Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification*

Eui-Hong (Sam) Han, George Karypis and Vipin Kumar

Department of Computer Science & Engineering, University of Minnesota
4-192 EE/CSci Building
200 Union Street SE, Minneapolis, MN 55455
{han, karypis, kumar}@cs.umn.edu

Abstract. Automatic text categorization is an important task that can help people finding information on huge online resources. Text categorization presents unique challenges due to the large number of attributes present in the data set, large number of training samples, attribute dependency, and multi-modality of categories. Existing classification techniques have limited applicability in the data sets of these natures. In this paper, we present a Weight Adjusted k-Nearest Neighbor (WAKNN) classification that learns feature weights based on a greedy hill climbing technique. We also present two performance optimizations of WAKNN that improve the computational performance by a few orders of magnitude, but do not compromise on the classification quality. We experimentally evaluated WAKNN on 52 document data sets from a variety of domains and compared its performance against several classification algorithms, such as C4.5, RIPPER, Naive-Bayesian, PEBLS and VSM. Experimental results on these data sets confirm that WAKNN consistently outperforms other existing classification algorithms.

Keywords: : text categorization, k-NN classification, weight adjustments

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 intranet. These documents (with other unstructured data) will become the predominant data type stored online. Automatic text categorization [24, 16, 10], which is the task of assigning text documents to prespecified classes (topics or themes) of documents, is an important task that can help people finding information on these huge resources.

* This work was supported by NSF ACI-9982274, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE grant LLNL/DOE B347714, 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/~han

Text categorization presents unique challenges due to the large number of attributes present in the data set, large number of training samples, attribute dependency, and multi-modality of categories. Existing classification algorithms [18, 2, 6, 10, 24] address these challenges to varying degrees [7].

k-nearest neighbor (k-NN) classification is an instance-based learning algorithm that has shown to be very effective for a variety of problem domains in which underlying densities are not known [6]. In particular, this classification paradigm works well in the data sets with multi-modality. It has been applied to text categorization since the early days of research [23], and has been shown to produce better results when compared against other machine learning algorithms such as C4.5 [18] and RIPPER [2]. In text categorization task, the class of a new text document is determined by computing the similarity between the test document and individual instances of the training documents, and determining the class based on the class distribution of the nearest instances. A major drawback of this algorithm is that it uses all the features while computing the similarity between a test document and training documents. In many text data sets, relatively small number of features (or words) maybe 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 (or words).

In this paper, we present a Weight Adjusted k-Nearest Neighbor (WAKNN) classification that learns weights for words. WAKNN finds the optimal weight vector using an optimization function based on the leave-one-out cross validation and a greedy hill climbing technique. WAKNN has better classification results than many other classifiers, but it has a high computational cost. One of the key challenges of the weight adjustment algorithm is how to reduce the high computational cost. We present two performance optimizations of WAKNN in this paper. The first optimization intelligently selects words used for weight adjustment. The second optimization reduces the computational cost involved with the evaluation of weight adjustments by clustering documents within each class. Experimental results show that these two optimizations do not compromise the quality of the classification, but improve the computational performance by a few orders of magnitude. We experimentally evaluated WAKNN-C, which is the performance-improved version of WAKNN, on 52 document data sets from a variety of domains and compared its performance against several classification algorithms, such as C4.5 [18], RIPPER [2], Naive-Bayesian [16], PEBLS [3], and Variable-Kernel Simulation Metric (VSM) [15]. Experimental results on these data sets show that WAKNN-C consistently outperforms other existing classification algorithms.

Section 2 describes challenges of text categorization and provides a brief overview of existing algorithms for text categorization. Section 3 presents the weight adjustment algorithm WAKNN and its performance improvements. Section 4 shows experimental evaluation of the WAKNN. Finally, Section 5 provides conclusions and directions for future research.

2 Previous Work

Text categorization is essentially a classification problem [22, 18, 2, 6]. The words occurring in the document sets become variables or features for the classification problem. The class profile or model is described based on these words. There are several classification schemes that can potentially be used for text categorization.

A classification 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 [18] 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 set is that there is a relatively large number of features that determine class. Decision tree based schemes like C4.5 or rule induction algorithms such as C4.5rules [18] and RIPPER [2] do not work very well in this scenario due to overfitting [7]. The overfitting occurs, because the number of samples is relatively small with respect to the distinguishing words, which leads to very large trees with limited generalization ability.

The Naive-Bayesian (NB) classification algorithm has been widely used for document classification, and has been shown to produce very good performance [13, 16]. Given a document, the Naive-Bayes classification tries to predict the probability that a document belongs to different classes according to the words present in the document. In particular, given a test document, NB computes the posterior probability of each class given the test document, and then selects the class with the highest probability as the class of the test sample. Even though Naive-Bayes classification techniques, such as Rainbow [16], are commonly used in text categorization [13, 16], the independence assumption limits their applicability in the document classes and unimodal density assumptions might not work well in document data sets with multi-modal densities [6].

There have been several approaches to learn feature weights for k-NN. PEBLS [3] and k-NN with mutual information [22, 16] are approaches that compute weights of features in prior to the classification learning. A problem with these approaches is that it computes the importance of a feature independent of all the other features. Variable-Kernel Simulation Metric (VSM) [15] learns the feature weights using non-linear conjugate gradient optimization. VSM has a very structured approach to find weights, but requires optimization functions to be differentiable and does not have the convergence guarantees like the linear conjugate gradient optimization. RELIEF-F [12] is another weight adjustment technique that learns weights based on the nearest neighbors in each class.

Support Vector Machines (SVM) is a new learning algorithm that was introduced to solve two-class pattern recognition problem using the Structural Risk Minimization principle [10]. Given a training set in a vector space, this method finds the *best* decision hyperplane that separates two classes. An efficient implementation of SVM and its application in text categorization of Reuters-21578 corpus is reported in [10].

3 Weight Adjusted k-Nearest Neighbor Classification (WAKNN)

The Weight Adjusted k-Nearest Neighbor Classification (WAKNN) tries to learn the best weight vector for classification. Given the weight vector W, the similarity between two documents X and Y is measured using the weighted cosine measure [19]:

$$cos(X,Y,W) = \frac{\sum_{t \in T}(X_t \times W_t) \times (Y_t \times W_t)}{\sqrt{\sum_{t \in T}(X_t \times W_t)^2} \times \sqrt{\sum_{t \in T}(Y_t \times W_t)^2}},$$

where T is the set of terms (or words), X_t and Y_t are normalized text frequency (TF) of word t for X and Y, respectively, and W_t is the weight of word t.

Figure 1 illustrates the objective of weight adjustment in the classification task. In the original data without weights, the test sample of class A is equally close to the training samples in class A and class B. After weight adjustment, it is much closer to the training samples in class A and thus is correctly classified.

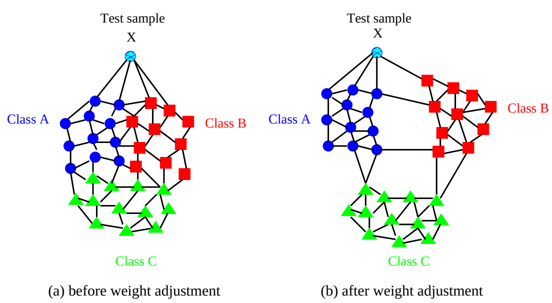

Fig. 1. Weight Adjustment in k-NN Classification.

We can visually see the effect of weight adjustment on real data sets using the class-preserving projection [5]. This projection tries to find the optimal 2-dimensional display of high-dimensional data sets with class labels. This projection works the best when there are 3 classes in the data sets, because 3 class means automatically determine a plane for 2-dimensional display. Hence we selected 3 classes from the training data set of *west1*, which is described in Section 4, and learned weights using WAKNN. Figure 2 shows the original data set on the left and the weight-adjusted data set on the right. Figure 2 shows that WAKNN was able to find weights that can separate data points in different classes. For instance, data points in class "010" (depicted as "x" in the figure) are almost completed separated from the other two classes. Classes "008" (depicted as "+") and "054" (depicted as "*") are also well separated except several points mixed up around the coordinate (-0.1, -0.2).

The weight adjustment process is a search process in the word weight vector space. The key design decisions in this process include which optimization function to use, which search method to use, and how to make the algorithm efficient in terms of run time and memory requirements. In the rest of this section, we will discuss these design decisions and present the algorithm.

3.1 Optimization Function and Search Strategy

A natural choice for optimization function in the k-NN classification paradigm is a function based on the leave-one-out cross validation. The overall value for

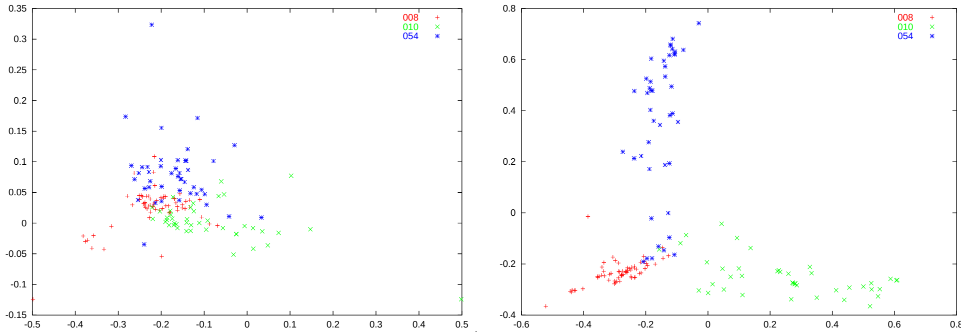

Fig. 2. 2-dimensional display of west1 data set before and after weight adjustments.

the optimization function is computed based on the contribution of each training sample used in the cross validation. Each training sample finds k-nearest neighbors using a similarity function with the current weight vector. The contribution of each training sample in the optimization function depends on the class labels of these k-nearest neighbors and similarities to these neighbors. The training sample should contribute to the overall optimization function only if it is correctly classified based on its k-nearest neighbors and its classification is clear cut. One way of achieving this goal is to use the following contribution function. Let D be a set of training documents and W be a weight vector. Let C be a set of classes of D and $class(x)$ be a function that returns the class of document x. For a training sample $d \in D$, let $N_d = \{n_1, n_2, \ldots, n_k\}$ be the set of k-nearest neighbors of d. Given N_d, the similarity sum of d's neighbors that belong to class c is:

$$S_c = \sum_{n_i \in N_d, class(n_i)=c} cos(d, n_i, W) \tag{1}$$

Then the total similarity sum of d is:

$$T = \sum_{c \in C} S_c$$

The contribution of d, $Contribution(d)$, is defined in terms of S_c of classes $c \in C$ and T:

$$Contribution(d) = \begin{cases} 1 \text{ if } \forall c \in C, c \neq class(d),\, S_{class(d)} > S_c \text{ and } \frac{S_{class(d)}}{T} \geq p \\ 0 \text{ otherwise} \end{cases}$$

In this contribution function, a document d contributes to the overall optimization function only if the sum of similarities of d's neighbors with class $class(d)$ is greater than that of any other class and is at least fraction p of the total similarity sum. Finally, the optimization function is defined as:

$$OPT(D, W) = \frac{\sum_{d \in D} Contribution(d)}{|D|}$$

Different $Contribution(d)$ functions describe different optimization functions. We have experimented with several contribution functions [7] and chosen the above contribution function for WAKNN. The optimization function based on this contribution function allows the weight adjustment to be flexible in terms of local minima and overfitting. If the optimization function uses a low p (close to 0.0), then the weight adjustment will tend to overfit the training documents but will move out of possible local minima more easily. On the other hand, if the optimization function uses high p (close to 1.0), then the weight adjustment will tend to avoid overfitting but have tendency to be trapped in a local minima.

Several space search techniques are available including hill-climbing (greedy) search [11], best-first search [11], beam search [11], or bidirectional search [11]. WAKNN searches for the best weight vector using hill-climbing (greedy) method. We chose a hill-climbing (greedy) search strategy, because this approach is computationally fast and memory efficient. In the greedy search of WAKNN, the weight for each word is perturbed slightly and its effect on the optimization function is evaluated. The perturbation of the weight is proposed by multiplying the current weight by different factors. The weight perturbation with the most improvement in the optimization function is chosen as the winner and its weight is updated. This process is repeated until no further progress is obtained.

3.2 Performance Improvements of WAKNN

The major drawback of WAKNN is its high computational cost. After each possible weight adjustment of a word in the greedy search, the merit of the possible weight adjustment is evaluated using the optimization function $OPT(D, W)$. The cost of this evaluation is $O(n^2)$ where n is the number of training documents, because finding the k nearest neighbors of each training document requires comparison to the rest of the training documents. Hence the computational cost of each iteration of the WAKNN algorithm is $O(mn^2)$ where m is the number of words in the document data sets. We propose two computational performance improvements of WAKNN. The first method (WAKNN-F) improves the performance by reducing the number of words used for the weight adjustment, i.e, reducing m in $O(mn^2)$. The second method (WAKNN-C) improves the performance by reducing the cost of evaluating $OPT(D, W)$, i.e., reducing n in $O(mn^2)$.

WAKNN-F In order to select words for weight adjustment, we need a ranking method and a selection method. We use mutual information [22, 16] to rank words. The mutual information can be computed in $O(nl)$ where n is the number of training documents and l is the average length of the document. In the single scan of training documents, we can compute class distributions of each word. Based on this information, the mutual information of each word can be computed. The preliminary experiments show that only 10 to 70 weights are changed under WAKNN and most of these words ranked in the top 1000 according to the mutual information of words. These results show that the ranking method based on mutual information could be effective in finding words that are important in weight adjustment. We also consider pairwise word dependency in the ranking by computing mutual information of the pair of words. The mutual information of a word is then determined as the maximum of the mutual information of the single word and any pair of words with this word in it.

There are three desired goals of the selection method. We want to include as many top ranked words as possible. In the same time, we want to make sure that the selected words cover as many documents as possible. It is possible that a small set of documents contain many words that have high mutual information. If we select words based purely on the mutual information, we will not be able to improve the classification accuracies of documents not in this set. Finally, we want to select as small number of words as necessary to control the computational cost.

We propose the following scheme to achieve these three goals. The overall steps of the scheme is shown in Figure 3. We rank words globally according to the mutual information. For each document, we sort its words according to the global ranking. We have chosen an incremental approach to determine the right level of coverage of documents. We start collecting top 10% (MinCov = 0.1) of the words according to the global ranking in each document. The union of all these words constitute the selected words. By doing this, we guarantee that words selected cover at least 10% of the words in each document. The k-NN classification accuracy of training documents using only these selected words is calculated. This accuracy is compared to the baseline classification accuracy that is computed by using all the words. If this ratio is less than a user specified minimum ratio (MinRatio), the selection process is repeated with the minimum coverage (MinCov) incremented by 0.05 (5%). The selection mechanism stops as soon as the classification power of the selected words is at least MinRatio of the classification power of all the words. The MinRatio controls the total number of words selected.

```
SelectWord(D, MinRatio)
  1. Rank words (W) in training document D according to mutual information.
  2. For each d ∈ D, sort words in d according to the global word ranking.
  3. BaseAcc = Accuracy(D, W)
  4. AccRatio = 0.0; MinCov = 0.1; Selected = {};
  5. While (AccRatio < MinRatio)
       5.1 for each d ∈ D
              Selected = Selected ∪ {top (MinCov * 100.0) % of words in d}
       5.2 Acc = Accuracy(D, Selected)
       5.3 AccRatio = Acc / BaseAcc
       5.4 MinCov = MinCov + 0.05
  6. return Selected
```

Fig. 3. Major steps of word selection scheme.

The proposed scheme guarantees that distinguishing or important words are selected, all the documents have minimum coverage, and selected words have classification capacity of at least MinRatio of the full set of words. However, when the classification using all the words is very poor, this scheme will select very few words and stop. We fixed this problem by selecting at least 500 words. We call this optimized version of WAKNN as WAKNN-F.

WAKNN-C One obvious choice for reducing the computational cost of $O(n^2)$ part of the overall computational cost is by sampling training data. However, sampling in the presence of large variations of class size is troublesome. Furthermore, this might mitigate the advantage of k-NN classification. k-NN performs well when there is multi-modal class distribution in which summarization of the whole class does not accurately depict the class distribution. In this multi-modal distribution, k-NN works well because each data point can find a small set of neighbors that are in the same class and are very close. Random sampling could thin out these neighbors such that each data point can be pulled toward other classes in k-NN classification.

We have chosen to cluster training documents. We cluster similar documents within each class and represent each cluster by a centroid. Then in the optimization function, we find k-nearest centroids instead of k-nearest neighbors. If we have c clusters, then we will have computational complexity of $O(cn)$ instead of $O(n^2)$. Assuming that we can find $c \ll n$, the computational cost would decrease dramatically.

In the top down clustering of documents within each class, we start with each class being a single cluster. We select the next cluster to divide based on the recall of each cluster. The recall of each cluster is defined as follows. For each cluster, we first find training samples (regardless of their class labels) that have the centroid of this cluster as the closest centroid. The recall for the cluster is defined as the percentage of these training samples that have the same class as the cluster.

We pick the cluster with the worst recall and divide this cluster into two clusters, and then perform the refinement similar to k-means clustering algorithm [9] with the weighted cosine similarity measure for assigning documents to clusters. Note that refinement of clusters is performed among clusters of the same class. This approach gives the natural stopping point: we stop clustering when all the clusters have perfect recalls. We augment this stopping criterion by forcing each class to have at least k clusters.

As the weight adjustment progresses, the weight changes can cause the cluster to change. After each weight change, we refine the clusters with new weights and repeat the recall-based clustering process to enforce that each cluster has a perfect recall. In general, the number of clusters increases as the weight adjustment progresses. It is entirely possible that better weights can allow clusters to be merged and yet have a perfect recall. In the current implementation this possibility has not been explored. We call this optimization of WAKNN-F as WAKNN-C.

The major steps of WAKNN-C are shown in Figure 4. In steps 1, 2 and 3, training document matrix is constructed and the matrix is normalized to mitigate the effect of different document lengths. In step 8.3, the weight for each word is perturbed slightly and its effect on the optimization function is evaluated. The perturbation of the weight is proposed by multiplying the current weight by different factors. The weight perturbation with the most improvement in the optimization function is chosen as the winner and its weight is updated. This process is repeated until no further progress is obtained.

```
1. Construct a training matrix D.
      - each row corresponds to a training document
      - each column corresponds to a word
      - value in the matrix D(i, j) corresponds frequency of word j in document i.
2. Let F be a multiplication factors and W be a weight vector.
3. Normalize word frequencies in each document such that they add up to 1.0.
4. For each j, W_j = 1.0.
5. Selected = SelectWord(D)
6. C = FindClusters(D, W)
7. bestopt = OPT(D, W, C); oldopt = 0;
8. While (bestopt > oldopt)
      8.1 oldopt = bestopt
      8.2 bestword = -1; newval = -1
      8.3 For each word j ∈ Selected
             For each f ∈ F
                W' = W
                W'_j = W_j × f
                if (OPT(D, W', C) > bestopt)
                       bestword = j
                       bestval = W'_j
                       bestopt = OPT(D, W', C)
      8.4 W_bestword = bestval
      8.5 RefineClusters(D, C, W)
```

Fig. 4. Major steps of WAKNN-C.

The differences between WAKNN-F and WAKNN are steps 5 and 8.3. Instead of checking all the words for weight changes, WAKNN-F only checks the words selected from the SelectWord function. The difference between WAKNN-C and WAKNN-F are step 6 in the algorithm for computing initial clusters and step 8.5 for refining clusters after a weight change. Another difference is the definition of neighbors in the computation of similarity sum in Eq 1. Now the N_d' is the set of k-nearest clusters of d. k-nearest clusters of d is determined by computing similarity to the centroids of clusters. S_c is computed based on the similarities between d and the centroids of k-nearest clusters.

4 Experimental Results

In all of the data sets, we have used stop words to remove common words and stemmed words using Porter's suffix-stripping algorithm [17]. For the data representation, we have followed the vector space model commonly used in Information Retrieval systems [19].

We have used 52 total data sets for experiments. More detailed information on these data sets is available in [7] [1]. Data sets *west1, west2, ..., west7* are from the statutory collections of the legal document publishing division of West Group described in [4]. Data sets *tr11, tr12, ..., tr45, fbis, trec6*, are derived from TREC-5 [21], TREC-6 [21], and TREC-7 [21] collections. The classes of the various TREC data sets were generated from the relevance judgment provided in these collections. Data sets *oh0, oh1, ..., oh19* are from OHSUMED collection [8] subset of MEDLINE database. We took different subsets of categories to construct these data sets. Data sets *re0* and *re1* are from Reuters-21578 text categorization test collection Distribution 1.0 [14]. 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 set *wap* is from the WebACE project (WAP) [1]. Each document corresponds to a web page listed in the subject hierarchy of Yahoo!.

We compare WAKNN against C4.5, RIPPER, Rainbow, k-NN, PEBLS, k-NN with mutual information (MI), RELIEF-F, and VSM. We have not included SVM in the comparison, because the SVM code [10] available was designed for two class problem. We implemented k-NN, MI, and RELIEF-F; we used publicly available implementations of C4.5, RIPPER, Rainbow, PEBLS, and VSM. In all the k-NN based algorithms (WAKNN-C, kNN, MI, RELIEF-F, and VSM) except PEBLS, the number of neighbors $k = 10$ is used. For PEBLS, we used the nearest neighbor (i.e. $k = 1$), because the results using more than one neighbor were significantly worse. For Rainbow, we used the multinomial event model as the document representation model, because results from other works [16, 24] indicate that this event model is better than the multi-variate Bernoulli event model and our preliminary experiments also confirmed this claim. For other parameters in these algorithms, we used default parameter values suggested by the developers of the algorithms.

Table 1 shows the classification accuracy of WAKNN compared to other classifiers in a subset of documents described above. In these experiments, we used $p = 0.5$ in the optimization function and $\{0.2, 0.5, 0.8, 1.5, 2.0, 4.0, 9.0, 15.0, 30.0, 50.0\}$ as multiplication factors for weight perturbation. Out of 19 data sets, WAKNN has the best classification accuracy on 13 data sets. None of the other classifiers has the best classification accuracy on more than one data set. Even for 6 data sets that WAKNN did worse than some other classification algorithms, the classification accuracy of WAKNN was within less than 2% of the best results.

Even though WAKNN provides better classification accuracies over other classifiers, the computational cost of WAKNN is very high. Table 2 shows the run time comparison of WAKNN, WAKNN-F, WAKNN-C, and VSM. The runtime of WAKNN varies from a few hours on 300 to 500 training sample size (west1 and tr11) to a few days on 1000 training sample size (oh8, oh12, and oh18). WAKNN-F and WAKNN-C significantly reduced the runtime of WAKNN. For instance, WAKNN-F reduced the runtime of WAKNN by a factor of 2 to 6.

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

	C4.5	RIPPER	Rainbow	kNN	PEBS	MI	RELIEF-F	VSM	WAKNN
west1	82.40	84.87	84.40	76.73	78.50	86.80	76.87	85.27	**89.40**
west2	74.20	72.22	72.11	68.33	67.80	75.89	68.44	76.11	**78.11**
west3	75.40	78.48	79.92	67.83	67.00	79.71	68.03	79.92	**88.32**
west4	80.50	73.88	79.61	67.26	70.70	74.96	67.80	75.49	**84.26**
west5	84.20	80.35	88.73	77.78	83.40	90.02	78.10	87.12	**92.91**
west6	76.60	78.14	85.66	72.27	74.30	82.92	72.40	85.11	**86.89**
west7	79.90	76.05	74.35	67.46	68.40	78.08	67.68	78.64	**81.58**
tr11	79.70	79.23	85.99	85.02	81.20	85.51	85.02	83.58	**86.96**
tr12	86.00	82.17	80.25	82.17	79.60	86.62	82.17	**87.90**	87.26
tr21	80.50	82.25	60.95	81.66	75.70	80.47	81.66	80.47	**88.76**
tr22	88.60	87.40	91.06	93.09	84.60	**93.50**	93.09	91.06	93.09
tr23	**93.20**	93.20	73.79	82.52	79.60	89.32	82.52	76.70	91.26
tr24	92.50	85.63	73.12	88.75	90.00	86.25	88.75	86.25	**93.75**
tr25	83.40	**91.72**	85.80	80.47	72.80	81.07	80.47	84.02	89.94
tr33	89.50	90.83	67.69	86.46	86.50	90.39	86.46	89.52	**94.76**
tr43	95.30	89.67	88.73	83.57	83.60	84.04	83.57	88.73	**95.77**
oh8	76.90	71.60	85.44	82.40	55.00	80.95	82.52	79.85	**86.77**
oh12	82.80	81.24	86.63	85.17	73.90	80.45	85.28	81.46	**88.54**
oh18	78.60	77.63	**82.44**	78.30	56.60	76.17	78.41	77.63	82.21

Table 1. Classification accuracies of different classifiers. Note that the highest accuracy for each data set is highlighted with bold font.

WAKNN-C reduced runtime dramatically in large data sets (oh8, oh12, and oh18) compared to WAKNN-F. While WAKNN-F and WAKNN-C reduced the runtime of WAKNN significantly, WAKNN-F and WAKNN-C have statistically equal quality classification according to the sign test [20, 7] and two sample significance test [20, 7]. This table also shows that WAKNN-C has a much better runtime than VSM for these data sets.

	WAKNN	WAKNN-F	WAKNN-C	VSM
west1	2 hr 20 min	1 hr 2 min	25 min	1 hr 58 min
west2	15 min	9 min	9 min	1 hr 22 min
west3	10 min	6 min	6 min	1 hr 9 min
west4	13 min	12 min	9 min	1 hr 51 min
west5	15 min	11 min	8 min	2 hr 4 min
west6	28 min	20 min	12 min	2 hr 30 min
west7	59 min	44 min	23 min	2 hr 21 min
tr11	4 hr 39 min	49 min	17 min	6 hr 46 min
tr12	1 hr 18 min	18 min	6 min	2 hr 10 min
tr21	1 hr 28 min	13 min	13 min	1 hr 4 min
tr22	7 hr 33 min	45 min	10 min	3 hr 31 min
tr23	20 min	4 min	4 min	34 min
tr24	1 hr 4 min	7 min	7 min	1 hr 48 min
tr25	1 hr 13 min	13 min	5 min	39 min
tr33	5 hr 48 min	37 min	15 min	8 hr 52 min
tr43	3 hr 19 min	34 min	15 min	4 hr 10 min
oh8	25 hr 57 min	5 hr 8 min	23 min	6 hr 21 min
oh12	35 hr 34 min	4 hr 57 min	19 min	6 hr 30 min
oh18	42 hr 5 min	7 hr 58 min	25 min	7 hr 11 min

Table 2. Runtime comparison of WAKNN, WAKNN-F, WAKNN-C, and VSM for selected data sets. The run times were measured on IBM Power 2 based workstations.

We now present the classification accuracy of WAKNN-C compared to other classifiers on all 52 data sets in Table 3. We used MinRatio=0.75 for selecting words in WAKNN-C. The number of words selected ranged from one half to one tenth of the original number of words in the data sets. The minimum coverage of each document with selected words ranged from 10% to 85%. The number of clusters found ranged from one half to one tenth of the original number of training documents. Detailed parameter studies of WAKNN-C are available in [7]. Out of 52 data sets, WAKNN-C has the best classification accuracy on 38 data sets. None of the other classifiers has the best classification accuracy on more than three data sets. These results show that WAKNN-C consistently outperforms other classifiers in most of the data sets. In order to verify whether these differences are statistically significant, we performed the sign test [20, 7] and the two sample significance test [20, 7]. The two sample significance test shows that WAKNN-C is not statistically worse than any of the classifiers on any data set. The results also show that WAKNN-C is statistically better (P-value ≤ 0.05) than C4.5, RIPPER, Rainbow, k-NN, PEBLS, MI, RELIEF-F, and VSM in 43, 39, 22, 34, 49, 41, 34 and 35 data sets respectively. According to the sign test, WAKNN-C is statistically better (with P-value ≤ 0.05) than all the other classifiers [2].

5 Conclusions and Directions for Future Research

In this paper, we presented a Weight Adjusted k-Nearest Neighbor (WAKNN) classification that retains the power of the k-NN while further enhancing its ability by learning feature weights. Experimental results show that WAKNN-C outperforms existing state of the art classification algorithms quite consistently on the document data sets from a variety of domains. Even though the focus of this paper has been the application of WAKNN in the text categorization task, WAKNN is applicable in any domain with large number of attributes.

Several issues that we identified in the course of this work deserve further attention. In the greedy search technique used in WAKNN, only one weight change is selected at a time. It will be rewarding to explore methods for changing multiple words at a time in the greedy search process. One possibility is to use association rules to identify set of words that are strongly related, and then use this knowledge in determining which sets of words to select for simultaneous weight adjustments. Another research focus is the incremental weight adjustment as new training samples are available.

References

1. 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*, 13(5-6), 1999.
2. W.W. Cohen. Fast effective rule induction. In *Proc. of the Twelfth International Conference on Machine Learning*, 1995.

[2] We also performed a set of experiments in which we kept only the top few hundred discriminating words according to mutual information or the same words that WAKNN-C chose for weight adjustment. According to the sign test and two sample significance test, WAKNN-C consistently outperformed other classifiers on these data sets as well [7].

	C4.5	RIPPER	Rainbow	kNN	PEBS	MI	RELIEF-F	VSM	WAKNN-C
west1	82.40	84.87	84.40	76.73	78.50	86.80	76.87	85.27	**87.40**
west2	74.20	72.22	72.11	68.33	67.80	75.89	68.44	76.11	**81.78**
west3	75.40	78.48	79.92	67.83	67.00	79.71	68.03	79.92	**85.86**
west4	80.50	73.88	79.61	67.26	70.70	74.96	67.80	75.49	**83.54**
west5	84.20	80.35	88.73	77.78	83.40	90.02	78.10	87.12	**94.04**
west6	76.60	78.14	85.66	72.27	74.30	82.92	72.40	85.11	**86.89**
west7	79.90	76.05	74.35	67.46	68.40	78.08	67.68	78.64	**81.69**
tr11	79.70	79.23	**85.99**	85.02	81.20	85.51	85.02	83.58	85.99
tr12	86.00	82.17	80.25	82.17	79.60	86.62	82.17	**87.90**	87.90
tr13	88.90	88.36	87.69	**93.10**	87.70	92.02	93.10	92.56	92.69
tr14	88.90	90.40	91.64	92.88	79.30	93.19	92.88	90.71	**95.05**
tr15	94.90	93.57	97.75	94.86	94.20	97.43	94.86	**98.07**	97.43
tr21	80.50	82.25	60.95	81.66	75.70	80.47	81.66	80.47	**90.53**
tr22	88.60	87.40	91.06	93.09	84.60	**93.50**	93.09	91.06	92.28
tr23	**93.20**	93.20	73.79	82.52	79.60	89.32	82.52	76.70	85.44
tr24	92.50	85.63	73.12	88.75	90.00	86.25	88.75	86.25	**96.25**
tr25	83.40	**91.72**	85.80	80.47	72.80	81.07	80.47	84.02	88.76
tr31	90.50	87.93	92.46	91.59	86.90	**95.69**	91.59	92.46	94.61
tr32	86.80	83.72	78.68	75.97	71.70	75.58	75.97	80.62	**90.31**
tr33	89.50	90.83	67.69	86.46	86.50	90.39	86.46	89.52	**95.63**
tr34	**94.00**	87.99	77.74	88.34	85.20	85.51	88.34	86.57	93.99
tr35	87.90	90.00	87.50	88.93	78.90	91.79	88.93	90.00	**96.07**
tr41	90.70	**95.67**	94.08	89.07	86.30	89.07	89.07	92.71	94.99
tr42	89.70	92.31	87.91	85.35	79.50	87.55	85.35	91.58	**94.87**
tr43	**95.30**	89.67	88.73	83.57	83.60	84.04	83.57	88.73	92.96
tr44	86.60	79.77	**89.31**	83.97	80.50	82.06	83.97	80.92	85.88
tr45	87.90	83.53	84.68	86.99	70.80	85.55	87.28	88.73	**96.53**
fbis	57.10	74.84	76.38	78.49	69.80	76.54	78.49	74.67	**83.52**
trec6	67.50	82.79	92.16	91.99	84.30	88.42	91.99	87.56	**94.21**
oh0	84.70	84.06	90.21	87.08	55.60	85.73	86.98	77.19	**90.73**
oh1	81.20	79.82	84.37	86.03	37.50	81.93	86.03	82.48	**87.36**
oh2	88.90	85.31	91.57	90.57	76.10	87.49	90.48	86.40	**92.38**
oh3	85.60	81.58	88.69	86.03	55.10	87.40	86.03	84.83	**91.35**
oh4	82.00	86.33	92.07	90.52	58.20	86.87	90.43	82.04	**93.53**
oh5	76.70	72.73	83.83	84.06	53.20	81.94	84.06	80.87	**88.78**
oh6	83.90	84.43	88.58	**90.71**	53.30	82.64	90.59	81.86	89.92
oh7	77.90	76.33	87.46	85.80	34.40	81.30	85.56	73.49	**87.81**
oh8	76.90	71.60	85.44	82.40	55.00	80.95	82.52	79.85	**87.99**
oh9	84.50	76.90	90.48	88.81	56.10	83.10	88.81	83.33	**91.19**
oh10	74.00	70.36	79.69	77.26	39.90	76.09	77.26	69.00	**84.55**
oh11	83.30	84.32	82.81	80.93	65.80	85.74	80.93	82.91	**91.31**
oh12	82.80	81.24	86.63	85.17	73.90	80.45	85.28	81.46	**88.88**
oh13	80.00	80.55	91.97	89.64	63.50	86.36	89.75	86.05	**92.92**
oh14	82.10	79.79	88.51	86.88	43.50	86.88	86.97	83.53	**90.13**
oh15	73.00	76.67	82.81	80.25	49.80	75.11	80.13	77.12	**84.26**
oh16	87.70	86.63	92.15	90.60	74.80	90.34	90.68	87.49	**93.70**
oh17	85.20	80.16	90.72	89.04	56.30	86.77	89.04	85.59	**92.20**
oh18	78.60	77.63	82.44	78.30	56.60	76.17	78.41	77.63	**86.02**
oh19	80.60	77.49	86.81	85.70	42.90	80.82	85.70	81.49	**87.25**
re0	66.90	67.27	76.45	81.04	67.90	79.44	**81.44**	81.04	78.44
re1	78.60	77.27	77.73	74.35	67.00	76.19	74.35	80.49	**84.49**
wap	71.30	72.95	83.97	81.92	41.80	79.23	81.92	68.08	**84.23**

Table 3. Classification accuracies of different classifiers. Note that the highest accuracy for each data set is highlighted with bold font.

3. S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1):57–78, 1993.
4. T. Curran and P. Thompson. Automatic categorization of statute documents. In *Proc. of the 8th ASIS SIG/CR Classification Research Workshop*, Tucson, Arizona, 1997.
5. I.S. Dhillon and D.M. Modha. Visualizing class structure of multi-dimensional data. In *Proc. of the 30th Symposium of the Interface: Computing Science and Statistics*, pages 488–493, 1998.
6. R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.
7. E.H. Han. *Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification*. PhD thesis, University of Minnesota, October 1999.
8. 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.
9. A.K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.
10. T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of the European Conference on Machine Learning*, 1998.
11. L. N. Kanal and Vipin Kumar, editors. *Search in Artificial Intelligence.* Springer-Verlag, New York, NY, 1988.
12. I. Kononenko. Estimating attributes: Analysis and extensions of relief. In *Proc. of the 1994 European Conference on Machine Learning*, 1994.
13. 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.
14. D. D. Lewis. Reuters-21578 text categorization test collection distribution 1.0. *http://www.research.att.com/ lewis*, 1999.
15. D.G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, pages 72–85, January 1995.
16. 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.
17. M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
18. J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.
19. G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.
20. G.W. Snedecor and W.G. Cochran. *Statistical Methods.* Iowa State University Press, 1989.
21. TREC. Text REtrieval conference.
22. 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.
23. Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *SIGIR-94*, 1994.
24. Y. Yang and X. Liu. A re-examination of text categorization methods. In *SIGIR-99*, 1999.