each member's approximate accuracy. The final committee (i.e., the one used for predicting on previously unseen examples) is the committee that results from the last epoch. Prediction is as described above and is referred to as being "by weighted majority vote".

Using the same general approach as was used for QBC-AM and QBag, we compute $O(\,)$ for QBoost (again using the winnow learner) as follows. At the start of epoch i, AdaBoost is called and it trains the committee. AdaBoost does this by doing the following for each learner. It has the learner use i examples for learning. It then computes an error rate (for that learner) by having the learner predict on each of the i examples in S, and then uses that error rate to adjust the distribution over S. Admittedly the i examples used by the learner and the i examples used for computing the error rate are most likely not the same, but that does not matter. The time taken by/on behalf of each learner is therefore the time to learn plus the time to compute the predictions for the error rate computation and to perform the distribution adjustments, which is $O(it_a + iO_p(\,) + i) = O(it_a + it_a + i)$. There are c learners, so AdaBoost requires time $O(cit_a + cit_a + ci)$. For the remainder of epoch i, QBoost does the same thing as is done by QBag – it chooses R training examples and uses committee prediction to decide which one of them to add to S. Therefore, for epoch i, QBoost requires time $O(cit_a + cit_a + ci + cRt_a)$. As before, c is a constant. So we have $O(it_a + it_a + i + Rt_a) = O(t_a(2i + R) + i)$. The first term dominates, so we get $O(t_a(i + R))$. And finally, over all epochs, $\sum_{i=1}^{i=e} i$ varies as e^2, so for an entire trial we have $O(t_a(e^2 + eR))$. This perhaps surprisingly is the same time complexity that we computed for QBag, but obviously QBoost is doing more since it is also computing error rates for each learner and adjusting the distribution over S in response to those error rates. The reason for the two time complexities being the same is that the only difference between QBag and QBoost is in what is done during the first step. In the first step of QBag, the behavior for the learning, in one epoch and for one learner, is $O(it_a)$. The first step of QBoost performs learning and computation of error rate and adjustment of distribution, which is $O(it_a + it_a + i)$ which reduces to $O(it_a)$. The QBag and QBoost methods have the same computational complexity when using winnow (or perceptron) as the learner. We would expect, however, to see a larger constant of proportionality for QBoost.

Our results from using QBoost were even worse than we obtained when using QBag. QBoost values of $F_{1.0}$ were lower by about 0.03 - 0.06 than for AllMargin and used more examples. Run times were quite large, about 25 - 160 times those for AllMargin.

7.3.4 General Observations

Since execution time did in fact become a major concern, we also developed and tested a great many modifications to the QBC-AM to QBag to QBoost methods. While several were reasonably successful in terms of running faster and producing reasonable $F_{1.0}$ results in this domain, none of them outperformed the AllMargin method.

There are two rationales behind the series of modifications that lead from QBC-AM to QBag to QBoost. One is that, by spending successively larger amounts of effort, one could do a better job of selecting informative examples to use for learning, thereby allowing the individual learners to learn from fewer examples. The other is that by spending this additional effort, one could also obtain more beneficial variations amongst the committee members so that, rather than having each learner perhaps only being slightly different from the others, we would tend to have each one specifically specializing in particular aspects of the document space. As is the case with our QBC-REP and AllMargin methods, these 3 methods also select examples based on their degree of informativeness, where this is measured by how small the margin is when the committee members predict the label for the example. In other words, all of these methods assume that a high degree of disagreement amongst the learners is an indication that, at least insofar as the learners' current state, the example is very informative and therefore is a good one to select for learning. QBC-REP, AllMargin, QBC-AM, QBag, and QBoost form a progression in terms of effort spent in selecting the next training example. QBC-REP expends very little effort, querying only 2 committee members and only considering whether or not to select the currently-offered example. AllMargin expends a little more effort; it also considers only the one example currently being offered, but it uses the predictions of all committee members. QBC-AM, QBag, and QBoost, instead of looking only at the next example being offered, examine some number R of the thus far unselected examples and then select the most informative example from that group.

Considering only total elapsed processor time, the methods are, in increasing order, QBC-REP, AllMargin, QBC-AM, QBag, QBoost. We of course want a method that runs fast, but we are also willing to sacrifice some run time in order to obtain better $F_{1.0}$ scores. This tradeoff obviously depends on the user. But certainly we do not want to use methods that run slower, give worse $F_{1.0}$ results, and use more training examples. As discussed before, AllMargin gives the best $F_{1.0}$ values of the methods tested.

We speculate as follows as regards why a method that uses less elapsed processor time actually achieves better $F_{1.0}$ scores in this domain. In particular, we will compare AllMargin and QBoost since the differences in $F_{1.0}$ and elapsed processor time are most dramatic for these two methods. Recall earlier, when we were describing the AllMargin method, we expressed the concern that if the examples that were selected as "informative" are actually noisy examples and were judged to be "informative" because of the errors that they contain, then our approach would not work very well, since the learners would then be using erroneous examples instead of correct ones. Based on the results of our experiments, this turned out not to be a problem with AllMargin. However, we feel that this in fact is what happens with QBoost in our domain. We have large amounts of noise and large numbers of attributes that are irrelevant. While example *selection* by QBoost does use degree of disagreement, which of the selected examples are *used* for training and to what degree is heavily influenced by the performance of previous learners on the set S. In other words, QBoost is in a sense driven by a desire to reduce training set error. Thus any examples in S that are quite noisy and also repeatedly not learned by the learners will be weighted more heavily for subsequent learners in that cycle. QBoost essentially tries to "force" each of the selected examples to be learned. We feel that, due to the noise and large number of irrelevant attributes in our domain, this can lead to overfitting.

The above conclusions are supported by the following results from our experiments. If "<" is used to stand for "performs worse than" and ">" is used to stand for "performs better than", then:

QBC-REP < AllMargin

-and-

AllMargin > QBC-AM > QBag > QBoost

The performance measure being used is $F_{1.0}$, although the ">" also hold for elapsed processor time.

We also empirically found that increasing R did not help much, and in fact at times made $F_{1.0}$ even worse.

7.4 Weighted Majority Algorithm

We also did experiments using the Weighted Majority Algorithm (WMA) [Littlestone 1989, Littlestone and Warmuth 1994, Cesa-Bianchi et al. 1993]. We used WMA as a mechanism to determine how to weight the votes of the individual committee members. These experiments were performed relatively early in the project. We essentially used the WML algorithm in [Littlestone and Warmuth 1994]. The basic approach we employed for implementing the WMA is as follows: each learner has a voting weight assigned to it, and all of these voting weights are initialized to the same value. The learners are winnows or perceptrons. As each member predicts during the learning process, that member's voting weight is reduced by a factor if the member's prediction is incorrect. One needs to specify parameters such as by what factor one is going to reduce the voting weight when a mistake is made and a parameter which basically puts a lower limit on the voting weight value of each member. The idea is that, as learning occurs, the voting weights will adjust in accordance with each member's performance on the training examples that are used. A lower limit on the voting weights prevents a learner from being completely removed from consideration due to a long series of bad predictions, since we want to allow for learners to recover from their earlier incorrect predictions.

We ran several experiments with various parameter settings, and generally the best performance was obtained by not weighting the member's predictions at all. In light of the above-presented experiments, this is probably not too surprising. The use of voting weights in the manner we have described sounds very similar to boosting, and in fact is discussed in [Freund and Schapire 1995]. In both WMA and boosting, one is using performance of the learner on training examples to set voting weights so that the votes of learners that make fewer errors will be weighted more heavily.

8. Conclusions

In this chapter, we first present the conclusions we have drawn from our research. This is followed by a discussion of the contributions this research makes to the body of knowledge in the fields of machine learning and text categorization. We finish by discussing some areas of possible future work.

In this project, we investigated and developed the ALC framework, which is an organization of machine learning methods based on active learning and committee prediction. ALC systems require minimal preprocessing of the input data and thus data preprocessing time is kept to a minimum. This is of benefit when the database being analyzed is large and/or changing frequently over time. ALC systems, when compared to the corresponding supervised systems, use fewer training examples, run faster, and still obtain high levels of accuracy and other contingency table based performance measures. Our domain of interest is text categorization, but most of the methods developed are quite general and are therefore applicable to other domains. Characteristics of the text processing domain that are shared with many other domains include data collections that are very large, change often, contain large numbers of examples, have high dimensionality, and contain attribute and class noise.

Our goal was to develop and analyze ALC systems, in order not only to design new types of machine learning systems but also to better understand why ALC systems behave as they do. Overall, the ALC systems we have developed have, when applied to the field of text categorization, achieved accuracies exceeding 0.97 and $F_{1.0}$ scores exceeding 0.70. These methods use only about 7% of the training documents available, and they are efficient in both total run time and in total memory requirements. Our results suggest that certain learning algorithms can in fact handle the high dimensionality of full text and can do so efficiently while still yielding good performance results. The two main methods that we developed are QBC-REP and AllMargin. Their performance on text categorization is summarized below:

method	*time*	*% tng ex used*	*accuracy*	$F_{1.0}$
QBC-REP	3	5%	0.965	0.644
AllMargin	30	7%	0.973	0.708

The AllMargin method takes more time than QBC-REP and uses more examples, but gives better $F_{1.0}$ scores.

An important point to note is that reducing the number of training examples needed for learning an accurate approximation to the target function can result in two areas of benefit, and such is the case in the ALC framework. Having fewer examples to label for training reduces human costs involved in this labeling. One also obtains shorter run times since only the selected training examples need to be processed by the learning algorithm. This latter point has applications to many real-world data mining applications in which the problem is not having too little data, but having too much [Burl et al. 1998]. Being able to select the better portions of the data for use in learning would allow the learning process to complete in much less time.

A summary of additional key conclusions drawn from our experiments follows. These experiments were all in the domain of text categorization with minimal preprocessing:

1. Two forces are at work in ALC systems as compared to standard single learner supervised systems. Active learning is used to select the more informative examples on which to learn. This results in a system that needs fewer training examples (which saves labeling costs) and also a system that executes faster (since it does not need to learn from as many examples). Prediction being done by a diverse committee of learners gives us the

benefit of being able to obtain accuracies that are higher than the accuracy of the most accurate member and also allows the system to learn a concept more complex than can be represented by a single learner. We found that both of these effects do in fact contribute to the better performance of ALC systems, and that active learning provides the greater amount of improvement.

2. We found that committees containing 7 - 20 members gave the best results. Larger committees consume more time and can even use more training examples, but give little if any gain in accuracy or $F_{1.0}$. Smaller committees tend to be dominated by one powerful member and so the committee functions more as that one member than as a committee of independent learners.

3. We compared winnow and perceptron learners and found that winnow gives better performance when one looks at the measures of most importance to the typical user of a text categorization system. It appears that the winnow learner is able to find a hyperplane location that provides good predictive capabilities using fewer training examples. We also found that in this domain the ability to learn negative weights was not necessary in order to obtain a good approximation to the target concept. This allowed us to use a version of the winnow algorithm that takes less space and also runs faster.

4. In spite of the fact that the winnow and perceptron algorithms do not automatically compute a rank raw score in the prediction process, we were able to obtain good ranking results using the committee of learners and simple computations based on information that was computed during prediction.

5. Using different definitions of mistake for mistake-driven learners did not seem to provide any benefit, in that the other definitions of mistake resulted in poor $F_{1.0}$ scores. The best definition of mistake seems to be the normal definition – making a prediction that differs from the actual label value.
6. Our examination of eager versus mistake-driven learning indicated that winnow does better in our domain when used as an eager learner.
7. The naive Bayes algorithm is a very interesting form of linear threshold learning algorithm. On the one hand, it performs very well (although it did not seem to scale up as well to full text as the other methods did). On the other hand, it is supervised and so uses all training examples. We decided to design an algorithm that would allow one to have a committee of independent naive Bayes learners that could be used in the ALC framework. The algorithm that we implemented does function in the ALC framework and performs well in terms of contingency table based measures, but it currently is too slow to be practical in this domain. We feel that it is a good starting point for future work.

8.1 Contributions

Our goals for this research were to develop machine learning methods for text categorization that perform minimal preprocessing of the documents, use fewer training examples, are efficient in both space and time complexity, are general, and that perform as well as supervised methods. We have largely accomplished our goals.

8.1.1 Minimal Preprocessing

The consensus in the field of text categorization has been that some type of preprocessing of documents to reduce the dimensionality of the document space allows one to obtain better contingency table based performance measure values and is even necessary as a practical matter for the use of many machine learning algorithms. Most text categorization systems, both commercial and research systems, use some form of dimensionality reduction. Common methods include the use of stop lists, stemming, singular value decomposition, latent semantic indexing, statistical attribute selection, and attribute replacement/creation. We have shown that performing text categorization without a time-expensive dimensionality reducing preprocessing step is feasible. We advocated this approach not only to save time, but also to allow the machine learning algorithm itself to decide which aspects of documents are and are not relevant. This approach also makes it easier for documents to be added to or removed from the collection being examined, and also generalizes the methods used to other domains.

8.1.2 Performance

The systems we have developed use fewer examples for training, in that they do not use all of the training examples available, which is the approach used by supervised learning algorithms. Since labeling of training data requires a human expert, it is desirable to develop methods of machine learning that require fewer labeled examples. Methods developed in this research typically use 5 - 7% of the available training examples, usually with no decrease in contingency table based performance measures as compared to their supervised counterparts, and often obtaining better scores in those performance measures. Using fewer examples also results in considerable time savings as compared to supervised learning, typically around 7 - 10% of the supervised learning time. The ALC systems therefore seem generally successful at determining which training examples are more informative and perform as well as or better than if all examples had been used.

A wrinkle in this conclusion is our experience with the naive Bayes supervised learning algorithm. It consistently performs very well in this domain. However, since it is supervised, it does use all training examples.

8.1.3 Efficiency

Several of the ALC systems that we developed are quite efficient in both time and space. We have not discussed space complexity very much, except to say that it became a non-concern because of the sparseness of the input data. If one has c committee members, d documents in the corpus, t tokens in the dictionary, and if each document contains on the average t_a unique tokens, then the space needed to store the corpus is $O(dt_a)$ (for the indices of the 1-valued attributes) and the space needed by the learner is $O(ct)$ (for the weights). Thus space complexity of the system is very manageable.

We also examined the time complexity for several typical ALC systems that were used in our experiments and found their behavior to be linear in the input size.

8.1.4 Generality and Robustness

The ALC systems we have developed and the ALC framework itself are all quite general. We have shown that the specific methods we use scale up reasonably well from titles only, which is already by machine learning standards a high dimensional domain, to full text. Except for the conversion of the raw data (in our case text) into attribute vectors, the system is not designed around the processing of text. Having several parameters to set in a system can make one suspicious about the robustness of the system. The main parameter in our system is δ, which is a measure of the assumed distance between the main bodies of the YES and NO clouds, but this is actually a parameter for the winnow and perceptron learning algorithms and not a requirement of the ALC framework. In fact, naive Bayes does not require any δ value. We also found in our experiments that when a value of δ is required, it can be within a fairly large range. Recall that δ is in (0,1). We usually found that varying δ by $\pm$ 0.1 did not result in much change in performance, and often variations of $\pm$ 0.2 were easily tolerated.

8.2 Future Work

We see many areas related to this research that are of interest for future research. We are especially interested in further examination, development, and analysis of the algorithms for committees of naive Bayes learners. As many others have noticed, the standard naive Bayes algorithm seems to do quite well in domains where one would, based on the algorithm's assumptions, not expect that its performance would be very good. The standard naive Bayes algorithm also performed very well for us in terms of contingency table based performance measures. It is however supervised and thus uses all training examples. The current version of our attempt at a committee of naive Bayes learners that could be used for active learning does well in the contingency table based performance measures and in using fewer training examples, but it is too slow for this domain. We would like to investigate ways to improve its computational complexity without sacrificing its other performance characteristics, so that its use is practical in our domain. It would also be interesting, since winnow has done so well in the ALC framework, to modify it so that token frequencies are used instead of boolean attributes.

Another area of interest is to apply the ALC methods that we have developed to other types of documents – documents in other fields of knowledge and written in other styles and vocabularies. We would like to experiment with the OHSUMED corpus, which is a subset of the MEDLINE bibliographic database of peer-reviewed medical literature [Hersh et al. 1994, OHSUMED]. We would also like to do experiments with collections of computer science technical reports, with corpora now available that contain documents such as newsgroup postings [20 Newsgroups, Nigam et al. 1998] and web page text [WebKB, Nigam et al. 1998], and also with some of the TREC document collections [Voorhees and Harman 1998] – of special interest to us are the Department of Energy abstracts, Federal Register, Wall Street Journal, Congressional Record, Foreign Broadcast Information Service, and Computer Selects (computer product reviews) collections.

The use of web pages as documents for text categorization also presents another opportunity, besides being an example of another type of document. [Blum and Mitchell 1998] point out that some forms of information have inherent in them the existence of multiple views. For example, web documents themselves have content (the tokens in the document), which is the standard source of information about a document. However, there is also information about web pages in the form of text related to the pointers to them from other documents (i.e., from other web pages). And one could perhaps generalize this pointer concept to also finding information about a web page in web pages to which it points. In other words, one can obtain information about a document $\mathcal{D}$ from the contents of $\mathcal{D}$ itself (as is the standard approach in text categorization) and also from information in other documents that reference $\mathcal{D}$ or that $\mathcal{D}$ references. This idea could perhaps be extended to other types of documents. For example, one could look at the content of a technical paper $\mathcal{P}$ and also the contents portions of other technical papers that point to (i.e., reference) $\mathcal{P}$ or that $\mathcal{P}$ references.

Nothing in the current ALC systems is language-specific except for the method used for tokenizing (i.e., breaking text at whitespace or punctuation), which is quite general and probably workable for most languages using the Latin alphabet. We would like to see how the methods work with languages other than English, such as German, French, and Spanish.

We have as a part of this project developed ALC as an active learning system framework whose purpose is to use active learning to categorize text. It would be interesting to adapt it to other text-related tasks. Active learning has been mainly used in categorization/classification tasks. [Thompson et al. 1999] have experimented with the use of active learning in other text-related tasks, specifically in natural language parsing and information extraction. An area of possible future research would be to expand ALC to in fact be more text-specific by also using active learning to parse the input text. This might allow one to make the document representation richer than it is now, the goal being to incorporate some natural language features into the bag of words concept and thereby improve system performance.

Except for the tokenizing, the ALC systems are not text-specific. We would like to test them on databases created for non-textual classification problems. Some differences between corpora and non-textual databases would need to be investigated to determine their impact on our existing methods, and perhaps modifications would need to be made to the systems due to these differences. For example, it may be that non-textual databases do not necessarily meet the "no negative weights are needed" conclusion that allowed us to use a more efficient form of the winnow learner. Also, non-textual databases may not be very sparse in terms of the average number of attributes that have a value of 1 (t_a) as compared to the total number of attributes (t), and non-textual databases may have much larger percentages of positive examples than in the typical text corpus. There are many non-textual databases that could be used, in a great many fields – for example in the UCI Repository of Machine Learning Databases [Blake et al. 1999]).

For most ALC systems we have examined, the computations that are performed to adjust weights (during learning) and to compute cross-products are very amenable to being parallelized. Given a training example, the weight adjustment for each token is independent of the weight adjustment for any other token. Similarly, during prediction, each token's individual contribution to the sum can be computed independently of the contributions of the other tokens. And perhaps parallelizing would make methods such as the committee of naive Bayes learners more practical in terms of elapsed computation times.

And of course one obvious area of future work is to investigate the use of other learning algorithms in the ALC framework. We have gone to considerable effort to keep the framework very general so that a wide variety of modules can be used for each of the main parts – deciding whether or not to see the label, learning, and forming the committee prediction.

Active learning is in general a topic of current interest. However, most forms of active learning select examples by computing some measure of informativeness based on the votes of individual members in a committee of learners, or on directly computed probabilities. This is done of course without looking at the actual label – that is the whole point. However, this approach encounters significant difficulties if there is a large amount of noise, and also if there are very few positive examples. In the former case, informative examples may in fact be incorrect examples and so the active learner passes over good examples to learn from incorrect ones. In the later case, informative examples will almost always be negative examples, so the learner is attempting to learn only from negative examples. One possible area for research would be to investigate other methods for picking the most informative examples that would, for example, allow favoring examples thought to be positive.

Bibliography

[20 Newsgroups] 20 Newsgroups database from the CMU Text Learning Group Data Archives, assembled by Ken Lang, available from: `http://www.cs.cmu.edu/~textlearning`

[Abe and Mamitsuka 1998] Naoki Abe and Hiroshi Mamitsuka, Query Learning Strategies using Boosting and Bagging, *Proceedings: ICML'98*, p. 1-9, 1998

[Aha et al. 1991] David W. Aha, Dennis Kibler, and Marc K. Albert, Instance-Based Learning Algorithms, *Machine Learning* 6(1):37-66, January 1991

[Almuallim and Dietterich 1991] Hussein Almuallim and Thomas G. Dietterich, Learning with Many Irrelevant Features, *Proceedings: AAAI-91*, 1991

[Angluin 1988] Dana Angluin, Queries and Concept Learning, *Machine Learning* 2:319-342, 1988

[Apté et al. 1994] Chidanand Apté, Fred Damerau, and Sholom M. Weiss, Automated Learning of Decision Rules for Text Categorization, *ACM TOIS* 12(2):233-251, July 1994

[Argamon-Engelson and Dagan 1999] Shlomo Argamon-Engelson and Ido Dagan, Committee-Based Sample Selection for Probabilistic Classifiers, *Journal of Artificial Intelligence Research 11:335-360*, 1999

[Aslam and Decatur 1996] Javed A. Aslam and Scott E. Decatur, On the Sample Complexity of Noise-Tolerant Learning, *Information Processing Letters* 57(4):189-195, 26 February 1996

[Bay 1999] S. D. Bay, The UCI Knowledge Discovery in Databases Archive, Department of Information and Computer Science, University of California at Irvine, 1999, `http://kdd.ics.uci.edu`

[Berry et al. 1995] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien, Using Linear Algebra for Intelligent Information Retrieval, *SIAM Review* 37(4):573-595, December 1995

[Bellman 1970] Richard Bellman, *Introduction to Matrix Analysis, Second Edition*, Mc-Graw-Hill, 1970

[Blake et al. 1999] C. Blake, E. Keogh, and C. J. Merz, UCI Repository of Machine Learning Databases, Department of Information and Computer Science, University of California at Irvine, 1999, `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[Blum 1995] Avrim Blum, Empirical Support for Winnow and Weighted-Majority Based Algorithms: Results on a Calendar Scheduling Domain, *Proceedings: ICML'95*, 1995

[Blum and Mitchell 1998] Avrim Blum and Tom Mitchell, Combining Labeled and Unlabeled Data with Co-Training, *Proceedings: COLT'98*, p. 92-100, 1998

[Blumer et al. 1989] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth, Learnability and the Vapnik-Chervonenkis Dimension, *Journal of the ACM* 36(4):929-965, October 1989

[Bradley and Mangasarian 1998] P. S. Bradley and O. L. Mangasarian, Feature Selection via Concave Minimization and Support Vector Machines, *Proceedings: ICML'98*, p. 82-90, 1998

[Breiman 1996a] Leo Breiman, Bagging Predictors, *Machine Learning* 24(2):123-140, 1996

[Breiman 1996b] Leo Breiman, Heuristics of Instability and Stabilization in Model Selection, *Annals of Statistics* 24(6):2350-2383, December 1996

[Buckley et al. 1994] Chris Buckley, Gerard Salton, and James Allen, The Effect of Adding Relevance Information in a Relevance Feedback Environment, *Proceedings: SIGIR'94*, p. 292-300, 1994

[Burl et al. 1998] Michael C. Burl, Lars Asker, Padhraic Smyth, Usama Fayyad, Pietro Perona, Larry Crumpler, and Jayne Aubele, Learning to Recognize Volcanoes on Venus, *Machine Learning* 30(2/3):165-194, February/March 1998

[Bylander 1998] Tom Bylander, Learning Noisy Linear Threshold Functions, Technical Report #PS129, submitted, 15 April 1998

[Castelli and Cover 1995] Vittorio Castelli and Thomas M. Cover, On the Exponential Value of Labeled Samples, *Pattern Recognition Letters* 16(1):105-111, January 1995

[Cesa-Bianchi et al. 1993] N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, and M. Warmuth, On-line Prediction and Conversion Strategies, *Proceedings: EuroColt'93*, p. 205-216, 1993

[Cestnik 1990] Bojan Cestnik, Estimating Probabilities: A Crucial Task in Machine Learning, *Proceedings: ECAI-90*, p. 147-149, 1990

[Cheeseman and Stutz 1995] Peter Cheeseman and John Stutz, Bayesian Classification (AutoClass): Theory and Results, in *Advances in Knowledge Discovery and Data Mining*, Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, Ramasamy Uthurusamy, editors, The MIT Press, 1996, chapter 6

[Church 1995] Kenneth Ward Church, One Term or Two?, *Proceedings: SIGIR'95*, p. 310-318, 1995

[Cohen and Singer 1996] William W. Cohen and Yoram Singer, Context-Sensitive Learning Methods for Text Categorization, *Proceedings: SIGIR'96*, p. 307-315, 1996

[Cohen and Singer 1999] William W. Cohen and Yoram Singer, Context-Sensitive Learning Methods for Text Categorization, *ACM Transactions on Information Systems* 17(2):141-173, April 1999

[Cohn et al. 1994] David Cohn, Les Atlas, and Richard Ladner, Improving Generalization with Active Learning, *Machine Learning* 15(2):201-221, May 1994

[Cook et al. 1996] Diane Cook, Joseph Potts, and Will Taylor, Peter Cheeseman, John Stutz; AutoClass C, version 2.7, software and documentation; available from: `ftp://csr.uta.edu:/pub`

[Cooper 1969] W. S. Cooper, Is Interindexer Consistency a Hobgoblin?, *American Documentation* 20:268-278, 1969

[Cormen et al. 1996] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1996

[D'Ambrosio 1996] Bruce D'Ambrosio, UAI-96 Panel Discussion -- UAI by 2005: Reflections on Critical Problems, Directions, and Likely Achievements for the Next Decade, 3 August 1996

[Dagan and Engelson 1995] Ido Dagan and Sean P. Engelson, Committee-Based Sampling for Training Probabilistic Classifiers, *Proceedings: ML95*, 1995, p. 150-157

[Dagan et al. 1997] Ido Dagan, Yael Karov, and Dan Roth, Mistake-Driven Learning in Text Categorization, *Proceedings of Second Conference on Empirical Methods in Natural Language Processing (EMNLP-2)*, August 1997

[Devaney and Ram 1997] Mark Devaney and Ashwin Ram, Efficient Feature Selection in Conceptual Clustering, *Proceedings: ICML'97*, p. 92-97, 1997

[Dietterich 1997] Thomas G. Dietterich, Machine-Learning Research: Four Current Directions, *AI Magazine* 18(4):97-136, 1997

[Doak 1992] Justin Doak, *An Evaluation of Feature Selection Methods and Their Application to Computer Security*, Technical Report CSE-92-18, Department of Computer Science, University of California at Davis, 1992

[Domingos and Pazzani 1996] Pedro Domingos and Michael Pazzani, Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier, *Proceedings: ICML'96*, p. 105-112, 1996

[Domingos and Pazzani 1997] Pedro Domingos and Michael Pazzani, On the Optimality of the Simple Bayesian Classifier under Zero-One Loss, *Machine Learning* 29(2-3):103-130, November-December 1997

[Duda and Hart 1973] Richard O. Duda and Peter E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973

[Dumais et al. 1998] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami, Inductive Learning Algorithms and Representations for Text Categorization, *Proceedings: CIKM'98*, p. 148-155, 1998

[Fayyad and Simoudis 1996] Usama Fayyad and Evangelos Simoudis, Knowledge Discovery and Data Mining, a tutorial presented at AAAI-96, 1996

[Franklin 1968] Joel N. Franklin, *Matrix Theory*, Prentice-Hall, 1968

[Freund et al. 1992] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby, Information, Prediction, and Query by Committee, *Proceedings: NIPS-92*, p. 483-490, 1992

[Freund and Schapire 1995] Yoav Freund and Robert E. Schapire, A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, AT&T Bell Laboratories, 20 September 1995

[Freund and Schapire 1996] Yoav Freund and Robert E. Schapire, Experiments with a New Boosting Algorithm, *Proceedings: ICML'96*, p. 148-156, 1996

[Freund et al. 1997] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby, Selective Sampling Using the Query by Committee Algorithm, *Machine Learning* 28(2-3):133-168, August-September 1997

[Golding and Roth 1999] Andrew R. Golding and Dan Roth, A Winnow-Based Approach to Context-Sensitive Spelling Correction, *Machine Learning* 34(1/2/3):107-130, February 1999

[Gravetter and Wallnau 1985] Frederick J. Gravetter and Larry B. Wallnau, *Statistics for the Behavioral Sciences*, West Publishing:St. Paul, Minnesota, 1985

[Hansen and Salamon 1990] Lars Kai Hansen and Peter Salamon, Neural Network Ensembles, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12(10):993-1001, October 1990

[Hanson et al. 1991] Robin Hanson, John Stutz, and Peter Cheeseman, Bayesian Classification Theory, Artificial Intelligence Research Branch, NASA Ames Research Center, Technical Report FIA-90-12-7-01, May 1991

[Harman 1996] Donna Harman, TREC-5 Guidelines, 21 March 1996

[Hayes et al. 1990] Phillip J. Hayes, Peggy M. Andersen, Irene B. Nirenburg, and Linda M. Schmandt, TCS: A Shell for Content-Based Text Categorization, *Proceedings of the 6th IEEE CAIA (Conference on Artificial Intelligence for Applications)*, p. 320-326, 1990

[Hayes and Weinstein 1991] Philip J. Hayes and Steven P. Weinstein, Construe-TIS: A System for Content-Based Indexing of a Database of News Stories, in *Innovative Applications of Artificial Intelligence 2*, Alain Rappaport, Reid Smith, editors, p. 48-64, AAAI Press/MIT Press, 1991

[Hearn and Baker 1997] Donald Hearn and N. Pauline Baker, *Computer Graphics, C Version, Second Edition*, Prentice Hall, 1997

[Helmbold et al. 1998] David P. Helmbold, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth, On-Line Portfolio Selection Using Multiplicative Updates, *Mathematical Finance*, to appear

[Hersh et al. 1994] William Hersh, Chris Buckley, T. J. Leone, and David Hickam, OHSUMED: An Interactive Retrieval Evaluation and New Large Test Collection for Research, *Proceedings: SIGIR'94*, p. 192-201, 1994

[Hersh 1996] William R. Hersh, *Information Retrieval: A Health Care Perspective*, Springer-Verlag, 1996

[Höffgen and Simon 1992] Klaus - U. Höffgen and Hans - U. Simon, Robust Trainability of Single Neurons, *Colt '92 Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, p. 428-439, ACM Press, 1992

[Hull 1996] David A. Hull, Stemming Algorithms: A Case Study for Detailed Evaluation, *Journal of the American Society for Information Science* 47(1):70-84, January 1996

[Joachims 1998] Thorsten Joachims, Text Categorization with Support Vector Machines: Learning with Many Relevant Features, *Proceedings: ECML'98*, 1998; also, a more detailed version is available as LS-8 Technical Report 23, Department of Computer Science, University of Dortmund, 19 April 1998, available from:
`http://www-ai.cs.uni-dortmund.de`

[Jelonek et al. 1995] Jacek Jelonek, Krzysztof Krawiec, and Roman Slowiński, Rough Set Reduction of Attributes and Their Domains for Neural Networks, *Computational Intelligence* 11(2):339-347, May 1995

[Kantor 1999] Paul B. Kantor, personal communication (ddlbeta mailing list issues dated 1 August 1999)

[Kearns and Li 1988] Michael Kearns and Ming Li, Learning in the Presence of Malicious Errors, Technical Report, 1988, available from:
`http://www.research.att.com/~mkearns`
a preliminary version appeared in *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing*, p. 267-280, 1988

[King et al. 1995] R. D. King, C. Feng, and A. Sutherland, Statlog: Comparison of Classification Algorithms on Large Real-World Problems, *Applied Artificial Intelligence* 9(3):289-333 , May-June 1995

[Kivinen and Warmuth 1995] Jyrki Kivinen and Manfred K. Warmuth, The Perceptron Algorithm vs. Winnow: Linear vs. Logarithmic Mistake Bounds When Few Input Variables Are Relevant, *Proceedings: COLT'95*, 1995

[Kohavi 1995] Ron Kohavi, Wrappers for Performance Enhancement and Oblivious Decision Graphs, Doctoral Thesis, Department of Computer Science, Stanford University, 1995

[Kononenko 1990] Igor Kononenko, Comparison of Inductive and Naive Bayesian Learning Approaches to Automatic Knowledge Acquisition, *Current Trends in Knowledge Acquisition*, IOS Press, 1990, p. 190-197

[Kreyszig 1962] Erwin Kreyszig, *Advanced Engineering Mathematics*, John Wiley and Sons, 1962

[Kudenko and Hirsh 1998] Daniel Kudenko and Haym Hirsh, Feature Generation for Sequence Categorization, *Proceedings: AAAI'98*, p. 733-738, 1998

[Langley 1993] Pat Langley, Induction of Recursive Bayesian Classifiers, *Proceedings: ECML-93*, 1993

[Lay 1993] David C. Lay, *Linear Algebra and Its Applications*, Addison-Wesley, 1993

[Lewis 1991a] David D. Lewis, Evaluating Text Categorization, *Proceedings of the Speech and Natural Language Workshop*, Asilomar, February 1991

[Lewis 1991b] David D. Lewis, Representation and Learning in Information Retrieval, Ph.D. Thesis, University of Massachusetts at Amherst, COINS Technical Report 91-93, December 1991

[Lewis 1992] David D. Lewis, Feature Selection and Feature Extraction for Text Categorization, *Proceedings of the Speech and Natural Language Workshop*, Arden House, February 1992

[Lewis and Gale 1994] David D. Lewis and William A. Gale, A Sequential Algorithm for Training Text Classifiers, *Proceedings: SIGIR 94*, p. 3-12, 1994

[Lewis and Ringuette 1994] David D. Lewis and Marc Ringuette, A Comparison of Two Learning Algorithms for Text Categorization, *Symposium on Document Analysis and Information Retrieval*, ISRI, University of Nevada at Las Vegas, p. 81-93, 1994

[Lewis et al. 1996] David D. Lewis, Robert E. Schapire, James P. Callan, and Ron Papka, Training Algorithms for Linear Text Classifiers, *Proceedings: SIGIR'96*, p. 298-306, 1996

[Liere and Tadepalli 1996] Ray Liere and Prasad Tadepalli, The Use of Active Learning in Text Categorization, *Working Notes of the AAAI 1996 Spring Symposium on Machine Learning in Information Access*, Stanford, CA, 1996

[Liere 1997] Ray Liere, Combining Active Learning and Learning by Committee to Categorize Text, Doctoral Proposal, (presented on 3 October 1996), Department of Computer Science, Oregon State University, Technical Report 97-30-1, 9 February 1997

[Liere and Tadepalli 1997] Ray Liere and Prasad Tadepalli, Active Learning with Committees for Text Categorization, *Proceedings: AAAI-97*, p. 591-596

[Liere and Tadepalli 1998a] Ray Liere and Prasad Tadepalli, Active Learning with Committees: Preliminary Results in Comparing Winnow and Perceptron in Text Categorization, *Proceedings of 1998 Conference on Automated Learning and Discovery (CONALD'98)*, 1998

[Liere and Tadepalli 1998b] Ray Liere and Prasad Tadepalli, Active Learning with Committees in Text Categorization: Preliminary Results in Comparing Winnow and Perceptron, *Learning for Text Categorization*, Technical Report WS-98-05, AAAI Press, 1998

[Littlestone 1988] Nick Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm, *Machine Learning* 2(4):285-318, 1988

[Littlestone 1989] Nicholas Littlestone, Mistake Bounds and Logarithmic Linear-Threshold Learning Algorithms, University of California at Santa Cruz, UCSC-CRL-89-11, March 1989

[Littlestone 1991] Nick Littlestone, Redundant Noisy Attributes, Attribute Errors, and Linear-Threshold Learning Using Winnow, *Proceedings: COLT'91*, p. 147-156

[Littlestone and Warmuth 1994] Nick Littlestone and Manfred K. Warmuth, The Weighted Majority Algorithm, *Information and Computation* 108(2):212-261, 1994

[Littlestone 1995] Nick Littlestone, Comparing Several Linear-Threshold Learning Algorithms on Tasks Involving Superfluous Attributes, *Proceedings: ICMP'95*, 1995

[McCallum and Nigam 1998] Andrew Kachites McCallum and Kamal Nigam, Employing EM and Pool-Based Active Learning for Text Classification, *Proceedings: ICML'98*, p. 350-358, 1998

[McCallum et al. 1998] Andrew McCallum, Ronald Rosenfeld, Tom Mitchell, and Andrew Y. Ng, Improving Text Classification by Shrinkage in a Hierarchy of Classes, *Proceedings: ICML'98*, p. 359-367, 1998

[McGill 1979] Michael J. McGill, An Evaluation of Factors Affecting Document Ranking by Information Retrieval Systems, School of Information Studies, Syracuse University, NTIS #PB80-119506, October 1979

[Mitchell 1997] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997

[Neapolitan 1990] Richard E. Neapolitan, *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*, John Wiley & Sons, 1990

[Nigam et al. 1998] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell, Learning to Classify Text from Labeled and Unlabeled Documents, *Proceedings: AAAI'98*, p. 792-799, 1998

[O'Kane and Winther 1994] Dominic O'Kane and Ole Winther, Learning to Classify in Large Committee Machines, *Physical Review E (Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics)* 50(4):3201-3209, October 1994

[OHSUMED] OHSUMED, a MEDLINE subset, available via anonymous ftp from: `ftp://medir.ohsu.edu:/pub/ohsumed/`

[Pearl 1988] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988

[Perlman] Gary Perlman, |STAT version 5.4, software and documentation, available from: `ftp://archive.cis.ohio-state.edu/pub/stat/`

[Prather 1988] R. E. Prather, Comparison and Extension of Theories of Zipf and Halstead, *The Computer Journal* 31(3):248-252, June 1988

[Quinlan 1996] J. R. Quinlan, Bagging, Boosting, and C4.5, *Proceedings: AAAI-96*, p. 725-730, 1996

[Ratsaby and Venkatesh 1995] Joel Ratsaby and Santosh S. Venkatesh, Learning from a Mixture of Labeled and Unlabeled Examples with Parametric Side Information, *Proceedings: COLT'95*, 1995

[Read and Cressie 1988] Timothy R. C. Read and Noel A. C. Cressie, *Goodness-of-Fit Statistics for Discrete Multivariate Data*, Springer-Verlag, 1988

[Reuters-22173] Reuters Ltd, Carnegie Group, David Lewis, Information Retrieval Laboratory at the University of Massachusetts; Reuters-22173 corpus, a collection of 22,173 indexed documents appearing on the Reuters newswire in 1987; available from:
`ftp://ciir-ftp.cs.umass.edu:/pub/reuters1`

[Reuters-21578] Reuters Ltd, Carnegie Group, David Lewis, Information Retrieval Laboratory at the University of Massachusetts; Reuters-21578 corpus, a collection of 21,578 indexed documents appearing on the Reuters newswire in 1987; available from:
`ftp://www.research.att.com/~lewis`

[Riloff 1995] Ellen Riloff, Little Words Can Make a Big Difference for Text Classification, *Proceedings: SIGIR'95*, p. 130-136, 1995

[Rocchio 1971] J. J. Rocchio, Jr., Relevance Feedback in Information Retrieval, in *The SMART Retrieval System: Experiments in Automatic Document Processing*, Prentice-Hall, 1971

[Rosenblatt 1962] Frank Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books: Washington D.C., 1962

[Roth 1998] Dan Roth, Learning to Resolve Natural Language Ambiguities: A Unified Approach, *Proceedings: AAAI'98*, p. 806-813, 1998

[Sakakibara et al. 1996] Yasubumi Sakakibara, Kazuo Misue, and Takeshi Koshiba, A Machine Learning Approach to Knowledge Acquisitions From Text Databases, *International Journal of Human-Computer Interaction* 8(3):309-324, 1996

[Salton 1971] Gerard Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, Prentice-Hall, 1971

[Salton and McGill 1983] Gerard Salton and Michael J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983

[Salton 1989] Gerard Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989

[Saracevic 1991] Tefko Saracevic, Individual Differences in Organizing, Searching, and Retrieving Information, *ASIS '91: Proceedings of the 54th ASIS Annual Meeting*, p. 82-86, 1991

[Saund 1989] Eric Saund, Dimensionality-Reduction Using Connectionist Networks, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11(3):304-314, March 1989

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

[Schütze et al. 1996] Hinrich Schütze, David A. Hull, and Jan O. Pedersen, A Comparison of Classifiers and Document Representations for the Routing Problem, *Proceedings: SIGIR'95*, p. 229-237, 1995

[Schwartze and Hertz 1993] H. Schwarze and J. Hertz, Learning from Examples in Fully Connected Committee Machines, *Journal of Physics A (Mathematical and General)* 26(19):4919-4936, 7 October 1993

[Scott and Matwin 1999] Sam Scott and Stan Matwin, Feature Engineering for Text Classification, *Proceedings: ICML'99*, p. 379-388, 1999

[Servedio 1999] Rocco A. Servedio, On PAC Learning Using Winnow, Perceptron, and a Perceptron-Like Algorithm, *Proceedings: COLT'99*, p. 296-307, 1999

[Seung et al. 1992] H. S. Seung, M. Opper, and H. Sompolinsky, Query by Committee, *Proceedings: COLT'92*, p. 287-294, 1992

[Sherrah 1998] Jamie Sherrah, Automatic Feature Extraction for Pattern Recognition, Doctoral Thesis, Department of Electrical and Electronic Engineering, University of Adelaide, South Australia, 10 July 1998

[Sievert and Andrews 1991] MaryEllen C. Sievert, and Mark J. Andrews, Indexing Consistency in Information Science Abstracts, *Journal of the American Society for Information Science* 42(1):1-6, January 1991

[Singh and Provan 1995] Moninder Singh and Gregory M. Provan, A Comparison of Induction Algorithms for Selective and non-Selective Bayesian Classifiers, *Proceedings: ML'95*, 1995

[Snedecor and Cochran 1989] George W. Snedecor and William G. Cochran, *Statistical Methods, Eighth Edition*, Iowa State University Press:Ames, Iowa, 1989

[Spackman 1988] Kent Alan Spackman, Creating Decision Criteria from Examples: The Criteria Learning System (CRLS), Doctoral Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1988

[Takiyama 1978] Ryuzo Takiyama, A Learning Method for the Committee Machine with an Arbitrary Logic, *Transactions of the Institute of Electronics and Communication of Japan. Section E: English* 61(5):392, May 1978

[Thompson et al. 1999] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney, Active Learning for Natural Language Parsing and Information Extraction, *Proceedings: ICML'99*, p. 406-414, 1999

[Valiant 1984] L. J. Valiant, A Theory of the Learnable, *Communications of the ACM* 27(11):1134-1142, November 1984

[van Rijsbergen 1979] C. J. van Rijsbergen, *Information Retrieval*, Second Edition, Butterworths, 1979

[Voorhees and Harman 1997] E. M. Voorhees and D. K. Harman, Evaluation Techniques and Measures, *Information Technology: The Fifth Text REtrieval Conference (TREC-5)*, NIST Special Publication 500-238; p. A-14, A-15; November 1997

[Voorhees and Harman 1998] Ellen M. Voorhees and Donna Harman, Overview of the Sixth Text REtrieval Conference (TREC-6), *Information Technology: The Sixth Text REtrieval Conference (TREC-6)*, NIST Special Publication 500-240, p. 1-24, August 1998

[WebKB] WebKB/The 4 Universities Database, WWW-pages collected from computer science departments of various universities in January 1997 by the World Wide Knowledge Base (WebKB) project of the CMU Text Learning Group, available from:
`http://www.cs.cmu.edu/~textlearning`

[Weiss and Kulikowski 1990] Sholom M. Weiss and Casimir A. Kulikowski, *Computer Systems that Learn*, Morgan Kaufmann, 1990

[Weiss et al. 1999] Sholom M. Weiss, Chidanand Apte, Fred J. Damerau, David E. Johnson, Frank K. Oles, Thilo Goetz, and Thomas Hampp, Maximizing Text-Mining Performance, *IEEE Intelligent Systems* 14(4):63-69, July/August 1999

[Yang and Chute 1994] Yiming Yang and Christopher G. Chute, An Example-Based Mapping Method for Text Categorization and Retrieval, *ACM Transactions on Information Systems* 12(3):252-277, July 1994

[Yang 1996] Yiming Yang, personal communication (ddlbeta mailing list issues dated 10 July 1996)

[Yang et al. 1998] Yiming Yang, Tom Pierce, and Jaime Carbonell, A Study on Retrospective and On-Line Event Detection, *Proceedings: SIGIR'98*, p. 28-36, 1998

[Yang 1999] Yiming Yang, An Evaluation of Statistical Approaches to Text Categorization, *Information Retrieval* 1(1/2):69-90, 1999

[Yang and Liu 1999] Yiming Yang and Xin Liu, A Re-examination of Text Categorization Methods, *Proceedings: SIGIR'99*, p. 42-49, 1999

[Zörnig and Altmann 1995] Peter Zörnig and Gabriel Altmann, Unified Representation of Zipf Distributions, *Computational Statistics & Data Analysis* 19(4):461-473, April 1995