International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:6, 2008

Meta Random Forests

Praveen Boinee, Alessandro De Angelis, and Gian Luca Foresti

***Abstract*—**Leo Breimans Random Forests (RF) is a recent development in tree based classifiers and quickly proven to be one of the most important algorithms in the machine learning literature. It has shown robust and improved results of classifications on standard data sets. Ensemble learning algorithms such as AdaBoost and Bagging have been in active research and shown improvements in classification results for several benchmarking data sets with mainly decision trees as their base classifiers. In this paper we experiment to apply these Meta learning techniques to the random forests. We experiment the working of the ensembles of random forests on the standard data sets available in UCI data sets. We compare the original random forest algorithm with their ensemble counterparts and discuss the results.

***Keywords*—** Random Forests [RF], ensembles, UCI.

I. Problem Domain

RANDOM Forests (RF) [1] are one of the most successful tree based classifiers. It has proven to be fast, robust to noise, and offers possibilities for explanation and visualization of its output. In the random forest method, a large number of classification trees are grown and combined. Statistically speaking two elements serve to obtain a random forest - resampling and random split selection. Resampling is done here by sampling multiple times with replacement from the original training data set. Thus in the resulting samples, a certain event may appear several times, and other events not at all. About 2/3rd of the data in the training sample are taken for each bootstrap sample and the remaining one-third of the cases are left out of the sample. This oob (out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance. The design of random forests is to give the user a good deal of information about the data besides an accurate prediction. Much of this information comes from using the oob cases in the training set that have been left out of the bootstrapped training set.

Random split selection is used in each trees growing process. It is computationally effective and offer good prediction performance. It generates an internal unbiased estimate of the generalization. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing. It generates an internal unbiased estimate of the generalization error as the forest building progresses and thus does not over fit. These capabilities of RF can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.

Several authors have noted that constructing ensembles of base learners can significantly improve the performance of learning. Bagging, boosting, are the most popular examples of this methodology. The success of ensemble methods is usually explained with the margin and correlation of base classifiers [13]. To have a good ensemble one needs base classifiers which are diverse (in a sense that they predict differently), yet accurate. The ensemble mechanism which operates on the top of base learners then ensures highly accurate predictions. Here we experiment with random forests as themselves as the base classifiers for making ensembles and test the performance of the model. The ensembles are applied on UCI standard data sets and compared with the original random forest algorithm.

The paper is organized as follows. In section II we introduce the decision tress the bases for constructing the random forests. Section III introduces the actual random forests algorithm. Section IV discusses the ensemble learning and making of bagged and boosted random forests. The experiments with UCI data sets are described in section V. Results are discussed in Section VI.

II. Decision Trees – A base for Random Forests

The decision-tree representation is the most widely used logic method for efficiently producing classifiers from the data. There is a large number of decision-tree induction algorithms described primarily in the machine-learning and applied-statistics literature. The decision tree algorithm is well known for its robustness and learning efficiency with its learning time complexity of $O(n\log_2 n)$. The output of the algorithm is a decision tree, which can be easily represented as a set of symbolic rules (IF…THEN). The symbolic rules can be directly interpreted and compared with the existing domain knowledge, providing the useful information for the domain experts.

A typical decision-tree learning system adopts a top-down strategy that searches for a solution in a part of the search space. It guarantees that a simple, but not necessarily the simplest, tree will be found. A decision tree consists of *nodes* that where attributes are tested. The outgoing *branches* of a node correspond to all the possible outcomes of the test at the

Manuscript received September 30, 2005.

Praveen Boinee is the PhD Student in Computer Science in Udine University, Udine, 33100, Italy (phone: 0039-0432-558231; e-mail: boinee@fisica.uniud.it).

Alessandro De Angelis is the Professor in Experimental and Computational Physics at Udine University, Udine, 33100, Italy (e-mail: deangelis@fisica.uniud.it).

Gian Luca Foresti is the Professor in Computer Science at Udine University, Udine, Italy (email: foresti@dimi.uniud.it).

node. A simple decision tree for classification of samples with two input attributes X and Y is given Fig. 1.

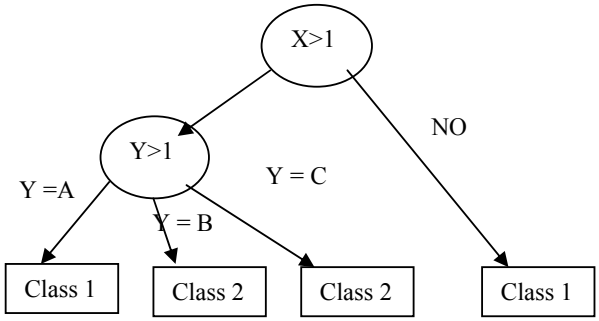


Fig. 1 A simple decision tree with the tests on attributes X and Y

All samples with feature values X>1 and Y=B belong to Class2, while the samples with values X<1 belong to Class1, whatever the value for feature Y. The samples, at a nonleaf node in the tree structure, are thus partitioned along the branches and each child node gets its corresponding subset of samples. Decision trees that use univariate splits have a simple representational form, making it relatively easy for the user to understand the inferred model; at the same time, they represent a restriction on the expressiveness of the model. In general, any restriction on a particular tree representation can significantly restrict the functional form and thus the approximation power of the model. A well-known tree-growing algorithm for generating decision trees based on univariate splits is Quinlan's *ID3* with an extended version called *C4.5 [6]*. Greedy search methods, which involve growing and pruning decision-tree structures, are typically employed in these algorithms to explore the exponential space of possible models and to remove unnecessary preconditions and duplication.

C4.5 applies a divide and conquers strategy to construct the tree. The sets of instances are accompanied by a set of properties. A decision tree is a tree where each node is a test on the values of an attribute, and the leaves represent the class of an instance that satisfies the tests. The tree will return a 'yes' or 'no' decision when the sets of instances are tested on it. Rules can be derived from the tree by following a path from the root to a leaf and using the nodes along the path as preconditions for the rule, to predict the class at the leaf. For developing random forests, we use the trees that randomly choose a subset of attributes at each mode.

III. Random Forests

A random forest is a classifier consisting of a collection of tree structures classifiers $\{h(x,\Theta_k), k=1,...\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x. The forest chooses the classification having the most votes over all the trees in the forest.

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

and the overall forest error rate depends on two things:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

After each tree is built, all of the data are run down the tree, and proximities are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

To formalize the working of the random forests, Let the forest contain K classifier trees $h_1(x), h_2(x),..., h_K(x)$ and the joint classifier be $h(x)$. Each learning instance is represented by an ordered pair (**x**,*y*), where each vector of attributes **x** consists of individual attributes $A_i, i=1,...,a$ (*a* is the number of attributes) and is labeled with the target value $y_j, j=1,...,c$ (*c* is the number of class values). The correct class is denoted as *y*, without index. Each discrete attribute *Ai* has values v_1 through v_{m_i} (m_i is the number of values of attribute *Ai*). We write $p(v_{i,k})$ for the probability that the attribute *Ai* has value v_k, $p(y_j)$ is the probability of the class y_j, and $p(y_j \mid v_{i,k})$ is the probability of the class y_j conditioned by the attribute *Ai* having the value v_k.

Each training set of n instances is drawn at random with replacement from the training set of n instances. With this sampling called bootstrap replication, on average 36.8% of training instances are not used for building each tree. These out of bag instances come handy for computing an internal estimate of the strength and correlation of the forest. Let set of out-of-bag instances for classifier $h_k(x)$ as $O_k(x)$. Let $Q(x, y_j)$ be the out-of-bag proportion of voted for class y_j at input x and an estimate of $P(h(x) = y_j)$:

$$Q(x, y_j) = \frac{\sum_{k=1}^{K} I(h_k(x) = y_j; (x, y) \in O_k)}{\sum_{k=1}^{K} I(h_k(x); (x, y) \in O_k)}$$

where I (.) is the indicator function.

Calculate the margin function which measures the extent to which the average vote for the right class y exceeds the average vote for any other class as follows.

$$mr(x, y) = P(h(x) = y) - \max_{\substack{j=1 \\ j \neq y}}^{c} P(h(x) = y_j)$$

It is estimated with $Q(x, y)$ and $Q(x, y_j)$. Strength is defined as the expected margin, and is computed as the average over the training set:

$$s = \frac{1}{n} \sum_{i=1}^{n} \left(Q(x_i, y) - \max_{\substack{j=1 \\ j \neq y}}^{c} Q(x_i, y_j) \right)$$

The average correlation is computed as the variance of the margin over the square of the standard deviation of the forest:

$$\bar{\rho} = \frac{\mathrm{var}(mr)}{sd(h(\ ))^2} = \frac{\frac{1}{n} \sum_{i=1}^{n} \left(Q(x_i, y) - \max_{\substack{j=1 \\ j \neq y}}^{c} Q(x_i, y_j) \right) - s^2}{\left(\frac{1}{k} \sum_{t=1}^{K} \sqrt{p_k + \hat{p}} + \left(p_k - \hat{p} \right)^2 \right)^2}$$

Where

$$p_k = \frac{\sum_{(x_i, y) \in O_k} I(h_k(x) = y)}{\sum_{(x_i, y) \in O_k} I(h_k(x))}$$

is an out-of-bag estimate of $P(h_k(x) = y)$ and

$$\hat{p}_k = \frac{\sum_{(x_i, y) \in O_k} I\left(h_k(x) = \hat{y}_j \right)}{\sum_{(x_i, y) \in O_k} I(h_k(x))}$$

is an out-of-bag estimate of $P\left(h_k(x) = \hat{y}_j \right)$ and

$$\hat{y}_j = \arg\max_{\substack{j=1 \\ j \neq y}}^{c} Q(x, y_j)$$

is estimated for every instance x in the training set $Q(x, y_j)$.

Breiman used unpruned decision tress as base classifiers and introduces additional randomness into the trees [4]. Namely, in each interior node of each tree a subset of r attributes is randomly selected and evaluated with the Gini index heuristics. The attribute with the highest Gini index is chosen as split in that node.

In classification problems, attribute evaluation methods are Gini index [5], Gain ratio [6], ReliefF [7], MDL [8], and KDM [9]. Random Forests uses the Gini index taken from the CART learning system [10]. The gini index is given by the formula

$$Gini(A_i) = -\sum_{i=1}^{c} p(y_i)^2 + \sum_{j=1}^{m_i} p(v_{i,j}) \sum_{i=1}^{c} p(y_i / v_{i,j})^2$$

IV. Ensembles of Random Forests

Ensemble methods became popular as a relatively simple device to improve the predictive performance of a base procedure. They combine "base classifiers" to predict the label for the new data points. Experiments on several benchmark data sets and real world data sets showed an improved classification results from these techniques. In this paper we concentrate on 2 ensembles techniques AdaBoost and Bagging. The bagging procedure turns out to be a variance reduction scheme, at least for some base procedures. On the other hand, boosting methods are primarily reducing the (model) bias of the base procedure. We experiment to construct these ensembles with random forests as base classifiers.

The training data set is a collection of the data points associated with labels. The data points, usually a vector of features (*x*), and the labels *y*, are bounded by an underlying function *f* such that *y = f(x)* for each training data point (x,y). Machine learning algorithms search for a best possible hypothesis *h* to *f* that can be applied to assign labels to new x values. Ensemble learning algorithms construct a set of hypothesis $\{h_1, h_2, ..., h_k\}$ and construct a voted classifier to predict the label of new data points, where T a criterion to combine the hypothesis.

$$H^*(x) = T(h_1(x), h_2(x), ..., h_n(x))$$

A. Bagging Random Forests

Bagging is a statistical re-sample and combine technique [14] based on bootstrapping and aggregating techniques. The basic idea of bagging is to use bootstrap re-sampling to generate multiple versions of a predictor which, when combined, should perform better than a single predictor built to solve the same problem. Bootstrapping is based on random sampling with replacement. Therefore, taking a bootstrap i.e., (random selection with replacement) of the training set X, one can sometimes avoid or get less misleading training objects in the bootstrap training set. Consequently, a classifier constructed on such a training set may have a better performance. Aggregation actually means combining classifiers [15]. Often a combined classifier gives better results than individual classifiers, because of combining the advantages of the individual classifiers in the final solution. Therefore, bagging might be helpful to build a better classifier on training sample sets with misleaders.

On average, when taking a bootstrap sample of the training set, approximately 37% of the objects are not presented in the bootstrap sample, meaning that possible 'outliers' in the training set sometimes do not show up in the bootstrap sample. Thus, better classifiers (with a smaller apparent error – classification error on the training data set) may be obtained by the bootstrap sample than by the original training set. These classifiers will be presented 'sharper' in the apparent error than those obtained on the training sets with outliers. Therefore, they will be more decisive than other bootstrap versions in the final judgment. Thus, aggregating classifiers in bagging can sometimes give a better performance than individual classifiers.

In this section we discuss the application of bagging algorithm to grow the ensembles of random forest. The random forest itself is considered to be the varied version of bagged decision trees. As the growth of trees in RF is based on randomization, we try to experiment the growth of ensembles of random forests with bagging in order to reduce the overall bias and variance of learning system. Fig. 2 describes the bagged random forest algorithm.

Algorithm

Input:

Training Set T <xi,yi>, xi is a d-dimension input vector, yi a univariate response or label of the input vector

Random Forest Algorithm h $: R^d \to R$, d is the dimension of the input vector

Integer J [Number of random forests to be generated]

1. Construct a bootstrap sample $(x_1^*, y^*_1), \ldots, (x_N^*, y^*_N)$ by randomly drawing n times with replacement from the data $(x_1, y_1), \ldots, (x_N, y_N)$
2. for each iteration i=1..j
3. {
4. Generate a random forest $RF_i(X)$ over the bootstrapped sample $(x_1^*, y^*_1), \ldots, (x_N^*, y^*_N)$ as described in section randomforest which can minimize the bias over the data set.
5. }
6. The final bagged ensemble of RF, $RF_{bag}(X)$ is formed by the combination of individual $RF_i(X)$
7. $\mathrm{RFbag(X)} = \mathrm{J}^{-1} \sum_{i=1}^{J} RF_i(X)$

Output: RFbag(X)

Fig. 2 Bagged random Forests

B. AdaBoosted Random Forests

Boosting works by repeatedly running a learning algorithm on various distributions over the training data, and then combining the classifiers produced by the learner into the single composite classifier [16]. The boosting algorithm takes as input a training set of m examples $S = ((x_1, y_1), ..., (x_m, y_m))$ where each instance x_i is a vector of attributes drawn from the input space X and y_i belonging to finite label set Y, is the class label associated with x_i. In boosting classifiers and training sets are obtained in a strictly deterministic way. Both training sets and classifiers are obtained sequentially in the algorithm, in contrast to bagging, where training sets and classifiers are obtained randomly and independently from the previous step of the algorithm. At each step of the boosting, training data are reweighed in such a way that incorrectly classified objects get larger weights in a new modified training set [17]. AdaBoost manipulates the training examples to generate multiple hypotheses. It maintains the probability distribution $p_l(x)$ over the training examples. In each iteration l, it weights the training samples with the probability distribution $p_l(x)$. The learning algorithm is then applied to produce the classifier h_l. The error rate ε_l of this classifier on the training examples is computed and used to adjust the probability distribution on the training examples. The effect of the change in the weights is to place more weight on training examples that were misclassified by h_l and less weight on examples that were correctly classified in the last stage. In subsequent iterations, therefore, AdaBoost tend to construct progressively more difficult learning problems. The final classifier, h_{final}, is constructed by a weighted vote of the individual classifiers $h_1, h_2, ...h_n$. Each classifier is weighted according to its accuracy for the distribution p_l that it was trained on.

A boosted random forest can be constructed in 2 ways. The first is to boost the forests. The main idea is to build Adaboost

for each random vector θ to obtain a simple Adaboost classifier, each with small number of variables. The second approach is to use random forests algorithms as a weak learner. Figure 3 describes the algorithm

Algorithm

Input: Train Set T , with the sequence of m-labels $<(x_1, y_1),...,(x_m, y_m)>$ with labels $y_i \in Y = \{1,...k\}$

Random Forest as the base learner **RF**
Integer T specifying number of iterations.
Distribution D_l over the m examples.
Initialize $D_l(i) = 1/m$ for all I, initialize the weights.
rf_{final} = 0; // the final hypothesis from boosted random forest

```
Dltotal = 0
for i=1;i<=m;i++
{
   Dltotal = Dltotal+ Dl(i);
}

for t=1 ;t<=T;t++
{

// compute the normalized weights


for i=1;i<=m;i++
{
   Dl(i)= Dl(i)/Dltotal;
}


// call the random forest to get the hypothesis ht

K = Number of trees to be generated in the random forest

for (k =1, k<=K, k++)
{

for( i=1;i<=m;i++)
   {
   Generate vector θk ;
   Construct a tree h(xi, θk) with C4.5 algorithm.
   Each tree casts 1 vote for the most popular class.
}
```

The final selection of class for a test row is selected by a voted majority

```
Return the hypothesis ht
}
//Calculate the error of ht :
```

$\varepsilon_t = 0$

```
for( i=1;i<=m;i++)
{
```

if ($h_t(x_i) \neq y_i$)

```
   {
```

$\varepsilon_t = \varepsilon_t$ + Dt(i);

```
   }
}
```

if $\varepsilon_t > ½$

```
  T =t-1;
Else
Exit(0); // abort loop
```

$\beta_t = \varepsilon_t / 1 - \varepsilon_t$;

// update distribution D_t

if ($h_t(x_i) = y_i$)

```
{
```

$D_{t+1}(i) = D_t(i) / D_l(i) * \beta_t$;

```
}
else
```

$D_{t+1}(i)$ = 1;

```
// upsate the final hypothesis

rffinal = rffinal + log (1/ βt) ;
}

Output: The final hypothesis
for (y=1;y<=k;y++)
{
```

$rf_{final} = \arg\max_{y \in Y} rf_{final}$;

```
}
```

Fig. 3 Boosted Random Forests

V. Experiments on UCI Data Sets

UCI machine learning repository [11] contains data sets that have been in use to evaluate learning algorithms. Each data file contains individual records described in terms of attribute-value pairs. The ensembles of RF along with the original RF have been evaluated on several datasets from the UCI Machine learning Repository. The data sets used in the experiment are summarized in the table I. These data sets show considerable diversity in the size, number of classes and number of type of attributes.

TABLE I
DESCRIPTION OF UCI DATA SETS

Name	Cases	Classes	Attributes Cont	Discr
Breast Cancer	699	2	9	0
Letter	20000	26	16	-
Labour	57	2	8	8
Vehicle	846	4	18	-
Sonar	208	2	60	-
Glass	214	6	9	-
Vowel	990	11	11	10
Hepatitis	155	2	6	13
Heart-c	303	2	8	5
Wave Form	300	3	21	-
Letter	20000	26	16	-

For the original random forest algorithm, the parameter T governing the number of classifiers generated was set at 20 for these experiments. In our experiments a random forest with 20 trees have shown significant improvements in the classification accuracies, further growth of trees have not shown any significant improvements in classification accuracies (figure 4). The experiment is conducted with unpruned trees, and the variable yielding the smallest gini index has been used for splitting, tree building is stopped when the number of instances in a node is 5 or less. RF facilitate to deal with the fractional instances, required when some attributes have missing values, which can be easily adapted to handle the instance weights used in the ensembles.

The following parameters are used for bagging random forest. A 100% bagSizePercent [Size of each bag, as a percentage of the training set size] without any out-of bag errors. We used 20 random forests to grow the bagged ensembles with a random seed of 1. Further increases in random forests have not shown any significant improvements. Most of the improvement from bagging is evident with in few replications, and it is interesting to see the performance improvement that can be bought by a single order of magnitude increase in computation.

The boosting random forests have been grown with the same parameters for RF described above. For boosting ensembles, 20 random forests have been grown with a random number seed of 1. The weight threshold for weight pruning has set to 100, only reweighing has been performed without any resampling of data sets.

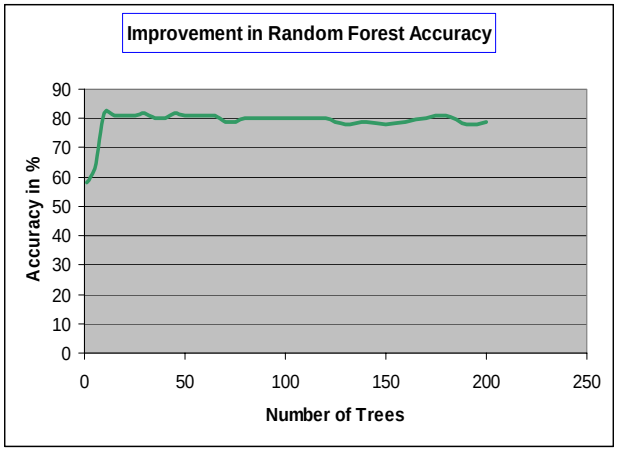


Fig. 4 Performance of random forests as function of number of trees

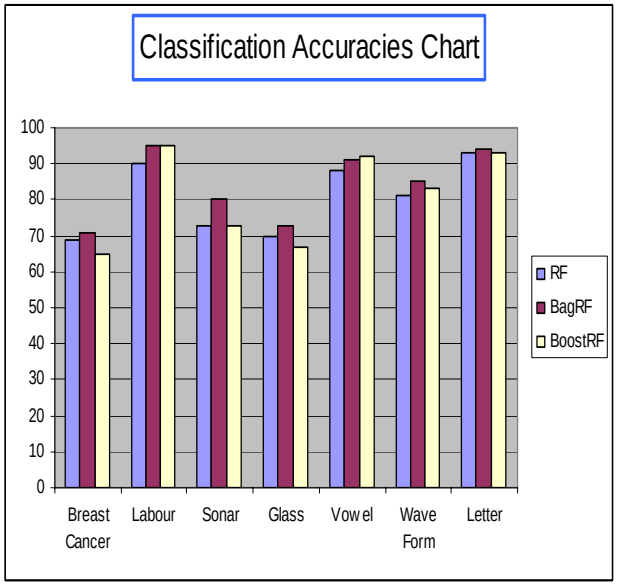


Fig. 5 Classification accuracies for RF and its ensembles for various UCI data sets

VI. COMPARATIVE STUDY OF CLASSIFICATION RESULTS

Table II shows the classification results for Random forests (RF), Bagged RF and Boosted RF on various UCI data sets. The data sets are splitted in 66% for training and remaining 34% for testing. Though the ensembles take much time than single classifiers they produce improved classification results. We saw significant improvements with bagged random forests. Adaboosted random forests also showed improved classification results though they are inferior to that of bagged ones.

The statistical measures used in the comparative study are the following:

- **ROC curves (AREA)**: A Receiver Operating Characteristic (ROC) curve summarizes the performance of a two-class classifier across the range of possible thresholds. It

is recommended for comparing classifiers, as it does not merely summarize performance at a single arbitrarily selected decision threshold, but across all possible decision thresholds. It plots the sensitivity (class two true positives) versus one minus the specificity (class one false negatives). An ideal classifier hugs the left side and top side of the graph, and the area under the curve is 1.0.

- **Classification accuracy (%):** The classification accuracy for a classifier is defined as percentage of number of correctly classified samples to the total number of samples.

- **Mean absolute Error (MAE):** It is the ratio of Incorrectly Classified Instances to that of Total Number of Instances.

Fig. 5 gives the classification accuracies as bar charts for various data sets. Fig. 6 shows the ROC curves. Bagged random forests have shown clear improvements over the original random forests, whereas boosted technique has ranged from the best to rather medicore results. When bagging and boosting are compared head to head, boosting leads to greater accuracies for vowel and hepatitis data sets.
But it also performed inferior to random forests in data sets like Glass, Breast cancer. Bagging has shown consistent performances and been less risky. It proved to more suitable for increasing the random forest accuracies.

VII. Discussion

The ensembles have shown classification improvements for the original random forests. In overall rankings Bagged random forests have shown superior results.

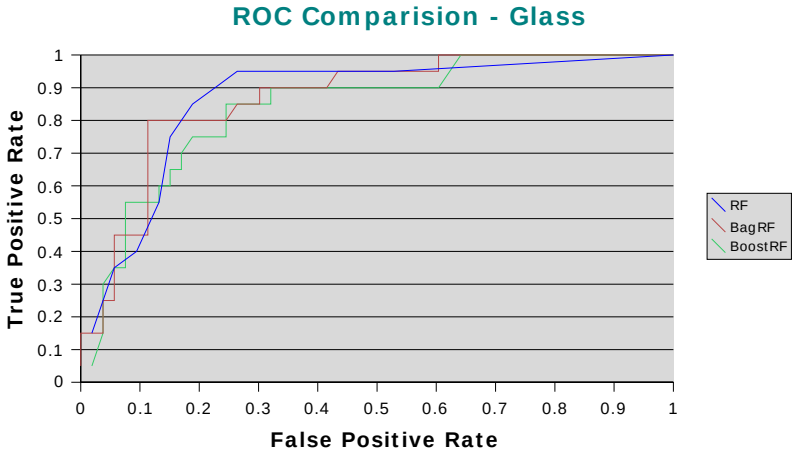
The possible reason for boosting failure is the deterioration in generalization performances.

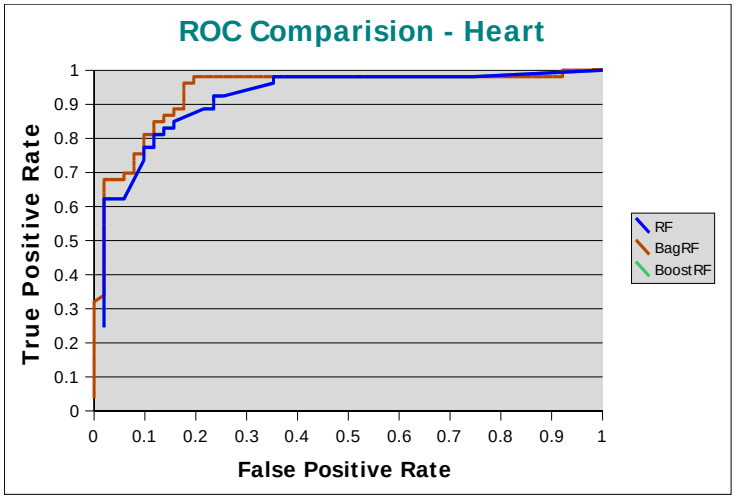
A large number of trials T allow the composite classifier rf_{boost} to become very complex. A simple alteration to avoid overfitting by keeping T as small as possible without impacting the objective. Adaboost.M1 stops when the error of any of any base classifier drops to zero, but does not address the possibility that the final classifier rfboost might correctly classify all the training data even though no base classifier does. Further trials in this situation have increased the complexity of the rf_{boost} but cannot improve its performance on the data sets.

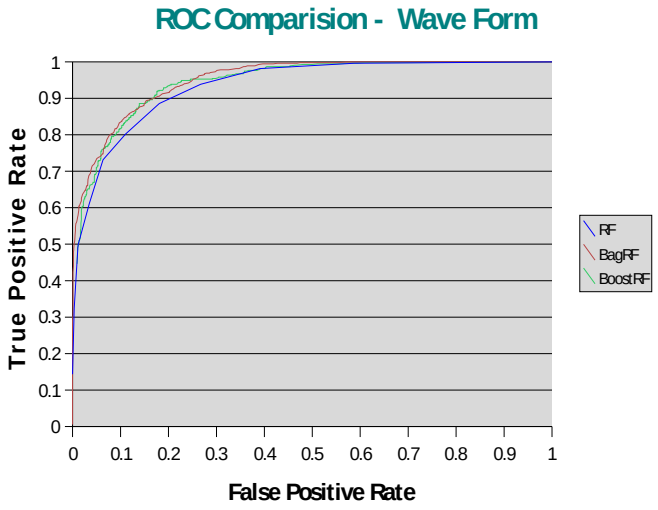
The undeniable benefits of boosting are not attributable just to producing a composite classifier rf_{boost} that performs well on the training data. It also calls into question the hypothesis that overfitting is sufficient to explain boosting's failure on some data sets, since much of the benefit realized by boosting seems to be caused by overfitting.

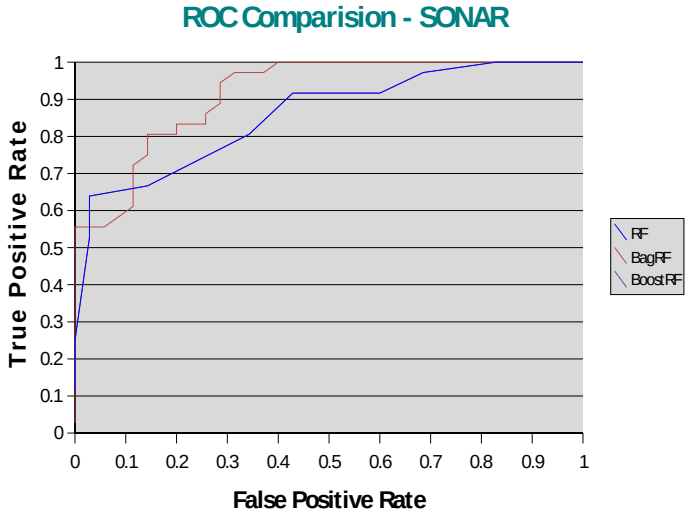
In this paper, we investigated the possibilities of improving the random forests. Experiments over diverse collection of data sets have confirmed that the ensembles of random forest have improved the performance of classifications. Boosting and Bagging both have a sound theoretical base and also have the advantage that the extra computation they require is known in advance – if T classifiers are generated, then both require T times the computational effort of random forests. In these experiments, a little increase in computation can buys a significant increase in the classification accuracies. In many applications, improvements of this magnitude would be well worth the computational cost.

The future work will be concentrated on experimenting with other ensemble techniques such as multi-boost, random committee.




ROC Comparision - Heart
True Positive Rate
False Positive Rate
RF
BagRF
BoostRF


ROC Comparision - Wave Form
True Positive Rate
False Positive Rate
RF
BagRF
BoostRF


ROC Comparision - SONAR
True Positive Rate
False Positive Rate
RF
BagRF
BoostRF

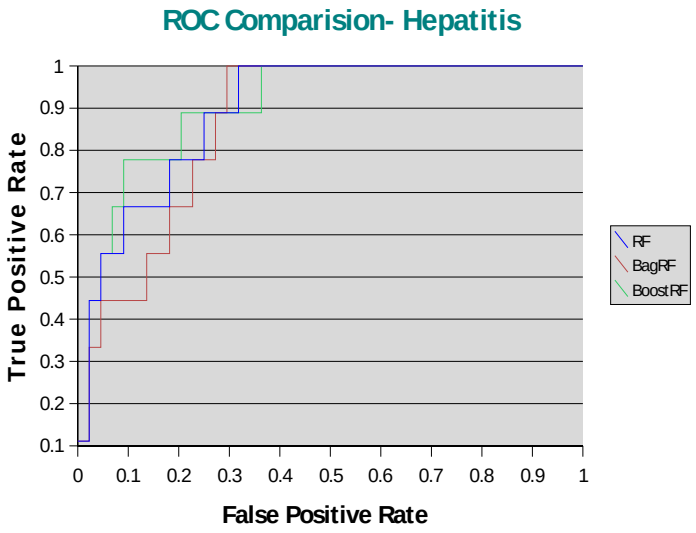


Fig. 6 ROC curves for various UCI data sets

TABLE II
THE CLASSIFICATION RESULTS FOR RANDOM FORESTS (RF), BAGGED RF AND BOOSTED RF ON VARIOUS UCI DATA SETS. NOTATIONS % FOR CLASSIFICATION ACCURACY, MAE FOR MEAN ABSOLUTE ERROR, ROC FOR ROC AREA

	RF			BAGRF			BOOSTRF		
DATA SETS	%	MAE	ROC	%	MAE	ROC	%	MAE	ROC
BREAST CANCER	69.23	.362	.634	73.46	.377	.683	66.32	.361	.624
HEART-C	82.69	.099	.917	84.61	.099	.937	82.69	.099	.917
GLASS	69.86	.108	.804	72.60	.118	.804	67.12	.093	.839
HEPATITIS	84.90	.221	.893	84.90	.272	.866	88.67	.113	.900
LABOUR	90	.205	.998	95	.210	.978	95	.216	.978
SONAR	73.23	.314	.857	80.28	.328	.915	73.23	.314	.857
VOWEL	88.13	.054	.999	91.09	.062	1	92.28	.013	1
LETTER	93 .14	.014	.993	93 .14	.014	.993	93 .14	.014	.993
WAVEFORM	80.82	.188	.935	84.76	.196	.952	83.05	.113	.944

REFERENCES

[1] Breiman, L.: *Random Forests Technical Report*, University of California, 2001.
[2] http://www.stat.berkeley.edu/users/breiman/RandomForests/cc_home.htm#intro
[3] Breiman, L.: *Looking Inside the Black Box*, Wald Lecture II, Department of Statistics, California University, 2002.
[4] Sikonja. M, *Improving Random Forests*. In J.-F. Boulicaut et al.(Eds): ECML 2004, LNAI 3210, Springer, Berlin, 2004, pp. 359-370.
[5] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and regression trees*. Wadsworth Inc., Belmont, California, 1984.
[6] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, 1993.
[7] Igor Kononenko. *Estimating attributes: analysis and extensions of Relief*. In Luc De Raedt and Francesco Bergadano, editors, Machine Learning: ECML-94, pages 171–182. Springer Verlag, Berlin, 1994.
[8] Igor Kononenko. *On biases in estimating multi-valued attributes*. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95), pages 1034–1040. Morgan Kaufmann, 1995.
[9] Thomas G. Dietterich, Michael Kerns, and Yishay Mansour. *Applying the weak learning framework to understand and improve C4.5*. In Lorenza Saitta, editor, Machine Learning: Proceedings of the Thirteenth International Conference (ICML'96), pages 96–103. Morgan Kaufmann, San Francisco, 1996.
[10] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and regression trees*. Wadsworth Inc., Belmont, California, 1984.
[11] http://www.ics.uci.edu/~mlearn/MLRepository.html
[12] J.R. Quinlan, *Bagging, Boosting, and C4.5*, In Proceedings, Fourteenth National Conference on Artificial Intelligence, 1996.
[13] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. *Boosting the margin: a new explanation for the effectiveness of voting methods.* In Douglas H. Fisher, editor, Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97), pages 322–330. Morgan Kaufmann, 1997.
[14] Breiman, L., *Bagging Predictors*, Machine Learning (1996) 24:123-140.
[15] Carney, J., Cunningham, P.: *The NeuralBAG algorithm: optimizing generalization performance in Bagged Neural Networks*. In: Verleysen, M. (eds.): Proceedings of the 7th European Symposium on Artificial Neural Networks (1999), pp. 3540.
[16] Freund, Y., Schapire, *RE.: Experiments with a new boosting algorithm*. In Proceedings 13th International Conference on Machine Learning (1996) 148–156.
[17] Skurichina, M., Duin, R.P.W.: *Bagging, Boosting and the Random Subspace Method for Linear Classifiers*, Vol. 5, no. 2, Pattern Analysis and Applications (2002) 121-135.