Boosting Naïve Bayesian Learning on a Large Subset of MEDLINE®

W. John Wilbur, MD, PhD

National Center for Biotechnology Information (NCBI)
National Library of Medicine, Bethesda, MD 20894

We are concerned with the rating of new documents that appear in a large database (MEDLINE) and are candidates for inclusion in a small specialty database (REBASE®). The requirement is to rank the new documents as nearly in order of decreasing potential to be added to the smaller database as possible, so as to improve the coverage of the smaller database without increasing the effort of those who manage this specialty database. To perform this ranking task we have considered several machine learning approaches based on the naïve Bayesian algorithm. We find that adaptive boosting outperforms naïve Bayes, but that a new form of boosting which we term staged Bayesian retrieval outperforms adaptive boosting. Staged Bayesian retrieval involves two stages of Bayesian retrieval and we further find that if the second stage is replaced by a support vector machine we again obtain a significant improvement over the strictly Bayesian approach.

INTRODUCTION

Many methods of machine learning have been tested on document classification tasks and we would refer the reader to several articles that summarize some of the recent work in the field[1-4]. Virtually all of this work has been done on data sets consisting of no more than a few thousand documents. However, we work with large data sets (as many as eleven million documents, the current size of MEDLINE). In such large sets neural nets, classification trees, rule-based systems, or support vector machines are difficult or impossible to train in realistic time. On the other hand naïve Bayes is a very efficient machine learning method, perhaps the most efficient method currently available[5]. Our aim has therefore been to find ways of improving the performance of naïve Bayes with as little decrement in efficiency as possible. Elkans[5] has shown that Adaboost[6] can be used to improve on naïve Bayes, with naïve Bayes as the weak learner. We find that a different form of boosting can yield even greater improvement.

A word is in order about our method of evaluation. All of our methods produce continuous scores rather than binary classifications. This is important because our purpose is to produce a ranking of the test set with the positive examples as near the top of the ranking as possible. All of our test sets are constructed to have one hundred positive examples and we rate a method by its average precision on the top one hundred ranks. This form of scoring is done because we expect a user of our system will not examine more than about one hundred documents and we wish to optimize for the relevant material they will find in that interval. Also we have only studied the particular data set on which our application is based. This too is motivated by practical considerations. Most methods of machine learning that have been published have shown good performance on some data set and it is abundantly clear that what works well on one data set may not work as well on another. We have therefore studied the data that is most pertinent to our application.

EVALUATION METHOD

The version of REBASE (a restriction enzyme database) we study consists of 3,121 documents comprising titles and abstracts mostly taken from the research literature. The majority (all but 692) of these documents are contained in the MEDLINE database and have Medical Subject Headings (MeSH®) assigned to them. Each REBASE document was used as a query in a form of vector cosine retrieval to obtain from the MEDLINE database the approximately 200 closest neighbor documents (scores were required to be greater than 0.1). When this set was pooled 117,476 MEDLINE documents were obtained that did not belong to REBASE but all had some level of similarity to at least one document in REBASE. We denote this set by NREBASE as they consist of documents that have already been rejected for membership in REBASE. Thus REBASE provides positive examples and NREBASE negative examples of what we wish to learn by machine methods.

In order to test the results of learning a form of cross validation was used. One hundred documents were randomly selected from REBASE, denoted RTEST, and the remainder denoted RTRAIN. The same fraction of NREBASE (3,764 documents) was likewise randomly selected for testing and denoted NRTEST while the remainder were denoted NRTRAIN. The training and test sets were thus composed as

$$\begin{aligned} &TRAIN = RTRAIN \cup NRTRAIN \\ &TEST = RTEST \cup NRTEST \end{aligned} \qquad (1)$$

This whole procedure was repeated one hundred times to produce a set of one hundred training and testing pairs $(\mathrm{TRAIN}_i, \mathrm{TEST}_i)_{i=1}^{100}$. This collection was used throughout our study for the evaluation of all learning methods. Any particular method was applied to learn on TRAIN_i and the results of learning were applied to rank TEST_i. The result of ranking TEST_i was computed as the fraction of the first one hundred ranks filled with members of RTEST_i, i.e., the precision, $prec_i$, over the top one hundred ranks. The figure of merit for a particular method was then the average over all numbers $\{prec_i\}_{i=1}^{100}$.

To test the significance of the difference between two learning methods a bootstrap method was used[7]. This method is a shift method[8] based on re-sampling the training-testing pairs and comparing the two methods on each re-sampling. This re-sampling is done 10,000 times in our case. This results in a significance test well able to detect differences significant at the 1% level.

ALGORITHMS

Throughout our discussion we deal with documents and each document is preprocessed into a list of key terms. This preprocessing step involves extracting individual words from titles and abstracts and discarding those on a list of 310 common stop words. No stemming is done. We have then considered two options. Option SINGLE in which all single nonstop terms and all MeSH terms are taken as the set of key terms to represent a document. Alternatively, option DOUBLE in which all single nonstop terms, and all adjacent pairs of nonstop terms without punctuation between, together with all MeSH terms are taken as the set of key terms to represent a document. Given a particular key term representation, a document is considered to have an attribute corresponding to each key term occurring in the database of all documents. The value of the attribute corresponding to a particular key term is 1 if the term is in the document and 0 otherwise. Thus each document is represented by a long vector consisting mostly of zeroes, but sparsely populated with 1's corresponding to the terms that actually occur in that document.

Naï ve Bayes

The naïve Bayesian algorithm is based on the assumption that the values of attributes are distributed independently within the classes to be learned. Thus each term can be weighted separately based on its distribution in the training set. One scores documents in the test set by summing the weights of the terms they contain and then ranks the documents based on the resultant scores. For details the reader may consult[9-11]. We have found that performance is improved with this learner if we remove from the scoring any terms with weights less than 1.5 and we will refer to this as feature selection (weight > 1.5). Such feature selection (weight>1.5) is used implicitly in all the applications of naïve Bayes in the following algorithms.

Adaptive boosting of naï ve Bayes

We have implemented the Adaboost algorithm of Freund and Shapire[6] with the naïve Bayesian algorithm as the weak learner. The general form of the algorithm is as follows.

Input: Sequence of training examples $\{(x_i, y_i)\}_{i=1}^{N}$, y_i the label of x_i;

Initial normalized set of weights $\{w_i^1\}_{i=1}^{N}$ assigned to the training examples;

WeakLearn, a weak learning algorithm;

Integer T specifying the number of iterations.

Do for $t = 1, \ldots, T$

1. Normalize the weights to probabilities, $\vec{p}^t = \vec{w}^t / \sum_{i=1}^{N} w_i^t$
2. Call WeakLearn, providing it with $\vec{p}^t$ and $\{(x_i, y_i)\}_{i=1}^{N}$, and get in return a hypothesis $h_t : X \rightarrow [0,1]$
3. Calculate the error $\varepsilon_t = \sum_{i=1}^{N} p_i^t |h_t(x_i) - y_i|$
4. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
5. Update the weight vector by $w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i) - y_i|}$, $i = 1, \ldots, N$

Output: combined hypothesis

$$h_f(x) = \sum_{t=1}^{T} \left(log \frac{1}{\beta_t} \right) h_t(x) / \sum_{t=1}^{T} \left(log \frac{1}{\beta_t} \right)$$

The combined hypothesis can be used for ranking or for categorization by defining

$$category(x) = \begin{cases} 1, & \text{if } h_f(x) \geq 1/2 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

In the category form Freund and Shapire[6] show that the error rate of the combined hypothesis obeys the bound

$$\varepsilon \leq 2^T \prod_{i=1}^{N} \sqrt{\varepsilon_t (1 - \varepsilon_t)}. \quad (3)$$

The Adaboost algorithm leaves ambiguity at two points and we must fill in the details. First, there is the necessity to define the function h_t on all training and testing documents at each iteration of the algorithm. Our procedure for defining h_t is as follows. On a given iteration when the weak learner has been trained it is used to assign a score to each training document. The scores are used to rank the training documents. We then apply the pool adjacent violators algorithm[12, 13] to find that probability distribution $prob_t$ which gives the probability that a document is in RTRAIN, which is non-decreasing as a function of score, and which assigns maximal likelihood to the training data. For any training example x_i we then define

$$h_t(x_i) = prob_t(score(x_i)) \tag{4}$$

For testing we must also be able to compute $h_t(x_j)$ when x_j is a test document. For this purpose we define

$$ms = Max_{x_i \in \text{TRAIN}}\, score(x_i). \tag{5}$$

Then we define a function f_t on the test set scores by

$$f_t(score(x_j)) = \begin{cases} ms, \text{ if } ms < score(x_j), \text{ else} \\ \text{Min}\{score(x_i) / x_i \in \text{TRAIN} \; \& \\ score(x_i) \geq score(x_j)\} \end{cases} \tag{6}$$

The function f_t assigns a training set score to each test set score and this allows us to extend h_t to the test set documents by

$$h_t(x_j) = prob_t(f_t(score(x_j))). \tag{7}$$

The second place in the iterative cycle where we must define behavior is after the weights have been updated. At this point it is left open how one selects a subset of the training set on which the weak learner is trained in the next cycle of the algorithm. The only consideration is to try to obtain a low error, ε_t, in the next step of the algorithm. Our procedure at this point is to rank all training documents in order so that the errors of the just completed predictions form an increasing sequence $\{|h_t(x_i) - y_i|\}$ (y_i is the label we are trying to predict, 1 if in REBASE, 0 if in NREBASE). The weights have already been normalized at this point to form the probabilities $\{p_i^t\}$ and we reorder these probabilities to correspond to the ranking based on the errors just described. The sum of the $\{p_i^t\}$ is 1 and we choose the largest J such that

$$\sum_{i=1}^{J} p_i^t \leq 0.2 \tag{8}$$

and the set $\{x_i\}_{i=1}^{J}$ is removed from TRAIN and the Bayesian learner is trained on the remainder in the next iteration.

Staged naï ve Bayes

We first train naïve Bayes on the whole training set and score both the whole training set and the whole test set. This scoring completes stage 1. For stage 2 we select from NRTRAIN just those documents that have scores > log(19) and denote this set by NRTRAIN*. The use of log(19) is motivated by the following argument. If one assumes a neutral prior and that the naïve Bayesian model perfectly fits the data, then Bayes stage 1 scores yield a prediction that a document is in RTRAIN with probability

$$p(x \in \text{RTRAIN}) = 1/\left(1 + e^{-score_1(x)}\right). \tag{9}$$

It is then not difficult to show that $p(x \in \text{RTRAIN}) \geq 0.95$ if and only if $score_1(x) \geq \log(19)$ where log is the natural log. This works well in our setting given the relative sizes of the sets RTRAIN and NRTRAIN. With a different data set the procedure may require some modification for best results. Typically NRTRAIN* has about 2000 documents in it. They represent those documents in NRTRAIN that based on stage 1 scoring have a high probability of belonging in RTRAIN.

We next train naïve Bayes on RTRAIN∪NRTRAIN*. The ranking of TEST is now done in two steps. First TEST is divided into two disjoint sets: TEST# which consists of those documents whose stage 1 scores are less than or equal to log(19) and the remainder, TEST*. Members of TEST# are ranked by their stage 1 scores. Members of TEST* are ranked by the combination score1+3*score2 of the score from stage 1 and the score from stage 2. Finally, all members of TEST* are ranked above the highest scoring member of TEST#. Here the factor 3 in the formula score1+3*score2 is empirically determined. It was chosen to give the best results. However, the results are not very sensitive to its precise value.

Staged Bayes-Svm

The first stage here is identical to Staged Bayes just described and the selection of the set NRTRAIN* is done in the same way using the score cutoff log(19). In the second stage, in place of naïve Bayes, we train a linear support vector machine on the set RTRAIN∪NRTRAIN*. The resultant scores are then used as stage 2 scores to rank the test set just as in staged Bayes, with the exception that we use score1+10*score2, instead of score1+3*score2 in ranking TEST*. The value 10 was chosen to offset the smaller size of the scores coming from the linear support vector machine as compared with naïve Bayes trained on the same second stage sets.

We use Platt's[14, 15] sequential minimal optimization method of training support vector machines. We have followed the author in taking the error tolerance for the KKT conditions to be 0.01. We tested 1.0, 0.5, 0.1, and 0.05 as values for C (the bound on Lagrange multipliers) and found that all gave close to the same results with 0.1 and 0.05 as the best and essentially equivalent. The value 0.01 gave a decreased performance by 0.5%. In the work reported here we use a C of 0.1. In order to obtain efficiency in training we prune the set of attributes by using a chi-square criterion[2]. Each term is assigned the chi-square value coming from the contingency table relating RTRAIN versus NRTRAIN and term-present versus term-absent. Those terms are retained that have a chi-square value greater than 3.84. The number 3.84 is chosen as the 95% confidence limit that terms actually have a distribution correlated with the division RTRAIN/NRTRAIN we desire to learn. The result is roughly from 800 to 900 terms coming from the 5,000 training documents. With the specifics stated here, a typical training run requires about 40 minutes on a sun ultra 10 processor. This must be repeated for each of the one hundred training-testing pairs in order to rate a particular choice of parameters.

RESULTS

The average precision over the top one hundred ranks based on the one hundred test sets are contained in Table 1 for the different methods tested.

Table 1. Test results for the six different algorithms examined.

Algorithm	*Average precision over top 100 ranks*
Naïve Bayes, SINGLE	71.4%
Naïve Bayes, DOUBLE	74.6%
Naïve Bayes, DOUBLE, weights>1.5	76.5%
Adaboost, naïve Bayes as WeakLearn	77.8%
Staged naïve Bayes	78.9%
Staged naïve Bayes-Svm	80.0%

The algorithms are listed in order of increasing effectiveness in Table 1. We used the bootstrap method mentioned in the EVALUATION METHOD section to test all consecutive pairs of algorithms and found that all differences are significant at the 1% level.

DISCUSSION

Feature selection is an important issue in applying a learning method. Bayesian learning has commonly been done with single words. Dumais, et al.[1], experimented with the addition of syntactic phrases, but found no benefit. Here we have included all contiguous word pairs without punctuation or stop words in moving from SINGLE to DOUBLE and see a strong benefit. This greatly increases the number of possible features from 265,234 to 1,562,939 for the 120,597 documents. Only about 28,000 features actually receive nonzero weights in the SINGLE case and 75,000 in the DOUBLE case. When we remove weights that fall below 1.5 in absolute value the 75,000 in the double case are reduced to about 35,000 weighted terms. In this way we obtain a richer representation of the documents and the terms are selected for significance. The result is improved performance and it is based on a purely statistical phrase selection. We are led to hypothesize that we are seeing some of the benefit that Cohen and Singer[4] have found with RIPPER and sleeping experts for phrases which incorporate features more complex than single words.

The implementation of Adaboost that we describe in the ALGORITHMS section is based on naïve Bayes, DOUBLE, weight>1.5. It is actually quite efficient. The error at each retraining is on the order of 0.01 and this leads to a decrease in the error limit[6] on the training set by a factor of 0.2 for each iteration. After eight iterations the error on the training set must go to zero, but we find slightly better performance by stopping the training after six iterations. One could hope for a greater improvement in performance from the use of Adaboost. Elkans[5] reports results of Adaboost on two data sets. On the "German credit" data set (1000 examples) he observed a decrease in error rate from 25.1% to 24.0% after three rounds of boosting and on the "Diabètes in Pima Indians" data set (200 examples) error decreased from 20.3% to 18.7% after ten rounds of boosting. Though these data sets are much smaller than ours, the improvements seen with boosting seem quite comparable to our results. Clearly the problem is not difficulty learning the training set, but rather an inability to generalize that learning to the test set. It appears that the combined hypothesis of Adaboost is simply too complex to generalize as well as we would desire.

The essence of boosting is to train and evaluate the training and then train again focusing on the problem cases that were found on the previous training. Staged Bayes does this in two steps. The first training is routine. The second training involves all the positive examples and about 2,000 negative examples that the first training has been unable to separate from the positive examples. Depending on the composition of the training set, the exact composition of the set on which the second training takes place could vary. In our case we keep all the positive examples because they are relatively few in number. There are two reasons, which suggest themselves as to why staged Bayes improves on our version of Adaboost. First, with only two training cycles the resultant combined hypothesis is much simpler and this could lead to improved generalization. Second, Adaboost trains repeatedly on subsets of the original training set and then these hypotheses take part in the scoring for any member of the test set. On the other hand our approach in staged Bayes is to perform the second training on a select subset and then to apply the criterion used to select that subset also to the test set. In this way we restrict the scoring done based on the second training to that part of the test set most like the examples used in its training. We believe this could result in improved performance.

Having found that Staged Bayes leads to desirable performance, it is a short step to substitute a support vector machine for the second stage Bayes learning. This still preserves a good level of efficiency on our data set, because the training set only consists of about 5,000 documents. This makes good intuitive sense, because the Bayesian first stage learning is used to remove from further consideration training examples that are so far from the margin between positive and negative examples that they would not likely function as support vectors in the training of a support vector machine even if applied to the whole training set[16].

In future work we hope to investigate whether one may use more stages in Staged Bayesian retrieval to further improve performance and whether the technique can be better understood theoretically.

REFERENCES

1. Dumais S, Platt J, Heckerman D, Sahami M. Inductive learning algorithms and representations for text categorization. In: Gardarin G, French J, Pissinou N, Makki K, Bouganim L, eds. Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management. Bethesda, MD: ACM Press, 1998:148-155.
2. Yang Y, Pedersen JO. A comparative study on feature selection in text categorization. Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97), 1997:412-420.
3. Lewis DD, Schapire R, Callan JP, Papka R. Training algorithms for linear text classifiers. In: Frei H-P, Harmon D, Schauble P, Wilkinson R, eds. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Zurich: ACM Press, 1996:298-306.
4. Cohen WW, Singer Y. Context-sensitive learning methods for text categorization. In: Frei H-P, Harmon D, Schauble P, Wilkinson R, eds. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Zurich: ACM Press, 1996:307-315.
5. Elkans C. Boosting and naive Bayesian learning. La Jolla, California: University of California, San Diego, 1997:
6. Freund Y, Schapire RE. A decision-theoretic generalization of on-line learning and an application to boosting. Murray Hill, New Jersey: AT & T Research, 1995:
7. Wilbur WJ. Nonparametric significance tests of retrieval performance comparisons. Journal of Information Science 1994;20(4):270-284.
8. Noreen EW. Computer Intensive Methods for Testing Hypotheses. New York: John Wiley & Sons, 1989.
9. Langley P. Elements of Machine Learning. San Francisco: Morgan Kaufmann Publishers, Inc., 1996.
10. Langley P, Iba W, Thompson K. An analysis of Bayesian classifiers. Tenth National Conference on Artificial Intelligence. San Jose: AAAI Press, 1992:223-228.
11. Langley P, Sage S. Induction of selective Bayesian classifiers. Tenth Conference on Uncertainty in artificial intelligence. Seattle, WA: Morgan Kaufmann, 1994:399-406.
12. Hardle W. Smoothing techniques: with implementation in S. New York: Springer-Verlag, 1991.
13. Ayer M, Brunk HD, Ewing GM, Reid WT, Silverman E. An empirical distribution function for sampling with incomplete information. Annals of Mathematical Statistics 1954;26:641-647.
14. Platt J. How to implement SVMs. IEEE Intelligent Systems 1998(July/August):26-28.
15. Platt JC. Fast training of support vector machines using sequential minimal optimitzation. In: Scholkopf B, Burges CJC, Smola AJ, eds. Advances in Kernal Methods. Cambridge, Massachusetts: The MIT Press, 1999:185-208.
16. Burges CJC. A tutorial on support vector machines for pattern recognition. Bell Laboratories, Lucent Technologies, 1999:1-43.