An Empirical Evaluation of Supervised Learning for ROC Area

Rich Caruana and **Alexandru Niculescu-Mizil** [1]

Abstract.

We present an empirical comparison of the AUC performance of seven supervised learning methods: SVMs, neural nets, decision trees, k-nearest neighbor, bagged trees, boosted trees, and boosted stumps. Overall, boosted trees have the best average AUC performance, followed by bagged trees, neural nets and SVMs. We then present an ensemble selection method that yields even better AUC. Ensembles are built with forward stepwise selection, the model that maximizes ensemble AUC performance being added at each step. The proposed method builds ensembles that outperform the best individual model on all the seven test problems.

1 Introduction

This paper presents the results of an empirical comparison of the AUC performance of seven different supervised learning algorithms. The algorithms are SVMs, neural nets, decision trees, memory-based learning, bagged trees, boosted trees, and boosted stumps. For each algorithm we test many different variants of the algorithm. For example, we compare ten different styles of decision trees, neural nets of many sizes, SVMs using many different kernels and many different values for C (the margin parameter) and gamma (the RBF parameter), etc. A total of 2000 models are tested on each problem. See Appendix 1 for details of how these 2000 models are trained.

We compare the AUC performance of the algorithms on 7 binary classification problems. ADULT, COVER_TYPE and LETTER are from UCI Machine Learning Repository [2]. ADULT is the only problem that has nominal attributes. For ANNs, SVMs and KNNs we transform nominal attributes to boolean. Each DT, BAG-DT, BST-DT and BST-STMP model is trained twice, once with the transformed attributes and once with the original attributes. COVER_TYPE has been converted to a binary problem by treating the largest class as the positive and the rest as negative. We converted LETTER to boolean in two ways. LETTER.p1 treats the letter "O" as positive and the remaining 25 letters as negative, yielding a very unbalanced binary problem. LETTER.p2 uses letters A-M as positives and the rest as negatives, yielding a well balanced problem. HYPER_SPECT is the IndianPine92 data set [9] where the difficult class Soybean-mintill is the positive class. SLAC is a problem from the Stanford Linear Accelerator and MEDIS is a medical data set. The characteristics of these data sets are summarized in Table 1.

Table 1. Description of problems

PROBLEM	#ATTR	TRAIN SIZE	TEST SIZE
ADULT	14/104	4000	35222
COVER_TYPE	54	4000	25000
LETTER.P1	16	4000	14000
LETTER.P2	16	4000	14000
MEDIS	63	4000	8199
SLAC	59	4000	25000
HYPER_SPECT	200	4000	4366

2 AUC of Learning Algorithms

In this section we compare the best AUC performance that can be achieved with each learning algorithm by varying each algorithm's control parameters. For each algorithm we use the control parameter settings described in the Appendix. We train all models on identical train sets containing 4000 points.

Parameter optimization is more difficult with some algorithms than with others. We do not want to handicap an algorithm for which parameter selection is difficult, so we run our experiments two different ways. In one set of experiments, we use a 1K validation set to select the best model from each learning algorithm, and then report the performance of this model on the large final test set. In the second set of experiments we perform model selection using the large final test sets (i.e. no validation set) and then report the performance of the best models on that same final test set. This second set of experiments approximates the performance that might be achieved if model selection were done optimally. In all of our experiments, the final test sets are large enough to make discerning small differences in AUC reliable.

Table 2 shows the AUC performance of the best models of each type when selection is done using the large final test sets. This table represents the best performance that could be achieved with each learning method if model selection was done perfectly. Table 3 shows the AUC performance of the best models of each type when model selection is performed using 1K validation sets. This presents a more realistic view of the AUC performance that might be obtained using a simple approach to model selection, but does not show the AUC performance that might be obtained if model selection was done more carefully.

[1] Department of Computer Science, Cornell University, Ithaca, NY 14853 USA email {caruana,alexn}@cs.cornell.edu

Table 2. AUC of the best model from each learning algorithm on each problem selected using the large final test set.

MODEL	COVER_TYPE	ADULT	LETTER.P1	LETTER.P2	MEDIS	SLAC	HYPER_SPEC	MEAN
BST-DT	**0.9154**	0.8918	**0.9978**	**0.9944**	0.8275	0.8004	**0.9764**	**0.9148**
BAG-DT	0.9020	0.9070	0.9947	0.9809	0.8216	**0.8018**	0.9623	0.9100
ANN	0.8690	0.8986	0.9936	0.9811	**0.8423**	0.7981	0.9733	0.9080
SVM	0.8711	0.8980	0.9963	0.9918	0.8290	0.7960	0.9694	0.9074
KNN	0.8767	0.8790	0.9954	0.9873	0.8254	0.7791	0.9528	0.8994
BST-STMP	0.8436	**0.9105**	0.9824	0.9136	0.8401	0.7777	0.9557	0.8891
DT	0.8476	0.8901	0.9738	0.9454	0.7747	0.7721	0.9212	0.8750
MEAN	0.8751	0.8964	0.9906	0.9706	0.8229	0.7893	0.9587	0.9005

Table 3. AUC of the best model from each learning algorithm on each problem selected using 1K validation sets.

MODEL	COVER_TYPE	ADULT	LETTER.P1	LETTER.P2	MEDIS	SLAC	HYPER_SPEC	MEAN
BST-DT	**0.9152**	0.8902	**0.9974**	**0.9944**	0.8232	0.8004	**0.9757**	**0.9138**
BAG-DT	0.9007	0.9057	0.9888	0.9806	0.8216	**0.8009**	0.9623	0.9087
SVM	0.8687	0.8980	0.9946	0.9918	0.8288	0.7954	0.9694	0.9067
ANN	0.8681	0.8980	0.9929	0.9772	0.8254	0.7940	0.9673	0.9033
KNN	0.8750	0.8790	0.9911	0.9870	0.8253	0.7690	0.9525	0.8970
BST-STMP	0.8428	**0.9074**	0.9813	0.9136	**0.8310**	0.7746	0.9557	0.8866
DT	0.8423	0.8742	0.9596	0.9454	0.7716	0.7721	0.9212	0.8695
MEAN	0.8733	0.8932	0.9865	0.9700	0.8181	0.7866	0.9577	0.8979

The entries in both tables show the AUC performance of the best model that was trained with each of the seven learning methods on each problem. The algorithm that has the best AUC on each problem is shown in bold. The last column shows the mean AUC of each algorithm across the seven problems.

In general, the loss in AUC performance for models selected using the 1k validation sets as opposed to using the final test set is small. On average, "optimal" model selection only improves AUC 0.0026 compared to selection with the 1k validation sets. More sophisticated selection procedures such as cross-validation might reduce this difference even further. We conclude that for these problems, 1k validation sets are sufficient to achieve good model selection for AUC. The most imbalanced data set (Letter.p1) has about 5% positives. The remaining data sets are more balanced. We suspect that 1k validation sets yield good model selection for AUC because AUC is more robust (i.e. low variance) than other measures such as accuracy even when data is moderately imbalanced.

Looking at Table 2, the model type with best overall AUC performance is boosted decision trees. It not only has the highest mean AUC performance, but boosted trees yield the best AUC on four of the seven test problems. No other learning method wins more than one test problem.

The next best learning methods for AUC are bagged decision trees, neural nets, and SVMs. (When selection is done using the 1k validation sets, SVMs move slightly ahead of the neural nets.) Boosted stumps and plain decision trees are not competitive, though boosted stumps are best on the Adult data set. It is interesting to note that boosting weaker stump models is clearly inferior to boosting full decision trees on most of the test problems: boosting full decision trees yields better performance than boosting stumps on five of the test problems and the mean performance for boosted trees is much higher than the mean performance for boosted stumps. Occasionally boosted stumps perform very well, but when they do not, they often perform very poorly, giving them poor average AUC performance.

Although it is clear that boosting full decision trees yields the best overall AUC performance on these test problems, the differences in performance between the top four models (boosted trees, bagged trees, neural nets, and SVMs) is only about 0.01. The mean difference between the best models (boosted trees) and the worst models (plain decision trees) is about 0.04. These results suggest that competitive AUC performance can be obtained with most learning algorithms if care is taken to optimize each learning method by exploring a large variety of control parameters and model variations.

For example, with KNN we not only vary K, but use a variety of distance functions and also explore variants such as locally weighted averaging and distance-weighted KNN. When the model space for KNN is explored this thoroughly, KNN lags behind the best model by only about 0.0150. KNN performs best when attributes are weighted by gain ratio instead of unweighted Euclidean distance. We suspect even smarter distance functions would improve KNN's performance. We are not sure what would have to be done to make single decision trees competitive with the other models. The decision trees that performed best typically used Bayesian smoothing as opposed to Laplacian smoothing or pruning and no smoothing. Bagging trees with Bayesian smoothing also yielded excellent performance. Although single ID3 trees perform poorly, bagged ID3 trees perform well.

On most problems, SVMs with RBF kernels perform better than SVMs with other kernels. (Because more SVMs with RBF kernels are trained than the linear or polynomial kernels, the RBF SVMs have a small advantage.) The width of the RBF kernel that gives best performance changes with the problem. With KNN, rank-based metrics such as AUC favor much larger values of K than other metrics such as accuracy. The best neural nets typically have large hidden layers. Nets with 32 or 128 hidden units typically performed best.

2.1 Computational Cost

With neural nets there are many parameters to adjust: net size and architecture, backprop method, update interval, learning rate, momentum, etc. Because each parameter can affect performance, both singly and in combinations, many different nets must be trained to adequately explore the parameter space. As expected, ANNs were one of the most expensive algorithms we trained, particularly nets with 128 hidden units.

SVMs also require adjusting many parameters, though fewer than ANNs. SVM parameters include the kernel, kernel parameters, and the regularization parameter. The kernel and kernel parameters have a big impact on SVM performance. We trained 121 SVMs on each problem. While most SVMs trained fast, some SVMs took much longer to train, and a few never finished. It is the cost of the few SVMs that take long to train that made SVMs as expensive as neural nets.

Although we experimented with a variety of KNN methods, distance measures, and many parameter settings for each method, KNN proved less expensive than neural nets and SVMs because our training sets contain only 4k points. Larger training sets would have made KNN much more expensive.

The decisions that have to be made with boosted trees and stumps are what base tree/stump type to boost and how many boosting iterations to do. What makes boosting expensive in practice is that it is not easy to parallelize. Training one tree or stump is fast, but training thousands takes time. Like boosting, bagging requires experimenting with the base tree type. Unlike boosting, however, bagging is easy to parallelize and usually only 25-100 iterations are needed. This makes bagged trees cheap, and rather attractive given their good performance. Simple unbagged/unboosted trees are the least expensive method we examined, even when many tree types are tried, but their performance was not that good.

3 Ensembles to Optimize AUC

The previous section compares the AUC performance of the best single models that we could train with each learning algorithm. In this section we will use the models trained in the previous section to try to build an ensemble that yields even better performance. An ensemble is a collection of models whose predictions are combined by weighted averaging or voting. Diettrich [6] states that "A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are accurate and diverse."

Many methods have been proposed to generate accurate, yet diverse, sets of models. Bagging [4] trains models of one type (e.g., C4 decision trees) on bootstrap samples of the training set. Opitz [16] bags features instead of training points. Boosting [20] generates a potentially more diverse set of models than bagging by weighting the training set to force new models attend to those points that are difficult to classify correctly. Error-correcting-output-codes (ECOC) [7] creates models with decorrelated errors by training models for multiclass problems on different dichotomies. Munro and Parmanto [15] created diverse neural nets via competition among nodes.

In this section we use the models trained for the empirical comparison in the previous section to build ensembles that have better AUC performance than any of the individual models. In total, we have about 2000 models to select from on each problem. Some of the models have excellent AUC performance. Many models, however, have mediocre or even poor AUC performance. Rather than combine all models, good and bad, in an ensemble, we use forward stepwise selection from the library of models to find a subset of models that when combined in an ensemble yield excellent performance. The basic ensemble selection procedure is very simple:

1. Start with the empty ensemble.
2. Add to the ensemble the model in the library that maximizes the ensemble's AUC on a hillclimbing test-set.
3. Repeat Step 2 for a fixed number of iterations or until all the models have been used.
4. Return the ensemble from the nested set of ensembles that has maximum AUC on the hillclimb set.

Models are added to an ensemble by averaging their predictions with the models already in the ensemble. This makes adding a model to the ensemble *very* fast, allowing ensembles with excellent performance to be found in minutes from collections of 2000 models. Moreover, the selection procedure allows us to optimize the ensemble to any easily computed performance metric such as AUC. This is important because it is difficult to optimize most of the base-level learning methods to metrics such as AUC.

The parameters we vary with each learning algorithm to generate the 2000 models are listed in the Appendix. Note that we do not determine what parameters yield best performance when training these models. All models are added to the library no matter how good or bad they are. For simplicity, we cache the predictions each model makes on the train and hillclimbing sets. This simplifies working with the library and makes model selection faster because the models do not have to be executed during selection.

This simple forward model selection procedure is fast and effective, but, because there are thousands of models to select from, sometimes overfits to the hillclimbing set, reducing ensemble performance. We made three modifications to this selection procedure to reduce overfitting. These are discussed in the next three subsections. Each of these methods may be useful in other applications where forward stepwise selection is prone to overfitting, such as in feature selection [12].

3.1 Selection with Replacement

With forward model selection *without* replacement, performance improves as the first models are added to the ensemble, peaks, and then quickly declines. Performance drops quickly because the best models in the library have been used and selection must now choose from models that are not so good and thus may reduce ensemble performance. Figure 1 shows this behavior for root-mean-squared-error. Unfortunately, most error metrics including AUC yield much bumpier graphs than this when hillclimbing is done with a small data set, making it difficult to reliably pick a good stopping point. The loss in performance can be significant if the true peak is missed.

Figure 1 shows that selecting models with replacement greatly reduces this problem. Selection with replacement allows models to be added to the ensemble multiple times. Once peak performance is reached, if the unused models all hurt ensemble performance, selection will prefer to add models that have been added before rather than suffer a drop in performance. This not only flattens the performance curve past the

peak, but allows the ensemble to weight the models. Models added to the ensemble multiple times receive more weight, allowing selection to fine tune the ensemble.

Selection with replacement flattens the curve so much that a test set is not needed to determine when to stop adding models to the ensemble. The hillclimbing set can be used to stop hillclimbing. This means ensemble selection does not need more test sets than the base-level models would have used to select model parameters. Ensemble selection uses the validation set that base-level models would use for parameter selection to do both parameter *and* model selection.

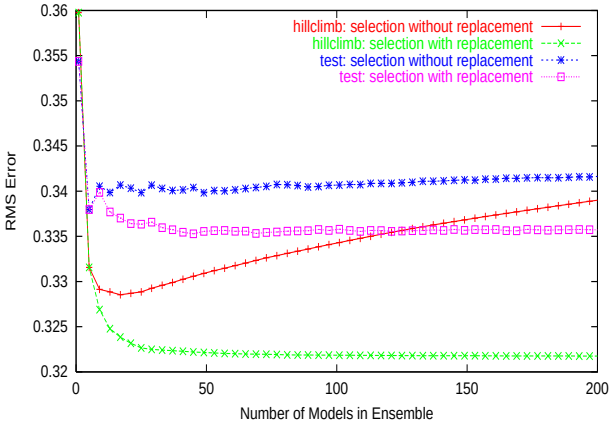

Figure 1. Selection With and Without Replacement.

3.2 Sorted Ensemble Initialization

Forward selection sometimes overfits early in selection when ensembles are small because sometimes it is possible to find a few models that when averaged together yield unrealistically good performance on the hillcimbing set. One way to prevent this is to initialize ensembles with more models. Instead of starting with an empty ensemble, we sort the models in the library by their performance, and put the best N models in the ensemble. N is chosen by looking at performance on the hillclimbing set. This typically adds 5-25 of the *best* models to an ensemble *before* greedy stepwise selection begins. Since each of the N best models performs well, they form a strong initial ensemble and it is more difficult for greedy selection to find models that overfit when added to the ensemble.

3.3 Bagged Ensemble Selection

As the number of models in a library increases, the chances of finding combinations of models that overfit the hillclimbing set increases. Bagging can minimize this problem. We reduce the number of models selection can choose from by drawing a random sample of models from the library and selecting from that sample. If a particular combination of M models overfits, the probability of those M models being in a random bag of models is less than $(1-p)^M$ for p the fraction of models in the bag. We use $p = 0.5$, and bag ensemble selection 20 times to insure that the best models will have many opportunities to be selected. The final ensemble is the average of the 20

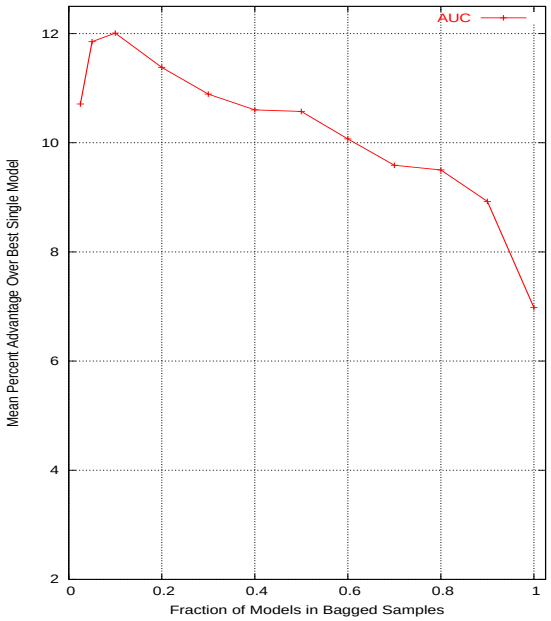

Figure 2. Benefit of Bagged Selection as a function of p (average over 7 problems). The right side of the graph at $p = 1$ represents no bagging.

ensembles. Having bags of ensembles may seem complex, but each ensemble is just a weighted average of models, so the average of a set of ensembles also is a simple weighted average of the base-level models.

Figure 2 shows the benefit to AUC of ensemble selection due to bagging as p, the fraction of models in the bags, varies from 1 down to 0.025. The percent benefit in the figure is percent increase in AUC averaged over the seven test problems. At $p = 1$, bags contain all models and this is equivalent to no bagging at all. The graph shows that bagged ensemble selection improves AUC performance about 10% at $p = 0.5$ and as much as 12% at $p = 0.1$. The results in Table 4 are for $p = 0.5$, a value selected before looking at the results in Figure 2. We are currently experimenting with using cross validation to pick a near optimal value of p for each problem individually.

3.4 Ensemble Selection AUC Performance

Table 4 compares the AUC performance of ensemble selection to the models trained with each learning algorithm that had best AUC when selection is done using 1k validation sets. Ensemble selection uses these 1k validation sets to perform model selection for the ensembles. The bottom part of this table has the same entries as Table 3. Again, bold entries in the bottom of this table show which model type had the best AUC. The top line of Table 4 shows the AUC of ensemble selection. All entries in this line are higher than the entries in the bottom of the table.

Because we generate so many different models, libraries usually contain a few models with excellent AUC performance on any problem. Just selecting the best single model from a library yields remarkably good AUC performance.

Ensemble selection, however, does find ensembles that outperform the best individual models. Ensemble selection out-

Table 4. AUC of the best model from each learning algorithm on each problem selected using 1K validation sets.

MODEL	COVER_TYPE	ADULT	LETTER.P1	LETTER.P2	MEDIS	SLAC	HYPER_SPEC	MEAN
ENS.SEL.	**0.9161**	**0.9125**	**0.9977**	**0.9948**	0.8381	**0.8086**	**0.9815**	**0.9214**
AVERAGE	0.9034	0.9054	0.9941	0.9900	**0.8419**	0.8085	0.9725	0.9165
BAYES	0.9016	0.9073	0.9970	0.9918	0.8413	0.8019	0.9698	0.9158
STACKING	0.8397	0.8330	0.9871	0.9905	0.6199	0.7097	0.9679	0.8497
BST-DT	**0.9152**	0.8902	**0.9974**	**0.9944**	0.8232	0.8004	**0.9757**	**0.9138**
BAG-DT	0.9007	0.9057	0.9888	0.9806	0.8216	**0.8009**	0.9623	0.9087
SVM	0.8687	0.8980	0.9946	0.9918	0.8288	0.7954	0.9694	0.9067
ANN	0.8681	0.8980	0.9929	0.9772	0.8254	0.7940	0.9673	0.9033
KNN	0.8750	0.8790	0.9911	0.9870	0.8253	0.7690	0.9525	0.8970
BST-STMP	0.8428	**0.9074**	0.9813	0.9136	**0.8310**	0.7746	0.9557	0.8866
DT	0.8423	0.8742	0.9596	0.9454	0.7716	0.7721	0.9212	0.8695

performs the best models trained with the seven learning methods on all test problems. This suggests that using different learning methods and parameter settings is an effective procedure for generating libraries that contain a diverse set of good-performing models. Combining models with Bayesian averaging, or stacking [22] with logistic regression, yielded performance no better than the best single models, suggesting that these ensemble methods have difficulty combining 2000 models [8].

The average AUC of ensemble selection across the problems is 0.9214. The average performance of the best single models (the average of the bold entries in the bottom of the table) is 0.9174. Although ensemble selection outperforms the best models seven out of seven times, the increase in AUC is only about 0.0040 compared to the best models, or an increase in AUC of about 0.5%. One might question if this small increase in AUC is worth the expense of ensemble selection. The main expense in ensemble selection is training the thousands of base-level models from which ensemble selection selects. Because no one learning method consistently outperforms the other learning methods, one needs to train all of these models if one is to find the model that has best AUC performance. If one has gone through the expense to train all of these models, selecting an ensemble from the models incurs little additional cost, and, as shown in the table, does yield better AUC. The main concern with ensemble selection probably is the complexity and poor intelligibility of the final model (which averages over a number of models of different types), as opposed to the small increase in computational cost in finding the ensemble.

3.5 Models Selected by the Ensembles

Table 5. Aggregate Weight Given to Different Types of Models in the Ensembles.

MODEL	ADULT	COV_TYPE
ANN	0.3646	0.0382
KNN	0.0368	0.2518
SVM	0.2841	0.1039
DT	0.0879	0.2381
BAG-DT	0.0038	0.0092
BST-DT	0.0572	0.2784
BST-STMP	0.1657	0.0804

Table 5 shows the total weight given to each type of model on the ADULT and COVER_TYPE test problems. KNN, BST-DT, and plain DT receive the most weight in the ensemble for COVER_TYPE. The weights are strikingly different for ADULT where ANN, SVM, and BST-STMP receive the most weight. This suggests that ensemble selection is able to exploit the different strengths and biases of the different learning algorithms when optimizing the ensemble's AUC performance on each problem. We probably can not specify in advance which kinds of models need to be trained, and what weight they should be given in the ensemble.

3.6 Optimizing To Any Performance Metric

ANNs usually are trained to minimize cross entropy or squared error. Trees and SVMs usually maximize accuracy. Boosting also is designed to maximize accuracy. Some metrics such as precision/recall and ROC are hard to optimize to. Because model averaging is fast, ensemble selection can try adding every model in the library to the ensemble at each step. If the performance of each of these ensembles can be evaluated quickly on the metric, the ensemble can optimize to that metric by this greedy, brute force search. A good ensemble usually will be found if some base-level models or if combinations of them yield good performance on that metric. Although we do not know how to optimize the base-level models to metrics such as AUC, the ensemble can be optimized to AUC.

Although in this paper we present results for the area under the full ROC plot, in many applications specific regions of the ROC plot are more important than others. For example, the high-precision end of the ROC plot typically is most important for information retrieval, but the high-recall end of the ROC plot typically is most important for medical screening tests. Because ensemble selection can optimize to any easily computed performance metric, the method can be used to find ensembles that maximize the area under any well-specified part of the ROC plot.

3.7 Validation and Hillclimbing Sets

Ensemble selection uses the *validation* set to "train" an ensemble, but hillclimbing on this set does not give ensemble selection an unfair advantage over single models. The validation set would be needed to select the parameters for each

algorithm (parameter selection), and then to pick the best algorithm (model selection). Ensemble selection uses the same validation set for parameter selection, model selection, and ensemble creation.

With some algorithms validation data can be put back in the train set and the model retrained once parameters are selected. This can also be done with ensemble selection. (Only the models used in the ensemble would be retrained.) Any strategy for using and reusing validation sets, including cross validation, can be used with ensemble selection. We currently are running experiments with 5-fold cross validation to increase the size of the hillclimbing set to include all of the training data, not just the held-out samples.

3.8 Computational Complexity

Training all of the models used by ensemble selection is expensive, but most of these models would need to be trained even if we only wanted to find the single model with the best performance on each problem. Because models are independent, it is easy to parallelize model creation and distribute training across machines. It takes about 48 hours to train the 2000 models using a cluster of ten Linux machines. Model training is automated. There is no parameter tuning or examining performance on validation sets. Usually, no one model is critical, so it is not necessary to wait until all models are trained to begin doing ensemble selection. This provides an any-time flavor to ensemble selection: ensembles can be trained using whatever models are available when the ensemble is needed. It is easy to add more models later.

Forward stepwise ensemble selection is efficient. Adding a model to an ensemble only requires averaging a model's predictions with the ensemble's predictions, which is $O(D)$ for D the size of the hillclimbing set. If there are M models to choose from, this is done M times for each selection step. If selection is run K steps, the cost of ensemble selection is only O(D*M*K) assuming the metric can be computed in O(D). If the metric is more expensive than O(D) (e.g., ROC requires sorting and thus is O(D*logD)), recomputing the metric dominates. Using a JAVA implementation, selecting an ensemble from a library of $M = 2000$ models, a hillclimbing set with $D = 1000$ points, and $K = 200$ takes about a minute on a medium-power workstation. If selection is bagged 20 times, it takes about 20 minutes to build the final ensemble.

4 Related Work

There are few comprehensive empirical studies comparing learning algorithms, and even fewer that use AUC as a performance metric. STATLOG is perhaps the best known empirical study of performance of learning algorithms [11]. STATLOG was a very comprehensive study when it was performed, but since then several new learning algorithms have emerged (e.g., bagging, boosting, SVMs) that have excellent performance. Also STATLOG didn't use AUC as a performance criterion. LeCun et al. [13] presents a study that compares several learning algorithms (including SVMs) on a handwriting recognition problem using three performance criteria: accuracy, rejection rate, and computational cost. Cooper et al. [5] present results from a study that evaluates nearly a dozen learning methods on a real medical data set using both accuracy and an AUC-like metric. Lim et al. [14] perform an empirical comparison of decision trees and other classification methods using accuracy as the main criterion. Bauer and Kohavi [1] present an impressive empirical analysis of ensemble methods such as bagging and boosting, but do not examine the AUC performance of these methods. Provost and Domingos [18] examine the issue of predicting probabilities with decision trees, including smoothed and bagged trees. Bradley [3] uses accuracy and AUC to compare the performance of a number of algorithms. Provost and Fawcett [19] discusses the importance of evaluating learning algorithms on metrics other than accuracy such as ROC.

There is a large body of work in ensemble learning. We mentioned some of the representative work in the area in Section 3. However, as far as we know, the ensemble selection method proposed in this paper is the first method that can directly optimize any performance metric, in particular AUC.

5 Conclusions

The best learning algorithms for AUC on the seven test problems are boosted full decision trees, bagged decision trees, neural nets, and SVMs. Surprisingly, maximum margin methods such as SVMs and boosted decision trees yield excellent AUC performance. We had not expected that maximizing the margin to a decision boundary would provide a good basis for ordering cases that fall far from those boundaries. We were able to obtain surprisingly good AUC performance with each learning algorithm by very thoroughly tuning each algorithms parameters. Nevertheless, KNN, plain decision trees (including smoothed probabilistic trees), and boosted stumps usually did not yield AUC performance that was competitive with the best models.

For the seven data sets we examined, model selection using 1k validation sets was nearly as good as optimal model selection using the final test sets.

Ensemble selection uses forward stepwise selection to select a subset of models that optimize the ensemble's AUC performance. Using a variety of learning algorithms and parameters for these algorithms proved to be an effective way of generating a collection of diverse, high quality models, some of which have excellent AUC. In our experiments with seven test problems, ensemble selection consistently found ensembles that had better AUC than the best models trained with any of the learning method, including models trained with bagging and boosting.

6 Appendix: Building Model Libraries

KNN: we use 26 values of K ranging from $K = 1$ to $K = |trainset|$. We use KNN with Euclidean distance and Euclidean distance weighted by gain ratio. We also use distance weighted KNN, and locally weighted averaging. The kernel widths for locally weighted averaging vary from 2^0 to 2^{10} times the minimum distance between any two points in the train set.

ANN: we train nets with gradient descent backprop and vary the number of hidden units {1, 2, 4, 8, 32, 128} and the momentum {0, 0.2, 0.5, 0.9}. We don't use validation sets to do

weight decay or early stopping. Instead, we stop the nets at many different epochs so that some nets underfit or overfit.

DT: we vary the splitting criterion, pruning options, and smoothing (Laplacian or Bayesian smoothing). We use all of the DT models in Buntine's IND package: Bayes, ID3, CART, CART0, C4, MML, and SMML. We also generate trees of type C44 (C4 with no pruning), C44BS (C44 with Bayesian smoothing), and MMLLS (MML with Laplacian smoothing). See [18] for descriptions of C44.

BAG-DT: we bag 25 trees of each type. Each tree trained on a bootstrap sample is added to the library, as well as the final bagged ensemble that averages all these trees. With **BST-DT** we boost each tree type. Boosting can overfit, so we add boosted DTs to the library after {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048} steps of boosting. With **BST-STMP** we use stumps (single level decision trees) with 5 different splitting criteria, each boosted {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192} steps.

SVMs: we use most kernels in SVMLight [10] {linear, polynomial degree 2 & 3, radial with width {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2}} and vary the regularization parameter C by factors of ten from 10^{-7} to 10^{3}. The output range of SVMs is $[-\infty, +\infty]$ instead of $[0, 1]$. To make the SVM predictions compatible with other models, we use Platt's method to convert SVM outputs to probabilities by fitting them to a sigmoid [17].

REFERENCES

[1] Eric Bauer and Ron Kohavi, 'An empirical comparison of voting classification algorithms: Bagging, boosting, and variants', *Machine Learning*, **36**(1-2), (1999).

[2] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[3] Andrew P. Bradley, 'The use of the area under the ROC curve in the evaluation of machine learning algorithms', *Pattern Recognition*, **30**(7), 1145–1159, (1997).

[4] Leo Breiman, 'Bagging predictors', *Machine Learning*, **24**(2), 123–140, (1996).

[5] G. F. Cooper, C. F. Aliferis, R. Ambrosino, J. Aronis, B. G. Buchanan, R. Caruana, M. J. Fine, C. Glymour, G. Gordon, B. H. Hanusa, J. E. Janosky, C. Meek, T. Mitchell, T. Richardson, and P. Spirtes, 'An evaluation of machine learning methods for predicting pneumonia mortality', *Artificial Intelligence in Medicine*, **9**, (1997).

[6] Thomas G. Dietterich, 'Ensemble methods in machine learning', *First International Workshop on Multiple Classifier Systems*, 1–15, (2000).

[7] Thomas G. Dietterich and Ghulum Bakiri, 'Solving multiclass learning problems via error-correcting output codes', *Journal of Artificial Intelligence Research*, **2**, (1995).

[8] Pedro Domingos, 'Bayesian averaging of classifiers and the overfitting problem', in *Proc. 17th International Conf. on Machine Learning*, pp. 223–230. Morgan Kaufmann, San Francisco, CA, (2000).

[9] A. Gualtieri, S. R. Chettri, R.F. Cromp, and L.F. Johnson, 'Support vector machine classifiers as applied to aviris data', in *Proc. Eighth JPL Airborne Geoscience Workshop*, (1999).

[10] T. Joachims, 'Making large-scale svm learning practical', in *Advances in Kernel Methods*, (1999).

[11] R. King, C. Feng, and A. Shutherland, 'Statlog: comparison of classification algorithms on large real-world problems', *Applied Artificial Intelligence*, **9**(3), 259–287, (May/June 1995).

[12] Ron Kohavi and George H. John, 'Wrappers for feature subset selection', *Artificial Intelligence*, **97**(1-2), (1997).

[13] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik, 'Comparison of learning algorithms for handwritten digit recognition', in *International Conference on Artificial Neural Networks*, eds., F. Fogelman and P. Gallinari, Paris, (1995). EC2 & Cie.

[14] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, 'An empirical comparison of decision trees and other classification methods', Technical Report 979, Madison, WI, (30 1997).

[15] P. Munro and B. Parmanto, 'Competition among networks improves committee performance', in *Advances in Neural Information Processing Systems*, (1996).

[16] David Opitz, 'Feature selection for ensembles', in *AAAI/IAAI*, pp. 379–384, (1999).

[17] J. Platt, 'Probabilistic outputs for support vector machines and comparison to regularized likelihood methods', in *Advances in Large Margin Classifiers*, eds., A.J. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans, pp. 61–74, (1999).

[18] F. Provost and P. Domingos, 'Tree induction for probability-based rankings', *Machine Learning*, **52**(3), (2003).

[19] Foster J. Provost and Tom Fawcett, 'Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions', in *Knowledge Discovery and Data Mining*, pp. 43–48, (1997).

[20] R. Schapire, 'The boosting approach to machine learning: An overview', in *In MSRI Workshop on Nonlinear Estimation and Classification*, (2001).

[21] V. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, New York, 1998.

[22] D. H. Wolpert, 'Stacked generalizatio', *Neural Networks*, **5**, 241–259, (1992).