Using the Feature Projection Technique based on a Normalized Voting Method for Text Classification

Youngjoong Ko and Jungyun Seo

Department of Computer Science
Sogang University
Sinsu-dong 1, Mapo-gu
Seoul, 121-742, Korea

kyj@nlpzodiac.sogang.ac.kr, seojy@ccs.sogang.ac.kr

1. **Corresponding author**: Youngjoong Ko

2. **Corresponding address**: NLP lab., Department of Computer Science, Sogang University, Sinsu-dong 1, Mapo-gu, Seoul, Korea, 121-742

3. **Corresponding telephone number**: 82-2-706-8954

4. **Corresponding fax number**: 82-2-704-8273

5. **Corresponding E-mail address**: kyj@nlpzodiac.sogang.ac.kr
Abstract

This paper proposes a new approach for text categorization, based on a feature projection technique. In our approach, training data are represented as the projections of training documents on each feature. The voting for a classification is processed on the basis of individual feature projections. The final classification of test documents is determined by a majority voting from the individual classifications of each feature. Our empirical results show that the proposed approach, *Text Categorization using Feature Projections* (TCFP), outperforms $k$-NN, Rocchio, and Naive Bayes. Most of all, TCFP is a faster classifier, up to one hundred times faster than $k$-NN in the Newsgroups data set. It is also robust from noisy data. Since the TCFP algorithm is very simple, its implementation and training process can be done very easily. For these reasons, TCFP can be a useful classifier in text categorization tasks, which need fast execution speed, robustness, and high-performance.

**Keywords**: Text Categorization, Text Classifier, Feature Projections, Instance-based Learning, $k$-NN
1. Introduction

An issue of text categorization is to classify documents into a certain number of pre-defined categories. Text categorization is an active research area in information retrieval and machine learning. A wide range of supervised learning algorithms has been applied to this issue using a training data set of categorized documents. The Nearest Neighbor (Yang et al., 2002), Rocchio (Lewis et al., 1996), Naive Bayes (McCalum et al., 1998), and Support Vector Machines (Joachims, 1998) are well-known algorithms.

Among these learning algorithms, we focus on the Nearest Neighbor algorithm. In particular, the \(k\)-Nearest Neighbor (\(k\)-NN) classifier in text categorization is one of the state-of-the-art methods including the Support Vector Machines (SVMs) and Boosting algorithms. Since the Nearest Neighbor algorithm is much simpler than other algorithms, the \(k\)-NN classifier is intuitive and easy to understand, and it learns quickly. But one of main weak points of \(k\)-NN is that its running time is much too slow, because its main computation is the on-line scoring of all training documents to find the \(k\) nearest neighbors of a test document. In order to reduce this scaling problem in on-line ranking, a number of techniques have been studied in the literature. Techniques such as the instance pruning technique (Wilson et al., 1999) and feature projections (Akkus et al., 1996) are well known.

The instance pruning technique is one of the most straightforward ways to accelerate classification speed in a nearest neighbor system. It reduces time and storage requirements by removing instances from the training set. So far a large number of such pruning techniques have been proposed. To name a few, there are the Condensed Nearest Neighbor Rule (Hart, 1968), IB2 and IB3 (Aha et al., 1991), the Typical Instance Based Learning (Zhang, 1992), and the Reduction Techniques (RT1-RT3) (Wilson et al, 1997). These pruning techniques and others were surveyed in depth by Wilson et al. (1999). They then developed several new pruning techniques such as DROP1-DROP5. Of these, DROP4 showed the best performance.

Another trial to overcome the problem exists in feature projections. Akkus and Guvenir presented a new approach to classification based on feature projections (Akkus et al., 1996). They called their resulting algorithm \(k\)-Nearest Neighbor on Feature Projections (\(k\)-NNFP). In this approach, the classification
knowledge is represented as the sets of projections of training data on each feature dimension. The classification of a test instance is based on the voting by the $k$ nearest neighbors of each feature of the test instance. The resulting system allowed the classification to be much faster than that of $k$-NN, and its performance were comparable with $k$-NN.

In this paper, we present a particular implementation of text categorization using feature projections. When we applied the feature projection technique to text categorization, we found several problems caused by the special properties of text categorization. We describe these problems in detail and then we propose a new approach to solve them. The proposed system shows better performance than $k$-NN, and it is much faster than $k$-NN. Furthermore, it has the additional advantage of robustness from noisy data.

The rest of this paper is organized as follows. Section 2 simply presents $k$-NN, the DROP4 pruning algorithm, and the $k$-NNFP algorithm. Section 3 explains a new approach using feature projections. Section 4 describes other classifiers used in our experiments. In section 5, we discuss empirical results in our experiments and an analysis for strong points of the new proposed classifier. The final section presents conclusions.

2. $k$-NN, the DROP4 Pruning Algorithm, and the $k$-NNFP Algorithm

In this section, we simply describe $k$-NN, the DROP 4 pruning algorithm, and the $k$-NNFP algorithm.

2.1 The $k$-Nearest Neighbor ($k$-NN) Algorithm

As an instance-based classification method, $k$-NN has been known as an effective approach to a broad range of pattern recognition and text classification problems (Duda et al., 2001; Yang, 1994). In the $k$-NN algorithm, a new input instance should belong to the same class as its $k$ nearest neighbors in the training
data set. After all the training data is stored in memory, a new input instance is classified with the class of k nearest neighbors among all stored training instances.

For the distance measure and the document representation, we use a conventional vector space model; each document is represented as a vector of term weights and similarity between two documents is measured by the cosine value of the angle between the corresponding vectors (Yang et al., 2002).

Let a document \(d\) with \(n\) terms (\(t\)) be represented as the feature vector:

\[
\vec{d} = \langle w(t_1, d), w(t_2, d), \ldots, w(t_n, d) \rangle
\]

We compute the weight vectors for each document using one of conventional TF-IDF schemes (Salton et al., 1988). The weight of term \(t\) in document \(d\) is calculated as follows:

\[
w(t, \vec{d}) = \frac{1 + \log (w(t, d)) \times \log \left(\frac{N}{n_t}\right)}{\|\vec{d}\|} \quad \text{(2)}
\]

where

i) \(w(t, \vec{d})\) is the weight of term \(t\) in document \(\vec{d}\)

ii) \(w(t, \vec{d})\) is the within-document Term Frequency (TF)

iii) \(\log \left(\frac{N}{n_t}\right)\) is the Inverted Document Frequency (IDF)

iv) \(N\) is the number of documents in the training set

v) \(n_t\) is the number of training documents in which \(t\) occurs

vi) \(\|\vec{d}\|\) is the 2-norm of vector \(\vec{d}\)

Given an arbitrary test document \(d\), the \(k\)-NN classifier assigns a relevance score to each candidate category \(c_j\) using the following formula:
where \( r_{i,(d)} \) denotes a set of the \( k \) nearest neighbors of document \( d \) and \( D_j \) is a set of training documents in class \( c_j \).

2.2 The DROP4 Pruning Algorithm

This section presents an instance pruning algorithm called the Decremental Reduction Optimization Procedure 4 (DROP4) (Wilson et al., 1999). The procedure of this algorithm is decremental, meaning that it begins with the entire training set and then removes instances that seem to be unnecessary.

Before the DROP4 algorithm is described, some notation is introduced here. A training set \( T \) consists of \( n \) instances (\( i \)). Each instance \( i \) has \( k \) nearest neighbors. Each instance \( i \) also has a nearest \textit{enemy} which is the nearest instance \( e \) with a different category from \( i \). Those instances, which have \( i \) as one of their \( k \) nearest neighbors, are called associates of \( i \).

\( DROP4 \) uses the following basic rule to decide if it is safe to remove an instance \( i \) from the instance set \( S \) (where \( S=T \) originally).

\[
\text{Remove instance } i \text{ from } S \text{ if at least as many of its associates in } T \\
\text{would be classified correctly without } i.
\]

The order of removal can be important to the success of a pruning algorithm. \( DROP4 \) initially sorts instances in \( S \) by the distance to their nearest \textit{enemy}. And then the instances are checked for removal by beginning at the instance which is farthest from its nearest enemy. This tends to remove instances farthest from the decision boundary first, which increases the chance of retaining border points. However, noisy instances are also border points, so it is desirable to remove the noisy instances before any of the others so
that the rest of the algorithm is not influenced heavily by the noisy instances. Therefore, DROP4 uses a noise-filtering pass before sorting the instances in $S$. For the noise-filtering pass, DROP4 removes each instance only if it is (1) misclassified by its $k$ nearest neighbors and (2) it does not hurt the classification of its associates.

2.3 The $k$-Nearest Neighbor on Feature Projection ($k$-NNFP) Algorithm

The $k$-NNFP is a variant of the $k$-NN method. The main difference is that instances are projected on their features in the $n$-dimensional space (see figure 1) and distance between two instances is calculated according to a single feature. The distance between two instances $d_i$ and $d_j$ with regard to $m$-th feature $t_m$ is $\text{dist}_m(t_m(i), t_m(j))$ as follows:

$$
\text{dist}_m(t_m(i), t_m(j)) = \left| w(t_m, \vec{d}_i) - w(t_m, \vec{d}_j) \right|
$$

where $t_m(i)$ denotes the $m$-th feature $t$ in an instance $\vec{d}_i$ and $w(t_m, \vec{d}_j)$ is the weight of term $t_m$ in document $\vec{d}_j$.

The classification on a feature is done according to votes of the $k$-nearest neighbors of that feature in a test instance; the category of a nearest neighbor for the vote follows that of document including the nearest neighbor. The final classification of the test instance is determined by a majority voting from individual classifications of each feature. If there are $n$ features, this method returns $n \times k$ votes whereas the $k$-NN method returns $k$ votes.
3. A New Approach of Text Categorization on Feature Projections

First of all, we show an example of feature projections in text categorization for more easy understanding. We then enumerate the problems to be considered when the feature projection technique is applied to text categorization. Finally, we propose a new approach using feature projections to overcome these problems.

3.1 An Example of Feature Projections in Text Categorization

We give a simple example of the feature projections in text categorization. To simplify our description, we suppose that all documents have just two features ($f_1$ and $f_2$) and two categories ($c_1$ and $c_2$). The TF-IDF value by formula (2) is used as the weight of a feature. Each document is normalized as a unit vector and each category has three instances: $c_1 = \{d_1, d_2, d_3\}$ and $c_2 = \{d_4, d_5, d_6\}$. Figure 1 shows how document vectors in a conventional vector space are transformed into feature projections and stored on each feature dimension. The result of feature projections on a term (or feature) can be seen as a set of weights of documents for the term. On feature projections, the category of each element for a feature is set to the category of documents including the feature. Since a term with 0.0 weight is useless, the size of the set equals the Document Frequency (DF) value of the term.

[Figure 1. Feature representation on feature projections in Text Categorization]

3.2 Problems in Applying Feature Projections to Text Categorization

There are three problems: (1) the diversity of Document Frequency (DF) values of terms, (2) the property of using TF-IDF values as the weight of features, and (3) the lack of contextual information.
3.2.1 The diversity of Document Frequency values of terms

Table 1 shows a distribution of the DF values of the terms in the Newsgroups data set. The numerical values of Table 1 are calculated from a training data set with 16,000 documents and 10,000 features chosen by feature selection. The $k$ in the fourth column means the number of nearest neighbors selected in $k$-NNFP; the $k$ in $k$-NNFP was set to 20 in our experiments.

Table 1. A distribution of the DF values of the terms in the Newsgroups data set

<table>
<thead>
<tr>
<th>Average DF</th>
<th>Maximum DF</th>
<th>Minimum DF</th>
<th>The number of features DF &lt; k (20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>54.59</td>
<td>8,407</td>
<td>4</td>
<td>6,489</td>
</tr>
</tbody>
</table>

According to Table 1, more than a half of the features have DF values less than $k$ (20). This result can be also explained by Zipf’s law. The problem is that some features have DF values less than $k$ while other features have DF values much greater than $k$. For features with the DF value less than $k$, all the elements of the feature projections on the feature could and should participate in voting. In this case, the number of elements chosen for voting is less than $k$. On the other hand, for features with the DF value more than $k$, only maximum $k$ elements among the elements of the feature projections should be chosen for voting. Therefore, we need to normalize the voting ratio for each feature. As shown in formula (5), we use a proportional voting method to normalize the voting ratio.

3.2.2 The property of using TF-IDF values as the weight of features

The TF-IDF value of a term is its presumed value for identifying the content of a document (Salton et al., 1983). Therefore, on feature projections, elements with a high TF-IDF value for a feature must become more useful classification criterions for the feature than any elements with low TF-IDF values. In order to apply this property to our algorithm, we use only elements with TF-IDF values above the average TF-IDF value for voting. The selected elements also participate in proportional voting with the same importance as the TF-IDF value of each element. The voting ratio of each category $c_j$ in a feature $t_m$ of a test document
\( \vec{d} \) is calculated by the following formula:

\[
r(c_i, t_n) = \frac{\sum_{t_n \in I_m} w(t_n, \vec{d}) \cdot y(c_i, t_n(l))}{\sum_{t_n \in I_m} w(t_n, \vec{d})}
\]

(5)

In the above formula, \( I_m \) denotes a set of elements selected for voting and \( y(c_i, t_n(l)) \in [0,1] \) is a function; if the category for an element \( t_n(l) \) is equal to \( c_i \), the output value is 1. Otherwise, the output value is 0.

### 3.2.3 The lack of contextual information

Since each feature votes separately on feature projections, contextual information is missed. Thus we use co-occurrence frequency to apply contextual information to our algorithm.

To calculate a co-occurrence frequency value between two terms \( t_i \) and \( t_j \), we count the number of documents that include both terms. It is separately calculated in each category of training data. Finally, the co-occurrence frequency value of the two terms is obtained by a maximum value among co-occurrence frequency values in each category as follows:

\[
c_{ij} = \max_{c_j} \left\{ c_{ij} \right\}
\]

(6)

where \( c_{ij} \) denotes a co-occurrence frequency value of \( t_i \) and \( t_j \), and \( c_{ij} \) denotes a co-occurrence frequency value of \( t_i \) and \( t_j \) in category \( c_j \).

TF-IDF values of two terms \( t_i \) and \( t_j \) in a test document \( d \) are modified by reflecting the co-occurrence frequency value. That is, terms with a high co-occurrence frequency value and a low category frequency value could have higher term weights as follows:
where i) \(tw(t,d)\) denotes a modified term weight assigned to term \(t\), ii) \(cf\) denotes the category frequency that is the number of categories in which \(t_i\) and \(t_j\) co-occur, and iii) \(\max \ co (t_i,t_j)\) is the maximum value among all co-occurrence frequency values.

### 3.2.4 The final voting score reflecting improvements and information of features

Through a feature selection process, we can measure how much information each feature contributes to our knowledge for correct classification; we employ the \(\chi^2\) statistics for feature selection in this paper. Moreover, since each feature in feature projections separately participates in voting, we can use information of features (calculated \(\chi^2\) statistics value) as an important weight in voting. As a result, in order to apply the improvements (formulae (5) and (7)) and the information of features to our algorithm, we calculate the voting score of each category \(c_i\) in the \(m\)-th feature \(t_m\) of a test document \(\hat{d}\) as the following formula:

\[
vs(c_i,t_m) = tw(t_m, \hat{d}) \cdot r(c_i,t_m) \cdot \log(1 + \chi^2(t_m))
\]

where \(\chi^2(t_m)\) denotes the calculated \(\chi^2\) statistics value of \(t_m\).

Here, since the modified TF-IDF value of a feature in a test document has to be also considered as an important factor, it is used for calculating the voting score.
A new text categorization algorithm using feature projections, named **TCFP**, is described in the following:

\[
d = \langle t_1, t_2, \ldots, t_n \rangle, \text{ a category set: } C = \{c_1, c_2, \ldots, c_m\}
\]

\[
\begin{align*}
\text{begin} \\
\quad & \text{for each category } c_j \\
& \quad \text{vote}[c_j] = 0 \\
\quad & \text{for each feature } t_i \\
& \quad tw(t_i, d) \text{ is calculated by formula (7)} \\
& \text{/* majority voting*/} \\
& \quad \text{for each feature } t_i \\
& \quad \quad \text{for each category } c_j \\
& \quad \quad \quad \text{vote}[c_j] = \text{vote}[c_j] + vs(c_j, t_i) \text{ by formula (8)} \\
& \text{for each category } c_j \\
& \quad \text{prediction} = \arg \max_{c_j} \text{vote}[c_j] \\
\text{return prediction} \\
\text{end}
\end{align*}
\]

In the training phase, our algorithm needs only simple process; the training documents are projected on each of their features, and numerical values for the proportional voting (formula (5)) and co-occurrence frequency values (formula (6)) are calculated.
4. Other Classifiers used in our Experiments

To compare TCFP to other classifiers, we implemented Rocchio, Naive Bayes, and SVMs. In this section, these classifiers are briefly described. For Rocchio and SVMs, we use the same TF-IDF scheme as formulae (1) and (2).

4.1 Rocchio

Rocchio is an effective method using relevance judgments for query expansion in information retrieval and filtering (Lewis et al., 1996; Salton et al., 1990). Applied to text categorization, it uses a vector to represent each class and document, and computes their similarity using the cosine value of these two vectors. A test document is then assigned to a category with the highest cosine score. The vector representation for a class, called prototype or centroid, is constructed by combining document vectors into a prototype vector \( \tilde{c}_i \) for each class \( c_i \). First, both the document vectors of the positive examples and those of the negative examples are summed up. The prototype vector is then calculated as a weighted difference of each as follows:

\[
\tilde{c}_i = \alpha \frac{1}{N^+} \sum_{d \in N^+} \tilde{d} - \beta \frac{1}{N^-} \sum_{d \in N^-} \tilde{d}
\]

where \( N \) is the number of documents in the training set, and \( \alpha, \beta \) are parameters that adjust the relative impact of positive and negative training examples.

4.2 Naive Bayes

Naive Bayes probabilistic classifiers are also commonly used in text categorization (McCallum et al. 1998; Ko et al., 2000; Yang et al., 2002). The basic idea is to use the joint probabilities of words and
categories to estimate the probabilities of categories given a document. The naive part of such a model is the assumption of word independence. We use the Naive Bayes classifier with minor modifications based on Kullback-Leibler Divergence (Ko et al., 2000; Craven et al., 2000). Given a document $d$ for classification, we calculate the probabilities of each category $c_j$ as follows:

$$
\text{Pr}(c_j \mid d) = \frac{\text{Pr}(c_j) \text{Pr}(d \mid c_j)}{\text{Pr}(d)} = \frac{\text{Pr}(c_j) \prod_{t=1}^{T} \text{Pr}(t_j \mid c_j)^{N(t_j,d)}}{\sum_{i=1}^{T} \text{Pr}(t_i \mid d)}
$$

where

i) $n$ is the number of words in $d$

ii) $t_i$ is the $i$-th word in the vocabulary

iii) $T$ is the size of the vocabulary

iv) $N(t_i,d)$ is the frequency of word $t_i$ in document $d$

The category predicted by this classifier for a given document is simply the category with the highest score.

Here, the Laplace smoothing is used for the estimate of the probability of word $t_i$ in class $c_j$ as follows:

$$
\text{Pr}(t_i \mid c_j) = \frac{N(t_i,c_j) + 1}{\sum_{j=1}^{T} N(t_j,c) + T}
$$

where $N(t_i,c)$ is the frequency of word $t_i$ in category $c$, and $T$ is the total number of unique words across all categories.
4.3 Support Vector Machines

Support Vector Machines (SVMs) were proposed by Vapnik (1995) for solving two-class pattern recognition problems. The SVM problem is to find the decision surface that maximizes the margin between positive examples and negative examples in a training data.

The decision surface by SVM for linearly separable space is a hyperplane as follows:

$$\vec{w} \cdot \vec{x} = b$$

(12)

$\vec{x}$ is an arbitrary test data vector, and the vector $\vec{w}$ and the constant $b$ are learned from a training data set.

The SVM problem can be solved using quadratic programming techniques (Vapnik, 1995). The algorithms for solving linearly separable cases can be extended for solving linearly non-separable cases by mapping the original data vectors to a space of higher dimensions.

Since SVM is a binary classifier, we must extend it to multi-class classifier. There are several methods of the multi-class classifier for SVMs (Weston et al., 1999). We employ the One-Against-the Rest method among them. This method requires $k$ binary classifiers $f_{c_i}(x)$ ($1 \leq c_i \leq k$) to be constructed for all categories. In training for category $c_i$, all data of category $c_i$ are used as positive examples and all data of the other categories as negative examples. Given a test example $\vec{x}$, its category $c$ is determined by the classifier that gives the largest discriminating function value as follows:

$$c = \arg \max_{c_i} f_{c_i}(x)$$

(13)

Joachims implemented an efficient SVMs toolkit, $SVM^light$ (Joachims, 1998). This toolkit was used in our experiments.
5. Empirical Evaluation

In this section, we provide empirical evidence that TCFP is a useful classifier for Text Categorization. We present experimental results with three different test data sets: UseNet newsgroups (Newsgroups), web pages (WebKB), and newswire articles (Reuters). Results show that TCFP outperforms k-NN and it is a much faster classifier than k-NN.

5.1 Data Sets and Experimental Settings

The Newsgroups data set, collected by Ken Lang, contains about 20,000 articles evenly divided among 20 UseNet discussion groups (McCallum et al., 1998). Many of the categories fall into confusable clusters; for example, five of them are comp.* discussion groups, and three of them discuss about religion. After removing words that occur only once or on a stop word list, the average vocabulary from five training data has 51,325 words (with no stemming).

The second data set comes from the WebKB project at CMU (Yang et al., 2002). This data set contains web pages gathered from university computer science departments. The pages are divided into seven categories: course, faculty, project, student, department, staff, and other. In this paper, we used the four most populous entity-representing categories: course, faculty, project, and student. The resulting data set consists of 4,198 pages with a vocabulary of 18,742 words. It is an uneven data set; the largest category has 1,641 pages and the smallest one has 503 pages.

The Reuters 21578 Distribution 1.0 data set consists of 12,902 articles and 90 topic categories from the Reuters newswire. Following other studies by Nigam (2001) and Joachims (1998), we built binary classifiers for each category to identify the news topic. Since the documents in this data set can have multiple class labels, each category is traditionally evaluated with a binary classifier. To split train/test data, we follow a standard ‘ModApte’ split. The standard ‘ModApte’ train/test split divides the articles by time, such that the later 3,299 documents form the test set, and the earlier 9,603 are available for training.
We used all the words inside the title and body. We used a stoplist and no stemming. The vocabulary from training data has 21,023 words.

For fair evaluation in Newsgroups and WebKB, we used the five-fold cross-validation method. That is, each data set was split into five subsets, and each subset was used once as test data in a particular run while the remaining subsets were used as training data for that run. The split into training and test sets for each run was the same for all the classifiers. Therefore, all results of the experiments are averages of five runs.

To compare TCFP with other algorithms for improving execution speed, we implemented $k$-NNFP and $k$-NN with pruning. We used DROP4 as a pruning technique (Wilson et al., 1999). With DROP4, only about 26% of the original training documents in data sets was retained. The $k$ in $k$-NNFP was set to 20 and the $k$ in $k$-NN with pruning was set to 30. In addition, we implemented other conventional classifiers: $k$-NN, Naive Bayes, Rocchio, and SVMs. The $k$ in $k$-NN was set to 30, and $\alpha=16$ and $\beta=4$ were used in the Rocchio classifier. For SVMs, we used the linear model offered by SVM$^\text{light}$.

As performance measures, we followed the standard definition of recall ($r$), precision ($p$), and $F_1$ measure ($2rp/(r+p)$) (Yang, 1999). Recall is the probability that a document belonging to any category is classified into this category and Precision is the probability that a document predicted to be in any category is classified into this category. The $F_1$ measure balances recall and precision in a way that gives them equal weight. Results on Reuters are reported as precision-recall breakeven points, which is a standard information retrieval measure for binary classification; Given a ranking of documents, the precision-recall breakeven point is the value at which precision and recall are equal (Yang, 1999). For evaluating performance average across categories, we used the micro-averaging method (Yang, 1999). The micro-averaging is to count the decisions for all the categories in a joint pool and compute the global recall, precision and $F_1$ values for that global pool.
5.2 Feature Selection

The size of vocabulary in our experiment is selected by ranking words according to their $\chi^2$ statistics with respect to the category. Using the two-way contingency table of a word $t$ and a category $c$, the word-goodness measure is defined as follows (Yang et al., 1997; Ko et al., 2003):

$$
\chi^2(t, c) = \frac{N \times (AD - CB)^2}{(A + C)(B + D)(A + B)(C + D)}
$$

where i) $A$ is the number of times $t$ and $c$ co-occur, ii) $B$ is the number of times $t$ occurs without $c$, iii) $C$ is the number of times $c$ occurs without $t$, iv) $D$ is the number of times neither $c$ nor $t$ occurs, and vi) $N$ is the total number of documents.

To measure the goodness of a word in a global feature selection, we combine the category-specific scores of a word as follows:

$$
\chi^2(t) = \max_{i=1}^{m} \{ \chi^2(t, c_i) \}
$$

where $m$ denotes the number of categories.

5.3 Experimental Results

5.3.1 Comparison of performances on the Newsgroups data set

Figure 2 and Table 2 show results from TCFP, $k$-NN, and other algorithms for improving execution speed ($k$-NNFP and $k$-NN with pruning). In addition, we added another type of TCFP to our experiment. This is TCFP without contextual information, which does not use formula (7) for contextual information.
Figure 2. The classification performances of TCFP, $k$-NN, $k$-NNFP, and $k$-NN with pruning according to the number of features on the Newsgroups data set

Table 2. The best micro-average $F_1$ results for each classifier on the Newsgroups data set

<table>
<thead>
<tr>
<th></th>
<th>TCFP</th>
<th>TCFP without context</th>
<th>$k$-NN</th>
<th>$k$-NNFP</th>
<th>$k$-NN with pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro-avg $F_1$</td>
<td>85.52</td>
<td>85.42</td>
<td>85.15</td>
<td>81.93</td>
<td>81.34</td>
</tr>
</tbody>
</table>

As a result, TCFP achieved the highest micro-average $F_1$ score. Also, TCFP without contextual information presented the nearly same performance as TCFP. Although, in all vocabulary sizes, TCFP without contextual information achieved a little lower performance than TCFP, it can also be a useful classifier due to its simplicity and fast execution speed (see Table 6). As shown in Table 2, $k$-NNFP and $k$-NN with pruning algorithms are inferior to TCFP although they achieved faster execution speed than $k$-NN (see Table 6).

For further evaluation, we implemented other conventional classifiers: $k$-NN, Naive Bayes, Rocchio classifier, and SVMs, and we compared them with TCFP. The classification performance of each classifier is shown in Figure 3 and Table 3.

Figure 3. The classification performances of TCFP, $k$-NN, Naive Bayes, Rocchio, and SVM according to the number of features on the Newsgroups data set

Table 3. The best micro-average $F_1$ results for each classifier on the Newsgroups data set

<table>
<thead>
<tr>
<th></th>
<th>TCFP</th>
<th>$k$-NN</th>
<th>SVM</th>
<th>NB</th>
<th>Rocchio</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro-avg $F_1$</td>
<td>85.52</td>
<td>85.15</td>
<td>87.32</td>
<td>82.51</td>
<td>81.68</td>
</tr>
</tbody>
</table>

The results show that TCFP is superior to $k$-NN, Naive Bayes, and Rocchio classifiers. But TCFP produced lower performance than SVMs, which has been reported as a classifier with the best
performance in this literature.

5.3.2 Comparison of performances in an uneven data set, WebKB.

In the above experiments, the Newsgroups data set, which is an evenly divided data set, was used. If an uneven data set is used, a problem can occur. The cause of the problem is that a larger category has more voting candidates than a smaller category. Thus we simply modified the majority voting score of TCFP with the following formula:

\[
vote \{ c_j \} = \frac{\max_{c_j} \{ \text{num} ( d, c_j) \} / \text{num} ( d, c_j) \}}{\text{vote} \{ c_j \}}
\]

where \(\text{num}(d,c_j)\) denotes the number of training documents in category \(c_j\).

By applying this formula to the TCFP algorithm, we can normalize the number of voting candidates for each category.

The results of the modified algorithm are shown in Figure 4 and Table 4. As we can see in this table, the modified TCFP algorithm also showed similar performance on the uneven data set, WebKB.

Table 4. The best micro-average F\(_1\) results for each classifier

<table>
<thead>
<tr>
<th></th>
<th>TCFP</th>
<th>TCFP without context</th>
<th>k-NN</th>
<th>k-NNFP</th>
<th>k-NN with pruning</th>
<th>SVM</th>
<th>NB</th>
<th>Rocchio</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro-avg F(_1)</td>
<td><strong>88.07</strong></td>
<td>86.52</td>
<td>84.83</td>
<td>82.78</td>
<td>81.33</td>
<td>91.75</td>
<td>85.22</td>
<td>85.98</td>
</tr>
</tbody>
</table>
5.3.3 Comparison of performances on the Reuters data set

Since the Reuters data set can have documents with multiple class labels, we built binary classifiers for each category. Table 5 shows the precision-recall breakeven points on the ten most frequent Reuters categories and their micro-averaging performance.

<table>
<thead>
<tr>
<th>Category</th>
<th>TCFP</th>
<th>TCFP without context</th>
<th>k-NN</th>
<th>k-NNFP</th>
<th>k-NN with pruning</th>
<th>SVM</th>
<th>NB</th>
<th>Rocchio</th>
</tr>
</thead>
<tbody>
<tr>
<td>acq</td>
<td>94.29</td>
<td>93.6</td>
<td>94.57</td>
<td>90.54</td>
<td>93.32</td>
<td>96.94</td>
<td>93.88</td>
<td>88.31</td>
</tr>
<tr>
<td>corn</td>
<td>75</td>
<td>75</td>
<td>78.57</td>
<td>60.71</td>
<td>78.57</td>
<td>91.07</td>
<td>64.28</td>
<td>66.07</td>
</tr>
<tr>
<td>crude</td>
<td>84.65</td>
<td>79.36</td>
<td>82.01</td>
<td>82.53</td>
<td>80.42</td>
<td>86.24</td>
<td>84.12</td>
<td>78.3</td>
</tr>
<tr>
<td>earn</td>
<td>97.42</td>
<td>96.59</td>
<td>95.86</td>
<td>94.2</td>
<td>95.03</td>
<td>98.62</td>
<td>96.41</td>
<td>96.13</td>
</tr>
<tr>
<td>grain</td>
<td>84.56</td>
<td>84.56</td>
<td>80.53</td>
<td>78.52</td>
<td>78.52</td>
<td>94.63</td>
<td>81.87</td>
<td>79.86</td>
</tr>
<tr>
<td>interest</td>
<td>76.33</td>
<td>71.75</td>
<td>74.04</td>
<td>69.46</td>
<td>72.51</td>
<td>78.62</td>
<td>74.04</td>
<td>73.28</td>
</tr>
<tr>
<td>money-fx</td>
<td>72.06</td>
<td>72.06</td>
<td>76.53</td>
<td>73.18</td>
<td>74.86</td>
<td>79.88</td>
<td>73.18</td>
<td>64.24</td>
</tr>
<tr>
<td>ship</td>
<td>84.26</td>
<td>83.14</td>
<td>78.65</td>
<td>87.64</td>
<td>79.77</td>
<td>87.64</td>
<td>82.02</td>
<td>80.89</td>
</tr>
<tr>
<td>trade</td>
<td>72.03</td>
<td>71.18</td>
<td>79.66</td>
<td>63.55</td>
<td>78.81</td>
<td>79.66</td>
<td>72.03</td>
<td>75.42</td>
</tr>
<tr>
<td>wheat</td>
<td>77.46</td>
<td>76.05</td>
<td>64.78</td>
<td>67.6</td>
<td>70.42</td>
<td>84.5</td>
<td>63.38</td>
<td>77.46</td>
</tr>
<tr>
<td>micro-avg</td>
<td><strong>90.01</strong></td>
<td><strong>88.8</strong></td>
<td><strong>88.93</strong></td>
<td><strong>86.26</strong></td>
<td><strong>88.23</strong></td>
<td><strong>93.32</strong></td>
<td><strong>88.62</strong></td>
<td><strong>86.47</strong></td>
</tr>
</tbody>
</table>

Results on the Reuters data set are also similar in performance to those of the previous two data sets. TCFP outperforms k-NN and other classifiers except SVMs. Although TCFP showed lower performance than SVMs, TCFP has several strong points in comparison with SVMs. Especially, TCFP is more robust from noisy data and it has faster execution speed than SVMs. These are discussed in the following discussion section in detail.

5.4 Discussions

TCFP overcomes the weaknesses of k-NN, including slow execution time and sensitivity to noise training data, while it maintains the strong points of k-NN, including the simplicity of algorithm, fast learning process, and high performance. On experimental results, TCFP produced high performance even though it showed a little lower performance than SVMs. Especially, TCFP outperforms k-NN over all three data sets.
Furthermore, since TCFP has a simple and fast learning process like \(k\)-NN, it can be a proper classifier for incremental learning. On the contrary, the training process of SVMs can be very complicated and time consuming, especially when dealing with noisy data. In addition, SVMs have some weaknesses such as slow running time and sensitivity to noisy data like \(k\)-NN. In order to verify the superiority of TCFP in execution speed and robustness from noisy data, we observed the running time in our experiments and conducted extra experiments for evaluating the robustness of each classifier from noisy data. The results are reported in the following subsections.

### 5.4.1 Running time observation

Table 6 shows the running times in CPU seconds for each classifier on each data set. Note that we included only testing phase for measuring the running time of each data set.

<table>
<thead>
<tr>
<th></th>
<th>TCFP without context</th>
<th>(k)-NNFP</th>
<th>Rocchio</th>
<th>TCFP</th>
<th>NB</th>
<th>SVM</th>
<th>(k)-NN with pruning</th>
<th>(k)-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Newsgroups</strong></td>
<td>0.7</td>
<td>0.85</td>
<td>0.8</td>
<td><strong>1.29</strong></td>
<td>1.22</td>
<td>14.71</td>
<td>37.97</td>
<td>142.54</td>
</tr>
<tr>
<td><strong>WebKB</strong></td>
<td><strong>0.14</strong></td>
<td>0.23</td>
<td>0.14</td>
<td><strong>0.55</strong></td>
<td>0.17</td>
<td>2.72</td>
<td>4.91</td>
<td>15.25</td>
</tr>
<tr>
<td><strong>Reuters</strong></td>
<td><strong>2.67</strong></td>
<td>2.7</td>
<td>3.34</td>
<td><strong>3.14</strong></td>
<td>7.01</td>
<td>39.94</td>
<td>15.88</td>
<td>65.86</td>
</tr>
</tbody>
</table>

Since the computations depend on the vocabulary sizes, we calculated the above numerical values by averaging running times from 1,000 to 10,000 features. In Table 6, the running time of TCFP is similar to other faster classifiers: Rocchio and Naive Bayes. Especially, it is about one hundred times faster than that of \(k\)-NN on the Newsgroups data set, and it also has much faster execution speed than SVMs. Though the results on each data set depend on the number of training documents and categories, TCFP showed fast execution speed over all the data sets. Especially, *TCFP without contextual information* is the fastest classifier over all the data sets. The time complexity of *TCFP without contextual information* is \(O(mc)\),
where \( m \) is the number of unique words in a test document and \( c \) is the number of categories. That is, the classification of TCFP without contextual information requires a simple calculation in proportion to the number of unique terms in the test document. On the other hand, in \( k \)-NN, a search in the whole training space must be done for each test document. Furthermore, TCFP without contextual information showed higher or similar performance than \( k \)-NN in our experiments.

5.4.2 Analysis of robustness from noisy data

We here analyze that TCFP is more robust from noisy data than \( k \)-NN and SVMs. For this experiment, we generated four data sets with increasing the number of noisy documents from 10% to 40% using the Newsgroups data set: these noisy documents were randomly chosen from each category and randomly assigned into other categories. The results of each classifier on each noisy data set are shown in Figure 5. These results are also obtained by a five-fold cross-validation method.

As shown in Figure 5, TCFP showed the best performance beginning from 20% noisy data set, and the decreasing rate of performance of TCFP is less than that of \( k \)-NN and SVMs. Especially, we observed that the performance of SVMs degraded rapidly when the number of noisy documents increased. In actual applications for text categorization, it is quite common that instances in the training collection contain some amount of noise due to various reasons such as wrong category assigned by humans and missing appropriate features. As seen from above results, the presence of noisy data will affect the performance of text categorization.

The robustness of TCFP is due to its voting mechanism. That is, the voting mechanism of TCFP, which
depends on separate voting in each feature, reduces the negative effect of possible noisy data and irrelevant features in classification.

6. Conclusions

In this paper, a new type of text categorization, TCFP, has been presented. This algorithm has been compared with $k$-NN and other classifiers in terms of classification performance, execution speed, and robustness from noisy data.

The experimental results show that, on the performance, TCFP outperforms Rocchio, Naive Bayes, $k$-NN, and other algorithms for improving execution speed ($k$-NNFP and $k$-NN with pruning), but it showed lower performance than SVMs. However, TCFP has several advantages over SVMs with respect to robustness from noisy data, fast execution speed, and simplicity of algorithm. Since each feature in TCFP individually contributes to the classification process, TCFP is robust from noisy data and irrelevant features. TCFP also has very fast execution speed. In running time observation, TCFP is about one hundred times faster than $k$-NN on the Newsgroups data set. Furthermore, by the simplicity of the TCFP algorithm, its implementation and training process can be done very easily.

Therefore, we can use TCFP in areas which require a robust, fast, and high-performance text classifier.

Acknowledgments

This work is a result of advanced backbone IT technology development project supported by Ministry of Information & Communication in republic of Korea (Project no. 2001-s-148) and it was partially supported by IAST of Sogang University.
References


Figure 1. Feature representation on feature projections in Text Categorization

Document representation in a conventional vector space

\( w \): weight, \( d \): document, \( c \): category

<table>
<thead>
<tr>
<th>( w, d, c )</th>
<th>( w, d, c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0, ( d_1, c_1 )</td>
<td>0.99, ( d_1, c_2 )</td>
</tr>
<tr>
<td>0.96, ( d_2, c_1 )</td>
<td>0.96, ( d_2, c_2 )</td>
</tr>
<tr>
<td>0.91, ( d_3, c_1 )</td>
<td>0.91, ( d_3, c_2 )</td>
</tr>
<tr>
<td>0.4, ( d_5, c_1 )</td>
<td>0.4, ( d_5, c_2 )</td>
</tr>
<tr>
<td>0.25, ( d_3, c_2 )</td>
<td>0.25, ( d_3, c_1 )</td>
</tr>
</tbody>
</table>

Feature representation in feature projections
Figure 2. The classification performances of TCFP, $k$-NN, $k$-NNFP, and $k$-NN with pruning according to the number of features on the Newsgroups data set.
Figure 3. The classification performances of TCFP, k-NN, Naive Bayes, Rocchio, and SVM according to the number of features on the Newsgroups data set.
Figure 4. The classification performances according to the number of features on the WebKB data set.
Figure 5. The classification performances of each classifier on four noisy data sets (10%, 20%, 30%, 40%)