Improved Stumps Combined by Boosting for Text Categorization[*]

DIAO Li-li, HU Ke-yun, LU Yu-chang, SHI Chun-yi

(*State Key Laboratory of Intelligent Technology and System, Tsinghua University, Beijing* 100084, *China*)

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

E-mail: diaolili99@mails.tsinghua.edu.cn

http://www.cs.tsinghua.edu.cn

Received October 15, 2001; accepted February 6, 2002

Abstract: Stumps, classification trees with only one split at the root node, have been shown by Schapire and Singer to be an effective method for text categorization when embedded in a boosting algorithm as its base classifiers. In their experiments, the splitting point (the partition) of each stump is decided by whether a certain term appears or not in a text document, which is too weak to obtain satisfied accuracy even after they are combined by boosting, and therefore the iteration times needed by boosting is sharply increased as an indicator of low efficiency. To improve these base classifiers, an idea is proposed in this paper to decide the splitting point of each stump by all the terms of a text document. Specifically, it employs the numerical relationship between the similarities of the VSM-vector of text document and the representational VSM-vector of each class as the partition criteria of the base classifiers. Meanwhile, to further facilitate its convergence, the boosting weights assigned to sample documents are introduced to the computation of representational VSM-vectors for possible classes dynamically. Experimental results show that the algorithm is both more efficient for training and more effective than its predecessor for fulfilling text categorization tasks. This trend seems more conspicuous along with the incensement of problem scale.

Key words: text categorization; machine learning; stump; boosting

Boosting is an iterative machine learning procedure that successively classifies a weighted version of the sample, and then re-weights the sample dependent on how successful the classification was. Its purpose is to find a highly accurate classification rule by combining many weak or base hypotheses (classifiers), many of which may be only moderately accurate[1]. Stumps, which are classification trees [2] with only one split at the root node, have been shown to be effective when embedded in a boosting algorithm.

Schapire and Singer devised a boosting-based text-categorization algorithm, called ADABOOST.MH, to efficiently represent and handle set of labels[3]. In this algorithm, a simple, stump-like one-level classification tree is used, as the base hypothesis or weak learner. Its splitting criterion is whether a certain term (all words and pairs of adjacent words in a document are potential terms) exists or not. Although this setting does work in improving the performance of text categorization, there are still some drawbacks. Firstly, as base classifier, it is problematic to

---

* Supported by the National Natural Science Foundation of China under Grant No.79990580 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.G1998030414 (国家重点基础研究发展规划 973 项目)

DIAO Li-li was born in 1974. He is a Ph.D. candidate at the Department of Computer Science and Technology, Tsinghua University. His research interests are text mining, machine learning and KDD. **HU Ke-yun** was born in 1970. He is a post-doctor at the Department of Computer Science and Technology, Tsinghua University. His research interests are rough set and data mining. **LU Yu-chang** was born in 1937. He is a professor at the Department of Computer Science and Technology, Tsinghua University. His current research areas are machine learning and KDD. **SHI Chun-yi** was born in 1935. He is a professor and doctoral supervisor at the Department of Computer Science and Technology, Tsinghua University. His current research areas are AI and MAS.

decide one document belongs to or does not belong to a certain class only by checking a single “term” appeared or not. As we know, text categorization problem is usually very complex. There are always many classes and many intersections among these classes, which make it nearly impossible to represent a document only by one single term. This rule is undoubtedly too weak to be practical. Secondly, the higher the error rate of base classifier is, the more the iteration times of boosting algorithm will take to achieve reasonable overall performance. Improving the base classifier will also reduce the time spent for training. Thirdly, under this splitting rule, complicated computations and comparisons have to be made for each possible term to find out the best one for partitioning. One fact is that a text document always contains hundreds or even thousands of terms. If the training document set were large, which is very likely to happen, the algorithm’s computational complexity would be unbearable for us, reaching $O(m\cdot s\cdot len)$. Here m means the documents number of training set, s means the number of possible classes, and len means the average length of training documents.

This paper introduces an idea to improve the design of stumps, which are specifically shaped by text categorization techniques. Experimental results are presented to show how good the new method is.

1 Pre-Processing of Training Documents

VSM (vector space model) is currently the most popular representational model for text documents[4]. Given a set of m training text documents, $D=\{Doc_1,Doc_2,\dots,Doc_m\}$, for any $Doc_i\in D$, $i=1,2,\dots,m$, it can be represented as a formalized feature vector $\vec{V}(Doc_i)=(val(t_{i1}),\dots,val(t_{ik}),\dots,val(t_{in}))$, $k=1,2,\dots,n$. Here n means the number of all possible terms in the space of training set, and t_{ik} represents the k-th term of Doc_i. $val(t_{ik})$ is a numeric value used to measure the importance of t_{ik} in Doc_i, $0\le val(t_{ik})\le 1$. By this means, the problem of processing text documents has been changed to the problem of processing numerical vectors, which is quite suitable to be solved by mathematical methods. $val(t_{ik})$ can be easily computed by $val(t_{ik})=\dfrac{tf_{ik}\cdot\log\left(\dfrac{m}{d_{ik}}+\alpha\right)}{\sqrt{\sum_{k=1}^{n}tf_{ik}^{2}\cdot\log^{2}\left(\dfrac{m}{d_{ik}}+\alpha\right)}}$. tf_{ik} means the appearance frequency of t_{ik} in Doc_i. d_{ik} denotes in how many training documents t_{ik} appears. α is a constant. In the experiments we choose $\alpha=0.5$.

While computing the frequency of terms, a Stop List is used to remove the function word such as “of”, “the”, etc. If the task is to classify Chinese text document, Chinese Word Segmentation will be needed before computing. With the value of $val(t_{ik})$, feature selection can be executed by defining an importance threshold for each term. Text categorization algorithms without feature selection cannot work well in relatively large training sets, but sometimes its accuracy might be better than those with feature selection in the sense that the latter algorithms may omit important terms or include misleading terms during feature selection. We define a parameter $\rho\in[0,1]$ called feature reduction factor to reflect the ratio of selected terms to all the terms in term space. $\rho=1$ implies no feature selection at all and $\rho=0$ implies no term is selected. In most cases ρ is between these two boundary points.

2 Improving Stumps

Now let’s observe the most commonly used accompanying classifiers, those that partition the domain of the predictor variables. The most well known example is the classification tree. Each classification tree partitions the training document space D into disjoint blocks $D_0,D_1,\dots,D_N$ whose union is D. All points within a given block are classified identically so if Doc, $Doc'\in D_j$ then $h(Doc)=h(Doc')$. Here $h(\cdot)$is the predicted class of hypothesis. As many classifications need to be made in the overall boosting algorithm there has been much focus on using stumps to make the classifications. These are just classification trees with only one splitting node so that the classifier, $h(Doc)$, splits the data into only two disjoint regions. These stumps are defined completely by the single splitting question that partitions the data.

As the reasons described in the introductory section, we put forward a general form to describe a new splitting criterion: $h(Doc,l)=\begin{cases} c_{0l} & \text{if} \quad Sim(Doc,l)<thrs \\ c_{1l} & \text{if} \quad Sim(Doc,l)\geq thrs \end{cases}$. $H(Doc,l)$, one of the base classifiers established by stumps designed for multi-class multi-label settings, aims to predict the relationship between a certain *Doc* and a certain class *l* by real value of c_{0l} (intends to deny) or c_{1l} (intends to affirm). General function $Sim(Doc,l)$ represents any function that numerically measures the relationship between any document and any class it may belong to. $thrs \in [0,1]$ is the threshold of judging whether a document belongs or not belongs to a certain class. In the design of function $Sim(Doc,l)$, we found it is nature to employ the conception of the *cosine* value of the cross-angle formed between two VSM vectors. Therefore, $Sim(Doc,l)$ is devised as following:

$$Sim(Doc,l)=\cos\left(\angle\left(\vec{V}(Doc),\overrightarrow{CV}(l)\right)\right)=\frac{\sum_{k=1}^{n} val_{\vec{V}(Doc)}(t_k)\cdot val_{\overrightarrow{CV}(l)}(t_k)}{\sqrt{\sum_{k=1}^{n} val_{\vec{V}(Doc)}(t_k)^2\cdot\sum_{k=1}^{n} val_{\overrightarrow{CV}(l)}(t_k)^2}} .\ 0\leq Sim(Doc,l)\leq 1 .$$

$\overrightarrow{CV}(l)$ denotes the central or representational vector of all training documents that belong to class *l*, the detailed computation method of which will be introduced in the next section. Since both $\vec{V}(Doc)$ and $\overrightarrow{CV}(l)$ can be computed out, the only problem left to the base classifier is how to choose the value of $thrs$.

In boosting algorithm, for each possible class, each document in training set is bound with a real-value: weight. The whole distribution of all weights is denoted as Δ. Let W_+^{jl} (W_-^{jl}) be the weight (respect to the distribution Δ) of the documents in partition D_{jl} (D_{0l} or D_{1l}, suppose $thrs$ split the training set D into two partitions: D_{0l}, $Sim(Doc,l)<thrs$, and D_{1l}, $Sim(Doc,l)\geq thrs$) that are (are not) labeled by *l*. For each possible label *l*, for $j\in\{0,1\}$, and for $b\in\{+1,-1\}$ (which is always denoted as $\{-,+\}$), $W_b^{jl}(thrs)=\sum_{i=1}^{m}\sum_{l\in y}\Delta(Doc_i,l)[\![Doc_i\in D_{jl}\wedge IsBelonging(Doc_i,l)=b]\!]$. Here *y* denotes the set of all possible classes in the training set. $\Delta(Doc_i,l)$ denotes the weight of training document Doc_i related with class (label) *l*. $[\![\cdot]\!]$ is a function that output 1 if its content keeps true and 0 otherwise. $IsBelonging(Doc_i,l)$ outputs +1 if Doc_i belongs to class *l* and –1 otherwise. The $W_b^{jl}(thrs)$ defined above is a function of $thrs$ since the partitions would be different with different $thrs$. For simplicity, we use W_b^{jl} to represent $W_b^{jl}(thrs)$. According to the theory of Schapire and Singer[5], the best $thrs$ should be the one that can split the whole training set into two partitions that minimize the value of $z=2\sum_{j\in\{0,1\}}\sum_{l\in y}\sqrt{W_+^{jl}W_-^{jl}}$. Here *z* is the score function defined for the base classifier.

Since $thrs\in[0,1]$, we need to choose discrete points from domain $[0,1]$ for finding out the best *thrs*. Let $A\in N$, by picking up the series of points of $0,\frac{1}{A},\frac{2}{A},\cdots,\frac{A-1}{A},1$, we can obtain $(A+1)$ points as the possible values $thrs$ may take. The larger the *A* is, the more accurate the final *thrs* is. For each point selected by *thrs*, the corresponding score (*z*) is computed. After all scores with regard to all the possible points of *thrs* are calculated, the *thrs* with lowest score should be selected as the threshold of stump, the base classifier of boosting.

The output of the stump therefore should be:

$$h(Doc,l)=\begin{cases} c_{0l}=\frac{1}{2}\log\left(\frac{W_{+}^{0l}}{W_{-}^{0l}}\right) & \text{if} \quad Sim(Doc,l)<thrs \\ c_{1l}=\frac{1}{2}\log\left(\frac{W_{+}^{1l}}{W_{-}^{1l}}\right) & \text{if} \quad Sim(Doc,l)\geq thrs \end{cases}.$$

3 Boosting Algorithm

Let χ denote the domain of possible text documents and let $y=\{y_1,y_2,\dots,y_s\}$ be a finite set of S labels or classes. Let $D=\{Doc_1,Doc_2,\dots,Doc_m\}$ denote the training set of m text documents, $D\subset\chi$. In the multi-label case, each document $Doc\in\chi$ may be assigned multiple labels in y. Thus, a labeled example is a pair $(Doc,Y(Doc))$ where $Y(Doc)\subseteq y$ is the set of labels assigned to Doc. Formally, the training set D transferred to boosting algorithms should be $D=\{(Doc_1,Y(Doc_1)),(Doc_2,Y(Doc_2)),\dots,(Doc_m,Y(Doc_m))\}$.

We employ the improved stumps described in Section 2 as the base classifier repeatedly called by boosting algorithm. As for the inputted information needed by the stumps, besides the training set D and distribution Δ, the central (representational) vectors of each possible label (class) $\overrightarrow{CV}(l)$ are also important. The traditional method is, for each class $l\in y$, to compute the arithmetical mean of all VSM vectors of the training documents for which l is one of the classes they belong to. Another version is assigning different weights to different training documents according to their importance and then averaging the weighted VSM vectors to obtain better central vectors. Since the distribution Δ maintained by boosting algorithm also reflects the relative importance of training documents according to each possible class, it might be nature to use Δ in computing $\overrightarrow{CV}(l)$ for each $l\in y$. Formally, $\overrightarrow{CV}(l)=\frac{\sum_{Doc,l\in Y(Doc)}\vec{V}(Doc)\cdot\Delta(Doc,l)}{\sum_{Doc,l\in Y(Doc)}\Delta(Doc,l)}$. This setting is expected to facilitate the converging speed of boosting, which, if true, would absolutely be a big privilege over traditional method for computing central (representational) class vectors.

Now we can start to illustrate the boosting algorithm slightly modified for collaborating with the settings defined above. The boosting algorithm is shown in Fig.1. This algorithm maintains a set of weights as a distribution Δ_t over example documents and labels (classes). Initially, this distribution is uniform. On each round t, the distribution Δ_t (together with the training set D and class central vectors $\overrightarrow{CV}_t$) is passed to the base classifier who computes weak hypotheses $\{h_t\}$ for all $l\in y=\{y_1,y_2,\dots,y_s\}$ where $h_t:\chi\times y\to R$. We interpret the sign of $h(Doc,l)$ as a prediction as to whether the label l is or is not assigned to Doc. The magnitude of the prediction $|h(Doc,l)|$ is interpreted as a measure of “confidence” in the prediction. Since the base classifier aims to minimize the training set error, according to the theory of R.Schapire, α_t should be set to 1. The final hypotheses rank documents using weighted votes of the base classifiers. This algorithm is derived using a natural decomposition of the multi-class, multi-label problem into S orthogonal binary classification problems. That is, we can think of each observed label set $Y(Doc)$ as specifying s binary labels (depending on whether a label l is or is not included in $Y(Doc)$), and we can then apply binary-prediction boosting algorithms.

Given: $D=\{(Doc_1,Y(Doc_1)),(Doc_2,Y(Doc_2)),\ldots,(Doc_m,Y(Doc_m))\}$ where $Doc_i\in\chi$, $Y(Doc_i)\subseteq y=\{y_1,y_2,\ldots,y_s\}$, $s\in N$. $i=1,2,\ldots,m$.

Initialize $\Delta_1(Doc_i,l)=1/(m\cdot s)$ for all $i=1,2,\ldots,m$ and all $l\in y$

For $t=1,2,\ldots,T$:

- For each class $l\in y$, compute the corresponding central (representational) VSM vector;
- Pass distribution $\Delta_t=\{\Delta_t(Doc_i,l)\}_{i,l}$ and central vectors $\overrightarrow{CV}_t=\{\overrightarrow{CV}_t(l)\}_l$ to the base classifier, say, the stump;
- Stump generates base classifiers $\{h_t(Doc,l)\}_l$ for all $l\in y$ where $h_t:\chi\times y\to R$;
- Set $\alpha_t=1$ and $z_t=\sum_{i=1}^{m}\sum_{l\in y}\Delta_t(Doc_i,l)\cdot e^{-\alpha_t\cdot IsBelonging(Doc_i,l)h_t(Doc_i,l)}$;
- Update: $\Delta_{t+1}(Doc_i,l)=\dfrac{\Delta_t(Doc_i,l)\cdot e^{-\alpha_t\cdot IsBelonging(Doc_i,l)h_t(Doc_i,l)}}{z_t}$ for all $l\in y$ and for all $i=1,2,\ldots,m$;

Output the final hypotheses: $\left\{f(Doc,l)=\sum_{t=1}^{T}\alpha_t h_t(Doc,l)\right\}_l$.

Fig.1 The boosting algorithm for text categorization

4 Experiments

We choose "Precision" as the main way for assessing and comparing the performances of text categorization methods. We call the new algorithm proposed here as "AdaBoost.SZ". Three other algorithms are also selected for comparison. They are TF-IDF, NAÏVE BAYESIAN, ADABOOST.MH. We have conducted a number of experiments to test their validity and to compare the differences of their performances. The parameters, such as *m*, *A*, *s*, *T*, ρ as defined previously, are adjusted to check the performances of these algorithms in different situations. For these experiments we used "YAHOO! CHINESE NEWS" as training and testing documents, which can be retrieved from http://cn.news.yahoo.com. The classes and corresponding training examples (documents) are selected according to the parameters adjusted. Tables 1 to 5 present the results.

Table 1 Precision on different feature reduction factors (ρ)

Algorithms	Feature reduction factor				
	0.01	0.03	0.06	0.10	1.00
AdaBoost.SZ	0.822 8	0.889 4	0.911 2	0.934 4	0.900 7
TF-IDF	0.710 0	0.757 3	0.772 2	0.789 6	0.776 5
NAÏVE BAYESIAN	0.765 0	0.810 4	0.834 4	0.850 5	0.830 3
ADABOOST.MH	0.784 6	0.840 3	0.876 5	0.890 9	0.901 2

Table 2 Precision on different training document numbers (*m*)

Algorithms	Training document number				
	500	1000	1500	2000	2500
AdaBoost.SZ	0.703 3	0.810 5	0.887 9	0.905 1	0.923 0
TF-IDF	0.626 0	0.715 1	0.759 2	0.775 6	0.781 6
NAÏVE BAYESIAN	0.670 9	0.730 6	0.794 4	0.832 8	0.865 9
ADABOOST.MH	0.658 3	0.751 1	0.842 9	0.889 4	0.900 1

Table 3 Precision on different possible class numbers (*s*)

Algorithms	Possible class number				
	5	10	15	20	40
AdaBoost.SZ	0.960 0	0.967 3	0.944 3	0.946 7	0.933 5
TF-IDF	0.824 3	0.819 6	0.800 8	0.792 1	0.772 2
NAÏVE BAYESIAN	0.889 4	0.875 3	0.866 0	0.854 2	0.841 7
ADABOOST.MH	0.923 3	0.926 5	0.911 8	0.909 5	0.893 9

Table 4 Precision on different values of *A*

Algorithm	Value of *A*				
	10	50	100	500	1000
AdaBoost.SZ	0.897 2	0.925 6	0.934 4	0.937 3	0.937 6

Table 5 Precision on different iteration times of boosting (T)

Algorithms	Iteration times				
	50	100	200	500	1000
AdaBoost.SZ	0.866 3	0.888 2	0.893 7	0.934 4	0.945 6
ADABOOST.MH	0.801 0	0.834 9	0.859 2	0.890 9	0.901 7

Generally speaking, TF-IDF is a very simple way for text categorization with relatively low accuracy. Naïve Bayesian is a little bit better than TFIDF, and since it is a probability-based method, when the training set becomes large enough, its accuracy would be improved. AdaBoost.MH is worse than NAÏVE BAYESIAN and even worse than TF-IDF when training set is small and iteration times is not large enough. But when parameters are adjusted to a reasonable place, its overall performance would be slightly better than NAÏVE BAYESIAN. In most cases AdaBoost.SZ outperform ADABOOST.MH.

From adjusting the parameters we observed some interesting phenomenon. The performance will always increase along with higher reduction factor ρ, which implies selecting more terms into feature set. But we found after the reduction factor is higher than 0.1, the improvement of accuracy is too little to be noticed. And when ρ is very large (close to 1) it is even slightly worse than that of ρ=0.1. The reason may be that, as we choose terms as feature, it would be probable to include some useless or even misleading terms. When the feature set is small, such “bad” terms are also sparse and hence have only trivial influence to the classification results. But when the feature set is large enough, such terms would be likely to play a role in making the final decision. From the experiments we also can see the accuracy of classifiers would increase along with the increase of training set size m. Boosting algorithms would have overwhelmingly advantages over non-boosting algorithms once m>1500. For each boosting algorithm, its iteration times T should be one of the most important parameters for achieving required accuracy with reasonable costs (CPU time). As T increases, boosting algorithms increase their accuracies accordingly, and over-fitting is hardly observed, which is a phenomenon consistent with others’ theoretical and practical analysis[3]. Theoretically the value of parameter A should be the higher, the better. But according to the experimental results, after A>100 the improvement of accuracy of such classifiers becomes very trivial and running such programs becomes an unbearable task with sharply increased consuming of time and space. The reason may be that 0.01 as a scale is already good enough for distinguishing between useful terms and useless terms.

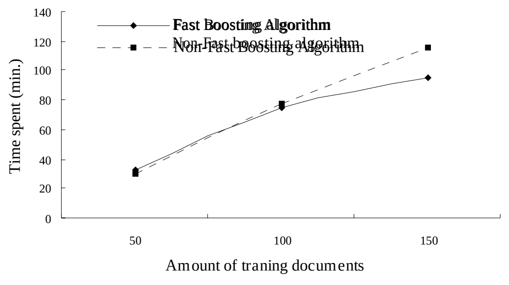

Fig.2 Efficiency between fast and non-fast boosting algorithms

We also compared the efficiency of our new idea of facilitating the converging speed of boosting algorithm with that does not employ this technique. Figure 2 presents the results, which clearly show the benefits this new idea can provide.

5 Conclusions

So far we discussed the idea for improving the performance of boosting algorithm with stumps as its base classifiers employed by multi-class multi–label text categorization tasks. In the experiments we proved that this is beneficial and promising. The further developments of such ideas might include incorporating the concept of SVM to the formation of stumps and using Bayesian theorem to improve the performance of boosting as a whole, etc. Stump is a kind of base classifier easy to be implemented, but its performance is still far from being perfect. Maybe its more complicated version, i.e. more partitions, or more nodes, can be more suitable to be integrated into boosting algorithms for resolving text categorization problems.

References:

[1] Freund, Y., Schapire, R. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 1997,55(1):119~139.

[2] Breiman, L., Friedman, J., Olshen, R., *et al*. Classification and Regression Trees. Belmont, CA: Wadsworth, 1984. 1~357.

[3] Schapire, R., Singer, Y. BoosTexter: a boosting-based system for text categorization. Machine Learning, 2000,39(2/3):135~168.

[4] Salton, G., Wong, A., Yang, C. A vector space model for automatic indexing. Communications of the ACM, 1995,18:613~620.

[5] Schapire, R., Singer, Y. Improved boosting algorithms using confidence-related predictions. Machine Learning, 1999,37(3): 297~336.

用 **Boosting** 方法组合增强 **Stumps** 进行文本分类

刁力力, 胡可云, 陆玉昌, 石纯一

(清华大学 智能技术与系统国家重点实验室,北京 100084)

(清华大学 计算机科学与技术系,北京 100084)

摘要: 为提高文本分类的精度,Schapire 和 Singer 尝试了一个用 Boosting 来组合仅有一个划分的简单决策树(Stumps)的方法.其基学习器的划分是由某个特定词项是否在待分类文档中出现决定的.这样的基学习器明显太弱,造成最后组合成的 Boosting 分类器精度不够理想,而且需要的迭代次数很大,因而效率很低.针对这个问题,提出由文档中所有词项来决定基学习器划分以增强基学习器分类能力的方法.它把以 VSM 表示的文档与类代表向量之间的相似度和某特定阈值的大小关系作为基学习器划分的标准.同时,为提高算法的收敛速度,在类代表向量的计算过程中动态引入 Boosting 分配给各学习样本的权重.实验结果表明,这种方法提高了用 Boosting 组合 Stump 分类器进行文本分类的性能(精度和效率),而且问题规模越大,效果越明显.

关键词: 文本分类;机器学习;stump;boosting

中图法分类号: TP181 文献标识码: A