Information Processing and Management 40 (2004) 191–208

INFORMATION PROCESSING & MANAGEMENT

www.elsevier.com/locate/infoproman

Using the feature projection technique based on a normalized voting method for text classification

Youngjoong Ko [*], Jungyun Seo

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

Received 20 August 2002; accepted 14 April 2003

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.
© 2003 Elsevier Ltd. All rights reserved.

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, Slattery, & Ghani, 2002), Rocchio (Lewis, Schapire, Callan, & Papka, 1996), Naive Bayes (McCallum & Nigam, 1998), and support vector machines (SVMs) (Joachims, 1998) are well-known algorithms.

[*] Corresponding author. Tel.: +82-2-706-8954; fax: +82-2-704-8273.
E-mail addresses: kyj@nlpzodiac.sogang.ac.kr (Y. Ko), seojy@ccs.sogang.ac.kr (J. Seo).

0306-4573/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0306-4573(03)00029-3

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 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 online scoring of all training documents to find the *k* nearest neighbors of a test document. In order to reduce this scaling problem in online ranking, a number of techniques have been studied in the literature. Techniques such as the *instance pruning technique* (Wilson & Martinez, 1999) and *feature projections* (Akkus & Guvenir, 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, Dennis, & Marc, 1991), the *Typical Instance Based Learning* (Zhang, 1992), and the *Reduction Techniques* (*RT1–RT3*) (Wilson & Martinez, 1997). These pruning techniques and others were surveyed in depth by Wilson and Martinez (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 & Guvenir, 1996). They called their resulting algorithm *k-NN 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 was 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 *DROP4* pruning algorithm, and the *k*-NNFP algorithm.

2.1. The k-nearest neighbor 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, Hart, & Stork, 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:

$$\vec{d} = \langle w(t_1,\vec{d}), w(t_2,\vec{d}), \ldots, w(t_n,\vec{d})\rangle \tag{1}$$

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

$$w(t,\vec{d}) = \frac{(1+\log tf(t,\vec{d})) \times \log(N/n_t)}{\|\vec{d}\|} \tag{2}$$

where

(i) $w(t,\vec{d})$ is the weight of term t in document d,
(ii) $tf(t,\vec{d})$ is the within-document Term Frequency (TF),
(iii) $\log(N/n_t)$ 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) $\|\vec{d}\| = \sqrt{\sum_{t\in\vec{d}} w(t,\vec{d})^2}$ 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:

$$s(c_j,\vec{d}) = \sum_{\vec{d}'\in R_k(\vec{d})\cap D_j} \cos(\vec{d}',\vec{d}) \tag{3}$$

where $R_k(\vec{d})$ 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 & Martinez, 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 *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).

Remove instance i from S if at least as many of its associates in T 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 *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 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 Fig. 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 mth feature t_m is $\mathrm{dist}_m(t_m(i), t_m(j))$ as follows:

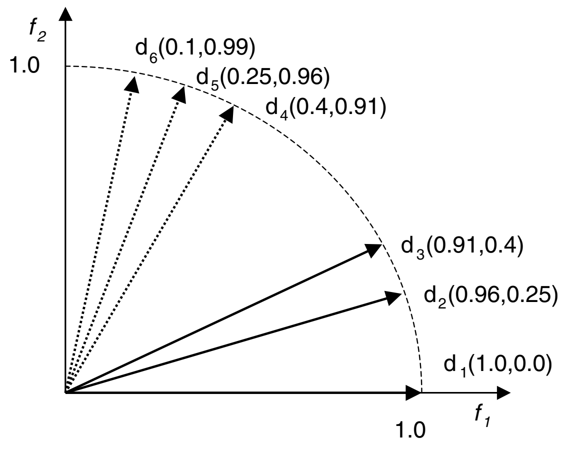
$$\mathrm{dist}_m(t_m(i), t_m(j)) = |w(t_m, \vec{d}_i) - w(t_m, \vec{d}_j)| \tag{4}$$

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

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.



Document representation in a conventional vector space

w: weight, d: document, c: category

w, d, c
1.0, d_1, c_1
0.96, d_2, c_1
0.91, d_3, c_1
0.4, d_4, c_2
0.25, d_5, c_2
0.1, d_6, c_2

feature projections on feature f_1

w, d, c
0.99, d_6, c_2
0.96, d_5, c_2
0.91, d_4, c_2
0.4, d_3, c_1
0.25, d_2, c_1

feature projections on feature f_2

Feature representation in feature projections

Fig. 1. Feature representation on feature projections in text categorization.

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 $c_2 = \{d_4, d_5, d_6\}$. Fig. 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.

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

Average DF	Maximum DF	Minimum DF	The number of features DF < k (20)
54.59	8407	4	6489

3.2. Problems in applying feature projections to text categorization

There are three problems: (1) the diversity of 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.

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 & McGill, 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 d is calculated by the following formula:

$$r(c_j, t_m) = \sum_{t_m(l) \in I_m} w(t_m, \vec{d}_l) \cdot y(c_j, t_m(l)) \Bigg/ \sum_{t_m(l) \in I_m} w(t_m, \vec{d}_l) \tag{5}$$

In the above formula, I_m denotes a set of elements selected for voting and $y(c_j, t_m(l)) \in \{0.1\}$ is a function; if the category for an element $t_m(l)$ is equal to c_j, 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_l, 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:

$$\mathrm{co}(t_i, t_l) = \max_{c_j}\{\mathrm{co}(t_i, t_l, c_j)\} \tag{6}$$

where $\mathrm{co}(t_i, t_l)$ denotes a co-occurrence frequency value of t_i and t_l, and $\mathrm{co}(t_i, t_l, c_j)$ denotes a co-occurrence frequency value of t_i and t_l 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:

$$\mathrm{tw}(t_i, \vec{d}) = w(t_i, \vec{d}) \cdot \left(1 + \left(\frac{1}{\log(\mathrm{cf} + 1)}\right) \cdot \left(\frac{\log(\mathrm{co}(t_i, t_j) + 1)}{\log(\mathrm{maxco}(t_i, t_j) + 1)}\right)\right) \tag{7}$$

where (i) $\mathrm{tw}(t_i, d)$ denotes a modified term weight assigned to term t_i, (ii) cf denotes the category frequency that is the number of categories in which t_i and t_j co-occur, and (iii) $\mathrm{maxco}(t_i, t_j)$ 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 χ^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 χ^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_j in the mth feature t_m of a test document d as the following formula:

$$\mathrm{vs}(c_j, t_m) = \mathrm{tw}(t_m, \vec{d}) \cdot r(c_j, t_m) \cdot \log(1 + \chi^2(t_m)) \tag{8}$$

where $\chi^2(t_m)$ denotes the calculated χ^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.

3.3. *A new text categorization algorithm using feature projections*

A new text categorization algorithm using feature projections, named **TCFP**, is described in the following:

```
test document: $\vec{d} = \langle t_1, t_2, \ldots, t_n \rangle$, a category set: $C = \{c_1, c_2, \ldots, c_m\}$

    begin
        for each category $c_j$
            vote$[c_j] = 0$
        for each feature $t_i$
            $tw(t_i, d)$ is calculated by formula (7)
    /*majority voting*/
    for each feature $t_i$
        for each category $c_j$
            vote[$c_j$] = vote[$c_j$] + $vs(c_j,t_i)$ by formula (8)

    for each category $c_j$
        prediction = $\arg\max_{c_j} vote[c_j]$

    return prediction
end
```

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 & Buckley, 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 $\vec{c}_j$ for each class c_j. 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:

$$\vec{c}_j = \alpha\frac{1}{|c_j|}\sum_{\vec{d}\in c_j}\vec{d} - \beta\frac{1}{N-|c_j|}\sum_{\vec{d}\in N-c_j}\vec{d} \tag{9}$$

where N is the number of documents in the training data set, and α, β 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 (Ko & Seo, 2000; McCallum & Nigam, 1998; 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 (Craven et al., 2000; Ko & Seo, 2000). Given a document d for classification, we calculate the probabilities of each category c_j as follows:

$$\Pr(c_j|d) = \frac{\Pr(c_j)\Pr(d|c_j)}{\Pr(d)} = \Pr(c_j)\prod_{i=1}^{T}\Pr(t_i|c_j)^{N(t_i,d)}$$
$$\propto \frac{\log \Pr(c_j)}{n} + \sum_{i=1}^{T}\Pr(t_i|d)\log\left(\frac{\Pr(t_i|c_j)}{\Pr(t_i|d)}\right) \tag{10}$$

where

(i) n is the number of words in document d,
(ii) t_i is the ith 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:

$$\Pr(t_i|c_j) = \frac{N(t_i, c_j) + 1}{\sum_{i=1}^{T} N(t_j, c_j) + T} \tag{11}$$

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

4.3. Support Vector Machines

SVMs were proposed by Vapnic (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:

$$\vec{w}\cdot\vec{x} - b = 0 \tag{12}$$

$\vec{x}$ 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 (Vapnic, 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 & Watkins, 1999). We employ the *One-Against-the Rest* method among them. This method requires k binary classifiers $f_{c_j(x)}(1 \leqslant c_j \leqslant k)$ to be constructed for all categories. In training for category c_j, all data of category c_j are used as positive examples and all data of the other categories as negative examples. Given a test example

x, its category c is determined by the classifier that gives the largest discriminating function value as follows:

$$c = \arg\max_{c_j} f_{c_j}(x) \tag{13}$$

Joachims implemented an efficient SVMs toolkit, SVM$^{\text{light}}$ (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 (**Newsgroup**), 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 & Nigam, 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 4198 pages with a vocabulary of 18,742 words. It is an uneven data set; the largest category has 1641 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 3299 documents form the test set, and the earlier 9603 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 14,219 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 & Martinez, 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 χ^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 (Ko, Park, & Seo, in press; Yang & Pedersen, 1997):

$$\chi^2(t,c) = \frac{N \times (AD - CB)^2}{(A+C) \times (B+D) \times (A+B) \times (C+D)} \tag{14}$$

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:

$$\chi^2(t) = \max_{j=1}^{m}\{\chi^2(t,c_j)\} \tag{15}$$

where m denotes the number of categories.

5.3. Experimental results

5.3.1. Comparison of performances on the Newsgroups data set

Fig. 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.

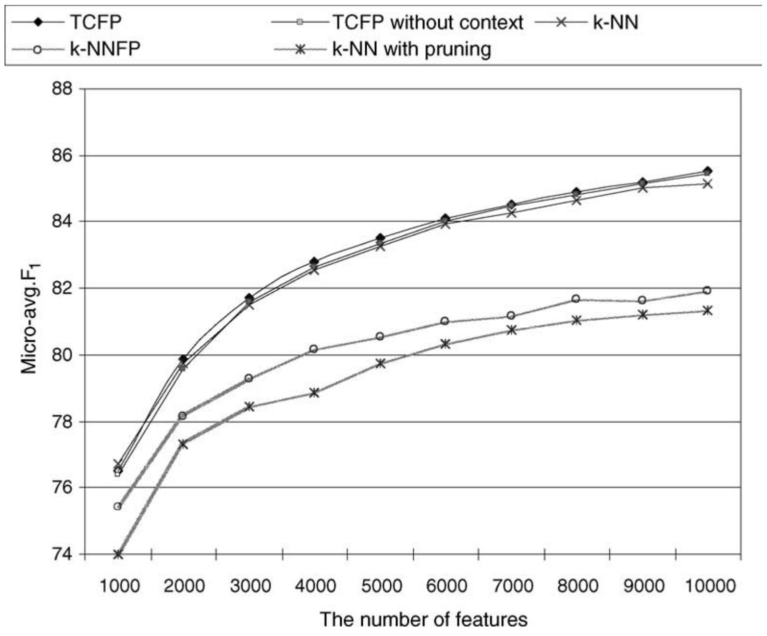


Fig. 2. The classification performances of TCFP, *k*-NN, *k*-NNFP, and *k*-NN with pruning according to the number of features on the Newsgroups data set.

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

	TCFP	TCFP without context	*k*-NN	*k*-NNFP	*k*-NN with pruning
Micro-avg. F_1	***85.52***	85.42	85.15	81.93	81.34

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 Fig. 3 and Table 3.

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

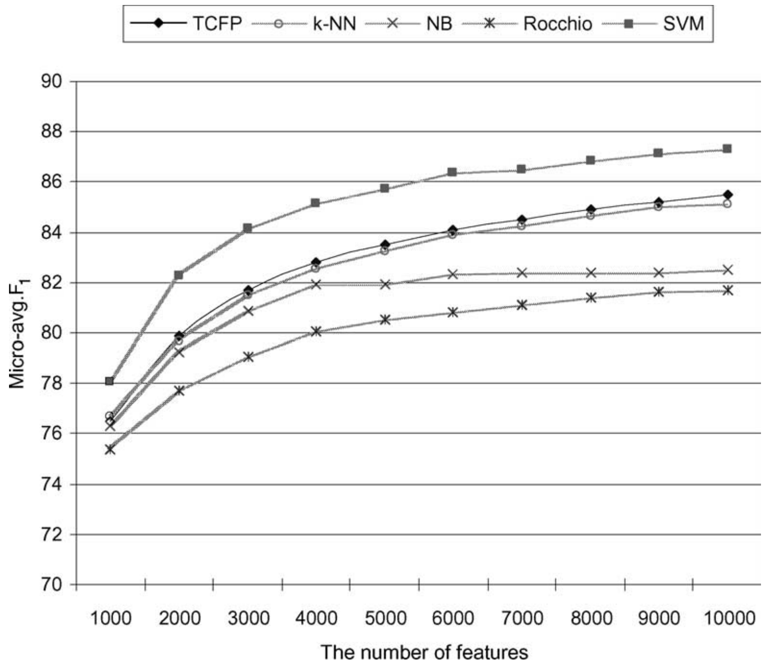


Fig. 3. The classification performances of TCFP, *k*-NN, Naive Bayes, Rocchio, and SVM according to the number of features on the Newsgroups data set.

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

	TCFP	*k*-NN	SVM	NB	Rocchio
Micro-avg. F_1	***85.52***	85.15	87.32	82.51	81.68

category has more voting candidates than a smaller category. Thus we simply modified the majority voting score of TCFP with the following formula:

$$\text{vote}[c_j] = \text{vote}[c_j] \cdot \left(\max_{c_i}\{\text{num}(d, c_i)\}/\text{num}(d, c_j) \right) \tag{16}$$

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 Fig. 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**.

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.

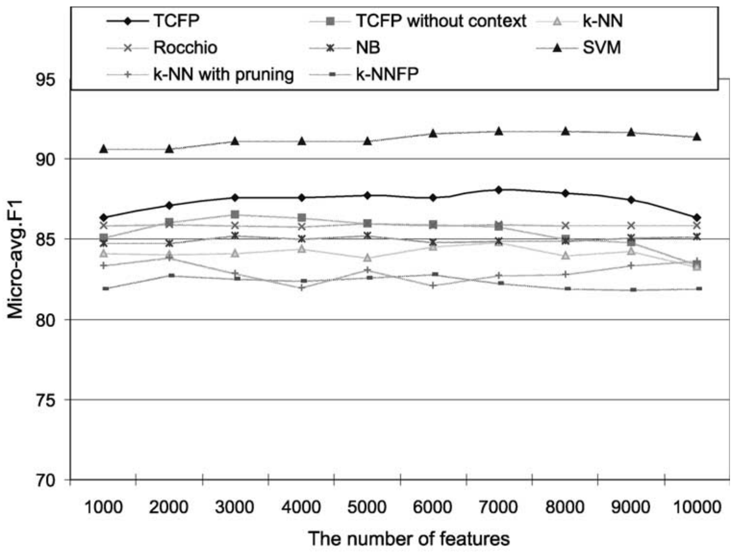


Fig. 4. The classification performances according to the number of features on the WebKB data set.

Table 4
The best micro-average F_1 results for each classifier

	TCFP	TCFP without context	*k*-NN	*k*-NNFP	*k*-NN with pruning	SVM	NB	Rocchio
Micro-avg. F_1	***88.07***	86.52	84.83	82.78	83.81	91.75	85.22	85.98

Table 5
Precision–recall breakeven points showing performances of binary classifiers on the Reuters data set

	TCFP	TCFP without context	*k*-NN	*k*-NNFP	*k*-NN with pruning	SVM	NB	Rocchio
Acq	94.29	93.6	94.57	90.54	93.32	96.94	93.88	88.31
Corn	75	75	78.57	60.71	78.57	91.07	64.28	66.07
Crude	84.65	79.36	82.01	82.53	80.42	86.24	84.12	78.3
Earn	97.42	96.59	95.86	94.2	95.03	98.62	96.41	96.13
Grain	84.56	84.56	80.53	78.52	78.52	94.63	81.87	79.86
Interest	76.33	71.75	74.04	69.46	72.51	78.62	74.04	73.28
Money-fx	72.06	72.06	76.53	73.18	74.86	79.88	73.18	64.24
Ship	84.26	83.14	78.65	87.64	79.77	87.64	82.02	80.89
Trade	72.03	71.18	79.66	63.55	78.81	79.66	72.03	75.42
Wheat	77.46	76.05	64.78	67.6	70.42	84.5	63.38	77.46
Micro-avg.	***90.01***	88.8	88.93	86.26	88.23	93.32	88.62	86.47

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 execution 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 sections.

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.

Since the computations depend on the vocabulary sizes, we calculated the above numerical values by averaging running times from 1000 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

Table 6
Average running time of each classifier on each data set

	TCFP without context	*k*-NNFP	Rocchio	TCFP	NB	SVM	*k*-NN with pruning	*k*-NN
Newsgroups	***0.7***	0.85	0.8	***1.29***	1.22	14.71	37.97	142.54
WebKB	***0.14***	0.23	0.14	***0.55***	0.17	2.72	4.91	15.25
Reuters	***2.67***	2.7	3.34	***3.14***	7.01	39.94	15.88	65.86

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 Fig. 5. These results are also obtained by a five-fold cross-validation method.

As shown in Fig. 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.

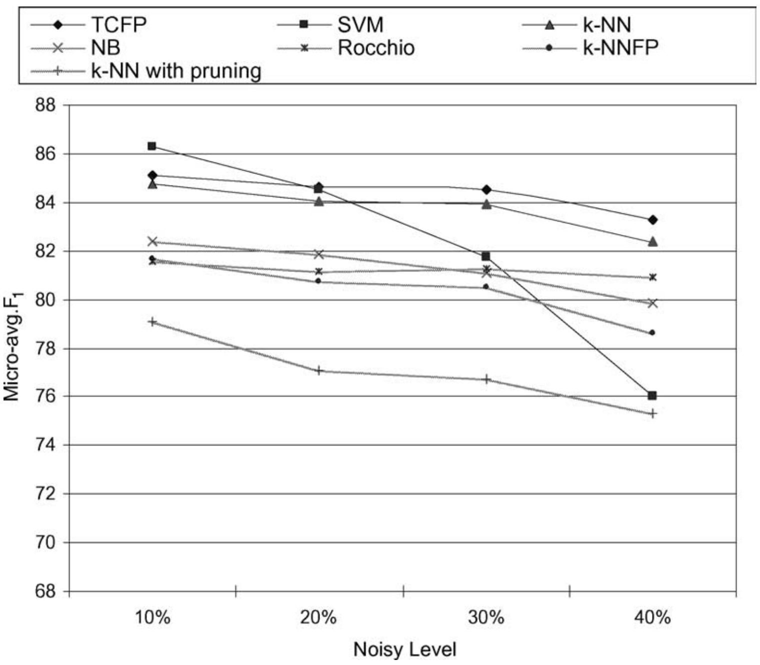


Fig. 5. The classification performances of each classifier on four noisy data sets (10%, 20%, 30%, 40%).

6. Conclusions

In this paper, a new type of text classifier, 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.

Acknowledgements

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

References

Aha, D. W., Dennis, K., & Marc, K. A. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37–66.

Akkus, A., & Guvenir, H. A. (1996). *K* nearest neighbor classification on feature projections. In *Proceedings of ICML'96, Italy* (pp. 12–19).

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (2000). Learning to construct knowledge bases from the world wide web. *Artificial Intelligence, 118*(1–2), 69–113.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). John Wiley & Sons.

Hart, P. E. (1968). The condensed nearest neighbor rule. *Institute of Electrical and Electronics Engineers Transactions on Information Theory, 14*, 515–516.

Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features. In *Proceedings of European conference on machine learning (ECML)* (pp. 137–142).

Ko, Y., Park, J., & Seo, J. (in press). Improving text categorization using the importance of sentences. *Information Processing and Management*.

Ko, Y., & Seo, J. (2000). Automatic text categorization by unsupervised learning. In *Proceedings of the 18th international conference on computational linguistics (COLING)* (pp. 453–459).

Lewis, D. D., Schapire, R. E., Callan, J. P., & Papka, R. (1996). Training algorithms for linear text classifiers. In *Proceedings of the 19th international conference on research and development in information retrieval (SIGIR'96)* (pp. 289–297).

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive bayes text classification. *AAAI'98 workshop on learning for text categorization* (pp. 41–48).

Nigam, K. P. (2001). Using unlabeled data to improve text classification, the dissertation for the degree of Doctor of Philosophy.

Salton, G., & Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management, 24*, 513–523.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of American Society for Information Sciences, 41*, 288–297.

Salton, G., & McGill, M. J. (1983). *Introduction to modern information retrieval*. McGraw-Hill, Inc.

Vapnic, V. (1995). *The nature of statistical learning theory*. New York: Springer.

Weston, V., & Watkins, C. (1999). Support vector machines for multi-class pattern recognition. In *Proceedings of the seventh European symposium on artificial neural networks (ESANN-99)*.

Wilson, D. R., & Martinez, T. R. (1997). Instance pruning techniques. In *Proceedings of the fourteenth international conference* (pp. 404–411).

Wilson, D. R., & Martinez, T. R. (1999). Reduction techniques for exemplar-based learning algorithms. *Machine Learning, 38*(3), 257–286.

Yang, Y. (1994). Expert network: effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of 17th international ACM SIGIR conference on research and development in information retrieval (SIGIR'94)* (pp. 13–22).

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval, 1*(1/2), 67–88.

Yang, Y., & Pedersen, J. P. (1997). Feature selection in statistical learning of text categorization. In *Proceedings of the fourteenth international conference on machine learning* (pp. 412–420).

Yang, Y., Slattery, S., & Ghani, R. (2002). A study of approaches to hypertext categorization. *Journal of Intelligent Information Systems, 18*(2).

Zhang, J. (1992). Selecting typical instances in instance-based learning. In *Proceedings of the ninth international conference on machine learning*.