Utilizing HTML Structure and Linked Pages to Improve Learning for Text Categorization

Michael Cline
Department of Computer Sciences
University of Texas
Austin, Texas
mike_cline@mail.utexas.edu

Undergraduate Honors Thesis
Advisor: Dr. Raymond Mooney, Ph. D.

May, 1999

1 Introduction

Since the Birth of the World Wide Web, the amount of information available on the Web has been increasing at an exponential rate. This growth has created the need for indexes which separate documents into categories based on their subject, such as the popular web site *Yahoo!*. The extraordinary amount of information needing to be organized is a motivation to look for intelligent methods of automatically categorizing documents. Since it would be difficult and tedious to build effective automatic categorizers by hand, and because experimental data is readily available, this problem is an interesting domain for the application of machine learning techniques.

Although there are a variety of machine learning algorithms that could be applied to this problem, this paper focuses on one method known as the naive Bayes classifier, a very simple and effective method of text categorization based on probability theory. The idea behind the naive Bayes classifier is straightforward: we can take a set of documents which are known to be in particular categories and observe the frequencies with which certain words appear in each category. Then, using these frequencies, we can estimate the probability of some new document being in each of these categories. The algorithm makes no attempt at true understanding of the document, since it does not even take into account the order that words appear in. In spite of this, and in spite of the fact that the algorithm makes simplifying assumptions which are not generally true in practice, naive Bayes delivers surprising accuracy which is often comparable to much more sophisticated and intelligent methods of text categorization (Pazzani and Billsus, 1997).

Traditionally, text categorization research has focused on straight-text documents, and ignored the additional information that is provided in the way an HTML document is structured. This paper focuses on the question of whether we can use the additional information such as hypertext links and formatting codes to improve the performance of a traditional naive Bayes text classifier.

In the typical approach to text categorization, each document is represented as a either the set or the bag of words in the document (in the "set" of words, no duplicate words are represented, whereas in the "bag" representation, duplicate words are stored). The HTML formatting codes give us a way to add more structure to the representation of the document. Some words are marked as the document's title, some words are marked as paragraph headings, some word are marked as links to other pages, and so on. It is reasonable to believe that a word from the title of a document would carry more information about the subject of the document than a word from the body. Similarly, we can hypothesize that the words contained in the links and headings of an HTML document may be more representative of the subject of the document than words in the body. If this is correct, our hope is that the naive Bayes algorithm will pick up on this and be able to use this extra information to improve accuracy. This is the first idea that this paper explores. Our results indicate that this approach can benefit the accuracy of the classifier, but the improvements did not show up in all cases, and were relatively small even in the best situations.

Another hypothesis we investigate in this paper is that HTML pages tend to be linked to other pages similar in subject. Therefore, if our representation for a document X includes the text of

documents that are pointed to by hypertext links in *X* (the text of the "children" of *X*), we would expect to be able to categorize more accurately. This experiment gave us some promising results, demonstrating that the accuracy of the classifier increased as we increased the number of child pages that we included in the representation of the parent.

2 Algorithm

2.1 Bayes' Formula

Probability theory tells us that for two events *E* and *F*, the probability of them both happening is equal to the probability of *E* given *F* is true, times the probability that *F* is true:

$$P(E \cap F) = P(E \mid F) \cdot P(F) \quad (1)$$

Since $P(EF) = P(FE)$, it follows that

$$P(E \cap F) = P(F \mid E) \cdot P(E) \quad (2)$$

Substituting equation 2 into equation 1 and dividing both side by *P(F)* yields Bayes' Formula:

$$P(E \mid F) = \frac{P(F \mid E) \cdot P(E)}{P(F)}$$

How is this rule useful in categorization problems? Imagine that we have some object *X* that we wish to categorize into to one of n different categories based on some attribute or set of attributes *A* of *X*. Let $P(X \in C_i)$ denote the probability that *X* is in category C_i. The goal is to decide which category is the one *X* most likely belongs to, that is, the category C_i which maximizes $P(X \in C_i \mid A)$. Bayes formula is useful because it allows us to calculate

P(X is in category C given X has attributes A)
in terms of
P(X has attributes A given X is in category C).

This second probability can be approximated by looking at a set of example objects ("training examples") which have already been categorized. For instance, imagine we are trying to categorize object *X* as either a fruit or a vegetable, given that *X* is green. Let *V* be the set of vegetables. Bayes formula tells us that

$$P(X \in V \mid green(X)) = \frac{P(green(X) \mid X \in V) \cdot P(X \in V)}{P(green(X))}$$

if we are given a set of fruits and vegetables and their respective colors as training examples, we can easily estimate the $P(\ green(X) \mid X \in V\ )$ by dividing the number of green vegetables in the training set by the total number of vegetables in the training set. Likewise, *P(X is vegetable)* and *P(X is green)* can be estimated by finding the fractions of vegetables and green objects in the set of training examples.

After we have estimated the probabilities of *X* being and fruit and *X* being a vegetable, all we need to do is compare these values to decide which is the most likely category.

How can we extend this idea to allow us to categorize a document based on its contents? One way is to define a vocabulary $\{w_1, w_2, w_3, w_4 \ldots\}$, where w_n are words, and let each document's attributes be a set of boolean values that determine whether or not each word in the vocabulary appears in that document.

A problem arises when using the categorization scheme as I have described it: what happens when we want to categorize an object based on a conjunction of a large number of attributes? As the number of different attributes gets larger, the chances of having a training example with the same attributes as the document being classified grows smaller very quickly. Returning to the vegetable and fruit example, what happens if we are not just classifying the objects based on their color, but also on the shape, dimensions, taste, geographic origin, and a variety of other attributes? Then the chances of finding a training example that matches the attributes of the mystery object (the object being classified) are very small.

This is also the case in text categorization, where the attributes of a document are the words in the document, and therefore each document has a large set of attributes. We need a way of looking at each attribute of the mystery object individually and considering its contribution to the categorization individually, and then combining these contributions into an overall guess for the most likely category of the mystery object.

2.2 Naive Bayes Classifier

The solution to the problem mentioned above is to adopt the assumption of conditional independence between the attributes of our uncategorized document, *X*. Two events *E* and *F* are conditionally independent with respect to some other event G when

$$P(EF \mid G) = P(E \mid G) \cdot P(F \mid G)$$

Or, equivalently,

$$P(E \mid FG) = P(E \mid G) \quad \textit{and} \quad P(F \mid EG) = P(F \mid G)$$

In a naive Bayes classifier, we assume that given any category, the attributes of an object are conditionally independent with respect to the event that the object belongs in that category.

In the context of a text categorization problem, E and F would be the event that specific words appear in a document, and G would be the event that the document belongs to some given category. It is obviously false to make this assumption in a text categorization problem (hence the name "naive Bayes"). It would mean, for example, that given some document that we knew to be in a "music" category, the word "Elvis" is no more likely to appear in the document that contains the word "Presley" than in one that does not. Although this conditional independence assumption is clearly false, it allows us to use a sparse training set and still achieve surprisingly accurate results in our categorizations.

Now, recall that the most probable category *C* for document *X*, where $\{a_1, a_2, a_3 \ldots\}$ are the boolean attribute values, is given by that category that maximizes $P(X \in C \mid a_1, a_2, a_3, \ldots)$

$$C = \underset{Ci \in \{C1 \ldots Cn\}}{\arg\max} \; P(X \in Ci \mid a1, a2, a3, a4 \ldots)$$

By Bayes' Formula,

$$\text{MAP} = \arg\max_{Ci \in \{C1 \ldots Cn\}} \frac{P(a1, a2, a3, a4 \ldots \mid X \in Ci) \cdot P(X \in Ci)}{P(a1, a2, a3, a4 \ldots)}$$

Since the denominator $P(a_1,a_2,a_3,a_4...)$ is the same for all C_i,

$$\text{MAP} = \arg\max_{Ci \in \{C1 \ldots Cn\}} P(a1, a2, a3, a4 \ldots \mid X \in Ci) \cdot P(X \in Ci)$$

Now, using the assumption that the attributes are conditionally independent given $X \in C$, this simplifies to

$$\text{MAP} = \arg\max_{Ci \in (C1 \ldots Cn)} P(X \in C) \cdot \prod_i P(ai \mid X \in C)$$

2.3 Bernoulli vs. Binomial Event Model

There are two commonly used methods for representing text data in the naive Bayes classifier. The first model, which I have already described, specifies that a document is represented by a vector of binary attributes indicating which words appear and which words do not appear. This type of description is called a *multi-variate Bernoulli* event model. This is the traditional approach used in the field of Bayesian networks (McCallum and Nigam, 1998). This model is equivalent to representing a document as the set of words in the document, meaning that the number of repetitions of the word is not captured. Intuitively, one would think that the number of times each word appears in a document is relevant in a categorization problem, so leaving this information out of our representation of the data seems like a bad idea.

The second model views each word occurrence as an event, and the document as a collection of events. This model is called the *multinomial* event model. This model is implemented by keeping the "bag" of words in the document. A bag is a type of mathematical container similar to set, except that duplicate elements are allowed. This is the model used in the experiments mentioned in this paper. Empirical evidence has shown that the multinomial event model usually outperforms the multi-variate Bernoulli event model in text categorization problems where the vocabulary is large, and almost always outperforms the multi-variate Bernoulli when the vocabulary size is chosen optimally for both (McCallum and Nigam, 1998).

2.4 Algorithm Specifics and Pseudo Code

Our algorithm uses two main functions: the first is `train-bayes` which takes a list of training examples as input and outputs a data structure containing the conditional probabilities of each word appearing in each category, as well as the prior probabilities of each category. The second is `test-bayes` , which takes as input a test example and the output from `train-bayes`, and outputs the name of the most likely category for the test example.

Each training examples consists of a category name and a list of lists of words. There is one list of words for each feature that we are using in the classification. In our first experiment, we are using a multi-feature data representation, meaning we build separate lists for the title-words, the heading-words, and the link text words, as well as one large list of all words in the document.

The data structure output by `train-bayes` consists of two vectors, `class-priors` and `feature-table-vector`. The first is a vector of prior probabilities that a document is each category. The `feature-table-vector` is a vector of hash tables, one for each feature in the data set. The hash tables are functions from words to vectors of conditional probabilities. So, for some word *w* and some feature *f*,

$$\textit{feature-table-vector[f](w)} = (P_1, P_2, P_3, \ldots P_n)$$

where P_i is the probability that given a document *x* known to be in category *Ci*, we could pull some random word out of its bag of words for feature *f* and that word would be *w*.

The training algorithm is made up of four steps:

1. Count the number of training examples in each category
2. Using these counts, construct a vector of prior probabilities for each category.
3. For each word, find the number of times that word occurs in a given feature for documents in each category.
4. For each feature, for each word in that feature, for each category, find the conditional probabilities of drawing this word randomly out of a document's bag of words for this feature given that the document is in this category

---

CALCULATE-CLASS-COUNTS (**examples, num-categories**)
```
let class-counts be an array of size num-categories
initialize each entry in class-counts to zero
for each example in examples
    {
    let c be the category of example
    increment class-counts[c]
    }
return class-counts
```

CALCULATE-CLASS-PRIORS (**class-counts, num-examples, num-categories**)
```
let class-priors be an array of size num-categories
for category from 1 to num-categories
    { class-priors[category] = class-counts[category] ÷ num-examples }
return class-priors
```

CALCULATE-CONDITIONAL-COUNTS (**examples**, **num-categories**, **num-features**)
```
let feature-table-vector be a vector of hash tables, one for each feature
let bag-sizes-vector be an vector of vectors, one for each feature
for feature from 1 to num-features
    {
    let cat-count-vector be an array of size num-categories
    let counts-hash-table be an empty hash table from words to floating point values
    initialize each entry in cat-count-vector to zero
    for each example in examples
        {
        let word-bag be example's bag of words for this feature
        let c be the category that this example belongs to
        for each word in word-bag
            {
```

```
                if counts-hash-table(word) is defined, then let word-category-vector be
                  counts-hash-table(word)
                else, let word-category-vector be an array of size num-categories,
                  initialized to zero
                increment cat-count-vector[c]
                increment word-category-vector[c]
                store word-category-vector in the counts-hash-table as counts-hash-
                  table(word)
                }
            }
        feature-table-vector[feature#] = counts-hash-table
        bag-sizes-vector[feature#] = cat-counts-vector
        }
    return (feature-table-vector, bag-sizes-vector)

CALCULATE-CONDITIONAL-PROBS (feature-table-vector, bag-sizes-vector, num-cats, num-
  features)
    for f = 1 to num-features
        {
        let hash-table be feature-table-vector[f]
        let bag-sizes = bag-sizes-vector[f]
        for each word with an entry in hash-table
            {
            let category-vector = hash-table(word)
            for c = 1 to num-cats
                {
                category-vector[c] = category-vector[c] ÷ bag-sizes[c]
                }
            }
        }
    return updated version of feature-table-vector

TRAIN-BAYES (examples)
    let num-cats be the number of categories in the examples
    let num-examples be the number of examples
    let num-features be the number of features in the examples
    class-counts = calculate-class-counts(examples,num-cats)
    class-priors = calculate-class-priors(class-counts,num-examples, num-cats)
    (feature-table-vector, bag-sizes-vector) = calculate-conditional-counts(examples,
      num-cats, num-features)
    feature-table-vector = calculate-conditional-probs (feature-table-vector, bag-sizes-
      vector, num-cats, num-features)
    return (class-priors,feature-table-vector)
```

The algorithm for `test-bayes` is fairly simple; it is just a matter of applying the formula for the naïve Bayes' clasifier to a document using the conditional probabilities collected by `train-bayes`.

$$C = \arg\max_{C_i \in (C_1 \ldots C_n)} P(X \in C) \cdot \prod_j P(a_j \mid X \in C_i) \qquad \text{(formula for naive Bayes classifier)}$$

$$= \arg\max_{C_i \in (C_1 \ldots C_n)} P(X \in C) \cdot \prod_{f \in Features} \prod_i^{|d_f|} P(w_{if} \mid X \in C_i)$$

(where $|d_f|$ is the number of words in X's bag for feature f, and w_{if} is the ith word in the document's bag for feature f)

$$= \arg\max_{C_i \in (C_1 \ldots C_n)} P(X \in C) \cdot \prod_{f \in Features} \prod_i^{|d_f|} feature-table-vector[f](w_{if})[C_i]$$

TEST-BAYES (**example**, (**class-priors**, **feature-table-vector**))
    let **probs** be an array of probabilities, one for each category
    initialize **probs** with **class-priors**
    for each feature **f** in example
        {
        let **hash-table** be **feature-table-vector**[**f**]
        let **word-bag** be **example**'s bag of words for feature **f**
        for each **word** in **word-bag**
        let **category-vector** be **hash-table**(**word**)
        if **category-vector** is defined for this word, then
            {
            for **c** = 1 to **num-categories**
                {
                if **probs**[**c**] > 0 then
                    { **probs**[**c**] = **probs**[**c**] * **category-vector**[**c**] }
                else
                    { **probs**[**c**] = **probs**[**c**] * **epsilon** }
                }
            }
        }
    let **most-probable-category** = **c** for which **prob**[**c**] is maximized
    return the name of **most-probable-category**

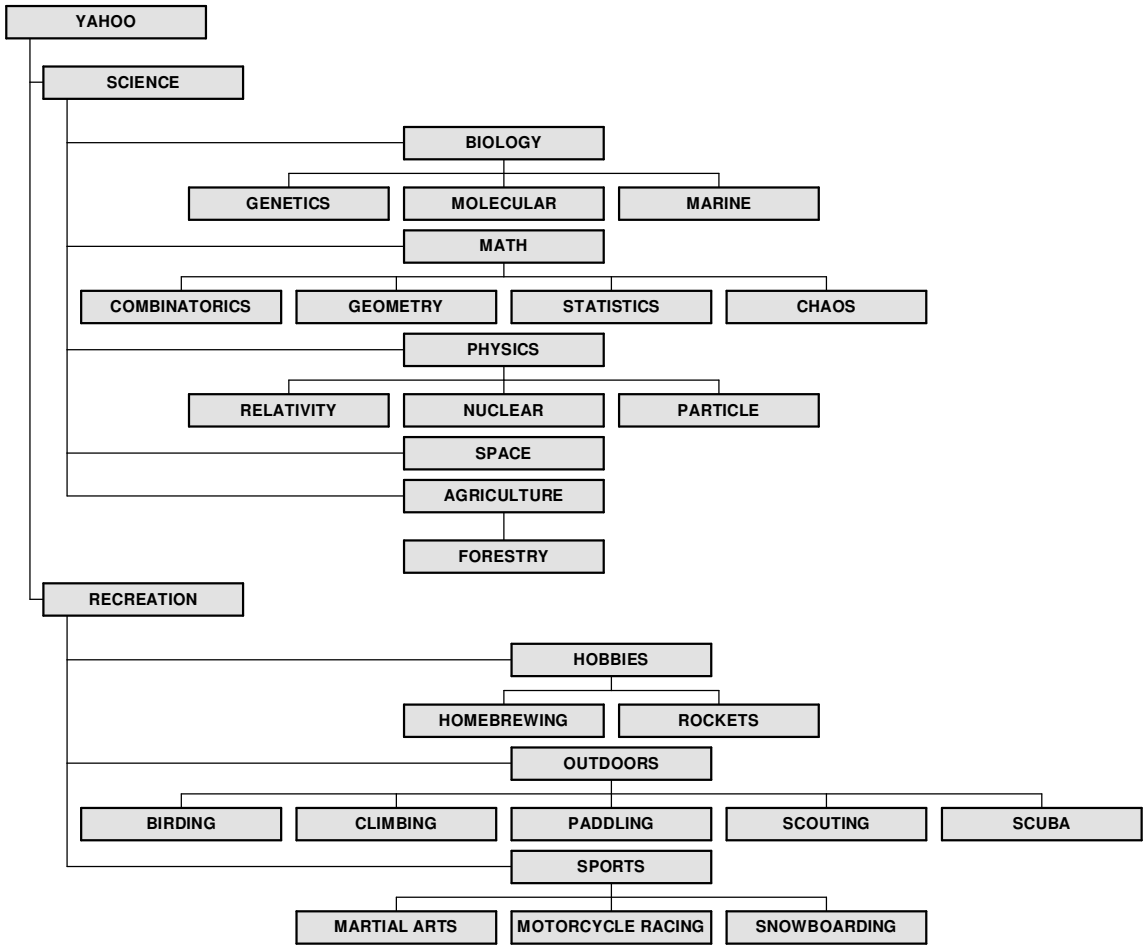
---

3 Experimental Methodology and Results

3.1 Experimental Data Set

In order to test our schemes for improving HTML text categorization, we gathered 612 web pages from the *Yahoo!* website's science and recreation categories. The web pages were divided into 26 disjoint classes. The categories downloaded are shown in Figure 3.1.1. Although the categories are shown here in a hierarchical organization, I make no effort to exploit this structure in my categorization experiments. Instead, I simply treat each node in Fig 3.1 as a separate category, except in the case where the node has no documents in it. In the case of the `sports` category, there were no general documents on sports, only documents on specific sports. In contrast to this, there was a large collection of general biology documents, so the `biology` category was treated as a disjoint category from it's sub-categories, such as `genetics`.

FIG 3.1.1: CATEGORY ORGANIZATION



3.2 Split-Sample Validation and Cross-Validation

Split-sample validation is a common way of estimating the accuracy of a learning system (Mitchell, 1997). In split-sample validation, the data set is divided into a training set, which is used to train the learning system, and a test set, which is left aside to test the accuracy of the system. Cross-validation is an improvement on split-sample validation. Cross validation is markedly superior to split-sample validation for small data sets (Goutte, 1997) All tests mentioned in this paper were performed with 10-fold cross-validation, which means that the data was divided randomly into ten subsets of equal size. The algorithm was trained and tested ten times, each time using nine of the ten subsets for training, and leaving one of the subsets out for testing.

We used a paired T-Test to determine whether the difference between average performances of two systems were statistically significant.

3.3 Multi-Feature vs. Single Feature model

In our first experiment, we tried to improve the accuracy of the naive Bayes classifier by using the HTML formatting codes to separate the words in each document into several bags of words. We used four bags: `words`, `titles`, `links`, and `headings`. Any words in the document that appeared inside the `<title>` and `</title>` tags were placed in the `title` bag. Any words that appeared inside `<h`*n*`>` `</h`*n*`>` tags, where *n* is any digit, were placed in the `headings` bag. Any words that appeared between a `<a href="`*URL*`">` and `</a>` tags were placed in the `links` bag. Finally, all words in the document were placed in the `words` bag.

Figure 3.3.1 shows the results of several experiments run on this data. We first ran tests on each of the features individually to get an idea of how much information each feature provides, relative to the other features. The curve marked `headings-only` shows the accuracy of a naive Bayes classifier on data that contains only information about the headings in the documents. Similar tests were performed on the other bags of words (`title-only`, `links-only`, `words-only`). As one would expect, the `words-only` test achieved higher accuracy than `title-only`, `links-only` and `headings-only`. As I predicted earlier, the words from the document titles contain a lot of information about the subject of the document. A classifier using only document titles achieved 40 to 45 percent accuracy, compared with 50 to 55 percent accuracy for the `words-only` classifier. This is impressive considering that the average document has 287.9 words in the `words` bag, yet only 3.4 words in the `title` bag. A classifier using only links achieved accuracy in the 30 to 40 percent range. The classifier that used only headings performed the poorest, with about 20 percent accuracy. The average numbers of words in the `links` and `headings` bags were 71.3 and 16.4, respectively.

The curve marked `multi-feature` shows the accuracy for a classifier which uses all four of the bags mentioned above. The `multi-feature` classifier performed marginally better than the `words-only` classifier for high numbers of training examples. The greatest improvements over `words-only` occur in the 300-500 training example range, and the improvements are statistically significant in the range from 325 to 375 training examples. The benefits of using the `multi-feature` classifier over the `words-only` classifier peak out at 375 training examples, where multi-features performs 8.75% better than `words-only`, a difference which is statistically significant at the $p = 0.01$ level. Although we were able to get slightly better results by using the multi-feature representation of the data, the improvement was slim for high-numbers of training examples, and non-existent for low numbers of training examples. The

words-only classifier performed substantially better than the multi-feature classifier for tests with less than 200 training examples. This behavior makes sense: the small bags of words used in the `multi-feature` representation (`titles`, `headings` and `links`) give it a disadvantage for small numbers of training examples because it takes many documents to sufficiently explore the space of words that commonly appear in these bags. At lower numbers of training examples, these extra bags of words just confuse the classifier because there is not enough data to be able to make accurate predictions of the conditional word-category probabilities.

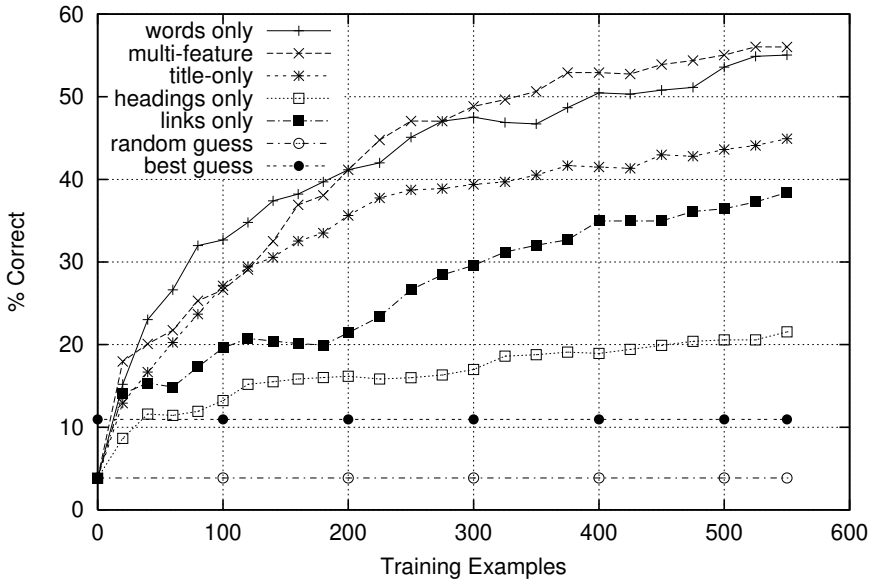


Figure 3.3.1: Comparison of accuracy on Multi-Feature and Single-Feature models: "Random Guess" is the probability of selecting the right category by making a random guess out of the 26 categories. "Best Guess" is the probability of selecting the right category by guessing the most probable category every time

3.4 Using the children of a page to gain accuracy

One of the unique features of HTML documents is the ability to make hyperlinks to other pages. Often, these hyperlinks take the reader to other pages with related material. The next question that this paper focuses on is whether we can take advantage of the relationship between linked web pages to gain accuracy in a naive Bayes classifier. Rather than a training example being a single document, a training example in this experiment consisted of a document *X*, plus all of the documents that are linked to by *X* (the "children" of *X*). There are a couple of ways this can be implemented. The first way is to append all of the text of the children of *X* to the end of *X*, and pretend that the training example is just one document. The second way is to pretend that each of *X*'s children is a separate training example from *X*. In fact, they are not really separate training examples because we are unsure of their true category. If we are using a naive Bayes classifier, then the only difference between these two methods of representing the data is that the first approach weights each training example the same, where the second approach weights each training example according to the number of child documents it contains. In both approaches, the conditional word-category probabilities are the same, but the prior probabilities for each category will be different in the first method than in the second method. The experiments in this section use the second approach: child documents are treated as separate training examples.

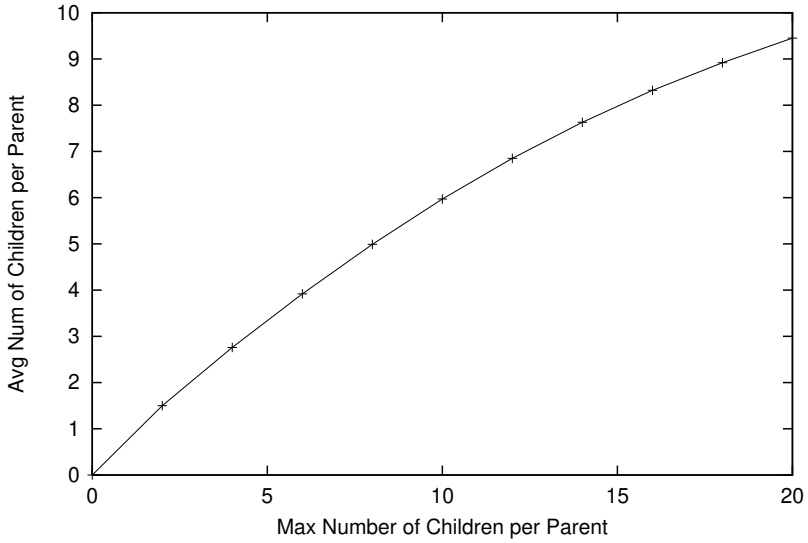
To test an example, we simply tested on the parent document in that example and ignored the child documents.

We added a wrapper around the existing algorithm to accomplish this:

TRAIN-BAYES-DEEP (**examples**)
initialize **new-examples** as an empty list of examples
for each **example** in **examples**
{
let **c** be the category of **example**
for each document **d** in example
{
let **new-example** be an example in category **c**, containing the single document **d** as the data
add **new-example** to **new-examples**
}
}
return **train-bayes** (**new-examples**)

TEST-BAYES-DEEP (**example training-result**)
let **c** be the category of **example**
let **new-example** be an example in category **c** containing just the first document in **example** (just the parent document's data)
return **test-bayes**(**new-example**)

For an experimental data set, we took all pages from the data set used in the previous experiment and followed the hyperlinks, downloading the child pages in a random order. For practical reasons, no more than twenty child pages were downloaded from any parent page. If the parent contained more than twenty children, then twenty were chosen randomly. The data was then divided into several test files with varying number of children per parent. One file contained just the data from the parent web pages, one file contained data from the parent plus up to two children, the next contained up to four children, and so on. Since some of the parent pages contained very few links, the average number of child pages per parent page was usually about half of the allowed number of children per parent. The graph below shows the relationship between the maximum number of children allowed in the data file and the average number of children per parent.



The results of this experiment were much more promising than those of the previous experiment. In some cases, we observed that a classifier that took advantage of the child-documents resulted in an accuracy improvement of as much as fifteen percent over a similar classifier which only used the parent documents.

Figure 3.4.1 plots the accuracy of the naive Bayes classifier as a function of the average number of child pages that were included in the representation of each training example. The there are several different curves plotted for classifiers using different numbers of training examples. Generally speaking, these curves tend to go up in accuracy until the number of links reaches about six or seven, after which the curves level off. This pattern is a bit more clear in Figure 3.4.2, which is averages these curves over 26 different experiments, each using a different number of training examples.

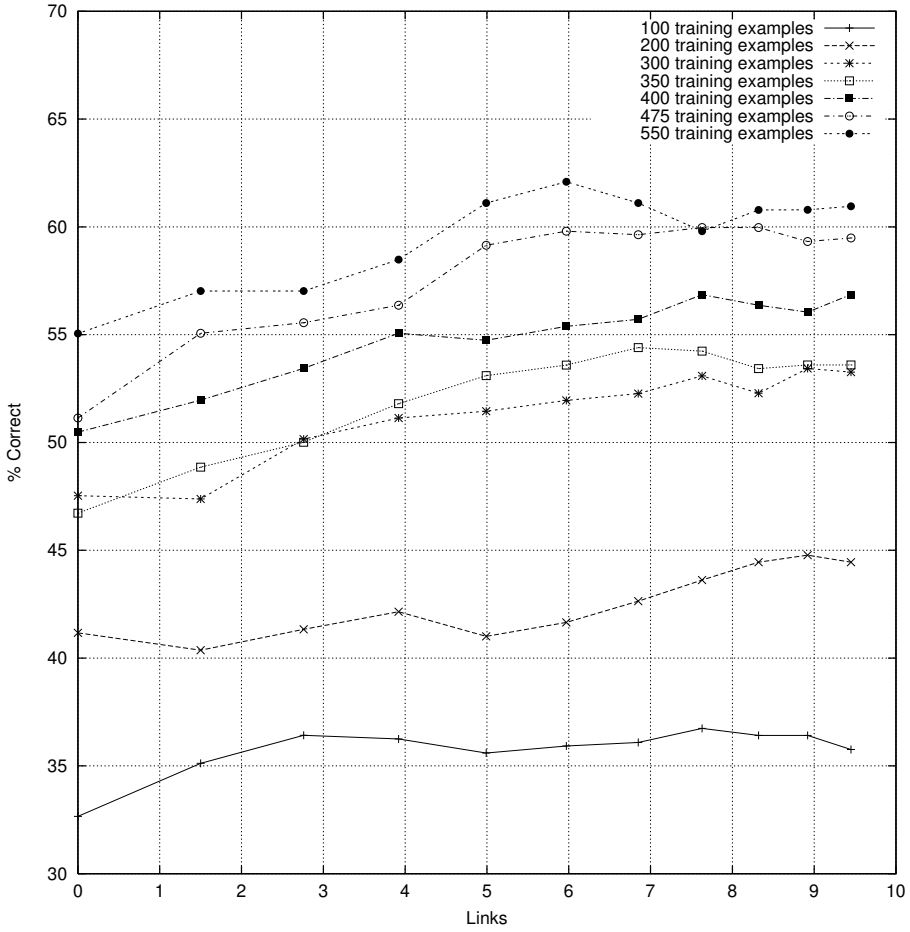


Fig 3.4.1: Accuracy as a Function of Number of Child Pages per Parent Page

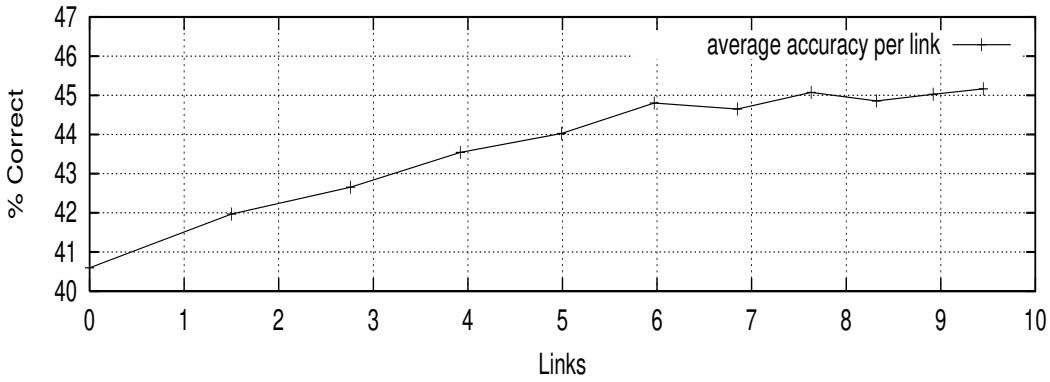


Fig 3.4.2: Accuracy as a Function of Number of Child Pages per Parent Page.
(averaged over various numbers of training examples, from 20-550)

Fig 3.4.3: Improved classifier compared with standard "unimproved" classifier

The graphs on this page are based on a comparison of the improved classifiers with a classifier which does not take advantage of the data from a page's children. The "% Improvement" axis of each graph plots $((\text{Accuracy}_{\text{improved}}/\text{Accuracy}_{\text{unimproved}})\text{-}1) \bullet 100\%$

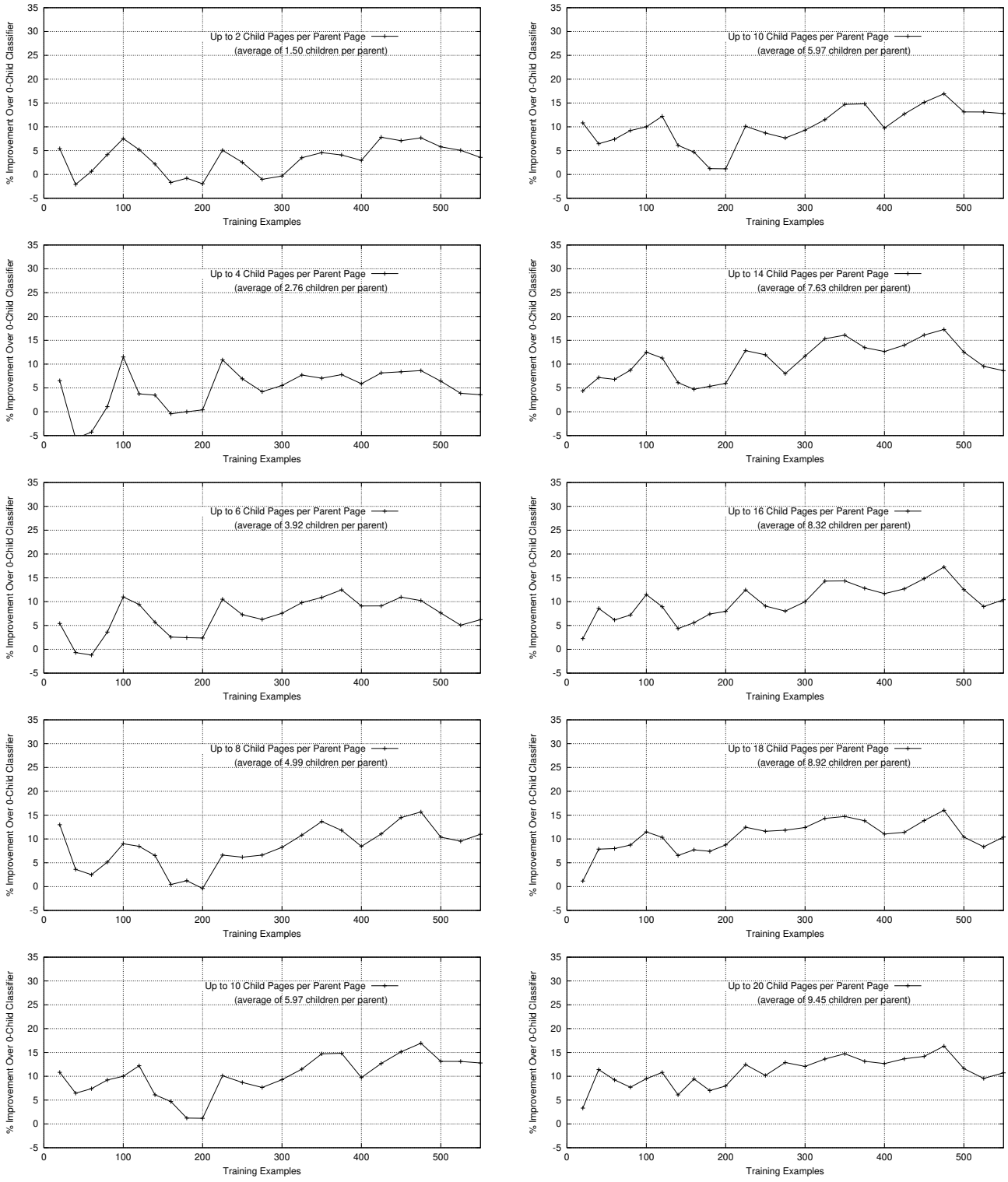


Figure 3.4.4, below, gives us some insight as to which categories the classifier tends to classify incorrectly. The number of contours around an intersection of two gridlines depends on how often the category represented on the horizontal gridline is mistakenly classified as the category represented on the vertical gridline. Only incorrect classifications show up on this graph; all intersections along the main diagonal are zero. Some of the categories that are confused are not so surprising. Documents in sub-areas of math (`geometry`, `chaos`, `combinatorics` and `statistics`) are frequently classified into the general `math` category. Documents in areas of math and sub-areas of physics are often classified mistakenly into the general `physics` category. Interestingly, but perhaps not surprisingly, `marine` biology is often mistakenly classified as `scuba`, so obviously the classifier is picking up on the ocean-related terminology used in these two areas. Some of the mistakes are more unexpected, such as `martial arts`, `forestry`, and `marine` biology being mistakenly classified as `homebrewing`. One thing the graph shows is that recreation documents are very rarely misclassified as science documents: the lower left section of the graph is bare.

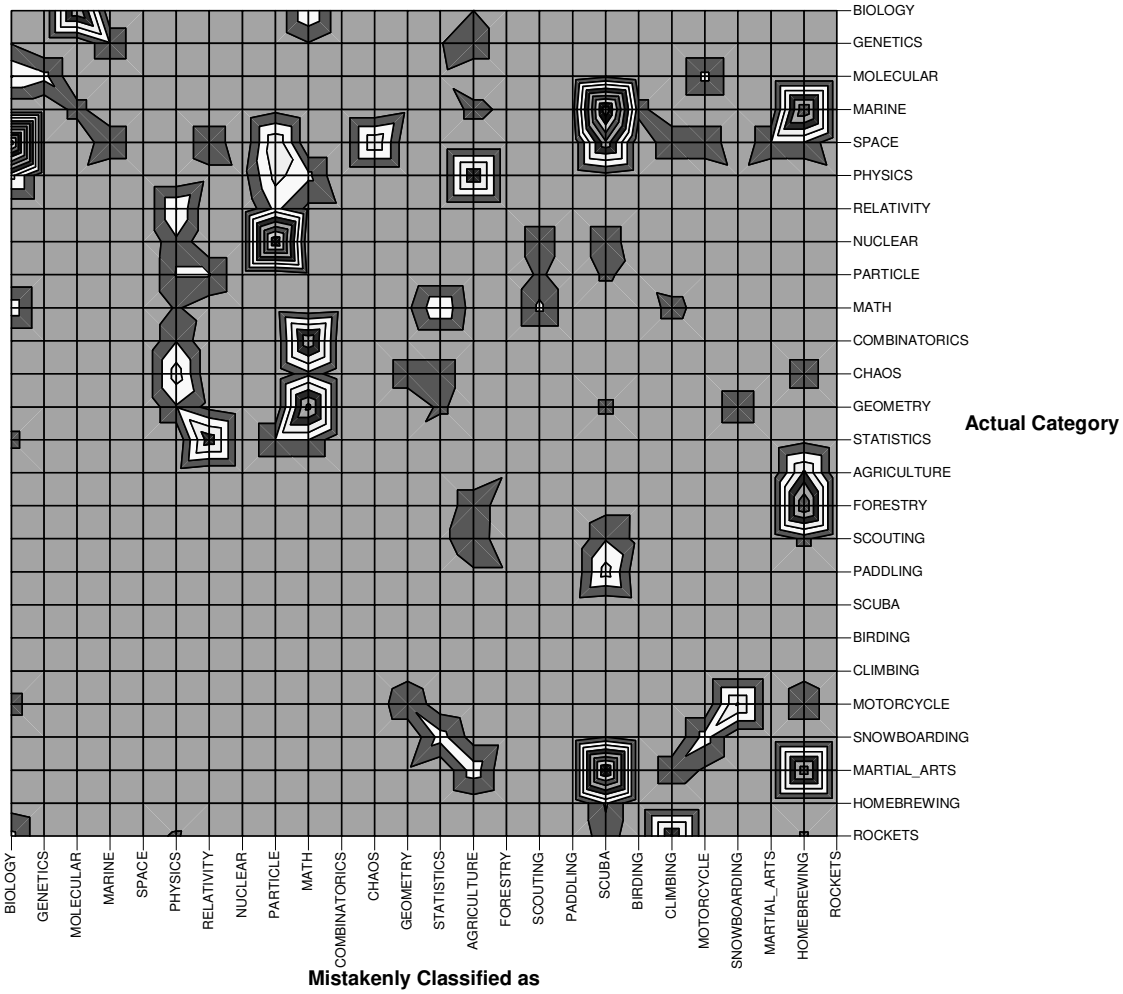


Fig 3.4.4: Confusion graph for an improved classifier using maximum of ten child pages per parent pages, trained with 375 training examples

4 Future Work and Related Work

There are a variety of avenues for improving upon the ideas presented in this paper:

More Ideas for Utilizing HTML Formatting

Choon Quek's paper *Classification of World Wide Web Documents* presented findings similar to my first experiment. Separating the document into several bags of words, such as titles, links, and headings, does not benefit the accuracy of a naïve Bayes classifier very much. Perhaps some of the other features of HTML documents can be used to improve a text classifier. For instance, as web pages become increasingly graphics-oriented, many of the headings and titles are presented as images. Sometimes the HTML code includes a description of the images for the convenience of those who don't have a browser with graphical capabilities. Adding an "image" bag of words to the representation may benefit the accuracy of the classifier.

Intelligent Selection of Links:

One problem with the current method of using the information on linked 'child' pages is that often a single document can be linked to a very large number of other pages. Following all of these links, downloading all of the child pages, and including all of this information in the representation of the parent may be impractical. Including all of these child pages may not even be a beneficial thing to do, since some links will lead to unrelated pages. For example, sometimes a page will contain links to Netscape's home page so that a user can download a plug-in required to view the page. Currently our method is to simply select links at random and set a limit on the maximum number of child pages that can be downloaded for each parent. Instead, it may be better to use machine learning techniques to help choose the best links to follow. There are several ways we could do this:

1) Treat the text of each link as another document, and use the existing classifier to decide which category the link is likely to belong to. Only follow links which are categorized into the same category as the page that contains them.
2) Develop a separate categorizers just for categorizing the text of links. For each category that a document may belong to, we could create a link classifier with two categories "keep" and "throw away", and then use the link classifier to determine whether or not to follow the link. There are a few different ways we could train these link classifiers. One is to categorize links by hand into positive and negative categories and use that data as training examples. Another way is to categorize the child pages by hand, or using a naive Bayes classifier, and then mark links as positive only when they lead to a child whose category is the same as the parent's category.
3) Use the surrounding context of the link to help determine if it is worth following. For example, rather than training a link classifier to use only the underlined text in the link to determine whether it is worth following, we could include some of the text surrounding the link to help decide if it was a promising link.

Including Deeper Levels of Linked Pages

If a web page X is linked to another web page Y, our hypothesis says that X and Y are similar in subject. If Y is linked to a third page Z, then Y and Z are similar in subject. Consequently, it is

likely that *X* and *Z* are related in subject. If taking advantage of the first level of linked pages is benificial to the accuracy of the classifier, then taking advantage of deeper levels of pages may also benefit the classifier. As anyone who has "surfed" the web before knows, we can sometimes get from one subject to another in a single click, while other times we can follow link after link and still find material related to the page we started from. The trick to using deeper levels of linked pages is intelligent selection of links and pages, so the methods mentioned above in "intelligent selection of links" would be useful here. It would also be useful to try to filter out unrelated pages as we traverse the web. Here is a possible algorithm for including deeper levels of linked pages:

---

expand-example(**training-example**)
    for each **link** in **training-example**
        {
        run link-classifier on **link**, let **link-result** be the result (either "keep" or "throw away")
        if **link-result** is "keep"
            {
            let **child** be the web page that **link** points to
            let **cat** be the category of **training-example**
            run naive-Bayes classifier on **child**; let **child-cat** be the result
            if (**cat** = **child-cat**)
                {
                let **training-example** be (**training-example** appended to **expand-example**(**child**))
                }
            }
        }
    return **training-example**

Take Advantage of a Hierarchical Subject Structure:

Indexes like *Yahoo!* Organize web pages into a hierarchical subject tree. One way to take advantage of this tree structure is to have a classifier for every non-leaf node in the tree. So, for example, for the data used in this paper we would have a classifier which distinguished between `science` and `recreation` documents. If that classifier determined that a document belongs to the `science` category, then it would be passed on to another classifier which distinguished between `physics`, `biology`, `math`, and `space` documents. If that classifier decided a document was a `biology` document, then another classifier would decide if it was `marine` biology, `molecular` biology, or `genetics`, and so on. Koller and Sahami show that there can be significant benefits to using this method, but these benefits cannot be realized with a naive Bayes classifier. They report that the success of this method depends on using more complex classifiers which account for dependencies between word occurrences. One obvious problem with their method is that if the classifiers make a mistake near the top of the hierarchy, they cannot recover from it. Instead of making absolute decisions at each node of the hierarchy tree, another method would be to explore more than one branch at each node in the hierarchy,

and come up with some method of combining the probabilities of classifiers at different levels in order to arrive at a final decision.

McCallum, Rosenfeld, Mitchell, and Ng (1998) present a method of improving classification in a hierarchy by using a technique borrowed from statistics called *shrinkage*. One of the problems with classification in a hierarchy is that the further down the hierarchy we go, the more specific the subjects become and there is less training examples which fall into that category. If the category is too specific, then there may not be enough data to accurately estimate all of the parameters of that node in the hierarchy. Shrinkage deals with that problem by estimating the parameters (conditional probabilities of word appearances given each category) by using a weighted sum of the observed parameters in that node and all of its ancestor nodes.

Active Learning

When the classifier is not very confident about it's categorization of a particular document, it could submit this to a human who would make a final decision. The classifier would then use this to re-train itself. This way, the classifier would be actively trying to make improvements in the areas where it is having the most difficulty. For more information on active learning, see *Improving Generalization with Active Learning* (Cohn, 1994).

Combining Several Classifiers Non-Hierarchically

In practical applications of text categorization, the classification problem may involve hundreds or thousand of possible categories instead of just ten or twenty. Introducing more categories will increase the number of parameters (i.e. words in the vocabulary) and can result in over-fitting of the training data [Koller and Sahami]. Therefore, the accuracy of the classifier will decrease as the number of categories grows. Breaking the problem up into several smaller sub-problems may improve the overall accuracy. Combining classifiers hierarchically is one way of doing this. Can we improve accuracy by using different, non-hierarchical organization of several classifiers? For example, we could build one classifier for each pair of categories. To determine the category of a document, we could test it with each of the pair-classifiers, and then compare the results of all of the pair-classifiers to determine which category is most likely.

5 Conclusions

This paper has examined two ways of taking advantage of special features of HTML documents to improve learning for text categorization. The first way was to use the HTML formatting codes to separate the document into several bags of words. We saw that using this richer representation of the web pages, we were able to improve classification accuracy by up to 8.75%. Although we were able to see significant benefits from this method, the benefits were confined to a small area on the learning curve. For smaller numbers of training examples, the richer representation of the data clearly performed worse the traditional straight-text representation of the data.

Our second method for improving classification of HTML documents was to include the text of linked pages in the representation of training examples. Using this method, we were able to see improvements in accuracy of up to fifteen percent over a representation which did not include linked pages. We saw the benefits in accuracy increase as we increased the number of linked

pages included in the representation of each training example. The benefits peaked out when an average of six linked pages were included with each example. Above six links, the accuracy remained at about the same level as we increased the number of links.

In the future, if automatic text classifiers can achieve high accuracy over a broad range of categories, then a system could conceivably build and maintain a *Yahoo*-like index to the world wide web entirely automatically. Although the goal of an automatic web-page classifier that could perform near the accuracy of a human seems far off, this work has taken a step towards this goal, and has shown that there are a variety of possible ways to improve on this work.

Acknowledgements

I would like to thank my advisor, Raymond Mooney, for providing many valuable ideas, and for guiding me through the process of conducting this research. I would also like to thank Mohammed Gouda and Risto Miikkulainen for supporting this work by agreeing to be in the committee of readers.

References

Cohn, D.; Atlas L.; Ladner, R. 1994. *Improving Generalization with Active Learning* in *Machine Learning*, 15(2):201-221

Goutte, C. 1997, *Note on Free Lunches and Cross-Validation*. in *Neural Computation*, 9, 1211-1215

Koller, D., and Sahami, M. 1997. *Hierarchically Classifying Documents Using Very Few Words.* In *IMCL-97: Proceedings of the Fourteenth International Conference on Machine Learning*, 170-178, Morgan Kaufmann.

McCallum, A., and Nigam, K.. 1998. *A Comparison of Event Models for Naive Bayes Text Classification.* In *Papers from the AAAI 1998 Workshop on Text Categorization* , 41-48. AAAI Press.

McCallum, A.; Rosenfeld, R.; Mitchell, T.; and Ng, A. 1998. *Improving Text Classification by Shrinkage in a Hierarchy of Classes.* In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann.

Pazzani, M., and Billsus, D. *Learning and Revising user Profiles: The Identification of Interesting Web Sites.* In *Machine Learning*, 27(3):313-331,1997.

Quek, C. *Classification of World Wide Web Documents*, 1998.

Mitchell, T. *Machine Learning*. McGraw-Hill, 1997

Yahoo! On-line Guide to the internet. www.yahoo.com