Classification Problem 1) Salary Prediction

Abstract:

The goal of the first classification problem is to predict whether individuals have an annual salary above or below $50,000 (binary prediction). The train and test datasets are on an individual-record level, and contain various attribute information such as education level, hours worked per week, marital status, age etc.

The train and test set each have a split between below / above $50K of approximately 76% / 24%. The more precise goal in developing the machine learning models on this dataset is to improve upon the 76% number, since guessing "less than $50K" for each test data point would result in a 76% accuracy level. The models used in developing predictions include decision trees, boosting, support vector machines, k-nearest neighbor, and neural networks.

The layout of this paper include a section comparing the overall performance of the algorithms used, followed by a section of analysis for each algorithm used, and an overall summary discussing various issues with the algorithms and data, as well as possible improvements in future analyses.

This dataset and problem were chosen because they demonstrate various strengths and weaknesses of the supervised learning algorithms utilized in the analysis, while improving the performance of random guessing. The dataset has a mixture of categorical and numerical data, so it also provides a good example of how to transform the categorical data in order to implement some of the algorithms (e.g. neural nets).

Note: Throughout the analysis for this problem, any mention of a "positive" label means an individual with greater than $50K as a salary; a "negative" label implies less than $50K as a salary.

Citation for dataset: **Becker, Barry. (1996). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/Adult] Irvine, CA: University of California, School of Information and Computer Science.**

Overall Algorithm Comparisons:

Model Performance:

In investigating model performance, several factors were examined—including accuracy, AUC, precision, recall, and specificity. These were measured by implementing the models on the test sets. Computational efficiency of the algorithms was measured as well by measuring the length of time for each algorithm to learn the training data. Except for K-NN, clock time was measured by the length of training time in seconds for each of the algorithms to run. For K-NN, since no training was done, the clock time was measured by the length of time it took for the algorithm to classify the entire testing data set. The results in the table below are from the runs of the algorithms showing the best accuracy, with the other information corresponding to those implementations. The K value in the K-NN algorithm's metrics is 21, as that was selected as the optimal value for K (more details found in the K-NN section).

The boosting and decision tree methods showed the best performance overall across most of the metrics, with further details later in the paper. One of the most significant notes in examining the metrics is the lower precision and recall numbers across the algorithms in comparison to specificity i.e. *the algorithms were able to predict true negatives much better than true positives.* One potential reason for this is that the training set contained three times as many data points for individuals labeled negative than positive. If more training data were available containing positive labels, this could lead to a potential improvement in the model. With no other training points available, stacking of several of the algorithms could provide a potential improvement to the performance of the models. Another potential issue with the algorithms' performance levels is correlation between some of the variables. A

future improvement could involve using principle components analysis, or another technique, to transform the data and aid against this issue.

Supervised Learning Metrics Table:

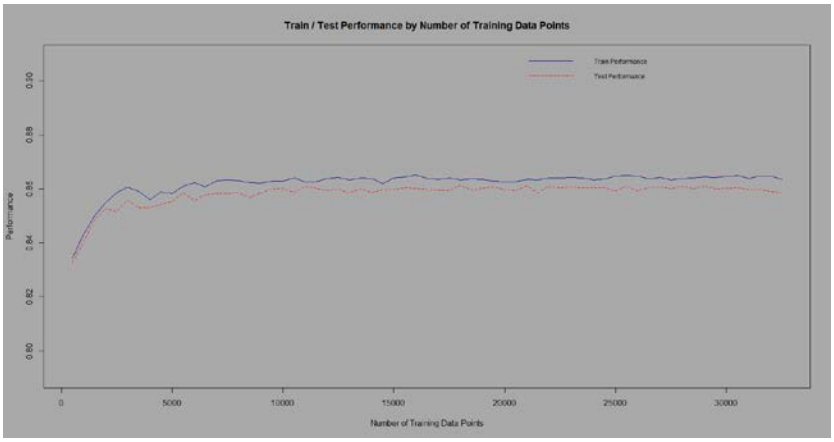
Algorithm	Testing Accuracy	AUC	Clock Runtime	Precision	Recall (Sensitivity)	Specificity	Training Accuracy
Decision Tree	85.8%	77.4%	6.18	74.1%	61.5%	93.4%	88.1%
Boosting	86.1%	76.9%	90.68	76.2%	59.5%	94.3%	86.4%
K-NN	84.5%	75.2%	1097.12	71.3%	57.5%	92.8%	100.0%
Neural Network	85.3%	76.8%	62.89	72.5%	60.7%	92.9%	85.4%
Support Vector Machine	85.3%	75.7%	260.36	74.7%	57.4%	94.0%	85.5%

Exhibits contrasting the train and test accuracy performances by each of the algorithms are shown below. For the decision tree implementation, the training performance has a general monotonic increase as the training size increases. Following the initial increase in the performance between 500 and 5000 training samples, the test set has a flatter performance across increasing numbers of training samples. The neural network exhibit shows a sharp initial increase in training and testing performance, but both remain relatively flat after around 2000 training data points. This seems to imply that the neural net model is having less of an issue with overfitting than that of the decision tree, which corresponds with the nature of both algorithms. More aggressive pruning could potentially improve the decision tree algorithm in this regard. Also, the neural net performance graph shows that this algorithm is able to generalize to the testing set more quickly—with fewer training points—than the other algorithms.

The boosting train / test performance graph is very similar to the neural network graph; this similarity probably due to both of these algorithms better handling the overfitting issue than the decision tree implementation. The K-NN test performance shows the tightest range of any of the algorithms as the training size increases, implying that greater number of training points do not impact the test performance as much as the other algorithms. This could be caused the same violation of the assumption that the nearest neighbors to a testing points are similar that testing point. Greater numbers of training points would not necessarily rectify this.The SVM test / train performance is very flat after an initial increase in performance at around 3000 training data points.

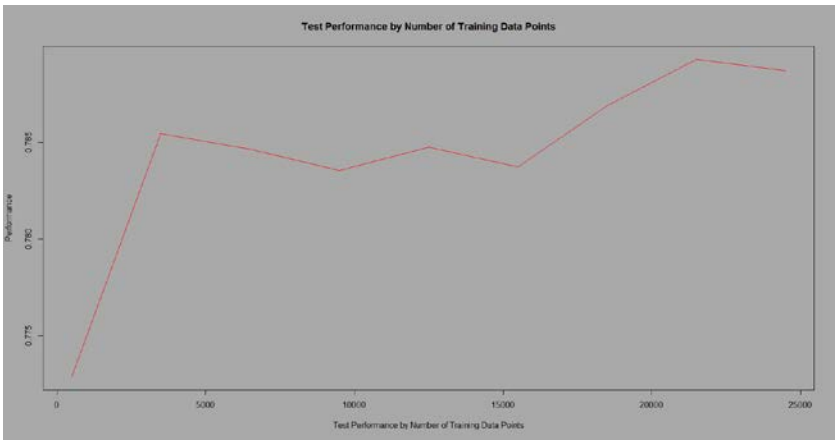
Decision Tree:

Boosting:

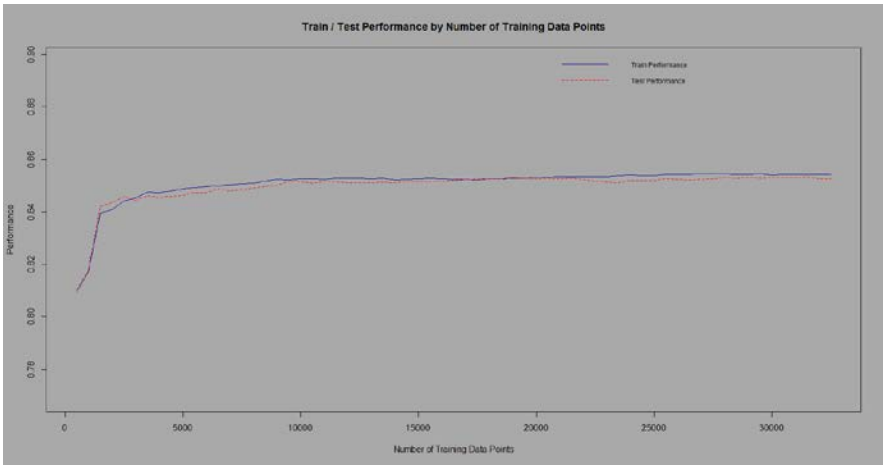


K-NN:

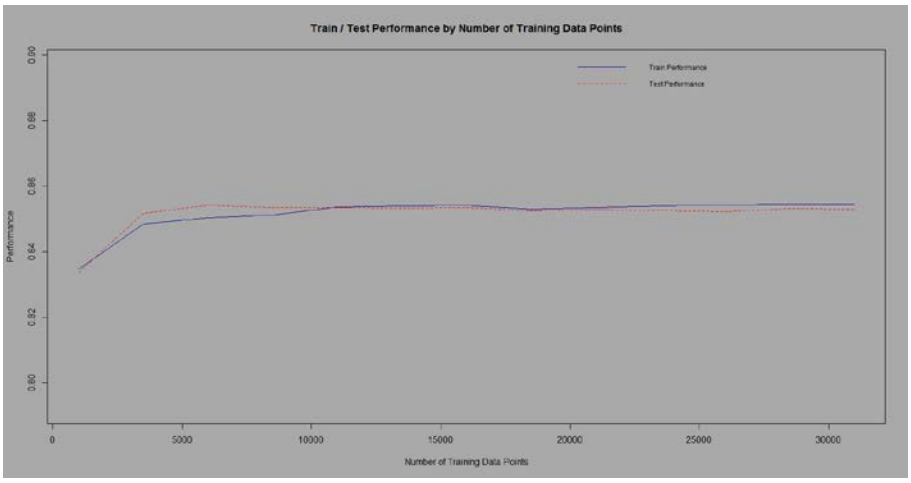
Since no dataset is trained for K-NN, only the test performance is shown. Also, since the algorithm runtime is much slower for this algorithm, 1-NN was used, rather than the optimized 21-NN used for the full training set.



Neural Network:



Support Vector Machine:

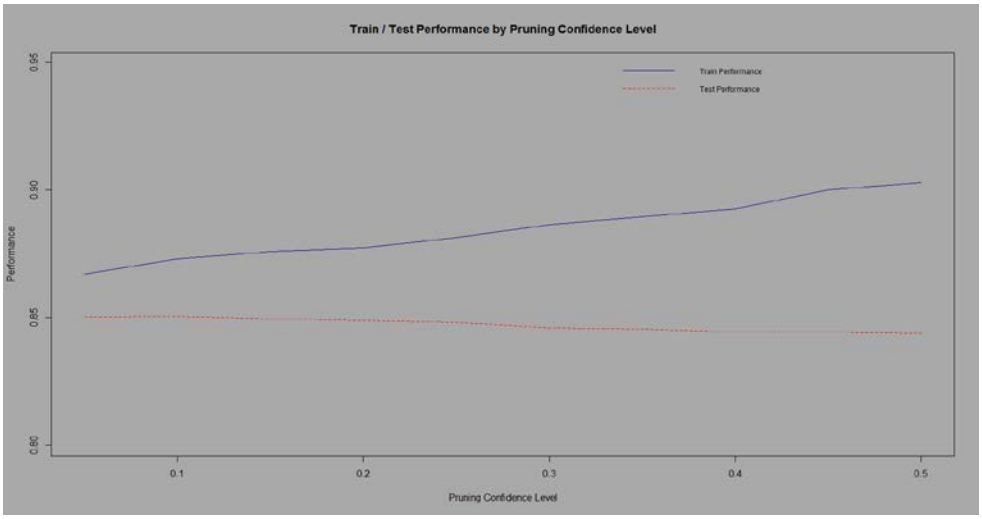


Decision Tree Approach:

In this approach, the C4.5 algorithm with pruning, and information gain as the split criteria was used with the following parameters: Minimum instances per leaf: 2; Confidence Threshold for Pruning: 0.25.

Varying the Error Rate Confidence Level for Pruning

Our next analysis compares the train and test performances by varying the error rate confidence level that is input into the C4.5 algorithm. The levels tested come from the set CL ∈ {0.05, 0.10, 0.15…0.50}. The result is a divergence of the train performance from the test performance as the confidence level increases. This demonstrates the effect of pruning on the trained decision tree model i.e. as the confidence level increases, pruning is less likely—thereby increasing the training performance, but lessening the test performance. The same is true in reverse for this data: the amount of pruning increases as the confidence level decreases and the training performance decreases, while the testing performance increases (see the graph in below).



Pruning versus Not Pruning

The purpose of this step in the analysis is to examine how much of an effect pruning has on the given training set.

Training (w/ Pruning)	Training (w/o Pruning)	Testing (w/ Pruning)	Testing (w/o Pruning)
88.1%	91.7%	85.8%	84.3%

Feature Selection

In this section, we use several approaches to examine training and testing performance against feature selection. First, we implement C4.5 on the training data using only one feature at a time (see results below). In a second approach, we train the decision tree classifier using only one single attribute at a time. With this method, note that approximately 75.9% of the training set and 76.4% of the test set have an income below $50K. With this in mind, nine of the fourteen predictors classified the entire training and test sets into the "≤ 50K" category when used alone. Of note, when used alone, the capital gain attribute had the best performance in terms of pure accuracy on both datasets—classifying over 81% of the test set correctly.

Number of Attributes	Train	Test
1	75.9%	76.4%
2	76.6%	77.0%
3	76.6%	77.0%
4	79.9%	79.5%
5	79.9%	79.5%
6	83.5%	83.1%
7	84.9%	83.6%
8	85.1%	83.3%
9	85.2%	83.3%
10	85.1%	83.6%
11	87.0%	85.0%
12	87.7%	85.0%
13	88.3%	84.6%
14	88.1%	84.8%

Attribute	Train	Test
age	75.9%	76.4%
capital_gain	**80.9%**	**81.3%**
capital_loss	78.1%	75.9%
education	78.0%	78.0%
education_num	78.0%	78.0%
fnlwgt	75.9%	76.4%
hours_per_week	75.9%	76.4%
marital_status	75.9%	76.4%
native_country	75.9%	76.4%
occupation	75.9%	76.4%
race	75.9%	76.4%
relationship	75.9%	76.4%
sex	75.9%	76.4%
workclass	76.3%	76.7%

The third approach involves starting with all fourteen attributes and testing which ones have the least impact on the model performance when left out of building the classifier. The initial results are below.

Feature	Train	Test
capital_gain	86.2%	83.2%
education_num	87.8%	83.2%
native_country	88.3%	84.6%
occupation	87.2%	84.6%
relationship	88.2%	84.8%
education	88.1%	84.8%
hours_per_week	87.6%	84.9%

Feature	Train	Test
sex	88.1%	84.9%
race	88.2%	84.9%
workclass	87.9%	84.9%
age	87.0%	85.0%
fnlwgt	87.8%	85.0%
capital_loss	87.6%	85.1%
marital_status	87.8%	85.3%

Boosting Approach:

In this approach, the AdaBoost algorithm was implemented using variations in several of its key parameters. Overall, most of the implementations of the AdaBoost method give an accuracy of around 86%, slightly better than

the decision tree method. The following are the results of the discrete boosting, Real Boost, and Gentle Boost versions of the algorithm using the parameters: Number of Boosting Iterations: 50; Max tree depth of any node = 30; Learning parameter (nu) = 0.1.

Algorithm Type / Metric	Accuracy	AUC
Discrete	86.0%	76.8%
Real	86.0%	77.0%
Gentle	85.8%	75.5%

Performing sensitivity analysis on the learning parameter to test how performance varies yields the following:

Learning Rate	Accuracy	AUC
.01	85.8%	75.5%
.05	85.9%	76.7%
.1	86.0%	76.9%
.12	86.0%	76.8%
.15	85.9%	76.6%

Next, we analyze the effect of changing the maximum tree depth in the boosting of the decision trees.

Max Tree Depth	Accuracy	AUC
5	86.1%	77.0%
10	86.1%	77.0%
15	86.1%	76.8%
20	85.8%	77.4%
25	85.8%	76.9%
30	86.0%	76.8%
35	86.0%	76.9%
40	85.8%	76.5%
45	85.9%	76.6%
50	86.0%	76.8%

Overall the effects of changing various parameters had little effect on shifting the boosting performance in either direction. It is possible that random noise in the dataset could be preventing the boosting classifier from improving much beyond the 86% accuracy level. An additional possible solution might be to increase the size of the training set (if possible).

K-NN Approach:

In this approach, an extension the K-NN algorithm is used, allowing for implementation on this dataset with categorical variables. For the categorical features, the phi coefficient (mean square contingency coefficient) is used to measure the similarity—or distance, between the different classes.

The overall performance tends to be slightly worse when using the K-NN algorithm versus the decision tree or boosting approaches. Testing several values for k—3, 5, 11, 21, 25—21 comes out as the optimal choice. This is was selected by performing three cross-validations and selecting the k-value with the smallest error rate. Feature selection was done using a permutation test with an improvement cutoff. Using k = 21 gives a test accuracy of 84.5% and an AUC of 75.2%.

One of the core assumptions of this algorithm is that the data points within a neighborhood of a particular testing point are similar to each other. This is not a perfect assumption, and in this scenario, could be lowering the performance of the model. Also, the inclusion of attributes that are not predictive of the target label could be lowering performance as well, since this fact has a greater impact on the K-NN algorithm than boosting or decision trees. More aggressive feature selection could help increase performance, by ignoring the unimportant attributes.

Neural Network Approach:

Here, resilient backpropagation algorithm with weight backtracking is implemented with the following key parameters:

Number of Hidden Layers	**1**
Threshold	0.01
Activation Function	Logistic
Max Steps in Training Process	100,000
Learning Rate Bounds	Lower: 0.5 Upper: 1.2

Overall, the neural network performed similarly to the other decision tree and boosting algorithms, and its testing performance was almost the same as its training performance.

Performing sensitivity analysis on the threshold level yields the following**:**

Threshold	**Accuracy**	**AUC**
0.01	85.3%	76.8%
10	85.2%	76.5%
100	83.3%	70.7%
1000	76.4%	50.0%

Adjusting the threshold does not have much of an impact until it is thrown far out (1000 +)—implying the training error rate is not affected at the different steps of the algorithm is not largely affected until the threshold is beyond a third order of magnitude. Adjusting other parameters when the threshold limit is this high would probably increase the accuracy as the threshold is adjusted during the steps of the algorithm.

Support Vector Machine Approach:

Here, the initial run of a support vector machine algorithm used a set of kernel families—radial, linear, polynomial, and sigmoid.

Kernel	**Accuracy**	**AUC**
Radial	85.3%	75.7%
Linear	85.3%	76.1%
Polynomial	84.4%	71.7%
Sigmoid	85.1%	75.3%

Next, looking at the polynomial kernel in more detail, there is a significant drop in performance after the degree of the polynomial increases above two, especially when judged by the AUC metric.

Degree	**Accuracy**	**AUC**
2	84.4%	71.7%
3	78.9%	56.3%
4	78.0%	54.2%
5	77.5%	52.6%

Classification Problem 2) Workout Analysis

Abstract:

This problem is designed to analyze the exercise movement an individual is performing based off various body movements, angles, etc. Each record in the datasets corresponds to an individual performing a Unilateral Dumbbell Biceps curl according to a specification—Class A) correctly performing the exercise; Class B) putting elbows out in front; Class C) lifting the dumbbell halfway; Class D) lowering the dumbbell halfway; Class E) putting hips out in front. Class A is the correct movement, while the other classes are common mistakes in the exercise. The goal of this classification problem is to use data about the various angles of a person's arms, legs, body etc. in order to predict what class he has actually performed. The models used in developing predictions include decision trees, boosting, support vector machines, k-nearest neighbor, and neural networks.

This problem was chosen because it has interesting and very large differences within each algorithm when common parameters are adjusted i.e. the kernels used in the SVM, the value of K in K-NN etc. Also, this classification problem has five possible labels, instead of the usual binary examples. The citation for the dataset is below:

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

Overall Algorithm Comparisons:

Model Performance:

In investigating model performance, several factors were examined—including accuracy and AUC by the models on the test sets, and the efficiency at which each algorithm is able to learn the training data. Like problem 1), clock time was measured by the length of training time in seconds for each of the algorithms to run, with the exception of K-NN—which was measured by the length of time it took for the algorithm to classify the entire testing data set.

The results in the table below are from the runs of the algorithms showing the best accuracy, with the other information corresponding to those implementations. For the K-NN metrics, K = 1, as that was selected as the optimal value for K (more details found in the K-NN section). The below table shows the best accuracy performance of each algorithm implementation, along with associated AUC and wall clock time performance.

Supervised Learning Metrics Table:

Overall	Testing Accuracy	AUC	Clock Time / Training Time	Training Accuracy
Decision Tree	96.6%	98.3%	7.38	99.4%
Boosting	99.6%	99.8%	95.09	99.9%
K-NN	95.8%	98.4%	153.69	100.0%
Neural Network	69.7%	79.4%	865.85	73.6%

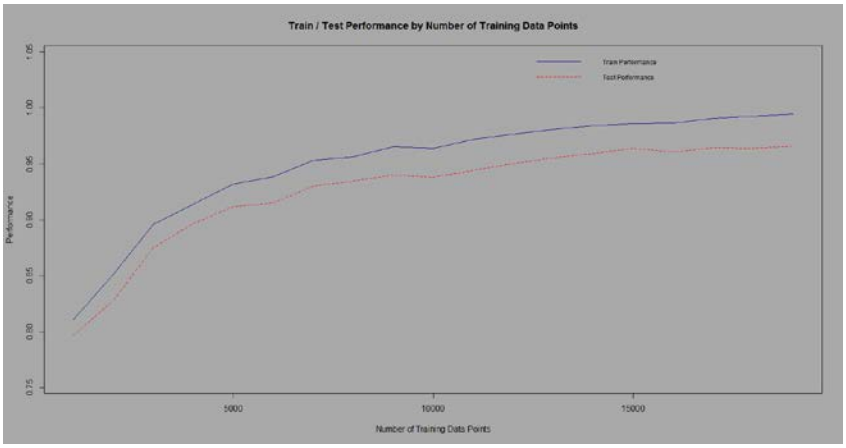
Support Vector Machine	95.7%	98.5%	168.05	95.8%

The neural network approach is the most notable, as it had a significantly lower performance rate than the other methods used. More detail on this will be provided in neural network section later in this analysis. Exhibits contrasting the train and test accuracy performances by each of the algorithms are shown below.

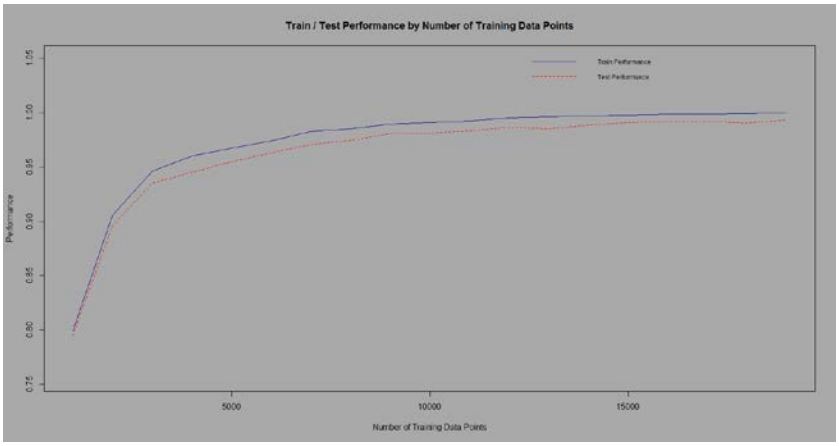
The decision tree and boosting exhibits follow similar monotonically increasing patterns, with the boosting test performance asymptotically approaching the training performance faster than the decision tree doing so with its test / train performance levels.

Unlike problem 1), the K-NN algorithm has a more drastic difference in test performance when the size of the training set is lower. This could mean that there is more random noise in the smaller sets of the data, especially given that K = 1 is used for the algorithm. More training data equates to more confidence in the training set.

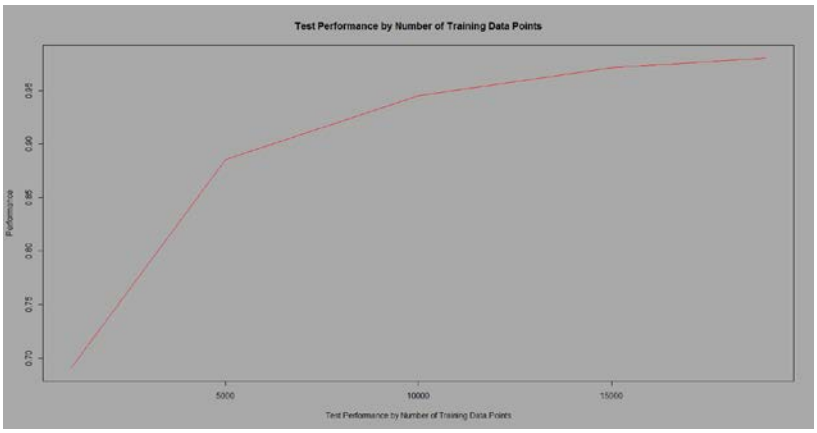
Decision Tree:



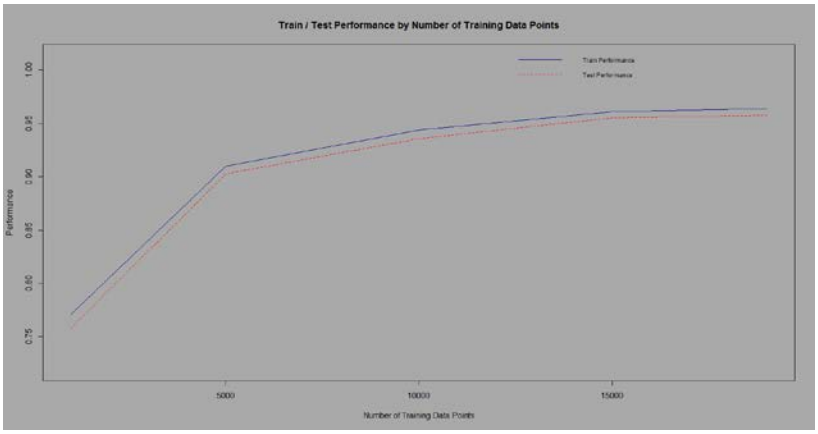
Boosting:



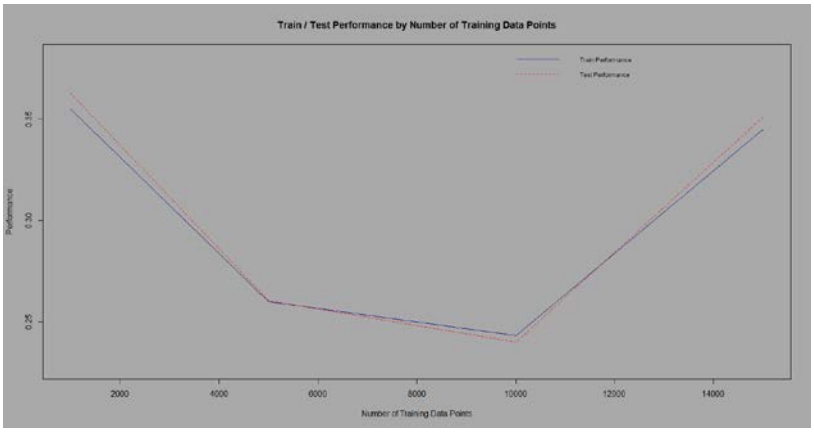
KNN:



SVM:



Neural Network:



Decision Tree Approach:

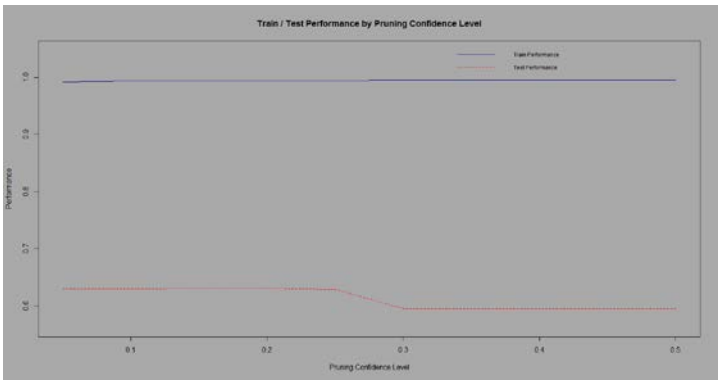
In this approach, the C4.5 algorithm was used. The algorithm shows a test accuracy at 96.6% versus a train accuracy of 99.4%. Using reduced error pruning, and varying number of folds for cross-validation, the following results were achieved:

Number of Folds	Train Accuracy	Train AUC	Test Accuracy	Test AUC

3	97.4%	98.6%	95.4%	97.6%
5	97.4%	98.6%	95.3%	97.7%
7	97.0%	98.5%	94.7%	97.3%
10	96.6%	98.4%	94.6%	97.3%

Pruning Confidence Level:

Looking at train / test accuracy as the pruning confidence level is adjusted yields flat results:



Support Vector Machine Approach:

The initial run of the SVM algorithm used a radial kernel with a unity cost parameter and resulted in a test accuracy of about 95.5%; there were 8185 support vectors.

Investigating by switching out kernel functions, we get the following results:

Kernel	**Accuracy**	**AUC**	**Number of Support Vectors**
Radial	95.7%	98.5%	8185
Linear	78.8%	85.6%	10,170
Polynomial (degree = 3)	94.9%	98.1%	8294
Sigmoid	**36.5%**	**62.7%**	**13,778**

The sigmoid kernel stands out, as it has by far the lowest accuracy and AUC metrics. Adjusting the cost parameter of the sigmoid kernel has varying results (the cost parameters below are shown by orders of magnitude of 10).

Cost Parameter	**Accuracy**	**AUC**
-3	28.5%	50.0%
-2	**46.8%**	**65.9%**
-1	**45.1%**	**66.1%**
0	36.5%	62.7%
1	36.6%	63.1%
2	37.4%	63.1%
3	36.6%	63.6%

The cost parameters showing the best performance are 0.01 and 0.1. Cross-validation was used in an attempt to increase performance, but the results were approximately the same.

K-NN Approach:

The K-NN approach had similar overall performance to decision tree and SVM implementations for k = 1. Running the algorithm by varying choices for *k*, we obtain the following results:

K Value	Accuracy	AUC
1	95.6%	98.4%
3	95.1%	98.1%
5	94.5%	97.9%
7	93.7%	97.4%
15	91.2%	95.8%
21	89.0%	94.4%

The results tend toward lower accuracy and AUC as the value of k increases, implying that the test data points become dissimilar to the training points in its neighborhood relatively quickly as the number of neighbors increases. The greater values of k tend more toward violating a core assumption of the K-NN algorithm—that data points are similar to the ones closest in distance—albeit, an 89% accuracy at 21 neighbors is still much better than a 20% random guess accuracy.

Neural Network Approach:

This approach used a neural network with one hidden layer, and a sigmoid activation function. Overall, this approach was less successful when tested than the decision tree or SVM algorithms. This could be due to the more complex nature of the algorithm, and number of parameters that need to be tuned. The number of steps in the algorithm was 37,371. **Note:** Similar to the sigmoid-kernel-variant of the SVM algorithm, this neural network using a sigmoid-based activation function also performs poorly. This could be that the sigmoid function simply does not fit well with the training data, unlike other functions (logistic, polynomial etc.).

Training a neural network classifier with one hidden layer, we vary the number of inputs in that layer with the following results (primary ones of note are highlighted):

Number of Inputs in Hidden Layer	Test Accuracy	Test AUC
10	**29.8%**	**60.1%**
20	36.7%	57.7%
30	52.8%	72.4%
40	48.1%	69.7%
50	**69.7%**	**79.4%**
60	65.8%	75.0%
70	67.5%	81.2%

Accuracy has an interesting trend across the number of inputs in the hidden layer; at 10 inputs, the network performs rather poorly, but has over twice the accuracy at 50 inputs. Possible ways of improving the performance would be to experiment with the activation function, the learning parameter, and number of hidden layers.

Boosting Approach:

The boosting implementation had the best accuracy of any of the algorithms discussed in this classification problem. The initial run of the algorithm obtained a test accuracy of 99.6%. The implementation used the AdaBoost M1 algorithm with 10 iterations. The C4.5-implented decision trees from the decision tree implementation were used as the base weak learners for boosting. The performance is most likely due to the nature of the boosting algorithm, and its ability in this case to conquer overfitting. As seen in the charts earlier in the paper, the boosting algorithm is also able to learn the training data well enough to effectively generalize on the testing data with only a few thousand data points (versus the entirety of the training set).