Default of Credit Card Clients and Adult Census Income Supervised Learning Analysis

CS 7641: Machine Learning Assignment 1

By Divya Paduvalli (GT ID: dpaduvalli3/902913716)

Abstract

In this report, two distinct data sets have been analyzed using five machine learning algorithms namely: Decision trees, Boosting, K-nearest neighbors, Neural networks, and Support Vector Machines. The analysis results presented in this report was accomplished using Weka GUI.

About the Data Sets

The datasets were downloaded from the UCI Machine Learning Repository. Please refer to the links provided in the README.txt to learn more about the attributes. The data sets used in this analysis are the following:

1. Default of Credit Card Clients: This data set was from a Taiwan-based credit card issuer whose customers had default credit card payments. There are 24 attributes, 30,000 instances, and 2 classes. The task was to predict the likelihood of default based on attributes such as past credit history, age, marital status etc.

2. Adult Census Income: This data set was an extraction from the 1994 census database. There are 14 attributes, 48,861 instances, and 2 classes. The task was to predict whether the income of an adult US citizen is either less than or greater than or equal to $50,000 per year based on attributes such as age, race, education, gender etc.

Why are the Classification Problems Interesting?

The datasets were selected mainly for their practical implications and for the results they gave when machine learning algorithms were applied. Both the data sets are non-trivial classification problems with more than 10,000 instances. This provides an opportunity to experiment with these datasets by applying different machine learning algorithms. Both the data sets have been resistant to good performance, since none of the five algorithms have yielded exceptionally good results. One possible reason for this could be due to the nature of the datasets having imbalanced output classes and noise. However, in a lot of real-world application problems, data is not always perfect, and class imbalance issues are expected. I had to research and experiment with different tools and techniques to deal with imbalanced classes. It is important to note that both the datasets have not been artificially synthesized and are a direct reflection (or a subset) of real-world classification problems.

In the last few years, credit card issuers have become one of the major consumer lending products in the United States. This industry represents around 30% of total consumer lending (1). In a well-developed financial system, risk prediction is essential for predicting business performance and individual customer's credit risk. Credit card defaults not only result in significant financial losses to the bank but also damaged credit rating to a customer. Machine learning can be used in accurately predicting which customers are most probable to default and help notify customers who are on the verge of default. This in turn helps in reducing damage and uncertainty.

Census data is used by a wide variety of government agencies, businesses, academics, researchers etc. for a variety of purposes. The ability to predict a person's income using machine learning can be useful in applications like recommending housing, marketing of consumer or financial products, help business find locations to set up based on income, help the provision of healthcare plans and education, help make policy changes to increase household income in impoverished neighborhoods etc.

Data Pre-Processing

Both the datasets contained rows with missing values which were removed prior to splitting the data. I also changed the categorical class values from the Adult data set to nominal values to make it easier for the algorithms to process the data. The datasets were then split into 70% training and 30% testing. Weka's resample filter instance was used to split the data. After splitting, the rows in the data sets were randomized to ensure there is no bias. The machine learning algorithms were applied to the training test first and then to the testing set. For both the data sets, the output class labels were significantly imbalanced. If the class labels are imbalanced, the algorithms will have trouble learning and will underperform resulting in poor accuracy. This also results in overfitting issues. I have used Synthetic Minority Over-Sampling Technique (SMOTE) to deal with class imbalance. SMOTE increases minority instances in every iteration. It draws an imaginary line between existing minority instances and creates synthetic instances somewhere on those lines. After synthesizing new minority instances, the imbalance shrinks a little bit. Below are the results of balancing minority classes (1) with SMOTE.

	Adult Data Set			
	Unbalanced		Balanced	
	0	1	0	1
Training	15876	5237	15876	10474
Test	6778	2271	6778	4542

	Credit Card Default Data Set			
	Unbalanced		Balanced	
	0	1	0	1
Training	16083	4637	16083	9274
Test	6913	1968	6913	3936

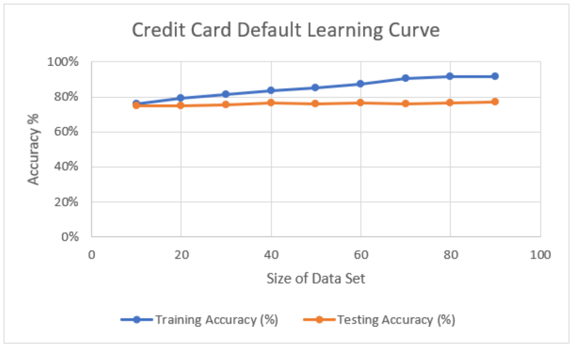
In addition, I also applied a cost sensitive classifier since SMOTE just achieves a reasonable balance as shown above and there is still a significant class imbalance which can lead to bias towards the majority class. The cost sensitive classifier increases the penalty for miss classifying the minority instance. However, doing this will decrease the accuracy but will correctly classify the minority instance. I have used a 5% penalty for misclassification and going beyond that did not have a significant effect on classification. The performance of the classification problems will be based on accuracy (% of correct instances).

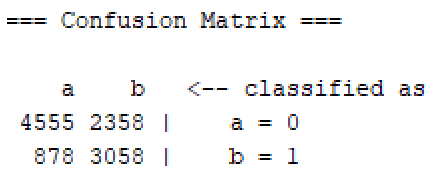
Algorithm Implementation

Highest testing accuracies have been highlighted in green in the tables represented in the sections below.

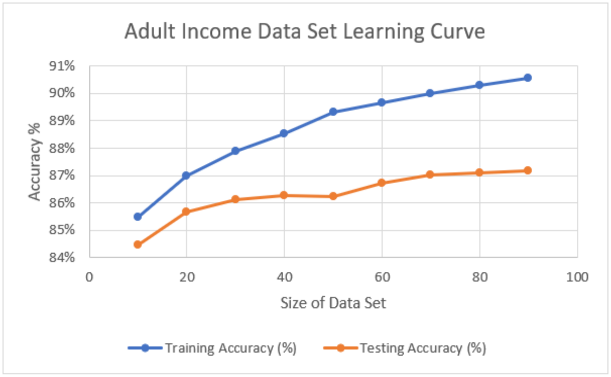
Decision Tress

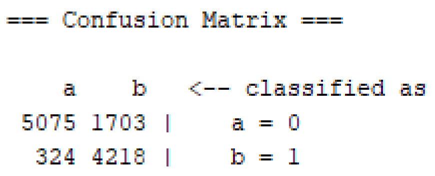
Credit Card Default Data Set									
Confidence Factor	Minimum Instance	Pruning	Build Model Time (seconds)	Test Model Time (seconds)	Size of Tree	Number of Leaves	Training Accuracy	10 Fold Cross Validation Accuracy	Testing Accuracy
0.125	2	Yes	2.28	2.27	1855	982	78.7593%	68.7305%	68.5685%
0.25	2	Yes	2.28	2.33	2598	1384	82.5847%	69.9412%	69.9143%
0.5	2	Yes	2.34	2.37	3283	1758	85.1165%	70.5801%	70.1447%
0.8	2	Yes	53.28	51.97	3543	1920	85.2546%	70.8456%	70.1724%




```
=== Confusion Matrix ===

    a    b   <-- classified as
 4555 2358 |    a = 0
  878 3058 |    b = 1
```

Adult Data Set									
Confidence Factor	Minimum Instance	Pruning	Build Model Time (seconds)	Test Model Time (seconds)	Size of Tree	Number of Leaves	Training Accuracy	10 Fold Cross Validation	Testing Accuracy
0.125	2	Yes	0.47	0.47	245	175	80.8653%	79.7723%	80.1325%
0.25	2	Yes	0.47	0.48	585	451	82.4023%	80.8083%	80.8216%
0.5	2	Yes	0.55	0.53	1649	1301	85.4839%	81.4383%	82.0936%
0.8	2	Yes	32.62	45.07	3838	3113	86.9564%	80.6528%	81.8110%



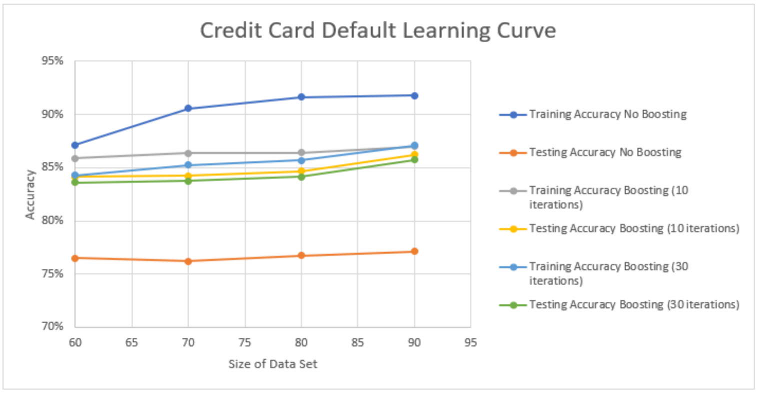

```
=== Confusion Matrix ===

    a    b   <-- classified as
 5075 1703 |    a = 0
  324 4218 |    b = 1
```

Weka's J48 algorithm was utilized to build the decision trees. Parameters such as confidence factor, pruning, and minimum instance were tuned. Confidence factor is a statistical test that is applied at each stage of pruning. Smaller confidence factor values incur heavy pruning. It is interesting to note that the size of the trees increases with increasing confidence factor. Minimum instances per leaf guarantees that at each split, at least 2 of the branches will have the minimum number of instances. I had set a default value of 2 for minimum instance. Pruning has been applied, by manipulating the confidence factor and minimum instance to reduce the size of the decision tree and to mitigate overfitting. Pruning improves accuracy slightly on the Credit Card Default test dataset and has a negligible impact on the Adult test dataset. For both the datasets, building training model took more time because of the greater number of instances in the training data set. In addition, increasing confidence factor also contributed towards increased duration. 10-fold cross validation was used to get a better estimate of the performance of the predictive models. I chose K to be 10 (which was also the default) because it will allow the size of each validation partition to be large enough to provide a fair estimate of the model's performance and create a lower bias. If you notice the 10-fold cross validation accuracy results for both the data sets is very close but slightly greater than testing accuracy results. This slight indifference could be because of the way the data got split between training and testing. In the Credit Card Default data set, the learning curves are very close to each other and both have relatively low scores. This might be due to high bias and under fitting. This high bias could be created due to the class imbalance and noise in the dataset. In the Adult dataset learning curve, the training curve has much better scores than testing curve. This is could be because of high variance and overfitting. Another reason for the large variance could be the nature of the dataset itself and the way training and testing datasets got distributed.

<u>Boosting</u>

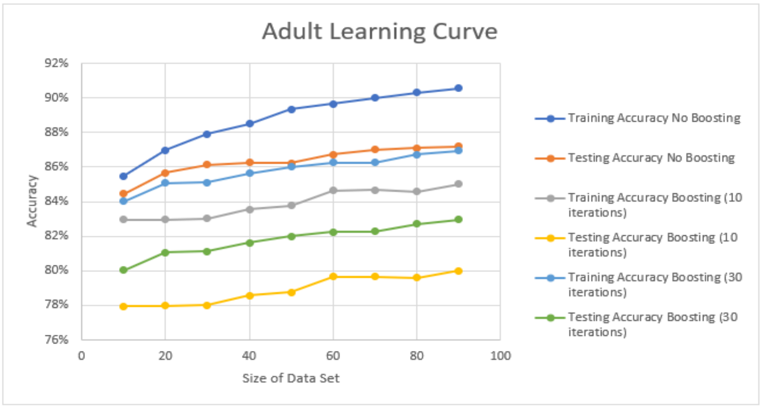
Credit Card Default Data Set

Iterations	Learner	Build Model Time (seconds)	Test Model Time (seconds)	Pruning	Training Accuracy	Testing Accuracy
10	0.125	36.9	36.87	Yes	99.9236%	78.1547%
10	0.25	38.49	37.03	Yes	99.8525%	78.2561%
10	0.5	39.3	39.26	Yes	99.6728%	78.8765%
10	0.8	40.87	50.17	Yes	99.1458%	78.7630%
30	0.125	126.36	124.62	Yes	99.5647%	80.9936%
30	0.25	125.78	132.8	Yes	99.9852%	81.2425%
30	0.5	118.19	119.24	Yes	99.6453%	81.3347%
30	0.8	230.15	245.12	Yes	99.4287%	81.4258%



Adult Data Set

Iterations	Learner	Build Model Time (seconds)	Test Model Time (seconds)	Pruning	Training Accuracy	Testing Accuracy
10	0.125	0.43	0.09	Yes	99.9696%	85.3092%
10	0.25	0.43	0.2	Yes	99.9924%	85.3975%
10	0.5	0.55	0.55	Yes	99.9944%	85.3269%
10	0.8	0.57	0.51	Yes	99.9932%	86.1823%
30	0.125	43.7	40.2	Yes	99.9962%	86.1042%
30	0.25	42	38.6	Yes	99.9975%	85.7244%
30	0.5	46.1	41.9	Yes	99.9961%	86.0512%
30	0.8	46.1	46	Yes	99.9957%	86.1886%

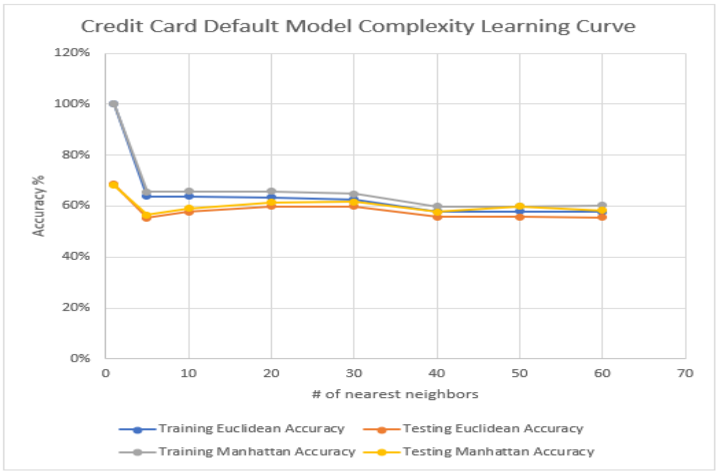


Boosting was performed using Weka's AdaBoostM1 and decision tree J48 algorithm. Boosting is typically used to improve prediction accuracy by applying information gain to learn over a subset of data. It creates a strong classifier by combining multiple weak classifiers. A weak classifier performs poorly but performs better than random guessing. The same confidence and minimum instance parameters used for decision tree were used for boosting. The only parameter that was changed was the number of iterations. Here the number of iterations is the number of weak classifiers. I manipulated with different iteration numbers and found that the models for both the datasets does not seem to improve significantly beyond 30.

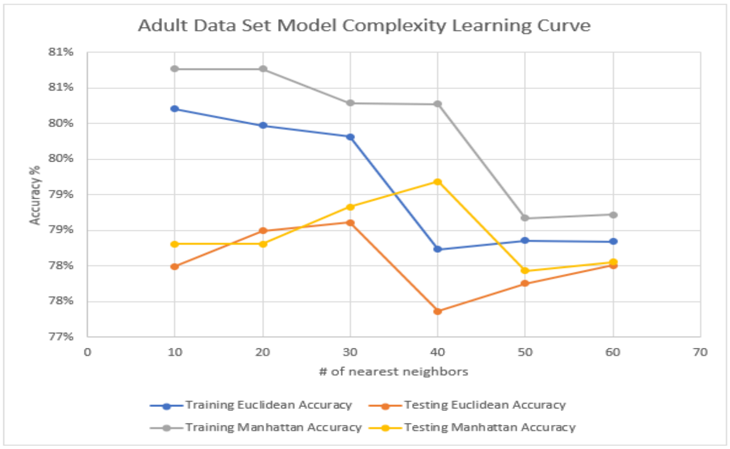
For both the datasets, the training accuracy was almost very close to 100%. When compared to the previous decision trees, the new results generated after applying boosting is significantly higher. In the Credit Card Default testing accuracy increased to between 10 to 11% and in the Adults data set the increase was between 5 to 6%. For both the data sets, the model building and testing time linearly increased with increased iterations, confidence factor, and pruning. The learning curves above has both the boosted (10 and 30 iterations) and non-boosted results. In both the data sets, the boosted results seem to be lower than non-boosted results. This could be because boosting typically reduces errors but not necessarily accuracy and the models could be overfitting training data.

<u>K Nearest Neighbors (KNN)</u>

Credit Card Default Data Set						
k	Training Euclidean Accuracy	Testing Euclidean Accuracy	Training Manhattan Accuracy	Testing Manhattan Accuracy	Diffrence Between Training Accuracies	Diffrence Between Testing Accuracies
1	100.000%	68.522%	100.000%	68.108%	0.000%	0.415%
5	63.856%	55.388%	65.576%	56.494%	-1.720%	-1.106%
10	63.876%	57.747%	65.848%	59.029%	-1.972%	-1.281%
20	63.438%	59.895%	65.714%	61.490%	-2.276%	-1.595%
30	62.705%	59.867%	64.728%	61.692%	-2.023%	-1.825%
40	57.862%	55.756%	59.778%	57.720%	-1.917%	-1.963%
50	57.870%	55.719%	59.956%	59.956%	-2.086%	-4.236%
60	57.720%	55.535%	60.208%	58.208%	-2.488%	-2.673%



Adult Data Set						
k	Training Euclidean Accuracy	Testing Euclidean Accuracy	Training Manhattan Accuracy	Testing Manhattan Accuracy	Diffrence Between Training Accuracies	Diffrence Between Testing Accuracies
1	100.0000%	81.1926%	99.9962%	81.1307%	0.000038	0.000619
5	79.7951%	75.7951%	80.0835%	75.9276%	-0.002884	-0.001325
10	80.2049%	77.9947%	80.7666%	78.3127%	-0.005617	-0.003180
20	79.9734%	78.4982%	80.7666%	78.3127%	-0.007932	0.001855
30	79.8140%	78.6131%	80.2884%	78.8339%	-0.004744	-0.002208
40	78.2353%	77.3675%	80.2770%	79.1873%	-0.020417	-0.018198
50	78.3605%	77.7562%	78.6755%	77.9329%	-0.003150	-0.001767
60	78.3416%	78.0124%	78.7211%	78.0565%	-0.003795	-0.000441



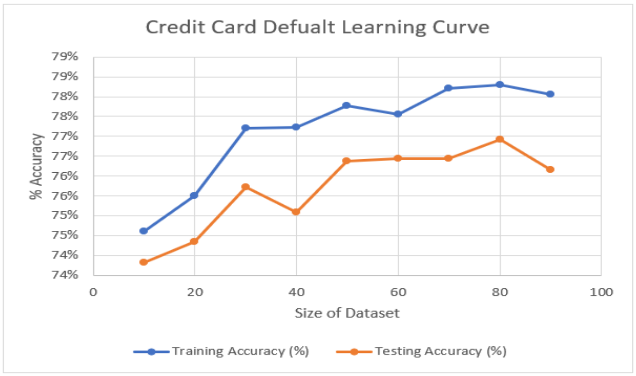
IBK algorithm in Weka was used to perform KNN classification. KNN is considered a lazy learning algorithm because it does not need any training data points (it memorizes training data) for generating a model and all the training data points are used in the testing phase. This in turn makes training faster and testing slower because it consumes more time and memory. K here represents the number of nearest neighbors. K was the only parameter that was changed in this algorithm. I experimented with different values of K for both the data sets and found that for the Credit Card Default data set the model did not significantly improve beyond K =30 and for the Adult data set it was K = 40. The time to build the model increased as K increased. But time allotted for testing was relatively fast. For both the data sets testing accuracy was the highest when K equaled 1. This is because of high recurrence of instances with same features and output class. When K greater than 1 testing set accuracy increased in both the data sets.

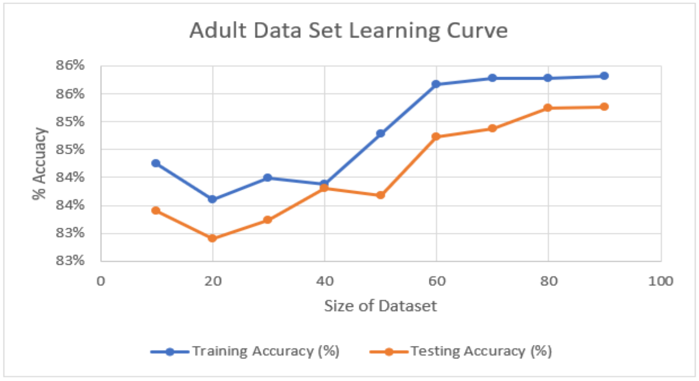
For both the datasets, Euclidean (default function in Weka) and Manhattan functions were applied. As per the Euclidean distance formula it puts more weight on the closest neighbors when compared to Manhattan distance. But on the contrary, Manhattan distance would be preferable in case of high dimensional data (which I have). I tested out both the distance functions on different values of K to evaluate model complexity. As shown in the model complexity chart, for the Credit Default dataset the difference between the Euclidean and Manhattan distances is very small, so distance does not seem to have a huge impact on the accuracies. But in Adult data set, there was a noticeable difference by the results generated by the two functions. Manhattan distance produced more accurate results when compared to Euclidean distance. One possible explanation for this behavior is because of the nature of the Adult Data set. The data set contains many discrete attributes. Manhattan distance assumes there are no continuous linear independent attributes and hence could have performed relatively better.

Neural Networks

Credit Card Default Data Set						
Learning Rate	Momentum	Hidden Layers	Build Model Time (seconds)	Test Model Time (seconds)	Training Accuracy	Testing Accuracy
0.1	0.2	15	132.86	137	78.3768%	77.5002%
0.3	0.2	8	74.74	73.26	78.0179%	77.1500%
0.5	0.1	10	93.19	93.49	77.5210%	77.0947%
0.8	0.5	12	103.37	103.54	77.7655%	77.1868%

Adult Data Set						
Learning Rate	Momentum	Hidden Layers	Build Model Time (seconds)	Test Model Time (seconds)	Training Accuracy	Testing Accuracy
0.1	0.2	15	324.06	320.02	80.5216%	79.0113%
0.3	0.2	8	229.76	229.03	86.2429%	85.3445%
0.5	0.1	10	293.7	291.43	85.4839%	85.4682%
0.8	0.5	12	320.12	318.33	80.6777%	79.0421%



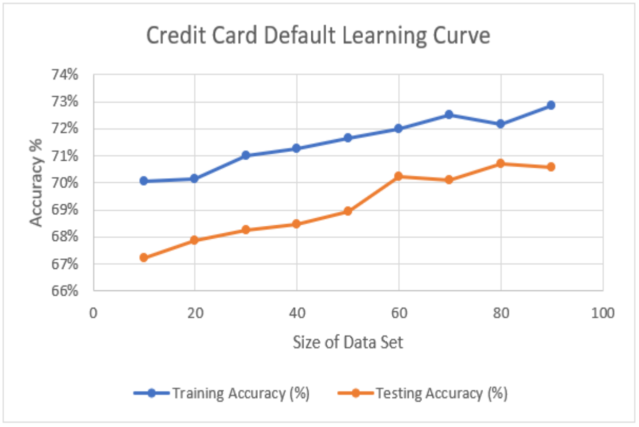


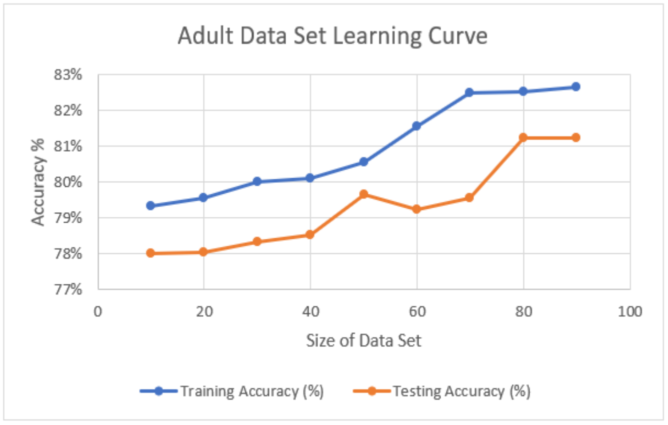
MultilayerPerception was the algorithm used in Weka to evaluate the performance and behavior of neural networks. In the algorithm, four parameters were tuned namely, learning rate, momentum, hidden layers, and iterations/epochs. Learning rate is how much weight is assigned to new examples. The momentum coefficient adds speed to the training of any multi-layer perceptron by adding part of the already occurred weight changes to the current weight change. Hidden layer is the layer between input layers and output layers. I experimented with different hidden layers. As a staring point, I used (average number of features + classes)/ number of output classes. For the Credit Card Default data set, the starting hidden layers will be 13 and for the Adult data set it will be 8. Accuracy generally improves when hidden layers increase. But on the contrary, in most practical cases it is advised not to use more than one hidden layer because the back-propagation algorithm become less effective. This also causes overfitting problems where the network will perform very well on the training set but may perform poorly on the test set. In the Credit Default data set, increasing hidden layers increased the testing accuracy slightly and in the Adult dataset, testing accuracy decreases rapidly. This algorithm is an eager learner and utilizes back propagation to assign weights for different perceptions. As far as the learning curves go, both the data sets have a training curves with better scores than the testing curves. This indicates that the parameters were too aggressive and had overfit the data which has also created a high variance.

Support Vector Machine Learning (SVM)

Credit Card Default Data Set							
Kernel	Exponent	C	Gamma (G)	Build Model Time (seconds)	Test Model Time (seconds)	Training Accuracy	Testing Accuracy
Polynomial Kernel	1	Not Applicable	Not Applicable	126.03	116.03	69.256%	68.756%
Polynomial Kernel	2	Not Applicable	Not Applicable	130.22	120.22	72.351%	71.851%
Polynomial Kernel	3	Not Applicable	Not Applicable	250.65	240.65	74.157%	73.657%
RBF Kernel	Not Applicable	1	1	225.89	215.89	74.237%	72.986%

Adult Data Set							
Kernel	Exponent	C	Gamma (G)	Build Model Time (seconds)	Test Model Time (seconds)	Training Accuracy	Testing Accuracy
Polynomial Kernel	1	Not Applicable	Not Applicable	110.04	105.04	81.6526%	81.1526%
Polynomial Kernel	2	Not Applicable	Not Applicable	190.06	189.14	81.9826%	81.4826%
Polynomial Kernel	3	Not Applicable	Not Applicable	210.55	207.55	82.1354%	81.6354%
RBF Kernel	Not Applicable	1	1	230.47	224.18	82.9858%	82.4858%





Weka’s SMO algorithm was used to build the SVM model. SVM models are considered eager learner that performs classification by finding the hyper-plane that differentiate the two classes very well. I experimented with two kernel types namely, polynomial kernel and radial basis function (RBF) kernel. The parameters that were tuned for the polynomial kernel was the exponent and for the RBF kernel was C and the gamma factor (G). The exponent controls the degree of the polynomial. For the exponent, I selected 1 for linear kernel, 2 for quadratic kernel, and 3 for cubic kernel. C in RBF controls the tradeoff between fitting the training data (large values) and maximizing the separating margin (small values). Values of C is in the range from 0.01 to 100. Values outside this range typically does not work well. Gamma factor controls the width of the RBF kernel. There is no set range for selecting G but I experimented with different values of C and G by doing a grid search using cross validation. The best performing pair (C, G) turned out to be (1, 1). For Credit Card Default data set the highest performing test accuracy was from the polynomial kernel with the exponent of 3. One possible explanation for this performance could be that higher degree polynomial kernels allows for a more flexible decision boundary when compared to a lower degree (less flexible). In the Adult data set, RBF kernel performed the best. I initially experimented with different values of C and G for both the data sets before doing a grid search. I noticed that the behavior of the model was extremely sensitive to the gamma parameter. If G was too large, the decision region is very broad and if too low, the region gets narrower. The time to build the models increased as the model complexity increased (ex: polynomial kernels).

<u>Summary</u>

The tables below summarizes all the results achieved from the five algorithms.

Credit Card Default Data Set		
Classifiers	**Parameters**	**Testing Accuracy**
Decision Tree	Confidence Factor =0.8 , Minimum Instance =2 , Pruning = Yes	70.1724%
Boosting	Iterations =30 , Learner = 0.8	81.4258%
KNN	K = 1	68.5224%
Neural Network	Learning Rate = 0.1, Momentum = 0.2, Hidden Layers = 15	77.5002%
SVM	Kernel = Polynomial, Exponent = 3	73.6568%

Adult Data Set		
Classifiers	**Parameters**	**Testing Accuracy**
Decision Tree	Confidence Factor = 0.5, Minimum Instance =2, Pruning =Yes	82.0936%
Boosting	Iterations = 30, Learner = 0.8	86.1886%
KNN	K = 1	81.1926%
Neural Network	Learning Rate = 0.5, Momentum = 0.1, Hidden Layers = 10	85.4682%
SVM	Kernel = RBF , C = 1, Gamma = 1	82.4858%

Sources

1. http://salserver.org.aalto.fi/opinnot/mat-2.4177/2018/McKinseyFinal.pdf
2. https://www.cs.waikato.ac.nz/ml/weka/
3. Adult Data Set: http://archive.ics.uci.edu/ml/datasets/Adult
4. Credit Card Default Data Set: http://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients