Efficient Handling of High-Dimensional Feature Spaces by Randomized Classifier Ensembles

Aleksander Kołcz
Personalogy, Inc.
3365 Orion Drive
Colorado Springs, CO 80906
a.kolcz@ieee.org

Xiaomei Sun
Department of Computer Science, University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, CO 80918
maysun1998@yahoo.com

Jugal Kalita
Department of Computer Science, University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, CO 80918
kalita@pikespeak.uccs.edu

ABSTRACT

Handling massive datasets is a difficult problem not only due to prohibitively large numbers of entries but in some cases also due to the very high dimensionality of the data. Often, severe feature selection is performed to limit the number of attributes to a manageable size, which unfortunately can lead to a loss of useful information. Feature space reduction may well be necessary for many stand-alone classifiers, but recent advances in the area of ensemble classifier techniques indicate that overall accurate classifier aggregates can be learned even if each individual classifier operates on incomplete "feature view" training data, i.e., such where certain input attributes are excluded. In fact, by using only small random subsets of features to build individual component classifiers, surprisingly accurate and robust models can be created. In this work we demonstrate how these types of architectures effectively reduce the feature space for sub-models and groups of sub-models, which lends itself to efficient sequential and/or parallel implementations. Experiments with a randomized version of Adaboost are used to support our arguments, using the text classification task as an example.

1. INTRODUCTION

Many challenges faced in data mining are due to the huge amounts of data available, which presents serious scaling problems for many machine learning algorithms. Although commonly the problem is caused by excessively many data entries, in domains such as text it is compounded by the very high dimensionality of the data. Much research has been directed at data pre-processing to eliminate irrelevant examples and/or features, and for high-dimensional problems a feature selection stage is often performed to reduce the number of attributes. Unfortunately, such approaches may lead to a loss of useful information [19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGKDD '02 Edmonton, Alberta, Canada
Copyright 2002 ACM 1-58113-567-X/02/0007 ...$5.00.

Recent advances in the area of machine learning resulted in architectures such as Support Vector Machines [17], that can handle large feature spaces without overfitting, so that the aspect of feature-space reduction becomes less of an issue. Scalability is a problem, however, and many datasets are too large to be processed efficiently if an algorithm requires access to all data during the learning process. Ensemble classifier techniques are attractive in this respect, since robust and accurate classifier aggregates can be learned even if each individual component classifier operates on incomplete training data, i.e., such where certain training instances and/or input attributes are eliminated, often at random [8].

An under-emphasized property of certain randomized classifier ensembles, such as Random Forests [8] or Randomized Boosting [3], is that each individual classifier may be induced using just a fraction of the available features. In this work we discuss how this characteristic can be used to build resource-efficient classifiers in situations where the number of features is very high. To focus our presentation, we will assume that the original high-dimensional dataset is too large to fit into the memory (RAM) space of a computing process, but it is possible to do so with a reduced number of features (i.e., the related problem of excessive numbers of data entries is ignored). We use the text classification task and a boosting algorithm to illustrate the effectiveness of our approach.

The paper is organized as follows: In Section 2 we briefly outline the context of the ensemble randomization ideas used in our work. Section 3 demonstrates how the architecture of Random Forests, when combined with depth constrained tree models, leads to an implicit feature subset selection, which has important implications for inducing models from high dimensional data. In Section 4 we discuss the applicability of random feature subset selection to boosting. Section 5 outlines the experimental setup, the results of which are given in Section 6. The paper is concluded in Section 7.

2. BACKGROUND

Ensemble classifiers have been quite popular in many data mining applications due to their high accuracy and potential for efficient parallel implementations. The success of these techniques rests largely on their ability to combine many weakly coupled models which are not very accurate when taken individually but which, as an aggregate, greatly reduce the overall generalization error rate.

Historically, a number of seemingly diverse ensemble schemes have been proposed, but recently efforts have been made to explain their properties under within a common framework, which can be viewed as statistical sampling from a model space where adequate model coverage is achieved via appropriate randomization of the learning set and/or the learning process [8][3]. Decision trees have been used predominantly as the ensemble base learner, and approaches based on random perturbations of the training set (bagging [6] and arcing [7]) received particularly much attention. Adaboost [11], a technique related to arcing that does not explicitly depend of randomization, has been especially well acclaimed.

Dietterich [9], instead of randomizing the training set, randomized the process of tree construction whereby, when deciding how to best split a tree node, all possibilities are ranked according to their utility, and a split is chosen at random from a small set of the most promising candidates[1]. Note that such an approach requires a complete training set (with all features) to be present for splitting each node of the tree. A different variant of randomization, which relaxes this constraint, is due to Amit et al. [3] and Breiman [8]. In [3] it is suggested to built each tree using random subsets of all possible attribute–threshold combinations when splitting each node of the tree. Breiman [8] proposed a similar process, where a random subset of attributes is selected prior to executing each split (the two approaches are equivalent when all features are binary). In both [3] and [8] the authors also propose unifying frameworks for different types of classifiers ensembles (under the name of Multiple Randomized Classifiers (MRCL) in [3] and under the term of Random Forests in [8]).

The ideas outlined in this paper are directly based on the types of randomization proposed by Amit et al. [3] and Breiman [8]. In the general discussion we will focus on the architecture of Random Forests, whose algorithmic implementation clearly puts emphasis on random selection of input features. We will then treat Randomized Boosting of [3] as a natural extension of Random Forests, and use Randomized Boosting as the platform for our experiments.

It is interesting to note that the idea of using random subsets of features in classifier system design has a long history. Bledsoe and Browining [5] applied it to built optical character-recognition systems using random access memories, by randomly interconnecting the address lines of each memory module to the image matrix. Their architecture became known as the N-tuple network [15] (since N randomly selected features were utilized by each model), and has been used with success in several pattern recognition applications (e.g., [1][2]). As will be shown, under certain circumstances, the new architectures such Random Forests can be seen to extend the basic N-tuple idea.

3. RANDOM FORESTS AS FEATURE SUBSET SELECTORS

3.1 Single-model effects

Although the definition of a Random Forest, as stated in [8], is quite general and has much in common with that of MRCL, we focus on its special instance Forest-RI, which is one of algorithmic embodiments of a Random Forest proposed by Breiman. We will thus narrowly understand a Random Forest as a classifier ensemble that consists of a number of trees, each grown without pruning such that only a random subset of F features is "seen" when splitting each node of the tree. For a given input, the Random Forest ensemble produces its output via uniform voting to choose the most popular class.

We begin with an observation that when the dimensionality of the feature space is large, by selecting only a small random subset of features for each split, the tree, as a whole, may have an opportunity to see only a small subset of the feature space. It is important to note that this limiting effect applies to the learning stage. Naturally, even a traditionally learned tree may effectively depend on just a small subset of features – that is, those which have been selected for splitting during tree construction.

Let us assume that a binary tree model is constrained to implement S splits (i.e., it has S internal nodes and its depth is at most S) only. If a random subset of F features (out of the total number of N available) is used to determine each split, then the total number of features seen by the tree during the learning process can be estimated as:

$$E = N\left(1 - \left(1 - \frac{F}{N}\right)^{S}\right) \quad (1)$$

For the special case of a stump (i.e., $S = 1$) the effect is particularly dramatic, since only F features are seen by each tree model. The severity of this feature subset selection depends on the total number of features, N, the complexity of the tree models (as determined by S) and the size of the feature window, F, used for each split. For example, for $N = 10,000$, $F = 100$, and $S = 10$ eq. (1) yields $E = 956$, i.e., only about 10% of features being visited by each tree. Since the time-complexity of each node split is usually linear with respect to the number of attributes involved, the overall speed-up with respect to a tree grown using all features can be expected to be proportional to $\frac{N}{F}$.

The above suggests that by knowing the average fraction of features, E, likely to be visited during the tree construction, we can attempt to speed up and/or simplify the learning process for each tree by handing it a random subset of features of size E, *in advance*, without otherwise modifying the tree-growing algorithm in any way. The advantage of such an approach is evident in cases where the entire training set is too big to be contained in the computer's memory, but where the training set slice corresponding to the random feature window (of size E) may do so. Also, note that each tree component of a Random Forest can be seen as performing a variant of N-tuple sampling of the input feature space.

3.2 Group effects

Let us assume that $E \ll N$, which may occur if the individual tree models are severely constrained (e.g., $S = 1$) and/or when the number of available features is quite large. Since individual trees sample the feature space independently from one another, the binomial model used to estimate E in eq. (1) can be used to estimate the number of unique features seen by a collection of K trees. Here E takes the place of F, while K takes the place of S, and the expected number of unique features visited by at least one

[1]For axis-parallel splits, a candidate combines the attribute (i.e., feature) to be split with a particular threshold value at which the attribute's range is to be divided.

tree in the collection is estimated as

$$E_K = N\left(1 - \left(1 - \frac{E}{N}\right)^K\right) \quad (2)$$

If E_K is significantly smaller than N for some value of K (e.g., $E_K/N \leq 0.5$) then, for large feature spaces, an attractive strategy of growing a randomized tree ensemble might be to construct it in blocks of K trees, where each block sees only E_K of the original input features (and assuming, of course, that the learning set with E_K features does not exceed the available memory resources of the running process). Note that the algorithm used to grow each individual tree remains the same.

Since the order in which the tree models are created is irrelevant (with exceptions, such as boosting), the process of building a large ensemble of trees for a high-dimensional problem might be parallelized by mapping the tree ensemble onto number of machines, each growing a collection of K trees over a reduced feature space (i.e., where the number of attributes is reduced, via random sampling, from N to E_K). In practice, for efficiency reasons one might want to grow more than K trees per block. This will result in the trees grown within a common block being more correlated with each other than with trees grown over different feature windows. An ensemble consisting of T trees implemented in this way may underperform one constructed using the original algorithm, although the effects might be negligible if the "overloading" performed within each block is not too extreme.

To summarize, in cases where the complete training set with all features is too big to fit into computer's memory (i.e., RAM), one might construct a Random Forest by:

- Step 1: choosing a random subset of F features (assuming that the feature-reduced training set overcomes the RAM limitations);
- Step 2: building K random trees by sampling from the pre-selected features only;
- Step 3: repeating steps 1 and 2 for a predefined number of times, or till the desired level of accuracy, or a convergence criterion has been reached. For parallel implementations, each sequence of steps 1 and 2 might be executed on a separate machine.

A suitable value of F might determined via cross-validation or heuristically. As demonstrated in [8], Random Forests are not overly sensitive to the choice of F and for rich feature datasets, good results are achieved for $F \ll N$.

4. APPLICATION TO BOOSTING

The popular Adaboost algorithm [11] (and its arcing variants [7]) is an apparent exception among other ensemble classifier schemes, since it requires the component classifiers to be built in a sequential fashion, and is thus more difficult to implement in parallel. The power of boosting stems from its continuous reweighting of training instances, so that difficult to classify examples receive more priority over time. Despite its sequential and deterministic nature, it has been argued in ([8] and [3]) that Adaboost can be seen as a special form of a Random Forest, or MRCL, since in the limit of many iterations it can be viewed as a special type of statistical sampling from a classifier distribution.

Typically, the problem of scaling up boosting has been attacked in the context of datasets having excessive numbers of training instances. A few effective parallelization approaches have been put forward, based on combining the arcing variant of boosting with synchronizing the way in which weights are assigned to individual training instances among the parallel nodes [14][20]. Adaboost assigns non-uniform weights to its component classifiers but, as reported for example in [7][3][14], the weights assigned by boosting can be substituted with uniform weights (which simplifies parallelization [14][20]) with little effect on the aggregate classifier performance. In purely sequential implementations, it has been proposed to modify arcing such that only much smaller subsets (in terms of the number of instances) of the original training data are selected for each iteration [13] (a similar approached using adaptive resampling was described in [10]). One can see that this latter approach bears strong similarity to the idea of using only small subsets of total features when building each model in Random Forests.

Focusing on the use of trees as base learners, Amit et al. [3] extended the randomization idea to boosting by defining Randomized Boosting, where during each boosting iteration a component tree classifier uses random subsets of features when choosing the node splits, just as in the case of Random Forests. For some data sets, Randomized Boosting was even reported to outperform standard (i.e., deterministic) boosting. Clearly, the feature subset selection aspects of Random Forests discussed in Section 3 apply to Randomized Boosting as well. In fact, Randomized Boosting can be seen as combining two approaches building classifier ensembles: boosting and randomization of tree decision splits. Interestingly, Breiman reported in [8] that practical implementations of Random Forests seem to perform better when combined with bagging, which could be seen as "Randomized Bagging".

The experiments performed in [3] used shallow trees in the form of stumps even for fairly moderate numbers of features and, indeed, to effectively apply Adaboost to multi-class feature-rich problems, such as text categorization, Schapire and Singer proposed to use stumps, in a collection of algorithms termed BoosTexter [16]. Boosting with stumps as base learners has also been reported in [3][10] and [12], for example. In the following experimental section we adapted the Real Adaboost.MH (we will drop the *Real* prefix for brevity) algorithm from BoosTexter [16] to be used with Randomized Boosting. Adaboost.MH utilizes a stump as the base weak learner and applies uniform weights to all component classifiers; the reader is referred to [16] for full details. The randomized version of Adaboost.MH simply uses just a random subset of F features during each boosting iteration, as discussed in Section 3.

5. EXPERIMENTAL SETUP

To evaluate the effectiveness of Randomized Boosting, and demonstrate the value of the associated feature subset selection, a number of text classification experiments were performed using the well-known Reuters-21578[2] document corpus. The text medium was chosen, since it represents a naturally high dimensional domain, common in many data mining applications. Following [16], we used the popular Mod-Apte split [4] of the Reuters corpus and focused on the

[2]http://www.research.att.com/~lewis/reuters21578.html

6 of the most populous categories {acq, com, earn, econ, engr, gnrl}, where documents belonging to multiple categories were removed, leaving the 6,089 training documents and 3,044 test documents. For pre-processing, all characters were converted to lower case, and words were defined as sequences of characters delimited by whitespace, which produced $N = 6,502$ unique features. Words are treated as binary attributes, i.e., only the presence/absence of a word in a document was taken into account.

The text categorization performance was measured by means of the macro- and micro-averaged F1 metric, often used in the information-retrieval community, defined as

$$F1 = 2\frac{precision \cdot recall}{precision + recall}$$

where *precision* is the ratio of the correctly classified documents for a given class to the total number of documents classified as belonging to that class, while *recall* is the ratio of the correctly classified documents for a given class to the total number of documents belonging to that class. In the case of macro-averaging, F1 is first measured when distinguishing each class from all others and then the results are averaged. Conversely, in micro-averaging, the contingency tables corresponding to all one-against-others classifiers are merged and then the overall average is computed. More details can be found in [18].

Given this data set, three sets of experiments were performed:

- Standard Adaboost using Adaboost.MH.
- Randomized Boosting based Adaboost.MH for different values of the feature window size, F. For each value of F, 10 runs of the experiment where performed with different random seeds, and the results were averaged.
- "Overloaded" Randomized Boosting, where instead of performing the randomization step at every iteration, it was performed at every I iterations for different settings of I. Overloading can be considered a special case of the block-oriented approach to the ensemble implementation (see Section 3.2), where E is made equal to F. It results in running deterministic boosting for I iterations on the feature-reduced data, before a new random feature subset is selected.
- Block oriented randomized boosting based on Adaboost.MH for a selection of the feature block size E_K, and the feature window size, F.

6. RESULTS

Standard Adaboost.MH achieved the micro and macro-averaged F1 test-set accuracy of 0.92 and, 0.95, respectively. Little change in accuracy occurred beyond 500 iterations, and for all subsequent experiments the overall number of boosting iterations (however divided between blocks, etc.) was kept at 500.

For Randomized Boosting, the dependence of the test-set F1 measure on the number of boosting iterations and different choices of F is depicted in Figures 1 and 2 where, instead of showing the absolute values of F, we used their relative values, with the total number of features used as a reference, i.e., $\alpha = F/N$, where $\alpha \in (0, 1)$. Note that for $\alpha = 1$ Randomized Boosting is equivalent to deterministic

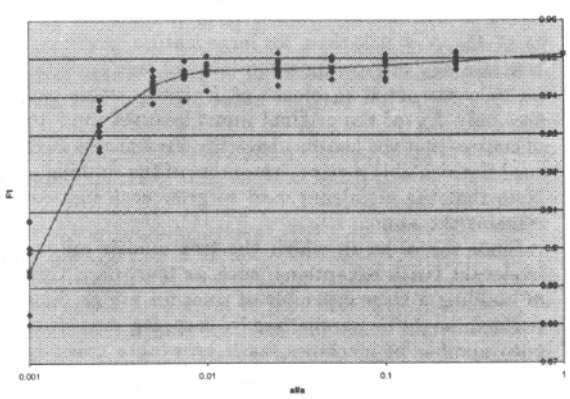

Figure 1: Micro-averaged F1 test-set accuracy of randomized Adaboost.MH for different choices of alpha.

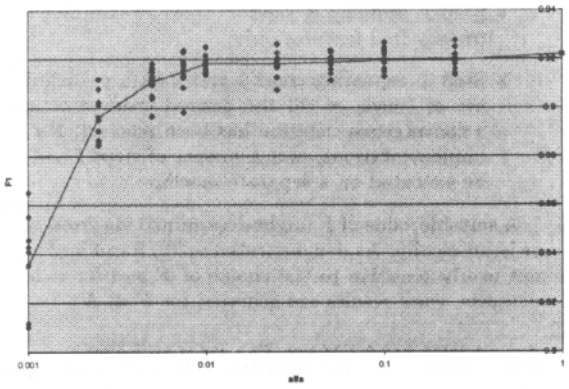

Figure 2: Macro-averaged F1 test-set accuracy of randomized Adaboost.MH for different choices of alpha.

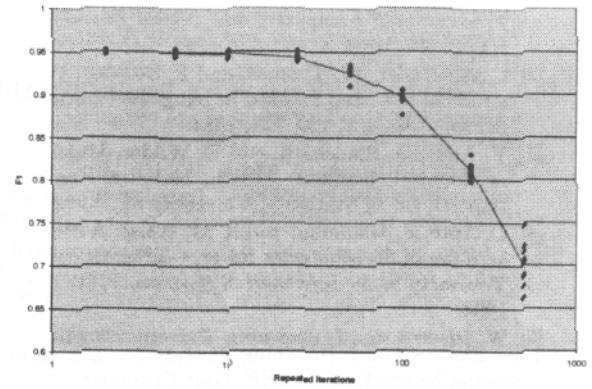

Figure 3: Micro-averaged F1 test-set accuracy of randomized Adaboost.MH for alpha=0.1 and different overloading factors. The x-axis indicates the number of boosting rounds performed for each random feature subset.

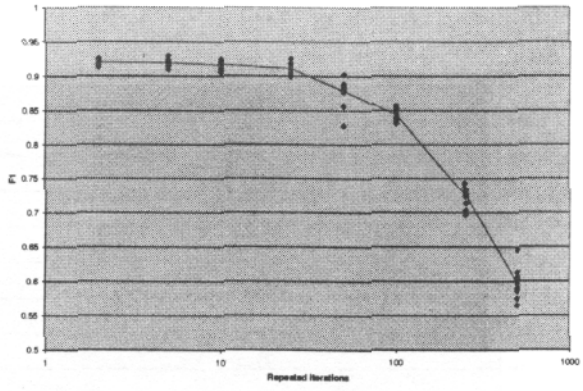

Figure 4: Macro-averaged F1 test-set accuracy of randomized Adaboost.MH for alpha=0.1 and different overloading factors. The x-axis indicates the number of boosting rounds performed for each random feature subset.

boosting. For each value of α, 10 runs of 500 iterations were performed using different random seeds. Clearly, for this data set, randomized boosting using just 10% of total features at each iteration achieves performance largely equivalent to that of standard boosting using all features, while resulting in a 10-fold overall speedup.

In experiments with the overloaded and block-oriented Randomized Boosting, we used the fixed setting of $\alpha = 0.1$. To overload Randomized Boosting, we performed I boosting iterations for each random choice of features (with I ranging from 2 to 500), while keeping the overall number of iterations at 500. The results are shown in Figures 3 and 4, which suggest that moderate overloading does not seem to harm the overall accuracy of the predictor, which may lead to more efficient implementations when dealing with large high-dimensional datasets.

To evaluate the effects of larger-sized blocks, we used $K = 5$ blocks, each sampling $\beta = 50\%$ of the available features. Within each block, Randomized Boosting was run for 100 iterations, so that the overall number of boosting iterations was kept at 500, as in the previous experiments. Six replicates of the experiment were performed, using different random seeds in each case. The averaged performance after 500 iterations was slightly lower that for standard Adaboost.MH and regular Randomized Boosting, but as shown in Figures 5 and 6 the accuracy of the block-oriented algorithm was still rising at the point of termination. Therefore, achieving higher accuracy could be a matter of a trade-off between the memory resources required to implement a block of models, and the run-time cost of a single iteration, which could be exploited in parallel implementations.

7. CONCLUSIONS

We have shown that, for depth-constrained tree models, the algorithmic procedures of randomized classifier ensembles of Random Forests and Randomized Boosting implicitly lead to an effective reduction in the number of input features actually used during the construction of each tree. This limits the amount of resources necessary to construct each component classifier of an ensemble, and can be extremely useful when working with large high-dimensional datasets that do not easily fit into the memory space available to running processes. Since each model needs to see only a fraction of the overall features, a methodology analogous to one used in N-tuple networks can be used to apply random feature-wise masks to the input data before passing it on to individual models. Moreover, the reduced feature requirements of a single model extend to groups of models as well, which can be exploited in both sequential and parallel randomized ensemble implementations.

Using a text categorization task as the test-bed, practical experiments with a randomized version with Adaboost.MH showed that Randomized Boosting using only 10% of the original number of features achieves accuracy levels of comparable standard Adaboost.MH without requiring more iterations. Further efficiency improvements are possible by performing several boosting iterations using the same random feature subset (i.e., overloading) and, for problems with particularly many features, by performing two-stage randomization of features (i.e., blocking). The latter case is more amenable to parallel implementations but there are trade-offs between the number of iterations necessary for good accuracy, the size of a feature window used and the block overloading factor. Such issues can be resolved by taking into account the particulars of the dataset and the availability of the computational resources.

Although we addressed only the problem of handling datasets with large numbers of attributes (and ignored the possibility of processing excessive number of data entries), it would be interesting to examine the effectiveness of combining randomized feature subset selection with adaptive sampling approaches (e.g., those used in [13] and [10]) to handle datasets with both large numbers of features and large numbers of data instances.

8. REFERENCES

[1] I. Aleksander and T. J. Stonham. A guide to pattern

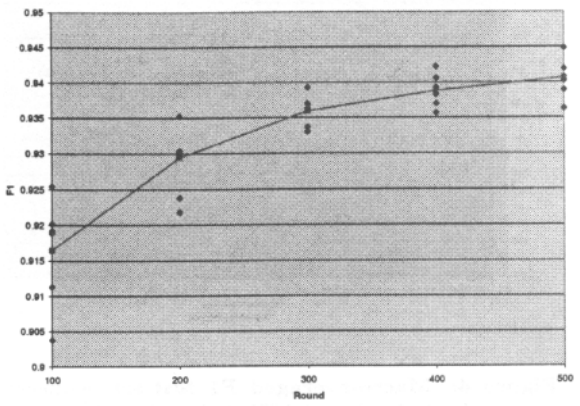

Figure 5: Micro-averaged F1 test-set accuracy of block-oriented randomized Adaboost.MH for alpha=0.1. Here, 100 randomized iterations proceeded within blocks, with each block having access to a random 50% subset of total features. The blocks were executed in sequence, with the test-set performance being updated at the end of each block.

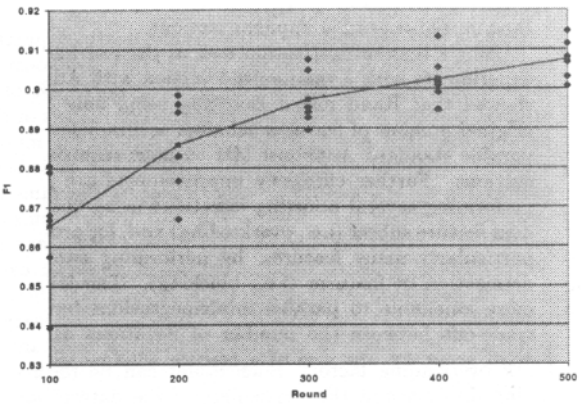

Figure 6: Macro-averaged F1 test-set accuracy of block-oriented randomized Adaboost.MH for alpha=0.1. Here, 100 randomized iterations proceeded within blocks, with each block having access to a random 50% subset of total features. The blocks were executed in sequence, with the test-set performance being updated at the end of each block.

recognition using random-access memories. *IEE Proceedings-E Computers and Digital Techniques*, 2(1):29–40, 1979.

[2] I. Aleksander, W. Thomas, and P. Bowden. WISARD, a radical new step forward in image recognition. *Sensor Rev.*, 4(3):120–124, 1984.

[3] Y. Amit, G. Blanchard, and K. Wilder. Multiple randomized classifiers: MRCL. Technical Report 446, Depertment of Statistics, University of Chicago, 2000.

[4] C. Apté, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

[5] W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *IRE Joint Computer Conference*, pages 225–232, 1959.

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

[7] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[8] L. Breiman. Random forests. *Machine Learning*, 24(2):5–32, 2001.

[9] T. G. Dietterich. An experimental comparison of three methods of constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[10] C. Domingo and O. Watanabe. Scaling up a boosting-based learner via adaptive sampling. In *Proceedings of the 2000 Pacific-Asia Conference on Knowledge Discovery and Data Mining(PAKDD-2000)*, pages 317–328, 2000.

[11] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Machine Learning Conference*, pages 148–156, 1996.

[12] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

[13] D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proceedings of the 2000 International Conference on Pattern Recognition*, 2000.

[14] J. A. Reichler and H. D. Harris. Parallel online continuous arcing and a new framework for wrapping parallel ensembles. In *Proceedings of IJCAI 2001: International Joint Conference on Artificial Intelligence, Workshop on Wrappers for Performance Enhancement in Knowledge Discovery in Databases*, pages 148–156, 2001.

[15] R. Rohwer and M. Morciniec. The theoretical and experimental status of the N-tuple classifier. *Neural Networks*, 11(1):1–14, 1998.

[16] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2-3):135–168, 2000.

[17] V. N. Vapnik. *Statistical Learning Theory*. John Wiley, New York, 1998.

[18] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages

42–49, 1999.

[19] Y. Yang and J. P. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, pages 412–420, 1997.

[20] C. Yu and D. B. Skillicorn. Parallelizing boosting and bagging. Technical Report 2001-442, Depertment of Computing and Information Science, Queen's University, Kingston, Canada, 2001.