A study of the NIPS feature selection challenge

Nicholas Johnson

November 29, 2009

Abstract

The 2003 Nips Feature extraction challenge was dominated by Bayesian approaches developed by the team of Radford Neal and Jianguo Zhang. In a follow-up report Bayesian neural networks were used in a more principled way and the performance was largely reproduced. This report investigates the performance of competing methods on the same features to determine whether the initial feature selection or the unique approach to classification was responsible for the good performance. We also compared against Gradient Boosted Trees which were not in wide use at the time of the contest.

1 Introduction

The 2003 Nips Feature extraction challenge consisted of five classification problems (Guyon et al., 2006). The theme was classification of high dimensional data, and four of the classification tasks were characterized by thousands or even hundreds of thousands of features. The fifth data set was constructed so that no single feature, pair or even 4-tuple of features was informative. The contest was dominated by the Bayesian approach of Radford Neal and Jianguo Zhang, and in (Guyon et al., 2006) Neal outlines a more principled approach to the feature reduction. With these feature sets he used Bayesian neural networks (Neal, 1996) to reproduce the original success.

Neal describes using an initial filter of univariate significance tests to reduce the number of features to a manageable size. A Bayesian neural network is trained using these features and examination of ARD hyperparameters further reduces the number of features. Just as Neal does, we will denote the initial reduction (feature selection) as "SEL" and the second ARD reduction as "RED". Principle components are also used as features, but we do not consider them as they rarely produced lower error rates.

As a comparison we trained Gradient Boosted Trees (Friedman, 1999), Random Forests (Breiman, 2001), boosted neural networks using RealBoost (Friedman et al., 2000), and bagged neural nets using a similar architecture as Neal's Bayesian nets (2 hidden layers with 20 and 8 hidden units respectively). We compare the performance with Neal's NewBayes Bayesian neural network approach and examine the benefit of the ARD feature reduction. Table (1) briefly

describes each dataset in terms of the number or training and validation examples and the number of features used in the SEL and RED datasets. In the first phase of the contest entrants were only given class labels to the training examples and they had to verify performance on the validation set through the competition website [http://www.clopinet.com/isabelle/Projects/NIPS2003/]. In the final week of the competition, validation labels were released and entrants were able to fit to these as well and submit predictions for a final test set.

Name	Total Features	"SEL" features	"RED" features	# examples	class prior
Arcene	10,000	468	100	200	50/50
Dexter	20,000	292	100	600	50/50
Dorothea	100,000	683	72	1150	90/10
Gisette	5000	1133	326	7000	50/50
Madelon	500	38	21	2600	50/50

Table 1: Dataset descriptions.

In sections 2 through 4 we describe the settings and choices made when fitting the neural network and tree based classifiers; in section 5 we compare performance of Bayesian logistic regression with L_1 penalized logistic regression; finally, in section 6 we compare the methods by their test set accuracies.

2 Boosted Trees and Random Forests

We used these classification methods as implemented by the 'gbm' package of (Ridgeway, 2005) and the 'randomForest' package of (**?**). Bagging was employed at each boosting iteration and the subsampling rate was set to 80%. Shrinkage was used in all cases, but the amount varied between 0.1 and 0.005 depending on training time considerations. When boosting we used Bernoulli loss rather than adaboost as suggested by (Hastie et al., 2001). Random Forests were fit using default settings as there was no evidence that tuning would significantly alter performance.

In most datasets cross validation runs indicated that fairly shallow trees were sufficient. A single exception was the "madelon" data set which shows in figure (1) that performance continued to improve with deeper trees. This data set is characterized by having no single feature or even four-tuple of informative features and so we expect that higher order interactions are necessary. The left panel shows error rates using "SEL" and the right panel shows cross validation error rates using "RED" and the red lines are Neal's test set error rates.

In general tree based methods saw little benefit from "RED" over "SEL". This is in contrast to boosted neural networks which we present next.

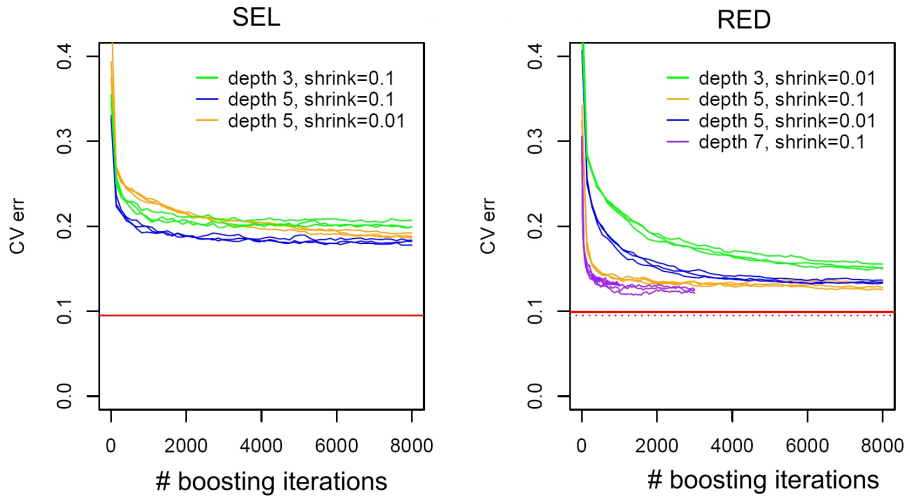

Figure 1: Cross validated error rates for boosted trees using Bernoulli loss on the "madelon" date set. SEL and RED are the two feature sets. Red lines are Neal's reported error rates

3 Boosted Neural Networks

Boosted neural networks were used by (LeCun et al., 1998) who fit a small number of very large networks. In contrast to their approach we follow (Hastie et al., 2001) a bit more closely and fit very weak "base learners". Neural networks have a tendency to overfit, so to regularize or weaken the ensemble members we applied a combination of weight decay and early stopping to single layer networks with between 2 and 4 hidden units.

The use of early stopping and small networks led to fitting times significantly shorter than say bagging much larger networks. Table (2) shows a running time comparison to fit Boosted Trees, Random Forests, Boosted Neural Nets, and Bagged Neural nets using the optimal settings we could determine where "optimal" means having lowest cross validated error rate.

Method	Train. Time	Ensemble Size	Train. Iterations	Hidden Units/Depth
Bagged Nets	4hrs	200	7000	20-8
Boosted Nets	272 s	90	50	4
Random Forests	5.3 s	1,500	NA	∞
Boosted Trees	185.7 s	15,000	NA	5

Table 2: Fitting times on Arcene "SEL" data.

Figure (2) shows the progression of boosted neural network fits on some toy

data. Supporting Friedman et al's interpretation of boosting we see a fitting rather than averaging of the decision boundary.

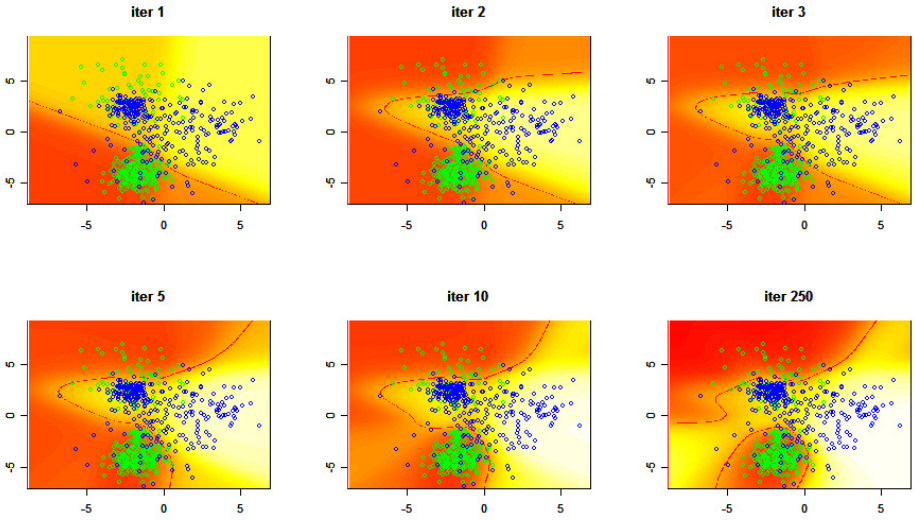

Figure 2: Progression of a boosted neural network fit on toy data. Each base learner uses two hidden units.

Tuning neural networks is time consuming due to the fact that there are three tuning parameters (weight decay, stopping time, and network size). Boosted trees are tuned simply by trying a handful of tree depths with the lowest being the obvious first choices. In contrast, it is not clear which early stopping time or value of weight decay parameter will give the best performance.

Figure (3) shows cross validated error rates when boosting neural networks on the Arcene data. The horizontal red lines are test error rates as posted by Neal. We see no evidence of overfitting with the number of boosting iterations. When compared with the test error rates in table (3) we see that these estimates are optimistic due to the initial feature selection which was performed on the training data.

4 Bagged Neural Networks

For completeness we also trained two layer neural networks using the same architecture as Neal (20 and 8 hidden units). Early stopping is the most efficient tuning parameter to cross validate since the network weights can be saved between backpropagation batches. With the exception of the Madelon data set, we set the weight decay to a very small number. We tried to modify Neal's own

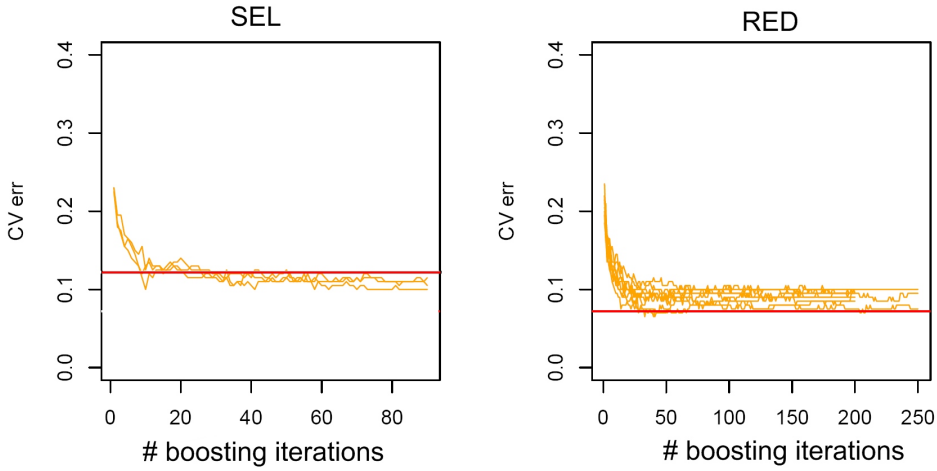

Figure 3: Cross validated error rates for boosted neural networks (each line is one fold) on the arcene data set. 50 and 80 training iterations used for each base learner in the left and right panels respectively and for both panels four hidden units and a weight decay of 0.1 was used. The red lines are Neal's test set performance (see table 3).

scripts to perform gradient descent, but switched to Matlab's neural network toolbox when it gave significantly better performance.

This is the most direct comparison with the Bayesian methodology since we use networks of the same structure and average over many sampled networks. Neal's approach will average over networks whose weights are in regions of the parameter space with high average likelihood. Bagging also averages over networks but with the goal of reducing variance. By averaging over bootstrap samples of the data we hope to reduce the variance more than just an average over restarts of backpropagation the algorithm.

We found that bagging smaller single layer networks often gave better performance. Two hidden layers and twenty hidden units may be too many for some of these datasets, and so it should be noted that Neal's methods give consistent performance with the same architecture in tuning in all data sets.

5 A brief comparison between logistic regression approaches

Since we are comparing popular frequentist approaches to Neal's Bayesian models, we compare logistic regression approaches as he has posted error rates for these. In fact, on the Dexter dataset Bayesian logistic regression produced the best error amongst all of his methods. We also found here that boosting two-

hidden-unit networks gave better performance than larger network sizes.

An L_2 penalized logistic regression is the posterior mode under a Gaussian prior. The Bayesian approach on the other hand will average over many sampled networks. So under L_1 penalization the mode is sparse; however, none of the Bayesian's sampled models would have zero coefficients. Neal's own prior is hierarchical and more complicated than either of these; we compare against his posted results.

We fit L_1-penalized logistic regression using the 'glmPath' package of (Park and Hastie, 2007) and examined cross validated error rates along several points of the regularization path. Figure (4) shows cross validated error rates along the regularization path, and the test error posted by Neal is shown as a horizontal red line. In both cases the Bayesian model shows lower error rates (despite the downward bias of the cross validation estimates).

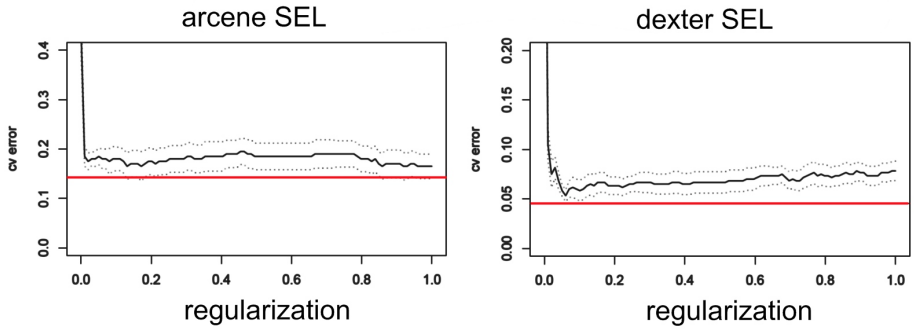

Figure 4: Cross validation error rates L_1 penalized logistic regression in the arcene and dexter data sets. Dotted curves are $\pm$ one standard deviation. The red lines are Neal's error rate using Bayesian logistic regression.

6 Test Error Rates

In the following tables we use a shorthand. 'gbm' indicated boosted trees, 'nnet rb' indicated RealBoosted neural networks, 'rf' are tree based random forests, and 'NewBayes' are Neal's bayesian neural networks. 'test err' is a balanced error misclassification error rate, and 'test auc' is an ROC area under the curve where predictions are ordered by say absolute log odds.

Most methods with the exception of boosted neural networks did not benefit from the ARD feature reduction. The gains by boosted neural networks on the reduced featureset may be due to the ease of parameter tuning. A smaller featureset reduces the computation time for cross validation runs. Boosted trees are much easier to tune, and current software implementations are very fast. Random Forests are generally not believed to be competetive with boosted

Method	SEL test err	RED test err
NewBayes	**12.18**	**7.2**
gbm	13.06	14.11
nnet rb	14.15	8.68
bagged nnet	16.53	17.59
rf	13.86	13.87

Table 3: Arcene Test Performance.

Method	SEL test err	RED test err
NewBayes	4.55	**5.55**
gbm	7.15	7.2
nnet rb	**4.2**	6.25
bagged nnet	4.75	7.7
rf	4.6	6.2

Table 4: Dexter Test Performance.

trees, yet here they posted competitive performance despite requiring no tuning and having the fastest run times.

7 Conclusions

Using the same features, Bayesian neural networks were competitive with or superior to the methods we considered. The test set performance of Bayesian nets was also generally better than bagged or boosted neural networks. This performance difference may be due the advanced Hybrid Monte Carlo search strategy (Liu, 2001) or it may be due to the more complex regularization of the ARD priors.

Two surprises are the good performance (relative to boosted trees) of Random Forests and boosted neural networks. Neural networks must be regularized by the number of training epochs or the weight decay which is not as easy to set as the depth of a binary decision tree, and so they are not typically boosted. The SEL test error rates show that they are a valid alternative.

8 Appendix : Implementation details

Settings for boosted trees are given in table (8) using Ridgeway's 'gbm' package for R. Eighty percent of the data was bagged at each boosting iteration.

For boosted neural networks we used the R 'nnet' package and used 4 hidden units on all datasets except for Dexter (RED/SEL) and Dorothea (SEL) where we used 2 hidden units. Weight decay and the number of fitting iterations were chosen by trial and error with goal of keeping the base learners from perfectly

Method	SEL test err	RED test err
NewBayes	**11.21**	**19.56**
gbm	17.96	22.57
nnet rb	13.62	22.02
bagged nnet	12.59	21.71
rf	20.98	23.82

Table 5: Dorothea Test Performance.

Method	SEL test err	RED test err
NewBayes	1.75	1.86
gbm	**1.74**	2.02
nnet rb	1.75	**1.78**
bagged nnet	1.89	2.11
rf	2.17	2.54

Table 6: Gisette Test Performance.

fitting the training data on the first iteration. The ensemble size was chosen by cross validation and the settings are given in table (9).

When bagging neural networks we stuck to the 20-8-2 architecture. The Matlab Neural Network toolbox (v5.1) was used on Dorothea (SEL/RED), Madelon (SEL/RED) and Gisette (RED). We used Neal's own scripts, modified to perform backpropagation, on the other datasets. We bagged between 50 and 200 networks depending on computation time; Gisette was the most time consuming dataset to train on and was averaged over only 50 networks. We set the number of training epochs by cross validation using the out-of-bag data. Settings are shown in table (10)

References

Breiman, L. (2001). Random forests. *Machine Learning 45*(1), 5–32.

Friedman, J. (1999). Greedy function approximation: a gradient boosting machine. Technical report, Dept. of Statistics, Stanford University.

Friedman, J., T. Hastie, and R. Tibshirani (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics. 38*(2), 337–74.

Guyon, I., S. Gunn, M. Nikravesh, and L. Zadeh (Eds.) (2006). *Feature Extraction, Foundations and Applications.* Springer-Verlag.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *Elements of Statistical Learning.* Springer-Verlag.

Method	SEL test err	RED test err
NewBayes	**9.5**	9.94
gbm	18.72	20.39
nnet rb	13	9.78
bagged nnet	14.44	10.22
rf	16.67	**9.5**

Table 7: Madelon Test Performance.

Data set	Depth	# Trees	Shrinkage
SEL			
Arcene	5	15000	0.005
Dexter	3	4000	0.01
Dorothea	5	2700	0.01
Gisette	5	8000	0.1
Madelon	5	8000	0.1
RED			
Arcene	5	15000	0.005
Dexter	3	15000	0.005
Dorothea	1	5000	0.01
Gisette	3	6000	0.1
Madelon	7	2000	0.1

Table 8: The settings above were used with boosted trees.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–324.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing.* Springer-Verlag.

Neal, R. (1996). *Bayesian Learning for Neural Networks.* Springer-Verlag.

Park, M. and T. Hastie (2007). An l1 regularization-path algorithm for generalized linear models. *Journal of Royal Statistical Society, Series B 69*(4), 659–677.

Ridgeway, G. (2005). Generalized boosted models: A guide to the gbm package.

Data set	# Hidden Units	Ensemble size	# Fitting Iterations	Weight decay
		SEL		
Arcene	4	90	50	0.1
Dexter	2	125	5	0.2
Dorothea	2	80	5	0.001
Gisette	4	500	20	0.4
Madelon	4	250	140	0.005
		RED		
Arcene	4	60	80	0.1
Dexter	2	300	5	0.1
Dorothea	4	70	15	0.001
Gisette	4	600	20	0.4
Madelon	4	200	140	0.005

Table 9: The settings above were used with boosted neural networks.

Data set	Ensemble Size	# Epochs	Software
	SEL		
Arcene	200	7000	FBM
Dexter	200	1500	FBM
Dorothea	100	300	Matlab
Gisette	50	3000	FBM
Madelon	50	5000	Matlab
	RED		
Arcene	200	90000	FBM
Dexter	200	8000	FBM
Dorothea	100	300	Matlab
Gisette	50	6000	Matlab
Madelon	50	5000	Matlab

Table 10: The settings above were used with bagged 20-8-2 neural networks. FBM denotes Neal's Flexible Bayesian Modeling software.