Weight adjustment schemes for a centroid based classifier *

Shrikanth Shankar and George Karypis

University of Minnesota, Department of Computer Science
Minneapolis, MN 55455
Technical Report: TR 00-035

{shankar, karypis}@cs.umn.edu

Abstract

In recent years we have seen a tremendous growth in the volume of text documents available on the Internet, digital libraries, news sources, and company-wide intra-nets. Automatic text categorization, which is the task of assigning text documents to pre-specified classes (topics or themes) of documents, is an important task that can help both in organizing as well as in finding information on these huge resources. Similarity based categorization algorithms such as k-nearest neighbor, generalized instance set and centroid based classification have been shown to be very effective in document categorization. A major drawback of these algorithms is that they use all features when computing the similarities. In many document data sets, only a small number of the total vocabulary may be useful for categorizing documents. A possible approach to overcome this problem is to learn weights for different features (or words in document data sets). In this report we present two fast iterative feature weight adjustment algorithms for the linear-complexity centroid based classification algorithm. Our algorithms use a measure of the discriminating power of each term to gradually adjust the weights of all features concurrently. We experimentally evaluate our algorithms on the Reuters-21578 and OHSUMED document collections and compare it against Rocchio, Widrow-Hoff and SVM. We also compared its performance in terms of classification accuracy on data sets with multiple classes. On these data sets we compared its performance against traditional classifiers such as k-nn, Naive Bayesian and C4.5. Experiments show that feature weight adjustment improves the performance of the centroid-based classifier by 2- 5% , substantially outperforms Rocchio and Widrow-Hoff and is competitive with SVM. These algorithms also outperform traditional classifiers such as k-nn, naive bayesian and C4.5 on the multi-class text document data sets.

*This work was supported by NSF CCR-9972519, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. Related papers are available via WWW at URL: **http://www.cs.umn.edu/~karypis**

1 Introduction

We have seen a tremendous growth in the volume of online text documents available on the Internet, digital libraries, news sources and company-wide intra-nets. It has been forecasted that these documents along with other unstructured data will become the predominant data type stored online. Automatic text categorization [33, 43, 16] which is the task of assigning text documents to pre-specified classes of documents, is an important task that can help people find information from these huge resources. Text categorization presents huge challenges due to a large number of attributes, attribute dependency, multi-modality and large training set.

The various document categorization algorithms that have been developed over the years [36, 1, 8, 11, 25, 16, 19, 2, 42, 20, 13] fall under two general categories. The first category contains traditional machine learning algorithms such as decision trees, rule sets, instance-based classifiers, probabilistic classifiers, support vector machines, *etc*., that have either been used directly or after being adapted for use in the context of document data sets. The second category contains specialized categorization algorithms developed in the Information Retrieval community. Examples of such algorithms include relevance feedback, linear classifiers, generalized instance set classifiers, *etc*.

A general class of algorithms that has been shown to produce good document categorization performance is similarity based. This class contains algorithms such as k-nearest neighbor[42], generalized instance set[19] and centroid based classifiers[12]. In these algorithms the class of a new document is determined by computing the similarity between the test document and individual instances or aggregates of the training set, and determining the class based on the class distribution of the nearest instances or aggregates.

A major drawback of these algorithms is that they use all the features while computing the similarity between a test document and the training set instances or aggregates. In many document data sets, only a relatively small number of the total features may be useful in categorizing documents, and using all the features may affect performance. A possible approach to overcome this problem is to learn weights for different features (i.e. words). In this approach, each feature has a weight associated with it. A higher weight implies that this feature is more important for classification. When the weights are either 0 or 1 this approach become the same as feature selection. We refer to such algorithms as feature weight adjustment or just weight adjustment techniques.

This report presents two fast iterative feature weight adjustment algorithms for the linear-complexity centroid based classification algorithm. These algorithms use a measure of the discriminating power of each term to gradually adjust the weights of all features concurrently. Analysis shows that this approach gradually eliminates the least discriminating features in each document thus improving its classification accuracy. We experimentally evaluate these algorithms on the Reuters-21578 [24] and OHSUMED [14] document collection and compare its performance in terms of precision and recall against Rocchio [32], Widrow-Hoff [39] and Support vector machines [35, 16]. We also compared its performance in terms of classification accuracy on data sets with multiple classes. These data sets are described in Section (6.3) and in [12]. On these data sets we compared its performance against traditional classifiers such as k-nn, Naive Bayesian and C4.5. Experiments show that feature weight adjustment improves the performance of the centroid-based classifier by 2- 5% , substantially outperforms Rocchio and Widrow-Hoff and is competitive with SVM. These algorithms also outperform traditional classifiers such as k-nn, naive bayesian and C4.5 on text document data sets.

The organization of the report is as follows. Section (2) describes some of the other classification schemes used on text data while section (3) gives a brief overview of the centroid based classifier. Section (4) describes the two weight adjustment schemes and discusses their computational complexity. Section (5) presents an analysis of the two schemes. Section (6) documents the results of these schemes on various data sets as well as the performance of other classifiers on the same data sets.

2 Previous work

2.1 Linear Classifiers

Linear classifiers [25] are a family of text categorization learning algorithms that learn a feature weight vector w, for every category. Weight learning techniques such as Rocchio [32] and Widrow-Hoff algorithm [39] are used to learn the feature weight vector from the training samples. These weight learning algorithms adjust the feature weight vector such that features or words that contribute significantly to the categorization have large values. In both Rocchio and Widrow-Hoff the weight vector w is used for classification as follows. When a test document x is to be classified a pre-defined threshold t is used. x is assigned to the positive class when $w \cdot x > t$. Note that this concept of a weight vector is very different from what we use

Rocchio Rocchio [32, 25] can be used as batch algorithm to learn a weight vector from an existing weight vector and a set of training examples. The jth component w_j of the new vector is

$$w_j = \alpha w_{1,j} + \beta \frac{\sum_{i \in C} d_{i,j}}{n_C} - \gamma \frac{\sum_{i \notin C} d_{i,j}}{n - n_C} \tag{1}$$

where n is the number of training instances, C is the set of positive training instances, and n_C is number of positive training instances.

Usually rocchio uses only positive weights so all negative weights are reset to 0.

Widrow-Hoff The Widrow-Hoff algorithm [39, 10, 25] is an online algorithm which runs through the training examples one at a time updating the weight vector.

The new weight vector $\vec{w}_{i+1}$ is computed from $\vec{w}_i$ as follows.

$$w_{i+1,j} = w_{i,j} - 2\eta(w_i \cdot d_i - y_i)d_{i,j} \tag{2}$$

where y_i is label of row d_i and is either 0 (negative class) or 1 (positive class). The parameter η controls how quickly the weight vector can change and how much influence a new example has on it.

While it may seem that the final vector $\vec{w}_{n+1}$ should be used there are theoretical results [25] that suggest that a better final weight vector is the average of all the weight vectors computed along the way.

$$\vec{w} = \frac{1}{n} \sum_{i=1}^{n+1} \vec{w}_i \tag{3}$$

2.2 Support Vector Machines

Support Vector Machines (SVM) is a new learning algorithm proposed by Vapnik [35]. This algorithm was introduced to solve two-class pattern recognition problem using the Structural Risk Minimization principle [35, 6]. Given a training set in a vector space, this method finds the *best* decision hyper-plane that separates two classes. The quality of a decision hyper-plane is determined by the distance (referred as margin) between two hyper-planes that are parallel to the decision hyper-plane and touch the closest data points of each class. The *best* decision hyper-plane is the one with the maximum margin. The SVM problem can be solved using quadratic programming techniques [35, 6]. SVM extends its applicability on the linearly non-separable data sets by either using soft margin hyper-planes, or by mapping the original data vectors into a higher dimensional space in which the data points are linearly separable. An efficient

implementation of SVM and its application in text categorization of Reuters-21578 corpus is reported in [16]. We use this implementation for our comparison purposes.

2.3 k Nearest Neighbor

k-nearest neighbor (k-NN) classification is an instance-based learning algorithm that has been applied to text categorization since the early days of research [27, 15, 41, 5], and has been shown to produce better results when compared against other machine learning algorithms such as C4.5 [31] and RIPPER [4]. In this classification paradigm, k nearest neighbors of a test document are computed first. Then the similarities of this document to the k nearest neighbors are aggregated according to the class of the neighbors, and the test document is assigned to the most similar class (as measured by the aggregate similarity). A major drawback of the similarity measure used in k-NN is that it uses all features equally in computing similarities. This can lead to poor similarity measures and classification errors, when only a small subset of the words is useful for classification. To address this problem, a variety of techniques have been developed for adjusting the importance of the various terms in a supervised setting. Examples of such techniques include preset weight adjustment using mutual information [9, 38, 37], RELIEF [17, 18], and variable-kernel similarity metric learning [26].

2.4 C4.5

A decision tree is a widely used classification paradigm in machine learning and data mining. The decision tree model is built by recursively splitting the training set based on a locally optimal criterion until all or most of the records belonging to each of the leaf nodes bear the same class label. C4.5 [31] is a widely used decision tree-based classification algorithm that has been shown to produce good classification results, primarily on low dimensional data sets. Unfortunately, one of the characteristics of document data sets is that there is a relatively large number of features that characterize each class. Decision tree based schemes like C4.5 do not work very well in this scenario due to over-fitting [5, 13]. The over-fitting occurs because the number of samples is relatively small with respect to the number of distinguishing words, which leads to very large trees with limited generalization ability. The C4.5 results were obtained using a locally modified version of the C4.5 algorithm capable of handling sparse data sets.

2.5 Naive Bayesian

The naive Bayesian (NB) algorithm has been widely used for document classification, and has been shown to produce very good performance [22, 23, 21, 28]. For each document, the naive Bayesian algorithm computes the posterior probability that the document belongs to different classes and assigns it to the class with the highest posterior probability. The posterior probability $P(c_k|d_i)$ of class c_k given a test document d_i is computed using Bayes rule

$$P(c_k|d_i) \quad = \quad \frac{P(c_k)P(d_i|c_k)}{P(d_i)}, \tag{4}$$

and d_i is assigned to the class with the highest posterior probability, that is,

$$\text{Class of } \; d_i = \arg \max_{1 \le k \le N} \{P(c_k|d_i)\} = \arg \max_{1 \le k \le N} \{P(c_k)P(d_i|c_k)\}, \tag{5}$$

where N is the total number of classes. The naive Bayesian algorithm models each document d_i, as a vector in the term space, *i.e.*, $d_i = (d_{i1}, d_{i2}, \ldots, d_{im})$, where d_{ij} models the presence or absence of the jth term. Naive Bayesian

computes the two quantities required in (5) as follows. The approximate class priors ($P(c_k)$) are computed using the maximum likelihood estimate

$$P(c_k) = \frac{\sum_{i=1}^{|D|} P(c_k|d_i)}{|D|}, \quad (6)$$

where D is the set of training documents and $|D|$ is the number of training documents in D. The $P(d_i|c_k)$ is computed by assuming that when conditioned on a particular class c_k, the occurrence of a particular value of d_{ij} is statistically independent of the occurrence of any other value in any other term $d_{ij'}$. Under this assumption, we have that

$$P(d_i|c_k) = \prod_{j=1}^{m} P(d_{ij}|c_k), \quad (7)$$

and because of this assumption this classifier is called "naive" Bayesian.

3 Centroid-Based Document Classifier

In the centroid-based classification algorithm, the documents are represented using the vector-space model [33]. In this model, each document d is considered to be a vector in the term-space. In its simplest form, each document is represented by the *term-frequency* (TF) vector $\vec{d}_{tf} = (tf_1, tf_2, \ldots, tf_n)$, where tf_i is the frequency of the ith term in the document. A widely used refinement to this model is to weight each term based on its *inverse document frequency* (IDF) in the document collection. The motivation behind this weighting is that terms appearing frequently in many documents have limited discrimination power, and for this reason they need to be de-emphasized. This is commonly done [33] by multiplying the frequency of each term i by $\log(N/df_i)$, where N is the total number of documents in the collection, and df_i is the number of documents that contain the ith term (*i.e.*, document frequency). This leads to the *tf-idf* representation of the document, *i.e.*, $\vec{d}_{tfidf} = (tf_1 \log(N/df_1), tf_2 \log(N/df_2), \ldots, tf_n \log(N/df_n))$. Finally, in order to account for documents of different lengths, the length of each document vector is normalized so that it is of unit length, *i.e.*, $\|\vec{d}_{tfidf}\|_2 = 1$. In the rest of the paper, we will assume that the vector representation $\vec{d}$ of each document d has been weighted using *tf-idf* and it has been normalized so that it is of unit length.

In the vector-space model, the similarity between two documents d_i and d_j is commonly measured using the cosine function [33], given by

$$\cos(\vec{d}_i, \vec{d}_j) = \frac{\vec{d}_i \cdot \vec{d}_j}{\|\vec{d}_i\|_2 * \|\vec{d}_j\|_2}, \quad (8)$$

where "$\cdot$" denotes the dot-product of the two vectors. Since the document vectors are of unit length, the above formula simplifies to $\cos(\vec{d}_i, \vec{d}_j) = \vec{d}_i \cdot \vec{d}_j$.

Given a set S of documents and their corresponding vector representations, we define the ***centroid*** vector $\vec{C}$ to be

$$\vec{C} = \frac{1}{|S|} \sum_{d \in S} \vec{d}, \quad (9)$$

which is nothing more than the vector obtained by averaging the weights of the various terms present in the documents of S. We will refer to the S as the ***supporting set*** for the centroid $\vec{C}$. Analogously to documents, the similarity between two centroid vectors and between a document and a centroid vector are computed using the cosine measure. In the

first case,

$$\cos(\vec{C_i}, \vec{C_j}) = \frac{\vec{C_i} \cdot \vec{C_j}}{\|\vec{C_i}\|_2 * \|\vec{C_j}\|_2}, \tag{10}$$

whereas in the second case,

$$\cos(\vec{d}, \vec{C}) = \frac{\vec{d} \cdot \vec{C}}{\|\vec{d}\|_2 * \|\vec{C}\|_2} = \frac{\vec{d} \cdot \vec{C}}{\|\vec{C}\|_2}. \tag{11}$$

Note that even though the document vectors are of length one, the centroid vectors will not necessarily be of unit length.

The idea behind the centroid-based classification algorithm [12] is extremely simple. For each set of documents belonging to the same class, we compute their centroid vectors. If there are k classes in the training set, this leads to k centroid vectors $\{\vec{C_1}, \vec{C_2}, \ldots, \vec{C_k}\}$, where each $\vec{C_i}$ is the centroid for the ith class. The class of a new document x is determined as follows. First we use the document-frequencies of the various terms computed from the training set to compute the *tf-idf* weighted vector-space representation of x, and scale it so $\vec{x}$ is of unit length. Then, we compute the similarity between $\vec{x}$ to all k centroids using the cosine measure. Finally, based on these similarities, we assign x to the class corresponding to the most similar centroid. That is, the class of x is given by

$$\arg \max_{j=1,\ldots,k} (\cos(\vec{x}, \vec{C_j})). \tag{12}$$

The computational complexity of the learning phase of this centroid-based classifier is linear on the number of documents and the number of terms in the training set. The computation of the vector-space representation of the documents can be easily computed by performing at most three passes through the training set. Similarly, all k centroids can be computed in a single pass through the training set, as each centroid is computed by averaging the documents of the corresponding class. Moreover, the amount of time required to classify a new document x is at most $O(km)$, where m is the number of terms present in x. Thus, the overall computational complexity of this algorithm is very low, and is identical to fast document classifiers such as Naive Bayesian.

4 Weight Adjustment for Centroid based Classifier

In this section we present algorithms to improve the classification performance achieved by the centroid-based classifier by adjusting the weight of the various features. In the rest of this section we first present two iterative weight-adjustment algorithms, and finally discuss how to improve the performance in the case of binary classification.

One approach to weight adjustment (and feature selection) is to choose a small number of features and adjust their weights in order to improve classification accuracy. This approach has been shown to achieve poor results in the text domain where the number of 'important' features is usually quite large. With this in mind our approach performs a simultaneous weight adjustment of all the features.

4.1 Fixed-Weight Adjustment Schemes

Any scheme that adjusts the weights of the various features (*i.e.*, terms) has to perform two tasks. First, it must rank the various features according to their discriminating power. Second, it must adjust the weight of the various features in order to emphasize features with high discriminating power and/or de-emphasize features with none or limited discriminating power.

Over the years, a number of schemes have been developed to measure the discriminating power of the various

features such as information gain, entropy , gini index, and χ^2 statistic. In our algorithm, the discriminating power of each feature is computed using a measure similar to the gini index, as follows. Let m be the number of different classes, and let $\{\vec{C}_1, \vec{C}_2, \ldots, \vec{C}_m\}$ be the centroid vectors of these classes. For each term i, let $\vec{T}_i = \{C_{1,i}, C_{2,i}, \ldots, C_{m,i}\}$ be the vector derived from the weight of the ith term in each one of the m centroids, and let $\vec{T'}_i = \vec{T}_i / ||\vec{T}_i||_1$ be the one-norm scaled version of $\vec{T}_i$. The discriminating power of the ith term P_i is given by

$$P_i = \sum_{j=1}^{m} T_{j,i}^2, \tag{13}$$

which is nothing more than the square of the length of the $\vec{T'}_i$ vector. Note that the value of P_i is always in the range $[1/m, 1]$. The lowest value of P_i is achieved when $T'_{i,1} = T'_{i,2} = \cdots = T'_{i,m}$ *i.e.*, a term is equally distributed amongst all the classes; whereas the highest is achieved when the ith term occurs in only a single class. Thus, a value close to one indicates that the term has a high discriminating power, whereas a value close to $1/m$ indicates that the terms has little if any discriminating power. In the rest of this paper, we will refer to P_i as the ***purity*** of the ith term, and we will refer to the vector $\vec{P} = \{P_1, P_2, \ldots, P_n\}$ of the purities of all the n terms as the ***purity vector***.

Having ranked the various terms using the purity as a measure of their discriminating power, the next step is to adjust their weights so that terms with higher discriminating power become more important than terms with lower discriminating power. A simple way of doing this, is to scale each one of the terms according to their purity. In particular, each document vector $\vec{d}$ is transformed to a new vector $\vec{d'} = \{P_1 d_1, P_2 d_2, \ldots, P_i d_i, \ldots P_n d_n\}$. Given this set of transformed document vectors, the centroid classification algorithm will proceed to scale each document to be of unit length, and then build a new set of centroid vectors for the various classes. A new document will be classified by first scaling its terms according to the purity vector and then computing its similarity to the new set of centroids. Since the purity values are always less or equal to one, the weight of the various terms in each transformed document $\vec{d'}$ will always be equal or smaller than their original weights. However, as will be discussed in section (5), the re-normalization operation performed by the centroid classification algorithm causes the purest terms in each document to actually gain weight, achieving the desired feature weight adjustments.

Unfortunately, this simple scheme has two drawbacks. The first is that this weight adjustment approach may cause too steep of a change in the weights of terms. When this happens the weight of a document tends to get concentrated into a very small number of terms. As a result there could a loss of information that can negatively affect the classification performance. The second is that in some cases, this simple one step processes may not sufficiently change the weights of the various terms. Consequently, the new representation will be similar to the original one, with almost no change in the classification accuracy. For this reason, our weight adjustment algorithm adopts a somewhat different approach that attempts to address these problems.

Our algorithms solve the first problem by changing the weights of the various features by a smaller factor than that indicated by their purity. In particular, for each term i, we scale its weight by $P_i^{1/\delta}$, where $\delta > 1$. Since, the purities are less than one, $P_i^{1/\delta}$ will be closer to one, thus leading to smaller changes. To address the second problem, we perform the weight-adjustment operation multiple times. For each data set, we use the classification accuracy on a portion of the training set (*i.e.*, validation set) in order to determine how many times to perform the weight adjustment. The weight-adjustment process is stopped when the classification performance on the validation set starts to decrease. The details of the algorithm are shown in Figure 1.

Once the number of weight adjustment iterations l has been computed, a new test document d is classified by first adjusting the weights of its terms by going through the same sequence of l weight adjustment iterations, and then

1. Split the training set T into training T' and validation V
2. Compute the accuracy on V of the centroid classifier built on T'
3. Compute $\vec{P}$ using the documents in T'
4. $l = 0$
5. For each document $d_i \in T$
6. For each term j
7. $d_{i,j} = P_j^{1/\delta} d_{i,j}$
8. Scale d_i so that $||d_i||_2 = 1$
9. Compute the accuracy on V of the centroid classifier built on T'
10. If accuracy does not decrease
11. $l = l + 1$
12. Goto 5

Figure 1: The fixed weight adjustment algorithm.

using the centroid classification algorithm on the weight-adjusted training set to determine its class. This process can be speeded up by using the fact that applying l iterations of weight-adjustment followed by unit-length scaling is the same as applying a single weight-adjustment in which the weight of each term j is multiplied by $P_j^{l/\delta}$, followed by a single unit-length scaling (see Appendix A for a proof).

Computational Complexity Any algorithm must be reasonably efficient to be of practical significance. One of the major advantages of the centroid-based classifier is that it is a linear-time classifier [12] which outperforms other more complex algorithms. In this section we discuss the effect of the feature weight adjustment algorithm on the computational complexity of the centroid-based algorithm. Both the fixed feature weight adjustment scheme and the centroid-based algorithms iterate over the document-term matrix. This matrix is a sparse matrix and is usually stored in a sparse representation. These representations have both space and time complexities of O (nnz), where nnz is the number of non-zeros in the document-term matrix. Such a representation of the document-term matrix is assumed in the following analysis.

With the fixed feature-weight adjustment step the classifier's learning stage consists of three steps. In the first step, the optimum number of iterations l is to be determined. In this step, weight adjustment is applied to some of the documents while the other documents are classified. Applying weights to a document and normalizing it is linear in the number of terms in the document, as is classifying a document. Each iteration is therefore O (nnz). From this, it is reasonable to assume that the determining l is O (nnz * l). Experimental observation indicate the l is a small constant (usually $l \leq 20$). Assuming constant l, the complexity of this step can be rewritten as O (nnz).

The second step consists of applying the weight vector l times to the complete training set. However as discussed in the previous section this can be done in 1 iteration through the document-term matrix. Using this optimization means the second step has complexity O (nnz).

The final step consists of computing the centroids of the transformed data set. The complexity of this step has been shown to be O (nnz) [12]. Putting the complexities of the three steps together, the overall complexity of the learning phase is O (nnz).

While classifying, the feature-weight vector is applied to the *tf-idf* representation , $\vec{x}$ of the test document to produce

the transformed test vector $\vec{x}'$. This step is O($|\vec{x}'|$) where $|\vec{x}'|$ is the number of terms in the document. Classifying this document using the centroid-based algorithm has complexity O($k * |\vec{x}'|$) where k is the number of classes (and hence the number of centroids).

Thus the fixed feature-weight adjustment does not affect the linear time complexity of the centroid-based scheme. This is important as the speed of the algorithm is one of its primary advantages.

4.2 Variable-Weight Adjustment

In the algorithm in Section (4.1) the feature-weight vector is computed once at from the original document-term matrix. After each application of the weight vector, the matrix changes. Both the centroid and the purity information change as a result. In this section, we present an algorithm which uses this changed purity information to recalculate the weight vector at each iteration. Since the weight vector changes at each iteration, this algorithm is called the variable-weight feature adjustment algorithm (VWA for short). This algorithm is summarized in Figure 2.

---

1. Split the training set T into training T' and validation V
2. Compute the accuracy on V of the centroid classifier built on T'
3. $l = 0$
4. Compute $\vec{P}^l$ using the documents in T'
5. For each document $d_i \in T$
6. For each term j
7. $d_{i,j} = (P_j^l)^{1/\delta} d_{i,j}$
8. Scale d_i so that $||d_i||_2 = 1$
9. Compute the accuracy on V of the centroid classifier built on T'
10. If accuracy does not decrease
11. $l = l + 1$
12. Goto 4

Figure 2: The variable weight adjustment algorithm.

For a test document x, with its *tf-idf* representation $\vec{x} = \{x_1, x_2, \ldots, x_t\}$, we compute the transformed test document as

$$\vec{x}' = \{x_1 w_1, x_2 w_2, \ldots, x_t w_t\} \tag{14}$$

This represents the test document in the transformed space and is used in computing similarity with the centroids. This weight vector can be computed during step 2 of the learning phase without any added complexity as

$$\vec{w} = \{w_1, \ldots, w_t\} = \{(P_1^1)^{1/\delta} \cdot (P_1^2)^{1/\delta} \ldots (P_1^l)^{1/\delta}, \ldots, (P_t^1)^{1/\delta} \cdot (P_t^2)^{1/\delta} \ldots (P_t^l)^{1/\delta}\} \tag{15}$$

This is a result of applying a weight (*i.e.*, multiplying each term and re-normalizing) being associative. (See Appendix A) We can track this vector by using a single vector initialized to 1 and updating it while computing purities in step (4) in Figure 2.

Computational Complexity Since the algorithm described is a modification of the algorithm in section (4.1), analysis here is based on the analysis in section (4.1). The main difference in the two algorithms is that once the optimum number of iterations l has been determined in the FWA algorithm, the training set can be weighted in one iteration. In the VWA algorithm, the weights are recomputed at the end of each iteration so l iterations need to be performed. As a result the complexity of this step becomes O (l * nnz) from O (nnz). However with the assumption that l is a constant, this revised algorithm does not affect the linear time complexity of the centroid-based algorithm. Computing the transformed test has complexity on the order of the number of terms in the test document. Again this does not affect the time complexity of the classification of the phase of the centroid-based classifier which remains O($k * |\vec{x}'|$).

4.3 Binary Classification

A common classification problem in information retrieval is that of developing a classifier that can correctly identify documents that belong to a particular *target* class from a large collection of documents. This is a typical binary classification problem in which we try to develop a model for the *target* class versus the *rest*. The weight adjustment scheme that we described in the previous sections can be directly used in this kind of problems. However, the centroid scheme does best when each class contains related documents. The negative class *i.e., rest* is however too diffuse for this. As a result instances of the negative class tend to get classified as positive. We propose the following solution to handle this. We cluster the negative set into k clusters. While computing the centroids and the purity we treat this as a $k+1$ class problem. When classifying, we compute the similarity of the document to the k negative clusters. We take the largest value amongst these, and treat it as the similarity of the document to the negative set. The similarity of the document to the positive set is directly computed from the centroid of the positive class. Similarly if we find that a class has a multi-modal distribution we can run a clustering algorithm to identify sub-classes within it. We could then treat each of these sub-classes as a separate class.

5 Analysis

We first present a model of how the weight adjustment process affects the documents. Consider the terms in a document sorted according to decreasing purity. When we apply the weight adjustment scheme to this document and re-normalize it, every term will gain some weight from the terms having less purity than it and lose some weight to purer terms. Consider the initial document vector as $\{d_1, d_2, \ldots, d_t\}$, where $\sum d_i^2 = 1$. Let us apply a weight vector $\vec{w} = \{w_1, w_2, \ldots, w_t\}$. We assume without loss of generality that $w_t \leq \ldots \leq w_2 \leq w_1 \leq 1$. Then the new document vector $\vec{d'}$ is given by

$$\vec{d'} = \{d'_1, d'_2, \ldots, d'_t\} \tag{16}$$

where

$$d'_i = \frac{w_i \cdot d_i}{\sqrt{\sum (w_j \cdot d_j)^2}} \tag{17}$$

We rewrite this as

$$d'_i = \frac{d_i}{\sqrt{L'_i + d_i^2 + G'_i}} \tag{18}$$

where $p_j = \frac{w_j}{w_i}$, $L'_i = \sum_{j=1}^{i-1}(p_j d_j)^2$, $p_j \geq 1$ and $G'_i = \sum_{j=i+1}^{t}(p_j d_j)^2$, $p_j \leq 1$. Also let $L_i = \sum_{j=1}^{i-1} d_j^2$ and $G_i = \sum_{j=i+1}^{t} d_j^2$. Thus $L'_i \geq L_i$ and $G'_i \leq G_i$

We also know that $L_i + d_i^2 + G_i = 1$. If $L'_i - L_i > G_i - G'_i$, then the denominator in (18) is greater than 1, thus $d'_i < d_i$ i.e. it loses weight. On the other hand if $L'_i - L_i < G_i - G'_i$ the denominator in (18) is less than 1 and $d'_i > d_i$. Thus even though all weights are less than or equal to 1 a term can actually gain weight.

We assume the weights remain constant through each iteration. An impure term may gain weight due to the presence of other terms of even lesser purity. However as more iterations are performed, the weight transfer process causes these terms to have lesser weight and thus reduces the weight transfer into higher terms. As a result of this process, initially only the terms having the lowest purity in the document will lose weight. As these lose weight, terms which are more pure will no longer be able to compensate their loss of weight from these terms and will also start losing weight. Thus the weight of each term will have a curve that looks like Figure 3 (B). The term having the low purity (Figure 3(A)) does not show the initial increase while the purest term (Figure 3(C)) does not exhibit the final falling part. The figures shows the change in the weight of a 3 terms with different purities in the same document for 10 iterations.

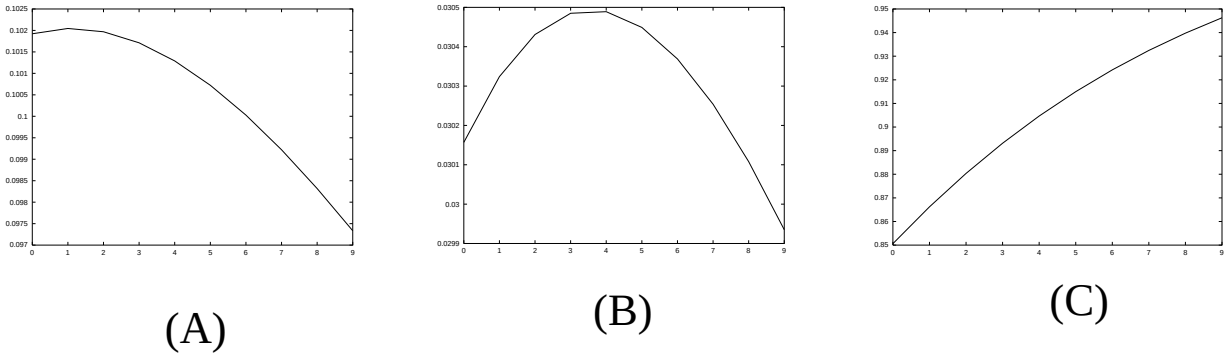

Figure 3: Column weight against number of iterations

We assume the following document model. Consider k classes $C_1 \ldots C_k$. Each class has a specialized vocabulary V_i. In addition there is a general vocabulary G. This model is illustrated in Figure 4. We assume the most general case that $\forall i, V_i \bigcap G \neq \emptyset$ and $\forall i, j, V_i \bigcap V_j \neq \emptyset$. Consider the set of words $G - (V_1 \bigcup \ldots \bigcup V_k)$. It is a reasonable assumption to make that the words belonging to this class do not have any affinity to a particular class. Thus these terms will have a purity close to $\frac{1}{k}$. As we perform the weight adjustment process, these terms will tend to go to zero first. Since these terms do not have any discriminant ability, there is no loss of information by weight transfer out of these terms. The next terms that would tend to go to zero would be those that occur about equally in $k-1$ of the classes. Thus this process removes terms in increasing order of discriminating ability. The process should stopped when the new representation starts losing terms which are important for discriminating between classes. This is why the validation portion of the training set is needed.

The algorithm in section (4.2) updates the purity vector after each iteration. The analysis is more complicated in this case. Each time the weight vector is applied the change of weights of terms in centroids is related to term dependencies. Consider two terms A and B which are perfectly correlated. This would mean their *tf-idf* values in each document vector are the same, and hence so are the centroid values, $C_{1,A} = C_{1,B}, C_{2,A} = C_{2,B}, \ldots, C_{k,A} = C_{k,B}$ as well as the purities $P_A = P_B$. Now when we apply the weight vector, and re-normalize we know that there is some

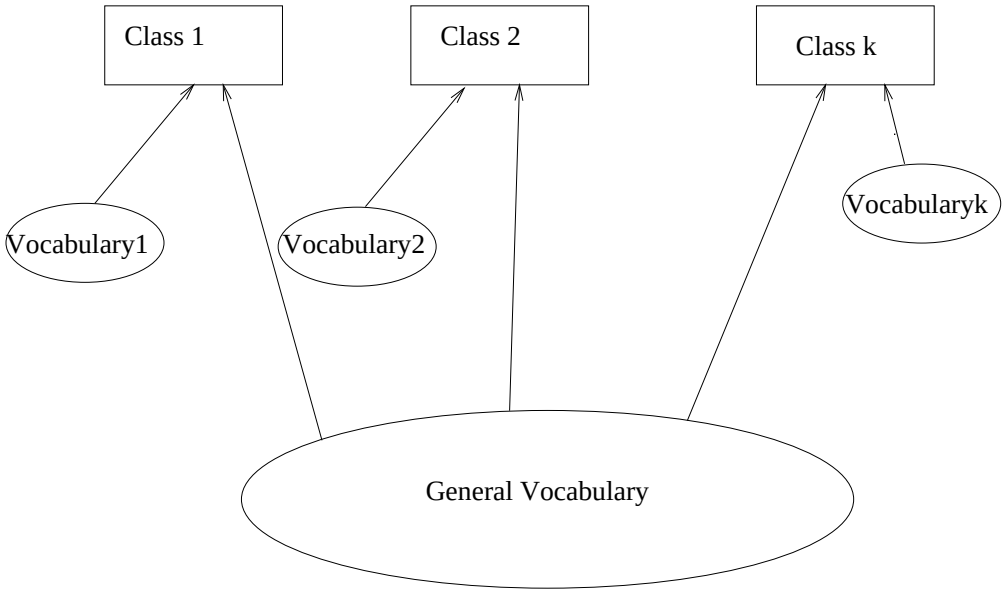

Figure 4: Document Model

weight transfer from A to each term and vice-versa. It is easy to see that the weight transfer from A to B is canceled by the weight transfer from B to A. If A and B were not correlated then there would be a net transfer from one to the other. Thus term dependencies affect the weight transfer within a document and as a result affect weights in the centroids. Recomputing the purity after each iteration uses these changed weights. We hypothesize that this allows the scheme to capture some information about the dependencies which caused the weight change in the first place.

6 Experimental Setup and Results

In this section we experimentally evaluate the effect of using feature weight adjustment schemes on the classification accuracy of a centroid based classifier. Three different sets of experiments are presented. The first two experiments focus on evaluating the accuracy of the classifier on data sets with multiple labels per document, by considering binary classification for each class. The third evaluates the classifier as a k way classifier.

In our experiments, we compared the performance of our weight adjustment schemes against the performance achieved by the following classifiers. We obtained results using two linear classifiers which used Rocchio and Widrow-Hoff respectively to learn the weight vectors. In the case of Rocchio we used $\alpha = 0, \beta = 16, \gamma = 4$ [25], whereas in WH we used $\eta = 0.5$ and an initial vector $w_1 = \{0, 0, \ldots, 0\}$. We also ran SVM^{lite} [16] using a polynomial kernel with d = 1 and a RBF kernel with $\gamma = 0.8$ against the data sets. For the feature weight adjustment schemes we used $\delta = 8$. Other classifiers used include k Nearest Neighbor with k = 10, C4.5 and the Naive Bayesian classifier. For the naive bayesian we used Rainbow [29] with the multinomial event model. Both the feature weighting schemes were run with $\delta = 8$. All the classifiers except Naive Bayesian and C4.5 were run against the *tf-idf* representation of the documents. These two were run against the boolean representation in which a particular entry $d_i j$ is 1 if term t_j occurs in document d_i.

6.1 Reuters

The first data set that we used was the Reuters-21578 [24] text collection. In particular, we used the "ModApte" split to divide the text collection into a set of 9603 training documents and 3299 test documents. After eliminating stop-words and removing terms that occur less than two times, the training corpus contains 11,001 distinct terms.

Table 1 shows the performance of the classifiers on the 10 most frequent classes in the Reuters data set. The columns labeled "Rocchio" and "WH" shows the performance achieved by a linear classifier using the Rocchio and Widrow-Hoff algorithms respectively to learn the weights. The next two columns show the performance of the SVM classifier using a degree one polynomial kernel and a RBF kernel. The fifth column labeled "Centroid" shows the performance of the centroid classifier. The last two column shows the performance of the centroid classifier after feature weights have been adjusted by the fixed feature-weight adjustment (FWA) and the variable feature-weight adjustment (VWA) algorithms respectively. Note that these algorithms were run without using clustering. Table 1 also shows the the micro-average [42] over all classes, the micro-average over the top 10 classes and the macro-average over the top 10 classes.

	Rocchio	WH	SVM(poly)	SVM(rbf)	Centroid	FWA	VWA
earn	96.23	95.86	98.62	98.71	93.74	96.32	96.41
acq	79.00	87.60	91.24	95.27	91.79	91.93	91.93
money-fx	55.31	67.04	70.39	78.21	63.68	66.48	66.48
grain	77.85	79.19	91.94	93.29	77.85	77.85	77.85
crude	75.66	72.49	86.77	89.42	85.71	84.12	84.12
trade	73.5	68.38	70.94	76.92	77.78	77.78	77.78
interest	70.23	66.41	63.36	74.81	75.56	75.56	75.56
wheat	74.65	85.92	78.87	84.51	74.65	80.28	80.28
ship	79.77	73.03	77.53	85.39	85.39	84.27	84.27
corn	60.71	64.29	80.36	85.71	62.5	62.5	62.5
Micro-average (top 10)	82.81	85.25	89.37	92.5	87.01	88.23	88.27
Micro-average (all)	76.73	76.57	83.49	86.62	80.62	81.4	81.42
Average (top 10)	74.29	76.02	81.00	86.22	78.87	79.71	79.72

Table 1: Precision / Recall break even points on Reuters

A number of interesting observations can be made from the results in this table 1. First, comparing Rocchio, Widrow-Hoff and the basic centroid scheme (the three fastest schemes), we see that overall the centroid scheme performs substantially better than the rest followed by WH and then Rocchio. In 6 of the top 10 categories the centroid scheme does best with WH dominating in the remaining 4. Second, we see that the weight adjustment schemes improve the performance of the centroid classifier, sometimes dramatically. However both weight adjustment schemes achieve the same level of performance. Third, SVM using a RBF kernel is the overall winner doing about 5% better than the other schemes.

In addition to this we also tested the effect of clustering of the negative set (as described in section 4.3). These results are presented in table 2 for 5, 10, 15 and 20 clusters. As can be seen clustering has a dramatic improvement in the performance of the scheme. The number of clusters only slightly affects overall performance but using 10 clusters gives the best results. Comparing the results after clustering with the SVM results we see that the SVM (poly) scheme now has an overall micro-average about one percent less than using weight adjustment. SVM (rbf) does better by about 2% now. The weight adjustment schemes dominate SVM(rbf) in 3 of the top 10 classes. Once again both weight adjustment schemes achieve the same performance.

	FWA				VWA			
	5	10	15	20	5	10	15	20
earn	95.58	95.76	94.94	94.66	95.68	95.21	94.94	94.66
acq	94.02	94.16	91.93	92.9	94.02	94.16	91.93	92.9
money-fx	72.07	77.1	77.1	77.65	73.18	76.54	77.1	77.65
grain	85.23	91.28	87.92	88.59	85.23	91.95	93.96	88.59
crude	85.71	84.66	86.24	86.24	85.71	84.65	86.24	86.24
trade	77.78	79.49	79.49	78.63	77.78	79.49	79.49	78.63
interest	71.76	73.28	74.05	74.05	71.76	73.28	74.05	74.05
wheat	87.32	87.32	85.92	85.92	85.92	87.33	85.92	85.92
ship	82.02	85.39	84.27	84.27	82.02	85.39	84.27	84.27
corn	76.79	87.5	85.71	87.5	76.79	87.5	85.71	87.5
Micro-average (top 10)	89.56	90.71	89.67	89.88	89.63	90.49	89.99	89.88
Micro-average (all)	83.16	84.69	84.25	84.33	83.21	84.59	84.56	84.35
Average (top 10)	82.83	85.59	84.76	85.04	82.81	85.55	85.36	85.04

Table 2: Effect of clustering

6.2 OHSUMED results

Table 3 gives the same data in Table 1 for the OHSUMED [14] data set. We used from the OHSUMED data, those documents with id's between 100000 and 120000 which had either the title or both the title and the abstract. The classification task considered here is to assign the document to one or multiple categories of the 23 MeSH "diseases" categories.. There were 19858 entries in the data set. The first 12000 of these were used in the training set and the remaining formed the test set. A total of 6561 documents did not have class label assigned to them. 6199 documents belonged to multiple classes.

Once again comparing Rocchio, Widrow-Hoff and the centroid scheme we see that the centroid scheme performs the best among the three on this data set. Rather surprisingly Widrow-Hoff has a very poor performance on this data set and is dominated completely by Rocchio, performing better in only 2 of the 23 categories. The centroid based scheme dominates both of them in all 23 categories. Both weight adjustment schemes achieve similar performances and improve the accuracy of basic centroid scheme by about 3 %. Even without clustering the schemes achieve a higher micro-average than the SVM(poly) scheme and perform better than it in 16 of the 23 classes. SVM(rbf) again performs the best. It achieves a micro-average about 5% higher than the FWA scheme and performs better in 22 of the 23 categories.

The results for the weight-adjustment schemes after clustering are shown in in table 4. Clustering has almost no effect on accuracy. In fact in some cases it actually reduces accuracy. Overall using 5 clusters gives the best result for this data set and the results are about 4-5% lower than those for the SVM(rbf) scheme. One interesting result in this table is that each of the different levels of clustering improves accuracy in some of the data sets For example, for FWA no clustering gives the best results for classes c23 and c0, while using 5 clusters gives the best result for classes c14 and c04. Similarly trends can be seen in the other classes.

6.3 Multi-Class results

Our final set of results are from using the centroid-based classifier as a k-way classifier. Table 6.3 shows the performance of the various classifiers. The column "NB" shows the performance of the naive-bayesian classifier. "C4.5" shows the performance of the standard decision tree classifier while "kNN" gives the the performance of the k nearest neighbor algorithm.

	rocc	wh	SVM(poly)	SVM(rbf)	centroid	FWA	VWA
c23	38.65	42.16	47.70	56.38	51.59	53.19	53.38
c20	46.72	37.80	57.48	70.08	53.81	64.83	65.88
c14	58.26	56.03	67.41	75.11	69.53	71.99	72.21
c04	48.96	46.19	51.21	64.19	59.34	60.21	60.21
c06	47.26	38.23	59.82	68.32	62.83	66.37	66.19
c21	47.51	42.04	52.02	58.57	57.96	60.10	60.10
c10	31.16	25.00	45.55	57.53	41.10	51.71	51.37
c08	43.96	37.92	51.34	58.72	52.35	54.36	54.36
c19	54.85	36.08	62.45	73.20	60.13	61.39	61.39
c17	31.45	20.49	50.53	59.72	51.94	58.30	59.01
c01	40.25	36.96	52.66	61.52	52.15	52.15	52.15
c05	29.51	24.59	46.45	56.83	42.08	43.17	43.17
c13	39.65	34.62	50.89	61.83	51.48	53.25	53.25
c12	46.12	38.36	52.97	61.64	50.91	51.37	51.37
c15	16.57	7.73	46.41	53.89	40.33	51.93	48.62
c16	43.34	33.10	47.44	58.36	50.85	52.21	52.22
c18	23.08	9.40	41.45	45.30	27.35	32.05	32.05
c11	52.73	14.55	60.91	66.36	62.73	63.63	63.64
c07	35.52	19.74	36.84	46.05	42.11	42.11	42.11
c09	56.69	24.41	63.78	66.93	58.27	62.20	62.20
c22	13.43	20.90	13.64	16.42	17.91	16.42	16.42
c03	36.13	21.85	42.86	50.42	37.81	37.82	37.82
c02	34.15	12.20	51.22	56.10	50.00	50.00	50.00
average	39.82	29.58	50.13	58.41	49.76	52.64	52.57
micro-average	43.30	36.98	53.12	62.23	53.89	57.04	57.12

Table 3: OHSUMED results

The detailed characteristics of the various document collections used in this experiments are available in [12] [1]. Note that for all data sets, we used a stop-list to remove common words, and the words were stemmed using Porter's suffix-stripping algorithm [30]. Furthermore, we selected documents such that each document has only one class (or label). In other words, given a set of classes, we collected documents that have only one class from the set.

The first three data sets *west1, west2, west3* are from the statutory collections of the legal document publishing division of West Group described in [7]. Data sets *tr11, tr12, tr21, tr23, tr31, tr41, tr45, fbis, la1, la2, la12*, and *new3* are derived from TREC-5 [34], TREC-6 [34], and TREC-7 [34] collections. Data sets *re0* and *re1* are from Reuters-21578 text categorization test collection Distribution 1.0 [24]. We removed dominant classes such as "earn" and "acq" that have been shown to be relatively easy to classify. We then divided the remaining classes into 2 sets. Data sets *oh0, oh5, oh10, oh15*, and *ohscal* are from OHSUMED collection [14] subset of MEDLINE database. Data set *wap* is from the WebACE project (WAP) [3]. Each document corresponds to a web page listed in the subject hierarchy of Yahoo! [40].

Table 6.3 shows the performance of the weight adjustment scheme as a k way classifier. The accuracy was measured by performing a 80-20 split on the data and a $10\times$ cross-validation. The numbers in bold faces indicate the winning algorithms on a particular data set. Between them the three centroid based schemes dominate in 19 of the 23 classes.If we compare k-NN, naive bayesian and C4.5 amongst each other we see that C4.5 does the worst followed by k-NN and Naive Bayesian performs the best. The basic centroid scheme dominates all three and weight adjustment tends to improve its performance.

[1] These data sets are available from *http://www.cs.umn.edu/~han/data/tmdata.tar.gz*.

	FWA				VWA			
	5	10	15	20	5	10	15	20
c23	50.19	43.37	42.73	41.14	50.51	43.37	42.79	41.14
c20	59.97	55.38	48.16	42.26	60.24	55.38	46.98	43.89
c14	71.99	69.98	65.74	65.29	74.11	69.87	66.96	64.17
c04	60.21	56.06	55.88	47.23	61.94	54.84	55.36	44.81
c06	64.96	64.07	62.48	58.23	64.96	64.53	62.65	57.87
c21	61.05	56.29	55.34	54.63	60.57	56.29	56.53	55.00
c10	54.79	54.11	46.92	45.21	54.79	53.77	47.95	43.84
c08	53.36	50.00	48.66	51.01	53.36	50.00	48.32	47.99
c19	69.20	70.25	70.04	63.71	69.41	70.25	70.04	64.14
c17	58.66	58.66	56.54	56.18	59.01	59.36	57.19	53.36
c01	56.71	56.20	60.76	57.97	56.71	56.20	61.01	57.21
c05	53.01	51.37	50.82	48.09	53.01	49.18	51.91	48.63
c13	52.66	52.37	55.03	54.44	52.67	53.25	54.44	55.33
c12	54.11	54.34	55.71	55.38	54.34	59.13	55.71	56.75
c15	49.72	45.86	43.82	41.98	49.17	44.75	41.99	39.44
c16	51.88	54.27	49.83	49.49	51.88	54.61	49.82	50.69
c18	40.17	37.18	36.32	29.49	40.60	38.46	35.47	29.49
c11	60.00	58.18	60.91	62.73	60.00	58.18	60.91	65.45
c07	42.11	40.79	42.11	40.79	42.11	40.79	42.11	40.79
c09	64.57	61.42	65.35	66.93	64.57	61.42	65.35	66.14
c22	17.91	17.91	14.93	13.43	17.91	17.91	14.93	13.43
c03	42.86	41.18	47.90	48.74	42.86	41.18	47.06	48.74
c02	43.90	47.56	48.78	53.75	45.12	47.56	48.78	50.00
average	53.79	52.03	51.51	49.92	53.90	52.19	51.49	49.49
micro-average	57.51	54.57	53.21	50.78	57.67	54.76	53.25	50.48

Table 4: OHSUMED clustering

In all three data sets, it can be seen that the VWA scheme does not do appreciably better than the FWA scheme. We believe that the VWA scheme has some inbuilt mechanisms for handling term dependencies based on change in centroid weights and the resulting change in purities. However the results do not seem to indicate this. Part of the reason seems to be that the purity changes relatively slowly because of our mechanism for dampening weight transfer. Since the number of iterations is small in most of the data sets, this means that the changes in purity, even added up, are probably not substantial enough to make a difference in the classification accuracy.

6.4 Efficiency

One of the advantages of our weight adjusted centroid scheme is its speed. As discussed in section 4 model learning time is linear in the number of non zero terms in the document-term matrix and classification time is linear in number of classes. A comparison of running time was performed between the svm_lite [16] code with the polynomial kernel, the RBF kernel and the centroid based scheme with VWA and is reported in Table 6. These times were obtained on a P3 500MHz workstation running Redhat 6 with 1 Gig of memory, under similar load conditions. Looking at this table we can see that VWA is about 2 - 10 times faster than SVM(rbf) in the learning phase and about 10 - 20 times faster in the classification phase.

	NB	C4.5	*k*NN	centroid	FWA	VWA
west1	86.0	85.5	82.9	90.8	92	**92.2**
west2	76.5	75.3	77.2	82.0	**84.2**	83.9
west3	75.1	73.5	76.1	83.0	83.6	**83.9**
oh0	89.1	82.8	84.4	90.8	92	**92.2**
oh5	87.1	79.6	85.6	89.3	91.4	**91.7**
oh10	81.2	73.1	77.5	81.9	**83.9**	**83.9**
oh15	84.0	75.2	81.7	85.9	86.4	**86.5**
re0	**81.1**	75.8	77.9	78.2	77.3	77.2
re1	80.5	77.9	78.9	**82.7**	82.4	82.4
tr11	85.3	78.2	85.3	86.5	86.9	**87**
tr12	79.8	79.2	85.7	**89.7**	88.5	88.5
tr21	59.6	81.3	89.2	90.7	91.1	**91.3**
tr23	69.3	90.7	81.7	87.0	**91.7**	91.1
tr31	94.1	93.3	93.9	94.5	**95.4**	**95.4**
tr41	94.5	89.6	93.5	**95.8**	95.6	95.7
tr45	84.7	91.3	91.1	94.7	**94.9**	**94.9**
la1	**87.6**	75.2	82.7	86.4	86.7	86.7
la2	**89.9**	77.3	84.1	88.5	88.4	88.2
la12	**89.2**	79.4	85.2	88.0	87.9	87.9
fbis	77.9	73.6	78.0	79.2	**79.3**	**79.3**
wap	80.6	68.1	75.1	**82.5**	**82.5**	**82.5**
ohscal	74.6	71.5	62.5	76.7	80.4	**80.5**
new3	74.4	73.5	67.9	**80.2**	80.1	80.1

Table 5: Classification Accuracy

7 Conclusion

In this report we showed how a weight adjustment scheme improves the accuracy of a centroid based classifier. This scheme retains the power of the centroid based classifier while further enhancing its ability. Also it retains much of the speed of the original scheme. In terms of future work, clustering has been shown to be useful in improving the accuracy of this scheme. Clustering is needed in order to handle multi-modal distributions which this scheme cannot handle in its current form. Automatically determining whether a class needs to be clustered and how many clusters it should be divided into would be an interesting problem. As it stands the analysis of the algorithm is still incomplete. It would be beneficial to have a more rigorous analysis of this scheme and its strengths and weaknesses.

	SVM(poly)		SVM(rbf)		VWA (10 clusters)	
	learn	classify	learn	classify	learn	classify
earn	66	17	136	36	19	1
acq	103	17	190	42	20	1
money-fx	125	10	144	23	24	1
grain	33	8	71	19	29	1
crude	46	9	78	20	21	1
trade	63	9	88	23	22	2
interest	146	7	119	15	23	1
wheat	39	5	55	11	24	2
ship	30	5	59	15	25	1
corn	25	6	41	11	31	1

Table 6: Run time comparison (all times in seconds)

References

[1] M. B. Amin and S. Shekhar. Generalization by neural networks. *Proc. of the 8th Int'l Conf. on Data Eng.*, April 1992.

[2] L. Baker and A. McCallum. Distributional clustering of words for text classification. In *SIGIR-98*, 1998.

[3] D. Boley, M. Gini, R. Gross, E.H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Document categorization and query generation on the world wide web using WebACE. *AI Review (accepted for publication)*, 1999.

[4] W.W. Cohen. Fast effective rule induction. In *Proc. of the Twelfth International Conference on Machine Learning*, 1995.

[5] W.W. Cohen and H. Hirsh. Joins that generalize: Text classification using WHIRL. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, 1998.

[6] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[7] T. Curran and P. Thompson. Automatic categorization of statute documents. In *Proc. of the 8th ASIS SIG/CR Classification Research Workshop*, Tucson, Arizona, 1997.

[8] D.J. Spiegelhalter D. Michie and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[9] W. Daelemans, S. Gills, and G. Durieux. Learnability and markedness in data-driven acquisition of stress. Technical Report TR 43, Institute for Language Technology and Artificial Intelligence, Tilburg University, Netherlands, 1993.

[10] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimizations and Machine Learning*. Morgan-Kaufman, 1989.

[12] E.H. Han and G. Karypis. Centroid-based document classification algorithms: Analysis & experimental results. Technical Report TR-00-017, Department of Computer Science, University of Minnesota, Minneapolis, 2000. Available on the WWW at URL *http://www.cs.umn.edu/~karypis*.

[13] Eui-Hong Han. *Text Categorization Using Weight Adjusted* k*-Nearest Neighbor Classification*. PhD thesis, University of Minnesota, October 1999.

[14] W. Hersh, C. Buckley, T.J. Leone, and D. Hickam. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *SIGIR-94*, pages 192–201, 1994.

[15] Makato Iwayama and Takenobu Tokunaga. Cluster-based text categorization: a comparison of category search strategies. In *SIGIR-95*, pages 273–281, 1995.

[16] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of the European Conference on Machine Learning*, 1998.

[17] K. Kira and L.A. Rendell. A practical approach to feature selection. In *Proc. of the 10th International Conference on Machine Learning*, 1992.

[18] I. Kononenko. Estimating attributes: Analysis and extensions of relief. In *Proc. of the 1994 European Conference on Machine Learning*, 1994.

[19] Wai Lam and Chao Yang Ho. Using a generalized instance set for automatic text categorization. In *SIGIR-98*, 1998.

[20] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *Proc. of the Fifth ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 16–22, 1999.

[21] D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Tenth European Conference on Machine Learning*, 1998.

[22] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *SIGIR-94*, 1994.

[23] D. Lewis and M. Ringuette. Comparison of two learning algorithms for text categorization. In *Proc. of the Third Annual Symposium on Document Analysis and Information Retrieval*, 1994.

[24] D. D. Lewis. Reuters-21578 text categorization test collection distribution 1.0. *http://www.research.att.com/~lewis*, 1999.

[25] David D. Lewis, Robert E. Shapire, James P. Callan, and Ron Papka. Training algorithms for linear text classifiers. In *Proceedings of the 19 th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages pages 298–306, 1996.

[26] D.G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, pages 72–85, January 1995.

[27] B. Masand, G. Linoff, and D. Waltz. Classifying news stories using memory based reasoning. In *SIGIR-92*, pages 59–64, 1992.

[28] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[29] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow, 1996.

[30] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[31] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[32] J.J. Jr. Rocchio. The SMART retrieval system: Experiments in automatic document processing. In Gerard Salton, editor, *Relevance feedback in information retrieval*. Prentice-Hall, Inc., 1971.

[33] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[34] TREC. Text REtrieval conference. *http://trec.nist.gov*.

[35] V. Vapnic. *The Nature of Statistical Learning Theory*. Springer, 1995.

[36] S.M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann, San Mateo, CA, 1991.

[37] D. Wettschereck, D.W. Aha, and T. Mohri. A review and empirical evaluation of feature-weighting methods for a class of lazy learning algorithms. *AI Review*, 11, 1997.

[38] D. Wettschereck and T.G. Dietterich. An experimental comparison of the nearest neighbor and nearest hyperrectangle algorithms. *Machine Learning*, 19:5–28, 1995.

[39] B. Widrow and S.D. Stearns. *Adaptive Signal Processing*. Prentic-Hall, Inc., 1985.

[40] Yahoo! Yahoo! http://www.yahoo.com.

[41] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *SIGIR-94*, 1994.

[42] Y. Yang and X. Liu. A re-examination of text categorization methods. In *SIGIR-99*, 1999.

[43] Y. Yang and J. Pederson. A comparative study on feature selection in text categorization. In *Proc. of the Fourteenth International Conference on Machine Learning*, 1997.

A Associativity of weighing

We wish to show that applying a weight vector $\vec{W}^i = \{W_1^i, W_2^i, \ldots, W_t^i\}$, during the i^{th} iteration and normalizing the vector after each iteration for L iterations is the same as applying the vector $\vec{W} = \{\Pi_{i=1}^L W_1^i, \Pi_{i=1}^L W_2^i, \ldots, \Pi_{i=1}^L W_t^i\}$ and normalizing once. This can be shown by induction. Consider a vector $\vec{a} = \{a_1, a_2, \ldots, a_t\}$. Applying the weight vector $\vec{W}^1$ once and normalizing it gives us the new vector $\vec{a}^1 = \{\ldots \frac{W_i^1 a_i}{\sqrt{\sum (W_i^1)^2 \cdot a_i^2}} \ldots\}$. This is the base case. Let us assume that k applications of this gives us the vector

$$\vec{a}^k = \{\ldots, \frac{(\Pi_{r=1}^k W_i^r) a_i}{\sqrt{\sum (\Pi_{r=1}^k W_i^r)^2 a_i^2}} \ldots\} \tag{19}$$

Applying the weight vector $\vec{W}^{k+1} = \{W_1^{k+1}, W_2^{k+1}, \ldots, W_t^{k+1}\}$ gives us

$$\{\ldots, \frac{(\Pi_{r=1}^{k+1} W_i^r) a_i}{\sqrt{\sum (\Pi_{r=1}^{k} W_i^r)^2 a_i^2}} \ldots\} \tag{20}$$

The magnitude of this vector is $\sqrt{\frac{\sum (\Pi_{r=1}^{k+1} W_i^r)^2 \cdot a_i^2}{\sum (\Pi_{r=1}^{k} W_i^r)^2 a_i^2}}$. Thus the normalized vector is obtained by dividing vector in (20) by its magnitude which gives us

$$\{\ldots, \frac{(\Pi_{r=1}^{k+1} W_i^r) \cdot a_i}{\sqrt{\sum (\Pi_{r=1}^{k=1} W_i^r)^2 a_i^2}} \ldots\} \tag{21}$$

Thus having shown that the statement is true for $L = 1$ and that if it is true for k, it is true for $k + 1$, the statement is true for all L.

The optimization discussed in Section (4.1) is a special case in which $W_i^1 = W_i^2 = \ldots = W_i^L$.