

A Novel Refinement Approach for Text Categorization

Songbo Tan^{1,2}, Xueqi Cheng¹, Moustafa M. Ghanem³, Bin Wang¹, Hongbo Xu¹

¹ Software Department, ICT, P.O. Box 2704, Beijing, 100080, CHINA

² Graduate School of the Chinese Academy of Sciences, 100080, CHINA

³ Dep. Of Computing, Imperial College London, 180 Queens Gate, London SW7 2BZ, UK
tansongbo@software.ict.ac.cn, cxq@ict.ac.cn, mmg@doc.ic.ac.uk, {wangbin, hbxu}@ict.ac.cn

ABSTRACT

In this paper we present a novel strategy, DragPushing, for improving the performance of text classifiers. The strategy is generic and takes advantage of training errors to successively refine the classification model of a base classifier. We describe how it is applied to generate two new classification algorithms; a Refined Centroid Classifier and a Refined Naïve Bayes Classifier. We present an extensive experimental evaluation of both algorithms on three English collections and one Chinese corpus. The results indicate that in each case, the refined classifiers achieve significant performance improvement over the base classifiers used. Furthermore, the performance of the Refined Centroid Classifier implemented is comparable, if not better, to that of state-of-the-art support vector machine (SVM)-based classifier, but offers a much lower computational cost.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval-search process; I.2 [Artificial Intelligence]: Learning; I.5 [Pattern Recognition]: Applications

General Terms

Algorithms, Performance, Experimentation

Keywords

Text Classification, Information Retrieval, Machine Learning

1. INTRODUCTION

With the ever-increasing volume of text data from Internet, it is an important task to categorize these documents into manageable categories. Text categorization aims at automatically placing pre-defined labels on previously unseen documents. It is an active research area in information retrieval, machine learning and natural language processing.

A number of machine learning algorithms have been introduced to deal with text classification, such as K-Nearest Neighbor (KNN) [1], Centroid Classifier [2], Rocchio [6], Naive

Bayes [5], Decision Trees [6], Winnow [10-11], Perceptron [10-11] and Support Vector Machines (SVM) [6].

In such classifiers, documents are first represented using the vector space model (VSM), where each document is considered to be a vector in a feature space. Thus, given a set of N training documents, $\{d_1, \dots, d_N\}$, the common approach starts by first constructing a feature vector table such as that shown in Table 1 where each document is represented by a score (w_{ij}) in relation to each of M chosen features. In the simplest case, such features are chosen to be the set of all words appearing in all documents, and the score is assigned based on the relative frequency of appearance of each term in each document. The table is then used as input to any traditional classification algorithm to generate a classification model. To classify an unseen document, a feature vector is constructed using the same set of M features and then passed as input to the classification model.

Table 1: A traditional feature vector table

D	F ₁	F ₂	F _M	Class
d ₁	w ₁₂	w ₁₃	...				w _{1M}	A
d ₂	w ₂₁	...					w _{2M}	B
...	
d _i	w _{i1}	...		w _{ij}			w _{iM}	B
...	
d _N	w _{N1}	...					w _{NM}	A

For example, using such a pre-labeled feature vector table, a lazy KNN classifier simply keeps the table in memory. It assigns an unseen document to the majority class of its k most similar documents in the table. On the other hand, a Centroid Classifier calculates a prototype vector (centroid) for each of the available document classes as an average vector of all the row entries belonging to that class in the training table. It then determines the class of an unseen document simply by assigning it to the class with the most similar centroid. An SVM classifier uses the table to find a small number of vectors (known as support vectors) that define a hyper-surface that separates between the different classes. It assigns the class of an unseen document in relationship to such a hyper-surface, etc.

The overall classification accuracy of such machine learning classifiers often suffers from what is known as an inductive bias or model misfit [9]. When the nature of the data fits the assumptions of the underlying classification strategy well, the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'05, October 31–November 5, 2005, Bremen, Germany.
Copyright 2005 ACM 1-59593-140-6/05/0010 ...\$5.00.

classification accuracy can be very high, and vice versa. For example, the Centroid Classifier is based on the assumption that a given document should be assigned a particular class if the similarity of this document to the centroid of its true class is the largest. Nevertheless, this supposition is often violated when there exists a document from class A sharing more similarity with the centroid of class B than that of class A .

Model misfits may occur for many reasons, including the choice of type of features used in the model (e.g. choice of terms vs. regular expressions, or choice of either from whole document or from restricted sections), choice of the number of features to use, the choice of scoring methods used in the vector space model (e.g. feature count vs. tf-idf method), the choice of similarity function (Euclidean Distance vs. cosine similarity), etc. Such factors may also compound, and the more serious the model misfit, the poorer the classification performance will be.

Whereas, for individual domains (e.g. text collections), the choice of the best classifier and best set of parameters can be found through tedious experimentation, a generic approach for addressing model misfits is typically needed. Numerous researchers have thus investigated the development of generic methods that can be used to improve the performance of base text classifiers automatically. These methods include Voting [8] (consisting of Bagging and Boosting), and Wu’s method [9].

In this paper, we propose an effective and yet efficient refinement strategy, “DragPushing”, to enhance the performance of text classifiers by means of on-line modification of the base classifier models. The main idea behind our strategy is to take advantage of training errors to successively refine classification model. The technique is simple to implement, flexible and yields outstanding results.

We have applied “DragPushing” to a Centroid Classifier and a Naïve Bayes Classifier to generate two new classification algorithms; a Refined Centroid Classifier (RCC) and a Refined Naïve Bayes Classifier (RNB). The results indicate that in each case, the refined classifiers achieve significant performance improvement over the base classifiers used. Furthermore, the performance of RCC implemented is comparable, if not better, to that of a state-of-the-art support vector machine (SVM)-based classifier, but offers a much lower computational cost.

The remainder of this paper is organized as follows: In Section 2 we describe the two base classifiers used (Centroid and Naïve Bayes). In Section 3, we describe the DragPushing strategy and its application to both classifiers. In Section 4, we present an extensive experimental evaluation of our classifiers. Finally, section 5 we present our summary, concluding remarks and directions of our future work.

2. TWO BASE CLASSIFIERS

2.1 Centroid Classifiers

Centroid Classifiers [2] belong to a class of common classification algorithms that provide a simple and efficient method for automatic document classification. In such classifiers, the entries in the feature vector table are typically, and without loss of generality, represented in a vector space using the words appearing in the document, and where the score of each word is assigned using the tf-idf (term frequency inverse document

frequency) formula (6). This score increases proportionally to the number of times a word appears in the document but is offset by how common the word is in all of the documents in the whole document collection.

Based on such representation, a prototype vector (centroid) is then calculated for each class as the average vector of all documents belonging to that class. The class of an unlabelled document is then determined simply by assigning it to the class with the most similar centroid. The computational complexity of the learning phase of such centroid-based classifiers is linear in the number of documents and the number of terms in the training set.

Mathematically, given K document classes, $\{C_1, \dots, C_K\}$ a document d is assigned a class label using the following rule

$$C = \arg \max_{C_i} (sim(d, C_i)) \quad (1)$$

Without loss of generality, the similarity between the feature vector of document d and each centroid C_i is typically calculated using the inner-product measure as follows:

$$sim(d, C_i) = \vec{d} \cdot \overline{C}_i^N \quad (2)$$

where \overline{C}_i^N is the *normalized centroid* of a class.

The *normalized centroid* is calculated based on the *summed centroid* as described below:

First, for each class C_i , we calculate a *summed centroid* by summing the vectors of the documents belonging to the class.

$$\overline{C}_i^S = \sum_{d \in C_i} \vec{d} \quad (3)$$

Then we compute the *normalized centroid* by dividing the *summed centroid* by its length.

$$\overline{C}_i^N = \frac{\overline{C}_i^S}{\|\overline{C}_i^S\|_2} \quad (5)$$

where $\|z\|_2$ denotes the 2-norm of z .

It is worth mentioning that the inner-product measure is equivalent to cosine measure since the document vector is computed using the tf-idf formula,

$$w_{tfidf}^N(t, \vec{d}) = \frac{tf(t, \vec{d}) \times \log(D/n_t + 0.01)}{\sqrt{\sum_{t \in \vec{d}} [tf(t, \vec{d}) \times \log(D/n_t + 0.01)]^2}} \quad (6)$$

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

2.2 Naïve Bayes Classifiers

The Naïve Bayesian algorithm is another widely used algorithm for document classification. Given a feature vector table, the algorithm computes the posterior probability that the document belongs to different classes and assigns it to the class with the highest posterior probability. There are two different approaches for using Naïve Bayesian approach for text categorization. In this paper, and without loss of generality, we adopt the multi-variate Bernoulli event model adopted by numerous authors, e.g. [5].

In this approach, a document is represented in the feature vector table using binary attributes indicating which word (feature) occurs and does not occur in the document. The number of times a word occurs in a document itself is not captured. When calculating the probability of a document one multiplies the probability of all attribute values, including the probability of nonoccurrence for words that do not occur in a document. In this case, the posterior probability can be calculated as follows:

$$PostP(c_k | d_i, \theta) = \frac{p(c_k | \theta) p(d_i | c_k, \theta)}{p(d_i | \theta)} \quad (8)$$

where θ denotes the parameter of mixture model and $p(c_k | \theta)$ denotes the class prior probability which can be estimated as the Maximum Likelihood Estimate:

$$p(c_k | \theta) = \frac{N_k}{N} \quad (9)$$

where N_k stands for the number of documents in class k , and N is the total number of training documents. For brevity, we use $p(c_k)$ to stand for $p(c_k | \theta)$ in the rest of my paper.

The document probability given its class can be estimated as follows:

$$p(d_i | c_k, \theta) = \prod_{j=1}^W p(t_j | c_k, \theta)^{d_{ij}} (1 - p(t_j | c_k, \theta))^{1-d_{ij}} \quad (10)$$

where W is the number of total words. d_{ij} is the binary value of word j in document i .

In formula (10) the word probability can be estimated as follows:

$$p(t_j | c_k, \theta) = \frac{N_{kj} + \alpha_1}{N_k + \alpha_2} \quad (11)$$

where N_{kj} is the number of documents in class k with word j . In our experiments reported in section 4, we set $\alpha_1 = 0.0001$ and $\alpha_2 = 0.0002$. For the sake of brevity, we substitute $p(t_j | c_k, \theta)$ with $p(t_j | c_k)$.

Consequently, we can illustrate the Bayes' rule for classification decision as follows:

$$C = \arg \max_{c_k} \left(\frac{p(c_k | \theta) p(d_i | c_k, \theta)}{p(d_i | \theta)} \right) \quad (12)$$

3. REFINING BASE CLASSIFIERS

3.1 Overview

The DragPushing refinement strategy for classifiers presented in this paper is based on taking advantage of misclassified examples in the training data to iteratively refine, and adjust the bias, in the model generated by a generic base classifier.

We first present a general overview of the rationale of the method as applied to the Centroid Classifier, followed by a detailed explanation of the formulae for the Centroid Classifier, and then by a detailed explanation of how the method and formulae are adapted easily to the Naïve Bayesian Classifier.

3.2 Rationale

The model bias inherent in the Centroid Classifier is captured within the prototype vectors, or class centroids, that are computed by the classifier. The core assumption is that a given document should be assigned a particular class if the similarity of this document to the centroid of its true class is the largest. Nevertheless, this supposition is often violated when there exists a document from class A sharing more similarity with the centroid of class B than that of class A .

An intuitive and straightforward solution to this problem is to make use of training errors to adjust class centroids so that the biases introduced by Centroid Classifier can be reduced gradually. Using a training data set, our approach thus first calculates the prototype vectors, or centroids, for each of the available document classes. Using misclassified examples, it then iteratively refines these centroids; by dragging the centroid of a correct class towards a misclassified example and in the same time pushing the centroid of an incorrect class away from the misclassified example. This algorithm is outlined in Figure 1.

-
- (1) Load training data and parameters;
 - (2) Calculate C_k^S and C_k^N for each class C_k ;
 - (3) For iter=1 to MaxIteration Do
 - (3.1) For each document d in training set Do
 - (3.1.1) Classify d labeled "A₁" into class "A₂";
 - (3.1.2) If (A₁≠A₂) Do
 - Drag normalized centroid of class A₁ to d ;
 - Push normalized centroid of class A₂ against d ;
-

Figure 1: The Outline of DragPushing for Centroid Classifier

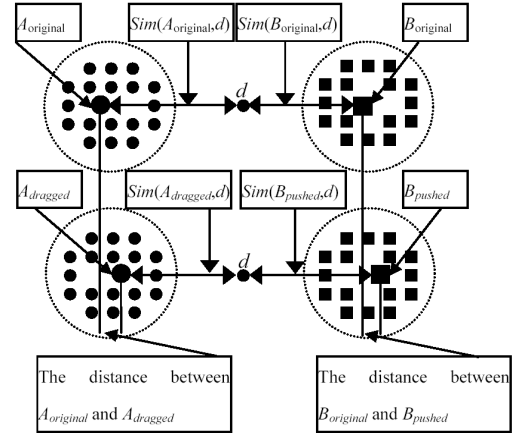


Figure 2: The description of DragPushing strategy

As an example, if one training example d with true class label "A₁" is misclassified into class "A₂", DragPushing "drags" the centroid of class "A₁" to example d , and "pushes" the centroid of class "A₂" away from d . After this operation, document d will be more likely to be correctly classified by the refined classifier.

The operation can be represented graphically as shown in Figure 2 when there are only two document classes (A and B). In the figure, d stands for the document from category A . $A_{original}$

and $B_{original}$ denote the original, normalized centroid of class A and class B respectively; $A_{dragged}$ and B_{pushed} denote the dragged, normalized centroid of class A and the pushed, normalized centroid of class B respectively. If the similarity $Sim(A_{original}, d)$ is smaller than the similarity $Sim(B_{original}, d)$, as illustrated in Figure 2, Centroid Classifier misclassifies document d into the class B .

With the intention of correct classification of example d , DragPushing utilizes “drag” operation to enlarge the similarity (or reduce the distance) between class A and document d , and uses “push” operation to reduce the similarity (or enlarge the distance) between class B and document d . From Figure 2 we can observe that after the “DragPushing” operation the similarity $Sim(A_{dragged}, d)$ is bigger than the $Sim(B_{pushed}, d)$ and the refined classifier can be more likely to correctly classify the document d . This is the refinement mechanism of DragPushing strategy for base classifier, i.e., Centroid Classifier.

3.3 DragPushing for Centroid Classifier

Based on the above overview, the operation of the refinement algorithm applied to centroid classifier can be described more formally as provided below:

Initialization: to start, we need to load training data and parameters including MaxIteration, DragWeight and PushWeight. Then for each category C_k we calculate one *summed centroid* $\overrightarrow{C}_k^{S,0}$ and one *normalized centroid* $\overrightarrow{C}_k^{N,0}$. Note that 0 denotes current iteration-step, i.e., the 0^{th} iteration-step.

DragPushing: in one iteration, we need to categorize all training documents. If one document d labeled as class “ A ” is classified into class “ B ”, DragPushing modifies the *summed* and *normalized centroids* of class “ A ” and class “ B ” by the following formulas:

$$C_{A,l}^{S,o+1} = C_{A,l}^{S,o} + dragweight \times d_l \quad \text{if } d_l > 0 \quad (13)$$

$$C_{A,l}^{N,o+1} = \frac{C_{A,l}^{S,o+1}}{\|C_A^{S,o+1}\|_2} \quad \text{if } C_{A,l}^{S,o} > 0 \quad (14)$$

$$C_{B,l}^{S,o+1} = [C_{B,l}^{S,o} - pushweight \times d_l]_+ \quad \text{if } d_l > 0 \quad (15)$$

$$C_{B,l}^{N,o+1} = \frac{C_{B,l}^{S,o+1}}{\|C_B^{S,o+1}\|_2} \quad \text{if } C_{B,l}^{S,o} > 0 \quad (16)$$

where o denotes the o^{th} iteration-step and l stands for feature index of document vectors and centroid vectors. $[z]_+$ denotes the hinge function which equals to the argument z if $z > 0$ and is zero otherwise, $[z]_+ = \max\{z, 0\}$. The reason for introduction of $[z]_+$ is that we found that nonnegative centroids performs better than real centroids in our experience.

We call formulas (13-14) as “drag” formulas and (15-16) as “push” formulas. Obviously after the execution of “drag” and “push” formulas, the centroid of class “ A ” will share more similarity with document d than that of class “ B ”. In other words, the similarity between document d and the centroid of class “ A ” will be enlarged while the similarity between document d and the centroid of class “ B ” will be reduced.

Time Requirements: Assume that there are D training documents, T test documents, W words in total, K classes and M iteration steps. The time complexity of calculating a *summed centroid* and *normalized centroid* for all classes, i.e., training a Centroid Classifier, is $O(DW+KW)$. Since $K < D$, the time complexity is $O(DW)$. For one iteration of DragPushing stage, we need to categorize D training documents. For each misclassified document we need to update four class centroids, therefore the running time is $O(D(KW+4W))$, i.e., $O(DKW)$ (when $K > 3$). Consequently the training of RCC can be done in $O(DW+MDKW)$, i.e., $O(MDKW)$. As a result, the training time of RCC scales linearly with the training documents (D). Since the final classifier still consists of K centroids, the prediction time required by RCC is the same as Centroid Classifier, i.e., $O(TKW)$. Accordingly RCC is still a linear classifier.

3.4 DragPushing for Naïve Bayes

DragPushing can be easily adapted for Naïve Bayesian classifiers. The main difference in this case, is that there is no notion of a prototype vector for each class. However, there is an equivalent notion of a “class representative” that can be calculated for each class based on document and word counts.

-
- (1) Load training data and parameters;
 - (2) Calculate N_k^0 and $N_{k,j}^0$ for each class C_k ;
 - (3) For iter=1 to MaxIteration Do
 - (3.1) For each document d in training set Do
 - (3.1.1) Classify d labeled “ A_1 ” into class “ A_2 ”;
 - (3.1.2) If ($A_1 \neq A_2$) Do
 - Drag the class-representative of A_1 to d ;
 - Push the class-representative of A_2 against d ;
-

Figure 3: The Outline of DragPushing for Naïve Bayes

The algorithm is shown in Figure 3, and its detailed operation can be described as follows:

Initialization: We obtain N_k^0 by counting the documents in each category C_k and $N_{k,j}^0$ by summing documents containing word j in category k . And then we calculate the word probability $p(t_j|c_k)$ and the class prior probability $p(c_k)$.

DragPushing: If one document d labeled as class “ A ” is misclassified into class “ B ”, DragPushing is employed to refine $N_{A,j}^o$, $N_{B,j}^o$, N_A^o and N_B^o by the following formulas:

$$N_{A,j}^{o+1} = N_{A,j}^o + dragweight \quad \text{if } d_l = 1 \quad (17)$$

$$N_A^{o+1} = N_A^o + dragweight \quad (18)$$

$$N_{B,j}^{o+1} = \begin{cases} N_{B,j}^o - pushweight & \text{if } N_{B,j}^o \geq 2 \text{ and } d_l = 1 \\ N_{B,j}^o & \text{if } N_{B,j}^o < 2 \text{ and } d_l = 1 \end{cases} \quad (19)$$

$$N_B^{o+1} = \begin{cases} N_B^o - pushweight & \text{if } N_B^o \geq 2 \\ N_B^o & \text{if } N_B^o < 2 \end{cases} \quad (20)$$

We then use the refined $N_{A,j}^{o+1}$, $N_{B,j}^{o+1}$, N_A^{o+1} and N_B^{o+1} to calculate new word probability $p(t_j|c_k)$ and class prior probability $p(c_k)$ using equations (11) and (9). Note that we call formulas (17-18) as “drag” formulas and (19-20) as “push” formulas.

Obviously after the “Drag” and “Push” operation, the posterior probability $PostP(A|d)$ involved with document d and class A will be enlarged while the posterior probability $PostP(B|d)$ related to document d and class B will be reduced.

Time Requirements: The time complexity of calculating prior class probability and word probability for all classes, i.e., training a Naïve Bayes Classifier, is $O(D+DW)$, i.e., $O(DW)$. For one iteration of DragPushing phase, we need to classify D training documents and for each misclassified document we need to update two class-representatives (i.e., prior class probability and word probability), therefore the running time is $O(D(KW+2W+2))$, i.e., $O(DKW)$. Consequently the training of RNB can be done in $O(DW+MDKW)$, i.e., $O(MDKW)$. As a result, the training phase of RNB scales linearly with the training documents (D). Since the final classifier still consists of K class-representatives, the testing time for RNB is the same as Naïve Bayes Classifier, i.e., $O(TKW)$. Accordingly RNB is still a linear classifier.

3.5 Comparison to Related Approaches

Many researchers [4][8] have made an attempt to alleviate the problem of model misfits by combining the predictions of multiple classifiers. This process is often denoted as Voting. A 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 combined to create a final classifier that is used to classify 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 uses a uniform probability to select a new training set while Boosting makes use of how often one example was misclassified by previous classifiers.

Compared to Voting, DragPushing has two particularities. First, DragPushing is of on-line refining fashion. Unlike Voting, our technique does not need to retrain the classifier multiple times on the different versions of the entire training set. Consequently our approach consumes much less training time than the Voting method. Second, DragPushing produces only one refined classifier. Hence the prediction is much faster than Voting method.

Wu et al. [9] presented another novel approach to handle the problem of model misfits. Once a prediction error appears on a training set, their technique retrains a sub-classifier using these misclassified training examples of each predicted class with the same learning method. In this way, it forces the classifier to learn from refined regions in the training data, making the model stronger and fit the training data better. As a result, their method leads to a classifier-model tree consisting of a large number of nodes (each node stands for a classifier).

Unlike Wu’s technique, our method does not need to train multiple classifiers on multiple subsets of the entire training set, and instead our strategy produces only one refined classifier. Accordingly, our technique has the superiority over Wu’s technique with regard to the CPU time. Meanwhile, our method does not need to split the training set, consequently the unbalance of the data set exerts less adverse influence on our technique than Wu’s technique.

Both Winnow and Perceptron [10-11] are derived from on-line mistake-driven learning model that takes place in the sequence of trails and they are both used to learn a classifier (or weight vector) w_c for each class c . On one trail, the learner first makes a prediction and then receives feedback, which may be used to update the current weight vector. A mistake-driven algorithm updates its weight vector only when a mistake is made. During the training phase, with a collection of training set, this process is repeated several times by iterating on the training data. Winnow and Perceptron differ by the way they update their weight vectors during the training phase.

In contrast to Winnow or Perceptron, the first difference is that DragPushing starts from the base classifier, e.g., Centroid Classifier or Naïve Bayes, while Perception and Winnow begin with randomly-selected weight vectors; the second distinction is that the training and classification of Perception and Winnow depend upon given thresholds; thirdly, the substantial dissimilarity is that both Perception and Winnow are two-class learning algorithms while DragPushing is base on multi-class learning mode.

4. EXPERIMENT RESULTS

We have conducted an extensive experimental evaluation of the two new classifiers (RCC and RNB) generated by DragPushing approach. We make comparisons to the base classifiers as well as to KNN, Winnow and SVM-based classifiers. In our experiment, we use three English corpora (WebKB¹, Industry Sector², and 20NewsGroup³) and one Chinese collection (TanCorp⁴).

WebKB The WebKB dataset contains Web pages collected from university computer science departments. There are approximately 8,300 documents in the set and they are divided into seven categories. Among the seven categories, student, faculty, course, and project are the four most populous. The subset we use consists only of these four categories and in total 4,199 documents, which is called WebKB4.

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 disregard the hierarchy. There were 9,637 documents in the dataset, which were divided into 105 classes. We use a subset called Sector-48 consisting of 48 categories and 4,581 documents.

20NewsGroup The 20Newsgroup (20NG) dataset contains approximately 20,000 articles evenly divided among 20 Usenet newsgroups. We use a subset consisting of total categories and 19,446 documents.

TanCorp The TanCorp corpus is collected and processed by Songbo Tan. The corpus is categorized in two hierarchies. The first hierarchy contains 12 big categories and the second hierarchy consists of 60 small classes. The total documents

¹ <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/>

² <http://www-2.cs.cmu.edu/afs/cs/project/theo-11/www/wwkb>.

³ <http://www-2.cs.cmu.edu/afs/cs/project/theo-11/www/wwkb>.

⁴ <http://lcc.software.ict.ac.cn/~tansongbo/corpus1.php>.

amount to 14,150. This corpus can serve as three categorization datasets: one hierarchical dataset (TanCorpHier) and two flat datasets (TanCorp-12 and TanCorp-60). In our experiment we use TanCorp-12.

4.1 The Performance Measures

We use the F1 measure introduced by van Rijsbergen [3]. This measure combines recall and precision in the following way:

$$\text{Recall} = \frac{\text{number of correct positive predictions}}{\text{number of positive examples}} \quad (21)$$

$$\text{Precision} = \frac{\text{number of correct positive predictions}}{\text{number of positive predictions}} \quad (22)$$

$$\text{F1} = \frac{2 \times \text{Recall} \times \text{Precision}}{(\text{Recall} + \text{Precision})} \quad (23)$$

For ease of comparison, we summarize the F1 scores over the different categories using the Micro- and Macro-averages of F1 scores:

$$\text{Micro - F1} = \text{F1 over categories and documents} \quad (24)$$

$$\text{Macro - F1} = \text{average of within - category F1 values} \quad (25)$$

MicroF1 and MacroF1 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.

4.2 Experimental Design

We use three-fold cross validation in our experiments by evenly splitting each dataset into three parts and use two parts for training and the remaining third for test. We perform the training-test procedure three times and use the average of the three performances as final result. In all experiments we employ Information Gain as feature selection method since it consistently performs well in most cases [7]. The algorithms are coded in C++ and execute on a Pentium-4 machine with one 1.60 GHz CPU and 512M memory.

For experiments on SVM we employed SVMTorch⁵. We adopted a linear kernel that has been found as competitive as other kernels in Reuter-21578 [1]. All parameters were left at default values. In the case of Winnow, we only run Balanced Winnow because it consistently yields better performance than Positive Winnow [10]. The Balanced Winnow keeps two weights for each feature l in category C_i , w_{il}^+ and w_{il}^- . The weight values are initialized as $w_{il}^+ = 2.0$ and $w_{il}^- = 1.0$ and the 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. We train Balanced Winnow for 40 rounds over the training data. For the sake of brevity, we substitute Balanced Winnow with Winnow. For KNN, we set the neighbor number k to 13. It is worth noticing that except for Winnow and SVM we do not introduce any thresholds investigated by Yang [12] because the adjusting of thresholds may incur significant computational costs.

⁵ www.idiap.ch/~bengio/projects/SVMTorch.html

4.3 Comparison and Analysis

Table 2 and Table 3 show the best-performance comparison in MicroF1 and MacroF1. Note that for RCC and RNB, MaxIteration is set to 8 and 3 respectively, and DragWeight and PushWeight are both fixed as 1.0. RCC and RNB both outperform all the other five methods on WebKB. SVM performs the best on the other corpora. RCC and RNB both achieve a significant improvement over their base classifier respectively on all corpora.

Table 2: The best MicroF1 of different methods

	RCC	Centroid	RNB	NB	KNN	Winnow	SVM
WebKB	0.8964	0.7995	0.9155	0.8645	0.7495	0.8140	0.8926
Sector-48	0.8812	0.8055	0.8450	0.8184	0.8086	0.8003	0.9026
20NG	0.8800	0.8429	0.8541	0.8353	0.8481	0.8105	0.8891
TanCorp-12	0.9381	0.9053	0.9276	0.8937	0.9035	0.8645	0.9483

Table 3: The best MacroF1 of different methods

	RCC	Centroid	RNB	NB	KNN	Winnow	SVM
WebKB	0.8802	0.7847	0.9063	0.8601	0.7015	0.7785	0.8766
Sector-48	0.8836	0.8152	0.8506	0.8278	0.8144	0.8389	0.9049
20NG	0.8769	0.8389	0.8522	0.8355	0.8461	0.8161	0.8876
TanCorp-12	0.9066	0.8632	0.8855	0.8513	0.8478	0.7587	0.9172

On WebKB, the MicroF1 of RCC is 89.64%, which is approximately 15% higher than that of KNN, 10% higher than that of Centroid and 8% higher than that of Winnow. On 20NewsGroup, the performance of SVM is about one percent higher than that of RCC, and three percent higher than that of RNB. In total RCC yields excellent performance approaching SVM. Consequently we can say that RCC and RNB are two efficient and competitive algorithms in text classification.

Table 4: Training time in seconds

	RCC	Centroid	RNB	NB	KNN	Winnow	SVM
WebKB	0.969	0.151	27.943	0.292	0	0.844	25.625
Sector-48	3.980	0.562	364.74	3.172	0	6.281	94.099
20NG	5.214	0.515	641.07	1.286	0	7.959	544.33
TanCorp-12	6.954	0.625	245.86	0.782	0	6.25	321.968

Table 5: Test time in seconds

	RCC	Centroid	RNB	NB	KNN	Winnow	SVM
WebKB	0.011	0.015	4.344	3.359	4.849	0.005	13.787
Sector-48	0.193	0.193	59.318	59.422	6.255	0.250	57.114
20NG	0.109	0.115	104.99	69.016	65.755	0.094	231.32
TanCorp-12	0.125	0.124	50.64	44.343	61.75	0.125	204.328

Table 4 and Table 5 report the training time and test time of seven methods on four text collections. Note that the running time does not include the seconds for loading data from hard disk. Feature number is set to 10,000. For RCC and RNB, MaxIteration is set to 8 and 3 respectively, and DragWeight and PushWeight are both fixed as 1.0. For training phase the CPU time required by SVM is about 20 times larger than that of RCC on WebKB and about 100 times larger than that of RCC on 20NewsGroup. The predicting speed of RCC is as fast as Centroid Classifier and about 100 times faster than SVM and NB.

Surprisingly, the time required by RNB is approximately equivalent to that of SVM. The reason for the large amount of running time required by RNB and NB is that when calculating the probability of a document they need to multiply the probability of all words (10,000), including the probability of words that do not occur in this document. The time saving of RCC is very obvious. In summary, these experiments have shown that RCC offers an alternate choice for text categorization.

Figures 4-7 display the MicroF1 and MacroF1 curves for different classification methods after term selection on four text collections. Note that for RCC and RNB, MaxIteration is set to 8 and 3 respectively, and DragWeight and PushWeight are both fixed as 1.0. On WebKB, RCC and RNB produces the best performance when feature number is smaller than 2,000. On all corpora, RCC, RNB and SVM consistently offer top performance.

Figure 8 displays the training error rate and prediction error rate of RCC and RNB vs. MaxIteration on 20NewsGroup. Feature number takes 1,000. For RCC and RNB, DragWeight and PushWeight are both fixed as 1.0. A very obvious observation can be obtained: with the increase of MaxIteration, the training error rate and prediction error rate of RCC and RNB can both be decreased. This observation validates the efficiency of DragPushing strategy for Centroid Classifier and Naïve Bayes.

Figure 9 illustrates the performance comparison of RCC and RNB vs. MaxIteration on three corpora. Note that feature number takes 1,000 for WebKB and 10,000 for the other two corpora. Both DragWeight and PushWeight take 1.0. MaxIteration taking 0 means that no DragPushing operation is used at all, i.e., Centroid Classifier or Naïve Bayes. From the figure we can observe that a wide margin improvement is achieved by running only one round of DragPushing strategy over training set. RCC produces the best MicroF1 around 7 for three corpora, while RNB yields the best result around 4 for WebKB, 6 for Sector-48, and 10 for 20NewsGroup. Consequently the empirical optimal value of MaxIteration is about 7 for RCC and ranges from 4 to 10 for RNB.

Figure 10 demonstrates the performance comparison of DragPushing vs. PushWeight on three corpora. Note that feature number takes 1,000 for WebKB and 10,000 for the other two corpora. MaxIteration is set to 8 and 3 for RCC and RNB respectively, and DragWeight is fixed as 1.0. For RCC, the peak performance on WebKB occurs around 1.0 and the corresponding peak on the other two corpora occurs near 2.0. For RNB, the performance begins to decrease on 20NewsGroup after 1.0 while it continues to ascend on the other two data sets. Consequently, the relatively stable value for RCC ranges from 1.0 to 2.0, while for RNB it is about 1.0.

Figure 11 illustrates the performance comparison of RCC and RNB vs. DragWeight on three corpora. Note that feature number

takes 1,000 for WebKB and 10,000 for the other two corpora. MaxIteration is set to 8 and 3 for RCC and RNB respectively, and PushWeight is set to 1.0. The performance of RCC on WebKB and Sector-48 begins to descend after 1.0 while on 20NewsGroup it keeps almost unchanged. After 1.0, the performance of RNB on three corpora keeps almost unchanged. Hence the stable value for DragWeight is around 1.0.

5. CONCLUSION REMARKS

In this paper we proposed an efficient and yet effective refinement strategy, i.e., DragPushing, to rectify the biases of text classifiers. We examined the effect of DragPushing strategy for Centroid Classifier and Naïve Bayes. Extensive experiments conducted on four text collections showed that DragPushing could make a significant difference on the performance of the two base classifiers. Surprisingly, RCC delivers top performance approaching state-of-the-art SVM and sometimes exceeding it without incurring significant computational costs. Our future effort is to seek new techniques to enhance the performance of DragPushing and to apply DragPushing to other classifiers.

6. ACKNOWLEDGMENTS

This work was supported in part by the national 973 fundamental research program under grants number 2004CB318109.

We would particularly like to acknowledge the efforts of Dr. Wai Lam who gave us many pertinent advices.

7. REFERENCES

- [1] Y. Yang, X. Lin. A re-examination of text categorization methods. SIGIR. 1999, 42-49
- [2] E. Han and G. Karypis. Centroid-Based Document Classification Analysis & Experimental Result. PKDD 2000
- [3] C. van Rijsbergen. Information Retrieval. Butterworths, London, 1979.
- [4] R. E. Schapire & Y. Singer. Boostexter: A boosting-based system for text categorization. Machine Learning, 2000,39, 135-168.
- [5] A. McCallum & K. Nigam. A Comparison of Event Models for Naive Bayes Text Classification. AAAI/ICML-98 Workshop on Learning for Text Categorization[C]. 1998.
- [6] F. Sebastiani. Machine learning in automated text categorization. ACM Computing Surveys. 2002, 34(1): 1-47
- [7] Y. Yang and J.O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. ICML. 1997, 412-420
- [8] K. Aas & L. Eikvil. Text Categorisation: A Survey. Report NR 941, Norwegian Computing Center, 1999, 15
- [9] H. Wu, T. H. Phang, B. Liu & X Li. A Refinement Approach to Handling Model Misfit in Text Categorization. SIGKDD. 2002, 207-216
- [10] P.P.T.M. van Mun. Text Classification in Information Retrieval using Winnow. <http://citeseer.ist.psu.edu/cs>
- [11] T., Zhang. Regularized Winnow Methods. Advances in Neural Information Processing Systems. 2001, 13: 703-709
- [12] Y. Yang. A study on thresholding strategies for text categorization. SIGIR 2001, 137-145

