ML Assignment 1

Kuhu Gupta

1 Data set Description

- **Housing's Sale Price Data** The dataset contains records of the prices of houses along with various deciding features. Among the features used in the classification includes exterior condition, house style, overall quality rating, overall condition, type of dwellimg, building type, lot size, neighborhood, year built and year sold. The dataset contains 1460 data cases and 11 features.
  While doing the initial review of the data, the classes were defined as pricing brackets divided into 100k groups. I.e: 0-100k, 100k-200k, 200k-300k, etc. . Therefore, it required me to transform the data by calculating the brackets based on the 'sale price'. I also dropped a few unnecessary features to improve the training.
  This dataset is fairly reasonable to explore prices of houses which depends on large amount of features which makes it more important to do a thorough price estimation. Modelling home prices is both difficult and lucrative task as it affects a larger population and could be really beneficial for everyone since house prices depend a lot on personal taste.

- **US Visa Applications Data** The dataset contains records of people who applied for US Visa. The problem is to classify whether their application will be accepted or denied based on the parameters of their application. In the original dataset, there are 154 features and 374365 total samples but for classification I am only using 7 features. The features are industry code, job class, wage rate, wage type, country of citizenship and employer location.
  The dataset is relatively clean but most of the features are textual in nature. To transform the features into numeric values suitable for the machine learning algorithms, I used a label encoder built into ScikitLearn. I also dropped the rows having missing values as well as all unnecessary columns to speed up with data processing and training process. Moreover, case status feature was transformed so that results such as 'certified' and 'certified withdrawn' are both concerned as 'approved' conditions whereas 'denied', 'invalidated', and 'rejected' were seen as 'denied'. After preprocessing the data, the dataset contains 18236 rows and 7 columns.
  The dataset is compelling due to the variety of features it offers that can directly affect a person's application. Moreover, the huge number of data cases creates a rich dataset and greater confidence in training and testing rates. The goal of using this data is to effectively and efficiently determine the application result in no time and money.

2 Support Vector Machine

- **Housing's Sale Price Data**

Kernel	Gamma	Train %	Train Time	Test %	Test Time
rbf	0.01	0.7541	0.2313	0.7123	0.0050
rbf	0.05	0.7938	0.2254	0.7397	0.0060
rbf	1.0	0.9216	0.7350	0.6986	0.0080
rbf	2.0	0.9505	1.0562	0.6781	0.0110
poly	0.01	0.6309	0.2244	0.5685	0.0040
poly	0.05	0.7374	0.2274	0.6507	0.0060
poly	1.0	0.9361	0.8387	0.6575	0.0040
poly	2.0	0.9254	1.0173	0.6301	0.0040

Table 1: Train%, Training Time, Test%, Testing Time for various kernel/gamma configurations

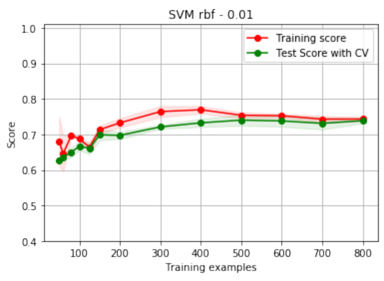

Figure 1: *
RBF SVM w/ 0.01 gamma

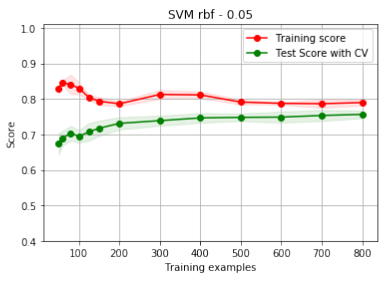

Figure 2: *
RBF SVM w/ 0.05 gamma

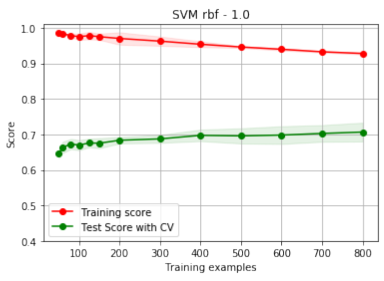

Figure 3: *
RBF SVM w/ 1.0 gamma

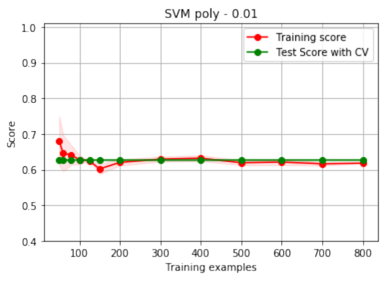

Figure 4: *
Poly SVM w/ 0.01 gamma

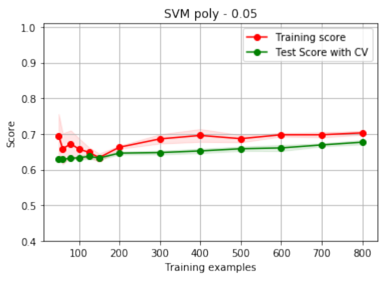

Figure 5: *
Poly SVM w/ 0.05 gamma

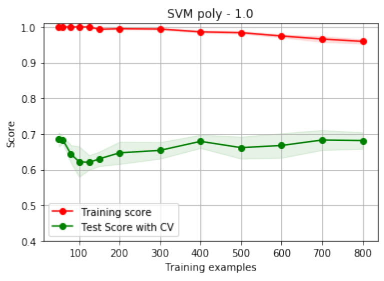

Figure 6: *
Poly SVM w/ 1.0 gamma

- **US Visa Applications Data**

Kernel	Gamma	Train %	Train Time	Test %	Test Time
rbf	0.01	0.8641	26.4614	0.8722	0.2214
rbf	0.05	0.8643	35.8407	0.8722	0.2343
rbf	1.0	0.8751	35.6659	0.8771	0.2563
rbf	2.0	0.8807	39.3931	0.8771	0.2681
poly	0.01	0.8640	13.1945	0.8722	0.1236
poly	0.05	0.8643	26.6073	0.8717	0.1386
poly	1.0	0.8496	35.4273	0.8563	0.1087
poly	2.0	0.8382	25.2476	0.8350	0.0638

Table 2: Train%, Training Time, Test%, Testing Time for various kernel/gamma configurations

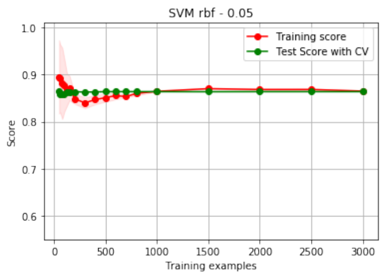

Figure 7: *
RBF SVM w/ 0.05 gamma

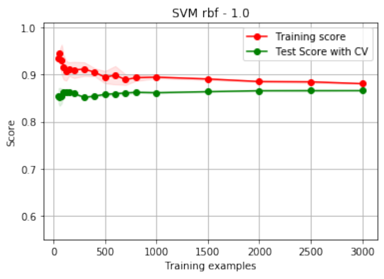

Figure 8: *
RBF SVM w/ 1.0 gamma

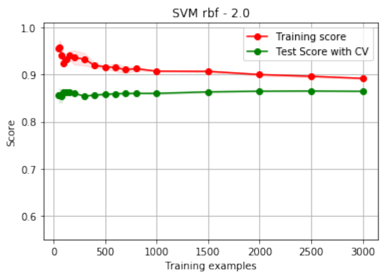

Figure 9: *
RBF SVM w/ 2.0 gamma

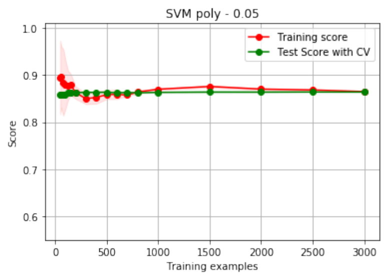

Figure 10: *
Poly SVM w/ 0.05 gamma

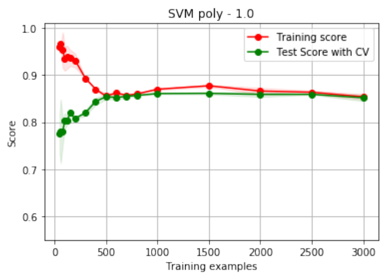

Figure 11: *
Poly SVM w/ 1.0 gamma

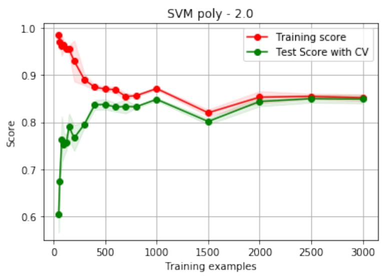

Figure 12: *
Poly SVM w/ 2.0 gamma

- **Analysis**
  The Support Vector Machines performed well compared to the other algorithms, though, on-average, took significantly longer to train. A gridsearch was used along with cross-validation to build a robust and efficient model since there were lot of parameters to tune. The two parameters were Kernel and Gamma. For each dataset, 'rbf' and 'poly' kernels were used. A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space. While the radial basis function kernel can map an input space in infinite dimensional space. For both datasets, the rbf kernel proved more effective.Gamma is the Kernel coefficient for 'rbf' and 'poly' and to tune it, values on a spectrum from 0.01 to 2.0 were used.
  Overall, the optimal configurations yielded a testing rate of 74% for the housing price data and 87.71% for US Visa Application Data. For the Housing Dataset, both rbf and poly for lower gamma tends to underfit and have high bias. As you increase the value of gamma, they tends to overfit which hints if they are could have been more data samples the model could have performed better. Rbf does a much better job at securing a robust fit as compared to poly kernel. For 0.05 gamma, poly kernel has many missclassifications in the training set and many in the test set which shows that it might might have high bias is probably underfitting data.
  For the US Visa Application Data, the training time increased significantly as it is a larger dataset. It is clearly because SVMs require a large amount of iterations and recalculation to converge to an optimal solution. The training time took approximately 30 seconds for the various configurations where as the smaller housing price dataset took less than a second on average. From the learning curve, we see the SVM becomes too rigid to the input data, and is not robust. When the kernel has too low of a gamma and does not accommodate feature relations properly, the model underfits the training and test data.

3 K-Nearest Neighbors

- **Housing's Sale Price Data**

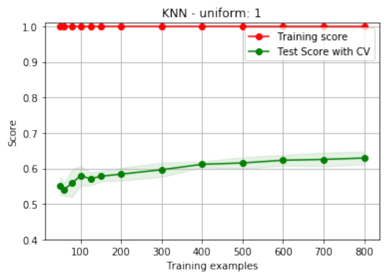

Figure 13: Uniform KNN - 1

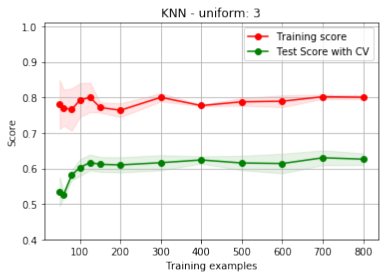

Figure 14: Uniform KNN - 3

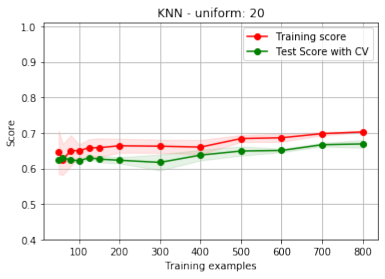

Figure 15: Uniform KNN - 20

Weight	K	Train %	Train Time	Test %	Test Time
uniform	1	1.0000	0.1122	0.7123	0.0020
uniform	2	0.8554	0.1240	0.6918	0.0051
uniform	3	0.8554	0.1378	0.7192	0.0051
uniform	4	0.8265	0.1604	0.7192	0.0040
uniform	5	0.8158	0.1441	0.6918	0.0024
uniform	10	0.7960	0.1765	0.6918	0.0062
uniform	15	0.7778	0.2043	0.6781	0.0094
uniform	20	0.7610	0.2064	0.6712	0.0067
uniform	30	0.7496	0.2347	0.6918	0.0117
uniform	50	0.7314	0.2758	0.6781	0.0104
distance	1	1.0000	0.0964	0.7123	0.0023
distance	2	1.0000	0.1358	0.7123	0.0040
distance	3	1.0000	0.1426	0.7329	0.0044
distance	4	1.0000	0.1565	0.7260	0.0136
distance	5	1.0000	0.1628	0.7397	0.0055
distance	10	1.0000	0.2358	0.7260	0.0117
distance	15	1.0000	0.2172	0.7055	0.0065
distance	20	1.0000	0.2233	0.6849	0.0060
distance	30	1.0000	0.3582	0.6986	0.0095
distance	50	1.0000	0.4440	0.7055	0.0152

Table 3: Train%, Training Time, Test%, Testing Time for various values of K

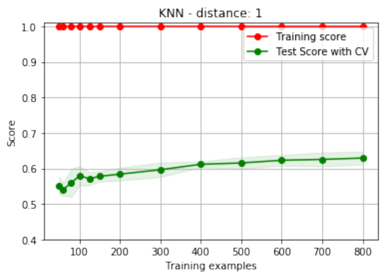

Figure 16: Distance KNN - 1

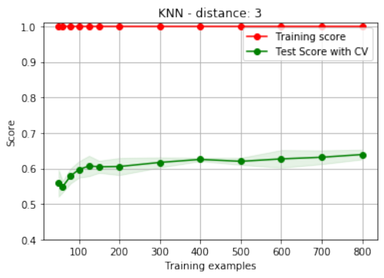

Figure 17: Distance KNN - 3

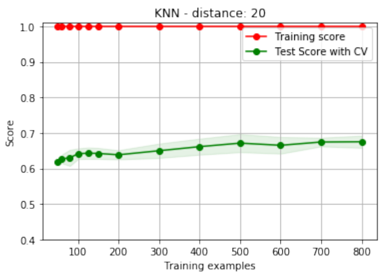

Figure 18: Distance KNN - 20

- **US Visa Applications Data**

Weight	K	Train %	Train Time	Test %	Test Time
uniform	1	0.9694	2.5381	0.8152	0.0423
uniform	2	0.8959	2.1247	0.7478	0.0683
uniform	3	0.9059	2.4009	0.8525	0.0728
uniform	4	0.8920	3.9149	0.8289	0.1282
uniform	5	0.8902	4.1302	0.8459	0.1092
uniform	10	0.8766	5.6508	0.8591	0.0988
uniform	15	0.8733	5.6460	0.8602	0.1172
uniform	20	0.8725	5.8509	0.8580	0.2103
uniform	30	0.8684	7.9636	0.8618	0.2268
uniform	50	0.8670	9.8560	0.8635	0.2618
distance	1	0.9694	2.3761	0.8152	0.0617
distance	2	0.9649	3.0030	0.8004	0.0995
distance	3	0.9770	3.4344	0.8438	0.0922
distance	4	0.9756	3.5783	0.8443	0.1186
distance	5	0.9772	3.9035	0.8459	0.1246
distance	10	0.9757	4.9010	0.8596	0.1220
distance	15	0.9777	5.9040	0.8635	0.1802
distance	20	0.9777	6.4120	0.8624	0.1732
distance	30	0.9777	7.6746	0.8657	0.2194
distance	50	0.9777	9.9897	0.8679	0.3249

Table 4: Train%, Training Time, Test%, Testing Time for various values of K.

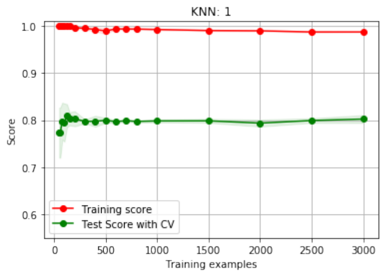

Figure 19: Uniform KNN - 1

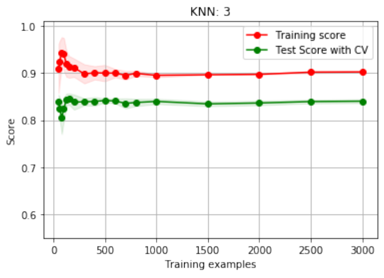

Figure 20: Uniform KNN - 3

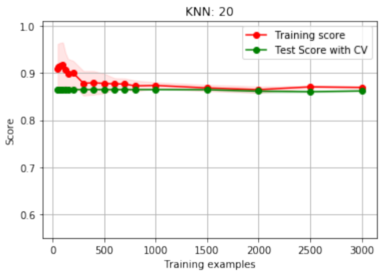

Figure 21: Uniform KNN - 20

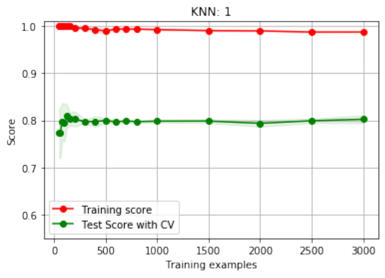

Figure 22: Distance KNN - 1

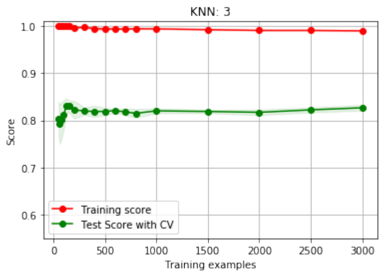

Figure 23: Distance KNN - 3

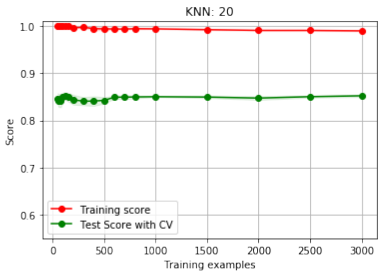

Figure 24: Distance KNN - 20

- **Analysis**
  K-Nearest Neighbors is one of the simplest model employed on both the data sets. Two different weight functions were used for prediction. First one is 'uniform' which means that all points in each neighborhood are weighted equally. Second one is 'distance', which means closer neighbors of a query point will have a greater influence than neighbors which are further away. In terms of training and testing time, it was extremely fast for both data sets since K-NN is a lazy learner and mostly requires log n time for querying.
  For the housing's sale price, the model worked relatively poorly because of its smaller size which led to over-fitting. It tend to perform worse using k-NN due to the smaller number of items to train on and the greater likelihood of values from a different class being taken into account as a neighbor The 'distance' function did not perform due to over fitting the data as training error was very high while testing score was low. This could be possibily be due to smaller size of the dataset. While using the 'uniform' function, it was better than 'distance' function. But as we decrease the value of K to 1, our predictions become less stable and you could clearly observe over-fitting of the data. However when we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point).The high variability in the training score hints at high bias. You could observe comparatively low training score but it is still better than other two charts. If the model could have been trained with more data, the results could have improved. Moreover at K=20, since there is many classifications both in training set as well as testing set there must be high bias and under-fitting of the data could be observed.
  In the US Visa Data, on working with uniform function it performs really well due to large dataset. For K=1, the bias is low and variance is high so it is visible that overfitting of data is taking place. On increasing K, predictions becomes more stable and results into good training and testing score with few misclassifications in training and test set. For K=20, it is visible that both the curves converges which means it does not require any data cases and underfits as well. For distance function, initially overfitting is observed but the performance improves as the value of K increases to a certain extent.

4 Decision Trees

Various values of max depth were tested as a means of pruning unnecessary leaves. Similarly, a grid search was used to test whether a 'gini' or 'entropy' criterion was more effective.

- **Housing's Sale Price Data**

Depth	Criterion	TreeSize	Train %	Train Time	Test %	Test Time
1	gini	3	0.0912	0.0355	0.0890	0.0006
3	entropy	15	0.5816	0.0401	0.5890	0.0003
6	entropy	91	0.6820	0.0442	0.6438	0.0003
10	gini	331	0.8575	0.0566	0.6918	0.0006
15	gini	593	0.9854	0.0695	0.7603	0.0003
20	gini	643	1.0000	0.0559	0.7603	0.0007
25	gini	639	1.0000	0.0550	0.7603	0.0006
35	gini	639	1.0000	0.0570	0.7603	0.0006

Table 5: Results at multiple depths for best criterion via grid search

	0-1	1-2	2-3	3-4	4-5
0-1	2	6	0	0	0
1-2	4	87	7	0	0
2-3	0	8	15	1	2
3-4	0	0	6	3	0
4-5	0	0	2	0	3

Table 6: Test Data Confusion matrix(classes in 100ks)

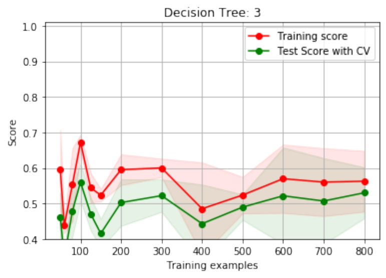

Figure 25: Learning Curve for max_depth = 3

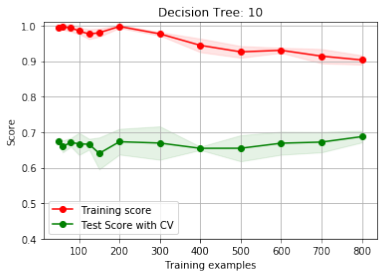

Figure 26: Learning Curve for max_depth = 10

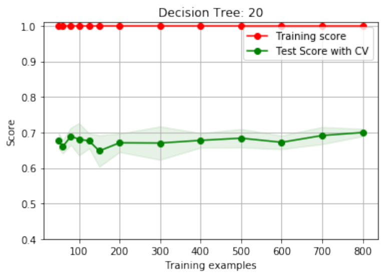

Figure 27: Learning Curve for max_depth = 20

- **US Visa Applications Data**

hline **Depth**	**Criterion**	**Tree Size**	**Train %**	**Train Time**	**Test %**	**Test Time**
1	gini	3	0.7657	0.1212	0.7680	0.0010
3	gini	15	0.6775	0.1417	0.6749	0.0009
6	entropy	105	0.7121	0.2605	0.6924	0.0011
10	gini	889	0.7584	0.3192	0.7072	0.0011
15	gini	3013	0.8628	0.4199	0.7582	0.0014
20	gini	4751	0.9299	0.4517	0.7917	0.0010
25	gini	5349	0.9545	0.5284	0.8032	0.0010
35	entropy	5349	0.9574	0.5116	0.8081	0.0010

Table 7: Results at multiple depths for best criterion via grid search

	Accepted	Denied
Accepted	96	135
Denied	217	1376

Table 8: Test Data Confusion Matrix

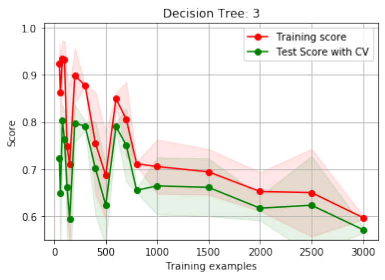

Figure 28: Learning Curve for max_depth = 3

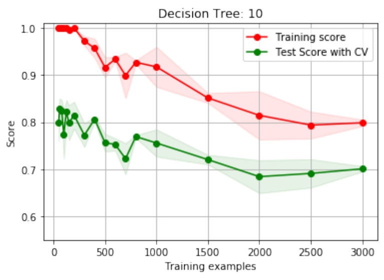

Figure 29: Learning Curve for max_depth = 10

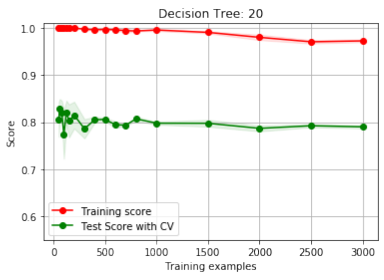

Figure 30: Learning Curve for max_depth = 20

- **Analysis** Looking at the bigger picture, decision trees worked quite well for both datasets but was prone to overfitting. On exammining the results of the classifier on both datasets will give better insights and basis for analysis.
  The learning curve shown above clearly indicates that the size of the training set affects the performance of the algorithm. The fact that the more we train our data, the more likely it is to witness the variablity in the data. This makes the algorithm less biased. Therefore, if we only train on a few data samples, then our algorithm tends to underfit the data because it will try to make decisions based on the features learned from small sample size. The learning curve for both datasets smooths out as the size of the data samples taken for training increases, depicting a more informed model with less variance. Moreover, cross validation normalizes the training samples and learning curve. If cross validation would not have been undertaken, the model was prone to unrepresentative, biased dataset samples during training and testing.
  Maximum tree depth and split criteria were the two parameters tuned for learning. The function to measure the quality of a split is known as split criteria. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Both these criteria were tested and evaluated using a gridsearch. Gini measurement is the probability of a random sample being classified incorrectly if we randomly pick a label according to the distribution in a branch.Entropy is a measurement of information. The other parameter refers to the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min samples split samples. For most of the trials run for the datasets, 'gini' proved to be more effective splitting criteria though the results for the two were nearly identical. Since 'Gini' is mathematically simpler in nature, it doesn't require me to compute logarithmic functions, minimizes classification and will tend to find the largest class. The depth of the tree poses a significant impact over the performance of our model.
  In the housing dataset, on increasing the tree depth, we are actually creating a more complex tree with more possible decisions. This makes the tree way more deep and it begins to overfit with very specific branches for very specific data items. Similarly in the US Visa application data, on increasing the depth overfitted the data.
  In terms of training time, housing prices took longer than the visa data to train. This was mainly due to a larger dataset size. As the depth of the tree increased, the training time also took longer for both datasets due to the increased tree size and complexity. In terms of accuracy, the visa data started comparatively high but was able to gain about 4% through optimizing the depth of the tree. The reason could be 'salary' and 'job type' proved to heavily influence the approval process.In the The housing data, I saw more greater improvements as depth increased. As a more diverse and variable dataset, the model was able to gain a lot of accuracy as the tree grew because it could accommodate more specific featurebranches. While the test data started out at <10%, it was able to grow to approximately 76%.

5 Boosting

For Boosting, Adaboost which is a boosted decision tree classifier is used. It is using the same base decision tree as the above decision tree. Based on the results of the earlier model, I would continue with 'gini' as the criterion for faster training.

- **Housing's Sale Price Data**

Depth	Learning Rate	# Estimators	Train %	Train Time	Test %	Test Time
1	0.1	15	0.6881	6.9413	0.6918	0.0018
3	0.1	5	0.7724	8.2909	0.7945	0.0013
5	0.1	3	0.8291	10.8559	0.7740	0.0011
10	1	50	1.0000	13.6713	0.7945	0.0057
15	1	100	1.0000	8.7526	0.8014	0.0145
20	0.1	15	1.0000	0.6862	0.6781	0.0009

Table 9: Results at multiple depths for best learning rate/ estimators via grid search

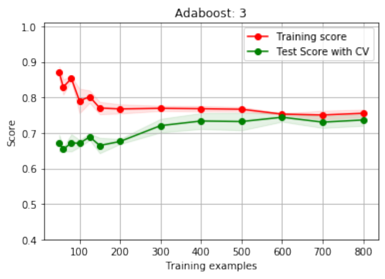

Figure 31: Learning Curve for max_depth = 3

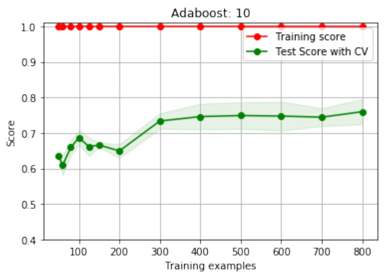

Figure 32: Learning Curve for max_depth = 10

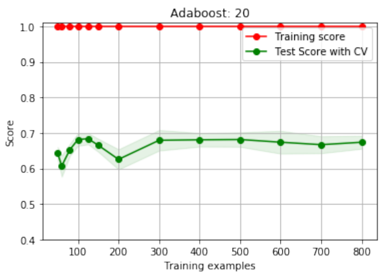

Figure 33: Learning Curve for max_depth = 20

- **US Visa Applications Data**

Depth	Learning Rate	# Estimators	Train %	Train Time	Test %	Test Time
1	0.1	150	0.8705	24.1484	0.8662	0.0327
3	0.1	100	0.8770	39.7266	0.8679	0.0216
5	0.1	5	0.8761	53.8227	0.8706	0.0021
10	0.1	150	0.9775	82.2192	0.8745	0.0504
15	1	150	0.9775	94.5996	0.8701	0.0420
20	1	150	0.9775	107.0742	0.8618	0.0452

Table 10: Results at multiple depths for best learning rate/ estimators via grid search

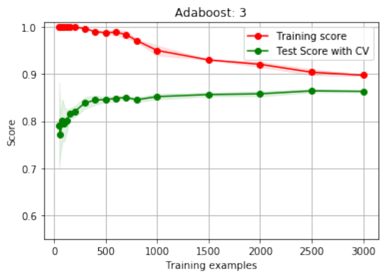

Figure 34: Learning Curve for max_depth = 3

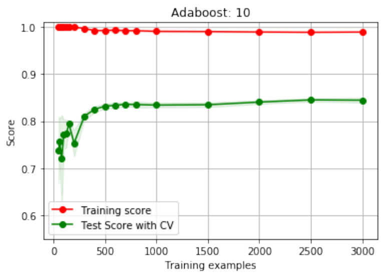

Figure 35: Learning Curve for max_depth = 10

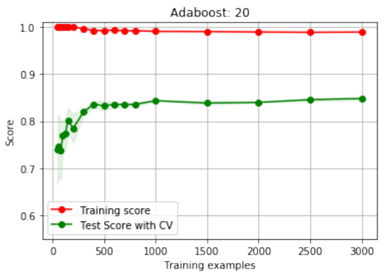

Figure 36: Learning Curve for max_depth = 20

- **Analysis**

I am using the AdaBoost boosted tree classifier which uses the decision tree as its base classifier. A boosted tree is basically an ensemble of weaker trees. A boosted tree classifier works with better accuracy than the original decision tree. On looking at the numbers in the table, it is clearly visible that for visa applications data, it is approximately 7% better in the optimal configuration. In the housing data, it is approximately 4% better.
However, the training time of boosted tree classifier is significantly greater than the original configuration and this clearly understandable from the the fact that boosted trees are ensemble which will possibly take more time. In the US Visa Dataset, it took me more than a minute to train for depths greater than 5.
On considering tuning of parameters, gridsearch is used to tune learning rate, depth, and other estimators. As of estimators increased, optimal learning rate tended to increase too. However,fewer estimators were needed for a deeper tree because the optimal learning rate was achieved sooner.

While both types of trees performed well on the datasets, AdaBoost performed exceptionally well. The AdaBoosted tree still overclassified at increased depth. By pruning depth to approximately 10 to 15, an optimal tree could be achieved.

6 Artificial Neural Network

For modelling a neural network, I am using ScikitLearn's multi-layer perceptron classifier. With respect to our datasets, a combination of different solvers, learning rates, and scaling was used to observe the functionality of the networks.

- **Housing's Sale Price Data**

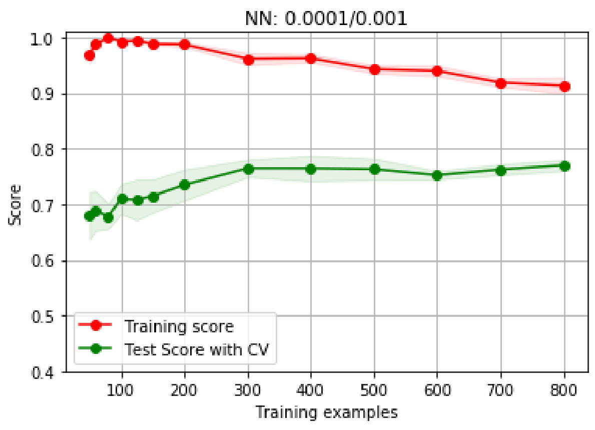

Figure 37: Housing Price NN Learning Curve

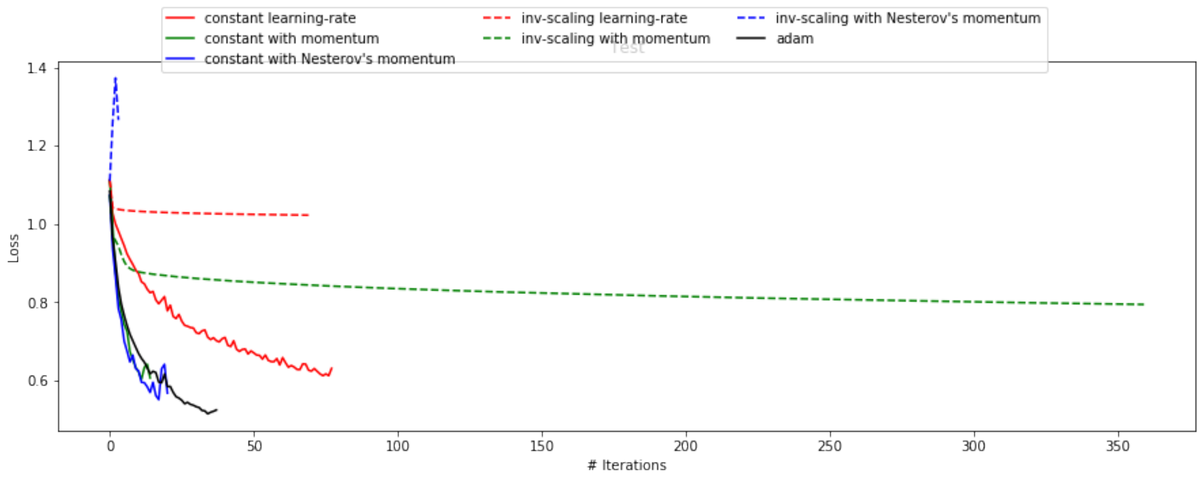

Figure 38: Housing Price NN Configurations Loss Curve

NN Config	Test Score	Loss	Train Time
sgd constant learning-rate	0.7664	0.6301	0.4272
sgd constant learning-rate with momentum	0.7498	0.6058	0.0889
sgd constant learning-ratewith Nesterov's momentum	0.7595	0.5669	0.1286
sgd inv-scaling learning-rate	0.6278	1.0219	0.3932
sgd inv-scaling with momentum	0.7009	0.7936	1.9339
sgd inv-scaling with Nesterov's momentum	0.6278	1.2652	0.0205
adam constant learning-rate	0.7795	0.5237	0.2093

Table 11: Housing Price results for various NN configs

- **US Visa Applications Data**

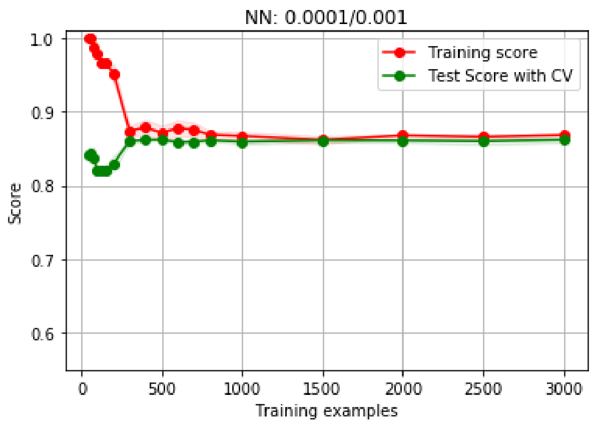

Figure 39: Permanent Visa Applicant NN Learning Curve

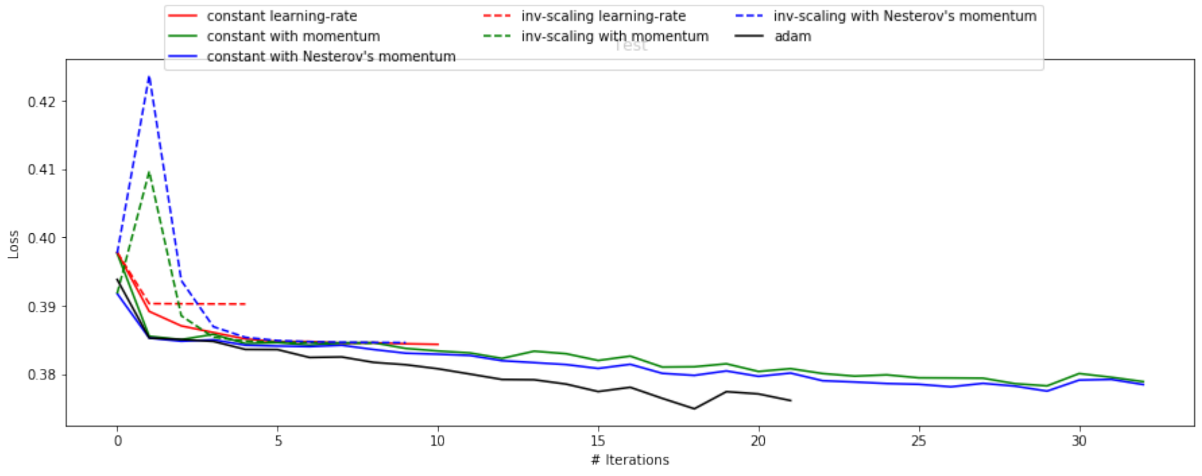

Figure 40: Permanent Visa Applicant NN Configurations Loss Curve

NN Config	Test Score	Loss	Train Time
constant learning-rate	0.8642	0.3842	0.6121
sgd constant learning-rate with momentum	0.8663	0.3823	0.7120
sgd constant learning-rate with Nesterov's momentum	0.8668	0.3779	1.4836
sgd inv-scaling learning-rate	0.8646	0.3905	0.2435
sgd inv-scaling with momentum	0.8646	0.3842	0.3750
sgd inv-scaling with Nesterov's momentum	0.8646	0.3848	0.2740
adam constant learning-rate	0.8677	0.3650	2.1166

Table 12: Permanent Visa results for various NN configs

- **Analysis** The neural network models didn't perform well as compared to then boosted decision tree. After several trials and figuring out suitable configurations it was clear that that the wide variety of neural network configurations performed differently on the given dataset.
  On training the US visa Dataset, it converged rapidly which hints at its low variability. The network model was able to very quickly learn the features–in less than 40–though its feature set is smaller than those of the housing prices. On the contrary, the housing price dataset started to converge slowly in regards to the learning curve. It required many more iterations to learn.
  On the test data, various neural network configuration were used which was created by changing the weight optimizer, learning-rate, momentum, and scaling. To aid with speed and performance, a pre-process Scaler was applied to the datasets. Moreover, a gridsearch approach was applied with various alpha, learning rate, solver, and hidden layer configurations. However, below we will look at some top performers.
  On training the permanent visa dataset, almost every algorithm performed the same with similar learning rates. But i gradually realised since the dataset converges on fewer iterations, it takes less effort and hyperparemeterization to find functional solutions.

For the housing price dataset, the 'adam' weight optimizer significantly outperformed the others. This was quite different than the Permanent Visa dataset, which performed rather homogenously. The 'adam' optimizer, or solver, is a 'stochastic gradient-based optimizer.' (From SciKit learn manual) It is noted to work especially well for large datasets. The learning rate was slightly smaller for this configuration, starting with an initial value of 0.01. Since the housing price network took longer to converge and performed quite differently for the various configuration, it is reasonable to believe that with a larger dataset, a more complex and accurate network could be trained. While this would increase training time, the trade off of increased accuracy and lower variance would likely be possible.