BUAN 6341 APPLIED MACHINE LEARNING

Project 3

Team Members:

Aji Somaraj

Neeraja Anil

Submitted on:

November 6th,2017

Naveen Jindal School of Management

University of Texas at Dallas

Dataset Description

Adult Income Dataset is a subset from the 1994 US Census form UCI data repository. Our objective is to create classification models using KNN and Artificial Neural Network to predict is if an "individual" has an income greater than $50,000 a year, by mining census data containing education, heritage and age (among others). It contains approximately 32000 observations, with 15 variables.

Bank Marketing Dataset available at the UC Irvine Machine Learning Repository. Our objective is to create classification models using KNN and Artificial Neural Network which can predict individuals who would subscribe a term deposit by mining the data related with direct marketing campaigns of a Portuguese banking institution containing information such as customer data, campaign activities and social and economic environment data. It contains approximately 45000 observations with 12 categorical variables and 7 numerical variables as its features and a binary category as its response variable.

Classification Algorithm

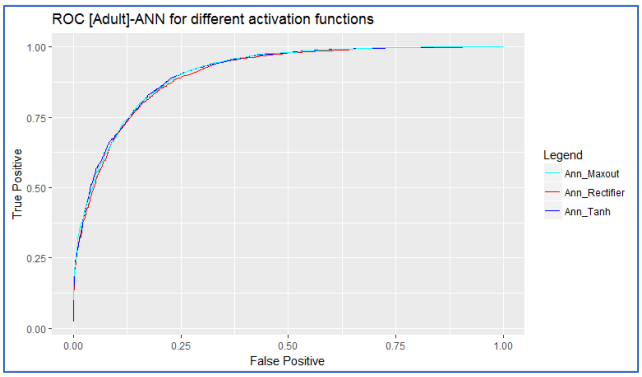
1. Artificial Neural Network (ANN)

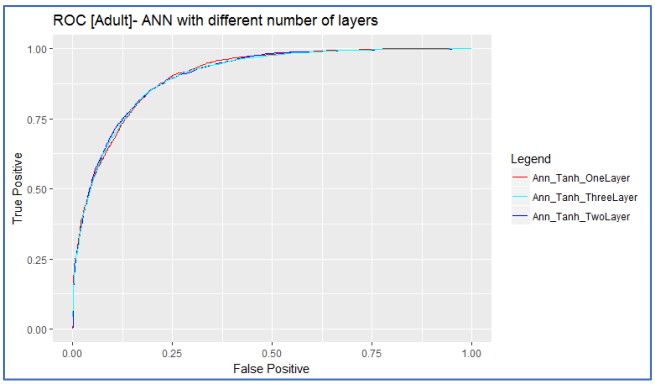
The H2o module in R was used to implement ANN. The usage of this package considerably reduces the number of parameters the user must specify since it includes automatic data standardization and handling of categorical variables and missing values. We experimented with the number and sizes of hidden layers and the activation function to study the model performance using 5-fold cross validation. Learning curve experiments were performed to determine the sensitivity of the algorithm's results to the size of the dataset and CPU clock time.

ANN was performed with 3 activation functions, namely, tanh, rectified linear and Maxout. The tanh function is a rescaled and shifted logistic function and its symmetry around 0 allows the training algorithm to converge faster. In Maxout, each neuron picks the larger output of k separate channels, each with its own weights and bias values. The Rectifier is the special case of Maxout where one channel always outputs 0. It is difficult to determine a "best" activation function to use; each may outperform the others in separate scenarios, but grid search models can help to compare activation functions and other parameters. The default activation function used here is the Rectifier.

The performance of the algorithm for varying number of hidden layers with rectifier activation function was observed. Furthermore, the ANN performance for varying number of layers with different number of activation nodes for each layer was experimented.

Dataset 1 [Adult]

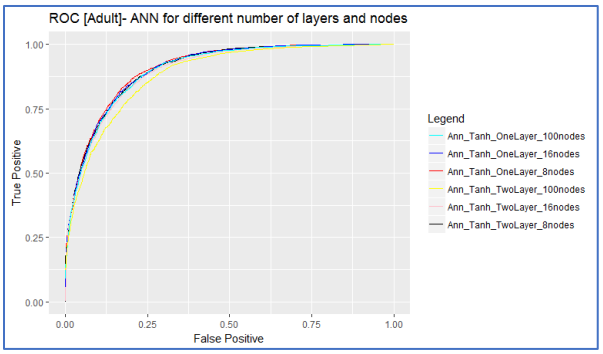




Activation Function	AUC	Accuracy	Sensitivity	Specificity
Rectifier	0.9038	83%	62.36%	92.68%
Tanh	0.9086	83.85%	64.67%	91.99%
Maxout	0.9083	83.54%	63.45%	92.72%

Number of Layers	AUC	Accuracy	Sensitivity	Specificity
One	0.9072	83.14%	62.57%	92.8%
Two	0.9065	84.08%	65.34%	91.83%
Three	0.9052	83.35%	62.96%	92.85%

Model evaluation metrics like AUC from ROC curve, accuracy, sensitivity and specificity for each model are tabulated. From the ROC plot of model with 1 hidden layer and 8 activation nodes, model with activation function as tanh outperforms the other two. Hence this activation function is used for further experimentation with the number of hidden layers and activation nodes. ROC plot for varying hidden layers shows that model with one hidden layer gives better results as compared to two or three hidden layered models.



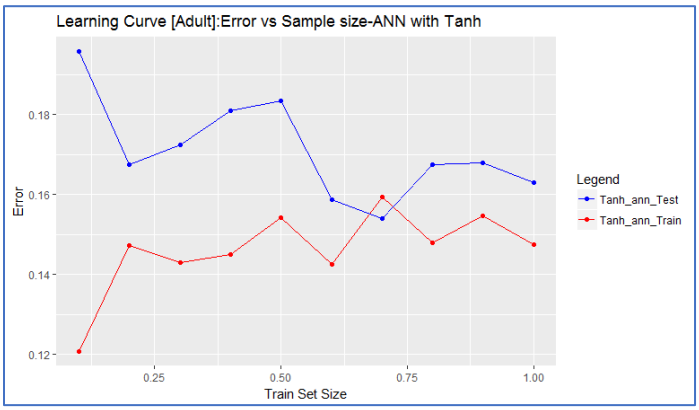
Layers/nodes per layer	AUC	Max Accuracy	Max Sensitivity	Max Specificity
one layer/100	0.9024	83.05%	62.65%	92.36%
one layer/16	0.9060	82.93%	61.97%	93.11%
one layer/8	0.9093	82.9%	61.59%	93.67%
Two layers/100	0.8845	80.77%	58.65%	91.51%
two layers/16	0.8994	82.27%	60.89%	92.74%
two layers/8	0.9058	83.64%	64.21%	91.98%

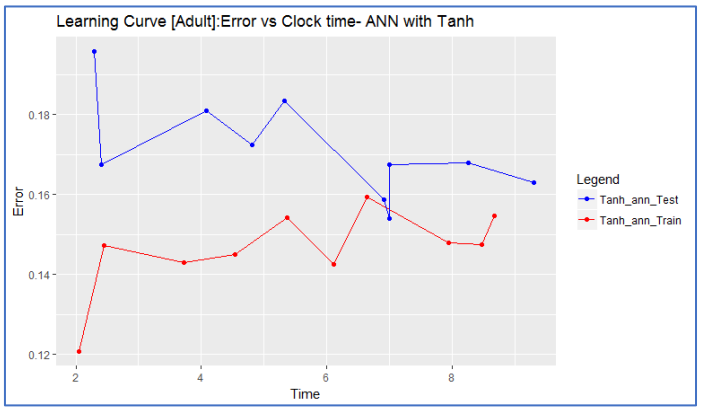
ROC plot for different number of layers and nodes shows that model with one hidden layer containing 8 nodes outperforms all the others in AUC, accuracy and sensitivity. Hence, a model with 1 hidden layer and 8 nodes using tanh activation function, as given below, is chosen for further experimentation.

Activation Function	Layers	Nodes per layer	AUC	Max Accuracy	Max Sensitivity	Max Specificity
Tanh	1	8	0.9086	83.85%	64.67%	91.99%

This implies that 84% of times, the model built with ANN has classified the income level correctly, 65% of the times, the income level being less than or equal to USD 50000 in classified correctly and 92% of the times, the income level being greater than USD 50000 is classified correctly.

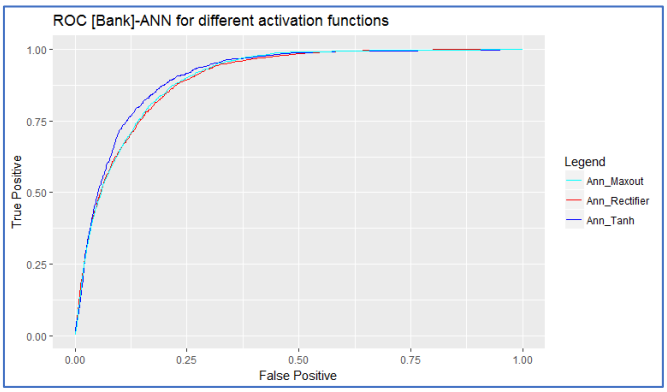
Learning Curves

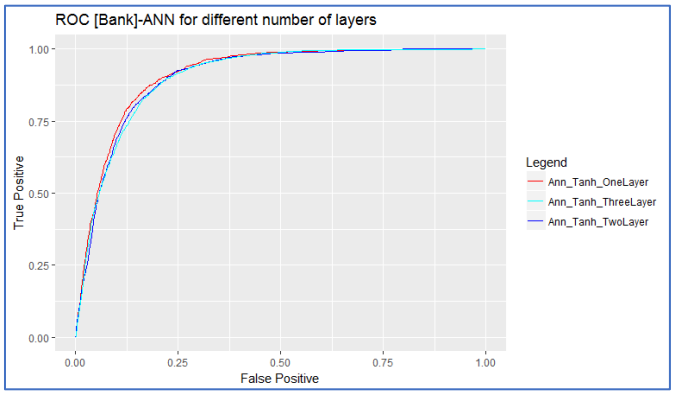




The Error vs sample size plot shows that, initially when the train size is low we can see that the errors are high for test and low for train, as the size increases the train error goes down and the test error also comes down before reaching a plateau., which implies a slight overfitting. The Error vs Clock time plot shows a similar trend to the previous learning curve. With increase in time, the algorithm becomes more confident in giving accurate results and error rate goes down.

Dataset 2 [Bank]



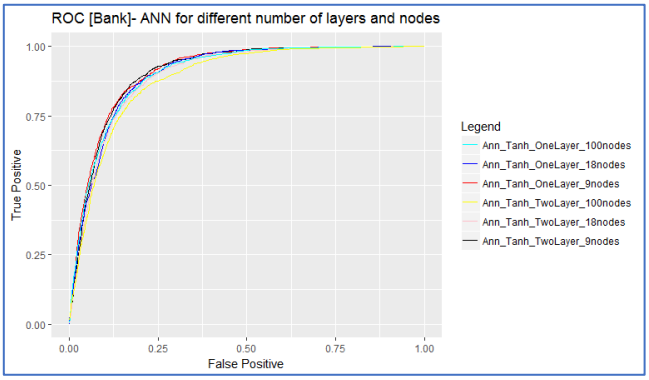


Activation Function	AUC	Max Accuracy	Max Sensitivity	Max Specificity
Rectifier	0.8985	88.03%	49.06%	94.64%%
Tanh	0.9093	87.96%	48.98%%	95.99%
Maxout	0.9007	87.16%	46.5%	95.03%

Number of Layers	AUC	Max Accuracy	Max Sensitivity	Max Specificity
One	0.9126	87.36%	47.47%%	96.45%
Two	0.9047	86.61%	45.59%	96.32%
Three	0.9042	86.68%	45.54%	95.82%

Model evaluation metrics like AUC from ROC curve, accuracy, sensitivity and specificity for each model are tabulated. From the ROC plot of model with 1 hidden layer and 9 activation nodes, model w ith activation function as tanh outperforms the other two. Hence

this activation function is used for further experimentation with the number of hidden layers and activation nodes. ROC plot for varying hidden layers shows that model with one hidden layer gives better results as compared to two or three hidden layered models.



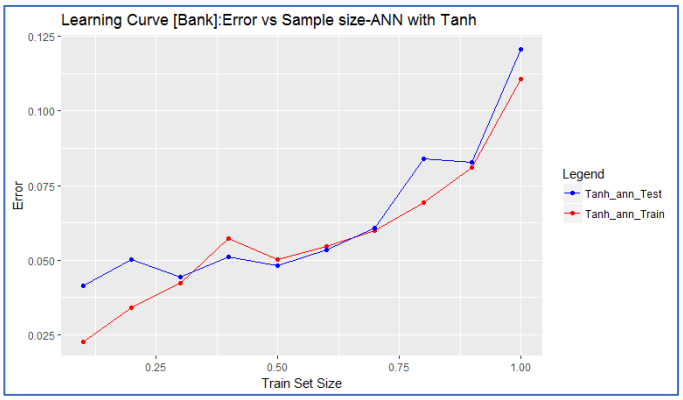
Layers/nodes per layer	AUC	Max Accuracy	Max Sensitivity	Max Specificity
one layer/100	0.9037	87.15%	46.74%	95.79%
one layer/18	0.9029	86.61%	45.46%	96%
one layer/9	0.9113	87.72%	48.37%	96.16%
Two layers/100	0.8849	85.4%	42.64%	95.88%
two layers/18	0.897	86.34%	44.8%	95.95%
two layers/9	0.9078	87.9%	48.83%	95.88%

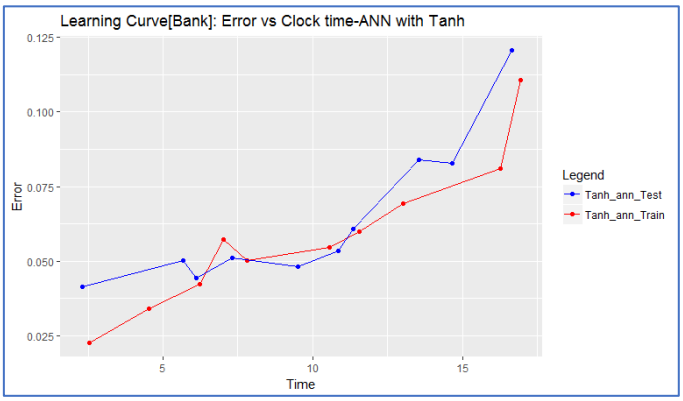
ROC plot for different number of layers and nodes shows that model with one hidden layer containing 9 nodes outperforms all the others in AUC, accuracy and specificity. Hence, a model with 1 hidden layer and 9 nodes using tanh activation function, as given below, is chosen for further experimentation.

Activation Function	Layers	Nodes per layer	AUC	Max Accuracy	Max Sensitivity	Max Specificity	False Positive	False Negative
Tanh	1	9	0.9093	87.96%	48.98%%	95.99%	27.03%	10.32%

This implies that 88% of times, the model built with ANN has classified the account openers correctly, 49% of the times, the account openers were classified correctly and 96% of the times, the non-account openers were classified correctly. We need a model with high specificity in this scenario, to limit marketing campaigns only to potential account openers.

Learning Curves



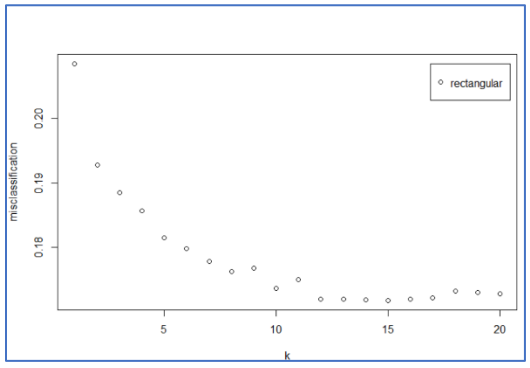


The Error vs sample size as well as the error vs clock time plot shows a similar trend for training and test data sets. Learning Curve of error vs size shows high bias which implies the model underfits the data and model performs poorly for large sample size. The error vs time plot shows the algorithm doesn't train well with the given data and the error rate goes up with more time, i.e., the algorithm has a poor learning rate.

2. K-Nearest Neighbors (KNN)

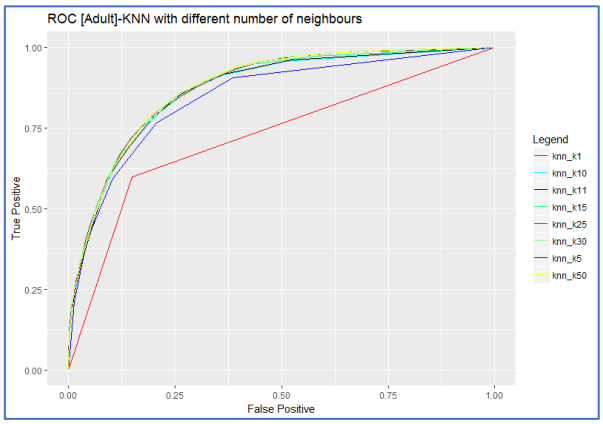
The package kknn in R was used to implement unweighted KNN with rectangular kernel on both the datasets. The distance metric used here is Euclidean distance (kknn uses Minkowski distance which is equal to Euclidean distance when distance parameter is 2). The expectation is that the squared distance in the Euclidean distance function more aggressively weighs the closest neighbors.

Dataset 1 [Adult]



The number of nearest neighbors, k, that are evaluated per instance were tested between 1 and 50. The model complexity chart on the left, shows that, k=15 gives the most accurate result.

ROC Curve for KNN with different number of neighbors is given on the right.

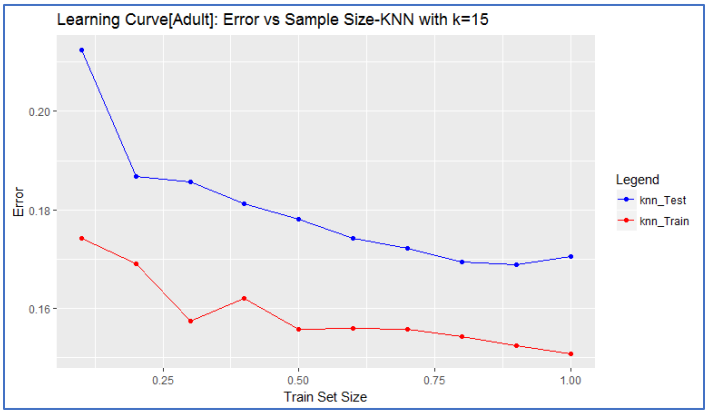


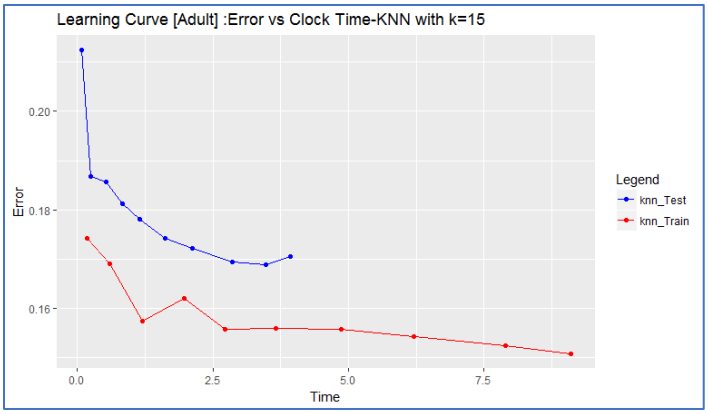
It can be observed from the ROC plot that k=15, outperforms the others and it is chosen for further experimentation with learning curves. The model evaluation metrics are tabulated as below:

k	AUC	Accuracy	Sensitivity	Specificity	Kappa
15	0.8751	83.23%	91.01%	58.61%	0.5211

This implies that 83% of times, the model built with KNN has classified the income level correctly, 91% of the times, the income level being less than or equal to USD 50000 in classified correctly and 59% of the times, the income level being greater than USD 50000 is classified correctly, the model classification is poor for high income class.

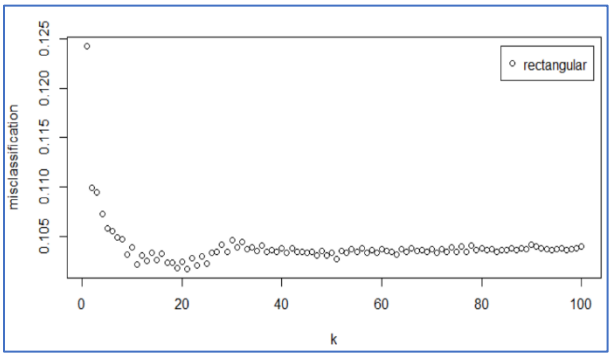
Learning Curves



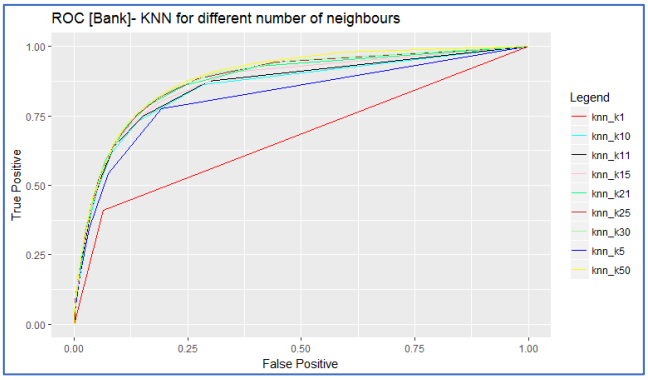


The Error vs sample size plot shows that, initially when the train size is low we can see that the errors are high for test and low for train. As the size increases, the train error goes down and the test error also comes down and reaches a plateau, which implies a slight overfitting as there is a small gap between the curves. The Error vs Clock time plot shows a similar trend to the previous learning curve. With increase in time, the algorithm becomes more confident in giving accurate results and error rate goes down, i.e., the algorithm exhibits a good learning rate.

Dataset 2 [Bank]



The number of nearest neighbors, k, that are evaluated per instance were tested between 1 and 100. The model complexity chart on the left, shows that, k=21 gives the most accurate result.



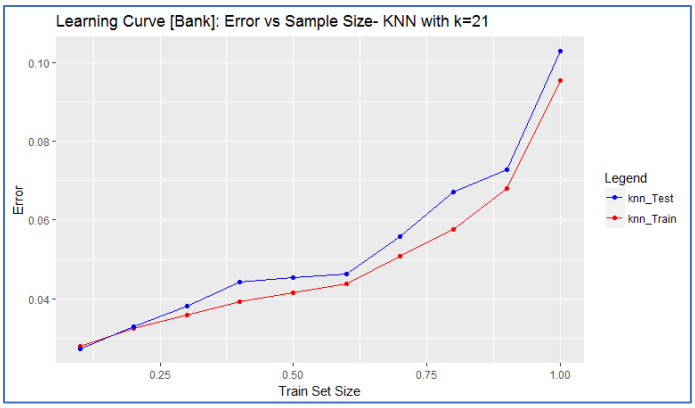
ROC Curve for KNN with different number of neighbors is given on the upper right.
It can be observed from the ROC plot that k=21, outperforms the others and it is chosen for further experimentation with learning curves. The model evaluation metrics are tabulated as below:

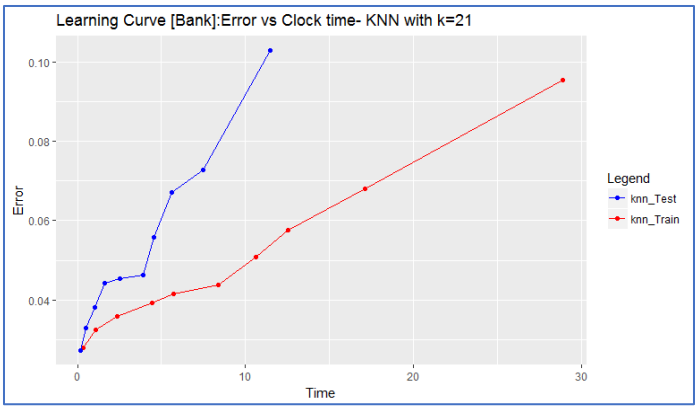
k	AUC	Accuracy	Sensitivity	Specificity	Kappa
16	0.8767	89.85%	97.92%%	28.92%	0.3533

This implies that 89% of times, the model built with KNN has classified the account openers correctly, 97% of the times, the account openers were classified correctly and 28% of the times, the non-account openers were classified correctly. We need a model with high specificity in this scenario, to limit marketing campaigns only to potential account openers. Since the specificity and kappa (less than 0.5), are very low, this model performance is very poor.

Learning Curves

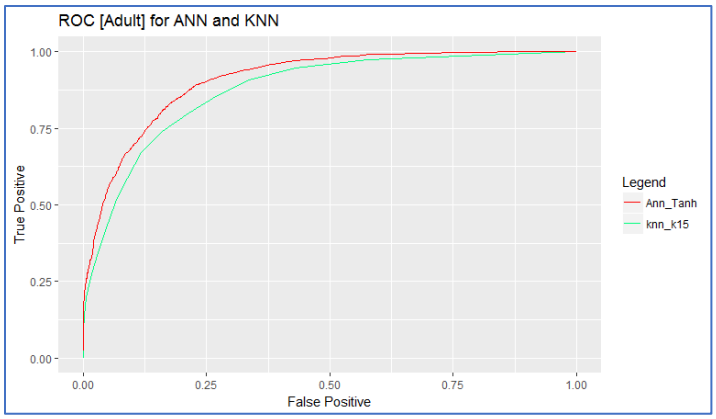
The Error vs sample size as well as the error vs clock time plot shows a similar trend for training and test data sets. Learning Curve of error vs size shows high bias which implies the model underfits the data and model performs poorly for large sample size. The error vs time plot shows the algorithm doesn't train well with the given data and the error rate goes up with more time, i.e., the algorithm has a poor learning rate.

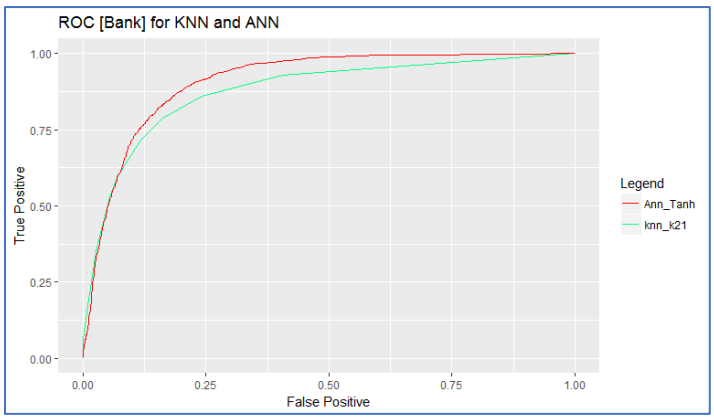




Summary

We built several classification models to predict whether a random U.S. citizen earns more than 50K a year and if a person would open a term-deposit due to campaign efforts utilizing ANN and KNN algorithms. ROC Curve of ANN and KNN on both the datasets are shown:





Model evaluation metrics is tabulated as shown below:

Dataset	Model	AUC	Accuracy	Sensitivity	Specificity
Adult	ANN – 1 layer/8 nodes	0.9086	83.85%	64.67%	91.99%
	KNN with k=15	0.8751	83.23%	91.01%	58.61%
Bank	ANN – 1 layer/9 nodes	0.9093	87.96%	48.98%	95.99%
	KNN with k=16	0.8767	89.85%	97.92%	28.92%

It is observed that ANN performs significantly better than KNN with high AUC and specificity values. 10-fold cross validation was performed for KNN while training the model to find the optimal model parameters. The model evaluation metrics for the 2 datasets on performing SVM, Decision Tree and XgBoosting are show below:

Adult Dataset

	SVM			Decision Tree		XgBoost	
	Sigmoid	Linear	Radial	Unpruned Tree	Pruned Tree	Unpruned	Pruned
Accuracy	0.6239	0.8442	0.8495	0.8316	0.8521	0.8567	0.8648
Sensitivity	0.7475	0.9332	0.9363	0.9079	0.9351	0.8681	0.8862
Specificity	0.2509	0.5755	0.5875	0.6012	0.6017	0.8024	0.7807
Kappa	-0.002	0.5496	0.5654	0.5306	0.5758	0.5741	0.6145
AUC	0.6072	0.8986	0.8986	0.8566	0.8911514	0.9144327	0.9242987

Bank Marketing Dataset

	SVM			Decision Tree		XgBoost	
	Polynomial	Linear	Radial	Unpruned Tree	Pruned Tree	Unpruned	Pruned
Accuracy	0.7447	0.7447	0.7303	0.8352	0.8486	0.9046	0.9073
Sensitivity	0.7238	0.724	0.7013	0.8444	0.851	0.9309	0.9272
Specificity	0.9023	0.9011	0.9496	0.7656	0.8311	0.6255	0.6614
Kappa	0.3363	0.3359	0.3327	0.4333	0.4822	0.4784	0.4686
AUC	0.8959333	0.8959	0.9058	0.8743113	0.8927807	0.9093191	0.9307

It can be observed that Xgboosting performs marginally better than ANN as ANN provides very low sensitivity values. However, the performance of ANN could be improved further by doing hyperparameter tuning to find out the optimal number of nodes and hidden layers through grid search at the cost of higher computational power. Grid search with 5-fold cross validation was performed to find out optimal model parameters utilizing all the cores in the machine. However, these results were not reproducib le since we did not use a single core H2O cluster and hence could not be relied upon. Hence the hidden layer configuration was determined through trial-and error and by following the two empirical thumb rules: (i) number of hidden layers equals one; and (ii) the number of neurons in that layer is the mean of the neurons in the input and output layers'. i.e., $\frac{14+2}{2} = 8$ neurons for adult dataset and $\frac{16+2}{2} = 9$ neurons for bank dataset.

The models for adult dataset could be improved by reducing the model flexibility by doing feature selection, as the learning curves show overfitting for both the algorithms. The models for bank dataset could be improved by increasing model flexibility by adding new domain-specific features, as learning curves for it indicate underfitting.