An Empirical Comparison of Supervised Learning Algorithms Using Different Performance Metrics

Rich Caruana and Alexandru Niculescu-Mizil

Computer Science, Cornell University, Ithaca NY 14850, USA

Abstract. We present the results of a large-scale empirical comparison between seven learning methods: SVMs, neural nets, decision trees, memory-based learning, bagged trees, boosted trees, and boosted stumps. A novel aspect of our study is that we compare these methods on nine different performance criteria: accuracy, squared error, cross entropy, ROC Area, F-score, precision/recall break-even point, average precision, lift, and probability calibration. The models with the best performance overall are neural nets, SVMs, and bagged trees. However, if we apply Platt calibration to boosted trees, they become the best model overall. Detailed examination of the results shows that even the best models perform poorly on some problems or metrics, and that even the worst models sometimes yield the best performance.

1 Introduction

This paper presents the results of an empirical comparison between seven supervised learning algorithms using nine performance criteria. The algorithms are SVMs, neural nets, decision trees, memory-based learning, bagged trees, boosted trees, and boosted stumps. For each algorithm we test many variants and parameter settings. For example, we compare ten styles of decision trees, neural nets of many sizes, SVMs using different kernels, etc. A total of 2000 models are tested on each problem.

The nine performance metrics are accuracy, F-score, Lift, ROC Area, average precision, precision/recall break-even point, squared error, cross-entropy, and a measure of probability calibration. We use this many performance metrics for several reasons:

1. different metrics are appropriate in different settings: in information retrieval precision and recall are more appropriate than accuracy; in marketing lift often is the appropriate metric; in medicine ROC often is the preferred metric.
2. it is not uncommon for a learning method that has optimal performance on one metric to be suboptimal on other metrics.
3. some learning methods are designed for certain criteria. For example, SVMs and boosting are designed to optimize accuracy, while neural nets typically optimize squared error or cross entropy.

We experiment with seven binary classification problems, yielding $9 * 7 = 63$ comparisons between learning algorithms. The results are somewhat surprising. To preview: if selection is done carefully, neural nets give the best performance if one considers all nine metrics across the test problems, but boosted decision trees are best if we restrict attention to the six metrics that do not depend on interpreting predictions as probabilities. SVMs perform nearly as well as neural nets and boosted trees. Bagged trees have good overall performance while being very easy to train. Boosting full decision trees dramatically outperforms boosting weaker stumps on most problems. If predictions generated by boosting trees are calibrated with Platt's method for calibrating SVM predictions, the boosted trees become the best performing model. Memory-based learning performs better than single decision trees and boosted stumps, but often is not competitive with the top performing methods.

Examining the results more carefully, however, reveals that these generalizations do not always hold. For example, boosted stumps, which perform poorly on average, are the best model type for 6 of 9 metrics on the adult data set. Five of the seven algorithms yield models that perform best for at least one metric and test problem.

2 Methodology

2.1 Learning Algorithms

This section summarizes the parameters used with each learning algorithm.
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 neural 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.
Decision trees (DT): we vary the splitting criterion, pruning options, and smoothing (Laplacian or Bayesian smoothing). We use all of the tree models in Buntine's IND package: BAYES, ID3, CART, CART0, C4, MML, and SMML. We also generate trees of type C44LS (C4 with no pruning and Laplacian smoothing), C44BS (C44 with Bayesian smoothing), and MMLLS (MML with Laplacian smoothing). See [1] for a description of C44LS.
Bagged trees (BAG-DT): we bag 25-100 trees of each type. We use each of the bootstrapped trees generated by bagging, and the final bagged ensemble. With **boosted trees (BST-DT)** we boost each tree type. Boosting can overfit, so we consider boosted DTs after 2,4,8,16,32,64,128,256,512,1024 and 2048 steps of boosting. With **boosted stumps (BST-STMP)** we use stumps (single level decision trees) generated with 5 different splitting criteria, each boosted for 2,4,8,16,32,64,128,256, 512,1024,2048,4096,8192 steps.
SVMs: we use the following kernels in SVMLight[2]: 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 by factors of ten from 10^{-7} to 10^3. The output range of SVMs is $[-\infty, +\infty]$ instead of $[0, 1]$. To make SVM predictions compatible with metrics such as squared error, cross-entropy or calibration, we apply Platt's method to convert SVM outputs to probabilities by fitting them to a sigmoid [3].

With ANN's, SVM's and KNN's we scale attributes to 0 mean 1 std. With DT, BAG-DT, BST-DT and BST-STMP we don't scale the data. In total, we train about 2000 different models.

Table 1. Accuracy on ADULT problem

MODEL	ACC	NORM SCORE
BST-STMP	0.8556	1.0000
BAG-DT	0.8534	0.9795
DT	0.8503	0.9494
SVM	0.8480	0.9267
BST-DT	0.8464	0.9113
ANN	0.8449	0.8974
KNN	0.8320	0.7731
BASELINE	0.7518	0.0000

2.2 Performance Metrics

We divide the performance metrics into three groups: threshold metrics, ordering/rank metrics and probability metrics.

The threshold metrics are accuracy (ACC), F-score (FSC) and lift (LFT). See [4] for a description of Lift Curves. Usually ACC and FSC have a fixed threshold (in this paper 0.5). For lift, often a fixed percent, p, of cases are predicted as positive and the rest as negative(in this paper $p = 25\%$). Note that for thresholded metrics, it is not important how close a prediction is to a threshold, only if it is above or below threshold.

The ordering/rank metrics look at predictions differently. If cases are ordered by their predicted values, these metrics measure how well the positive cases are ordered before negative cases. The rank metrics can be viewed as a summary of the model performance across all possible thresholds. The rank metrics we use are area under the ROC curve (ROC), average precision (APR), and precision/recall break even point (BEP). See [5] for a discussion of ROC from a machine learning perspective. Rank metrics depend only on the ordering of the predictions, not the actual predicted values. If the ordering is preserved it makes no difference if the predicted values are between 0 and 1 or between 0.49 and 0.51.

The probability metrics are performance metrics that are uniquely minimized (in expectation) when the predicted value for each case coincides with the true underlying probability of that case being positive. The probability metrics are squared error (RMS), cross-entropy (MXE) and calibration (CAL). CAL measures the calibration of a model: if the true probability that a case is positive is 85% a well calibrated model should predict .85 for that case. CAL is calculated as follows: Order all cases by their predictions and put cases 1-100 in the same bin. Calculate the percentage of these cases that are true positives. This approximates the true probability that these cases are positive. Then calculate the mean prediction for these cases. The absolute value of the difference between the observed frequency and the mean prediction is the calibration error for these 100 cases. Now take cases 2-101, 3-102, ... and compute the errors in the same way. CAL is the mean of all these binned calibration errors.

2.3 Comparing Across Performance Metrics

Some performance metrics such as accuracy or squared error have range $[0, 1]$, while others (lift, cross entropy) range from 0 to p where p depends on the data set. For some metrics lower values indicate better performance. For others higher values are better. Metrics such as ROC area have baseline rates that are independent of the data, while others such as accuracy have baseline rates that depend on the data. If baseline accuracy is 0.98, an accuracy of 0.981 probably is not good performance, but on another problem the Bayes optimal rate might be 0.60 and achieving an accuracy of 0.59 might be excellent performance.

To permit averaging across metrics or problems, each performance needs to be placed on the same scale. One way to do this is to scale the performance for each problem and metric from 0 to 1, where 0 is baseline performance and 1 is Bayes optimal. Unfortunately, we cannot estimate the Bayes optimal rate on real problems, so we use the performance of the best observed model as a proxy. We use the following baseline model: predict p for every case, where p is the percent of positives in the test set. We normalize performances to the range $[0, 1]$, where 0 is baseline and 1 represents best performance. If a model performs worse than baseline, its normalized score will be negative. See Table 1 for an example of normalized scores. CAL, the metric used to measure probability calibration, is unusual in that the baseline model has excellent calibration.[1] This creates a problem when normalizing CAL scores because the baseline model and Bayes optimal model have similar CAL scores. Unlike the other measures, CAL is scaled so that the minimum observed CAL score is 0.0 and the mean observed CAL score is 1.0. The disadvantage of normalized scores is that recovering the raw performance requires knowing the performances that define the top and bottom of the scale, and as new best models are found the top of the scales changes. We will make the performances that define the top and bottom of our scales for each problem and each metric (126 numbers) available on the web so that others can compare to our normalized scores.

2.4 Data Sets

We compare the algorithms on 7 binary classification problems. ADULT, COV_TYPE and LETTER are from UCI Machine Learning Repository [6]. 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. COV_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. HS is the IndianPine92 data set [7] 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 2.

[1] Because of this, measures like CAL typically are not used alone, but are used in conjunction with other measures such as ROC to insure that only models with good discrimination *and* good calibration are selected. This does not mean CAL is a poor metric – it is effective at distinguishing poorly calibrated models from well calibrated models.

Table 2. Description of problems

PROBLEM	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
ADULT	14/104	4000	35222	25%
COV_TYPE	54	4000	25000	36%
LETTER.P1	16	4000	14000	3%
LETTER.P2	16	4000	14000	53%
MEDIS	63	4000	8199	11%
SLAC	59	4000	25000	50%
HS	200	4000	4366	24%

2.5 Experimental Design

The goal of our work is to compare the best performance that can be achieved with each learning algorithm. 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 set (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. Another benefit of selecting the model parameters using the large final test set is that it eliminates the complexity and variance arising from model selection.

For each algorithm we use all the parameters described in Section 2.1. We train all models on identical train sets of 4000 points. The results we present average over multiple problems and metrics, but the distributions over problems and metrics are not homogeneous so we do not know how to apply statistical tests to determine the significance of the differences we observe across problems and metrics. Multiple trials would allow us to compare the performance of learning algorithms on single problems and metrics, but would not help comparisons across *different* problems and metrics. Instead of running multiple trials on a few problems, we experiment with more problems and average over the problems to obtain a more complete view of the performance of the algorithms across multiple problems. The test sets are large enough to make discerning small differences in performance reliable.

3 Performances by Metric

Tables 3 and 4 show the normalized performance for each algorithm on nine metrics. For each test problem we find the best parameter settings for each algorithm and compute it's normalized score. Each entry in the table averages these scores across the seven problems. The last two columns are the mean normalized scores over the nine metrics (in boldface), and the SAR performance (SAR averages RMS, ACC, and ROC; see Section 3.1). Higher scores indicate better performance. The models in the table are ordered by mean overall performance.

Table 4 shows the 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

Table 3. Normalized scores for each learning algorithm by metric when model selection is done using the 1k validation set (average over seven problems)

MODEL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	CAL	MEAN	SAR
SVM	0.835	**0.914**	0.950	0.962	0.935	0.941	**0.881**	0.882	0.912	**0.912**	**0.909**
ANN	0.885	0.880	0.951	0.957	0.920	0.925	0.865	0.874	**0.928**	**0.909**	0.900
BAG-DT	0.854	0.867	0.951	0.969	0.945	0.925	0.861	**0.895**	0.808	**0.897**	0.905
KNN	0.776	0.855	0.911	0.938	0.888	0.894	0.778	0.754	0.883	**0.853**	0.855
BST-DT	**0.893**	0.905	**0.961**	**0.978**	**0.961**	**0.950**	0.615	0.627	0.437	**0.814**	0.872
DT	0.530	0.796	0.859	0.875	0.778	0.809	0.586	0.636	0.762	**0.737**	0.755
BST-STMP	0.742	0.793	0.908	0.919	0.863	0.836	0.304	0.287	0.415	**0.674**	0.661

Table 4. Normalized scores for each learning algorithm by metric when model selection is done using the big final test set (average over seven problems)

MODEL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	CAL	MEAN	SAR
ANN	0.924	**0.922**	0.965	0.967	0.945	0.947	**0.891**	0.892	**0.975**	**0.936**	0.911
SVM	0.885	**0.922**	0.955	0.963	0.943	0.949	0.890	0.895	0.931	**0.926**	**0.912**
BAG-DT	0.864	0.873	0.962	0.971	0.949	0.930	0.865	**0.900**	0.825	**0.904**	0.907
KNN	0.798	0.877	0.932	0.943	0.897	0.911	0.785	0.768	0.892	**0.867**	0.863
BST-DT	**0.934**	0.919	**0.977**	**0.981**	**0.969**	**0.969**	0.630	0.636	0.523	**0.838**	0.879
DT	0.661	0.836	0.884	0.884	0.809	0.827	0.621	0.668	0.783	**0.775**	0.781
BST-STMP	0.780	0.805	0.917	0.924	0.872	0.858	0.315	0.298	0.427	**0.688**	0.666

could be achieved with each learning method if model selection was done perfectly. Table 3 shows the 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 performance that might be obtained using a simple approach to model selection, but does not show the performance that might be obtained if model selection was done more carefully.

As expected, the high variance models have the biggest drop in performance when selection is done using the 1k validation sets. Single decision trees are the most affected, having a drop of 0.038. ANN follows, with a drop of 0.024. The most notable difference between the two tables is that when selection is done using the final test set, ANN's have better overall performance than SVMs. However, SVMs are a more stable method, they have a smaller drop in performance when selection is done using the small validation set and move ahead of ANN's in Table 3. Overall however, the changes in the performance are small and the results are qualitatively similar. Moreover, if a more sophisticated selection procedure (e.g. cross validation) would be used, the differences between the two tables would be even smaller. Because of this, in the rest of the paper, we only present and discuss the results obtained when selection is done on the large final test set.

In addition to normalized scores, we also rank the algorithms by their performance on each metric and problem (when selection is done using the large final test sets). Table 5 shows the average ranks of the algorithms. Lower ranks indicate better performance. An average rank of one would indicate that that algorithm always yielded best performance on that metric for all seven test problems. The column

Table 5. Avg. ranks for each learning algorithm by metric (average over 7 problems)

MODEL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	CAL	MEAN	SAR
ANN	3.3	**3.1**	3.4	3.3	3.7	3.0	**2.3**	2.4	**1.9**	**2.93**	2.4
SVM	3.1	**3.1**	3.9	3.1	3.0	2.9	2.4	**2.0**	2.9	**2.93**	**2.1**
BAG-DT	3.6	4.3	2.7	3.4	3.3	3.7	2.4	2.3	3.3	3.22	2.7
BST-DT	**2.3**	3.3	**2.1**	**2.1**	**2.1**	**2.3**	5.1	5.1	6.3	3.41	4.0
KNN	5.0	4.0	4.9	4.6	4.9	5.0	3.7	3.9	3.1	4.34	4.1
BST-STMP	4.9	5.3	4.7	4.9	5.0	4.9	6.9	6.9	5.9	5.49	6.9
DT	5.9	4.9	6.3	6.6	6.0	6.3	5.1	5.4	4.7	5.69	5.7

labeled "MEAN" is the average rank over all nine performance metrics. The models are sorted by mean rank.

Overall, the best performing models are neural nets, SVMs, and bagged trees. Surprisingly, neural nets outperform all other model types if one averages over the nine metrics. ANNs appear to be excellent general purpose learning methods. This is not to say that ANNs are the best learning algorithm, but because they rarely perform poorly on any problem or metric, they have excellent overall performance.

The SVMs perform almost as well as ANNs. Note that SVM predictions on $[-\infty, +\infty]$ are not suitable for measures like cross-entropy, calibration, and squared error. SVMs do well on these metrics because we use Platt's method to transform SVM predictions to calibrated probabilities. (See Section 6.) None of the test problems used in this paper are very high dimension, a regime in which SVMs are known to perform well. Nevertheless, SVMs are still one of the top performers. In Table 5 SVMs tie for the the lowest rank among all the model types we considered.

Interestingly, although single decision trees yield very poor performance, bagged trees yield performance nearly as good as the neural nets and SVMs. Bagging improves decision tree performance on all metrics, and yields particularly large improvements on the probability metrics. Like neural nets, bagged trees appear to be a safe, general purpose, high performance learning method. Given how easy it is to train bagged trees, they may be the method of choice in many problems.

Boosted trees outperform all other learning methods on ACC, LFT, ROC, APR, and BEP. Boosting wins 2 of 3 threshold metrics and 3 of 3 rank metrics, but perform poorly on the probability metrics: squared error, cross-entropy, and calibration. We suspect maximum margin methods such as boosted trees yield poorly calibrated probabilities. (SVMs perform well on these because of Platt scaling. Platt scaling also helps improve the performance of boosted trees on the probability metrics. See Section 6.) Overall, boosting wins 5 of the 6 metrics for which it is well suited, and would easily be the top performing learning method if we consider only the 6 threshold and ordering metrics.

MBL methods such as k-nearest neighbor perform well if attributes in the distance function are scaled. Scaling attributes by their gain ratio typically yielded the best performance. But the MBL/KNN methods were not competitive with the better algorithms. The average rank for KNN is 4.39, a full rank below the next better model which has average rank 3.37.

Single decision trees did not perform as well as most other methods. We suspect this is partly because recursive partitioning quickly runs out of data with 4k train

Table 6. Norm. scores of each learning algorithm by problem (averaged over 9 metrics)

MODEL	COV_TYPE	ADULT	LETTER.P1	LETTER.P2	MEDIS	SLAC	HS	MEAN
ANN	0.8294	0.9425	0.9555	0.9327	**0.9885**	0.9431	**0.9625**	**0.9363**
SVM	0.8433	0.9235	**0.9643**	**0.9846**	0.8838	0.9274	0.9552	**0.9260**
BAG-DT	**0.9314**	**0.9836**	0.8969	0.8642	0.8486	**0.9515**	0.8541	**0.9043**
KNN	0.8844	0.8504	0.9284	0.9478	0.7472	0.8717	0.8380	**0.8668**
BST-DT	0.9069	0.8557	0.8903	0.8757	0.5752	0.8943	0.8652	**0.8376**
DT	0.7429	0.9342	0.7853	0.7719	0.6287	0.8594	0.7007	**0.7747**
BST-STMP	0.7022	0.8729	0.3782	0.6239	0.7421	0.8078	0.6907	**0.6883**

Table 7. Norm. scores of each learning algorithm by problem (averaged over 6 metrics)

MODEL	COV_TYPE	ADULT	LETTER.P1	LETTER.P2	MEDIS	SLAC	HS	MEAN
BST-DT	**0.9949**	0.9381	**0.9857**	**0.9980**	0.8867	0.9290	0.9745	0.9581
ANN	0.8647	0.9435	0.9625	0.9530	**0.9827**	0.9316	**0.9748**	0.9447
SVM	0.8694	0.9453	0.9648	0.9903	0.8806	**0.9417**	0.9619	0.9363
BAG-DT	0.9397	0.9835	0.9212	0.9474	0.8392	0.9365	0.9066	0.9249
KNN	0.9223	0.8741	0.9574	0.9747	0.7663	0.8837	0.8715	0.8929
BST-STMP	0.7526	**0.9918**	0.8137	0.7706	0.9422	0.8582	0.8847	0.8591
DT	0.7973	0.9450	0.8406	0.8531	0.6365	0.8799	0.7656	0.8169

sets, and partly because small trees are not good at predicting probabilities [1]. In our experiments we tested many different kinds of decision trees, including smoothed unpruned trees, and then picked the best, so the poor performance of trees is not due to any one tree type being inferior, but because all of the many tree types we tested did not perform as well as other methods.

Surprisingly, boosting weaker stump models does not perform as well as boosting full decision trees. Boosted stumps do outperform the single trees on 5 of the 6 threshold and rank metrics. Their last-place ranking below decision trees is due to their extremely poor performance on the three probability measures.

3.1 The SAR Metric

We categorize the performance metrics into three groups: threshold, rank, and probability metrics. SAR combines a metric from each of these categories into one metric. SAR stands for "Squared error", "Accuracy", and "ROC area" and is defined as: $((1-RMS)+ACC+ROC)/3$. We choose accuracy for SAR because it is the most commonly used threshold metric, ROC Area because it is the most principled of the three ordering metrics, and RMS because it is better behaved than the other two probability metrics. We believe good SAR performance indicates that a model is robust and less likely to overfit the train set. Table 4 shows that SVMs, ANNs, and bagged trees have SAR performance significantly better than the four remaining models. Note that SAR requires *one* model have good performance on all three metrics, not that different parameter settings for an algorithm yields good models, one for each of the three SAR metrics.

4 Performances by Problem

Table 6 shows the average normalized score for each algorithm on each of the seven problems. Each entry is an average over the nine performance metrics. There is no universally best learning algorithm. Each model type performs well on some problems and is mediocre on others. Neural nets have good performance on most problems, but on COV_TYPE have the third worst normalized score. At the other end of the spectrum, vanilla decision trees perform better than boosted decision trees on ADULT, even though, on average, boosting the decision trees significantly improves performance.

In Table 7 we present the same results as Table 6, but each entry in the table averages only over ACC, FSC, LFT, ROC, APR and BEP, ignoring the three probability metrics. Boosted trees have trouble with probability metrics, so it is no surprise that their performance improves dramatically. Boosted trees now win on three problems. Neural Nets win on two problems and SVMs on one. Even boosted stumps, which have poor average performance across all problems, win on all these six metrics on ADULT. Boosted stumps also outperform boosted trees on MEDIS on these six metrics.

Although some methods clearly outperform others *on average*, to get the best performance on any one problem, one should if possible try as many model types as possible to find the model type with best performance. As an example, for the SLAC problem, there is a custom designed performance metric called SLQ from which one can estimate the amount of accelerator time needed for an experiment. Increasing SLQ by 0.01 can save SLAC a hundred thousand dollars or more in accelerator time per month. The difference between the best model (a bagged MML decision tree) and the second best model (a boosted Bayesian tree) is 0.005 on a large test set, which translates to a potential savings of more than 50,000 dollars per month.

5 Performances by Algorithm

Comparing learning methods on 7 test problems and 10 performance metrics (including the SAR metric) yields 70 comparisons. Interestingly, KNN and single decision trees are never best on any of these 70 comparisons. On 51 of 70 comparisons, 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 good performance. Although single ID3 trees perform poorly, bagged ID3 trees perform well.

On most problems and metrics, SVMs with RBF kernels perform better than SVMs with other kernels. The width of the RBF kernel that gives best performance changes with both the problem and the metric. With KNN, rank metrics such as ROC Area, and the probability metrics such as RMS, favor much larger values of K than threshold metrics such as accuracy. The best neural nets typically have large hidden layers. Nets with 32 or 128 hidden units performed best 55 out of 70 times.

Table 8. Normalized scores for boosted decision trees before and after Platt conversion

MODEL	RMS	MXE	CAL
CONV_BST-DT	0.9481	0.9501	0.9592
BST-DT	0.6305	0.6357	0.5235

Boosting full decision trees yields better performance than boosting stumps on five of the problems. Occasionally boosted stumps perform very well, but when they do not, they often perform very poorly, giving them poor average performance. On two problems, the first iteration of boosting hurts the performance of *all* tree types, then never recovers. On these problems single decision trees outperform their boosted counterparts. Bagged trees, however, consistently outperform unbagged trees on all seven problems. Bagging is "safer" than boosting, even on the six metrics for which boosting yields the best overall performance.

6 Platt Probability Calibration

For RMS, MXE and CAL, SVM outputs must be converted from $[-\infty, +\infty]$ to $[0, 1]$. We do this by fitting SVM predictions to a sigmoid with Platt's method [3]. This scales the predictions without reordering them. Fitting the sigmoid on the same data the SVM was trained on introduces bias that hurts the quality of the conversion. Following Platt we use 3-fold cross-validation on the train set to train 3 SVMs. Each SVM is trained on two folds, and the predictions from the 3 test folds are used to train the sigmoid. We then retrain the SVM on the full 4k train set and use the sigmoid to scale the outputs. This transformation is not needed for the threshold or ordering metrics and is used only for the three probability metrics. With this conversion SVMs are the second best algorithm on RMS and cross-entropy, suggesting that Platt conversion is effective.

Tables 4 and 5 show that boosted trees have excellent performance on the threshold and rank metrics, but perform poorly on the probability metrics. This lead us to try Platt's conversion on boosted trees. Table 8 presents normalized scores on the probability metrics for boosted trees before and after conversion. Platt scaling significantly improves the quality of the probabilities predicted by boosted trees. After conversion, boosted trees become the best overall models with a mean normalized score of 0.956 over the 7 problems and 9 metrics.

We also applied Platt's conversion to the other algorithms. As a cheaper alternative to cross validation, we used an additional 1k data set for conversion. This significantly improved the performance of the boosted stumps on the three probability metrics, but hurt the performance of the neural nets and bagged trees, which already had excellent performance on these metrics before conversion. (With SVMs and BST-DT, conversion using an additional 1k test set gave similar performance as conversion using 3-fold cross validation.) For the main results of this paper we use cross-validated Platt conversion only with SVMs because only the SVMs require this conversion, and because we have not yet done a thorough study of the effect of Platt conversion on the other model types.

7 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, 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. Platt scaling increases the cost of training SVMs by a factor of 2.2 [3], making SVMs more expensive than neural nets when probabilities are needed.

Although we experimented with a variety of MBL methods, distance measures, and control parameters, MBL proved less expensive than neural nets and SVMs because our training sets contain only 4k points.

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 all around good performance. Simple unbagged/unboosted trees are the least expensive method, even if many tree types are tried, but their performance usually is not that good.

8 Related Work

There are few comprehensive empirical studies comparing learning algorithms. STATLOG is perhaps the best known study [8]. 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. LeCun et al. [9] 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. [10] present results from a study that evaluates nearly a dozen learning methods on a real medical data set using both accuracy and an ROC-like metric. Lim et al. [11] perform an empirical comparison of decision trees and other classification methods using accuracy as the main criterion. Bauer and Kohavi [12] present an impressive empirical analysis of ensemble methods such as bagging and boosting. Provost and Domingos [1] examine the issue of predicting probabilities with decision trees, including smoothed and bagged trees. Provost and Fawcett [5] discusses the importance of evaluating learning algorithms on metrics other than accuracy such as ROC.

9 Conclusions

We conducted an empirical comparison of seven learning algorithms on nine performance criteria. Methods that performed well on the threshold metrics also performed well on the rank metrics. Performance on the probability metrics, however, did not mirror performance on other metrics. For example, boosted trees were best on the threshold and rank metrics, but performed poorly on the probability metrics. Platt's calibration procedure, however, boosted the performance of boosted trees on the probability metrics significantly, making them the best performing models overall. Other excellent models included neural nets, bagged decision trees, and SVMs. (SVM predictions *must* be calibrated with Platt's method for the probability metrics.) The models that performed poorest were k-nearest neighbor, boosted stumps, and decision trees. It is interesting that boosting weak models such as stumps did not perform as well as boosting more powerful decision trees. It is also interesting that for most of these models, model selection using small 1k validation sets yields performance nearly as good as optimal selection using the large final test sets.

Although some of the methods clearly perform significantly better or worse than other methods *on average*, there is variability across problems and metrics. Even the best models sometimes perform poorly, and the models with the worst average performance occasionally perform best. Despite the fact that a few methods perform extremely well on average, it is still important to try as many methods as possible if one must achieve the very best performance on any specific problem and metric.

References

1. Provost, F., Domingos, P.: Tree induction for probability-based rankings. Machine Learning **52** (2003)
2. Joachims, T.: Making large-scale svm learning practical. In: Advances in Kernel Methods. (1999)
3. Platt, J.: Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In Smola, A., Bartlett, P., Schoelkopf, B., Schuurmans, D., eds.: Advances in Large Margin Classifiers. (1999) 61–74
4. Giudici, P.: Applied Data Mining. John Wiley and Sons, New York (2003)
5. Provost, F.J., Fawcett, T.: Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In: Knowledge Discovery and Data Mining. (1997) 43–48
6. Blake, C., Merz, C.: UCI repository of machine learning databases (1998)
7. Gualtieri, A., Chettri, S.R., Cromp, R., Johnson, L.: Support vector machine classifiers as applied to aviris data. In: Proc. Eighth JPL Airborne Geoscience Workshop. (1999)
8. King, R., Feng, C., Shutherland, A.: Statlog: comparison of classification algorithms on large real-world problems. Applied Artificial Intelligence **9** (1995) 259–287
9. LeCun, Y., Jackel, L.D., Bottou, L., Brunot, A., Cortes, C., Denker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Simard, P., Vapnik, V.: Comparison of learning algorithms for handwritten digit recognition. In Fogelman, F., Gallinari, P., eds.: International Conference on Artificial Neural Networks, Paris, EC2 & Cie (1995)
10. Cooper, G.F., Aliferis, C.F., Ambrosino, R., Aronis, J., Buchanan, B.G., Caruana, R., Fine, M.J., Glymour, C., Gordon, G., Hanusa, B.H., Janosky, J.E., Meek, C., Mitchell, T., Richardson, T., Spirtes, P.: An evaluation of machine learning methods for predicting pneumonia mortality. Artificial Intelligence in Medicine **9** (1997)

11. Lim, T.S., Loh, W.Y., Shih, Y.S.: An empirical comparison of decision trees and other classification methods. Technical Report 979, Madison, WI (1997)
12. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning **36** (1999)
13. Vapnik, V.: Statistical Learning Theory. John Wiley and Sons, New York (1998)
14. Schapire, R.: The boosting approach to machine learning: An overview. In: In MSRI Workshop on Nonlinear Estimation and Classification. (2001)
15. Breiman, L.: Bagging predictors. Machine Learning **24** (1996) 123–140