CS7641: ML: Assignment 1

GT Student ID: 903739946, GT account:sadiserla3

Introduction:

In this assignment, I implement five supervised learning algorithms for classification model training on two datasets, tune them to improve the Accuracy, F1, recall, precision scores, and analyze the results. I will compare the performance and fit times of the algorithms

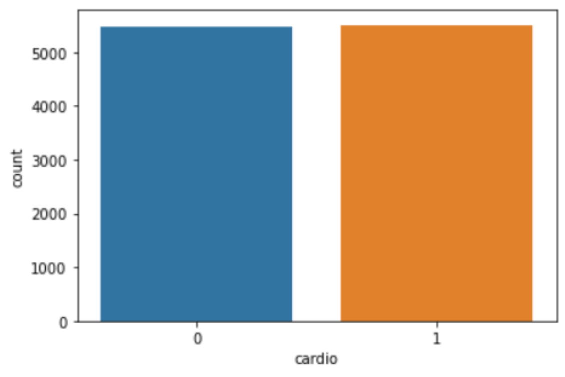
Data Sets Information:

1. Cardiovascular disease dataset -binary classification
2. Wine quality dataset- discrete classification (used as binary classification data for this assignment)

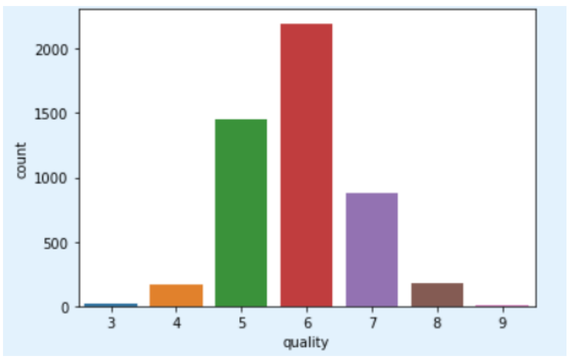
Exciting aspects of the data sets:

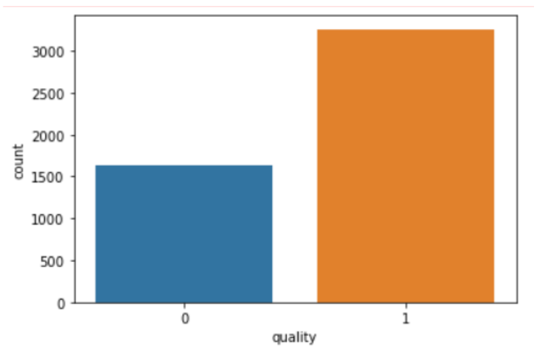
I spent a considerable amount of time figuring out what data fits the definition of exciting data. In this journey, I learned about the specific type of data that serves this assignment's purpose. I filtered out the data with continuous values and picked the above two data sets that can be used for classification. The cardiovascular disease dataset provides the binary classification output (0&1). The wine quality dataset has discrete target values from 1 to 10. To achieve the assignment's goal, modified the output value >= 5 as 1 and below 5 as 0.

I picked the cardiovascular dataset to work on, a dataset based on factual information, and this binary classification prediction has practical usage in the real world. This dataset has 12 features and a binary target class. The number of features are neither too small or too high, and all the features are crucial in predicting the target class. This is a balanced dataset with 11000(approx.) samples. The number of samples are good enough to train the algorithms. Balanced data helps to predict the with higher accuracy and low training error. For the first problem, I decided to have data free of imbalanced data issues like a bias towards the significant class of the data. The below figure shows the data distribution based on the target/class variable(cardio).



The second dataset is the Wine quality dataset. I found this dataset on multiple websites and termed it one of the best classification/regression tasks. As mentioned in the assignment description, I use the same datasets for future assignments; I felt this could be used for regression-based tasks. There are two datasets available, one for white wine and one for red wine. I selected the white wine quality dataset as it has more samples(4900 vs. 1600). This dataset has 12 variables and a discrete target class variable. More features help to train the algorithms better with multiple combinations of variance and bias and strike a balance to find the optimal variance and bias to train the model well. All the features can be used with different weightage and build an approximate polynomial function to improve the models. I wanted to challenge myself to solve the issues with imbalanced data like skewing, overfitting the major class of data. The below diagram shows the binary classification of data.(>= 5 as 1 and below 5 as 0)





Libraries and data clean up information:

- Machine learning algorithms: scikit-learn (python),GridSearchCV(tuning)
- Scientific computing: numpy, Matlab
- Plotting: matplotlib (python), seaborn (python), Matlab
- Data reading: pandas

I read the data from the csv files and checked sample data using the head operation. Checked the target variable data distribution to see if data is balanced/imbalanced. I chose the datasets to balance one of the datasets (cardiovascular), and the other is imbalanced (white-wine quality). As the first significant step, preprocessed the data to improve the quality of the data. There are no null values in the data, or extremely high values used the standard scaling to standardize the data. The cardiovascular dataset was divided into training and test sets in the ratio 75:25, and wine quality data was split into 80:20 in the same order.

Process steps:

- Initialize the classifiers with default values and fit the data
- Print the statistics for Accuracy, F1 scores, ROC curve, confusion Matrix
- Find the best parameters to tune the dataset. In some cases, re-ran with the varied values if optimization is not converged or to find the different combinations to improve the models
- Tune the classifier with the best parameter value identified from the hyperparameter tuning.

- Print the statistics for Accuracy, F1 scores, ROC curve, confusion Matrix
- Plot the learning and validation curves and analyze the results.
- Compare the fit times

Tuning Information of Algorithms:

I used the hyperparameter tuning technique to improve the algorithms. Grid Search is one of the hyperparameter tuning methods to identify the best parameter to train the model. GridSearchCV from the scikit-learn library is used for hyperparameter tuning, with most of the default values. After tuning, different parameter values were selected for each parameter tuning and refitted the data to verify the results.

Decision Tree Algorithm Implementation:

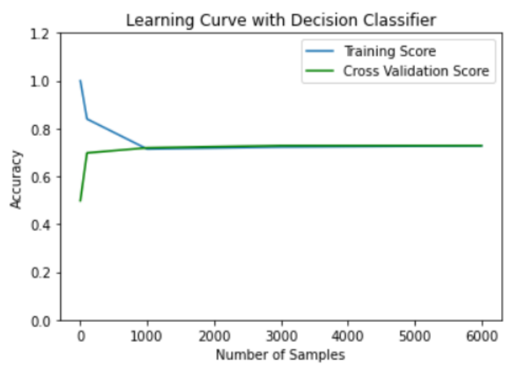
The decision tree is one of the supervised learning algorithms, which falls under inductive learning algorithms. It is a classification algorithm that can divide the data into binary or discrete value output. I use the best attribute at the top to get the best split to classify the target values correctly with minimum depth, which is a greedy approach. I apply a concept called prune the decision tree accuracy of predicting/classifying the data with a minimum decision tree depth.

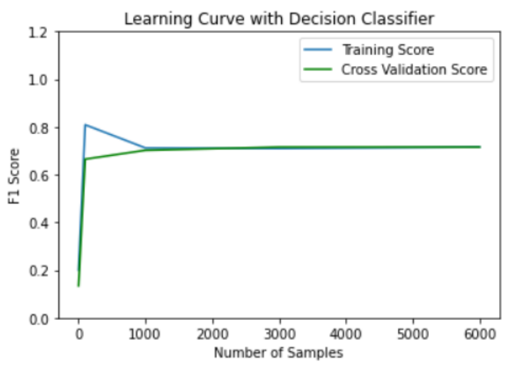
I used the scikit-learn library for this assignment and the DecisionTreeClassifier to classify the data. Parameter entropy is used to achieve information gain to identify the best splitter, splitter used with the value "best" for getting the best possible split. Initially, max_depth is set to None to let the classifier expand the tree to all possible leaf nodes. Selected the default values for the min_weight_fraction_leaf,max_features, min_samples_leaf,min_samples_split not to complicate it at the first go. I verified the various scores like accuracy, f1, recall, plotted roc curve, and the confusion matrix for both datasets.

Cardiovascular dataset:

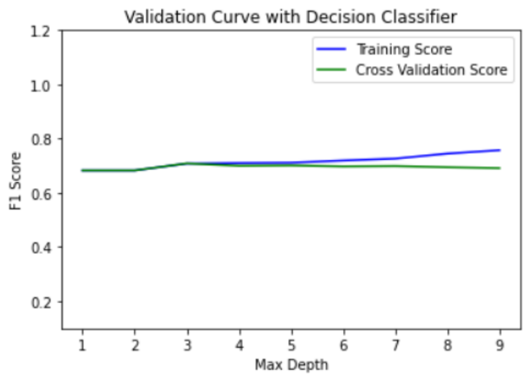
The maximum possible depth of the decision tree has been identified as 47. Using GridSearchSV identified the best value for max_depth for this dataset to avoid the overfitting problem and minimize the tree size. I plotted the learning curve after refitting the data with the max_depth='5'. Accuracy score, f1 score has improved considerably with the hyperparameter tuning from .6 to .71 and .6 to .68, respectively. All other scores also performed well with the hyperparameter tuning. Confusion matrix

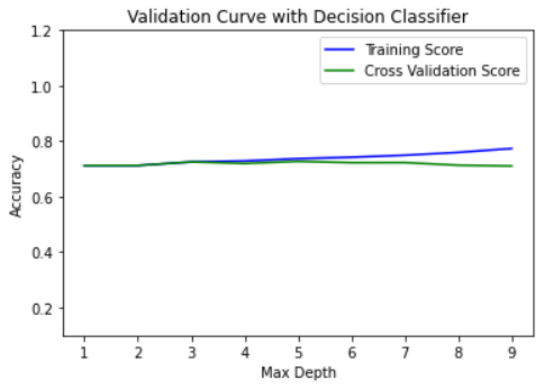
Whereas the Learning curve shows that the Training score initially had an accuracy close to 1, and it eventually settled down at an Accuracy of .71, the validation score initially was around .5. It improved with the number of samples and achieved the same accuracy as the training samples. Accuracy might not have been improved with more samples. The right side graph is drawn with the F1 score on the y-axis as it is the better metric to measure the false negatives and false positives.





From the Validation curve, it is clear that the classifier suffered the overfitting with the training data and achieved the accuracy of 1 as the Decision tree depth increased. However, it started with a score of .7 for the low values for the max_depth. The validation score has gone down considerably as the max_depth value increased, though it started at the same accuracy for the training data. Based on the validation curve, we can conclude that classifying the data with the maximum number of leaf nodes overfits the training data but will not classify the test data as expected. I used the early stopping as a solution to avoid the overfitting problem for this model. I tried to tune the classifier with max_depth value five and found that more are less the same.



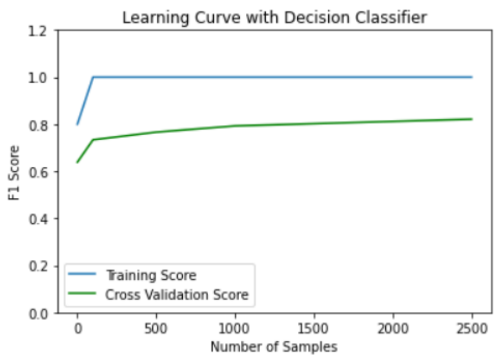


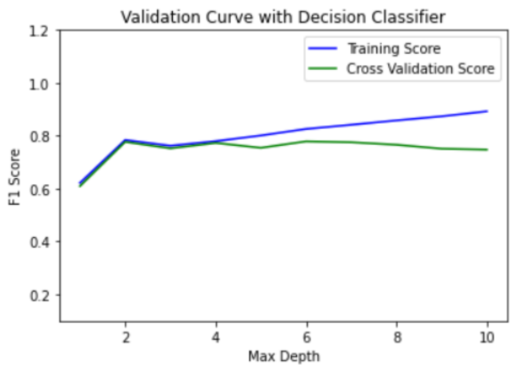
Wine quality dataset:

The maximum possible depth of the decision tree is identified as 26. GridSearchCV identified the best value for the hyperparameter max_depth after the tuning as 22. I used this value to train the DecisionTreeclassifier and refit the data.

The learning curve shows that the DecisionTreeclassifier is overfitting with high variance. The model is not improving as the number of samples increases. It is clear from the overall learning curve that the increasing number of samples will not help.

The validation curve shows that the training and validation scores are the same as the minimum height decision tree, and the model is underfitting. After verifying the results with various max depth values and the effect of depth on the model, we can conclude that the increase in depth of the tree, model is overfitting, and variance increases with the depth.





Neural networks Implementation:

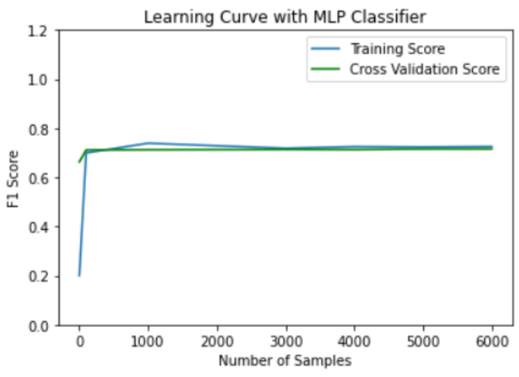
Artificial Neural networks(ANN) can draw analogies from biological neurons in a way neurons transfer the information from one to other when they reach the threshold of energy. ANN has an input layer, hidden layers, and output layer. For binary classification, we can use a perceptron algorithm based on bitwise operators. ANN can be used for solving both classification and regression problems. In this case, they are using ANN for the binary classification of the datasets.

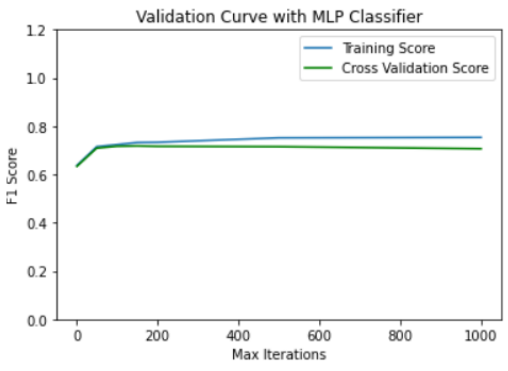
I initialized the MLPClassifer(Multi-Layer Perceptron classifier) with default parameters and two hidden layers(8,4). The number of hidden layer nodes is chosen based on the input features' size and the binary class output variable. The default activation function used is relu, which is easier to train and transforms the input into the binary output of 0,1 for the hidden layers. The default solver value is adam, which is suitable for large datasets with thousands of samples. Default max_iter is set to 200. Upon execution system thrown warning that the convergence is not achieved with the default max_iter

Cardiovascular dataset:

Various scores remained close to ~.72 with the default initialization of the classifier. The confusion matrix shows that there is a high number of false positives. Hyperparameters values for activation: 'relu' and max_iter: 100 have been identified by the tuning process using the gridsearchCV. I used these values to tune the classifier and refit the data.

Plotted the curve using the F1 score rather than the accuracy score as Often F1 score is a better measure than the accuracy score in the case where we need better identification of the false positives and the false negatives compared to overall accuracy. This data needs a better understanding of the false cases, and the F1 score can achieve that. From the learning curve, we can observe that the Training score remained the same at the same value. The validation score started with a low value but eventually reached the value of the training score of ~.72.



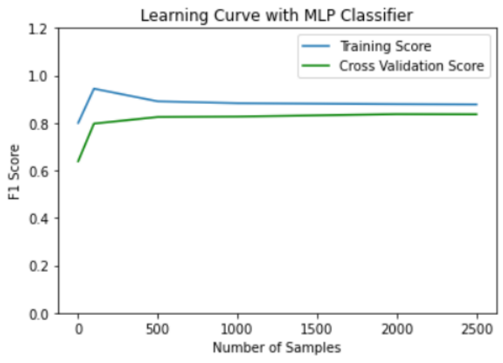


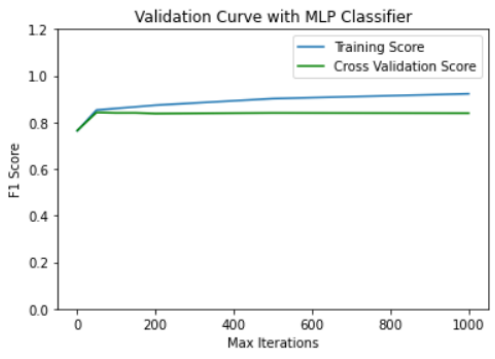
The validation curve shows that both training and Cross-validation scores follow the same trajectory with fewer iterations. They diverge a little bit as the number of iterations increases. The classifier is neither overfitting nor underfitting based on the above two curves. The number of iterations has minimal effect after the optimum value identified from the tuning.

Wine quality dataset:

MLP classifier has been initialized with all the default values and two hidden layers. Neural networks have an impressive f1 score. The f1 score is based on the harmonic mean score of false negatives and false positives as it can be used better to understand the falsely identified data than the accuracy. Optimized the hyperparameters values used are activation='relu', max_iter=20. Instead of using a single hyperparameter for tuning, I went with two parameters, in this case, to see if it improved the model.

Learning curves can tell that both the training and validation scores almost follow the same trajectory before they stay constant as the number of examples grows. The validation curve shows that though they start with the same score at a lower number of iterations, both increase with the number of iterations and remain constant but diverge as the number of iterations reaches 1000. It is evident from the validation curve that the number of iteration will not improve the model much after the tuned max_iter value of 20.





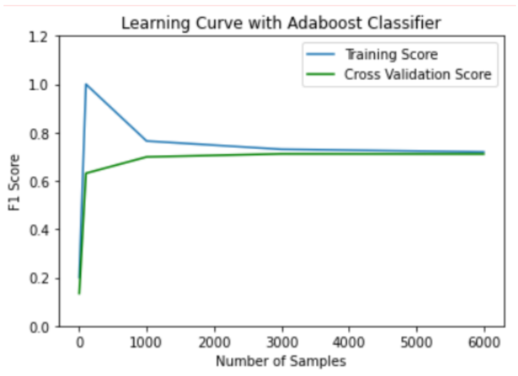
Boosting Implementation:

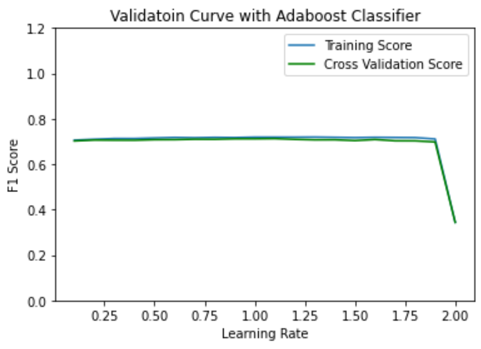
Boosting algorithm is used to improve the weak learning classifiers build a better classifier to classify the data accurately. We can improve the accuracy and reduce bias and variance errors. There are two types of boosting algorithms, Adaboost and gradient boost. I picked the Adaboost algorithm as it is designed for classification problems.

Cardiovascular dataset:

Identified the hyperparameter values {'learning_rate': 1.1, 'n_estimators': 70} from the tuning and applied the values and refit the data for the AdaBoost algorithm. As the learning rate increases contribution of all the algorithms used goes up. N_estimators must be less at a higher learning rate and vice versa for better performance of the Adaboost.

From the learning curve, what stands out is that the training score spiked initially to 1, followed by a gradual decrease in the score and then almost constant after that at the value of ~.7. On the other hand, the Validation score started with a low value and spiked to a value of .6, then increased gradually to ~.7 and then almost converged with the training score. With a low number of samples model is underfitting and has high bias. Adding more samples may not have helped as there is no change with the increasing number of samples.



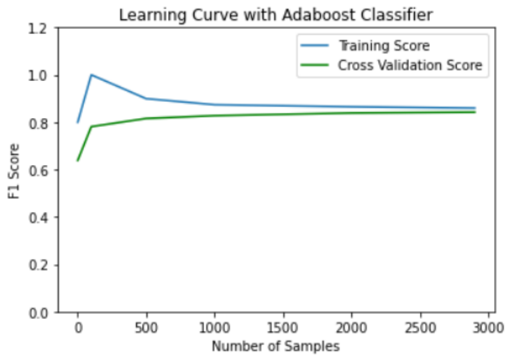


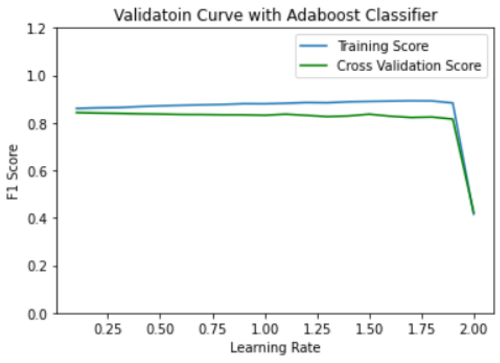
The validation curve has been plotted between the learning rate and the F1 score. We can safely say that the training score and validation score overlap as the learning rate increases. Both validation and training scores almost remained constant as the learning rate changed from 0 to 1.75. Once it crossed this point F1 score and validation score dived, which shows that the model becomes underfitting. We can conclude that with increased learning rate Adaboost algorithm did not perform well.

Wine quality dataset:

I identified the below hyperparameter values are to tune the AdaboostClassifier as 'learning_rate': 0.1, 'n_estimators': 500. The learning curve shows that the Training score initially spikes with fewer samples but reverses the direction as the sample size grows before reaching a plateau. Validation score follows the same path as Training score with a difference of ~.2 before almost converging as the sample size comes to 3000. It is evident that no additional information can be obtained even if we increase the sample size.

The validation curve follows a different path from the learning curve. Initially, training and validation scores start at the same level but diverge very little as the learning rate grows before a sudden drop at a learning rate close to 2. Model suffers high bias as the learning rate increases and model becomes underfitting.





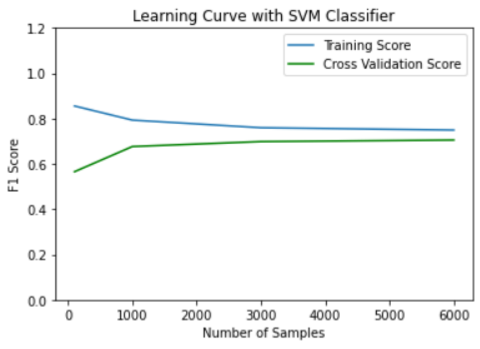
Support Vector Machines Implementation:

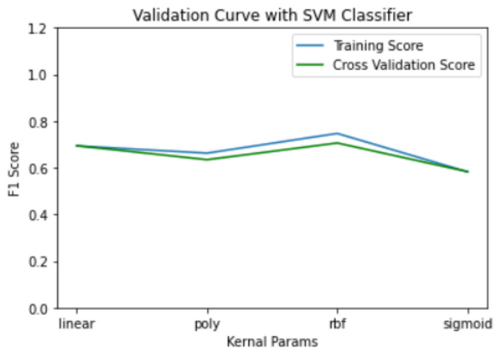
Support Vector Machine is an algorithm that helps to classify the data points using hyperplanes. Hyperplanes can be a line or a plane or can be n-dimensional based on the features. Support vectors are the points close to the hyperplane that affect the hyperplane behavior and help to improve the classifier's margin to classify the data. SVM is better at solving classification problems, while it can solve regression problems as well. I used the SVC classifier as this works fine for few thousand samples. Classifiers sometimes errored out with the message that the class variable does not have more than one value though the class variable has two discrete values. I used the kernel parameter to swap between any of the kernel functions linear, rbf, sigmoid and poly. Identified the best kernel function and C to tune the classifier and plotted the results.

Cardiovascular dataset:

Initialized the Support Vector machine with the default values and fitted the training data. All the scores are almost the same at a value of ~.7. There is a high number of false positives and false negatives. I identified the list of keys that are suitable for hyperparameter tuning. Hyperparameter tuning identified the below hyperparameter value (C : 1)(kernel function tuning parameter is not used here and plotted validation curve with kernel functions) best to improve the SVC classifier. Trying to tune the multiple parameters negatively affected the accuracy, f1, and other scores. Using the above-tuned hyperparameter values reduced the scores drastically down to ~.5 range.

Below Learning, the curve shows that the Training score is at ~.8 with a low number of samples, and it goes down a little bit as the samples increase, stays the same until it reaches ~3000 samples, and goes down a little more after that. With a low number of samples, the model has high variance. The validation score starts at ~.6, increases as opposed to the training score, and remains almost steady before converging with the training score as the sample size grows.



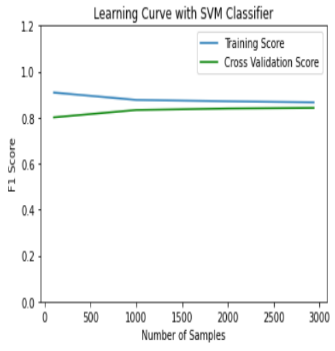


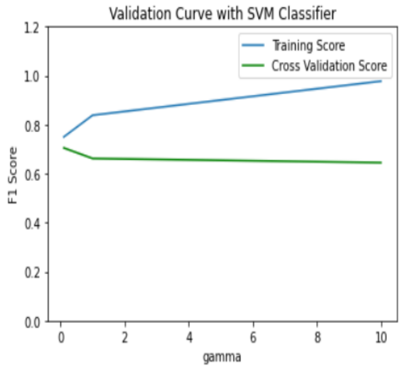
The validation curve shows the training and validation scores for kernel functions linear, poly, rbf, and sigmoid. Linear and Sigmoid show that the training and validation coincide with each other, while with poly, the model becomes a little underfitting and with rbf scores improve, but there is an increase in the variance.

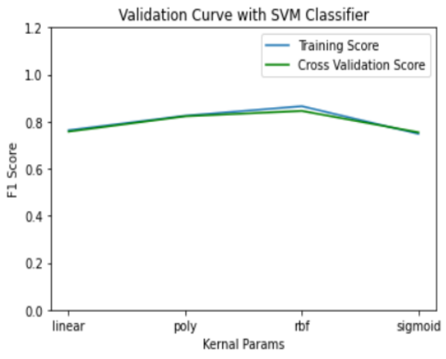
Wine quality dataset:

I applied the below best hypermeter values from the tuning 'C': 1, 'degree': 1, 'max_iter': 1000 on the SVC classifier to improve the binary classification and plotted below learning and validation curves.

The learning curve indicates that both training and learning scores start with little difference but eventually converge with the number of samples grows. At the same time, scores slightly change with the number of samples. The validation curve is plotted between the gamma and F1 scores. As the gamma value increases, we can see that the model is overfitting with the high variance. A model performs well with rbf, though it has more variance, whereas with the linear, poly, and sigmoid kernel params, there is low variance and low score compared to rbf.





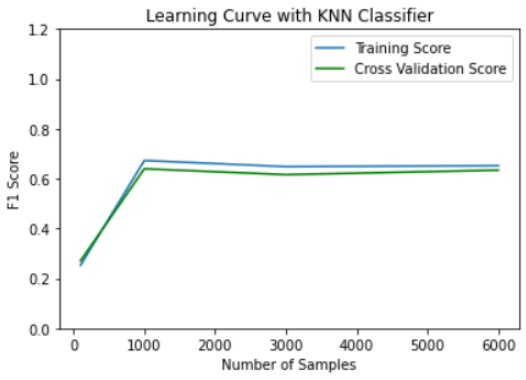


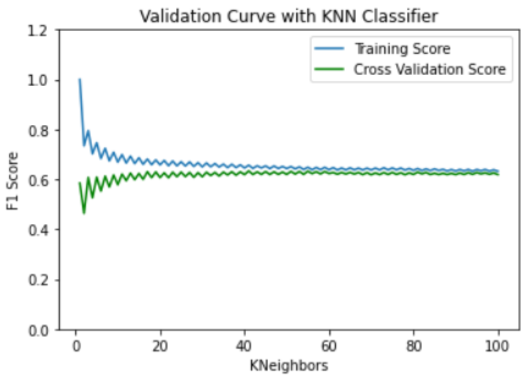
k-nearest neighbors Implementation:

k-nearest neighbors algorithm works on the principle of similarity and distance. Similar things that are close to each other can be grouped. This algorithm works for both classification and regression.

I used the KNeighborsClassifier, which is based on the k nearest neighbors to the test point. With the default parameters Accuracy, f1, and all other scores remain close to ~.63. The confusion matrix shows that there is a very high number of false positives and false negatives. Using the GridSearchCV found out that the optimum value for the n_neighbors is 55. After refitting the data with hyperparameter from the tuning, the classifier did not improve much.

Cardiovascular dataset:





KNClassifer started with a meager training score and spiked rapidly as the sample size grew to ~1000. Based on the scores, the model suffers from high bias and underfitting with a low number of samples. The training score remained almost constant after that. Initially, the model suffered from underfitting but improved with many samples.

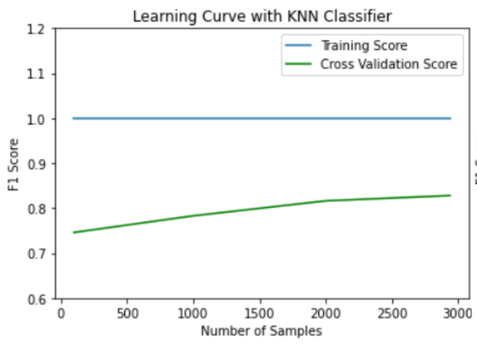
The validation curve shows that the Kneighborsclassifier starts with high bias and overfitting with low values of K. However, the model becomes stable with higher values of K, but it stabilizes with a low F1 score. Overall KNN model did not perform well.

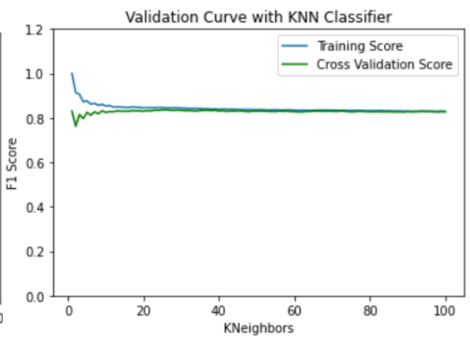
Wine quality dataset:

Initialized the KNeighblorsClassifer with the default param values and then improved the classifier with the hyperparameter tuning. Used the n_neighbors value as 1 and 26 and drawn the learning curve and validation curves

The learning curve shows that the training score is remained constant at the value of 1 and did not change with the number of samples, whereas the validation score improved with the number of samples a little bit, but ~.2 below the training score. This model is a clear example of overfitting. It did not improve with more number of samples.

The validation curve shows that the training score initially starts at 1, however it goes down before settling down at a constant value. This model worked well with increasing values of K with the high true positive rate of around .8





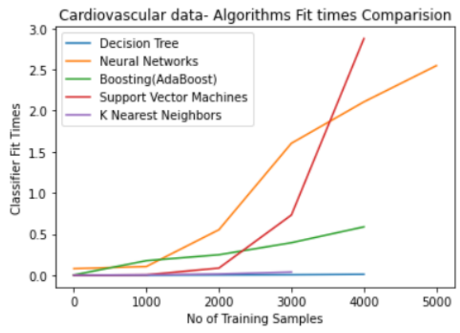
Cardiovascular dataset:

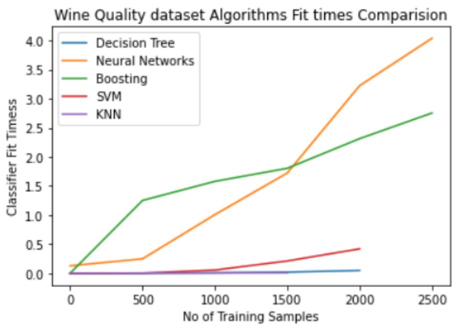
Classifier	Accuracy	F1 Score	Recall Score	Precision Score
Decision Tree	0.71	.68	.64	.73
Neural Network	0.71	.69	.65	.73
Boosting	0.71	.68	.61	.76
SVM	0.716	.69	.65	.74
KNN	.63	.61	.60	.63

White Wine quality dataset:

Classifier	Accuracy	F1 Score	Recall Score	Precision Score
Decision Tree	.79	.84	.84	.84
Neural Network	.76	.83	.87	.80
Boosting	.75	.82	.87	.78
SVM	.77	.83	.88	.79
KNN	.75	.82	.86	.78

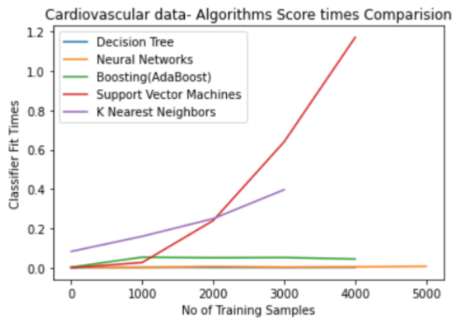
Comparison of fit times:

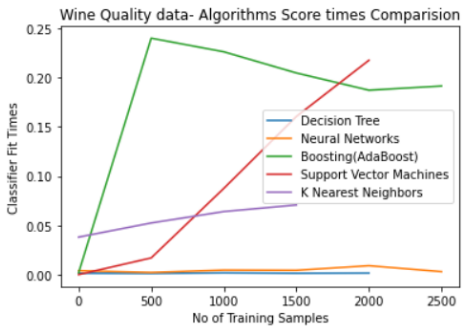




From the above fit times graphs, we can see that the Decision tree & KNN performed better than other models for Cardiovascular data. When it comes to Wine Quality data Decision Tree and SVM performed better

Comparison of score times:





From the above score times graphs, we can see that Neural networks and Boosting performed better than other models for Cardiovascular data. When it comes to Wine Quality data, Decision Tree and Neural Networks performed better

Conclusion:

To sum up, I used five algorithms for binary classification of the cardiovascular and white wine quality data sets. Tuned each algorithm using the hyperparameter and captured the learning curve and the validation curves. Analyzed the Accuracy and F1 scores before and after tuning the algorithms and how the algorithms are overfitting with high variance, underfitting with a low number of samples. All the algorithms have similar performance for the wine quality dataset. Except for KNN, all other algorithms performed better in the case of the cardiovascular dataset. Based on the fit times, score times, and F1 score for the cardiovascular dataset Neural network is the best one, and KNN performed better for the Wine Quality dataset.

References:

https://scikit-learn.org/stable/modules/generated/scikit-learn .tree.DecisionTreeClassifier.html

https://scikit-learn.org/stable/modules/generated/scikit-learn .model_selection.GridSearchCV.html

https://scikit-learn.org/stable/modules/generated/scikit-learn .neural_network.MLPClassifier.html#scikit-learn .neural_network.MLPClassifier

https://scikit-learn.org/stable/modules/generated/scikit-learn .ensemble.AdaBoostClassifier.html

https://numpy.org/doc/stable/index.html

https://scikit-learn.org/stable/modules/generated/scikit-learn .neighbors.KNeighborsClassifier.html

https://scikit-learn.org/stable/modules/generated/scikit-learn .svm.SVC.html

https://scikit-learn.org/stable/modules/learning_curve.html

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html

https://seaborn.pydata.org/generated/seaborn.countplot.html

https://scikit-learn.org/stable/user_guide.html

https://archive.ics.uci.edu/ml/index.php (dataset)

https://www.kaggle.com/ (dataset)

Mitchell, T. M. (2021). *Machine Learning by Tom M. Mitchell (1997–03-01)*. McGraw-Hill Education; 1 edition (1997–03-01).