Large margin DragPushing strategy for centroid text categorization

Songbo Tan *

Software Department, Institute of Computing Technology, Chinese Academy of Sciences, P.O. Box 2704, Beijing, 100080, PR China
Graduate School of the Chinese Academy of Sciences, PR China

Abstract

Among all conventional methods for text categorization, centroid classifier is a simple and efficient method. However it often suffers from inductive bias (or model misfit) incurred by its assumption. DragPushing is a very simple and yet efficient method to address this so-called inductive bias problem. However, DragPushing employs only one criterion, i.e., training-set error, as its objective function that cannot guarantee the generalization capability. In this paper, we propose a generalized DragPushing strategy for centroid classifier, which we called as “Large Margin DragPushing” (LMDP). The experiments conducted on three benchmark evaluation collections show that LMDP achieved about one percent improvement over the performance of DragPushing and delivered top performance nearly as well as state-of-the-art SVM without incurring significant computational costs.

© 2006 Published by Elsevier Ltd.

Keywords: Text classification; Information retrieval; Machine learning

1. Introduction

With the exponential growth of text data from internet, text categorization has received more and more attention in information retrieval, machine learning and natural language processing community. Numerous machine learning algorithms have been introduced to deal with text classification, such as K-Nearest Neighbor (KNN) (Yang & Lin, 1999), centroid classifier (Han & Karypis, 2000), Naive Bayes (Lewis, 1998), support vector machines (SVM) (Joachims, 1998), Rocchio (Joachims, 1997), decision trees (Apte, Damerau, & Weiss, 1998), Winnow (Zhang, 2001), Perceptron (Zhang, 2001) and voting (Schapire & Singer, 2000).

Among all these methods, centroid classifier is a simple and effective method for text categorization. However it often suffers from inductive bias or model misfit (Liu, Yang, & Carbonell, 2002; Wu, Phang, Liu, & Li, 2002) incurred by its assumption. As we all know, centroid classifier makes a simple assumption that a given document should be assigned a particular class if the similarity of this document to the centroid of the class is the largest. However, this supposition is often violated (misfit) when there exists a document from class A sharing more similarity with the centroid of class B than that of class A. Consequently, the more serious the model misfit, the poorer the classification performance.

In order to address this issue effectively, numerous researchers have devoted their energy to refining the classifier model by proposing efficient strategies. One of the popular strategies is voting (Schapire & Singer, 2000). Wu et al. (2002) presented another novel approach to cope with this problem. However, these methods both suffer from their own shortcomings respectively. The computational costs of voting is quite significant and its application to text classification community is often severely limited; Meanwhile Wu’s strategy may be sensitive to the imbalance of text corpora for it needs to split the training set.

In order to improve the performance of centroid classifier, an effective and yet efficient strategy, i.e., “DragPushing”, was introduced to refine the centroid classifier model (Tan, Cheng, Ghanem, Wang, & Xu, 2005). The main idea behind this strategy is that it takes advantage
of training errors to successively refine classification model. For example, if one training example \( d \) labeled as class A is misclassified into class B, DragPushing “drags” the centroid of class A to example \( d \), and “pushes” the centroid of class B against example \( d \). After this operation, document \( d \) will be more likely to be correctly classified by the refined classifier.

However, DragPushing employs only one criterion, i.e., training-set error, as its objective function that cannot guarantee the generalization capability. In this paper, we further generalize DragPushing to Large Margin DragPushing (LMDP) which refines the centroid classifier model not only using training errors but also utilizing training margins. The goal of LMDP is to reduce the training errors as well as to enhance the generalization capability.

Extensive experiments conducted on three benchmark document corpora show that LMDP always performs better than DragPushing. Like DragPushing, LMDP is also much faster than many state-of-the-art approaches, e.g., SVM, while delivers performance nearly as well as SVM.

The rest of this paper is constructed as follows: Next section describes basic centroid classifier. DragPushing strategy is presented in Section 3. LMDP is introduced in the Section 4. Experimental results are given in Section 5. Finally Section 6 concludes this paper.

2. Centroid classifier

In our work, the documents are represented using vector space model (VSM). In this model, each document \( d \) is considered to be a vector in the term-space. For the sake of brevity, we denote summed centroid and normalized centroid of class \( C_i \) by \( \overline{C_i} \) and \( \overline{C_i'} \) respectively.

First we compute summed centroid by summing document vectors in class \( C_i \):

\[
\overline{C_i} = \sum_{d \in C_i} \overrightarrow{d}
\]

Next we evaluate normalized centroid by following formula:

\[
\overline{C_i'} = \frac{\overline{C_i}}{\left\| \overline{C_i} \right\|_2}
\]

where \( \left\| z \right\|_2 \) denotes the 2-norm of vector \( z \).

Afterwards we calculate the similarity between a document \( d \) and each centroid \( C_i \) using inner-product measure as follows:

\[
sim(d, C_i) = \overrightarrow{d} \cdot \overrightarrow{C_i'}
\]

It is worth mentioning that the inner-product measure using formula (3) is equivalent to cosine measure since document vector \( d \) is computed using TFIDF formula (19) which is equal to counting TFIDF by following formula (4) and then normalizing it by 2-norm.

\[
w_{\text{fid}}(t, \overrightarrow{d}) = tf(t, \overrightarrow{d}) \times \log(D/n_t + 0.01)
\]

Finally, based on these similarities, we assign \( d \) the class label corresponding to the most similar centroid:

\[
C = \arg \max_{C_i} \left( \overrightarrow{d} \cdot \overrightarrow{C_i'} \right)
\]

3. DragPushing strategy

Since the text collection may conflict the assumption of Centroid Classifier to some degree, inevitably the classification rules, i.e., class representatives or class centroids, induced by centroid classifier may contain some kinds of biases that result in degradation of classification accuracy on both training and test documents. An intuitive and straightforward solution to this problem is to make use of training errors to adjust class centroids so that the biases can be reduced gradually.

Initialization: to start, we need to load the training data and parameters including MaxIteration and ErrorWeight. Then for each category \( C_i \), we calculate one summed centroid \( \overline{CS_i} \) and one normalized centroid \( \overline{CS_i'} \). Note that 0 denotes current iteration-step, i.e., the 0th 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 following formulas:

\[
C_{4, l+1}^A = C_{4, l}^A + \text{ErrorWeight} \times d_i \text{ if } d_i > 0
\]

\[
C_{4, l+1}^N = \frac{C_{4, l}^N}{\left\| C_{4, l}^N \right\|_2} \text{ if } C_{4, l}^N > 0
\]

\[
C_{B, l+1}^S = C_{B, l}^S - \text{ErrorWeight} \times d_i \text{ if } d_i > 0
\]

\[
C_{B, l+1}^N = \frac{C_{B, l}^N}{\left\| C_{B, l}^N \right\|_2} \text{ if } C_{B, l}^N > 0
\]

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

We call the former formulas (6,7) as “drag” formulas and the latter (8,9) as “push” formulas. Obviously after the executing of “drag” and “push” formulas, the centroid of class “A” will share more similarity with document \( d \) than that of class “B”. As a result, 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: Assumed that there are \( D \) training documents, \( T \) test documents, \( W \) words in total, \( K \) classes
4. Large margin DragPushing strategy

4.1. Motivation

Margin is introduced to measure how far the positive and negative training examples are separated by the decision surface. Based on margin conception, Vapnik (1995) defined the objective function in SVM as both minimization of training-set error and maximization of margin width. Just so, SVM has turned out to be a top-performer in text categorization, often resulting in the best performance.

Accordingly DragPushing is still a linear classifier. Margin is introduced to measure how far the positive and negative training examples are separated by the decision surface. Based on margin conception, Vapnik (1995) and negative training examples are separated by the decision surface. Based on margin conception, Vapnik (1995) and Navot, & Tishby, 2002) of instance \( x_p \) with respect to a set of points \( P \) by following definition,

\[
HM_P(x_p) = \frac{1}{2}(||x_p - x_R|| - ||x_p - x_M||)
\]

where \( x_R \) and \( x_M \) denote the nearest point to \( x_p \) in \( P \) with the same and different label, respectively. Similar to above definition of hypothesis-margin, we formulate the margin of one instance \( x_i \) with respect to a centroid classifier \( C \) as following,

\[
HM_C(x_i) = (Sim(x_i, C_R) - Sim(x_i, C_M))
\]

where \( C_R \) and \( C_M \) denote the most similar Centroid to \( x_i \) with the same and different label, respectively.

4.2. The LMDP algorithm

Correspondent to the definition of training error, we introduce a generalized error “MarginError” for each example. If the hypothesis-margin of one example \( d \) is bigger than zero but smaller than a small positive constant, such as 0.05, we say centroid classifier make “MarginError” with respective to the example \( d \). For convenience, we call the small positive constant as “MinMargin”. Furthermore, we say the example \( d \) is MinMargined by the centroid classifier.

In order to apply “drag” and “push” formulas to MinMargined example, we introduce parameter MarginWeight which revises formulas (6) and (8) as follows:

\[
C_{A,l}^{S,0+1} = C_{A,l}^{S,0} + \text{MarginWeight} \times d_l \quad \text{if} \quad d_l > 0
\]

\[
C_{B,l}^{S,0+1} = \left[ C_{B,l}^{S,0} - \text{MarginWeight} \times d_l \right]_+ \quad \text{if} \quad d_l > 0
\]
In one iteration, if example $d_i$ is misclassified, LMDP updates the two centroids using formulas (6)–(9); if example $d_i$ is MinMargined, LMDP refines the two centroids using formulas (12), (7), (13) and (9). The outline of LMDP is demonstrated in Fig. 2.

Since LMDP conducts “drag” and “push” operation not only for misclassified examples but also for MinMargined examples, the running time of one iteration is $O(D(KW + 4W))$, i.e., $O(DKW)$ (for $K > 3$) which equals that of DragPushing. As a result, the training time of LMDP is the same as DragPushing. Consequently LMDP is still a linear classifier.

5. Experiment results

5.1. The datasets

In our experiment, we use three corpora: OHSUMED$^1$, Reuter-21578$^2$ and Sector-48$^3$.

5.1.1. OHSUMED

The OHSUMED dataset is a bibliographical document collection: developed by William Hersh and colleagues at the Oregon Health Science University, which is a subset of MEDLINE database. We use a subset (called oshscal$^4$ in Han & Karypis (2000)) from OHSUMED dataset that contains 11,162 documents and in total 10 categories: antibodies, carcinoma, DNA, in vitro, molecular-sequence-data, pregnancy, prognosis, receptors, risk-factors and tomography.

5.1.2. Reuter-21578

The Reuter-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 use its subset: one consisting of 92 categories and in total 10,346 documents.

5.1.3. 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 9,637 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 4,581 documents.

5.2. The performance measure

To evaluate a text classification system, we use the F1 measure introduced by van Rijsbergen (1979). This measure combines recall and precision in the following way:

\[
\text{Recall} = \frac{\text{number of correct positive predictions}}{\text{number of positive examples}} \quad (14)
\]

\[
\text{Precision} = \frac{\text{number of correct positive predictions}}{\text{number of positive predictions}} \quad (15)
\]

\[
F_1 = \frac{2 \times \text{Recall} \times \text{Precision}}{(\text{Recall} + \text{Precision})} \quad (16)
\]

For ease of comparison, we summarize the F1 scores over the different categories using the Micro- and Macro-averages of F1 scores (Lewis, Schapire, Callan, & Papka, 1996):
MicroF1 = F1 over categories and documents \( (17) \)  
MacroF1 = average of within-category F1 values \( (18) \)

The 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.

5.3. Experimental design

We evenly split the each dataset into three parts. Then we 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. This is so-called three-fold cross validation.

In all experiments we employ Information Gain as feature selection method for it consistently performs well in most cases (Yang & Pedersen. J.O., 1997). Algorithms are coded in C++ and running on a Pentium-4 machine with 3.0 GHz CPUs and 512M memory.

We employ TFIDF other than binary word occurrences as input features. The formula for calculating TFIDF can be written as follows:

\[
 w_{td}(t,d) = \frac{tf(t,d) \times \log(D/n_t + 0.01)}{\sqrt{\sum_{i \in d} [tf(t,d) \times \log(D/n_i + 0.01)]}} \tag{19}
\]

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 \( d \).

For experiments involving SVM we employed SVM-Torch, which uses one-versus-the-rest decomposition and can directly deal with multi-class classification problems. (www.idiap.ch/~bengio/projects/SVMTorch.html). We employed a linear kernel that has been found as competitive as other kernels in Reuter-21578 (Yang & Lin, 1999). All parameters were left at default values.

5.4. Comparison and analysis

Now we present and discuss the experimental results. Here we compare LMDP against DragPushing, centroid classifier and SVM on three text corpora.

Tables 1 and 2 show the best-performance comparison in MicroF1 and MacroF1. Note that for DragPushing, MaxIteration is set to 8, and ErrorWeight is fixed as 0.2. LMDP outperforms all the other three methods on OHSUMED. SVM performs the best on Sector-48. On three corpora, the MicroF1 of LMDP beats DragPushing by approximately 1%. Therefore, we make our conclusion that hypothesis-margin can boost the performance of DragPushing.

For LMDP, MaxIteration is set to 8, and MarginWeight is fixed as 0.2. LMDP outperforms all the other three methods on OHSUMED. SVM performs the best on Sector-48. On three corpora, the MicroF1 of LMDP beats DragPushing by approximately 1%. Therefore, we make our conclusion that hypothesis-margin can boost the performance of DragPushing.

Tables 3 and 4 report the training time and test time of four methods on three text collections. Note that the running time does not include the seconds for loading data from hard disk and feature number is set to 10,000; for

<table>
<thead>
<tr>
<th>Table 2</th>
<th>The best MacroF1 of different methods on three corpora</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMDP</td>
</tr>
<tr>
<td>OHSUMED</td>
<td>0.7916</td>
</tr>
<tr>
<td>Reuter-21578</td>
<td>0.6113</td>
</tr>
<tr>
<td>Sector-48</td>
<td>0.8969</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Training time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMDP</td>
</tr>
<tr>
<td>OHSUMED</td>
<td>29.766</td>
</tr>
<tr>
<td>Reuter-21578</td>
<td>22.901</td>
</tr>
<tr>
<td>Sector-48</td>
<td>9.854</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Testing time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMDP</td>
</tr>
<tr>
<td>OHSUMED</td>
<td>0.068</td>
</tr>
<tr>
<td>Reuter-21578</td>
<td>0.906</td>
</tr>
<tr>
<td>Sector-48</td>
<td>0.521</td>
</tr>
</tbody>
</table>

Fig. 3. The error rate vs. MaxIteration on Sector-48.
DragPushing. MaxIteration is set to 8, and ErrorWeight is fixed as 1.0; for LMDP, MaxIteration is set to 8, and MarginWeight is fixed as 0.2. For training phase the CPU time required by SVM is about 10 times larger than that of LMDP on Sector-48. The predicting speed of LMDP is as fast as Centroid and about 100 times faster than SVM. On Sector-48 and Reuter-21578, the training time is nearly the same as DragPushing.

Fig. 3 illustrates the error rate vs. MaxIteration on Sector-48. Note that feature number takes 10,000; for DragPushing, ErrorWeight is fixed as 1.0; for LMDP, MarginWeight is fixed as 0.2. DragPushing yields nearly the same training-error curve as LMDP. However, LMDP can decrease the margin-error with the MaxIteration. That is to say, LMDP decreases the training error as well as enlarges the training margin.

6. Conclusion remarks

In this paper we propose a generalized DragPushing strategy for Centroid Classifier, i.e., Large Margin DragPushing (LMDP). LMDP refines the centroid classifier model not only using training errors but also employing training margins. Extensive experiments conducted on three corpora showed that LMDP achieved nearly one percent improvement over the performance of DragPushing and delivered top performance nearly as well as state-of-the-art SVM without incurring significant computational costs.

References


