Expert Systems with Applications 30 (2006) 290–298

Expert Systems with Applications

www.elsevier.com/locate/eswa

An effective refinement strategy for KNN text classifier

Songbo Tan[a,b,*]

[a]Software Department, Institute of Computing Technology, Chinese Academy of Sciences, P.O. Box 2704, Beijing 100080, People's Republic of China
[b]Graduate School of the Chinese Academy of Sciences, People's Republic of China

Abstract

Due to the exponential growth of documents on the Internet and the emergent need to organize them, the automated categorization of documents into predefined labels has received an ever-increased attention in the recent years. A wide range of supervised learning algorithms has been introduced to deal with text classification. Among all these classifiers, *K*-Nearest Neighbors (KNN) is a widely used classifier in text categorization community because of its simplicity and efficiency. However, KNN still suffers from inductive biases or model misfits that result from its assumptions, such as the presumption that training data are evenly distributed among all categories. In this paper, we propose a new refinement strategy, which we called as DragPushing, for the KNN Classifier. The experiments on three benchmark evaluation collections show that DragPushing achieved a significant improvement on the performance of the KNN Classifier.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: KNN; Text classification; Information retrieval; Data mining

1. Introduction

With the exponential growth of textual information available from the Internet, there has been an emergent need to find and organize relevant information in text collections. For this purpose, automatic text categorization becomes a significant tool to utilize text information efficiently and effectively. Text categorization aims to automatically place the pre-defined labels on previously unseen documents. It is an active research area in information retrieval, machine learning and natural language processing. In the research community, the dominant approach to this problem is based on machine learning techniques: a general inductive process automatically builds a classifier by learning, from a set of pre-classified documents, the characteristics of the categories. A wide range of supervised learning algorithms has been applied to this area, such as *K*-Nearest Neighbor (KNN) (Sebastiani, 2002), Centroid-Based Classifier (CB) (Han & Karypis, 2000), Naive Bayes (Sebastiani, 2002), Decision Trees (Sebastiani, 2002), Winnow (van Mun), Voting (Kjersti Aas & Line Eikvil), and Support Vector Machines (SVM) (Sebastiani, 2002).

Among all these algorithms, *K*-Nearest Neighbor is a widely used text classifier because of its simplicity and efficiency. Its training-phase consists of nothing more than storing all training examples as classifier, thus it has often been called as lazy learner since 'it defers the decision on how to generalize beyond the training data until each new query instance is encountered' (Sebastiani, 2002).

In despite of its merits, *K*-Nearest Neighbor still suffers from inductive biases or model misfits. For examples, it takes the assumption that training data are evenly distributed among all categories. In practice, however, there is no guarantee that the training set is balanced populated, such as Reuter-21578 and TDT-5, and especially for Reuter-21578 in which the documents are extremely unevenly distributed. For unbalanced text corpora, the majority class tends to have more examples than minority category in the *K*-nearest-neighbor set for one test document *d*. If we employ traditional KNN decision rule to classify the test document *d*, the test document d_0 tends to be assigned the majority class label. As a result, the big category tends to have high classification accuracy, while the other the minority class tends to have low classification accuracy. Therefore, the total performance of KNN will be inevitably harmed.

In this work, we propose DragPushing as a refinement strategy to enhance the performance of KNN by means of

* Address: Software Department, Institute of Computing Technology, Chinese Academy of Sciences, P.O. Box 2704, Beijing 100080, People's Republic of China. Tel.: +86 10 8844 9181x713; fax: +86 10 8845 5011.
E-mail address: tansongbo@software.ict.ac.cn

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.07.019

on-line modification of the KNN classifier models. The main idea behind my strategy is that we could take advantage of training errors to successively refine classification model on the training data. Furthermore, our technique is very simple and flexible, which requires nothing more than one classification method.

Since KNN uses all training documents to predict labels of test documents, we can take all training documents within one class as class-representative of that category. If one training example d_0 labeled as A is misclassified into the class B, then our technique 'drag' the class-representative (all examples in class A) A_R to the example, and 'push' the class-representative B_R against the example. That is to say, DragPushing increases the similarities of all or most examples in class A to the example d_0 while reduces the similarities of all or most examples in class B to the example d. Obviously, after the DragPushing, the correct class A tends to put more examples in K-nearest-neighbor set and these nearest neighbors share larger similarities with test document d, and vice versa. Consequently, after a few times of DragPushing operation, the document d_0 will be more likely to be correctly classified by the refined KNN classifier.

Extensive experiments conducted on three benchmark document collections show that the DragPushing Strategy achieves a significant improvement for KNN and still shares the excellent properties of KNN, i.e. simplicity and efficiency.

The rest of this paper is constructed as follows. Section 2 reviews related work for performance improvement on text classifiers. Section 3 describes the traditional KNN classifier. Application of DragPushing Strategy to KNN is introduced in the Section 4. Experimental results are given in Section 5. Finally, Section 6 concludes the paper.

2. Related work

Many researches to improve the performance of text classifier, by alleviating the problem of model misfits, have been conducted in information retrieval community. One of the popular frameworks is meta-learning or classifier committees (Larkey & Croft, 1996), which are based on the idea that, given a task that requires expert knowledge to perform, k experts may be better than one if their individual judgments are appropriately combined. In text categorization community, the scheme is to apply k different classifiers to the same classification task and then combine their predictions appropriately. A classifier committee is characterized by two specialties: a choice of k classifiers and a choice of a combination function. Larkey's experiment (Larkey & Croft, 1996) on classifier committees seemed to provide strong evidence to the statement that classifier committees can somewhat profits from the complementary strengths of their individual members.

As compared to classifier committees, our strategy does not need to train multiple different classifiers on total training data. Instead, our method merely refines a classifier using the misclassified examples of the training data. Consequently, the CPU time consumed by our technique is much less than classifier committees.

Another popular framework is Voting method (Sebastiani, 2002; Kjersti Aas & Line Eikvil), which combines the predictions of multiple same classifiers to boost classification accuracy. This process is often denoted as Voting. Voting algorithm takes a classifier and training set as input and trains the classifier multiple times on different versions of the training set. The generated classifiers are then unified to create a final classifier that is used to categorize the test set. Voting algorithms can be divided into two types: Bagging and Boosting. The main difference between the two types is the way the different versions of the training set are created. Bagging use a uniform probability to select a new training set while Boosting according to how often one example was misclassified by previous classifiers to select one example to create a new training set.

Compared with Voting, our strategy does not need to train multiple same classifiers on total training data. Instead, our method merely refines a classifier using the misclassified examples of the training data. Therefore, the training and prediction of Voting method is much slower than DragPushing.

Error-Correcting Output coding (ECOC) is also a form of combination of multiple classifiers (Rayid Ghani). The ECOC method is borrowed from data transmitting task in communication. Its main idea is to add redundancy to the data being learned (transmitted) so that even if some errors occur due to the biases (noises) in the learning process (channel), the data can be correctly classified (received) in prediction stage (at the other end). It works by converting a multi-class supervised learning problem into a large number (L) of two-class supervised learning problems (Rayid Ghani). Any learning algorithm that can handle two-class learning problems, such as Naïve Bayes (Sebastiani, 2002), can then be applied to learn each of these L problems. L can then be thought of as the length of the codewords with one bit in each codeword for each classifier.

Our technique is different significantly from ECOC for our approach needs no converting of a multi-class problem into multiple two-class problems and combination of multiple binary classifiers. Like Voting scheme, the training and prediction of ECOC is also much slower than DragPushing.

3. The KNN classifier

To classify an unknown document d_0, the KNN classifier ranks the document's neighbors among the training documents, and use the class labels of k most similarity neighbors to predict the class of the input document. To measure the similarity efficiently, we make use of the cosine distance as follows:

$$\mathrm{Sim}(\vec{d}_1,\vec{d}_2)=\frac{\vec{d}_1\bullet\vec{d}_2}{\|\vec{d}_1\|_2\|\vec{d}_2\|_2}=\frac{\sum_{l=1}^{V}d_{1l}\times d_{2l}}{\sqrt{\sum_{l=1}^{V}d_{1l}^2}\sqrt{\sum_{l=1}^{V}d_{2l}^2}} \tag{1}$$

where V denotes the dimension size of document vector $\vec{d}_1$, $\vec{d}_2$.

The classes of these neighbors are weighted using the similarity of each neighbor to d_0 as follows:

$$\mathrm{score}(\vec{d}_0,C_i)=\sum_{\vec{d_j}\in KNN(\vec{d_0})}\mathrm{Sim}(\vec{d}_0,\vec{d}_j)\delta(\vec{d}_j,C_i) \tag{2}$$

where KNN($\vec{d}$) indicates the set of K-nearest neighbors of document $\vec{d}_0$. $\delta(\vec{d}_j,C_i)$ stands for the classification for document $\vec{d}_j$ with respect to class C_i, that is,

$$\delta(\vec{d_j},C_i)=\begin{cases}1 & \vec{d_j}\in C_i\\ 0 & \vec{d_j}\notin C_i\end{cases} \tag{3}$$

Consequently, the decision rule in KNN classification can be written as:

$$\begin{aligned}C&=\arg\max_{c_i}(\mathrm{score}(\vec{d}_0,C_i))\\&=\arg\max_{c_i}\left(\sum_{\vec{d}_j\in \mathrm{KNN}(\vec{d}_0)}\mathrm{Sim}(\vec{d}_0,\vec{d}_j)\delta(\vec{d}_j,C_i)\right)\end{aligned} \tag{4}$$

KNN is a lazy learning instance-based method that does not have an off-line training phase. The main computation is the on-line scoring of training documents given a test document to find the k nearest neighbors. We use D and T to stand for the size of training corpus and test corpus, respectively. The computation of similarities of d_0 to all documents in the training corpus can be done in $O(DV)$. And the sorting of the D similarities takes $O(\mathrm{D}\log(D))$. Accordingly the total running time is $O(T(D\log(D)+DV))$.

4. DragPushing strategy based KNN classifier

4.1. The motivation

As discussed in Section 1, the KNN Classifier itself often introduces inductive biases for it takes the assumption that training data are equally distributed among all categories. If the supposition is violated by unbalance of training data, the KNN classifier often delivers quite poorer performance. Consequently, we can take the inductive biases introduced by KNN as the main factor leading to higher error rate, including training error rate and test error rate. Accordingly, very intuitively and straightforward, we could make use of training errors to refine the KNN classifier by 'dragging' and 'pushing' operation, as we called 'DragPushing Strategy'.

In order to execute the 'dragging' and 'pushing' operation on class-representatives, we introduced a weight vector W_i for each class C_i with equivalent size as total vocabulary V. Note that we initialize $W_{il}=1$, which indicates that no 'dragging' and 'pushing' operation is performed. Then we can derive the modified similarity measurement formula as follows:

$$\mathrm{Sim}(\vec{d}_1,\vec{d}_2,\vec{W}_i)=\frac{\sum_{l=1}^{V}w_{il}\times d_{1l}\times d_{2l}}{\sqrt{\sum_{l=1}^{V}d_{1l}^2}\sqrt{\sum_{l=1}^{V}d_{2l}^2}} \tag{5}$$

In the same way, we can amend the class score formula as follows:

$$\mathrm{score}(\vec{d}_0,C_i,\vec{W}_i)=\sum_{\vec{d}_j\in \mathrm{KNN}(\vec{d}_0)}\mathrm{Sim}(\vec{d}_0,\vec{d}_j,\vec{W}_i)\delta(\vec{d}_j,C_i) \tag{6}$$

Given a training set consisting of only two categories, i.e. the class A and the class B. d_0 stands for the document from category A. If score(d_0,A,W_A) calculated by Eq. (6) is smaller than score(d_0,B,W_B), then we misclassify document d_0 into the class B. Consequently, we utilize 'drag' to enlarge the weight vector W_A, and use 'push' to reduce the weight vector W_B. Accordingly after a few times of executing 'DragPushing' operation score(d_0,A,W_A) has the tendency to be bigger than the score(d_0,B,W_B) and the refined classifier can be more likely to correctly classify the document d. This is the refinement mechanism of DragPushing strategy for the KNN Classifier.

Now we give the reason why DPSKNN can overcome the problem of imbalance of text corpus. If we take class A as minority category and class B as majority category, then according to traditional KNN decision rule, the examples in category A tends to be classified into class B. As a result, the weight vector W_A has more times of 'Drag' operation than W_B. After a few rounds of 'DragPushing' operation, the minority category A tends to have much larger weight vector than majority category B. Consequently, the different weight vector associated with each category could counteract the impact of unbalance of training corpus to a high degree.

4.2. DragPushing strategy based KNN classifier

4.2.1. Initialization

To start, we need to load the training data and parameters including max-iteration-step, drag_weight and push_ weight. In order to guarantee the validity and rationality of each change of the weight W_i for each class C_i, we introduce two centroids: the unaltered sum centroid C_i^u that keeps unchanged in the training phase and the altering sum centroid C_i^a that updates itself with each 'dragging' and 'pushing' operation. We calculate the unaltered sum centroid C_i^u as following formula:

$$\vec{C}_i^u = \sum_{d \in c_i} \vec{d} \tag{7}$$

and initialize C_i^a as C_i^u, i.e. $C_{il}^{a,0} = C_{il}^u$. As discussed above, we initialize $W_{il}^0 = 1$. Note that '0' denotes current iteration-step, i.e. the *0*th iteration-step.

4.2.2. DragPushing

In each iteration, we need to classify all training documents. If one document d_0 labeled as class A is classified into class B, DragPushing is used to adjust $C_A^{a,0}$, $C_B^{a,0}$, W_A^0 and W_B^0 by following formulas:

$$C_{A,l}^{a,0+1} = C_{A,l}^{a,0} + \text{drag\_weight} \times d_{0l} \quad \text{if } d_{0l} > 0 \tag{8}$$

$$W_{A,l}^{0+1} = \frac{C_{A,l}^{a,0+1}}{C_{A,l}^u} \quad \text{if } d_{0l} > 0 \tag{9}$$

$$C_{B,l}^{a,0+1} = \begin{cases} C_{B,l}^{a,0} - \text{push\_weight} \times d_{0l} & \text{push\_weight} \times d_{0l} < C_{B,l}^{a,0} \\ 0 & \text{push\_weight} \times d_{0l} \geq C_{B,l}^{a,0} \end{cases} \tag{10}$$

$$W_{B,l}^{o+1} = \frac{C_{B,l}^{a,o+1}}{C_{B,l}^u} \text{if } d_{0l} > 0 \tag{11}$$

It is worth mentioning that we call the formulas (8) and (9) as 'drag' formulas and (10) and (11) as 'push' formulas. Obviously after the 'Drag' and 'Push' operation, all or most of elements in the weight vector W_A will be increased while all or most of elements in the weight vector W_B will be decreased. As a result, the similarities of all or most documents in the class A to document d_0 will be increased and the similarities of all or most documents in the class B to document d_0 will be decreased. Accordingly, score(d,A,W_A) involved with document d_0 and the class A will be enlarged while score(d,B,W_B) related to document d_0 and the class B will be reduced.

The weights, i.e. drag_weight and push_weight, are used to control the step-size of 'drag' and 'push'. It is worth mentioning that in our experiments, if we set both drag_weight and push_weight to 1.0, DragPushing could consistently achieve relatively stable significant performance improvement on the KNN classifier.

4.2.3. Termination

In practice, it is rather hard to make a decision how many iteration steps should be run to refine the KNN Classifier to be the best. But in accordance with our practice, if we set max-iteration-step to 5, we could always obtain a high-performance refined classifier (Fig. 1).

4.2.4. Time requirements

Assume that there are D training documents and T testing documents. The number of total words is V and Max-Iteration-Step is fixed as M. For one iteration of DragPushing phase, we need to classify D training documents and for each misclassified document we need to update weight vectors of two categories; therefore, the running time is $O(D(DV+D\log(D)+4V))$, i.e. $O(D(DV+D\log(D)))$. Consequently, the training of DPSKNN can be done in $O(MD(DV+D\log(D)))$. Since the final classifier obtained by DragPushing still consists of all training examples, the test time complexity for DPSKNN is the same as the KNN Classifier, i.e. $O(T(DV+D\log(D)))$.

```
Load Training Data;
Calculate $C^u_{il}$ and $C^{a,0}_{il}$ and initialize $W^0_{il}$=1 for each class $C_i$;
For iter=1 to Max-Iteration-Step Do
    For each document $d_0$ in training set Do
        Classify $d_0$ labeled "$A_1$" into class "$A_2$";
        If ($A_1$!=$A_2$) Do
            Drag the class-representative of $A_1$ to $d_0$;
            Push the class-representative of $A_2$ against $d_0$;
        End If
    End For
End For
```

Fig. 1. The outline of dragpushing strategy based KNN classifier.

5. Experiment results

5.1. The datasets

In our experiment, we use three corpora: Reuter-21578[1], Industry Sector[2] and TDT-5[3].

5.1.1. Reuter-21578

The Reuters-21578 text categorization test collection contains documents collected from the Reuters newswire in 1987. It is a standard text categorization benchmark and contains 135 categories. We used its subset: one consisting of 92 categories and in total 10,346 documents.

5.1.2. Sector-48

The Industry Section dataset is based on the data made available by Market Guide, Inc. (www.marketguide.com). The set consists of company homepages that are categorized in a hierarchy of industry sectors, but we disregarded the hierarchy. There were 9637 documents in the dataset, which were divided into 105 classes. We use a subset called as Sector-48 consisting of 48 categories and in all 4581 documents.

[1] Available at http://www.research.att.com/~lewis/reuters21578.html.
[2] Downloadable at http://www-2.cs.cmu.edu/afs/cs/project/theo-11/www/wwkb.
[3] http://www.ldc.upenn.edu/Projects/TDT5/.

5.1.3. TDT-5

TDT-5 is the NIST Topic Detection and Tracking text corpus version 1.1 released in September 10, 2004. This corpus contains news data collected daily from news sources in three languages (American English, Mandarin Chinese and Arabic), over a period of 6 months (April 1–September 30 in 2003). The documents were manually annotated using 250 target topics, approximately 25% of the topics are monolingual English (ENG), 25% are monolingual Mandarin Chinese (MAN), 25% are monolingual Arabic (ARB), and 25% are multilingual. We selected the English documents having annotated topics. The resulting dataset contains 126 categories and in total 6364 documents.

5.2. The performance measure

To evaluate a text classification system, we use the F_1 measure that combines recall and precision in the following way:

$$\text{Recall} = \frac{\text{number of correct positive predictions}}{\text{number of positive examples}} \tag{12}$$

$$\text{Precision} = \frac{\text{number of correct positive predictions}}{\text{number of positive predictions}} \tag{13}$$

$$F_1 = \frac{2 \times \text{Recall} \times \text{Precision}}{(\text{Recall} + \text{Precision})} \tag{14}$$

For ease of comparison, we summarize the F_1 scores over the different categories using the Micro- and Macro-averages of F_1 scores:

$$\text{Micro-}F_1 = F_1 \text{ over categories and documents} \tag{15}$$

$$\text{Macro-}F_1 = \text{average of within-category } F_1 \text{ values} \tag{16}$$

The Micro- and Macro-F_1 emphasize the performance of the system on common and rare categories, respectively. Using these averages, we can observe the effect of different kinds of data on a text classification system (Kian Ming Adam Chai, Hwee Tou Ng, & Hai Leong Chieu, 2002).

5.3. Experiment design

In all our experiments, we adopt three-fold cross-validation. We split each dataset into three parts. Then we use two parts for training and the remaining third for test. We conduct the training-test procedure three times and use the average of the three performances as final result.

We employed Information Gain as feature selection method for it consistently performs well in most cases (Yang & Pedersen, 1997). Algorithms are coded in C++ and running on a Pentium-4 machine with single 2.0 GHz CPU.

Except for C4.5 and NB, we utilize TFIDF other than binary word occurrences as input features. The formula for calculating TFIDF can be written as follows:

$$W(t,\vec{d}) = \frac{tf(t,\vec{d}) \times \log(D/n_t + 0.01)}{\sqrt{\sum_{t\in\vec{d}}[tf(t,\vec{d}) \times \log(D/n_t + 0.01)]^2}} \tag{17}$$

where D is the total number of training documents, and n_t is the number of documents containing the word t. $tf(t,\vec{d})$ indicates the occurrences of word t in document $\vec{d}$.

In our experiments, we only run Balanced Winnow for it consistently yields better performance than Positive Winnow (van Mun). The Balanced Winnow keeps two weights for each feature l in category i, w_{il}^{+} and w_{il}^{-}. The initial values for Balanced Winnow are $w_{il}^{+}=2.0$ and $w_{il}^{-}=1.0$. The initial threshold was set to 1.0. The promotion parameter α and the demotion β (learning rates) were fixed as 1.2 and 0.8, respectively. In our experiments we train Winnow for 40 rounds over the training data.

It is worth noticing that except for Winnow we do not introduce any thresholds investigated by Yang (2001) because the adjustment of thresholds may incur significant computational costs.

For the Centroid Classifier, we compute its normalized centroid using following equation:

$$\vec{C}^N = \frac{\sum_{d\in C_i}\vec{d}}{\|\sum_{d\in C_i}\vec{d}\|_2} \tag{18}$$

For decision tree, we use Quinlan's source code C4.5 Release 8.[4] All parameters are left as default.

In our experiments, we adopt multi-variate Bernoulli event model that has been used for text classification by numerous people (Kalt, 1996). We estimate the word probability using the following formula:

$$p(t_l|c_i,\theta) = \frac{N_{il}+\alpha_1}{N_i+\alpha_2} \tag{19}$$

N_{il} is the number of documents in class i with word t_l. In our experiments reported below, we set $\alpha_1=0.0001$ and $\alpha_2=0.0002$.

5.4. Comparison and analysis

Now we present and discuss the experimental results. Here we compare DPSKNN against KNN, Centroid, Winnow, NB and C4.5 on three text corpora.

Tables 1 and 2 show the best-performance comparison in MicroF1 and MacroF1. We set $k=7$ for KNN and $k=60$ for DPSKNN. Note that for DPSKNN, Max-Iteration-Step is set to 5, Drag-Weight and Push-Weight are both fixed as 1.0. DPSKNN outperforms all other methods except for Centroid. DPSKNN beats KNN by a wide margin on all corpora.

On Reuter-21578, the MicroF1 of DPSKNN is 84.97%, which is approximately 14% higher than that of C4.5, 9%

[4] Downloadable at http://www.rulequest.com/Personal/c4.5r8.tar.gz.

Table 1
The best MicroF1 of different methods on three corpora

	DPSKNN	KNN	Centroid	Winnow	NB	C4.5
Reuter-21578	0.8497	0.8218	0.7818	0.8263	0.7582	0.7131
Sector-48	0.8544	0.8188	0.8055	0.7966	0.8184	0.6523
TDT-5	0.9251	0.9136	0.8877	0.8895	0.8801	0.8226

Table 2
The best MacroF1 of different methods on three corpora

	DPSKNN	KNN	Centroid	Winnow	NB	C4.5
Reuter-21578	0.5530	0.5089	0.5617	0.4891	0.3897	0.3256
Sector-48	0.8585	0.8235	0.8152	0.8349	0.8278	0.6570
TDT-5	0.7627	0.7214	0.7267	0.7042	0.7363	0.5788

higher than that of NB, 7% higher than that of Centroid and 3% higher than that of KNN, and 2% higher than that of Winnow. On Sector-48, the MicroF1 of DPSKNN beats C4.5 by about 20%, Winnow by 6%, Centroid by 5%, both KNN and NB by approximately 4%. In a word DPSKNN yields top-notch performance among these algorithms. Consequently, we can say that DPSKNN is a competitive algorithm in text classification.

Figs. 2–4 display the MicroF1 and MacroF1 curves for different classification methods after term selection using Information Gain on three text collections. We set $k=7$ for KNN and $k=60$ for DPSKNN. Note that for DPSKNN,

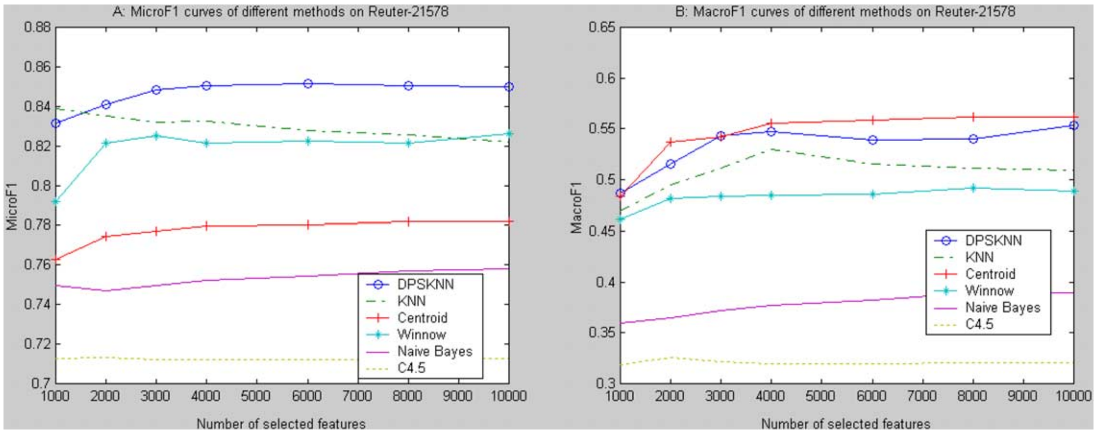

Fig. 2. Performance curves of different methods on Reuter-21578.

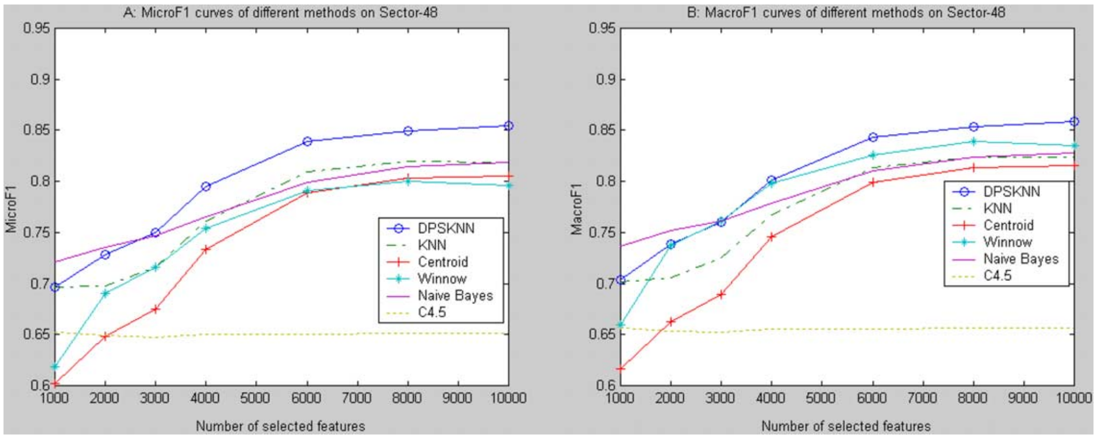

Fig. 3. Performance curves of different methods on Sector-48.

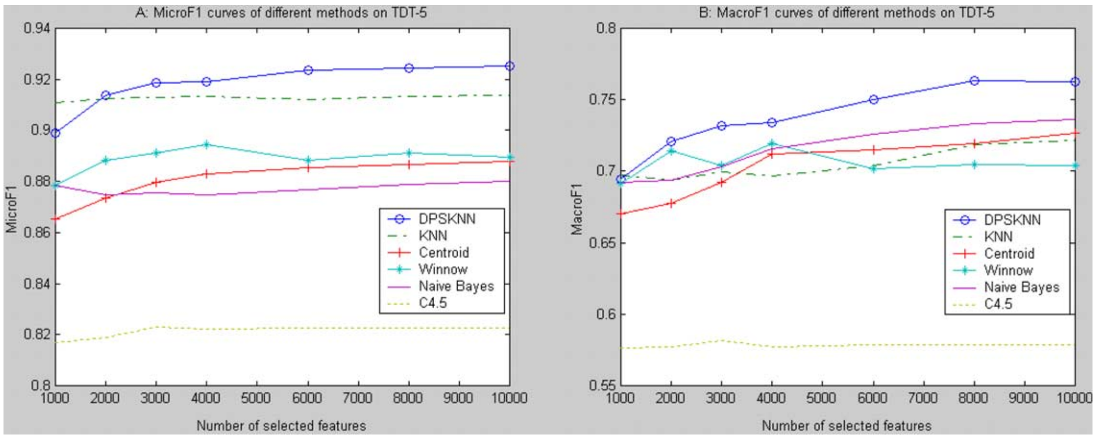

Fig. 4. Performance curves of different methods on TDT-5.

Max-Iteration-Step is set to 5, Drag-Weight and Push-Weight are both fixed as 1.0. On all three corpora, DPSKNN exceeds all the other five methods under almost all feature numbers except for Reuter-21578. We can see an observation that C4.5 yields the worst performance on three corpora when feature number is bigger than 2000 but it is insensitive to the feature number.

Fig. 5 shows the performance of KNN and DPSKNN vs. the nearest neighbors k. Note that the feature number is set to 10,000, Max-Iteration-Step is set to 5, Drag-Weight and Push-Weight are both fixed as 1.0. With the increase of k, KNN delivers worse and worse results on both MicroF1 and MacroF1. Consequently, in our experiments, we set $k=7$ for KNN. On the contrary, DPSKNN yields better and better

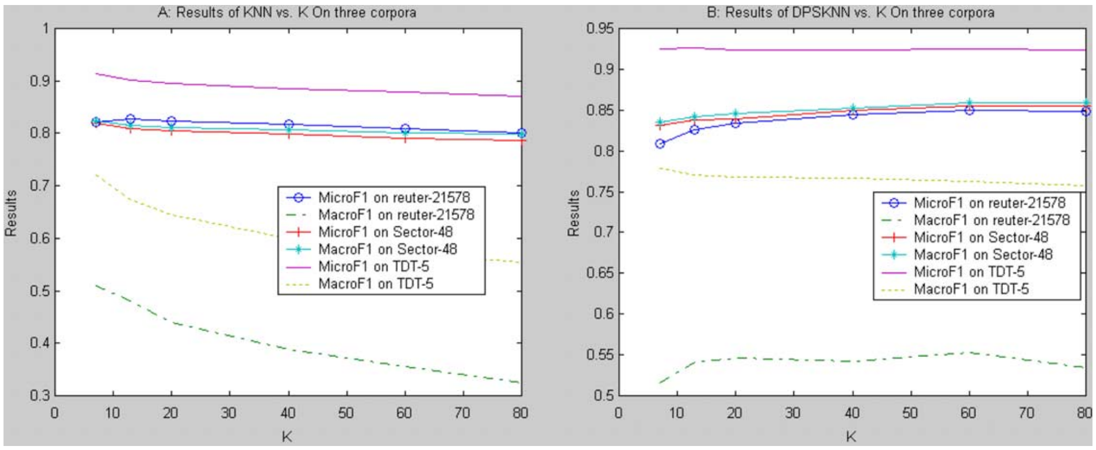

Fig. 5. Performance curves of KNN and DPSKNN vs. k on three corpora.

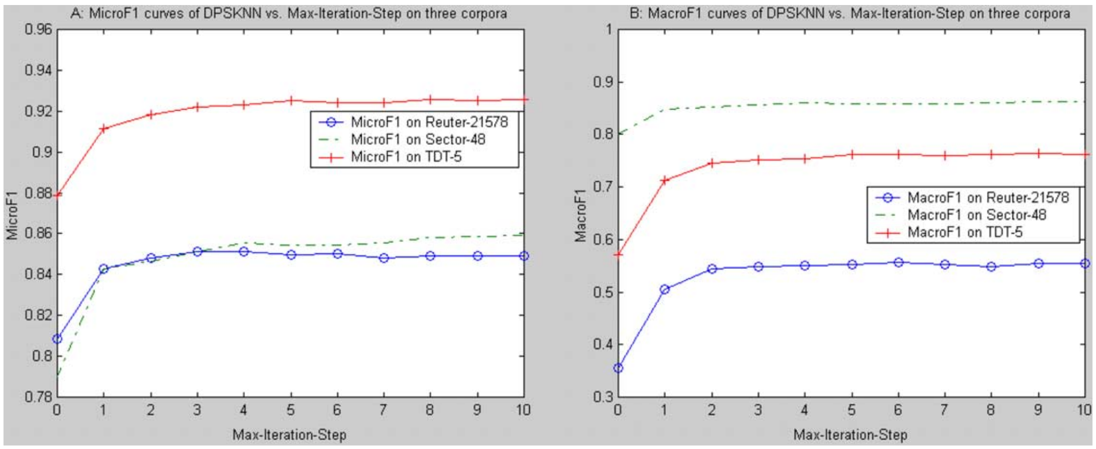

Fig. 6. Performance curves of DPSKNN vs. Max-Iteration-Step on three corpora.

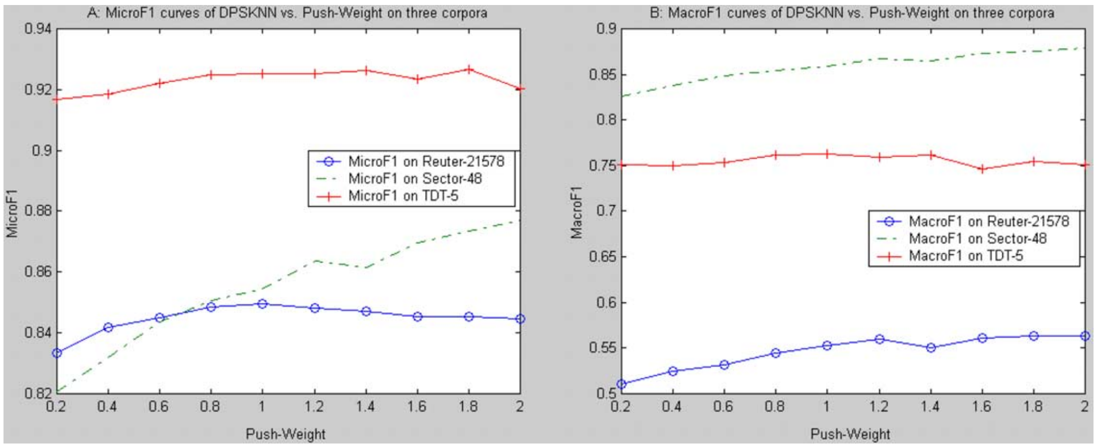

Fig. 7. Performance curves of DPSKNN vs. Push-Weight on three corpora.

results for Reuter-21578 and Sector-48, and the peaks of the four curves appear around 60. For TDT-5, the number k of nearest neighbors hardly exerts influence on MicroF1 and results in a bit decrease on MacroF1. Accordingly, in order to obtain more stable results for all corpora, we fix $k=60$ for DPSKNN.

Fig. 6 displays the performance comparison of DPSKNN vs. different Max-Iteration-Step on three corpora. Note that the feature number takes 10,000 and nearest neighbors k takes 60. Drag-Weight and Push-Weight are both fixed as 1.0. Max-Iteration-Step taking 0 means that DPSKNN uses no DragPushing operation at all, namely, the KNN Classifier. Form the two figures, we can see a wide margin improvement is achieved by running only one round of DragPushing operation. DPSKNN produces the best results around 4 for Reuter-21578, around 5 for TDT-5 and around 10 for Sector-48. More over, the Max-Iteration-Step bigger than 3 hardly makes a difference for categorization quality. Consequently, the empirical optimal value for Max-Iteration-Step approaches 5.

Fig. 7 illustrates the performance comparison of DPSKNN using different Push-Weight on three corpora. Note that the feature number takes 10,000 and nearest neighbors k takes 60. Drag-Weight is fixed as 1.0. From the two figures we could observe that the MicroF1 curve of Reuter-21578 begins to decrease after 1.0 and the MacroF1 curve of TDT-5 also begins to descend after 0.8, while the other curves consistently heighten with the increase of Push-Weight. Consequently, the relatively stable value for Push-Weight is about 1.0.

Fig. 8 demonstrates the performance comparison of DragPushing using different Drag-Weight on three corpora. Note that the feature number takes 10,000 and nearest neighbors k takes 60. Push-Weight is set to 1.0. The peak value of DPSKNN occurs near 0.2 for Sector-48, near 0.6 for Reuter-21578 and near 0.8 for TDT-5. Unlike Push-Weight the stable value for Drag-Weight ranges from 0.2 to 0.8.

6. Conclusion remarks

In this paper we proposed an effective refinement strategy, called DragPushing Strategy, for the KNN classifier. Our technique does not need to generate sophisticated models but only requires simple statistical data and the traditional KNN classifier.

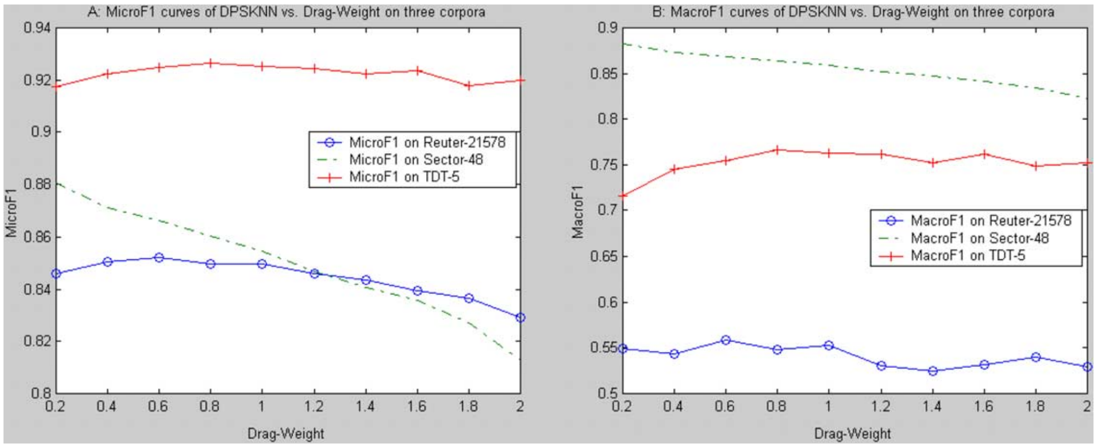

Fig. 8. Performance curves of DPSKNN vs. Drag-Weight on three corpora.

The experiments on three benchmark evaluation collections showed that DPSKNN could make a significant difference on the performance of the KNN Classifier and delivered better performance than other five commonly used methods.

The results reported here are not necessarily the best that can be achieved. Our feature effort is to seek new techniques to enhance the performance of DPSKNN and to apply DragPushing to other classifiers.

References

Sebastiani, Fabrizio (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47.

Han, E., & Karypis, G. (2000). *Centroid-based document classification analysis and experimental result PKDD 2000.*

Kjersti Aas, Line Eikvil. Text ctegorisation: A survey. http://citeseer.ist.psu.edu/aas99text.html.

Larkey, L. S., & Croft, W. B. (1996). *Combining classifiers in text categorization SIGIR* pp. 289–297.

Chai, Kian Ming Adam, Ng, Hwee Tou, & Chieu, Hai Leong (2002). *Bayesian online classifiers for text classification and filtering SIGIR* pp. 97–104.

P.P.T.M. van Mun. Text Classification in information retrieval using winnow. http://citeseer.ist.psu.edu/cs.

Yang, Y., & Pedersen, Jan O. (1997). *A comparative study on feature selection in text categorization ICML* pp. 412–420.

T. Kalt and W.B. Croft. A new probabilistic model of text classification and retrieval. Technical Report IR-78. University of Massachusetts Center for Intelligent Information Retrieval. 1996 http://ciir.cs.umass.edu/publications/index.shtml.

Rayid Ghani. Using error-correcting codes for text classification. http://citeseer.ist.psu.edu/ghani00using.html.

Yang, Y. (2001). *A study on thresholding strategies for text categorization SIGIR* pp. 137–145.