An Empirical Evaluation of Supervised Learning in High Dimensions

Rich Caruana CARUANA@CS.CORNELL.EDU
Nikos Karampatziakis NK@CS.CORNELL.EDU
Ainur Yessenalina AINUR@CS.CORNELL.EDU
Department of Computer Science, Cornell University, Ithaca, NY 14853 USA

Abstract

In this paper we perform an empirical evaluation of supervised learning on high-dimensional data. We evaluate performance on three metrics: accuracy, AUC, and squared loss and study the effect of increasing dimensionality on the performance of the learning algorithms. Our findings are consistent with previous studies for problems of relatively low dimension, but suggest that as dimensionality increases the relative performance of the learning algorithms changes. To our surprise, the method that performs consistently well across all dimensions is random forests, followed by neural nets, boosted trees, and SVMs.

1. Introduction

In the last decade, the dimensionality of many machine learning problems has increased substantially. Much of this results from increased interest in learning from text and images. Some of the increase in dimensionality, however, results from the development of techniques such as SVMs and L_1 regularization that are practical and effective in high dimensions. These advances may make it unnecessary to restrict the feature set and thus promote building and learning from data sets that include as many features as possible. At the same time, memory and computational power have increased to support computing with large data sets.

Perhaps the best known empirical studies to examine the performance of different learning methods are STATLOG (King et al., 1995) and (Caruana & Niculescu-Mizil, 2006). STATLOG was a very thorough study, but did not include test problems with high dimensions and could not evaluate newer learning methods such as bagging, boosting, and kernel methods. More recently, (Caruana & Niculescu-Mizil, 2006) includes a number of new learning algorithms that emerged after the STATLOG project, but only examined performance on problems with low-to-medium dimension. One must question if the results of either of these studies apply to text data, biomedical data, link analysis data etc. where many attributes are highly correlated and there may be insufficient data to learn complex interactions among attributes. This paper attempts to address that question.

There are several limitations to the empirical study in (Caruana & Niculescu-Mizil, 2006). First, they performed all experiments using only 5000 training cases, despite the fact that much more labeled data was available for many problems. For one of the problems (COVTYPE) more than 500,000 labeled cases were available. Intentionally training using far less data than is naturally available on each problem makes the results somewhat contrived. Second, although they evaluated learning performance on eight performance metrics, examination of their results shows that there are strong correlations among the performance measures and examining this many metrics probably added little to the empirical comparison and may have led to a false impression of statistical confidence. Third, and perhaps most important, all of the data sets examined had low to medium dimensionality. The average dimensionality of the 11 data sets in their study was about 50 and the maximum dimensionality was only 200. Many modern learning problems have orders of magnitude higher dimensionality. Clearly learning methods can behave very differently when learning from high-dimensional data than when learning from low-dimensional data.

In the empirical study performed for this paper we complement the prior work by: 1) using the natural size training data that is available for each problem; 2) using just three important performance metrics: ac-

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

curacy, area under the ROC curve (AUC), and squared loss; and 3) performing experiments on data with high to very high dimensionality (750-700K dimensions).

2. Methodology

2.1. Learning Algorithms

This section summarizes the algorithms and parameter settings we used. The reader should bear in mind that some learning algorithms are more efficient at handling large training sets and high-dimensional data than others. For an efficient algorithm we can afford to explore the parameter space more exhaustively than for an algorithm that does not scale well. But that's not unrealistic; a practitioner may prefer an efficient algorithm that is regarded as weak but which can be tuned well over an algorithm that might be better but where careful tuning would be intractable. Below we describe the implementations and parameter settings we used. An algorithm marked with an asterisk (e.g. **ALG***) denotes our own custom implementation designed to handle high-dimensional sparse data.

SVMs: We train linear SVMs using SVM^{perf} (Joachims, 2006) with error rate as the loss function. We vary the value of C by factors of ten from 10^{-9} to 10^5. For kernel SVMs we used LaSVM, an approximate SVM solver that uses stochastic gradient descent (Bordes et al., 2005), since traditional kernel SVM implementations simply cannot handle the amounts of data in some of our experiments. To guarantee a reasonable running time we train the SVM for 30 minutes for each parameter setting and use the gradient based strategy for the selection of examples. We use polynomial kernels of degree 2 and 3 and RBF kernels with width $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2\}$. We vary the value of C by factors of ten from 10^{-7} to 10^5.

ANN***:** We train neural nets with gradient descent backpropagation, early stopping and no momentum (cf. section 5). We vary the number of hidden units $\{8, 16, 32\}$ and learning rate $\{10^{-4}, 10^{-3}, 10^{-2}\}$.

Logistic Regression (LR): We use the BBR package (Genkin et al., 2006) to train models with either L_1 or L_2 regularization. The regularization parameter is varied by factors of ten from 10^{-7} to 10^5.

Naive Bayes (NB***):** Continuous attributes are modeled as coming from a normal distribution. We use smoothing and vary the number of unobserved values {0,0.1,0.2,0.5,1,2,5,10,20,50,100}.

KNN***:** We use distance weighted KNN. We use the 1000 nearest neighbors weighted by a Gaussian kernel with 40 different kernel widths in the range $[0.1, 820]$. The distance between two points is a weighted euclidean distance where the weight of each feature is determined by its information gain.

Random Forests (RF***):** We grow 500 trees and the size of the feature set considered at each split is $s/2, s, 2s, 4s$ or $8s$ where s is the square root of the number of attributes, as suggested in (Breiman, 2001).

Bagged Decision Trees(BAGDT***):** We bag 100 ID3 trees. The same implementation is used for **boosted stumps (BSTST*****)** and **boosted trees (BSTDT*****)** but because full-size ID3 trees can't easily be boosted, we stop splitting when a node contains less than 50 examples. We do 2^{10} and 2^{15} iterations of boosting for trees and stumps respectively and use the validation set to pick the number of iterations from the set $\{2^i | i = 0, 1, ...15\}$.

Perceptrons (PRC***):** We train voted perceptrons (Freund & Schapire, 1999) with 1,5,10,20 and 30 passes through the data. We also average 100 perceptrons each of which is obtained by a single pass through a permutation of the data.

With SVM, ANN, LR, KNN and PRC and datasets with continuous attributes we train both on raw and standardized data. This preprocessing can be thought of as one more parameter for these algorithms. To preserve sparsity, which is crucial for the implementations we use, we treat the mean of each feature as zero, compute the standard deviation, and divide by it.

2.2. Performance Metrics

To evaluate performance we use three metrics: accuracy (ACC), a threshold metric, squared error (RMS), a probability metric, and area under the ROC curve (AUC), an ordering metric. To standardize scores across different problems and metrics, we divide performances by the median performance observed on each problem for that metric. For squared error we also invert the scale so that larger numbers indicate better performance as with accuracy and AUC.

2.3. Calibration

The output of some learning algorithms such as ANN, logistic regression, bagged trees and random forests can be interpreted as the conditional probability of the class given the input. The common implementation of other methods such as SVM and boosting, however, are not designed to predict probabilities (Niculescu-Mizil & Caruana, 2005).

To overcome this, we use two different methods to map the predictions from each learning algorithm to calibrated probabilities. The first is Isotonic Regression (Zadrozny & Elkan, 2002), a method which fits a non-parametric non-decreasing function to the predictions. The second calibration method is Platt's method (Platt, 1999) which fits a sigmoid to the predictions. (Niculescu-Mizil & Caruana, 2005) suggests that Platt's method outperforms isotonic regression when there is less than about 1000 points available to learn the calibration function, and that calibration can hurt predictions from methods such as ANN and logistic regression (Caruana & Niculescu-Mizil, 2006). We will revisit those findings later in the discussion. Finally, note that calibrating can affect metrics other than probability metrics such as squared loss. It can affect accuracy by changing the optimal threshold (for calibrated predictions the optimum threshold will be near 0.5) and Isotonic Regression can affect AUC by creating ties on calibration plateaus where prior to calibration there was a definite ordering.

To summarize our methodology, we optimize for each dataset and metric individually. For each algorithm and parameter setting we calibrate the predictions using isotonic regression, Platt's method, and the identity function (no calibration) and choose the parameter settings and calibration method that optimizes the performance metric on a validation set.

2.4. Data Sets

We compare the methods on 11 binary classification problems whose dimensionality ranges from 761 to 685569. The datasets are summarized in Table 1.

Table 1. Description of problems

Problem	Attr	Train	Valid	Test	%Pos
Sturn	761	10K	2K	9K	33.65
Calam	761	10K	2K	9K	34.32
Digits	780	48K	12K	10K	49.01
Tis	927	5.2K	1.3K	6.9K	25.13
Cryst	1344	2.2K	1.1K	2.2K	45.61
KDD98	3848	76.3K	19K	96.3K	5.02
R-S	20958	35K	7K	30.3K	30.82
Cite	105354	81.5K	18.4K	81.5K	0.17
Dse	195203	120K	43.2K	107K	5.46
Spam	405333	36K	9K	42.7K	44.84
Imdb	685569	84K	18.4K	84K	0.44

TIS[1] is from the Kent Ridge Bio-medical Data Repository. The problem is to find Translation Initiation Sites (TIS) at which translation from mRNA to proteins initiates. CRYST[2] is a protein crystallography diffraction pattern analysis dataset from the X6A beamline at Brookhaven National Laboratory. STURN and CALAM are ornithology datasets.[3] The task is to predict the appearance of two bird species: sturnella neglecta and calamospiza melanocorys. KDD98 is from the 1998 KDD-Cup. The task is to predict if a person donates money. This is the only dataset with missing values. We impute the mean for continuous features and treat missing nominal and boolean features as new values. DIGITS[4] is the MNIST database of handwritten digits by Cortes and LeCun. It was converted from a 10 class problem to a hard binary problem by treating digits less than 5 as one class and the rest as the other class. IMDB and CITE are link prediction datasets.[5] For IMDB each attribute represents an actor, director, etc. For CITE attributes are the authors of a paper in the CiteSeer digital library. For IMDB the task is to predict if Mel Blanc was involved in the film or television program and for CITE the task is to predict if J. Lee was a coauthor of the paper. We created SPAM from the TREC 2005 Spam Public Corpora. Features take binary values showing if a word appears in the document or not. Words that appear less than three times in the whole corpus were removed. Real-Sim (R-S) is a compilation of Usenet articles from four discussion groups: simulated auto racing, simulated aviation, real autos and real aviation.[6] The task is to distinguish real from simulated. DSE[7] is newswire text with annotated opinion expressions. The task is to find Subjective Expressions i.e. if a particular word expresses an opinion.

To split data into training, validation and test sets, if the data came with original splits for train and test sets (i.e. DIGITS, KDD98) we preserved those splits and created validation sets as 10% of the train set. If the data originally was split into folds, we merged some folds to create a training set, a validation set and a test set. (We did this because running these experiments is so costly that we could not afford to perform N-fold cross validation as this would make the experiments about N times more costly.) DSE came in 10 folds plus a development fold twice as big as other folds. We used the development fold as the validation set and merged the first 5 folds for the train set and the rest for the test set. CRYST came in 5 folds. One fold became the validation set, 2 folds were merged for training and the rest became the test set.

For the rest of the datasets we tried to balance be-

[1]http://research.i2r.a-star.edu.sg/GEDatasets/Datasets.html
[2]http://ajbcentral.com/CrySis/dataset.html
[3]Art Munson, Personal Communication
[4]http://yann.lecun.com/exdb/mnist/

[5]http://komarix.org/ac/ds
[6]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets
[7]Eric Breck, Personal Communication

tween the following factors: (a) The test sets should be large enough so that differences between learning algorithms are apparent. (b) The test sets can be larger when the learning task is easy, but more data should be kept in the training set when the learning task is hard. (c) Some datasets would inevitably have more attributes than examples in the training set (IMDB, CITE, SPAM); for the rest we tried to put enough examples in the training set so that methods with small bias might learn something interesting. (d) The validation sets should be big enough so that parameter selection and calibration works well. In general we split the data so that we have around 50% in the training set and 50% in the test set. Validation data is drawn from the training set.

3. Results

Table 2 shows the performance of each learning method on each of the eleven problems. In the table, the problems are arranged left to right in order of increasing dimensionality. The table is broken into four sections. The top three sections report results for Accuracy (ACC), Squared Error (RMS), and Area under the ROC Curve (AUC). The bottom section is the average of the performance across these three metrics.

For each metric and problem, the performances have been standardized by dividing by the median performance observed for that problem and metric. Without standardization it is difficult to perform an unbiased comparison across different datasets and metrics. A score of one indicates that the method had typical performance on that problem and metric compared to the other learning methods. Scores above one indicate better than typical performance, while scores less than one indicate worse than typical performance. The scale for RMS has been reversed so that scores above one represent better than typical, i.e., lower, RMS.[8] The median performance for each problem and metric is included in the table to allow calculating raw performances from the standardized scores. The last column in the table is the average score across the eleven test problems. In each section of the table, learning algorithms are sorted by these average scores. The last column of the last section represents the average score across all problems and metrics.

Examining the results in the bottom section shows that on average across all problems and metrics, random forests have the highest overall performance. On average, they perform about 1% (1.0102) better than the typical model and about 0.6% (1.0102 vs. 1.0039) better than the next best method, ANN. The best methods overall are RF, ANN, boosted decision trees, and SVMs. The worst performing methods are Naive Bayes and perceptrons. On average, the top eight of ten methods fall within about 2% of each other. While it is not easy to achieve an additional 1% of performance at the top end of the scale, it is interesting that so many methods perform this similarly to each other on these high-dimensional problems.

If we examine the results for each of the metrics individually, we notice that the largest differences in performance among the different learning algorithms occur for AUC and the smallest differences occur for ACC. For accuracy, boosted decision trees are the best performing models followed by random forests. However, a closer examination of the table shows that boosted trees do better in accuracy mostly because of their excellent performance on the datasets with relatively low dimensionality. Comparing boosted trees with random forests in the left part of the table we see that random forests outperform boosted trees only on the TIS dataset. The situation is reversed on the right part of the table where boosted trees outperform random forests only on the CITE dataset. As dimensionality increases, we expect boosted trees to fall behind random forests.

In RMS, random forests are marginally better than boosted trees. This is confirmed by a bootstrap analysis (cf. Section 4): random forests have 33% and 35% chance of ranking 1st and 2nd respectively, while for boosted trees the corresponding probabilities are 31% and 21%. However, in AUC random forests are clear winners followed by, somewhat surprisingly, KNN.

Interestingly, although ANN is the 2nd best method overall in the bottom of the table, is does not perform 1st or 2nd for any of the individual metrics in the top of the table. It is 2nd overall only because ANNs consistently yield very good, though perhaps not exceptional, performance on all metrics.

A fact that is not apparent from the table is that calibration with isotonic regression works better than calibrating with Platt's method, or no calibration, on most problems and thus was used for almost all of the results reported in the table. Since our validation sets always are larger than 1000 examples, this confirms the findings in (Niculescu-Mizil & Caruana, 2005) that isotonic regression is preferred with large validation sets.

[8]This is different and simpler than the normalized scores used in (Caruana & Niculescu-Mizil, 2006). We have experimented with several ways of standardizing scores and the results change little with different methods. The learning methods that rank at the top (and the bottom) are least affected by the exact standardization method.

Table 2. Standardized scores of each learning algorithm

DIM	761	761	780	927	1344	3448	20958	105354	195203	405333	685569	—
ACC	Sturn	Calam	Digits	Tis	Cryst	Kdd98	R-S	Cite	Dse	Spam	Imdb	Mean
MEDIAN	0.6901	0.7337	0.9681	0.9135	0.8820	0.9494	0.9599	0.9984	0.9585	0.9757	0.9980	—
BSTDT	0.9962	**1.0353**	1.0120	0.9993	**1.0178**	0.9998	0.9904	1.0000	0.9987	0.9992	1.0000	1.0044
RF	0.9943	1.0103	1.0076	1.0025	1.0162	**1.0000**	0.9995	0.9998	1.0013	**1.0044**	1.0000	1.0033
SVM	1.0044	1.0018	1.0024	1.0060	1.0028	0.9999	**1.0156**	**1.0008**	1.0004	1.0008	**1.0003**	1.0032
BAGDT	1.0001	1.0350	0.9976	1.0017	1.0111	**1.0000**	0.9827	1.0000	0.9996	0.9959	1.0000	1.0021
ANN	0.9999	0.9899	1.0051	1.0007	0.9869	**1.0000**	1.0109	1.0001	**1.0018**	1.0029	**1.0003**	0.9999
LR	1.0012	0.9896	0.8982	**1.0108**	1.0080	**1.0000**	1.0141	1.0001	1.0014	1.0026	0.9999	0.9932
BSTST	1.0077	1.0298	0.9017	0.9815	0.9930	**1.0000**	0.9925	0.9999	0.9948	0.9905	0.9989	0.9900
KNN	**1.0139**	0.9982	**1.0122**	0.9557	0.9972	0.9999	0.9224	1.0000	0.9987	0.9698	0.9996	0.9880
PRC	0.9972	0.9864	0.9010	0.9735	0.9930	**1.0000**	1.0119	0.9999	1.0007	1.0041	1.0001	0.9880
NB	0.9695	0.9347	0.8159	0.9230	0.9724	**1.0000**	1.0005	1.0000	0.9878	0.9509	0.9976	0.9593
RMS	Sturn	Calam	Digits	Tis	Cryst	Kdd98	R-S	Cite	Dse	Spam	Imdb	Mean
MEDIAN	0.5472	0.5800	0.8449	0.7455	0.7051	0.7813	0.8257	0.9623	0.8154	0.8645	0.9597	—
RF	0.9980	1.0209	1.0186	1.0102	1.0277	1.0003	1.0011	0.9988	1.0072	**1.0118**	1.0006	1.0087
BSTDT	0.9993	1.0351	1.0363	0.9977	**1.0323**	0.9998	0.9781	1.0003	0.9983	1.0007	1.0003	1.0071
ANN	1.0042	0.9987	1.0088	1.0109	1.0014	1.0005	1.0315	1.0011	1.0068	1.0077	**1.0022**	1.0067
SVM	0.9979	0.9882	1.0076	**1.0149**	0.9972	0.9992	**1.0409**	**1.0091**	1.0067	0.9993	1.0004	1.0056
BAGDT	1.0007	**1.0357**	0.9924	1.0023	1.0218	0.9998	0.9587	1.0000	0.9994	0.9782	1.0012	0.9991
LR	1.0010	0.9963	0.8169	1.0232	0.9935	**1.0007**	1.0367	1.0009	**1.0082**	1.0073	0.9988	0.9894
PRC	0.9976	0.9841	0.8115	0.9537	0.9919	0.9998	1.0313	0.9979	1.0006	1.0071	0.9997	0.9796
BSTST	1.0078	1.0205	0.8202	0.9757	1.0021	**1.0007**	0.9861	1.0000	0.9900	0.9695	0.9952	0.9789
KNN	**1.0119**	1.0013	**1.0365**	0.9309	0.9986	1.0000	0.8468	0.9988	0.9983	0.9270	0.9941	0.9768
NB	0.9793	0.9509	0.7236	0.9031	0.9454	1.0000	0.9989	0.9981	0.9828	0.8984	0.9731	0.9412
AUC	Sturn	Calam	Digits	Tis	Cryst	Kdd98	R-S	Cite	Dse	Spam	Imdb	Mean
MEDIAN	0.6700	0.7793	0.9945	0.9569	0.9490	0.5905	0.9913	0.7549	0.9008	0.9957	0.9654	—
RF	0.9892	1.0297	1.0017	1.0069	1.0134	1.0140	1.0009	1.0962	**1.0304**	**1.0022**	**1.0209**	1.0187
KNN	**1.0397**	0.9992	1.0024	0.9509	1.0007	1.0165	0.9905	**1.1581**	1.0027	0.9902	0.9648	1.0105
LR	1.0045	0.9903	0.9424	**1.0136**	1.0070	1.0492	**1.0041**	1.0272	1.0293	0.9999	1.0084	1.0069
ANN	1.0132	1.0008	1.0001	1.0042	0.9992	1.0461	1.0031	0.9779	1.0105	1.0001	1.0021	1.0052
BSTST	1.0199	1.0304	0.9468	0.9901	0.9993	**1.0512**	0.9991	0.9956	0.9973	0.9989	1.0036	1.0029
SVM	0.9870	0.9645	1.0002	1.0077	0.9909	0.9324	1.0032	1.1120	1.0100	1.0011	0.9979	1.0006
BSTDT	0.9991	1.0492	**1.0033**	0.9958	**1.0137**	0.9605	0.9962	0.9646	0.9881	1.0015	1.0041	0.9978
BAGDT	1.0009	**1.0551**	0.9999	1.0062	1.0116	0.9768	0.9890	0.9673	0.9691	0.9925	0.9809	0.9954
PRC	0.9973	0.9630	0.9372	0.9749	0.9937	0.9724	1.0036	0.9991	0.9777	1.0006	0.9477	0.9788
NB	0.9329	0.8936	0.8574	0.9407	0.9574	0.9860	0.9990	1.0009	0.9917	0.9798	0.8787	0.9471
AVG	Sturn	Calam	Digits	Tis	Cryst	Kdd98	R-S	Cite	Dse	Spam	Imdb	Mean
RF	0.9938	1.0203	1.0093	1.0065	1.0191	1.0048	1.0005	1.0316	**1.0130**	**1.0061**	**1.0072**	1.0102
ANN	1.0058	0.9965	1.0047	1.0053	0.9958	1.0156	1.0152	0.9930	1.0064	1.0036	1.0015	1.0039
BSTDT	0.9982	1.0399	**1.0172**	0.9976	**1.0212**	0.9867	0.9882	0.9883	0.9950	1.0004	1.0014	1.0031
SVM	0.9965	0.9848	1.0034	1.0095	0.9970	0.9772	**1.0199**	1.0406	1.0057	1.0004	0.9995	1.0031
BAGDT	1.0006	**1.0419**	0.9966	1.0034	1.0148	0.9922	0.9768	0.9891	0.9894	0.9889	0.9940	0.9989
LR	1.0022	0.9921	0.8858	**1.0159**	1.0028	1.0166	1.0183	1.0094	1.0129	1.0033	1.0024	0.9965
KNN	**1.0219**	0.9996	1.0170	0.9458	0.9988	1.0055	0.9199	**1.0523**	0.9999	0.9623	0.9862	0.9917
BSTST	1.0118	1.0269	0.8896	0.9824	0.9982	**1.0173**	0.9926	0.9985	0.9941	0.9863	0.9992	0.9906
PRC	0.9974	0.9778	0.8832	0.9674	0.9929	0.9907	1.0156	0.9990	0.9930	1.0039	0.9825	0.9821
NB	0.9606	0.9264	0.7989	0.9223	0.9584	0.9953	0.9995	0.9997	0.9874	0.9430	0.9498	0.9492

3.1. Effect of Dimensionality

In this section we attempt to show the trends in performance as a function of dimensionality. In Figure 1 the x-axis shows dimensionality on a log scale. The y-axis is the *cumulative* score of each learning method on problems of increasing dimensionality. The score is the average across the three standardized performance metrics where standardization is done by subtracting the median performance on each problem.[9] Subtracting median performance means that scores above (below) zero indicate better (worse) than typical performance. The score accumulation is done left-to-right on problems of increasing dimensionality. A line that tends to slope upwards (downwards) signifies a method that performs better (worse) on average compared to other methods as dimensionality increases. A horizontal line suggests typical performance across problems of different dimensionality. Naive Bayes is excluded from the graph because it falls far below the other methods. Caution must be used when interpreting cumulative score plots. Due to the order in which scores are aggregated, vertical displacement through much of the graph is significantly affected by the performance on problems of lower dimensionality. The end of the graph on the right, however, accumulates across all problems and thus does not favor problems of any dimensionality, The slope roughly corresponds to the average relative performance across dimensions. From the plot it is clear that boosted trees do very

[9]Here we subtract the median instead of dividing by it because we are accumulating relative performance. Standardization by subtracting the median yields similar rankings as dividing by the median.

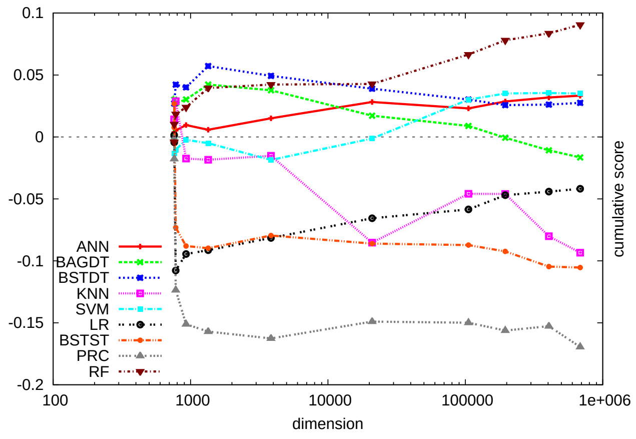

Figure 1. Cumulative standardized scores of each learning algorithm as a function of the dimension.

well in modest dimensions, but lose ground to random forests, neural nets, and SVMs as dimensionality increases. Also, linear methods such as logistic regression begin to catch up as dimensionality increases.

Figure 2 shows the same results as Figure 1, but presented differently to avoid the complexity of accumulation. Here each point in the graph is the average performance of the 5 problems of lowest dimension (from 761 to 1344), the 5 problems of highest dimension (21K to 685K) and 5 problems of intermediate dimension (927 to 105K). Care must be used when interpreting this graph because each point averages over only 5 data sets. The results suggest that random forests overtake boosted trees. They are among the top performing methods for high-dimensional problems together with logistic regression and SVMs. Again we see that neural nets are consistently yielding above average performance even in very high dimension. Boosted trees, bagged trees, and KNN do not appear to cope well in very high dimensions. Boosted stumps, perceptrons, and Naive Bayes perform worse than the typical method regardless of dimension.

Figure 3 shows results similar to Figure 2 but only for different classes of SVMs: linear-only (L), kernel-only (K) and linear that can optimize accuracy or AUC (L+P) (Joachims, 2006). We also plot combinations of these (L+K and L+K+P) where the specific model that is best on the validation set is selected. The results suggest that the best overall performance with SVMs results from trying all possible SVMs (using the validation set to pick the best). Linear SVMs that can optimize accuracy or AUC outperform simple linear SVMs at modest dimensions, but have little effect when dimensionality is very high. Similarly, simple linear SVMs though not competitive with kernel SVMs at low dimensions, catch up as dimensionality increases.

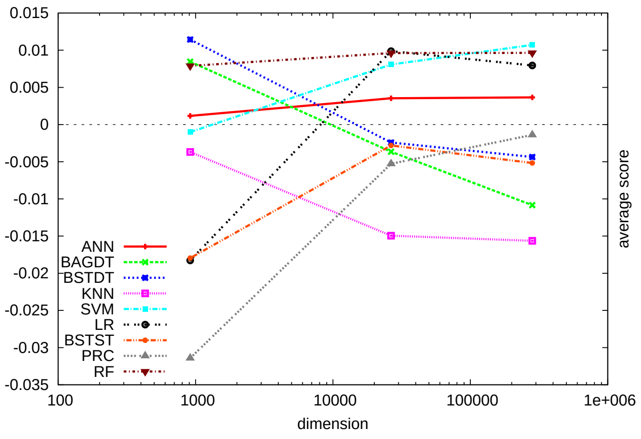

Figure 2. Moving average standardized scores of each learning algorithm as a function of the dimension.

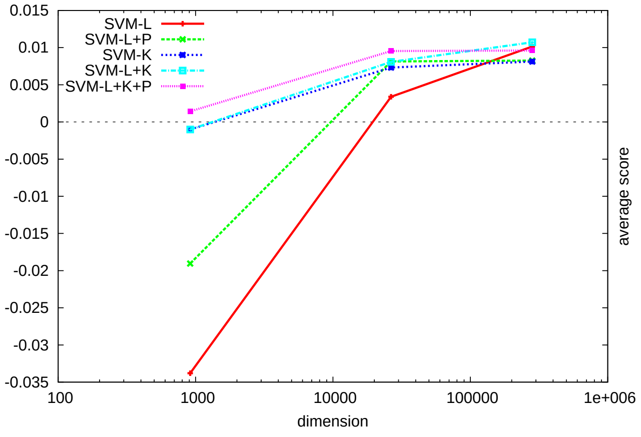

Figure 3. Moving average standardized scores for different SVM algorithms as a function of the dimension.

4. Bootstrap Analysis

We could not afford cross validation in these experiments because it would be too expensive. For some datasets and some methods, a single parameter setting can take days or weeks to run. Instead we used large test sets to make our estimates more reliable and adequately large validation sets to make sure that the parameters we select are good. However, without a statistical analysis, we cannot be sure that the differences we observe are not merely statistical fluctuation.

To help insure that our results would not change if we had selected datasets differently we did a bootstrap analysis similar to the one in (Caruana & Niculescu-Mizil, 2006). For a given metric we randomly select a bootstrap sample (sampling with replacement) from our 11 problems and then average the performance of each method across the problems in the bootstrap sample. Then we rank the methods. We repeat the bootstrap sampling 20,000 times and get 20,000 potentially different rankings of the learning methods.

Table 3 shows the results of the bootstrap analysis. Each entry in the table shows the percentage of time that each learning method ranks 1st, 2nd, 3rd, etc. on bootstrap samples of the datasets. Because of space limits, we only show results for average performance across the three metrics.

The bootstrap analysis suggests that random forests probably are the top performing method overall, with a 73% chance of ranking first, a 20% chance of ranking second, and less than a 8% chance of ranking below 2nd place. The ranks of other good methods, however, are less clear and there appears to be a three-way tie for 2nd place for boosted trees, ANNs, and SVMs.

5. Computational Challenges

Running this kind of experiment in high dimensions presents many computational challenges. In this section we outline a few of them.

In most high dimensional data features are sparse and the learning methods should take advantage of sparse vectors. For ANN, for example, when inputs are sparse, a lot of computation in the forward direction can be saved by using a matrix times sparse vector procedure. More savings happen when the weights are updated since the gradient of the error with respect to a weight going out of a unit with zero value vanishes. This is why our ANN implementation does not use momentum. If it did, all weights would have to be updated each iteration.

Another caveat is that for tree learning algorithms, indexing the data by feature instead of by example can speed up queries about which examples exhibit a particular feature. These queries are common during learning and one should consider this indexing scheme. Our random forest implementation indexes by feature.

Boosted decision trees on continuous data was the slowest of all methods. For bagged trees running times were better because we only grew 100 trees that can be grown in parallel. The same holds for random forests which have the added benefit that computation scales with the square root of dimensionality. Training ANNs was sometimes slow, mainly because applying some of the techniques in (Le Cun et al., 1998) would not preserve the sparsity of the data. For SVMs and logistic regression, we didn't have computational problems thanks to recent advances in scaling them (Genkin et al., 2006; Bordes et al., 2005; Joachims, 2006; Shalev-Shwartz et al., 2007). As a sanity check we compared the performance of the approximate kernel SVM solver with the exact SVM^{light} on some of our smallest problems and found no significant difference. Naive Bayes and perceptrons are among the fastest methods. KNN was sufficiently fast that we didn't have to use specialized data structures for nearest neighbor queries.

6. Related Work

Our work is most similar to (Caruana & Niculescu-Mizil, 2006). We already pointed out shortcomings in that study, but we also borrowed much from their methodology and tried to improve on it. STATLOG (King et al., 1995) was another comprehensive empirical study that was discussed in Section 1. A study by LeCun (LeCun et al., 1995) compares learning algorithms not only based on traditional performance metrics but also with respect to computational cost. Our study addresses this issue only qualitatively. Clearly, computational issues have to be taken into consideration in such large scale. A wide empirical comparison of voting algorithms such as bagging and boosting is conducted in (Bauer & Kohavi, 1999). The importance of evaluating performance on metrics such as AUC is discussed thoroughly in (Provost & Fawcett, 1997). The effect of different calibration methods is discussed in (Niculescu-Mizil & Caruana, 2005).

7. Discussion

Although there is substantial variability in performance across problems and metrics in our experiments, we can discern several interesting results. First, the results confirm the experiments in (Caruana & Niculescu-Mizil, 2006) where boosted decision trees perform exceptionally well when dimensionality is low. In this study boosted trees are the method of choice for up to about 4000 dimensions. Above that, random forests have the best overall performance. (Random forests were the 2nd best performing method in the previous study.) We suspect that the reason for this is that boosting trees is prone to overfitting and this becomes a serious problem in high dimensions. Random forests is better behaved in very high dimensions, it is easy to parallelize, scales efficiently to high dimensions and performs consistently well on all three metrics.

Non-linear methods do surprisingly well in high dimensions if model complexity can be controlled, e.g. by exploring the space of hypotheses from simple to complex (ANN), by margins (SVMs), or by basing some decisions on random projections (RF). Logistic regression and linear SVMs also gain in performance as dimensionality increases. Contrary to low dimensions, in high dimensions we have no evidence that linear SVMs can benefit from training procedures that directly op-

Table 3. Bootstrap analysis of rankings by average performance across problems

AVG	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
RF	0.727	0.207	0.054	0.011	0.001	0	0	0	0	0
ANN	0.053	0.172	0.299	0.256	0.119	0.072	0.019	0.011	0	0
BSTDT	0.059	0.228	0.18	0.222	0.18	0.075	0.044	0.012	0.001	0
SVM	0.043	0.195	0.213	0.193	0.156	0.088	0.08	0.031	0.001	0
LR	0.089	0.132	0.073	0.075	0.108	0.177	0.263	0.081	0	0
BAGDT	0.002	0.012	0.109	0.123	0.251	0.284	0.123	0.078	0.016	0
KNN	0.023	0.045	0.051	0.057	0.085	0.172	0.122	0.177	0.258	0.01
BSTST	0.004	0.009	0.021	0.063	0.086	0.109	0.3	0.387	0.02	0
PRC	0	0	0	0	0.013	0.024	0.047	0.222	0.695	0
NB	0	0	0	0	0	0	0	0	0.01	0.99

timize specific metrics such as AUC.

The results suggest that calibration never hurts and almost always helps on these problems. Even methods such as ANN and logistic regression benefit from calibration in most cases. We suspect that the reasons for this are the availability of more validation data for calibration than in previous studies and that high dimensional problems are harder in some sense.

Acknowledgments

We thank all the students who took CS678 at Cornell in the spring of 2007 and helped with this study. We especially thank Sergei Fotin, Michael Friedman, Myle Ott and Raghu Ramanujan who implemented KNN, ANNs, Random Forests and Boosted Trees respectively. We also thank Alec Berntson Eric Breck and Art Munson for providing the crystallography, DSE and ornithology datasets respectively. Art also put together the calibration procedures. Finally, we thank the 3 anonymous reviewers for their helpful comments. This work was supported in part by NSF Awards 0412930 and 0412894.

References

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *MLJ*, *36*, 105–139.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *JMLR*, *6*, 1579–1619.

Breiman, L. (2001). Random Forests. *MLJ*, *45*, 5–32.

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *ICML '06*, 161–168.

Freund, Y., & Schapire, R. (1999). Large Margin Classification Using the Perceptron Algorithm. *MLJ*, *37*, 277–296.

Genkin, A., Lewis, D., & Madigan, D. (2006). Large-scale bayesian logistic regression for text categorization. *Technometrics*.

Joachims, T. (2006). Training linear SVMs in linear time. *SIGKDD*, 217–226.

King, R., Feng, C., & Shutherland, A. (1995). Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, *9*, 259–287.

Le Cun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient backprop. In *Neural networks, tricks of the trade*, LNCS 1524. Springer Verlag.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. *International Conference on Artificial Neural Networks*, *60*.

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. *ICML '05*, 625–632.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, *10*.

Provost, F. J., & Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *KDD '97* (pp. 43–48).

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. *ICML '07* (pp. 807–814).

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. *KDD '02*, 694–699.