

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

Selective Naive Bayes Classification

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Computer Science

by

Jeremy Buchmann

Dr. Sushil Louis/Thesis Advisor

December 2003

UMI Number: 1417238



UMI Microform 1417238

Copyright 2004 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

©Jeremy L. Buchmann 2003

UNIVERSITY
OF NEVADA
RENO

THE GRADUATE SCHOOL

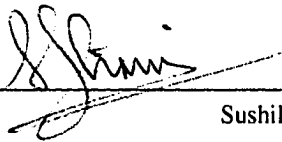
We recommend that the thesis
prepared under our supervision by

JEREMY L. BUCHMANN

entitled
Selective Naive Bayes Classification

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE



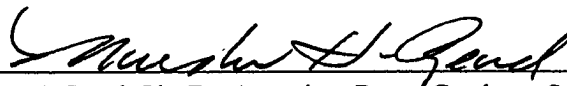
Sushil J. Louis, Ph.D., Advisor



Monica N. Nicolescu, Ph.D., Committee Member



M. Sami Fadali, Ph.D., Graduate School Representative



Marsha H. Read, Ph. D., Associate Dean, Graduate School

December 2003

Abstract

Text classification is a problem that grows more relevant each day. It was recently estimated that there are almost 800 MB of data produced per person, per year, with 92% of that data being stored on magnetic media. Much of this data is text (memos, reports, emails, Web pages) which is often classified according to some criteria set by the user. The ability to automatically classify this data would save time and improve the usability of computers systems. In this thesis, we introduce a technique called selection for improving the performance of an automatic text classification system in a hierarchical class space as well as a flat class space. To demonstrate selection, we developed an automatic text classifier called Leach. Using selection, we were able to achieve an increase in classification accuracy using a common Usenet data set.

Contents

Abstract	i
List of Figures	iv
List of Tables	v
1 Introduction	1
2 The Selective Naive Bayes Algorithm	6
2.1 Classification	6
2.2 The Naive Bayes Algorithm	7
2.3 The Naive Bayes Algorithm Applied to Text Classification	9
2.4 Flaws in the Naive Bayes Algorithm	11
2.5 Hierarchical Classification	12
2.6 Selection	13
2.7 Leach Algorithm	14
2.8 An Example Run of Leach	16

3	Experimental Setup	21
4	Results	24
5	Conclusions and Future Work	31
A	List of Stop Words	33

List of Figures

2.1	A flowchart of the first stage of Leach	15
2.2	A flowchart of the second stage of Leach	16

List of Tables

2.1	Frequency distributions for the two example documents.	17
2.2	Frequency distribution for the input document.	18
3.1	How we made the 20 Newsgroups data set hierarchical	22
4.1	Results for the Autos classes	25
4.2	Results for the Computers classes	25
4.3	Results for the Other class	26
4.4	Results for the Politics classes	26
4.5	Results for the Religion classes	27
4.6	Results for the Science classes	28
4.7	Results for the Sports classes	28
4.8	Overall Results	30

Chapter 1

Introduction

Classifying large amounts of data is quickly becoming a more important and more time-consuming task for businesses, researchers, journalists, and everyday computer users. It was recently estimated that there are almost 800 MB of data produced per person, per year, with 92% of that data being stored on magnetic media [1]. The sheer amount of data available on the Internet and from other sources makes organizing and interpreting the data a tedious task. Search engines are useful for extracting documents on a keyword basis, but are subject to contextual problems. Classifying documents by hand is a tedious process for anything more than a trivial number of documents.

The goal of automated text classification is to group related documents together in a structure that is logical and easy to navigate. This structure provides several uses:

- A particular document can be located by finding its class and then looking at each document in the class. This is faster than searching every document in the entire archive.
- Related documents can be found by looking at the other documents in the class.

A classifier is a tool used to perform the classification, and there are many different classification algorithms and techniques [2]. Even if they are not 100% accurate, text classifiers can significantly reduce the time it takes to classify a large number of documents. For example, consider two companies that agree to merge and must combine their entire collections of documents ranging from technical to legal. There may be hundreds of thousands of documents that need to be grouped and rearranged into a common hierarchy. A document classifier could be used as a first step in the process of organizing documents. Later, humans could go back and verify the correct classifications and fix the incorrect classifications. Using a document classifier as a pre-processing step can significantly reduce the time it would take to organize the documents.

For another example, consider the act of saving a file. Many systems provide a default location to save a file. In Windows, this is often the “My Documents” folder; in UNIX or Linux, it is often the user’s home directory. From there, the user may have to navigate deep into the filesystem hierarchy to find an appropriate location for the file. It would be much more convenient if the system could locate an appropriate location for the file based on the file’s contents and suggest this location to the user.

The examples of classifying company documents and suggesting a location to save a file are examples of supervised learning. Most supervised learning techniques create a function that predicts a class label for a given input object. In the given examples, the training data were documents that had already been manually classified. This is different from unsupervised learning, where we usually do not have pre-labeled examples, and where the goal is to generate a model to accommodate the input data. A form of unsupervised learning is clustering, where the goal is to group together input data without knowing the structure of the groups in advance.

Text classification has been an active topic in computer science for over forty years [3]. Many different classification algorithms have been created, and some have proven to be better at certain tasks than others. Naive Bayes and its variations have remained popular for text classification for several reasons. First, they are simple to create. Second, they are fast to train and classify. Other classification methods such as Support Vector Machines, neural networks, Linear Least Squares Fit, and k-nearest neighbor have demonstrated good text classification accuracy [4]. Mitchell's text provides an excellent introduction to classification and several classification techniques [2].

Thanks to new applications like spam filtering, naive Bayes classifiers have enjoyed increased popularity due to their speed and ease of adding to their training examples [5]. Some classifiers require a complete retraining cycle each time a new example is added to the training corpus, however, a naive Bayes classifier does not. This is a very desirable feature to have in an email program where users do not expect the program to require frequent lengthy retraining cycles. Visible projects such as Mozilla and SpamAssassin use Bayesian classifiers in their spam recognition systems [6, 7].

Hierarchical classification has been performed in several different ways with positive results. Koller and Sahami were able to achieve improved accuracy using hierarchical classification and very few words [8]. Instead of using a naive Bayes classifier, they used a Bayesian classifier with weak word association and restricted the number of words to keep training time reasonable. They approached hierarchical classification by creating a hierarchy of classifiers, one for each non-leaf node in the tree. By doing this, they were able to reduce the necessary number of features. However, their technique may not be sufficient on data sets with large vocabularies.

McCallum, Rosenfield et al. used a different approach, but were also able to improve

accuracy versus classification of a flat class space [9]. They started with a naive Bayes classifier and added a technique called shrinkage, which pushes similar word occurrence probabilities to a common value.

This thesis defines a new technique called selection which is designed to improve the performance of text classifiers. We show that selection can slightly improve overall classification performance and can significantly improve performance in certain cases for a naive Bayes classifier. Selection should also work with any classifier that generates a probability or score that depends on data from other classes. For example, selection should work with statistical classifiers, but not classifiers that use vector-based distance measures.

We start by describing the system, Leach (Little Elegant Java Classifier Hierarchical), that we developed for investigating the hierarchical and selection concepts. Leach implements the selective naive Bayes algorithm and can classify hierarchical data of any depth. Results using Leach on the 20 Newsgroups data set, a commonly used benchmark, show that when using selection, Leach produces a small but statistically significant improvement on overall classification results.

Chapter 2 starts by describing classification in general, and then covers the basic naive Bayes algorithm. We then describe hierarchical classification and our extension to naive Bayes: selection. Following that is a discussion of hierarchical classification and how the selection process is applied to naive Bayes. Chapter 2 closes with a detailed description of how Leach operates and then works through a simple example.

In Chapter 3, we introduce the classifier we used for a comparison, the data set we used for testing the classifiers, and how we modified the data set to turn it into a hierarchical data set.

Chapter 4 presents the results and provides an analysis of Leach's performance. We

look at each of the top-level classes and analyze Leach's performance with and without selection. Afterwards, we analyze the overall performance.

Finally, in Chapter 5, we provide our conclusions and suggestions for future work. We conclude that selection improves overall performance and future work could involve error recovery and classification into more than one class.

Chapter 2

The Selective Naive Bayes Algorithm

Before going into an explanation of naive Bayes classification, we first discuss classification in general and then show how naive Bayes classification is just one implementation of a classifier. This background is necessary to understand later sections.

2.1 Classification

When classifying an object, you are grouping it with other objects that are like it in some way. A classifier is a tool used to perform the classification, and there are many different classification algorithms and techniques [2]. Some require extensive training, while some need very little. Each has its own strengths and weaknesses, but they all operate using similar inputs and outputs. In this thesis, we are only interested in classifying objects into a pre-existing class space.

The inputs to a classifier are usually composed of *feature vectors*. A feature vector is an ordered collection of attributes that describe an instance belonging to a class. For example, consider a sample of college admission applications taken from a university's

admissions office. Each application can be represented by a feature vector. The applications might show such *features* as an applicant's age, high school GPA, activities, etc. If the same data is available for each applicant, you would have a collection of data that can be organized into a set of *vectors*, each of which can be represented by $(v_1, v_2, \dots, v_{|V|})$ where v_1 might represent the applicant's age, v_2 might represent the applicant's high school GPA, and so on. A *feature vector* provides a simple but useful representation of a student's application.

The output of a classifier is the class that the feature vector should belong to. In this example, you may have classes such as "Admit With Scholarship", "Admit Without Scholarship", "Postpone Decision", and "Deny". This classifier could provide much help to university admission offices by pre-processing applications and making recommendations for further review. In combination with a Web-based applications process for prospective students, this classifier could save a university time and provide the applicant with a faster response.

2.2 The Naive Bayes Algorithm

The naive Bayes algorithm is a statistical machine learning algorithm that uses probabilities to decide the class to which a given vector belongs. The probability we are looking for can be described as "The probability of class c given vector V ". In mathematical terms, this can be described as:

$$P(c | V)$$

With multiple classes, we want to find the class that has the highest probability of owning vector V . This probability is called a *posterior probability*. It tells us the

probability that vector V belongs to class c . The posterior probability cannot be gathered directly from the data, but it can be calculated using Bayes theorem.

The naive Bayes algorithm uses a simplification of Bayes theorem to calculate the probability of a vector belonging to a class. McCallum and Nigam call this the probability of a class *producing* a vector, but in this thesis, we use the former terminology [10].

In this thesis, we use the multinomial event model which is one of several statistical models used in naive Bayes classifiers. A full discussion of the different event models is outside the scope of this work, but since the multinomial model has been shown to produce better results on larger data sets, it is what we use [10].

Assume V is a feature vector $(v_1, v_2, \dots, v_{|V|})$ that represents an instance that the classifier is trying to classify. The set of classes is represented by $C = (c_1, c_2, \dots, c_{|C|})$. Bayes theorem allows us to calculate the posterior probability from the *likelihood* and the *prior probabilities*:

$$P(c_i | V) = \frac{P(V | c_i) \times P(c_i)}{P(V)} \quad (2.1)$$

$P(V | c_i)$ is known as the likelihood and is the probability of the occurrence of vector V given class c_i . $P(c_i)$ is called the prior probability and is the probability of the occurrence of class c_i . The meanings behind these terms is explained in Section 2.3.

Naive Bayes depends on the *independence assumption*. The independence assumption says that the probability of the occurrence of a word is independent of the existence of all other words as well as its location in the document. This assumption is almost certainly wrong. For example, the phrase “Red Sox” would be hard to imagine as pertaining to something other than baseball. However, using the independence as-

sumption, the words are assumed to be completely independent since the naive Bayes classifier does not take the proximity of the two words into account. The advantage of the independence assumption is that it allows us to greatly simplify our learning method while still providing good classification accuracy [10]. In contrast, calculating N -grams (probabilities of phrases with N tokens) is an $O(N!)$ problem because there are $N!$ permutations of N tokens.

2.3 The Naive Bayes Algorithm Applied to Text Classification

The general idea behind naive Bayes text classification is surprisingly simple. However, before going any further, we need to properly define a document. For our purposes, a document is a set of words separated by whitespace and/or punctuation. This document is usually stored as a file, but it could also be stored as a record in a database or streamed in from some network location.

Assume that we have a document archive which contains two or more classes of documents. Each word in an archive of documents has a probability of being in each class in the archive. An estimate of this probability is the number of times the word was found in the class divided by the total number of words in the class. By keeping a database of these probabilities, we can look up the word/class probabilities when we need to calculate the posterior probability.

For naive Bayes text classification, the feature vectors are the words and their counts from each document. This requires the feature vector to only be as long as the number of distinct words in the document that produced the vector. In contrast, many other classifiers may, depending on how they are trained, require all feature vectors to be

as long as the number of distinct words in the entire archive [2].

The equation for calculating the probability of a document belonging to a class is given by Bayes' Theorem:

$$P(class | words) = \frac{P(words | class) \times P(class)}{P(words)} \quad (2.2)$$

That is, the probability of a class owning a given document is the product of the probability of the occurrence of the document's words given the class and the probability of the class divided by the probability of the occurrence of the document's words. Since $P(words)$ is the same for every class, we can drop it from the equation. The prior or *class*¹ probability $P(class)$ is the number of documents in the given class divided by the number of documents in the entire archive. The probability of the occurrence of a set of independent words given a particular class $P(words | class)$ is the product of the probability of the occurrence of each word given the class:

$$P(words | class) = P(word_1 | class) \times P(word_2 | class) \times \dots \times P(word_N | class) \quad (2.3)$$

Recall from the previous section that this term is called the likelihood. After calculating the probabilities for each class, a typical classifier would simply choose the class with the highest probability. Leach works a little differently, as you will see in Section 2.6.

¹Since the prior probability is represented by the term $P(class)$, it is easier to just call it the "class probability".

2.4 Flaws in the Naive Bayes Algorithm

There are several problems with a straight implementation of the naive Bayes classifier. These are:

- Any word that does not already exist in the training data will have a probability of occurrence equal to zero, which will make the posterior probability zero.
- Scores are often so low they underflow even with double precision arithmetic.
- Accuracy is not optimal with a very large number of classes [8].

The first problem can be addressed by changing the method we use to estimate the likelihood. Instead of a straight Bayes theorem implementation, we change the method to the *m-estimate* which is described in detail in [2]. The m-estimate provides a way to retain a fairly accurate estimate of the probability of a word occurrence, but prevents the probability from becoming zero if the word does not exist in the class. The m-estimate is given by:

$$P(v_i | c_j) = \frac{o_{v_i c_j} + n_A p}{n_{c_j} + n_A} \quad (2.4)$$

That is, the probability of a word v_i occurring in class c_j is equal to the sum of the number of occurrences $o_{v_i c_j}$ of word v_i in class c_j and the product of the number of words in the archive n_A and prior estimate p divided by the sum of the number of words in the class n_{c_j} and the number of words in the archive. We do not know p , so we set it to $\frac{1}{n_A}$, which gives us an effective equation of:

$$P(v_i | c_j) = \frac{o_{v_i c_j} + 1}{n_{c_j} + n_A} \quad (2.5)$$

This ensures that the posterior probability, or “class score”, will always be a positive, non-zero value. The “class score” is just the posterior probability that we calculated using Bayes theorem, however, we have changed the nomenclature here in order to provide a better context for the discussion and to avoid confusing the posterior probability and the class probability that is used in Bayes theorem.

The second problem can be addressed by shifting the calculations into log-space. Instead of multiplying the likelihood terms together, we add the logs of the likelihood terms and rescale the class scores while they are still in log-space by adding the negative of the highest log value.

The third problem can be addressed by organizing the classes into a hierarchy. Previous work has shown that hierarchical classification can be more accurate overall than classifying documents using a flat class space [8, 9].

2.5 Hierarchical Classification

In a hierarchical class space, each class may contain other classes, forming a hierarchy. In contrast, a flat class space may only contain classes that contain documents.

Real life data can be, and usually is, organized into a hierarchical structure so that a given document can be found more quickly and easily. For example, any modern filesystem allows hierarchical data storage. The benefit of this method of storage is analogous to sorting a list of numbers so they may be searched using a binary search instead of a linear search. Even though the number of classes you have to go through is larger, the number of documents you must search at the end is much, much smaller. This is why Web site indexes are typically organized hierarchically by topic. Leach, the classifier we developed for this thesis, is a hierarchical classifier and can classify a

document into any hierarchy it is given. However, there is such a thing as over-fitting. It would not be beneficial in most cases to create a document hierarchy where classes had only one or two documents. Each class should have enough documents to provide a classifier with enough information to learn what documents to assign it.

2.6 Selection

Typically, a naive Bayes text classifier will simply assign the given document to the class with the highest score. Unlike other naive Bayes classifiers, Leach removes the most unlikely classes and then recalculates the class scores for each remaining class. The removal of the worst performing classes changes the number of words in the archive which changes the likelihood produced by equation (2.5). Remaining classes are removed again and again until only one class remains, which is the class with the highest probability of owning the document. This is the selection process, which is the contribution of this work to the field of text classification.

Because Leach is a hierarchical classifier, the selected class may have subclasses. Scores for those subclasses are also calculated and run through the same selection process until a class with no subclasses is chosen. The term “selection” comes from the selection process in a genetic algorithm, where only certain individual solutions are allowed to continue to the next generation.

Selection is not limited to use with naive Bayes classifiers. It should work on any classifier that computes a score for each class that is at least partially based on data that depends on the existence of other classes, i.e. the number of words or documents in the entire archive. A score that is computed from information that is entirely specific to a class will not change after the removal of other classes, and therefore the

classifier will not benefit from selection.

2.7 Leach Algorithm

The classification process begins by giving Leach a training directory that contains the top-level classes, which are themselves just directories. Leach uses the names of the directories as the names of the classes, and traverses each class looking for subclasses. When it has located all the subclasses, Leach gathers the files beneath each class and begins the scanning process. The scanning process tokenizes words, breaking on whitespace and punctuation. Many words which are common in English grammar do not provide useful information to a classifier [11]. These are words such as “the”, “it”, “of”, etc. These “stop words” are ignored by Leach. A complete list of stop words is provided in Appendix A. All numbers encountered by Leach are mapped to a single token, which becomes a measure of how “numerical” the document is. This can help if you have a class which contains a lot of number-heavy documents. Leach keeps a record of the number of times each word² is found in each class. This frequency distribution of word counts is the ultimate goal of the first stage. Although we could calculate the word probabilities next, it would not be helpful because later on we use the selection feature of Leach which requires word counts, not probabilities. Figure 2.1 shows the general flow of Leach’s first stage.

The second stage of Leach uses word counts to classify a given document. First, the document is scanned in the same way that the training files were. The word counts are extracted and a likelihood estimate is calculated in each class using equation (2.5). In order to prevent underflows, we take the log of the result of the equation

²The more proper term here would be token, since not all tokens parsed from the document will be words. However, the vast majority will be words, and it should help make the explanation a little clearer.

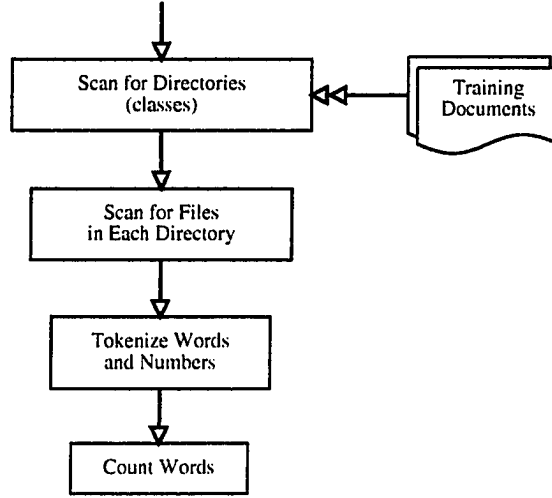


Figure 2.1: A flowchart of the first stage of Leach

and add it to a running total. This has the same effect as multiplying the results, but allows us to rescale the class score later to prevent underflows. After all the class scores have been calculated, the scores are rescaled by adding the negative of the highest score to each class score. For each class score, the log of the corresponding class probability is then added and the result is taken out of log space by taking the exponent. Mathematically, this is expressed as:

$$P(c_j | V) = \exp \left[\left(\sum_{i=1}^{|V|} \log \left(\frac{o_{v_i c_j} + 1}{n_{c_j} + n_A} \right) \right) + \log \left(\frac{f_{c_j}}{f_A} \right) \right] \quad (2.6)$$

The class scores are then sorted and the top 50% of the classes are kept. The classification process is then repeated until there is only one class remaining. The “keep” percentage is adjustable, however after much experimentation, keeping 50% of the classes turned out to be the best compromise between accuracy and speed. Beyond 50%, there was little improvement in accuracy. Figure 2.2 shows the general flow of Leach’s second stage.

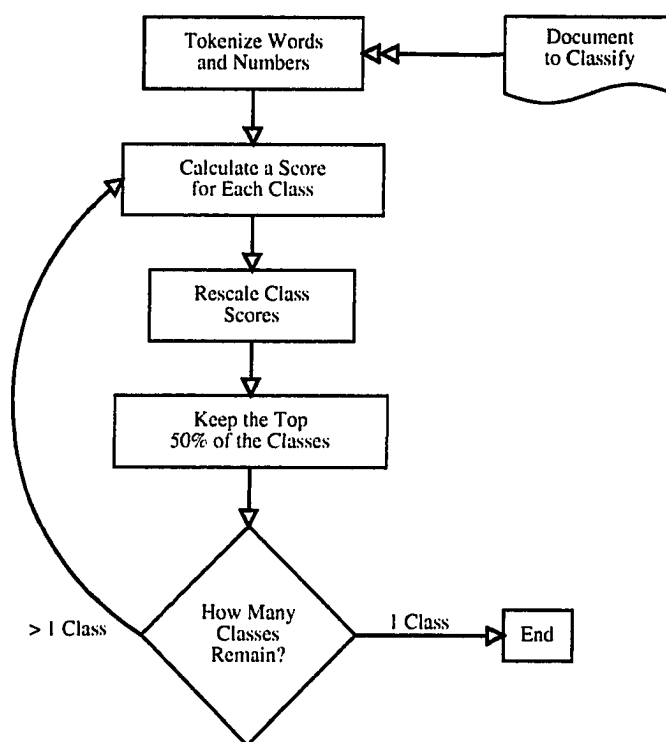


Figure 2.2: A flowchart of the second stage of Leach

2.8 An Example Run of Leach

To provide a clearer picture of Leach’s operation, we will go through a small example using two classes composed of one document each. We will call the first class “Bikes” and the second class “Photography”. Here is the single document from the Bikes class:

Although you have to *ride* on the *freeway* for a little while, the *ride* to *Verdi* is still very *pretty*, especially in the *spring* and *fall*. In the *height* of *summer*, it gets a little *hot* unless you go very *early* in the *day*.

And here is the single document from the Photography class:

Even though I *prefer* to *shoot wildlife*, I still *enjoy* doing *portrait sessions* with *friends* or *family* that need some *nice pictures* for the *mantle*.

The emphasized words in each document are the words that were not removed by the stop word filter. You can see that the stop word filter removes many words.

After tokenization, the words are counted and made into a frequency distribution. Since we are only using one document for each class, the frequency distribution for each class will be the same as the frequency distribution for the the corresponding document. Normally, the frequency distributions for the classes would be a combination of all of the documents in each class, but in this simple example, they will be the same. Table 2.1 shows what the frequency distributions for both classes would look like. This is the end of Leach's first stage. The data we have generated can be saved to a file for future use.

Table 2.1: Frequency distributions for the two example documents.

Bikes	Count	Photography	Count
ride	2	prefer	1
freeway	1	shoot	1
verdi	1	wildlife	1
pretty	1	enjoy	1
spring	1	portrait	1
fall	1	sessions	1
height	1	friends	1
summer	1	family	1
hot	1	nice	1
early	1	pictures	1
day	1	mantle	1

The second stage begins by reading an input document and creating a frequency distribution as in the first stage. Here is our input document (the document we want to classify):

Once on a *ride* through the *woods* near *Truckee*, I saw a *deer* in the *shadows* of a *tree*, *taking a break* from the *hot sun*.

Table 2.2 shows the word frequency distribution for our input document.

Table 2.2: Frequency distribution for the input document.

Input	Count
ride	1
woods	1
truckee	1
deer	1
shadows	1
tree	1
taking	1
break	1
hot	1
sun	1

Now, we are ready to begin calculating the class scores. Recall from Section 2.4 that likelihood is calculated using the m-estimate:

$$P(v_i | c_j) = \frac{o_{v_i c_j} + 1}{n_{c_j} + n_A}$$

Plugging this into equation (2.2), we get:

$$P(class | words) = \prod_{i=1}^{|V|} \left(\frac{o_{v_i c_j} + 1}{n_{c_j} + n_A} \right) \times P(class)$$

And after converting $P(class | words)$ and $P(class)$ to mathematical notation, we get:

$$P(c_j | V) = \prod_{i=1}^{|V|} \left(\frac{o_{v_i c_j} + 1}{n_{c_j} + n_A} \right) \times \frac{f_{c_j}}{f_A} \quad (2.7)$$

Let us go through each term to clarify their meanings.

- c_j – the class for which we are calculating the score.
- V – the feature vector of the input document. $|V|$ is the number of words in the input document.
- $o_{v_i c_j}$ – the number of occurrences of the word v_i in the class c_j .
- n_{c_j} – the total number of words in class c_j .
- n_A – the number of words in the archive.
- f_{c_j} – the number of documents in class c_j .
- f_A – the number of documents in the archive.

As explained in Section 2.7, Leach uses logs and rescaling to prevent underflows. However, we do not need that complexity for our simple example, so we just use equation (2.7) to calculate the class scores.

Since we only have two classes, we only have two class scores to calculate. Let us start with the Bikes class. The only words that match in the Bikes class are “ride” and “hot”. The word “ride” occurred twice in the Bikes class and “hot” occurred once. The likelihood for these words will be as follows:

$$\frac{o_{v_i=\text{ride}, c_j=\text{Bikes}} + 1}{n_{c_j=\text{Bikes}} + n_A} = \frac{2 + 1}{11 + 22} = \frac{3}{33} = 0.\overline{09}$$

$$\frac{o_{v_i=\text{hot}, c_j=\text{Bikes}} + 1}{n_{c_j=\text{Bikes}} + n_A} = \frac{1 + 1}{11 + 22} = \frac{2}{33} = 0.\overline{06}$$

The other eight words in the input document are not present in the class. Therefore, they all end up being:

$$\frac{0+1}{11+22} = \frac{1}{33} = 0.\overline{03}$$

Multiplying all of these together, we get a likelihood of:

$$0.\overline{09} \times 0.\overline{06} \times 0.\overline{03}^8 \approx 3.92 \times 10^{-15}$$

Now the only remaining step is to multiply by the class probability:

$$3.92 \times 10^{-15} \left(\frac{f_{c_j}}{f_A} \right) = 3.92 \times 10^{-15} \left(\frac{1}{2} \right) = 1.96 \times 10^{-15}$$

For the Photography class, no words match, so we end up with a likelihood of:

$$0.\overline{03}^{10} \approx 6.53 \times 10^{-16}$$

and a class score of:

$$6.53 \times 10^{-16} \left(\frac{1}{2} \right) = 3.265 \times 10^{-16}$$

At this point, Leach would sort the classes by their class scores and remove half of the classes with the lowest scores, then repeat the process for the remaining classes. In this case, we only need to remove one class with the lowest score, namely the Photography class. We remove it, which leaves the Bikes class as the class that most closely matches the input document. This makes sense, as the input document is obviously about a bike ride and has nothing to do with photography.

Chapter 3

Experimental Setup

To test Leach, we selected the 20 Newsgroups data set [12]. The 20 Newsgroups data set is commonly used for testing text classifiers, and is a compilation of 20 Internet newsgroups, each with approximately 1,000 postings. The newsgroups cover a broad range of subjects, with some groups' subjects overlapping others while some are unique.

Since Leach is a hierarchical classifier, we organized the 20 Newsgroups data set into a two level hierarchy. We attempted to combine the groups into logical classes based on their topic. Most groups fit neatly into a top-level class. For example, all of the comp.* groups were put into a top-level class called Computers. However, another group, sci.electronics, was put into a top-level class called Science even though some of its discussion was related to computer topics. The misc.forsale group did not fit anywhere else, so we created it's own top-level class called Other. Table 3.1 shows the top-level classes, the subclasses, and the number of files in each subclass.

The 20 Newsgroups data set has headers which show meta-information about the message, including the name of the newsgroup to which the message was posted. We

Table 3.1: How we made the 20 Newsgroups data set hierarchical

Class	Subclass	# Files
Autos	rec.autos	998
Autos	rec.motorcycles	997
Computers	comp.graphics	997
Computers	comp.os.ms-windows.misc	992
Computers	comp.sys.ibm.pc.hardware	997
Computers	comp.sys.mac.hardware	996
Computers	comp.windows.x	998
Other	misc.forsale	991
Politics	talk.politics.guns	1000
Politics	talk.politics.mideast	1000
Politics	talk.politics.misc	999
Religion	alt.atheism	999
Religion	soc.religion.christian	997
Religion	talk.religion.misc	1000
Science	sci.crypt	999
Science	sci.electronics	999
Science	sci.med	998
Science	sci.space	999
Sports	rec.sport.baseball	999
Sports	rec.sport.hockey	998

removed the headers to prevent them from skewing the results. After removal of the headers, some of the files in the 20 Newsgroups data set were either empty or non-text and were removed.

The data set was divided into training and testing sets, with 30% of the messages being used for training and 70% used for testing. The files for training and testing were selected randomly. Ten different 30%/70% splits were used, and the scores were averaged over all splits to ensure that the scores were representative of the average case.

Even though we could not locate any pre-existing, freely-available classifiers that also performed hierarchical classification, we decided to use Rainbow, a well-known

naive Bayes classifier [13]. Rainbow was able to perform classification using only the top-level classes, which provides a useful comparison to Leach's ability to classify documents into the top-level classes.

We ran Rainbow with its default set of options, which is comparable to the options Leach uses. These include removing stop words and not using word stemming. The stop words for both classifiers are identical, as we copied Rainbow's stop words when developing Leach. Stemming is a technique for reducing the number of words that the classifier must deal with by reducing each word to its root form. For example, the words "insuring", and "insured" would be reduced to the word "insure". Stemming is also sometimes referred to as lemmatization.

In the next chapter, we present the results of the experiments and analyze the differences between running Leach while not using selection and running Leach while using selection. We also compare Leach's scores to Rainbow's scores and discuss possible reasons for differences.

Chapter 4

Results

The overall results show a slight, but statistically significant improvement in classification accuracy when using selection versus not using selection. For a point of reference, we have included scores for Rainbow. Since Rainbow does not do hierarchical classification, it only has scores for the top-level classes. Overall, Rainbow is a better classifier, though Leach was able to win a couple of classes and stay close in several others. All claims of improvement in this section are statistically significant with 95% certainty.

Leach's scores for the Autos classes were poor compared to Rainbow's, but the Autos class showed a strong 7.7% increase in accuracy using selection. Looking at a sample of misclassified documents, we noticed that a few were off-topic, a few could be considered ambiguous, and a smaller number were on-topic. Many were misclassified into the Computers and Science classes, which are the largest classes. This may indicate that the class probabilities played a role in the misclassification, since the class probability is the ratio between the number of documents in the class and the number of documents in the archive. This would indicate that Leach had trouble

generating a likelihood large enough to overcome the smaller class probability of the Autos class. Table 4.1 shows the results for the Autos classes.

Table 4.1: Results for the Autos classes

Top-level class	Subclass	No Selection	Selection	Rainbow
Autos	Overall	38.6	46.3	90.3
	rec.autos	29.3	36.4	
	rec.motorcycles	46	53.9	

Leach posted an excellent score for the top-level Computers class, beating Rainbow easily. However, selection did not increase performance at all. This is understandable, though, as it would be difficult to show any significant improvement in the already high score. The subclasses had much lower scores, indicating that Leach had a much harder time differentiating between them. The only subclass that showed a significant increase was comp.sys.ibm.pc.hardware. The comp.sys.mac.hardware class showed a 4% difference, but the variability in the sample scores was too high to consider it a significant increase. Table 4.2 shows the results for the Computers classes.

Table 4.2: Results for the Computers classes

Top-level class	Subclass	No Selection	Selection	Rainbow
Computers	Overall	95.3	95.3	91.1
	comp.graphics	75.3	75.6	
	comp.os.ms-windows.misc	55.7	48.4	
	comp.sys.ibm.pc.hardware	68.4	73.7	
	comp.sys.mac.hardware	60.3	64.3	
	comp.windows.x	78.6	79.6	

The Other class was even worse than Autos, with a correct classification being a rare event. Rainbow also performed uncharacteristically poorly on this class, but not nearly as bad as Leach. We would surmise that the reason for the poor performance

is due to two factors. First, the Other class is small compared to the other classes. It only contains one subclass, whereas others contain up to five. This decreases the class probability which hurts the class scores. Second, Other contains the misc.forsale group which is populated by ads from individuals selling everything from cars to computer manuals. In short, there is little to no consistency in the group, which makes it difficult for Bayesian classifiers to handle. In addition, many of the items for sale overlap with other groups in the data set (motorcycles, car parts, computer equipment), which would easily fool a Bayesian classifier. A small victory here is that selection improved the score. Table 4.3 shows the results for the Other class.

Table 4.3: Results for the Other class

Top-level class	Subclass	No Selection	Selection	Rainbow
Other	misc.forsale	1.2	2.0	33.4

Like the Computers class, Leach was very good at correctly classifying documents into the Politics class. However, unlike the Computers class, Leach's selection improved the score, making it higher than Rainbow's. Results for the subclasses were mixed, with talk.politics.mideast showing a high score and a significant improvement using selection while talk.politics.guns and talk.politics.misc posted mediocre scores and no improvement using selection. Table 4.4 shows the results for the Politics classes.

Table 4.4: Results for the Politics classes

Top-level class	Subclass	No Selection	Selection	Rainbow
Politics	Overall	92.8	94.6	92.8
	talk.politics.guns	79.2	79.7	
	talk.politics.mideast	91.7	94.3	
	talk.politics.misc	73.3	74.5	

Leach did a good job classifying the Religion class and it showed improvement us-

ing selection, but still fell behind Rainbow. On the surface, the Religion subclasses appear to be very difficult to differentiate. All subclasses are heavily if not totally biased towards the discussion of Christianity, with other religions such as Islam and Buddhism mentioned only sparingly, even in `talk.religion.misc`. However, Leach did a very good job at classifying the `soc.religion.christian` class and showed an improvement using selection. The scores for the `alt.atheism` class were mediocre and the scores for `talk.religion.misc` were poor. Table 4.5 shows the results for the Religion classes.

Table 4.5: Results for the Religion classes

Top-level class	Subclass	No Selection	Selection	Rainbow
Religion	Overall	79.8	83	89.2
	<code>alt.atheism</code>	69.6	71.3	
	<code>soc.religion.christian</code>	84.1	87.7	
	<code>talk.religion.misc</code>	30.6	32.1	

Like the Religion class, the Science class attained a very good score, but still fell a little short of Rainbow. Unlike the Religion class, Leach was not able to improve the overall score using selection. With the exception of `sci.electronics`, Leach did a good job classifying the Science subclasses. This is not too surprising, since the subclasses are all fairly unique. Only `sci.crypt` showed a statistically significant improvement. The performance on `sci.med` actually decreased with selection. The rest had too much variability to qualify for a significant increase. Table 4.6 shows the results for the Science classes.

The Sports class showed a strong improvement using selection, however, it was still short of Rainbow's excellent score. Of the subclasses, Leach did a good job with `rec.sport.hockey`, but surprisingly poorly on `rec.sport.baseball`. After an examination of Leach's raw output, it appears Leach misclassifies many documents from

Table 4.6: Results for the Science classes

Top-level class	Subclass	No Selection	Selection	Rainbow
Science	Overall	83.1	84	89.8
	sci.crypt	84.2	85.3	
	sci.electronics	53	56.1	
	sci.med	84.2	83.9	
	sci.space	86.7	87.7	

rec.sport.baseball into rec.sport.hockey. Looking at the misclassified documents, there are several plausible reasons for the misclassifications. First, some people had signature lines with the names of several different sports teams' names on them, including the names of hockey teams and other hockey-type references. For example, one signature referred to Candlestick Park in San Francisco, the old home of the Giants baseball team, as "The Stick". A stick is the main tool of a hockey player and as such, Leach probably weighs the word "stick" very heavily towards hockey. Second, many of the misclassified documents contained many numbers, usually from someone posting statistics about a team or league. Leach maps all numbers to a single token, so if the training documents in the hockey group contained more numbers than the baseball group, that would push the score towards the hockey side. Table 4.7 shows the results for the Sports classes.

Table 4.7: Results for the Sports classes

Top-level class	Subclass	No Selection	Selection	Rainbow
Sports	Overall	80.4	84.3	95.4
	rec.sport.baseball	68.2	71	
	rec.sport.hockey	85.4	89.3	

Table 4.8 shows the Overall results from our experiment. Of the seven top-level classes, Leach's selection improved the accuracy of five of them. Of the twenty sub-

classes, Leach's selection improved nine of them. The fact that selection improved the scores for the subclasses a smaller percentage of the time was not unexpected, because the number of subclasses was always smaller than the number of top-level classes, and the strength of selection is its ability to narrow down the number of classes by removing the low-scoring classes.

Rainbow is clearly a better overall classifier in a flat class space, and, if extended, would probably be better in a hierarchical class space also. There are several important differences between Rainbow and Leach that may account for Rainbow's higher overall score. One difference between the two classifiers is how they perform the tokenization step. Rainbow uses its own method of tokenizing text while Leach relies on the Java class `STREAMTOKENIZER` to perform its tokenization. Even though Leach and Rainbow should extract very similar words and word counts, subtle differences in tokenization are possible. These differences can include whether hyphenated words are split, how underscores are handled, and whether or not other punctuation is kept and mapped to input tokens. Though these differences should be small, it is possible they could affect the classification of some documents. Another difference is that Rainbow performs a lot of "data massaging" that Leach does not do. This is a result of years of work that has gone into Rainbow by many different people. Leach was created for the purpose of testing selection on flat and hierarchical class spaces. Even with these differences, Rainbow should also benefit from selection if it were implemented properly.

In the next chapter, we summarize our work, our findings, and discuss several ideas for extensions of this work.

Table 4.8: Overall Results

Top-level class	Subclass	No Selection	Selection	Rainbow
Autos	Overall	38.6	46.3	90.3
	rec.autos	29.3	36.4	
	rec.motorcycles	46	53.9	
Computers	Overall	95.3	95.3	91.1
	comp.graphics	75.3	75.6	
	comp.os.ms-windows.misc	55.7	48.4	
	comp.sys.ibm.pc.hardware	68.4	73.7	
	comp.sys.mac.hardware	60.3	64.3	
	comp.windows.x	78.6	79.6	
Other	misc.forsale	1.2	2.0	33.4
Politics	Overall	92.8	94.6	92.8
	talk.politics.guns	79.2	79.7	
	talk.politics.mideast	91.7	94.3	
	talk.politics.misc	73.3	74.5	
Religion	Overall	79.8	83	89.2
	alt.atheism	69.6	71.3	
	soc.religion.christian	84.1	87.7	
	talk.religion.misc	30.6	32.1	
Science	Overall	83.1	84	89.8
	sci.crypt	84.2	85.3	
	sci.electronics	53	56.1	
	sci.med	84.2	83.9	
	sci.space	86.7	87.7	
Sports	Overall	80.4	84.3	95.4
	rec.sport.baseball	68.2	71	
	rec.sport.hockey	85.4	89.3	
Total	Top-level classes	65.8	68	83.1
Total	Subclasses	65.2	67.3	

Chapter 5

Conclusions and Future Work

In this thesis, we presented a new technique called selection which improved the performance of a naive Bayes classifier. We implemented selection in a classifier called Leach which we tested against Rainbow, a well-known classifier. As our experiments have shown, the process of selection improves the performance of a naive Bayes classifier in a hierarchical class space as well as a flat class space. Selection incurs a speed penalty, however, it is often worth the extra time for the additional accuracy.

Although the purpose of this thesis was to demonstrate selection, Rainbow was run to provide a context for Leach's scores. With a couple exceptions, Leach's performance was close to Rainbow's. An interesting result from the experiments is that Leach is able to occasionally beat Rainbow. As we noted before, Rainbow is a very mature program and is very fine-tuned. Although this tuning helps the overall score, it is not always optimal. Since the source code for Rainbow is freely available, we experimented with some of the additional things Rainbow does to improve class scores. We found that some of them increase the accuracy on classes with a typically low score, but sometimes decrease the performance on classes with a typically high score. While this

trade-off may be beneficial in the overall case, it may explain why Rainbow sometimes falls a little behind Leach.

The data set used can also significantly affect the performance of a classifier. Rainbow tends to perform very well with the 20 Newsgroups data set, but not as well with other data sets, such as Reuters-21578 [4]. The difference could indicate that Rainbow was developed using 20 Newsgroups as the test data set, or that there is something about Rainbow that is particularly well-suited to the 20 Newsgroups data set. This could be related to the independence assumption, or to the numerical "massaging" that Rainbow performs.

One thing we noticed while examining misclassified documents was that large documents, even if they were on topic, were sometimes misclassified. For the future, it would be interesting to find out why this occurs.

Also, it would be interesting to change the prior estimate p in equation (2.4) after the first round of selection has taken place. Since the previous value has a good chance of being higher than $\frac{1}{n_A}$, this may provide the class with a better score than it would otherwise receive.

Currently, Leach only attempts to assign a given document to one class. For certain tasks, it would be beneficial to assign a document into multiple classes. For example, many documents in the sci.electronics class would also be appropriate in the comp.sys.ibm.pc.hardware class. Leach could be modified to always return the top two classes, or one class normally and two classes if the difference between their scores is small.

If Leach makes a misclassification at the top-level, it has no way of recovering. Devising a reasonably accurate method for error recovery would be a very interesting task.

Appendix A

List of Stop Words

a, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, around, as, aside, ask, asking, associated, at, available, away, awfully, b, be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by, c, came, can, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, course, currently, d, definitely, described, despite, did, different, do, does, doing, done, down, downwards, during, e, each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, exactly, example, except, f, far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore, g, get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings, h, had, happens, hardly, has, have, having, he, hello, help, hence, her, here, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however, i, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates, inner, insofar, instead, into, inward, is, it, its, itself, j, just, k, keep, keeps, kept, know, knows, known, l, last, lately, later, latter, latterly, least, less, lest, let, like, liked, likely, little, look, looking, looks, ltd, m, mainly, many, may, maybe, me,

mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself, n, name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere, o, obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own, p, particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides, q, que, quite, qv, r, rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right, s, said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, since, six, so, some, somebody, somehow, someone, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure, t, take, taken, tell, tends, th, than, thank, thanks, thanx, that, thats, the, their, theirs, them, themselves, then, thence, there, thereafter, thereby, therefore, therein, theres, thereupon, these, they, think, third, this, thorough, thoroughly, those, though, three, through, throughout, thru, thus, to, together, too, took, toward, towards, tried, tries, truly, try, trying, twice, two, u, un, under, unfortunately, unless, unlikely, until, unto, up, upon, us, use, used, useful, uses, using, usually, uucp, v, value, various, very, via, viz, vs, w, want, wants, was, way, we, welcome, well, went, were, what, whatever, when, whence, whenever, where, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, whoever, whole, whom, whose, why, will, willing, wish, with, within, without, wonder, would, would, x, y, yes, yet, you, your, yours, yourself, yourselves, z, zero

Bibliography

- [1] Peter Lyman and Hal R. Varian. How much information, 2003. <http://www.sims.berkeley.edu/research/projects/how-much-info-2003/>.
- [2] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [3] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.
- [4] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US.
- [5] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.
- [6] The Mozilla Foundation. Mozilla.org, 2003. <http://www.mozilla.org/>.

- [7] Spamassassin.org, 2003. <http://spamassassin.org/>.
- [8] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 170–178, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.
- [9] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.
- [10] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. AAAI-98 Workshop on Learning for Text Categorization, 1998. <http://citeseer.nj.nec.com/mccallum98comparison.html>.
- [11] G. Salton and M.J. McGill. The smart and sire experimental retrieval systems. In K. Sparck Jones and P. Willet, editors, *Readings in Information Retrieval*, pages 381–399. Morgan Kaufmann Publishers, Inc., San Francisco. 1997.
- [12] J. Rennie. The 20 newsgroups data set, a collection of approximately 20,000 newsgroup documents. <http://www.ai.mit.edu/people/jrennie/20Newsgroups/>, 1997.
- [13] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. <http://www.cs.cmu.edu/~mccallum/bow/>, 1996.