Ensemble Selection from Libraries of Models

Rich Caruana CARUANA@CS.CORNELL.EDU
Alexandru Niculescu-Mizil ALEXN@CS.CORNELL.EDU
Geoff Crew GC97@CS.CORNELL.EDU
Alex Ksikes AK107@CS.CORNELL.EDU
Department of Computer Science, Cornell University, Ithaca, NY 14853 USA

Abstract

We present a method for constructing ensembles from libraries of thousands of models. Model libraries are generated using different learning algorithms and parameter settings. Forward stepwise selection is used to add to the ensemble the models that maximize its performance. Ensemble selection allows ensembles to be optimized to performance metric such as accuracy, cross entropy, mean precision, or ROC Area. Experiments with seven test problems and ten metrics demonstrate the benefit of ensemble selection.

1. Introduction

An ensemble is a collection of models whose predictions are combined by weighted averaging or voting. Dietterich (2000) 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 (Breiman, 1996) trains models of one type (e.g., C4 decision trees) on bootstrap samples of the training set. Opitz (1999) bags features instead of training points. Boosting (Schapire, 2001) 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. Sullivan et al. (2000) boost features instead of training points. Error-correcting-output-codes (ECOC) (Dietterich & Bakiri, 1995) creates models with decorrelated errors by training models for multi-class problems on different dichotomies. Munro and Parmanto (1996) created diverse neural nets via competition among nodes.

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

Here we generate diverse sets of models by using many different algorithms. We use Support Vector Machines (SVMs), artificial neural nets (ANNs), memory-based learning (KNN), decision trees (DT), bagged decision trees (BAG-DT), boosted decision trees (BST-DT), and boosted stumps (BST-STMP). For each algorithm we train models using many different parameter settings. For example, we train 121 SVMs by varying the margin parameter C, the kernel, and the kernel parameters (e.g. varying gamma with RBF kernels.)

We train about 2000 models for each problem. Some models have excellent performance, equal to or better than the best models reported in the literature. Other models, however, have mediocre or even poor performance. Rather than combine good and bad models in an ensemble, we use forward stepwise selection from the library of models to find a subset of models that when averaged together 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 performance to the error metric on a hillclimb (validation) 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 performance on the hillclimb (validation) 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 libraries with 2000 models. Moreover, the selection procedure allows us to optimize the ensemble to any easily computed performance metric. We evaluate the performance of ensemble selection on ten performance metrics. We believe this is the first time a learning method has been evaluated across such a wide variety of performance metrics.

On each performance metric we compare ensemble selection to the model in the library that performs best on that metric. Because we generate so many different models, libraries usually contain a few models with excellent performance on any performance metric. Just selecting the best single model from a library yields remarkably good performance. Ensemble selection, however, finds ensembles that outperform the best single models. This suggests that using different learning methods and parameter settings generates libraries containing a diverse set of good-performing models.

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

2. Improving Ensemble Selection

The simple forward model selection procedure presented in the Introduction is fast and effective, but sometimes overfits to the hillclimbing set, reducing ensemble performance. We made three additions to this selection procedure to reduce overfitting. These are discussed in the next three sub-sections. These methods may be useful in other applications where forward stepwise selection is prone to overfitting, such as in feature selection (Kohavi & John, 1997).

2.1. Selection with Replacement

With model selection *without* replacement, performance improves as the best models are added to the ensemble, peaks, and then quickly declines. Performance drops because the best models in the library have been used and selection must now add models that hurt the ensemble. Figure 1 shows this behavior for root-mean-squared-error. Unfortunately, most error metrics yield much bumpier graphs than this when hillclimbing is done with small data sets, making it difficult to reliably pick a good stopping point. The loss in performance can be significant if the peak is missed.

Figure 1 also 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 adds models that were added before rather than hurt performance. This flattens the performance curve past the peak, and allows selection to fine tune ensembles by weighting models: models added to the ensemble multiple times receive more weight.

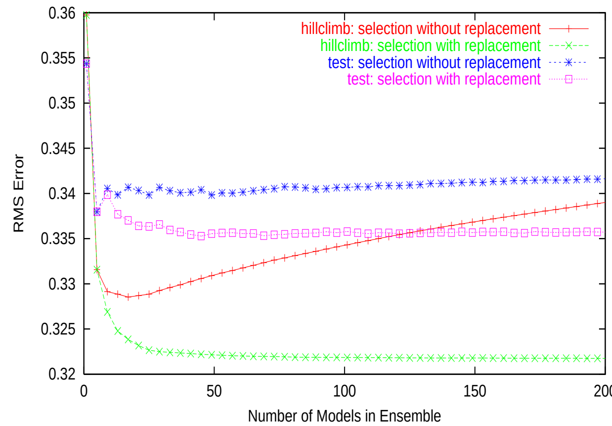

Figure 1. Selection With and Without Replacement.

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 to do both parameter *and* model selection.

2.2. Sorted Ensemble Initialization

Forward selection sometimes overfits early in selection when ensembles are small. One way to prevent this is to initialize ensembles with more models. Instead of starting with an empty ensemble, 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.

2.3. 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 ensembles. Bags of ensembles 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. Bagging is discussed in Section 5.3.

3. Data Sets

We experiment with seven problems: ADULT, COVER_TYPE, LETTER.p1, and LETTER.p2 from the UCI Repository (Blake & Merz, 1998), MEDIS, a pneumonia data set, SLAC, data from collaborators at the Stanford Linear Accelerator, and HYPER_SPECT, the Indian Pines hyperspectral data (Gualtieri et al., 1999). ADULT, MEDIS and SLAC are binary problems. COVER_TYPE, LETTER, and HYPER_SPECT are 7, 26, and 16 class problems, respectively. We converted these to binary because many of the metrics we study are defined on binary problems, and because learning methods such as SVMs and boosting are easier to apply to binary problems. COVER_TYPE was converted to binary by treating the largest class as class 1. LETTER was converted two ways. LETTER.p1 treats the confusable letter 'O' as class 1 and the remaining 25 letters as class 0, yielding a very unbalanced binary problem. LETTER.p2 uses letters 1-13 as class 0 and letters 14-26 as class 1, yielding a difficult, but balanced, problem. HYPER_SPECT was converted to binary by treating the large confusable class Soybean-Mintil as class 1.

These data sets were selected because they are large enough to allow moderate size train and validation sets, and still have data left for large final test sets. For our experiments, we used training sets of 5000 points. Each training sample was split into a train set of 4000 points and a hillclimbing/validation set of 1000 points. The final test sets for most of these problems contain 20,000 points – enough to make discerning small differences in performance reliable.

4. Performance Metrics

We use ten performance metrics: accuracy (ACC), root-mean-squared-error (RMS), mean cross-entropy (MXE), lift (LFT), precision/recall break-even point (BEP), precision/recall F-score (FSC), average precision (APR), ROC Area (ROC), and a measure of probability calibration (CAL). The tenth metric is $SAR = (ACC + ROC + (1 - RMS))/3$. SAR is a robust metric to use when the correct metric is unknown. An attractive feature of ensemble selection is that it *can* optimize to metrics such as SAR. We compare performance using ten metrics because different metrics are appropriate in different settings and because learning methods that perform well on one metric do not always perform well on other metrics.

5. Empirical Results

In this section we compare ensemble selection to the best models trained with any of the learning algorithms, and also to several other ensemble methods.

5.1. Normalized Scores

Performance metrics such as ACC or RMS range from 0 to 1, while others (LFT, MXE) range from 0 to p where p depends on the data. For some metrics lower values indicate better performance. For others higher values are better. The baseline rates of metrics such as ROC are independent of the data, while others such as ACC have baseline rates that depend on the data. If baseline ACC = 0.98, ACC = 0.981 probably is poor performance, but if the Bayes optimal ACC = 0.60, achieving ACC = 0.59 might be excellent performance.

To allow averaging across problems and metrics, we convert performances to a normalized $[0, 1]$ scale. On this scale, 0 represents baseline performance for the metric and problem (e.g., 0.5 for ROC, or the baseline ACC for each problem), and 1 represents the best performance seen for any model on the final test set.[1]

The bottom of Table 1 shows normalized scores for each learning algorithm when its parameters are selected for each problem and metric using the validation sets ensemble selection hillclimbs on. Entries in the table are averages over the seven problems. Scores near 1 indicate that a model performs close to the best performance observed on all seven problems. Negative entries mean the best models perform below baseline. Bold entries in the bottom of Table 1 show which learning algorithms yield the best performance on each metric. The last column is the mean normalized score across the ten metrics.

Ignoring the ensemble methods in the top of the table, the best algorithms overall are bagged trees, SVMs and ANNs with mean normalized scores 0.8842, 0.8634, and 0.8557, respectively. Boosted trees do not perform well on the probability metrics (RMS, MXE,

[1] We would set the top of the scale to the Bayes optimal performance for each problem and metric if we knew it.

Table 1. Normalized Scores for the Best Single Models of Each Type (bottom of tbl), and for Ensemble Selection, Bayesian Model Averaging, Stacking with Regression, Averaging All Models, and Picking the Best Model of Any Type (top of tbl).

MODEL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	CAL	SAR	MEAN
ENS. SEL.	**.9560**	**.9442**	**.9916**	**.9965**	**.9846**	**.9786**	**.9795**	**.9808**	**.7870**	**.9958**	**.9554**
BAYESAVG	.9258	.8906	.9785	.9851	.9773	.9557	.9504	.9585	.6678	.9412	.9211
BEST	.9357	.9217	.9754	.9876	.9588	.9581	.9601	.9673	.3197	.9439	.8872
AVG_ALL	.8363	.8007	.9815	.9878	.9721	.9606	.8271	.8086	.3639	.9156	.8376
STACK_LR	.2753	.7772	.8352	.7992	.7860	.8469	.3317	-.9897	.1414	.7050	.4226
BAG	.8114	.8609	.9465	.9674	.9416	.9220	.8588	**.8942**	**.7549**	.9036	**.8842**
SVM	.8134	**.9092**	.9480	.9621	.9335	.9377	**.8767**	.8778	.5124	**.9055**	.8634
ANN	.8769	.8752	.9487	.9552	.9167	.9142	.8532	.8634	.4977	.8956	.8557
BST-DT	**.8904**	.8986	**.9574**	**.9778**	**.9597**	**.9427**	.6066	.6107	.1909	.8710	.7816
KNN	.7557	.8463	.9095	.9370	.8847	.8890	.7612	.7354	.2754	.8470	.7771
DT	.5261	.7891	.8503	.8678	.7674	.7954	.5564	.6243	.0234	.7445	.6445
BST-STMP	.7319	.7903	.9064	.9187	.8610	.8336	.3038	.2861	-1.485	.6589	.4607

and CAL), but are excellent on the threshold metrics (ACC, FSC, and LFT) and ordering metrics (ROC, APR, and BEP). If mean performance was measured over just these six metrics, boosted trees would outperform bagged trees, SVMs, and ANNs.

The top of Table 1 shows normalized scores for ensemble selection, Bayesian model averaging, the best individual models of any type (BEST), a simple average of all models (AVG_ALL), and stacking with logistic regression (STACK_LR). Stacking (Wolpert, 1992) with logistic regression performs poorly because regression overfits dramatically when there are 2000 highly correlated input models and only 1k points in the validation set. An unweighted average of all the models (AVG_ALL) is better than weighting the models via regression because unweighted averaging cannot overfit the validation set. Averaging, however, performs worse than just picking the best single models because some of the models have poor performance and thus hurt the ensembles. As expected, picking the best single model from the library (BEST) performs better than any one learning algorithm in the bottom of the table because it can pick the best model from any learning method for each metric and test problem.

Bayesian model averaging (Domingos, 2000) has a mean score of 0.9211, significantly better than bagged trees (the best single model). In Bayesian model averaging, each model in the library is weighted by the likelihood of the data (validation set) given the model when model predictions are treated as probabilities:

$$W_M = \prod_{i=1}^{N} Targ_i * Pred_i + (1 - Targ_i) * (1 - Pred_i)$$

where W_M is the weight of model M, $Targ_i$ is the 0/1 target for case i, and $Pred_i$ is the predicted probability that case i is class 1. Although Bayesian model averaging has higher mean performance than BEST, it outperforms BEST only on 3 metrics, while BEST outperforms it on 7. The main reason why BEST has lower mean performance than Bayesian averaging is because BEST performs poorly on the CAL metric.[2]

With a mean normalized score of 0.9554, ensemble selection is the clear winner. It outperforms the other ensemble methods (and the best individual models) on all 10 of the 10 metrics (significant at $p = 0.001$). We believe ensemble selection consistently outperforms the other ensemble methods for two reasons:

- ensemble selection is able to optimize the ensemble differently for each performance metric
- overfitting is a serious problem when there are 2000 models to combine. The methods presented in Section 2 to combat overfitting contribute to ensemble selection's excellent performance.

The consistently strong performance of ensemble selection suggests that using many different learning methods and parameter settings is an effective way of generating a diverse collection of models. The consistent performance also suggests that forward stepwise selection is an effective way of selecting high-performance ensembles from these models for a variety of performance metrics if overfitting is carefully controlled.

[2]CAL a high variance metric. When selecting the best model from 2000 models using a small 1k validation set for a high variance metric, sometimes models that look good on the validation set but perform poorly on the final test set are selected. Because of this, BEST actually performs worse than bagged trees, SVMs, and ANNs on CAL.

Table 2. Percent Reduction in Loss of Ensemble Selection Over the Best Models From Any Learning Algorithm.

PROBLEM	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	CAL	SAR	MEAN
ADULT	3.63	1.98	6.28	5.40	5.82	3.60	0.71	1.36	1.56	4.02	3.436
COVER_TYPE	*-0.42*	*-1.08*	4.63	*-1.27*	3.52	*-0.87*	3.79	6.66	9.63	4.05	2.864
LETTER.P1	17.47	26.72	0.10	41.88	28.75	15.02	24.82	15.34	*-5.73*	34.74	19.911
LETTER.P2	3.24	4.27	100.00	0.96	1.93	10.13	4.46	9.65	17.90	6.27	15.881
MEDIS	0.30	*-6.39*	0.83	2.53	4.00	6.53	0.04	0.12	19.22	1.57	2.875
HYPER_SPECT	5.40	5.29	5.98	20.96	27.28	13.55	7.71	18.75	3.14	11.91	11.997
SLAC	2.19	5.55	8.85	3.56	2.82	2.18	0.36	0.71	15.94	1.67	4.383
MEAN	4.54	5.19	18.10	10.57	10.59	7.16	5.98	7.51	8.81	9.18	8.76

5.2. Percent Reduction in Loss (Error)

In this section we compare ensemble selection to the best single models (BEST in Table 1) using percent reduction in error. One advantage of percent reduction in error compared to normalized scores is that normalized scores change as better models are found and shift the top of the scale up. The percent reduction in error from model A to B depends only on the performances of models A and B, not on some other model that defines the top of the performance scale.

To calculate percent reduction in error, we first convert each metric to loss where 0 represents perfect performance and 1 represents worst performance. Perfect prediction yields $ACC = BEP = FSC = ROC = APR = SAR = 1$ and $RMSE = MXE = CAL = 0$, all of which have loss 0. An example will help. If ACC improves from 0.75 to 0.77, the losses are 0.25 and 0.23, respectively, and the percent reduction in error is 8%. One potential disadvantage of percent reduction in loss is that it gives more emphasis to reductions at low loss: reducing loss from 0.02 to 0.01 is a 50% reduction, wheras reducing loss from 0.20 to 0.19 (same absolute change), is only a 5% reduction. In some domains this bias is appropriate. In others it is not. Normalized scores do not have this bias.

Table 2 shows the percent reduction in loss for ensemble selection on the 7 test problems and 10 metrics, compared to the best models selected for each problem and metric. As with normalized scores, final performances are estimated on large final test sets not used for training and is the average over two trials with each problem. Positive entries in the table mean error was reduced, i.e. ensemble selection performed better.

Ensemble selection wins 64 of 70 times (significant at $p = 0.001$). On average, ensemble selection reduces loss 8.76% over the best models for each problem and metric. To make this concrete, if the best model has accuracy 90.00%, loss is 0.1000, and an 8.7% reduction in loss corresponds to reducing loss to 0.0913 or increasing accuracy to 90.87%. Similarly, if the best model has RMS 0.2000, an 8.7% reduction in loss corresponds to reducing RMS to 0.1826. Test sets contain 20,000 cases. Increasing accuracy 0.87% means 174 more cases are predicted correctly by the ensembles.

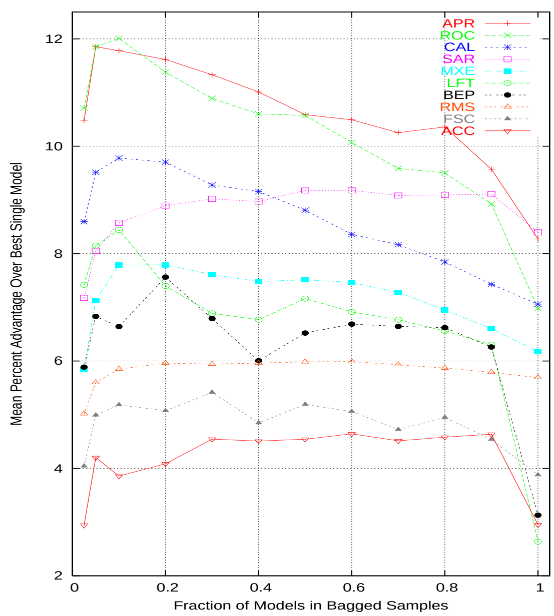

Figure 2. Benefit of Bagged Selection on the 10 Metrics. The right side of the graph at $p = 1$ represents no bagging.

The improvement is not dramatic, but consistently increasing accuracy 0.87% or reducing RMS 0.0174 compared to the best of 2000 models is impressive when the best models have such good performance. But we don't want to overstate the improvement of ensemble selection. For comparison, bagging or boosting trees yields a 20% reduction in loss compared to raw trees, 2.5 times the benefit of ensemble selection. But then, improving the performance of mediocre, high variance models such as decision trees is easier than improving the performance of the very best models.

5.3. Bagging to Minimize Overfitting

Figure 2 shows the percent reduction in loss of ensemble selection with bagging as p, the fraction of models in the bags, varies from 1 down to 0.025 for the 10 metrics averaged across the 7 problems. At $p = 1$, bags contain all models which is equivalent to not bagging. On average, bagged ensemble selection reduces loss an additional 2.5% at $p = 0.5$. The results in Tables 1 and 2 are for $p = 0.5$, a value we selected before looking at these results, and which Figure 2 shows is suboptimal. This 2.5% is about a third of the total 8.7% benefit we see with ensemble selection. Figure 2 (which uses the final test sets) suggests p in the range $0.1 - 0.3$ would yield further improvement. We are currently experimenting with using cross validation to pick a near optimal value of p for each problem and metric.

6. Case Study: Classifying Sub-Atomic Particles

Here we present a case study where optimizing an ensemble to the correct metric has impact on an application of machine learning to a particle physics problem. At the Stanford Linear Accelerator Center (SLAC), high energy particle beams are collided to generate subatomic particles. A major challenge in these experiments is to correctly classify the particle tracks. Performance is measured with the SLAC Q-score:

$$\mathrm{SLQ} = \varepsilon(1 - 2w)^2$$

where ε is the percent of events accepted for prediction, and w is the probability of misclassification. SLQ is an application-specific performance metric that estimates the statistical power of the model. Increasing SLQ by 5% is equivalent to having 5% more data, which potentially saves hundreds of thousands of dollars or more in accelerator time.

Bagged trees have the best SLQ performance, increasing SLQ 0.0355 (a 12% improvement) over well-optimized neural nets and decision trees. SLQ behaves like a cross between accuracy and calibration, so it is not surprising that bagged trees – the models with the best calibration performance – are best on SLQ. When an ensemble is optimized to SLQ with ensemble selection, SLQ performance increases 6% over the best bagged trees. This 6% increase in effective sample size represents a large potential savings in accelerator time.

7. Discussion and Future Work

7.1. Validation and Hillclimbing Sets

Ensemble selection uses the *validation* set to "train" ensembles. Hillclimbing on the validation set does not give ensemble selection an unfair advantage over other 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 validation set for parameter selection, model selection, and ensemble creation. With many 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. 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 1k samples.

7.2. Models Selected by the Ensembles

Table 3 shows the total weight given to each type of model for each metric on ADULT and COVER_TYPE. The average across the ten metrics (right column) shows that KNN and BST_DT receive the most weight in the ensemble for COVER_TYPE. The weights are strikingly different for ADULT where BST_STMP, ANN, and DT receive the most weight. There also are substantial differences between the model types preferred for different metrics. For example, on COVER_TYPE, FSC gives high weight to boosted trees and low weight to KNN, but these weights are reversed for RMS and LFT. Also, ANNs get modest weight for MXE and RMS (what the ANNs optimize), but low weight for ACC and LFT. This suggests that ensemble selection is able to exploit the different strengths and biases of the different learning algorithms when optimizing an ensemble to each metric and to each problem.

7.3. Computational Cost

Building libraries is expensive. 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 use the library. 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. Libraries can be built before the performance metric is known because the libraries themselves do not depend on the metric that will be used to optimize the ensemble. This makes model libraries very reusable.

Table 3. Aggregate Weight Given to Different Types of Models in the Ensembles.

ADULT	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	SAR	AVG_WT
ANN	0.0709	0.1316	0.1012	0.3646	0.4304	0.2432	0.1667	0.0941	0.5732	0.2720
KNN	0.0205	0.0148	0.5865	0.0368	0.0286	0.0486	0.0000	0.0000	0.0492	0.0981
SVM	0.0006	0.0000	0.1096	0.2841	0.2743	0.0924	0.0574	0.0000	0.0224	0.1051
DT	0.0201	0.0346	0.0070	0.0879	0.0491	0.0188	0.7456	0.8674	0.2345	0.2581
BAG_DT	0.0021	0.0008	0.0000	0.0038	0.0051	0.0022	0.0057	0.0102	0.0136	0.0054
BST_DT	0.1100	0.1523	0.0250	0.0572	0.0319	0.0470	0.0245	0.0283	0.0748	0.0689
BST_STMP	0.7759	0.6658	0.1707	0.1657	0.1806	0.5478	0.0000	0.0000	0.0321	0.3173
COV_TYPE	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	SAR	AVG_WT
ANN	0.0114	0.0102	0.0007	0.0382	0.0228	0.0086	0.0871	0.0966	0.0521	0.0410
KNN	0.1790	0.1665	0.5759	0.2518	0.2948	0.2016	0.4364	0.4266	0.3643	0.3621
SVM	0.0206	0.0161	0.0878	0.1039	0.1058	0.0507	0.0103	0.0134	0.0379	0.0558
DT	0.0606	0.0539	0.0124	0.2381	0.2423	0.0295	0.4083	0.3685	0.2000	0.2017
BAG_DT	0.0049	0.0056	0.0019	0.0092	0.0150	0.0064	0.0156	0.0222	0.0438	0.0156
BST_DT	0.5533	0.6134	0.1298	0.2784	0.2397	0.6438	0.0423	0.0727	0.2923	0.3582
BST_STMP	0.1702	0.1343	0.1914	0.0804	0.0797	0.0593	0.0000	0.0000	0.0095	0.0906

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.

7.4. 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 be optimized to that metric by this greedy, brute force search. A good ensemble usually will be found if some base-level models or combinations of them yield good performance on that metric. Although we do not know how to optimize the base-level models to many of these metrics, the ensemble can be optimized to them.

7.5. Beyond Binary Classification

Ensemble selection is straightforward for binary classification and regression. If the base-level models make predictions for multiple classes, no modification to the ensemble selection procedure is necessary for multi-class problems. If some base-level models make predictions one dichotomy at a time (e.g. SVMs), ensemble selection is easiest if the base-level models are combined so that they return a predicted probability for each class. We have not yet experimented with multi-class ensemble selection.

8. Conclusions

Ensemble selection uses forward stepwise selection from libraries of thousands of models to build ensembles that are optimized to the given performance metric. Using a variety of learning algorithms and parameter settings appears to be effective for generating libraries of diverse, high quality models. Ensemble selection's most important feature is that it can optimize ensemble performance to any easily computed performance metric. Experiments with seven test problems and ten performance metrics show that ensemble selection consistently finds ensembles that outperform all other models, including models trained with bagging, boosting, and Bayesian model averaging.

9. 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 (Provost & Domingos, 2003) 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 (Joachims, 1999) {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 (Platt, 1999).

Acknowledgments

Charles Chiu and Kohsuke Kawaguchi helped with the initial design of ensemble selection. We thank Tony Gualtieri for help with the HYPER_SPECT data, and our collaborators at SLAC for help with the SLAC data and SLQ performance metric.

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *First International Workshop on Multiple Classifier Systems*, 1–15.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*.

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

Gualtieri, A., Chettri, S. R., Cromp, R., & Johnson, L. (1999). Support vector machine classifiers as applied to aviris data. *Proc. Eighth JPL Airborne Geoscience Workshop*.

Joachims, T. (1999). Making large-scale svm learning practical. *Advances in Kernel Methods*.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*.

Meek, C., Thiesson, B., & Heckerman, D. (2002). *Staged mixture modeling and boosting* (Technical Report MSR-TR-2002-45).

Munro, P., & Parmanto, B. (1996). Competition among networks improves committee performance. *Advances in Neural Information Processing Systems*.

Opitz, D. (1999). Feature selection for ensembles. *AAAI/IAAI* (pp. 379–384).

Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. *Advances in Large Margin Classifiers* (pp. 61–74).

Provost, F., & Domingos, P. (2003). Tree induction for probability-based rankings. *Machine Learning*, *52*.

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

Sullivan, J., Langford, J., Caruana, R., & Blum, A. (2000). Featureboost: A meta-learning algorithm that improves model robustness. *Proceedings of the Seventeenth International Conference on Machine Learning*.

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