**First Classification Case:**

**1. Problem Definition**

Using some numeric and categorical data of a person such as age, education, race, job title etc., predict if a person’s annual income is more than $50,000 or not.

**2. Data**

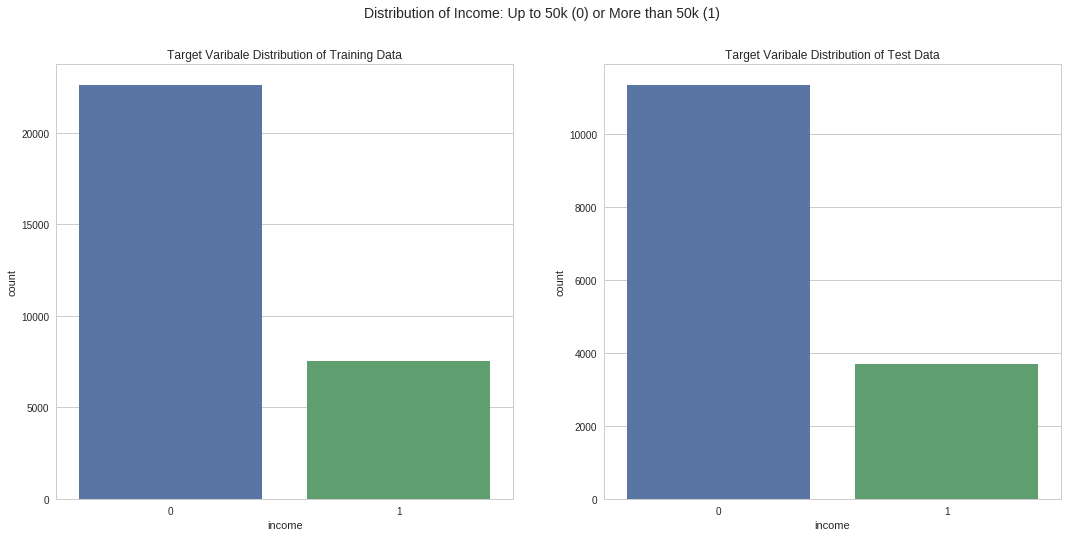
To build the corresponding classifier, we used “adult” data set (Blake and Merz 1998) from the UCI repository. (source: <http://archive.ics.uci.edu/ml/datasets/Adult>).

**3. Project Purpose**

Observe the behavior of a few machine learning algorithms such as Decision trees, Neural networks, Boosting, Support Vector Machines, and k-nearest neighbors by building solution models for the above-mentioned problem.

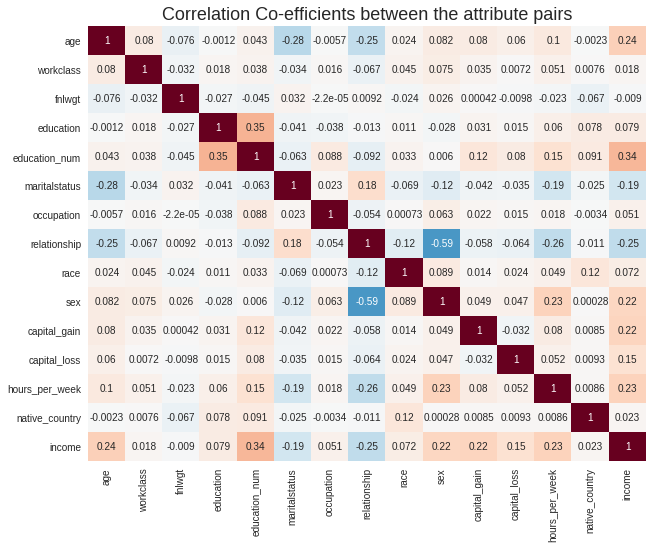
**4. EDA & Interesting Points**

Provided training and test datasets have 32561 and 16281 instances respectively. After removing the unknown and duplicate data, we have 30162 training instances and 15060 test instances. Both sets are properly labeled and have 14 predictor variables of categorical and numeric types. One of the interesting features of the dataset is that it is not well balanced. In both training and test dataset about 25% instances belong to the class of people with higher income (>50k) and about 75% are in the other class. However, the datasets are already shuffled for experiments.



**Figure 1. The distribution of the instances in the datasets by target classes**

In the training data, the predictor variables do not show any kind of strong correlation with the target variable. Some of the variables show almost zero correlation. **Some of the machine learning algorithms use distances between data points for classification, and some use the categorical features. This dataset would, therefore, be an interesting example to show the differences in performance of these algorithms**.

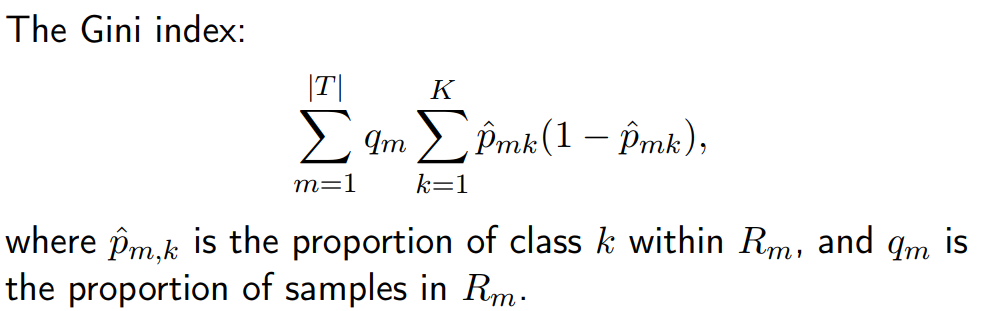


**Figure 2: Correlation heat-map of the variable pairs of training dataset**

**5. Experiments and Approaches**

**5.1. Decision Trees:**

Decision trees classification model is a non-parametric supervised learning method that predicts the value of a target variable by learning simple decision rules inferred from the data features. There are various algorithms to implement this method. We used an optimized version of CART (classification and regression trees) algorithm, which constructs binary trees using the numeric features and threshold that yield the largest information gain at each node. CART uses GINI index as a measure of sanity.



**Figure 3: Mathematical interpretation of GINI index**

At first, we implanted a vanilla version of the classifier without any restriction or pruning using scikit-learn python library. Later we used a grid search and cross-validation approach to find out the optimal depth of the tree and minimum number of data samples at a node before it is split to grow the tree further.

**5.3. Boosted Decision Trees:**

To observe the effect of boosting, we implemented Adaboost classifier from scikit learn library. We experimented with 10, 50, 250, and 500 estimators with a learning rate of 0.01. A grid search k-fold cross validation was performed to tune the hyper parameters for the best performance.

**5.2. KNN:**

We also implemented KNeighborsClassifier from the scikit-learn library. This classification method is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. A query point is classified based on which class has the most representatives within the **k** number of nearest neighbors of the point. We have experimented both with and without pruning the classifier. We also performed cross validation with different hyper-parameters to find the optimal K-value.

**5.3. Support Vector Machine**

SVM uses a subset of training points in the decision function (called support vectors) near the decision boundary for the classification. We implemented SVC classifier from scikit-learn library with a linear kernel and a non-linear kernel(‘rbf’). Initially, we fed the dataset without scaling, and the server crashed each time regardless of reducing cross validation folding, hyper-parameter tweaking, and using powerful hardware.

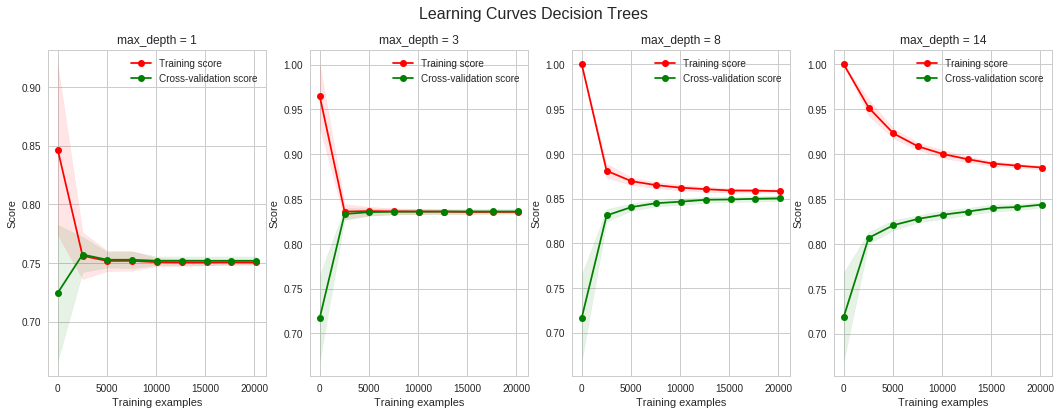
**5.4 Neural Network**

Finally we implemented deep neural network using python’s Tensoflow based Keras framework. We experimented with 3 to 5 densely connected hidden layers with ‘**relu’** activation function and **adam** optimizer with different learning rate. Each hidden layer contain 200 nodes. 14 scaled feature variables are used for input and one-hot-coded output was produced by the output layer using ‘sigmoid’ activation function. We compared the binary cross entropy loss of the training and validation data to determine the optimal network architecture and hyper-parameters.

**6. Observations and Results Analysis:**

**6.1 Decision Trees:**

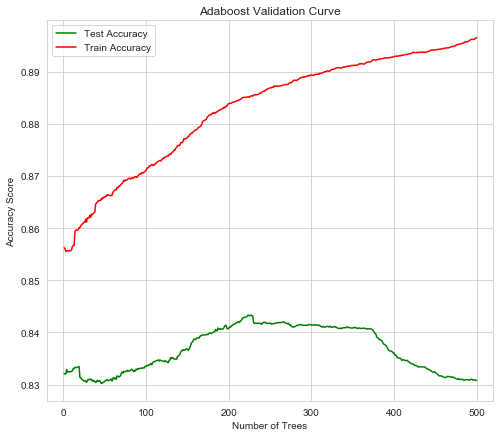
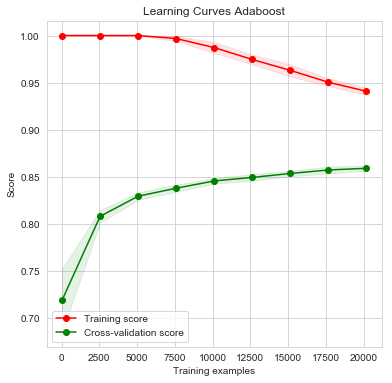
1. **Default version with no depth limitation (high complexity)**
   1. Result : training accuracy 99.99% and test accuracy 81.44% (**Overfitting)**
   2. Due to the high complexity and no restriction on branching, trees don’t generalize well.
   3. We tune max\_depth and min\_samples\_split for improved performance



1. Pruning: max\_depth = 1 and max\_depth = 3 , (min\_sample\_split = 0.001)
   1. At both depth, as we can see from the learning curve, the models are stagnant most of the time. It does not learn from the data.
   2. Overall accuracy for both training and validation set are also very low. These models suffer from Underfitting due to the High bias.
2. Pruning: max\_depth = 14
   1. This models remain unchanged from the untuned model. It suffers from the Overfitting due to the high complexity of the trees.
3. Pruning: max\_depth = 8
   1. At this depth the training accuracy drops close to 85% and then plateaus. Similarly, the validation data accuracy increases over the data size. Eventually it reaches close to the convergence. Therefore, the optimal depth would be around 8. We performed k-fold cross validation to with parameter grids to find the optimal hyper-parameters.
4. **Best Model with max\_depth = 9 and min\_samples\_split = 0.001**
   1. K-fold cross validation was performed over the grid of parameters: min\_samples\_split of 0.001, 0.01, 0.03, and 0.05 (since more than 5% is not recommended) and max\_depth ranging from 3 to 10.
   2. With optimal max\_depth = 9 and min\_samples\_split=0.001 training accuracy dropped to 85.86% and test accuracy improved to 82.83%. The overfitting is reduced significantly, because of the controlled branching and complexity.

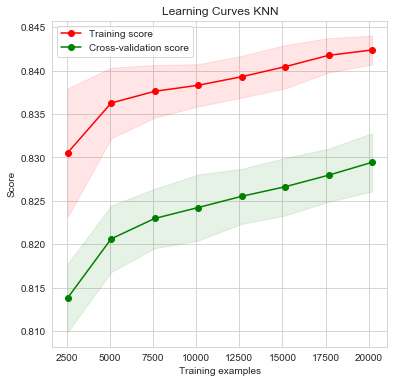
**6.3. Boosting with Adaboost classifier:**

1. **Default Adaboost classifier(n\_estimator=50, lr=1) on the pruned Decisiton Trees**
   1. Train accuracy is 0. 986 and test accuracy is 0.813 (Highly overfitted model)
   2. Potential Cause of Overfitting: Adaboost classifier (SAMME.R algorithm) uses probability estimate to update the additive model. Since the data is imbalanced, the higher probability of the ‘0’ class always has a greater effect on the process.
2. **Hyper-parameter tuned Adaboost (n\_estimator=300, lr=0.01)**
   1. From validation curve it is apparent that the optimal number of iterations would be between 150 to 350. In grid search cross validation we found it to be 300 and learning rate 0.01.
   2. Training Score lowered to 0.916 but the test score improved to 0.837**.** Yet the overfitting remains. From validation curve we see that it increases with the iteration. Even after tuning, it can not avoid the effect of the imbalanced data. Handling data imbalance would increase the accuracy which requires more time and powerful hardware.



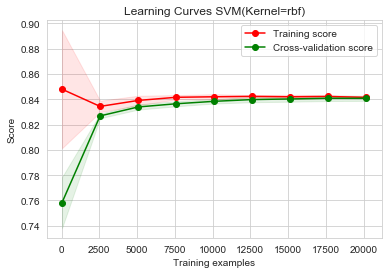
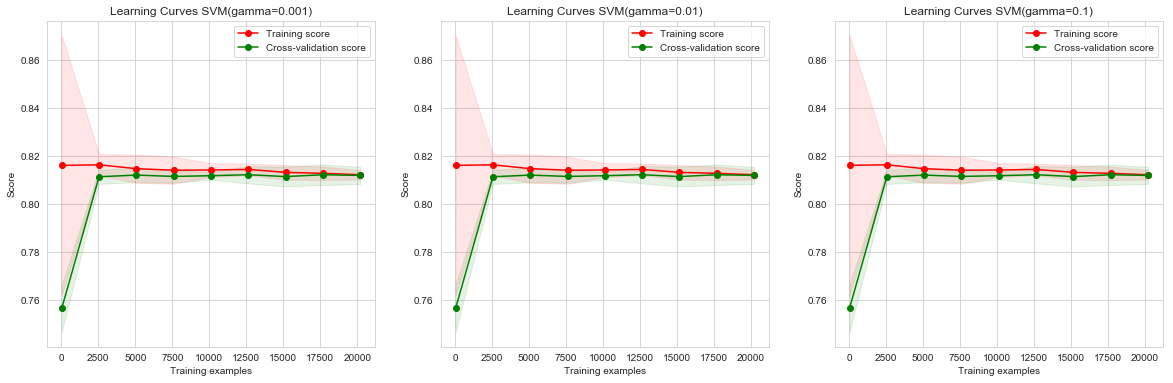
**6.4. KNN classifier:**

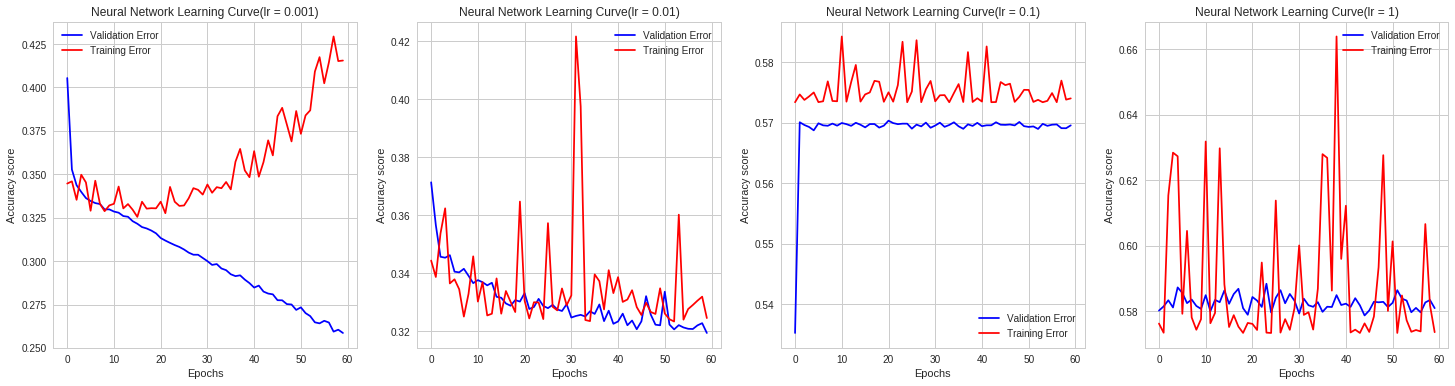
1. Defualt KNN classifier with k-value =1:
   1. Due to high bias it overfits. Training score is 0.871 and test score 0.817
   2. Cross validation with different Neighbor size would be helpful to improve the situation
2. **Hyper-parameter Tuned KNN (Number of Neighbors =23)**
   1. From grid search cross validation over the range of 5 to 30 neighbors, the best performing n\_neigbors =23
   2. Train score is 0.844 and test score is 0.829. The overfitting problem is solved but the model lacks generalization in learning from data. As we can see in the learning curve thetraining curve and testing curve do not converge.



**6.4. SVM classifier:**

1. **SVM model with Linear kernel (gamma=0.001, C=1)**

****



|  |  |  |  |
| --- | --- | --- | --- |
| Model Description | Train Accuracy  (%) | Test Accuracy  (%) | Wall time |
| Default Decision trees | 99.99 | 81.44 | 0.1s |
| Tuned Decision trees (max\_depth =9, min\_samples\_split=0.001) | 85.86 | 82.83 | 5.7s |
| Adaboost (Tuned Decision trees, estimators=50, learning\_rate =1) | 98.60 | 81.30 | 32.9s |
| Adaboost(Tuned Decision Trees, estimators=300, learning\_rate=0.01) | 91.60 | 83.71 | ~3min |
| Default KNN classifier | 87.17 | 81.74 | 0.3s |
| Tuned KNN classifier (n\_neighbors =23) | 84.43 | 82.93 | ~4min |
| SVM Linear Kernel (gamma = 0.001, C=1)(cross-validated) | 81.25 | 81.05 | ~4mins |
| SVM rbf Kernel (gamma=0.1, C=10)(cross-validated) | 84.29 | 81.19 | ~4mins |
| Deep Neural Network( 3 hidden layers with 100 nodes at each) | 84.75 | 84.30 | 37s |

**References**

[1] Blake, C. L. and C. J. Merz (1998). UCI repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, Irvine, CA. Available at <http://www.ics.uci.edu/~mlearn/MLRepository.html>

[2]