

**CS578 Empirical Methods in Machine Learning
and Data Mining**

Final Project Write-Up

The Weak Learners

Biswaroop Chatterjee (bc228)

Mayank Gandhi (mg327)

Aakash Jain (aj79)

Venkat Krishnaraj (vk73)

Fall 2004

1	Introduction.....	3
2	The Plan and Approach.....	4
3	Running the Initial Experiments	5
3.1	Decision Trees	5
3.2	kNN.....	7
3.3	ANN.....	9
3.4	SVM.....	13
4	Optimizing performance	16
4.1	Feature Selection.....	16
4.2	Bagging Decision Trees.....	17
4.3	Boosting Decision Trees.....	18
4.4	Bagged Boosted Decision Trees	19
4.5	Bagging ANN	20
4.6	Boosting ANN	21
4.7	Unweighted and Weighted Combination.....	22
5	Final Selection and Conclusions.....	24
	Contributions	25
	Appendix A.....	26
	Resilient Backpropagation (trainrp)	26

1 Introduction

The report serves as the write-up for the CS578 final mini-competition/project, summarizing our technical approach, our experimentation plan and the results.

The goal of the project was to apply decision trees, neural networks, k-nearest neighbor (kNN) and/or SVMs to a data set to train a best possible model for the dataset. As the report outlines, we have applied all these techniques, along with crossvalidation, feature selection, bagging and boosting, and model averaging to train our final model and compute predictions for the 20000 test data points.

Section 2 outlines our plan and approach. Section 3 deals with the basic results we obtained on decision trees (3.1), kNN (3.2), artificial neural networks (ANNs) (3.3), and SVMs (3.4).

Section 4 deals with how we implemented additional techniques with our basic performance, using feature selection (4.1) and bagging and boosting (4.2 – 4.6). Finally, we combined our predictions from bagged and boosted decision trees and bagged neural networks by computing their weighted average into one final model, which gave us an accuracy of 88.70% on our test set. This is explained in section 4.7. Section 5 contains the conclusion and some evaluations. Contributions are in section 6.

2 The Plan and Approach

Our plan was to first identify, train and tune a best possible model for decision trees, kNN, neural networks, and SVMs. This would give us an idea about the basic performance for each of the models before we further optimized or combined predictions from any of the models.

Having done this, we identified decision trees and neural networks as being the most promising models and decided to re-train the neural networks with feature selection implemented. We used the decision trees to quantify the importance of each of the features and run feature selection.

While retraining and retuning the neural networks for optimal performance with feature selection, we also decided to bag and boost the predictions from the neural network. Boosting did not improve performance significantly, however, bagging did help. Simultaneously we also bagged and boosted the predictions of the MML decision trees which gave us significant improvements in performance.

At this stage we were reaching accuracy levels in the ~83-84% level using one of the two best methods mentioned above following boosting and/or bagging. We then decided to combine the predictions of the Decision Trees and the Neural Networks with the idea that averaging the results might increase performance. We immediately saw an increase in the performance on all the metrics. So we tried combining the predictions from various neural networks and decision tree combinations.

Finally, we combined the (weighted) predictions of the bagged and boosted decision trees with that of the bagged neural networks into one final model and used it to compute the predictions that were submitted for the 20000 points of the test data set.

3 Running the Initial Experiments

Section 3.1 through 3.4 provide the methodology and results of running plain vanilla decision trees, kNN, neural networks and SVMs respectively on the supplied dataset.

3.1 Decision Trees

We used the IND decision tree package to grow trees of type C4, ID3, MML, SMML and Bayes on a randomly drawn training sample of size 3000. For each tree type we grew 10 trees, each time randomly picking the 3000 training samples and testing on the remaining 2000 samples.

Figure 3.1.1, below, shows 10 such runs for the c4 decision tree. c4 was especially picked for this figure, since it shows the most variation in tree size and accuracy, and the apparent lack of correlation between the two over the 10 runs, which we found to be interesting.

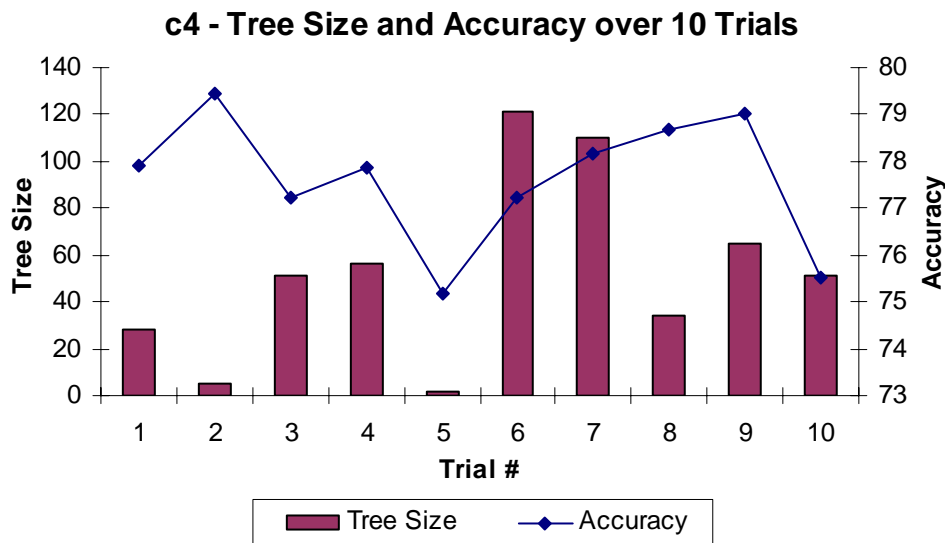


Figure 3.1.1

Table 3.1, below, shows the mean accuracy, RMSE, and tree size, over 10 trials, for each of the trees, along with the corresponding variances:

Tree Type	Accuracy	Variance	RMSE	Variance	Tree Size	Variance
c4	77.62	1.96	0.4038	0.00035	52.3	1546.7
id3	76.75	1.98	0.4821	0.00022	343.0	44.0
mml	80.01	0.84	0.3771	0.00007	393.9	167.7
smml	79.07	0.66	0.3772	0.00002	14.5	4.5
bayes	77.78	1.16	0.4210	0.00009	418.7	213.1

Table 3.1: Mean Accuracy, RMSE and Tree Size

Plotting the accuracy and RMSE values from table 3.1 in figure 3.2 below, we see that on average MML gives best performance based on both accuracy and RMSE. SMML follows closely behind. Even though it builds

smaller trees and shows only a slightly worse RMSE performance with in fact a much smaller variance, we let the mean values for accuracy and RMSE rule our decision on which decision tree to pick for further experimentation and optimization (as described in section 4), thus picking MML.

Accuracy and RMSE for Each of the Grown Decision Trees

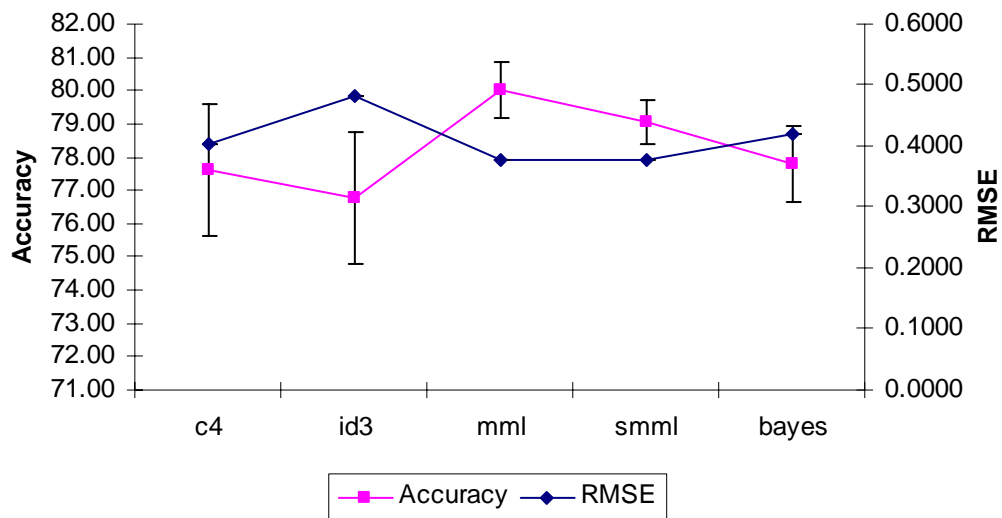


Figure 3.1.2

C4, ID3 and Bayes all performed in a consistently and considerably worse manner on the performance metrics.

Using Look-ahead resulted in similar averaged performance for each of the grown trees, thus we do not present the results of Look-ahead. Besides, we are not too concerned about tree size here, since we just want to optimize our predictions to give the best accuracy and RMSE.

3.2 kNN

To study the performance of kNN on the given test set, we used Leave-One-Out-Cross-Validation (LOOCV) to give us an estimation regarding the performance. We used different values of k and studied how this had an effect on the performance. We used the unscaled approach and scaling. The scalings we used were MinMax and standard deviation.

Figure 3.2.1 below depicts the accuracy plotted against increasing values of k for the various attribute scalings.

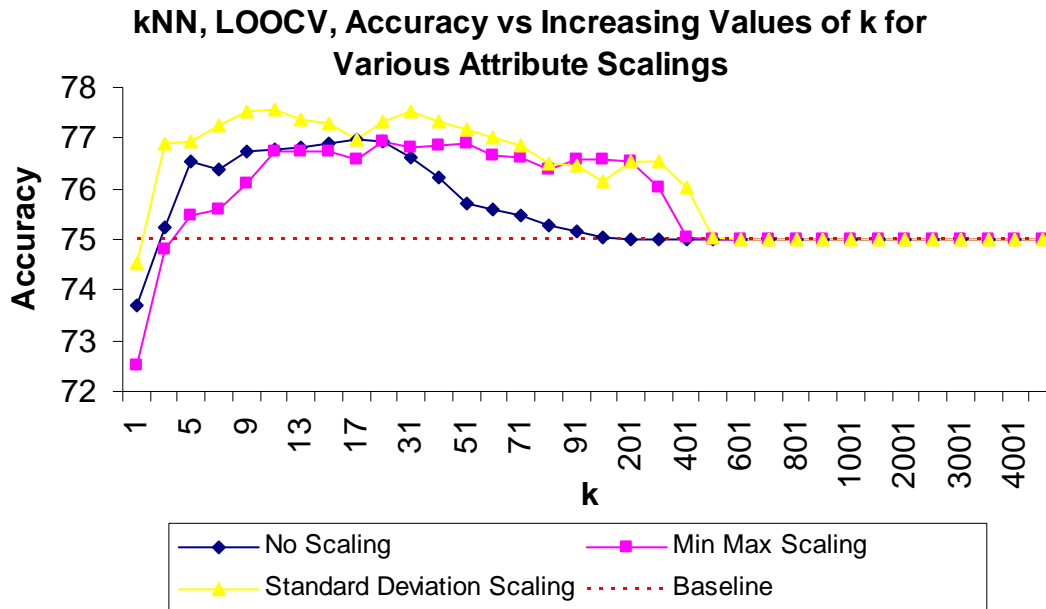


Figure 3.2.1

We observed that standard deviation scaling worked the best amongst the three options. The maximum accuracy that we were able to reach was 77.56% for k=11. The trends that are observed on this data set are typical as is usually seen with kNN. The accuracy starts off below baseline for all three scaling methods and rises with the number of neighbors used to make the final prediction, i.e. the k values.

The unscaled method achieved roughly the same accuracy performance as the MinMax scaling (76.95%) for k values 17 and 19 respectively which is different from the peak value for standard deviation scaling seen earlier (k=11). The kNN performance with standard deviation scaling drops to about 76.98% for k=17 and then rises again, which was a bit unusual.

Figure 3.4 below depicts the RMSE plotted against increasing values of k for the various attribute scalings.

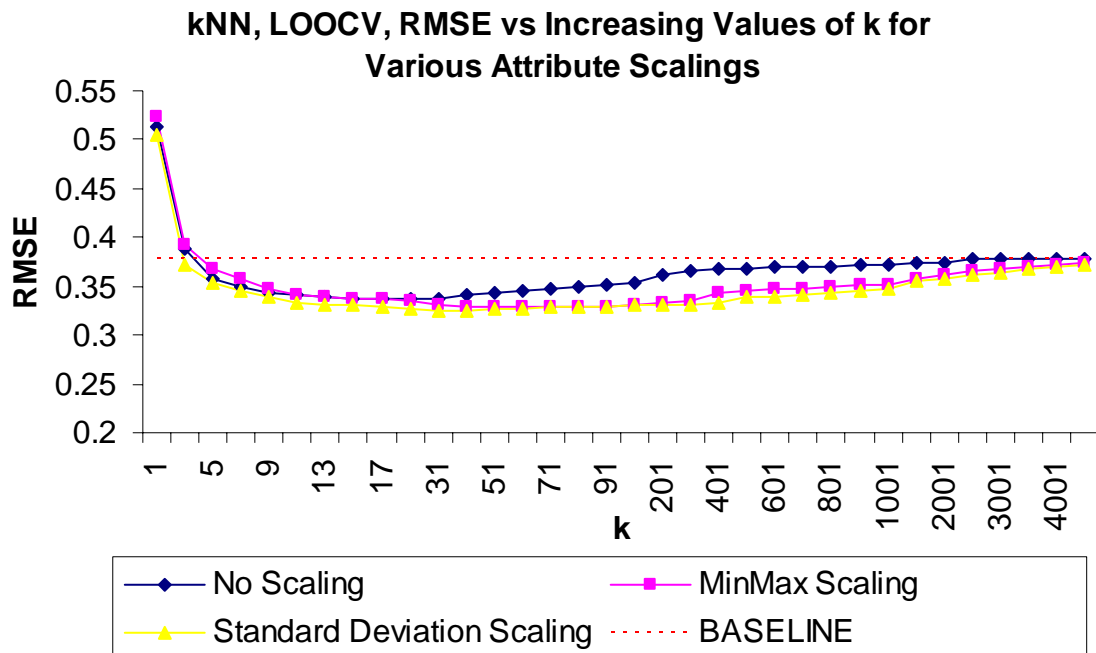


Figure 3.2.2

With regards to the RMSE, the standard deviation scaling again performed the best achieving 0.3239 for k=31. The unscaled method and the MinMax scaled method achieved minimum RMSE values of 0.3364 (k=19) and 0.32864 (k=51) respectively.

Though the kNN seems to perform reasonably well with regards to the RMSE, it is woefully inadequate performance-wise in the accuracy domain. This seems to indicate that though the kNN doesn't get it right very often, averaging over k nearest neighbors yields results which may give the wrong prediction but does not err by much. Based on this observation, it might have been prudent to try out weighted model averaging (as described in section 4) using the predictions from kNN along with models trained using neural networks and decision trees. However, we did not do this!

3.3 ANN

We ran ANN on 3000 training units. We kept 1000 for finding the early stopping point and saved the last 1000 for the test set. Initially we first tried traingdm (Vanilla backprop, learning rate 0.1, momentum = 0.7, and varying number of hidden units from 4 to 32 in multiples of 2. We then performed the same experiment using trainrp We found out that trainrp and traingdm yield almost the same results, except that trainrp converges much faster. Therefore since trainrp runs faster, hence we decided to use trainrp so that we can experiment with boosting and bagging with the ANNs which take time. Now for the purposes of the ANN experiment we needed to figure out the optimal learning rate, number of hidden units, the type of scaling, the early stopping point, the type of function to use and of course the training method.

Initially we tried to figure out the optimal learning rate, we ran the experiment using 8 hidden units ANN, but varying the learning rate from 0.01 to 0.1 in steps of 0.1. We found out that the best performance was on 0.06. Using 8 hidden units, one hidden layer and one output unit, we first trained the ANN to determine an optimal learning rate. Figure 3.3.1 shows the results for a selected region where the ANN was performing the best. The graph only shows some selecting learning rates which showed the best performance.

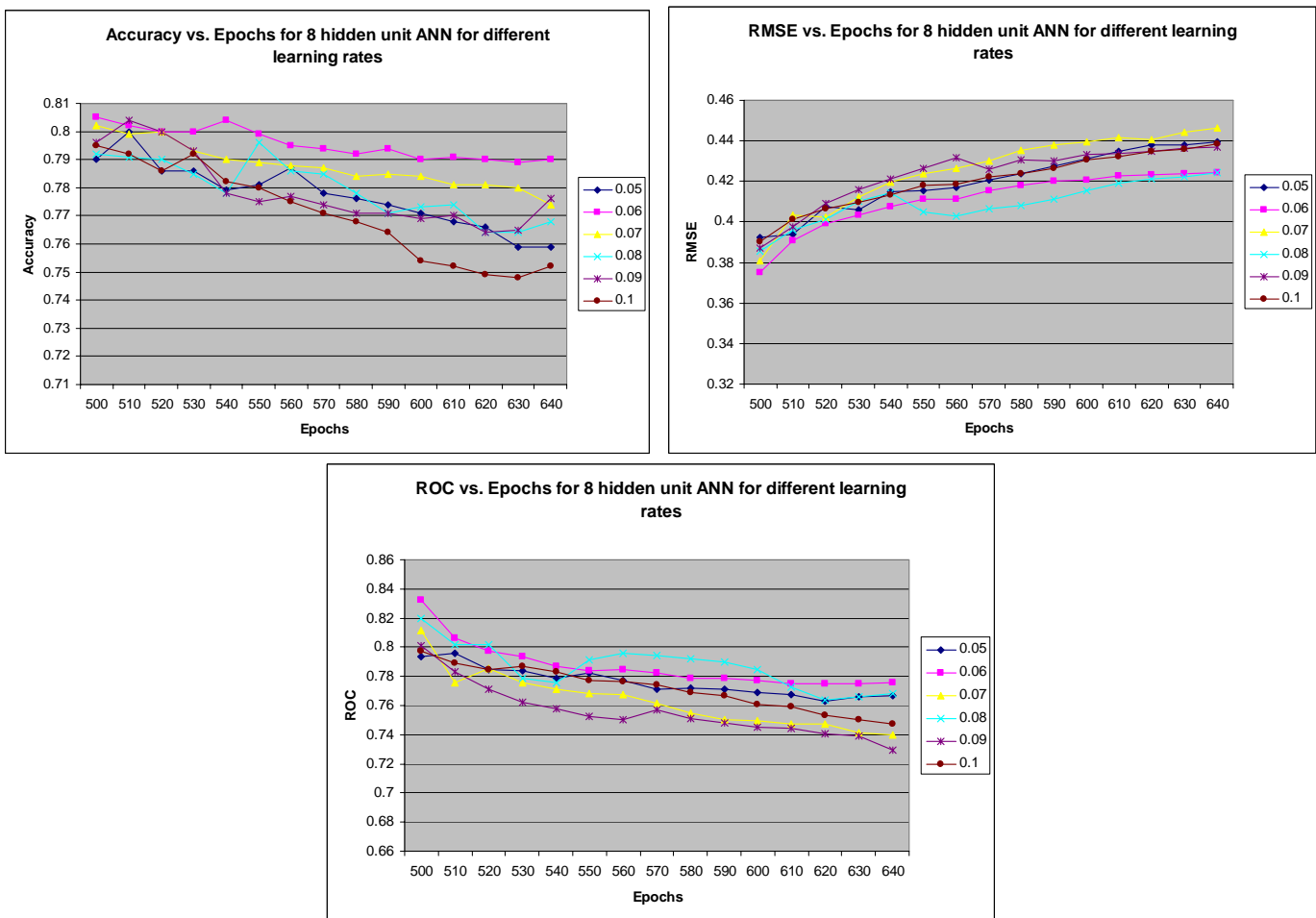


Figure 3.3.1

Based on the graphs above, we picked a value of 0.06 as the optimal value for the learning rate which clearly performed the best based on accuracy, RMSE and ROC. Though the learning rate of 0.08 was a not too distant second it wasn't performing very well accuracy wise so we decided against it.

Then we decided to experiment with the number of hidden units. From the results for the three metrics Accuracy, RMSE and ROC, we could see that 8 hidden units were performing the best of the lot though the results weren't totally conclusive so we could not eliminate 16 hidden units totally. We also found out that the early stopping point was similar around the 500 epochs mark from the learning rate experiment as well as this hidden unit experiment. Figure 3.3.2 shows the results of the experimentation with the different hidden units. For the purposes of clarity we will only show the results for a range of epochs from 500 to 900. Here we can see that 8 hidden units gets a good accuracy of about 82.5% and a RMSE value of ~0.36 and the ROC is also the best of the lot at 82%

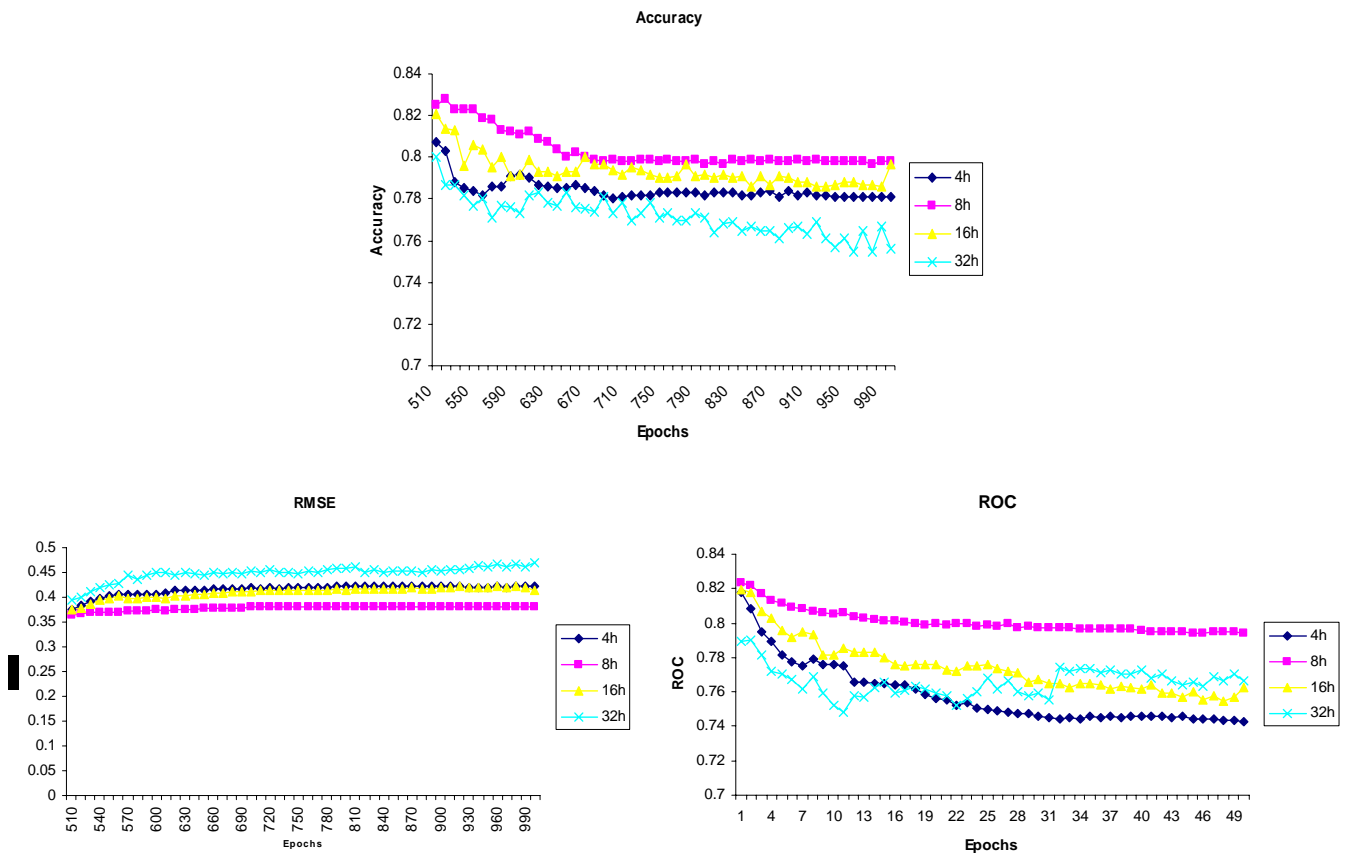


Figure 3.3.2

Then, keeping other parameters constant, we used tansig instead of the standard logsig. We found out that tansig did much better. Here we are showing the graph around the region of the early stopping point near 510 epochs where we can see that Tansig function gives us the best accuracy of slightly above 82%.

The RMSE values with the Tansig function were also slightly better than with the nearest competitor, the Logsig function giving us a result of 0.37626. Comparing the performances of the various functions with respect to the ROC metric also reinforced our belief that Tansig function was the best function for this dataset. Figure 4.3 shows the relevant graphs for this section again showing the best range of epochs from 500 to 800.

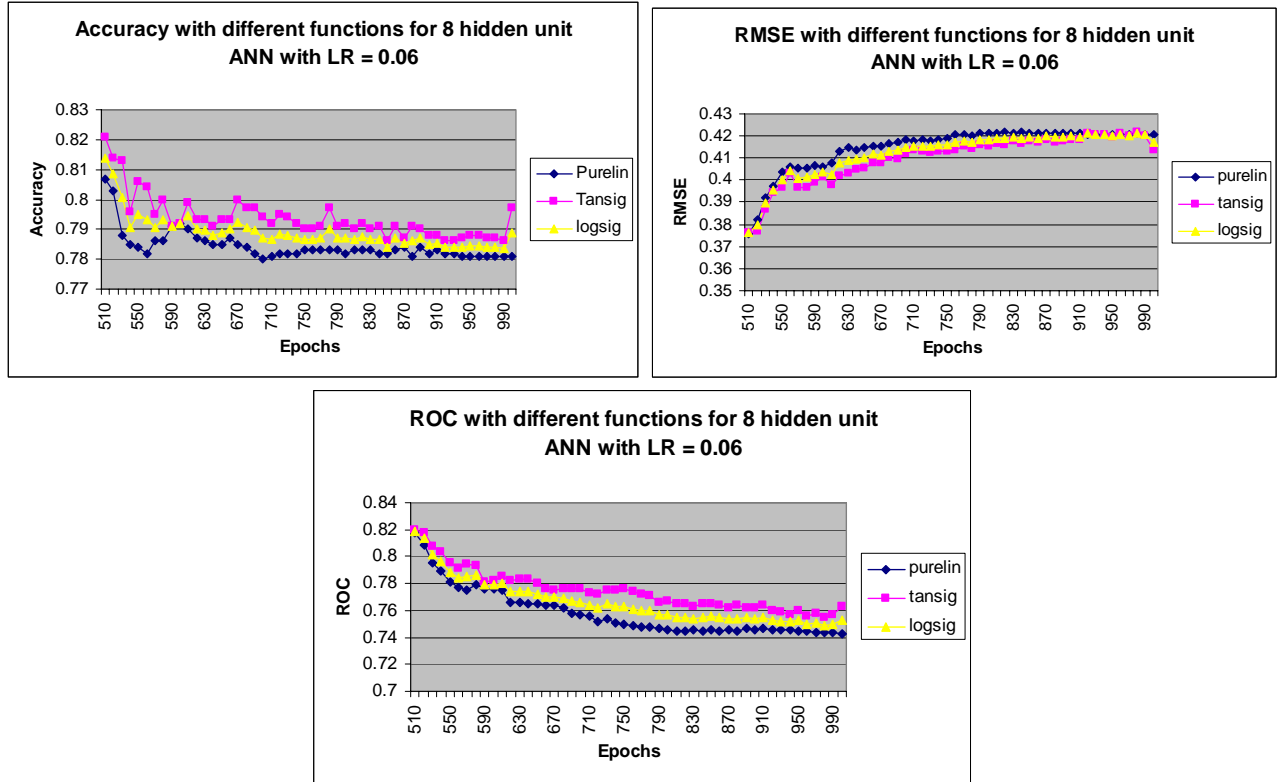
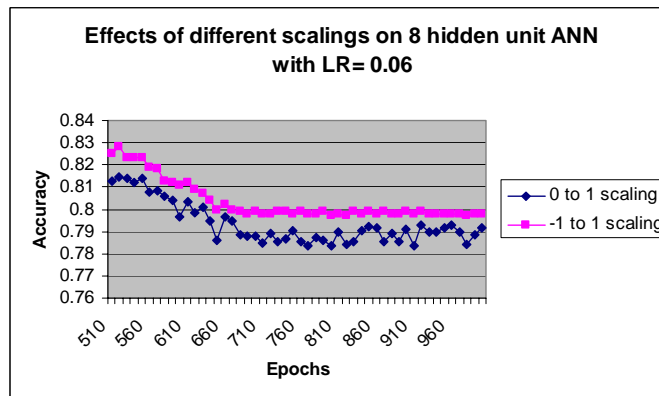


Figure 3.3.3

Next we scaled all the minmaxed inputs to the ANN from -1 to 1 instead of from 0 to 1 . We built up the ANN for this experiment from the earlier settings of that we had gotten i.e. hidden units= 8, learning rate = 0.06

The results from this yielded that -1 to 1 scaling seemed to work better for this dataset. The change in the scaling yielded an accuracy reaching 82.8% near the early stopping point at 520 epochs. The RMSE with the new scaling was significantly lower as well reaching 0.365 and the ROC performance was also superior than the 0 to 1 scaling. Therefore we decided to include this scaling for the further experiments with ANN.

Figure 3.3.4. shows the graphs relating to the experiments with different scalings.



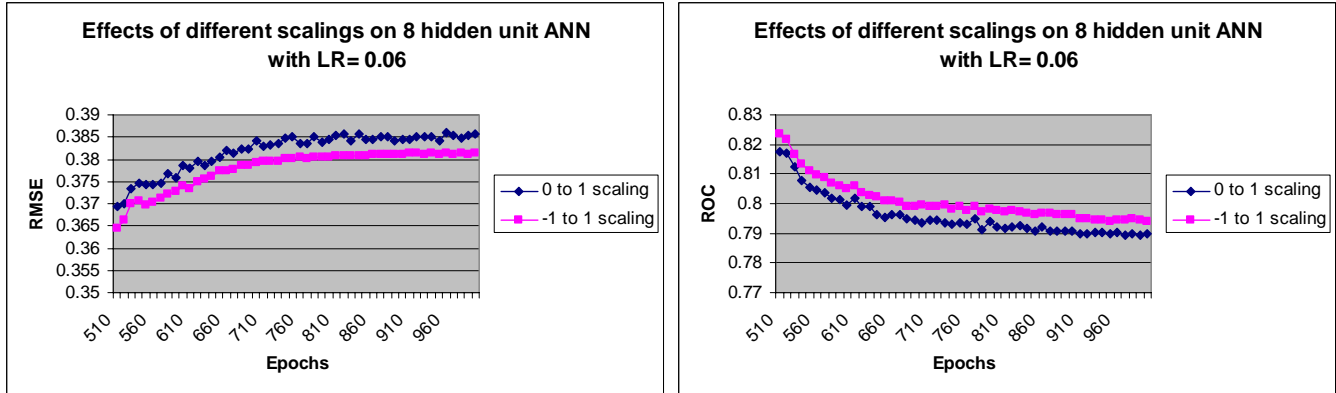


Figure 3.3.4

This brings us to the conclusion of the experiments with the basic ANN. We were quite pleased with the results shown by the ANN especially with the TrainRP method which was saving a lot of time. We knew this would be helpful when we would go on to try bagging and boosting (Though in the end we did not use boosting). Overall the neural net predictions seemed to perform on par with the Decision Trees and was definitely better than KNN and SVMs.

In the later sections we kept experimenting with different hidden units though we kept all the other settings similar to the ones described in this section.

3.4 SVM

We used SVM Light, by Professor Thorsten Joachims to study the performance of SVM on the given test set; we experimented with different Trade Off values at different degrees.

We inputted the target values as -1 and +1 in place of 0 and 1 as in the original set. The predications we achieved were classes of negatives and positives indicating the predicted value. Dividing the train set in 4000 train points and 1000 test points, we first scaled the points by standard deviation which gave us a performance accuracy of 41%, which is way below the baseline.

Therefore, we then scaled the set with MinMax which gave us results above baseline performance. Initially we tested with unforced Trade Off value with svm_learn to analyze the results when the learning tool chose its own value. After running svm_classify to classify each model we achieved from the learning tool on the test set of 1000 points, following accuracies were achieved.

The figure below depicts the accuracies we achieved at different degrees across the unforced trade off values.

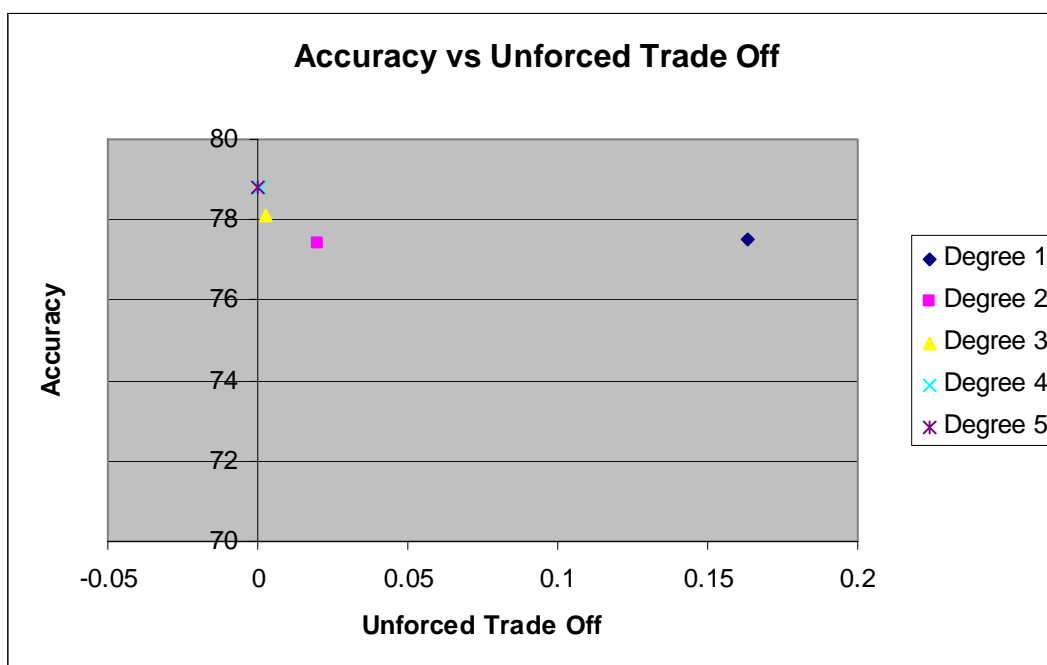


Figure 3.4.1

The performance we achieved was near or just about baseline. Therefore we then tested for different trade off values at different degrees to find the most optimal point for the given set of test points.

Following were the results:

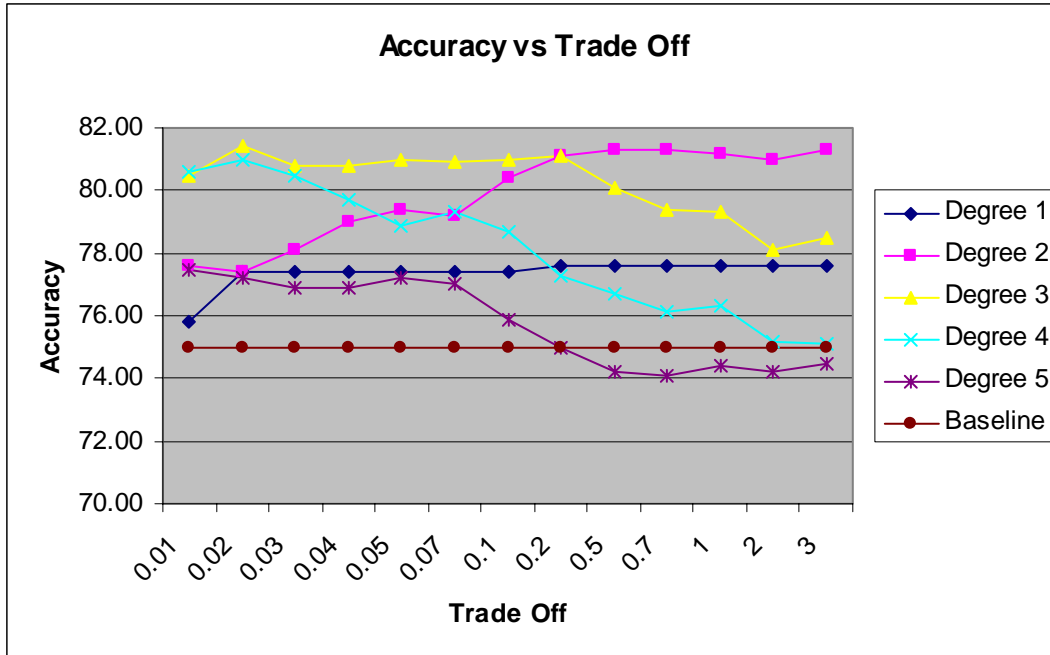


Figure 3.4.2

The best accuracy seemed around degree 2 at 0.2. Though the values dipped and peaked around that interval we investigated to test more to find the best point. As we found degree 2 to be the most accurate, we tested for a range of C values for it:

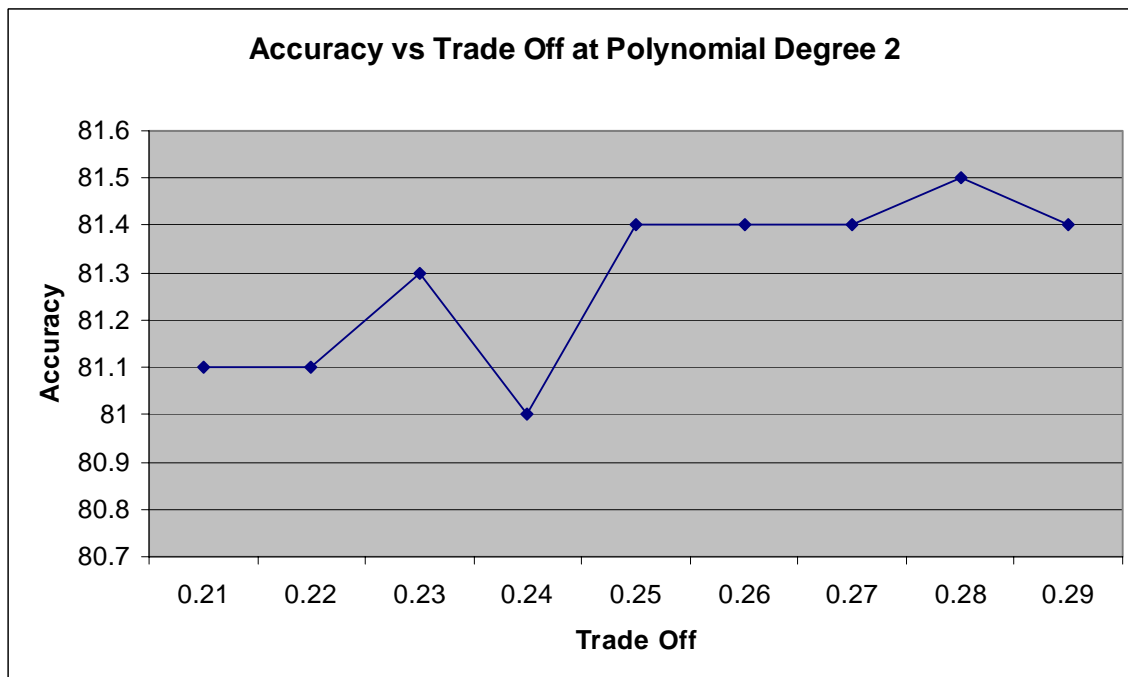


Figure 3.4.3

The best value was found at 0.28 for degree 2 giving an accuracy of 81.50%. We proceeded then with Leave one out cross validation (LOOCV) for the optimal values but found that the results gave a similar accuracy of 81.50%. We found no benefit of doing a LOOCV on the given test.

We wanted to study the effect of increasing Trade Off value at degree 2 to view how it affected the performance, therefore we ran the learning and classification tool on the same 4000 learn set to create different models and then classified the test set with each model to compare their accuracies.

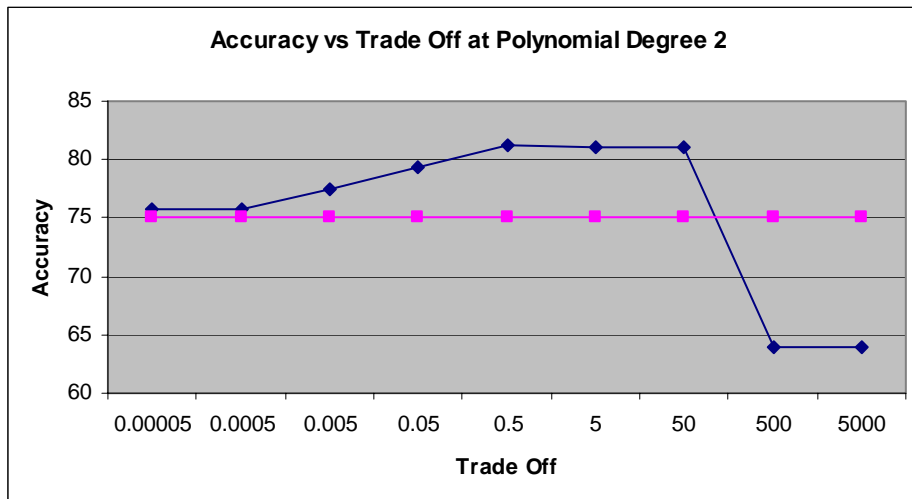


Figure 3.4.4

4 Optimizing performance

4.1 Feature Selection

To do feature selection, we ran 10 random trials of mml trees to find which features it installs and splits on.

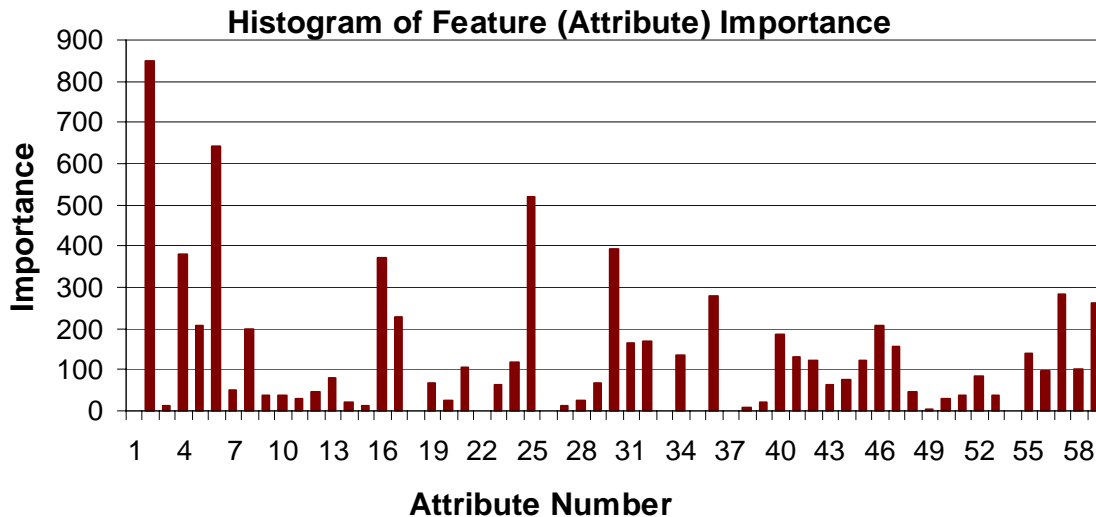


Figure 4.1.1

As we can see from Figure 4.1.1 above, the 2nd attribute had a lot of importance (1st attribute is the class output, and thus its frequency bar is missing on the histogram above).

We used this information of feature importance to rank the attributes and eliminate 7 attributes (the 7 blanks in the histogram above). Interestingly, we noted that most of the attributes that were selected higher up in the table were continuous attributes, and that boolean attributes were ranked towards the end and eliminated.

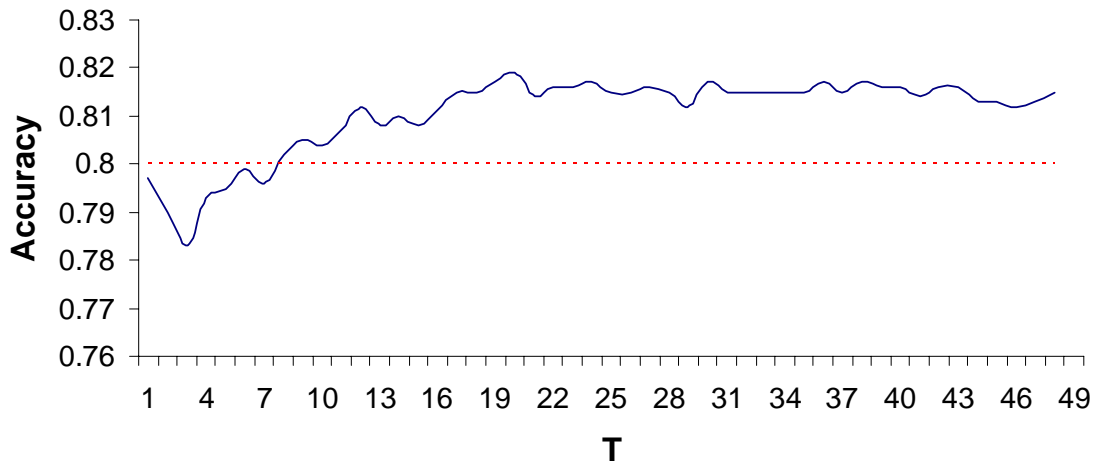
We then systematically eliminated attributes in increasing order of importance and trained ANNs with this reduced feature set. However, we only got inconclusive and noisy results, with no clear indication of which features beyond the 7 initial ones to eliminate for improved performance. Therefore, we decided to then proceed with just a reduced feature set 51 attributes, only eliminating 7 attributes which clearly showed no importance in the MML tree runs. All other attributes were retained with equal weighting, no matter how much ever importance they showed in Figure 4.1.1.

4.2 Bagging Decision Trees

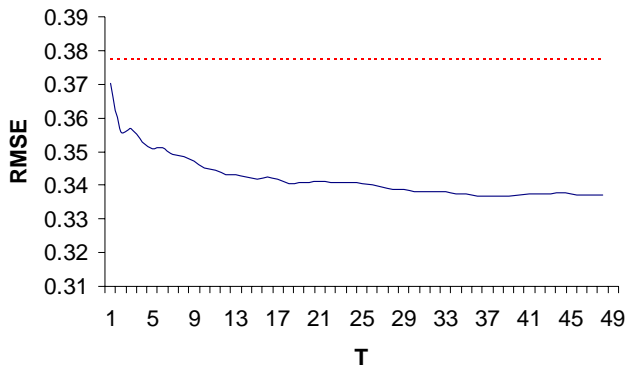
For the purposes of bagging our MML decision trees, we randomly drew 50 bootstrapped samples of size 2000 from 4000 data points and kept a 1000 point test set constant in an attempt to deal with the variance problem in datasets.

Figure 4.2.1 shows the results of bagging. The red dotted line represents the non bagged results obtained in section 3.1.

Accuracy vs T (Number of Bootstrap Samples)



RMSE vs T



ROC Area vs T

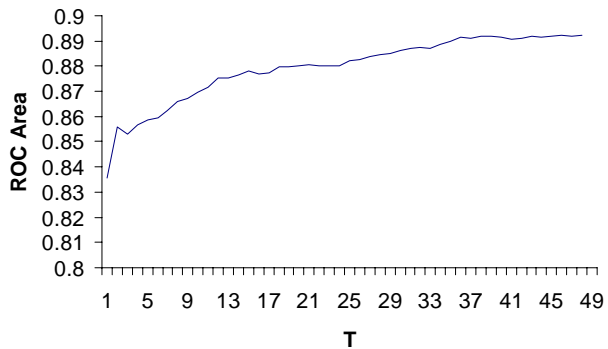


Figure 4.2.1

From figure 4.1 we clearly see an improvement in performance with bagging, and thus decided to bag up to 30 trials, since after that both the RMSE and accuracy curves relatively flatten. Ideally we would like to bag up to 50 or even 100 trials; however, we were also constrained by time.

4.3 Boosting Decision Trees

To implement boosting of the decision trees, we decided to replicate the cases predicted wrong over every trial and add them to the data set and giving it a weighting factor.

Figure 4.3.1 and 4.3.2 show the results of accuracy and RMSE as the number of boosting trials is increased. Both the graphs clearly show improving performance with number of trials.

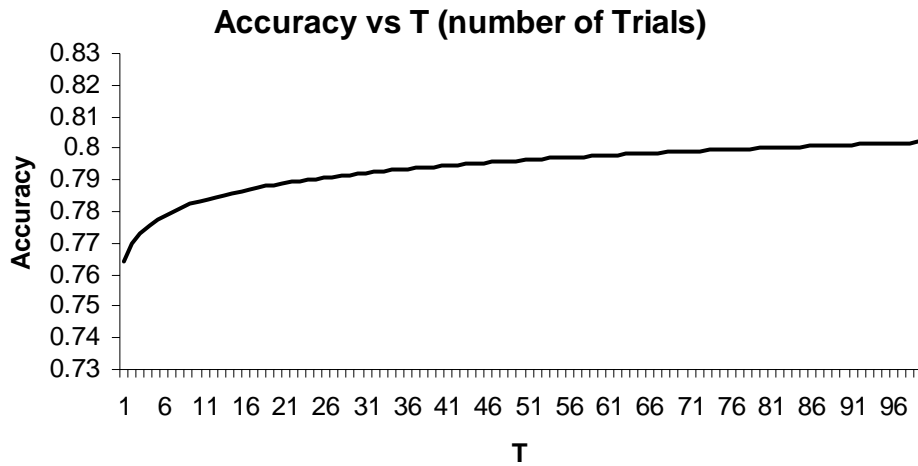


Figure 4.3.1

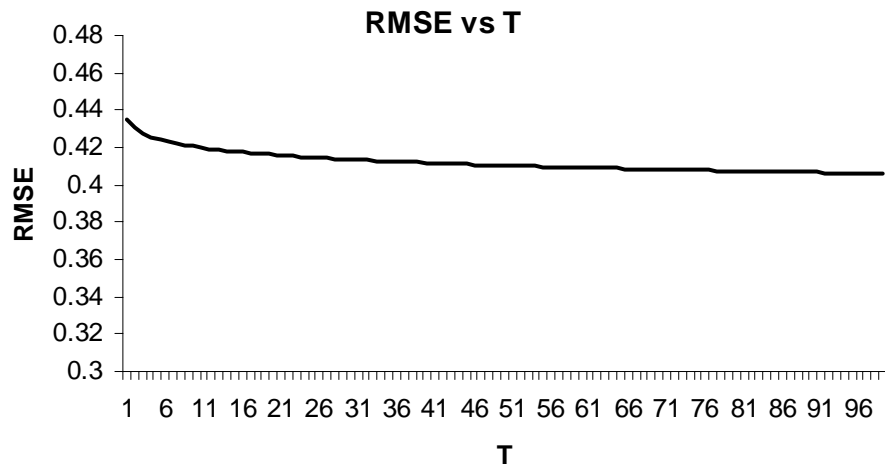


Figure 4.3.2

4.4 Bagged Boosted Decision Trees

We now combine the bagging and boosting of decision trees. We first bagged the decision trees to 30 trials, following which we then boosted the bagged trees to get the figures below. Based on the curves below, we see that it is enough to boost the trees to around 20 trials, beyond which the curves flatten.

Furthermore, we also see that bagged and boosted decision trees perform better than without bagging and boosting, as shown in the accuracy graph below where the red dotted line represents performance without bagging and boosting. Even the RMSE graph is always below the RMSE performance of ~ 0.3800 without bagging and boosting.

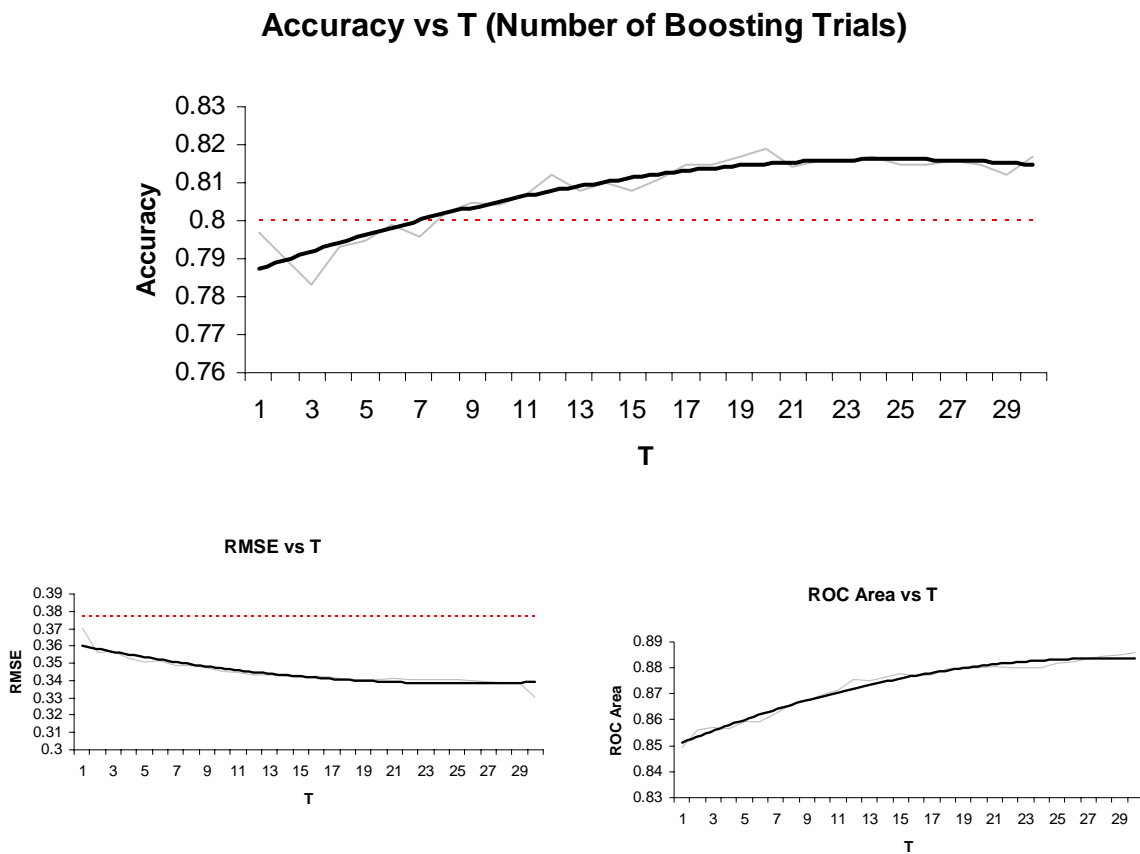
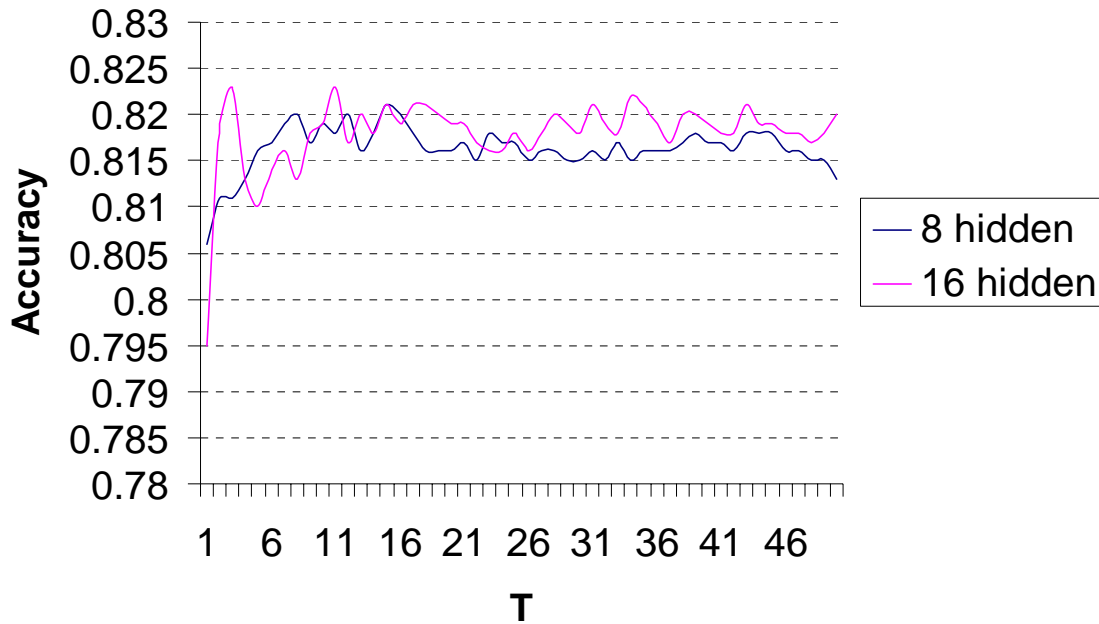


Figure 4.4.1

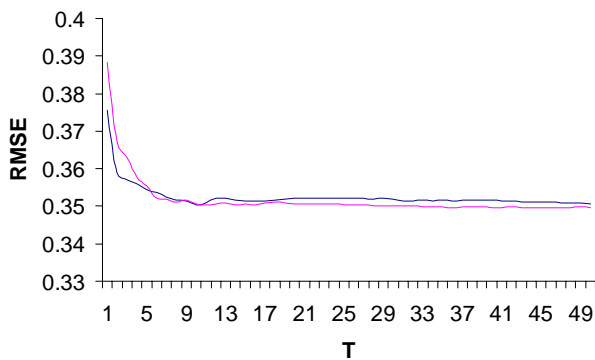
4.5 Bagging ANN

The ANNs were bagged with 50 bootstrapped samples of size 2000 drawn randomly from 4000 data points and the test set was kept at 1000. The results are presented below.

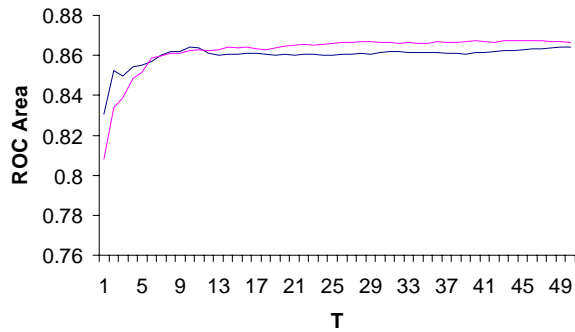
Accuracy vs T



RMSE vs T



ROC Area vs T



The results of bagging ANN showed some improvement for the 8 hidden units though there was a little bit more improvement for the 16 hidden units for all three metrics. The overall accuracies shown in this graph was not for the fully optimized ANN so that is why some of the values will seem a little lower.

4.6 Boosting ANN

As we can see from the graphs below, boosting ANNs only very slightly improved performance when using an un-optimized neural network. We decided not to use boosted ANNs for our final model due to time constraints and relatively little performance gain.

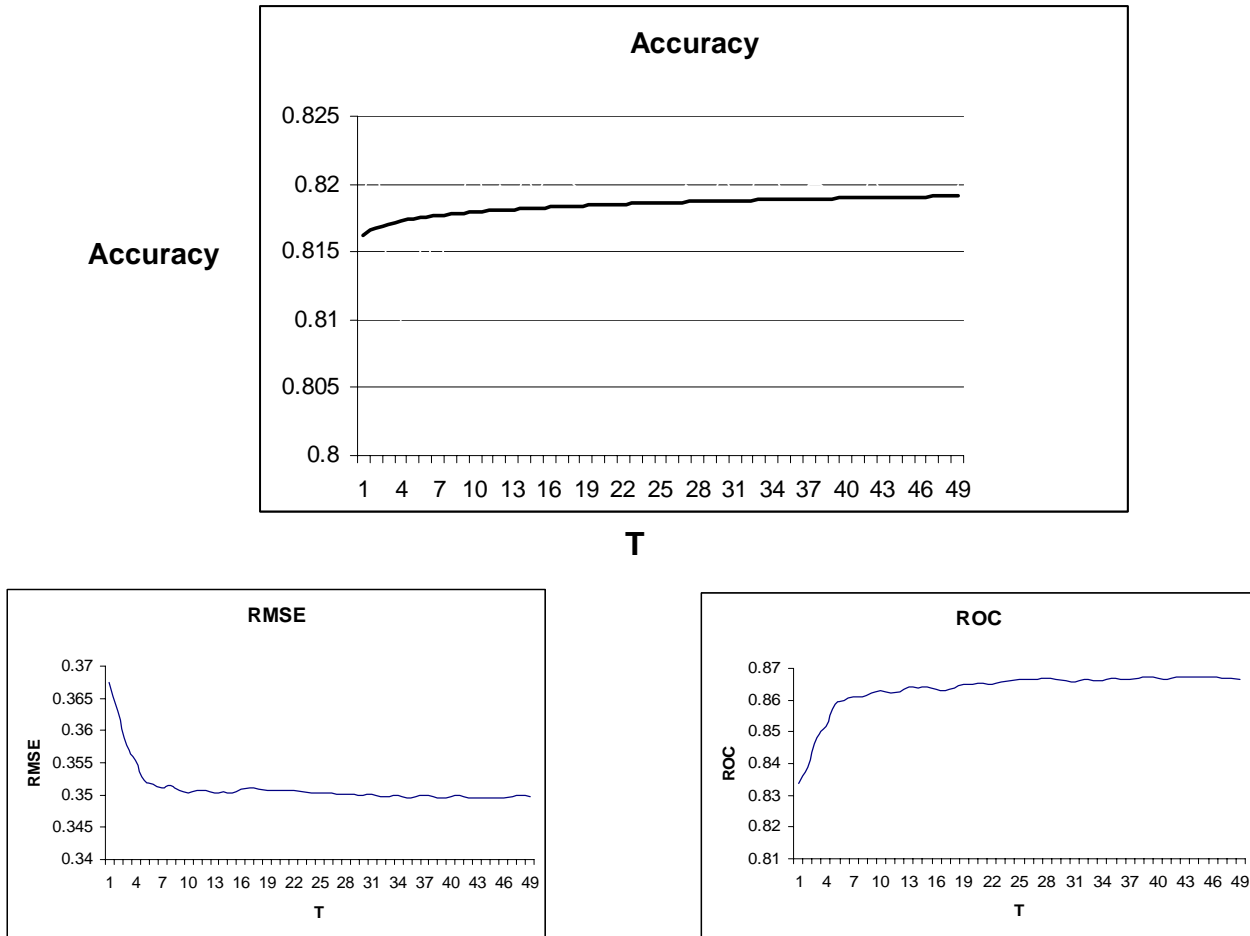


Figure 4.7

4.7 Unweighted and Weighted Combination

Having bagged and boosted decision trees, and bagged neural networks we then proceed to compute the unweighted average of the predictions of the optimally bagged and boosted decision trees with different sizes of bagged neural networks. Table 4.7.1 below summarizes the results:

	Accuracy	RMSE	ROC
4	82.20	0.353	0.856
8	83.80	0.350	0.864
16	83.70	0.349	0.866
4+8	83.00	0.351	0.861
8+16	83.50	0.349	0.866
4+8+16	83.40	0.350	0.864
DT+4	86.50	0.341	0.895
DT+8	87.30	0.339	0.900
DT+16	87.00	0.339	0.902
DT+4+8	84.90	0.341	0.890
DT+8+16	85.40	0.339	0.894
DT+4+16	84.90	0.341	0.890
DT+4+8+16	84.40	0.341	0.886

Table 4.7.1: Combining Predictions of Decision Trees (DTs) and Neural Networks

Note: In the table above, the numbers 4, 8 and 16 in the left-most column represent neural network size. DT represents an optimal bagging and boosting of decision trees as obtained in section 4.5.

As we can see from table 4.7.1, the equally weighted combination of decision trees and neural networks results in a major performance improvement over the predictions of neural networks alone (as shown in the first few lines of the table), or bagged and boosted decision trees alone (as shown in section 4.4). Decision trees combined with a bagged neural network of size 8 produces best results, closely followed by combining the same with a neural network of size 16.

Following this, we then tried weighted combination of optimally bagged and boosted decision trees with neural networks of size 8. Table 4.7.2 below summarizes the results. The left-most column shows the weighting schemes. For example, 0.4, 0.6 would suggest a weight of 0.4 on the predictions of the decision trees and of 0.6 on that of the neural networks.

We see that with a weighting combination of 0.7, 0.3, i.e. giving more weight to the decision tree predictions, we reach best accuracy of 88.6%. An ad-hoc trial of the same but this time using a neural network of size 16 resulted in an even more improved performance (though very slightly) in terms of accuracy and ROC Area.

	Accuracy	RMSE	ROC
DT+8			
0.4, 0.6	85.6		
0.6, 0.4	88.2		
0.65, 0.35	88.4		
0.7, 0.3	88.6	0.343	0.894
0.75, 0.25	88.2		
0.8, 0.2	87.9		
DT+16			
0.7, 0.3	88.7	0.343	0.897

Table 4.7.2: Weighted Combination of Predictions

Having run out of time, we finally submitted predictions that were a 0.7, 0.3 weighted combination of the predictions from optimally bagged and boosted decision trees and bagged neural networks with 16 hidden units. This optimizes accuracy by achieving 88.7% accuracy on the test set.

5 Final Selection and Conclusions

The final selection that we used was the weighted average between the predictions of ANNs with 16 hidden units and Decision Trees, giving a higher weight of 0.7 to the decision tree predictions. The ANNs were bagged across 50 models and had a learning rate of 0.06. The training method that was used was trainrp and we ran the training till the early stopping point of 500 epochs. The decision trees were mml trees with bagging and boosting. We bagged with 30 bootstrapped samples and boosted each of these bagged samples 20 times giving more weight to the errors progressively. This yielded us a performance of 88.70% accuracy on the final test set and RMSE value of 0.343. The ROC value was 0.897 and the threshold, which we also submitted, was 0.432.

Weighted model combination has yielded much better results on the test set than any one model alone and has proved to be a very powerful technique. Indeed through these efforts we have managed to increase prediction performance on our test sets by ~14% above baseline in terms of accuracy, which is much more than what we had been able to achieve on the homework assignments.

The results we submitted were only optimized for accuracy. It would have been prudent to also optimize the predictions based on RMSE and ROC Area, which would require us to repeat the experiments of table 4.1, however, with different weightings (such as in table 4.2) instead of the equal weightings of table 4.1. However, due to lack of time, we could not do this.

Since we achieved an accuracy of 88.70% on our test set, we expect to achieve a similar accuracy or slightly lower on the final test set of 20000 points. ROC and RMSE figures are expected to be around 0.343 and 0.897 too respectively.

We ran an ad-hoc test where we boosted decision trees to 100 trials and then subsequently bagged this to 50 trials. We then combined its (unweighted) predictions with those of bagged neural networks of size 4, 8 and 16 (DT+4+8+16), and got an accuracy of 87.40%. A similar combination, but bagging the decision tree to only 30 trials and boosting to 20 resulted in just 84.40% accuracy. Thus, given more time, we are certain that we could have achieved better accuracy on our test set by bagging and boosting more and trying various weightings for the final model combination and averaging.

Some of the other things we could've investigated were to see the performance of kNN with feature selection and combining these predictions into our final model.

Contributions

Mayank was responsible for SVMs and writing the scripts for computing the performance of the bagging and boosting experiments and those for combining models for a final averaged prediction. Venkat and Aakash were responsible for ANNs. Aakash also compiled a major part of this report and generated the graphs. Venkat also implemented the bagging and boosting code and ran the decision tree experiments. Biz worked on kNN, creating the bootstrap samples, running some of the ANN experiments, creating the graphs for some sections of the report. As a group we charted out the plan before starting, analyzed intermediate results, decided which model was best at the end and wrote parts of this report.

Appendix A

Resilient Backpropagation (trainrp)

Resilient backpropagation (`trainrp`) is a neural network training algorithm provided in the neural network toolbox in MATLAB. This method was used since it converges much faster.

Resilient backpropagation speeds up convergence of feedforward networks using sigmoid activations for their hidden units by considering only the sign of the partial derivatives to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. Sigmoid functions are characterized by the fact that their slope must approach zero as the input gets large. This causes a problem when using steepest descent to train a multilayer network with sigmoid functions, since the gradient can have a very small magnitude; and therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.