Boosting to Correct Inductive Bias in Text Classification

Yan Liu, Yiming Yang, Jaime Carbonell
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-8213, USA
{yanliu, yiming, jgc}@cs.cmu.edu

ABSTRACT

This paper studies the effects of boosting in the context of different classification methods for text categorization, including Decision Trees, Naive Bayes, Support Vector Machines (SVMs) and a Rocchio-style classifier. We identify the inductive biases of each classifier and explore how boosting, as an error-driven resampling mechanism, reacts to those biases. Our experiments on the Reuters-21578 benchmark show that boosting is not effective in improving the performance of the base classifiers on common categories. However, the effect of boosting for rare categories varies across classifiers: for SVMs and Decision Trees, we achieved a 13-17% performance improvement in macro-averaged F_1 measure, but did not obtain substantial improvement for the other two classifiers. This interesting finding of boosting on rare categories has not been reported before.

Categories and Subject Descriptors

I.5.1 [**Computing Methodologies**]: Pattern Recognition—*Models*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

Text Classification, Machine Learning, Boosting, Inductive Bias

1. INTRODUCTION

Boosting is an important method in machine learning, which has gained popularity in recent years. It works by repeatedly running a given weak learning algorithm on various training examples sampled from the original training pool, and combining the classifiers produced by the weak learner into a single composite classifier. Boosting has its root in PAC (probably approximately correct) learning theory [23]. Kearns and Valiant were the first to propose the idea of boosting a "weak" learning algorithm that performs just slightly better than random guessing into a "strong" learning algorithm. As a general approach to improving the effectiveness of learning, boosting has been investigated in many variants in recent years, and has been the subject of both theoretical analysis and practical applications. However, the differential effect of boosting different base classifiers as a function of per-category training data size has not been previously investigated, and is the focus of this paper.

AdaBoost, first introduced by Freund and Schapire in 1995[5], is an iterative algorithm which induces a series of classifiers ("base classifiers") using a training set of labelled examples. Schapire[21] proved that the training error of AdaBoost drops exponentially fast if the error rate of each base classifier is slightly smaller than 0.5 and gave a qualitative weak bound on the generation error. Friedman[7] suggested a connection between boosting and logistic regression. Freund and Shapire also connected boosting with game theory and linear programming. Lebanon and Lafferty demonstrated a clear theoretical connection (a near equivalence) between boosting and exponential models [11]. These studies give a set of theoretical evidence that given sufficient data and a weak learner, we can reliably construct moderately "strong" classifiers using boosting.

Applying boosting to text categorization tasks, Schapire and Singer[22] evaluated AdaBoost on a benchmark corpus of Reuters news stories (the Apte version of Reuters-21450, see Section 5.1). They obtained the results comparable to the best results of Support Vector Machines and k-Nearest Neighbor methods[26, 25, 20], and better performing than Sleeping-experts, Rocchio, Naive Bayes and PrIF/DF[22].

An interesting and effective variant of the boosting method is the *resampling* approach by Weiss et al.[25]. Instead of assigning explicit weights to training instances and using a specific formula to update those weights, the *resampling* boost constructs a different training sample for each iteration by updating the underlying distribution of the training documents (i.e., increasing the population of misclassified documents) and then randomly sampling from the new population. To make the subsequent classifiers emphasize the difficult examples, the system maintains the number of cumulative errors made by previous classifiers on each training document, and updates the weight of that document using a simple pseudo-probability function (Section 2). The composite final classifier consists of an unweighted vote of the de-

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.

cisions made by the base classifiers. It has been proved that unweighted vote and resampling with any technique that can effectively increase the likelihood of the erroneously classified documents in subsequent training cycles will achieve improvement[3].

Although the previous work provided theoretical foundations for boosting methods and strong empirical results in text categorization evaluations, further research is needed for deeper understanding of the power and limitations of boosting approaches. In this paper, we study the issue of using boosting to reduce the inductive biases of different classification algorithms, a research question which has not been explicitly explored previously. Specifically, we are interested in how well boosting works with "non-weak" learning algorithms, by which we mean classification methods whose performance is much stronger than a near random classifier or even decision stumps. We chose to use Decision Trees, Naive Bayes, Support Vector Machines (SVMs) and a Rocchio classifier which have been among the top-performers in text categorization evaluations [9, 26, 27, 22, 12] as base classifiers for our investigation on boosting. We focus on answering the question of whether boosting is an effective way of correcting the inductive biases in those classifiers. We hope our analysis and experiments suggest useful insights for boosting with a broader range of classifiers as well.

2. BOOSTING ALGORITHMS

The version of boosting mainly investigated in this paper is AdaBoost.M1 (Freund and Schapire). In order to illustrate that sampling with any technique that can effectively increase the weight of the erroneously classified documents will achieve improvement, not constrained to the weight updating function of AdaBoost, we implemented *resampling* boost algorithms as well for boosting Naive Bayes, SVMs and Rocchio. These two algorithms are outlined below.

AdaBoost: Let W_i^t be the weight of document i on training iteration t. Initialize W_i^1 to be $\frac{1}{N}$, where N is the size of the training set of documents. Particularly, in the resampling version of boosting, W_i^t reflects the probability of document i to be sampled. In each training iteration t=*1,2,...,*T, the system constructs a training sample of documents chosen according the weight $\vec{W}^t$ and uses that sample to train a base classifier C^t (using a given learning algorithm). Defining

$$\epsilon^t = \sum_{i:C^t(x_i)\neq y_i} W_i^t$$

$$\alpha^t = \frac{1}{2} ln(\frac{\epsilon^t}{1-\epsilon^t})$$

and the weight of a document is updated by

$$W_i^{t+1} = \frac{W_i^t}{Z^t} \times \{ \begin{array}{ll} e^{-\alpha_t} & if\ C^t(x_i) = y_i \\ e^{\alpha_t} & if\ C^t(x_i) \neq y_i \end{array} \quad (1)$$

where Z^t is a normalization factor. Boosting will terminate if ϵ^t is greater than 0.5 or equal to 0. The resulting classifier is the weighted vote of the base classifiers $C^1, C^2, ..., C^T$ where the weight of C^t is α^t.

Resampling Boost: The idea of *resampling* boost is similar to AdaBoost except that the weight updating function is different. During each iteration in boosting, the training documents that are misclassified by the previously induced base classifier receive an increased weight. That is, their probabilities for being selected in the next iteration is updated as follows: let $e(d_i)$ be the cumulative number of times that document d_i is misclassified in previous iterations, and N denotes the number of documents in the training set. The probability that document d_i will be sampled is:

$$P(d_i) = \frac{1 + e(d_i)^3}{\sum_{i=0}^{N}(1 + e(d_i)^3)} \quad (2)$$

Finally, the resulting classifier is the unweighted majority vote of all the multiple base classifiers.

3. INDUCTIVE BIASES OF CLASSIFIERS

Inductive learning is the problem of inducing classification functions from a set of training examples[15]. Depending on how the classification functions are induced, classifiers may have different inductive biases, by which we mean different assumptions underlying their inductive inferences. In general, there are two types of inductive biases: *preference bias* and *restriction bias*. The former comes from a preference in the search strategy; that is, the algorithm prefers certain types of hypotheses over others (shorter hypotheses over longer ones in a disjunctive normal form learner, for example, or a more general MDL criterion in other symbolic classifiers). The latter comes from a restriction on the search space; that is, there is a restriction on classification functions to be considered (linear functions only, for example, or independence assumptions among the input features to the function in Naive Bayes).

Inductive learning classifiers are able to classify unseen instances only because they have an implicit bias for selecting one consistent hypothesis over another. On the other hand, an inductive bias may be the cause of classification errors when it is inconsistent with the target concepts. Our task, therefore, is to identify the inductive biases of the classifiers and find ways to correct them when they are not suitable for the target concepts. To make our points clear, we discuss below the inductive biases of the four classification methods which are common in text categorization. Those classifiers are Decision Trees[1, 25], Naive Bayes[14], Support Vector Machines (SVMs)[9] and Rocchio[22].

A **Decision Trees** is a tree structure grown from the root downward using a greedy algorithm which selects the next best feature (that is, word for text classification) for each new decision branch added to the tree, typically with an entropy-minimization criterion such as information gain. The resulting tree is equivalent to *Disjunctive Normal Form* (DNF) rules that cover the complete hypothesis space; thus, Dtrees have no restriction bias with respect to the given set of features.[1] On the other hand, Dtrees have a clear preference bias in their search strategy, that is, the greedy feature selection based on information gain and a preference for smaller trees over larger trees (either during initial Dtrees construction, or by post-construction automated pruning).

Naive Bayes classifiers estimate the probability of each candidate category for a given document using the joint

[1] In reality, for text categorization, Dtrees are typically induced using a much smaller, selected set of features instead of the full vocabulary found in training documents in order to accommodate the computation cost. Restriction bias would also be an issue in such a case.

probabilities of features in that instance (e.g. words in a document) given the category. An obvious inductive bias of NB corresponds to the "naive" assumption that the conditional probability of a word given a category is independent of the conditional probabilities of other words given that category. This bias is a consequence of the expressive power of its hypothesis representation, i.e., a strong restriction bias. Another more subtle inductive bias of NB is that each training example (document) has equal weight during the computation of the conditional probabilities.

SVMs are defined over a vector space, and the problem is to find a decision surface that "best" separates the data points into two classes based on the Structural Minimization Principle. Specifically, the "best" decision surface in a linearly separable space is a hyperplane that maximizes the "margin", that is the distance between two parallel hyperplanes that separate the two classes of data points in the training set. Maximizing the margin as the optimization criterion introduces a preference bias: only the data points on the margin (namely, the support vectors) have a non-zero weight in the classification function. As a consequence, the decision boundary is likely to be too sensitive to outliers around the margin (if any), and not sufficiently sensitive to the density of data points beyond the margin. The SVMs algorithm for linearly separable cases can be extended for solving the linearly non-separable cases by either introducing a soft margin hyperplane, or by transforming the original data vectors to a higher dimensional space where the data points become linearly separable. The transformed solution has the same inductive bias as a linear SVMs, and the soft-margin solution has the following inductive bias: the data points within the margin have non-zero weights, but the data points beyond the margin are still ignored. Similar biases exist if other (non-linear) kernel functions are used for SVMs.

Rocchio classifiers compute a *prototype* vector for each category as a weighted average of positive and negative training examples. A commonly used formula for the prototype of a category (c) is:

$$\vec{c}(\gamma, k) = \frac{1}{|R_c|} \sum_{u_i \in R_c} \vec{u}_i - \gamma \frac{1}{|\bar{R_{c,k}}|} \sum_{v_i \in \bar{R_{c,k}}} \vec{v}_i \qquad (3)$$

where R_c is the "centroid" of positive training examples; $\bar{R_{c,k}}$ is the centroid of the negative training examples as selected in the "query zone"[2][22] and γ is the weight of the negative centroid. For both efficiency and effectiveness, we retain only the top p_{max} non-zero elements in the prototype vector[3]. Rocchio estimates the relevance score of each category with respect to a test example using the cosine of the prototype vector of the category and the document vector, and thresholds on those scores for classification decisions.

An inductive (restriction) bias in Rocchio is that it only allows a two-centroid representation for the prototype of a class, which may not be suitable or sufficient for the kind of categories whose examples are naturally clustered around more than two centroids. Another inductive (preference) bias is that all the positive (or negative) training examples are given an equal weight when computing a positive (or negative) centroid, which is clearly different from SVMs.

4. BOOSTING FOR INDIVIDUAL CLASSIFIERS

DTreeBoost: The original decision-trees algorithm covers a complete hypothesis space and has no restriction bias. However, for text categorization we typically select only a subset of the features instead of the full vocabulary. This saves substantial computation cost, but simultaneously introduces a restriction bias. Boosting has particular advantages on correcting this bias for the reason that this modified decision trees have a larger variance than the original Dtrees, that is, when the training documents are slightly changed, the trees induced from these data may differ greatly. When boosting gradually updates the weight of each training document, the set of selected features change as a function of the weight, and so the inductive bias adjusts accordingly.

NBBoost:Boosting Naive Bayes strengthens the expressive power of a hypothesis space by breaking the independence assumption. Another inductive bias comes from the implicit assumption that each training document has equal weight. Let us see in detail how boosting addresses this bias: Naive Bayes classifier applies Bayes' rule when making the classification judgement, that is

$$P(c_j|d_i) = \frac{P(c_j)P(d_i|c_j)}{P(d_i)}.$$

The result category we assigned to this document is

$$C = argmax_{c_j} P(c_j|d_i)$$

In the *multinomial* model, the probability of the word w_f given category c_j and the prior probability of category c_j are computed as follows:

$$P(w_f|c_j) = \frac{\sum_{i=1}^{N} n_{if}}{\sum_{s=1}^{V} \sum_{i=1}^{N} n_{is}},$$

$$P(c_j) = \frac{\sum_{d_i \in C_j} 1}{\sum_{k=1}^{N} 1}.$$

After plugging in the weight by boosting, we get

$$P'(w_f|c_j) = \frac{\sum_{i=1}^{N} n_{if} W_i^t}{\sum_{s=1}^{V} \sum_{i=1}^{N} n_{is} W_i^t}, \qquad (4)$$

$$P'(c_j) = \frac{\sum_{d_i \in C_j} W_i^t}{\sum_{k=1}^{N} W_k^t}. \qquad (5)$$

where W_i^t is the weight of document i in iteration t, n_{if} is the number of appearances of word f in document i, N refers to the number of training documents and V refers to the vocabulary size. In this way, boosting assigns larger weights to the misclassified training documents in previous iterations and smaller weights to the correct ones.

SVMBoost:The inductive bias of SVMs we are concerned about is that only the training points on the margin (the support vectors) have a non-zero weight in the classification function. To allow other training examples to also have a contribution, the minimization function is modified. To

[2] That is, the k top-ranking documents retrieved from the negative training examples when using the centroid of the positive training examples as the query

[3] These choices for the values of p_{max}, γ and k are based on parameter optimization using k-fold cross validation over a training set (in our experiments, k=5)

illustrate how the weight of each training document influences each base classifier we built, we choose the linearly non-separable case, in which the original problem is reduced to a quadratic programming problem:

Maximize

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j}$$

subject to

$$0 \leq \alpha_i \leq C_i,$$

$$\sum_{i=1}^{i=N} \alpha_i y_i = 0$$

the solution is given by

$$\vec{w} = \sum_{i=1}^{N_s} \alpha_i y_i \vec{x_i},$$

where $(\vec{x_i}, y)$ corresponds to one training document; N is the number of whole training set; α_i is Lagrange multipliers, and C is a constant.

After adding our modification for boosting, the formula above become:

Maximize

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} W_k^t W_l^t \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \qquad (6)$$

subject to

$$0 \leq \alpha_i \leq C_i,$$

$$\sum_{i=1}^{i=N} \alpha_i y_i = 0.$$

where W_k^t is the weight of document k in iteration t.

In this way, the documents that are misclassified by previous classifiers will have larger weights in the classification function of the current iteration. By this error-driven correction technique, boosting helps to correct the preference bias of SVMs in terms that the data points beyond the original margin (that is the margin without boosting) will also have an effect on the final classification, whether great or slight. The number of documents to be selected on each training round is a parameter that can be tuned using cross validation.

RocchioBoost: Rocchio assumes that each training document is given equal weight when computing the centroid (positive or negative). Boosting corrects this bias by increasing the weights of the misclassified documents in each training round. As a result, the centroid will move toward these data points in the next iteration and the resulting classification function will have a preference bias for those documents that are particularly hard to classify. For the implementation of RocchioBoost we could also use the resampling strategy, as we did for SVMs and Naive Bayes. However, we instead directly added a per-document weight in the Rocchio formula for computing the prototype of a category because it was a straightforward and even simpler solution. The modified formula is:

$$\vec{c}(\gamma, k) = \frac{\sum_{u_i \in R_c} W_i^t \vec{u}_i}{\sum_{u_i \in R_c} W_i^t} - \gamma \frac{\sum_{v_j \in R_{c,k}^{-}} W_j^t \vec{v}_j}{\sum_{v_j \in R_{c,k}^{-}} W_i^t} \qquad (7)$$

where W_i^t and W_j^t are the updating weights of document i and j, respectively, at time t.

5. EMPIRICAL VALIDATION

We conducted experiments for boosting with the four classifiers: Decision Trees, Naive Bayes, SVMs and Rocchio.

5.1 Data, Text Representation and Metrics

For our data collection we chose the ApteMod version of Reuters-21578 corpus which has been used as a benchmark in recent text categorization evaluations, and is similar to the previous version named the Apte set of Reuters-21450[27]. It consists of a training set of 7,769 documents and a test set of 3,019 documents, with 90 categories each of which has at least one occurrence in both sets. The number of categories per document is 1.3 on average.

We pre-processed the documents, including down-casing, tokenization, removal of punctuation and stop words, stemming and supervised statistical features selection. We defined tokens to be maximal sequences of letters and digits, followed by an optional “’s” or “’nt”. Tokens which were purely numbers were discarded, as were words on the SMART stoplist[19]. Tokens were stemmed with the Porter Stemmer. The resulting documents had a vocabulary of 24,240 unique words. Features corresponding to stemmed words were ranked using the χ^2-max criterion, i.e. the maximum of χ^2 over the 90 categories [18, 28]. Feature sets of several sizes were chosen by going down this list to various depths. The optimal feature set size was chosen separately for each classifier (Dtrees, SVMs, kNN, or Rocchio) by 5-fold cross validation so as to optimize macro-averaged or micro-averaged F_1. Document vectors based on these feature sets were computed using the SMART *ltc* version of TF-IDF term weighting [4]. This gives term t in document d a weight of:

$$w_d(t) = (1 + \log_2 n(t, d)) \times \log_2(|\mathcal{D}|/n(t)) \qquad (8)$$

where $n(t)$ is the number of documents that contain t.

For the evaluation metric, we used a common effectiveness measure, F_1, defined to be [13, 24]:

$$F_1 = \frac{2A}{2A + B + C} \qquad (9)$$

where A is the number of documents the system correctly assigns to the category, B is the number of documents incorrectly assigned to the category (aka false-alarms), and C is the number of documents which belong to the category but are misclassified as non-members. It is easy to show that F_1 is equivalent to the *harmonic average* of precision (defined to be $p = \frac{A}{A+B}$) and recall (defined to be $r = \frac{A}{A+C}$). That is,

$$F_1 = \frac{2rp}{r + p} \qquad (10)$$

To measure overall effectiveness we use both the *micro-average* (effectiveness computed from the sum of per-category contingency tables) and the *macro-average* (unweighted average of effectiveness across all categories). The former is dominated by the system's performance on common categories while the latter is dominated by the performance on rare categories, if the majority of categories in the data collection are rare as is typically the case [26].

5.2 Experiment Results on Boosting

In our experiment, we tested both the AdaBoost and the *resampling* boost. We also ran the classifiers without boosting as a baseline for comparison. We set the iteration number in the boosting runs to be 5, 10 and 20 respectively.

In the following sections, Figure 1-4 shows the performance curves (interpolated) of AdaBoost and *resampling* boost as a function of the iteration number in boosting[4]. Table 1-4 presents the peak scores of each method at the corresponding iterations and when micro- and macro-averaged F_1 were chosen as the measure, respectively [5].

SVMBoost In our experiments with SVMs we used the *SVM_Light* package ([10], *http://www.joachims.org/svm_light*). We employed SVMs with a linear kernel which we have found as competitive as other kernels in a previous study on this corpus[27]. All parameters were left at default values. In feature selection we found the performance of SVMs not sensitive to the feature-set size when the set is sufficiently large, which is consistent to previously reported observations[9]; therefore we used the entire feature space. The results of boosting SVMs are shown in Figure 1 and Table 1.

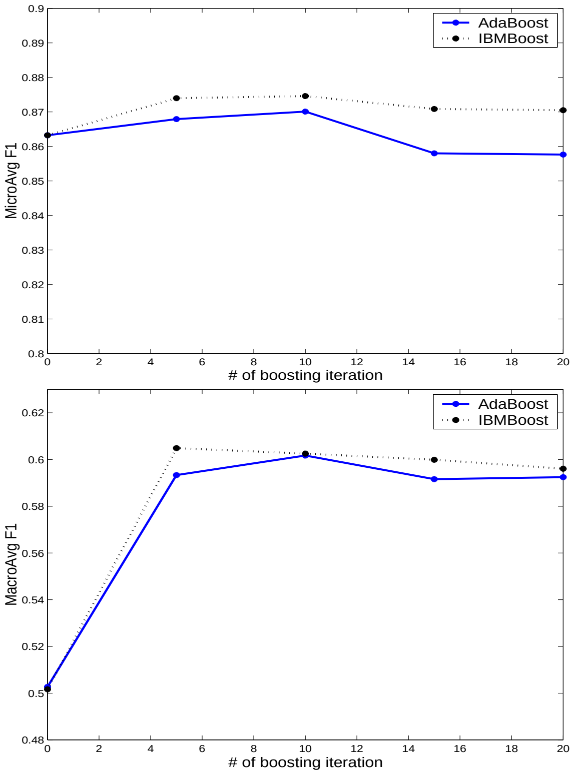

Figure 1: Perfermance of SVMBoost on Reuters-21578

Comparing these micro-averaged scores to previously published results on the same data collection, the score (0.863 in F_1) of our baseline SVMs is similar to the scores (0.860 - 0.864 in F_1) reported by Joachims for SVMs without boosting[9]. Our scores (0.875 in F_1) for SVMs with *resampling* boost are comparable to the results reported by Weiss et al. for their experiments of *resampling* boost with decision trees for which they obtained the break-even point scores (the simple average of recall and precision) of 0.859 - 0.878 when the number of boosting iterations varied from 10 to 100. Our score (0.870 in F_1) for SVMs with AdaBoost is slightly higher than the result (0.853 in F_1) reported by Schapire and Singer for AdaBoost (MH) on Reuters-21450, an older version of the Reuters-21578 dataset.

Micro-averaged Performance

	iterations	miPre	miRec	miF1	improvement
AdaBoost	10	.8904	.8507	.8701	+0.8%
Resampling Boost	10	.8931	.8568	.8746	+1.3%
NoBoost	0	.9228	.8109	.8632	n/a

Macro-averaged Performance

	iterations	maPre	maRec	maF1	improvement
AdaBoost	10	.6662	.5689	.6027	+17.1%
Resampling Boost	5	.6612	.5758	.6045	+17.4%
NoBoost	0	.8230	.4708	.5147	n/a

miPre = micro-averaged precision; maPre = macro-averaged precision

miRec = micro-averaged recall; maRec = macro-averaged recall

miF_1 = micro-averaged F_1; maF_1 = macro-averaged F_1

Table 1: Results summary of SVMBoost on Reuters-21578

The most noticeable improvement by SVMBoost over baseline SVMs (without boosting) is in the macro-averaged F_1 scores, suggesting that boosting is more effective for improving the decision boundaries by SVMs for rare categories than it does for common categories. For the macro-averaged performance scores, the only results previously reported on this collection were those by Yang & Liu[27] who obtained a F_1 score of 0.525 for SVMs without boosting. That score is very similar to our baseline SVMs result, and significantly worse than the result (0.6045 in F_1) of SVMs with *resampling* boost. More interestingly, both the curves for AdaBoost and *resampling* boost decline when the number of iterations exceed a certain value, suggesting that boosting can indeed lead to overfitting, at least with strong classifiers.

DtreeBoost The C4.5 decision trees algorithm [17] is used for the experiment. The software package we employed is the Weka Data Mining Toolkit[6]. All parameters were left at default values except otherwise specified. For feature selection, we selected 500 words with the highest Chi-squared score [28]. We tested only the AdaBoost version for decision trees because the *resampling* boost on the same collection has been explored by Weiss[25]. The results of DtreeBoost are shown in Figure 2 and Table 2.

The micro-averaged score of our baseline (0.8099 in F_1) is very similar to the scores (0.789 in break-even point) reported by Apte for Decision Trees (C4.5)[25] on Reuters-21450, an older version of the Reuters-21578 dataset. Our experiment results shows only a slight improvement for boosting with decision trees using AdaBoost in micro-averaged score.

The impressive improvement by boosting decision trees in the macro-averaged F_1 scores (13.4%) strengthens the point that boosting is more effective in correcting the inductive bias of rare categories than common categories, which was

[4] Iteration 0 corresponds to individual classifier without boosting.

[5] IBMBoost refers to resampling boost in these figures.

[6] We have also tried the C5.0 software package and the results we got are very similar to the ones with Weka software package.

Micro-averaged Performance

	iterations	miPre	miRec	miF1	improvement
AdaBoost	10	.7863	.8472	.8156	+0.8%
NoBoost	0	.7829	.8389	.8099	n/a

Macro-averaged Performance

	iterations	maPre	maRec	maF1	improvement
AdaBoost	5	.4780	.5643	.5176	+13.4%
NoBoost	0	.4183	.5017	.4562	n/a

Table 2: Results summary of DTreeBoost on Reuters-21578

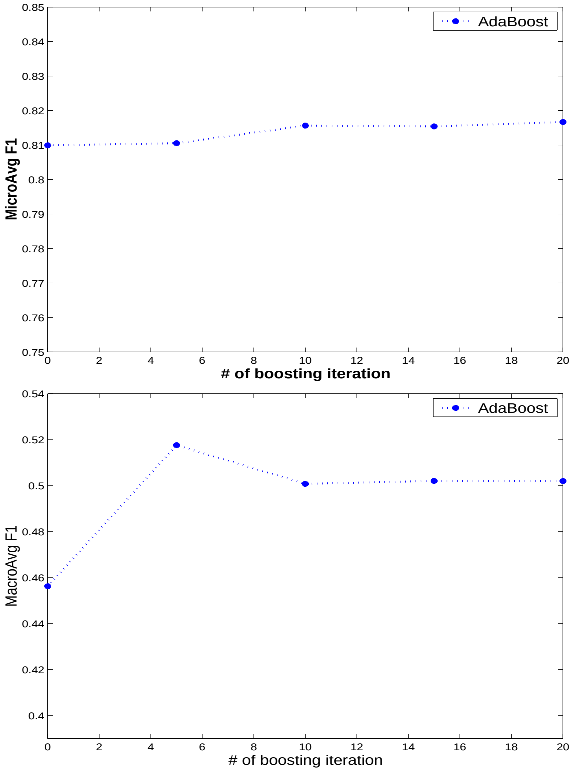

Figure 2: Performance of DtreeBoost on Reuters-21578

also suggested by the results of boosting with SVMs.

NBBoost For Naive Bayes, we used McCallum's Rainbow system. There are two models for naive bayes in Rainbow: the *multinomial mixture model* and the *multi-variate Bernoulli model*. In our experiment we use the *multinomial model* which is tested better than the *Bernoulli* model [14]. We tuned the feature-set size and selected 2000 features to optimize the $F1$ measure. The results of NBBoost are shown in Figure 3 and Table 3. We compared our results with the ones obtained by Joachims[8]. For the ten most common categories, the micro-F1 of our method is comparable or better than Joachims'; as for the averaged micro-F1 over the whole corpus, ours (0.7908) is significantly higher than the NB score(0.720) obtained by Joachims. Our averaged micro-F1 on the ten most common categories is also comparable to the results by McCallum at al [14].

Micro-averaged Performance

	iterations	miPre	miRec	miF1	improvement
AdaBoost	5	.8936	.7202	.7977	+1.0%
Resampling Boost	10	.8920	.7189	.7961	+1.0%
NoBoost	0	.8006	.7813	.7908	n/a

Macro-averaged Performance

	iterations	maPre	maRec	maF1	improvement
AdaBoost	5	.7636	.2811	.3362	-8.0%
Resampling Boost	10	.8205	.2986	.3558	-2.7%
NoBoost	0	.4499	.3517	.3658	n/a

Table 3: Results summary of NBBoost on Reuters-21578

From Table 3 we can see that boosting achieved only marginal improvement with respect to micro-F1 measure while the macro-F1 got worse than without boosting. Similar observation that boosting Naive Bayes does not work well for some datasets has been reported before. Compared with SVMs and Decision Trees, the different behavior of boosting with NB for rare categories, might be related to the fact that Naive Bayes uses a generative model for classification. Boosting NB will change the distribution model instead of modifying the decision boundary gradually. However, to thoroughly explain this result we might need further research such as running the same experiments on some other datasets.

Micro-averaged Performance

	iterations	miPre	miRec	miF1	improvement
AdaBoost	10	.8851	.8066	.8440	-0.4%
Resampling Boost	20	.8740	.8040	.8375	-1.2%
NoBoost	0	.8731	.8231	.8474	n/a

Macro-averaged Performance

	iterations	maPre	maRec	maF1	improvement
AdaBoost	15	.7249	.5966	.6088	+2.9%
Resampling Boost	3	.7072	.5860	.5953	+1.0%
NoBoost	0	.7288	.5681	.5914	n/a

Table 4: Results summary of RocchioBoost on Reuters-21578

RocchioBoost All the parameters were tuned using 5-fold cross validation on the original training set before boosting. Parameters in Rocchio were set to be: $p_{max} = 1000$ (the maximum number of features with non-zero weights in the class centroid), $n = 1000$ (the size of the "query-zone" in terms of the number of the documents), $\gamma = 1.0$ (the weight for the negative centroid), $ft_{mi} = 2000$ (the number of features selected for maximizing the micro-averaged F_1 performance of the system), and $ft_{ma} = 2000$ (the number of features selected for maximizing the macro-averaged F_1 performance of the system). The results of RocchioBoost are shown in Table 4. From the results, we can see that there is no significant trend for boosting Rocchio (increase or decrease) in terms of both Micro-averaged and Macro-averaged measures.

5.3 Discussion

The observation that boosting is effective to correct the inductive bias of SVMs and Dtree for rare categories, but not the other two classifiers is interesting. To see this improvement in detail, table 5 compares the results of ten

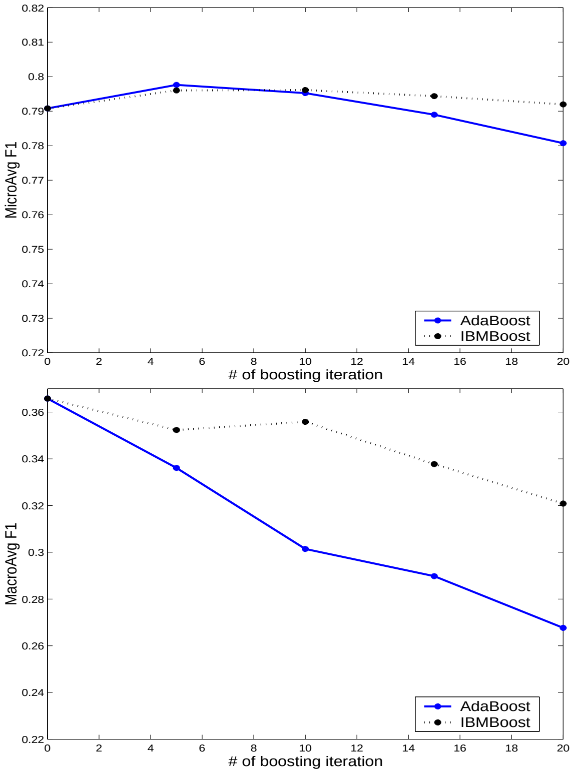

Figure 3: Performance of NBBoost on Reuters-21578

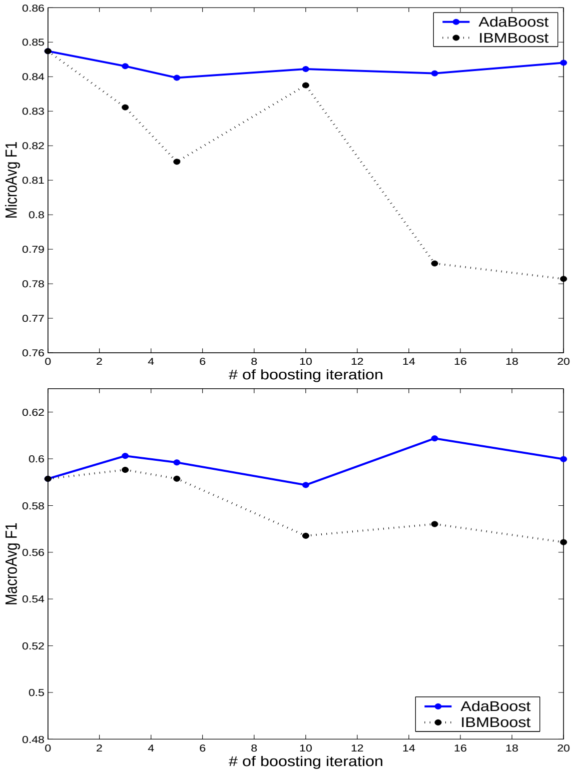

Figure 4: Performance of RocchioBoost on Reuters-21578

most common categories and ten rare categories for SVMs without boosting and SVMBoost in terms of F1 measure.[7] From the table, we can see that boosting is indeed effective to improve the performance for rare categories. Although most classifiers did comparably well for common categories, boosting can make better use of the training examples when the training data are not enough. But why this positive effects of boosting does not work for NB and Rocchio? One reason might be that boosting implicitly requires the instability of classifiers induced by base classifiers[16]. As a generative model, Naive Bayes is more stable than the discriminative classifiers such as Dtree [2]. Small changes of the training data will seldom if ever result in big changes to the estimated probabilities. The Rocchio algorithms build the prototype for each category beforehand, which is also relative stable. This is especially the case for the rare categories that have only one or two training documents because the range that the prototype can change is small. Therefore, boosting NB and Rocchio may not be able to generate multiple models with sufficient diversity, thus cannot effectively correct the inductive bias of the base classifiers.

Performance Comparison in F1 measure

Common Categories	SVMs	SVMBoost	Rare Categories	SVMs	SVMBoost
earn	.9980	.9834	sun-meal	0	1.000
acq	.9645	.9678	naphtha	0	.4000
money-fix	.7331	.7393	potato	.5000	.8000
grain	.9143	.9300	propane	0	.8000
crude	.8432	.8526	cpu	0	1.000
trade	.7366	.7354	coconut	.6667	.6667
interest	.7456	.7737	coconut-oil	0	.4000
ship	.8077	.8092	platinum	0	.4000
wheat	.8444	.8345	instal-debt	1.000	1.000
corn	.8889	.8624	sun-oil	0	.5000

Table 5: Results of SVMs and SVMBoost on Ten Most Common Categories and Ten Rare Categories

6. CONCLUSIONS

This paper studies the effects of boosting applied to different classification methods for text categorization, including Decision Trees, Naive Bayes, Support Vector Machines (SVMs) and a Rocchio-style classifier. We identify the inductive biases of these classifiers and explore how boosting reacts to those biases.

The main observations of our experiments are: (1) In general, boosting is not significantly effective to improve the performance of the base classifiers on common categories. One point worth noticing is that boosting SVMs works the best among the four cases while boosting NB does by far the worse. By Boosting SVMs we obtained comparable results in the micro-averaged F_1 measure as the best one ever reported on this benchmark collection with boosting. (2) The effect of boosting for rare categories varies across classifiers:

[7] The rare categories are selected as follows: we sort the categories in ascending order and select the ten first categories, while omitting those for which both F_1 results of SVMs and SVMBoost are zero.

for SVMs and Decision Trees, we achieve 13-17% performance improvements by boosting in the macro-averaged F_1 measure; however, boosting with Rocchio and Naive Bayes did not obtain substantial improvement. Some analysis is given to explain this interesting observation.

Most current work on boosting focuses on common categories. We hope our exploration of both rare and common categories will shed light to a deeper understanding of boosting and furthermore contribute to solving the problem of how to optimally combine different classifiers.

7. ACKNOWLEDGMENTS

We thank Fan Li who helped to prepare the baseline results of the Reuters-21578 corpus. The work presented in this paper was funded in part by the National Sceince Foundation under their KDI program, award #SBR-9873009. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect those of the sponsor.

8. REFERENCES

[1] C. Apte, F. Damerau, and S. Weiss. Towards language independent automated learning of text categorization models. In *Proceedings of the 17th Annual ACM/SIGIR conference*, 1994.

[2] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. pages 292–300, London, 1994. Springer-Verlag.

[5] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[6] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[7] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting, 1998.

[8] T. Joachims. Making large-scale svm learning practical. In *Advances in Kernel Methods - Support Vector Learning*, Cambridge, USA, 1998. M.I.T. Press.

[9] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *European Conference on Machine Learning (ECML)*, pages 137–142, Berlin, 1998. Springer.

[10] T. Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML'99)*, San Francisco, CA, 1999. Morgan Kaufmann.

[11] G. Lebanon and J. Lafferty. Boosting and maximum likelihood for exponential models. In *Neural Information Processin Systems (NIPS)*, 2001.

[12] D. Lewis, F. Li, T. Rose, and Y. Yang. The reuters corpus volume i as a text categorization test collection. In *SIGIR 2002 (submitted).*

[13] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *SIGIR '95: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–254, New York, 1995. Association for Computing Machinery.

[14] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[15] T. Mitchell. *Machine Learning.* McGraw Hill, 1996.

[16] J. R. Quinlan. Bagging, boosting, and c4.5. *Proceedings of the 13th National Conference on Artifitial Intelligence on Machine Learning*, pages 322–330.

[17] J. R. Quinlan. C4.5: Programs for machine learning. Morgan Kaufmann.

[18] M. Rogati and Y. Yang. High performing and scalable feature selection for text categorization. In *AAAI-02 (submitted)*, 2002.

[19] G. Salton, editor. *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[20] R. Schapire and Y. Singer. Boosttexter: Aboosting-based system for text categorization. In *Machine Learning*, volume 39(1/3), pages 135–168, 2000.

[21] R. E. Schapire. Theoretical views of boosting and applications. In *Proc. 10th International Conference on Algorithmic Learning Theory - ALT '99*, volume 1720, pages 13–25. Springer-Verlag, 1999.

[22] R. E. Schapire, Y. Singer, and A. Singhal. Boosting and rocchio applied to text filtering. In *Proceedings of the Twenty-first Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 215–223, New York, 1998. The Association for Computing Machinery.

[23] L. G. Valiant. A theory of the learnable. In *ACM Symposium on Theory of Computing*, pages 436–445, 1984.

[24] C. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.

[25] S. Weiss, C. Apte, F. Damerau, D. Johnson, F. Oles, T. Goets, and T. Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems, Special Issue on Applications of Intelligent Information Retrieval*, 14(4):63–69, 1999.

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

[27] Y. Yang and X. Liu. A re-examination of text categorization methods. In *The 22th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 42–49, 1999.

[28] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In J. D. H. Fisher, editor, *The Fourteenth International Conference on Machine Learning (ICML'97)*, pages 412–420. Morgan Kaufmann, 1997.