CS 7641
Assignment-1 (Supervised Learning)

Muzaffer Estelik
estelik.muzaffer@gatech.edu
GTID : 903473214
https://github.com/MuzafferEstelik/CS7641_Supervised_Learning

Introduction:
The purpose of this project is to explore some techniques in supervised learning. I've tried implementing some simple learning algorithms and to compare the performances of the models.

Datasets :
I've selected 2 datasets from Kaggle:

- **Sloan Digital Sky Survey DR14 Classification of Stars, Galaxies and Quasars:**
  The data consists of 10,000 observations of space taken by the SDSS (Sloan Digital Sky Survey). Every observation is described by 17 feature columns and class column which identifies it to be either a star, galaxy or quasar. It's a multi-class classification problem to predict the target variable ("class").
- **Heart Disease UCI:**
  This database contains 303 samples with 14 features. The aim is to predict the presence of heart disease. It's a binary classification problem (0:no presence, 1:presence).

My full codes and datasets are also available at my github account :
https://github.com/MuzafferEstelik/CS7641_Supervised_Learning

Analysis:
I've implemented five machine learning algorithms:

- Decision trees with some form of pruning
- Neural networks
- Boosting
- Support Vector Machines
- k-nearest neighbors

Experiment-1 (Sloan Digital Sky Survey DR14 Classification of Stars, Galaxies and Quasars):

Before implementing machine learning algorithms, I made a quick exploratory data analysis to see if there's any missing value or any need of data cleaning, etc.
I also scaled the data in order to get more accurate results especially with the models that are using distance formulas (Small&large numbers can be meaningful for human brain, but while making some calculations, they might affect our models badly. For example, square feet (ex:2200sqft, 2100sqft) and room numbers (ex:2,3,4) of a house might be affecting a house price equally but numbers of those attributes should be normalized in order to be used the same model.)
I've splitted the data as train (80%) and test (20%) sets.
Below are the comparisons of my models' performances.

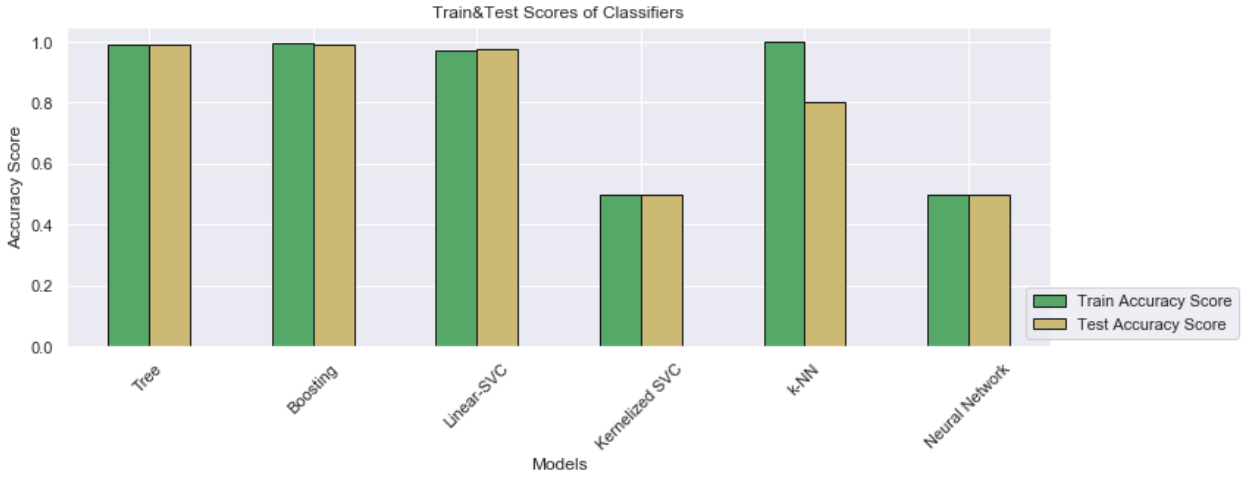

Figure-1: Train and Test Accuracy Scores of the Machine Learning Models

I got the best results with Decision Tree classifier and boosting with decision tree base learner. Linear SVC also performed very well which I've noticed mainly because of the solving overfitting problem by L1 Regularization (Lasso).
I've experimented and compared the regularization's affect by selecting both L1 (Lasso) and L2 (Ridge). According to my experiments, I believe that the low performance of Kernelized SVC and Neural Network classifier of Sklearn is because of lack of L1 regularization parameter. So, I tried dimensionality reduction using PCA and applied again to my Kernelized SVC, k-NN and Neural network learners.

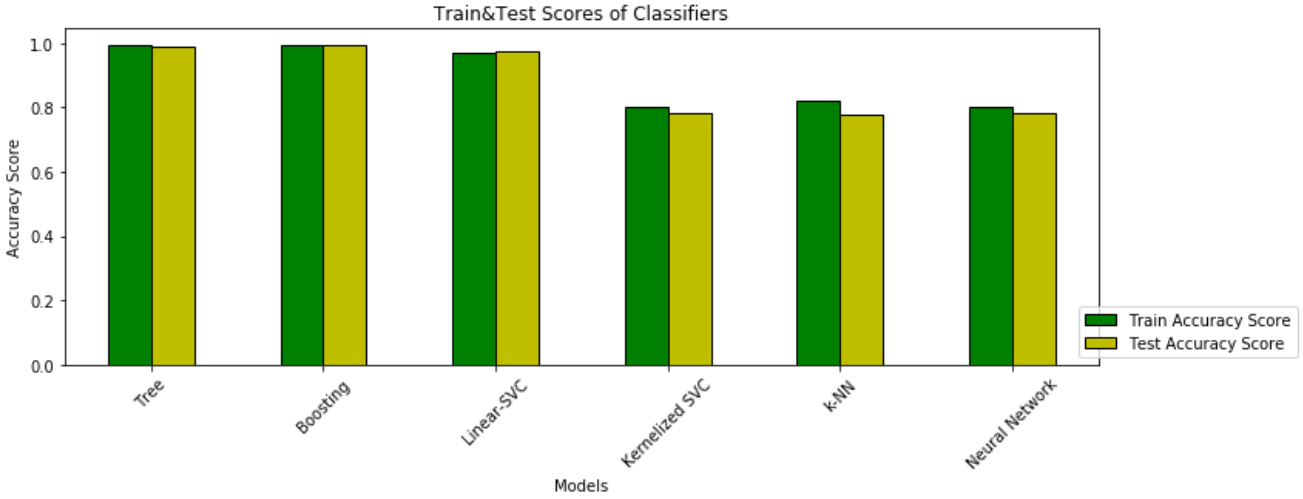

Figure-2: Train and Test Accuracy Scores of the Machine Learning Models
(applied dimensionality reduction for Kernelized SVC, k-NN and neural network)

By reducing the dimension of the dataset, I managed to handle with overfitting problem of Kernelized SVC, k-NN and neural network learners but they couldn't outperform tree based models and Linear SVC.

Figure-3: Train Process Times of Machine Learning Models

Thinking about the sample size (10.000), it's pretty normal to see long process time with boosting, which is creating sub-sets of the data sequentially as per the number of estimators. Similarly, neural network also has high iterative computations. k-NN performed fastest and decision tree classifier is the second fast model as expected. Linear SVC and kernelized SVC models are not that fast but somewhere on an average speed which I believe due to computations for distance calculations and the regularization.

Decision Tree Classifier:

I initialized my model and performed hyper-parameter optimization using the parameters below:

{ 'max_depth': [3, 4, 5], 'criterion': ['gini','entropy'], 'splitter' : ['best','random'], 'min_samples_split' : [2,3], 'min_samples_leaf' :[1, 2], 'min_weight_fraction_leaf' : [0.0,0.001], 'max_features':[None, 'auto', 'sqrt', 'log2'], 'min_impurity_decrease' : [0.0, 0.02] }

I've trained my model by fine-tuning the parameters (edge values were shifted and obtained the best parameters). I've chosen "accuracy" as the evaluation metric but checked the confusion matrix and classification report as well.
I used GridSearchCV with cross validation using 5 folds .

max_depth: The maximum depth of the tree

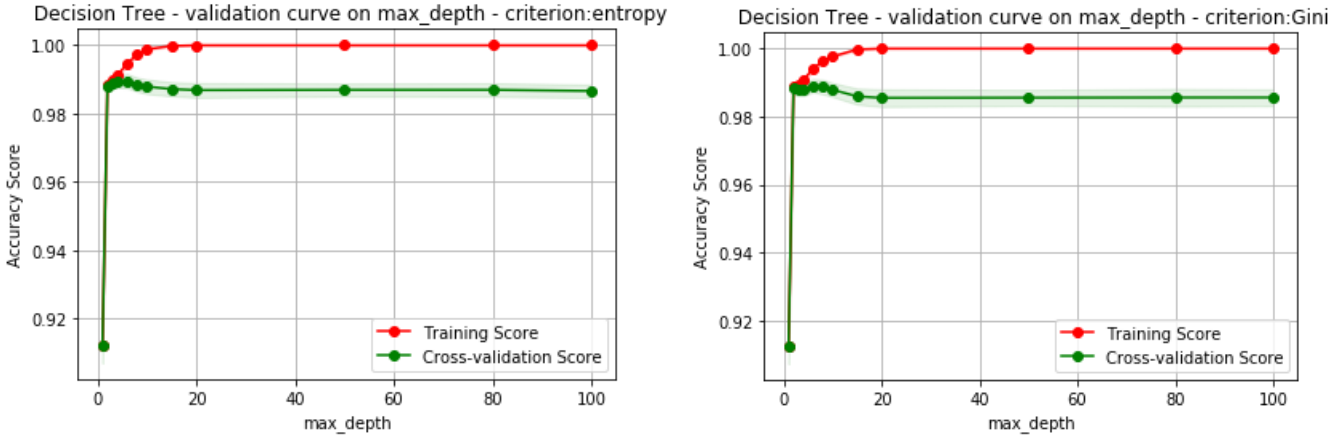

Figure-4: Decision Tree Validation Curve on Maximum Depth with Gini and Entropy Criterion

If we don't define a maximum depth, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. There's no significant difference between the Gini and Entropy criterions which are used calculating the splits. At maximum depth 4, they're providing the best results. It might look like there's overfitting but there is not. When we look at the scale, we'll see that it's almost perfect with around 99% accuracy.

min_samples_leaf: The minimum number of samples required to split an internal node

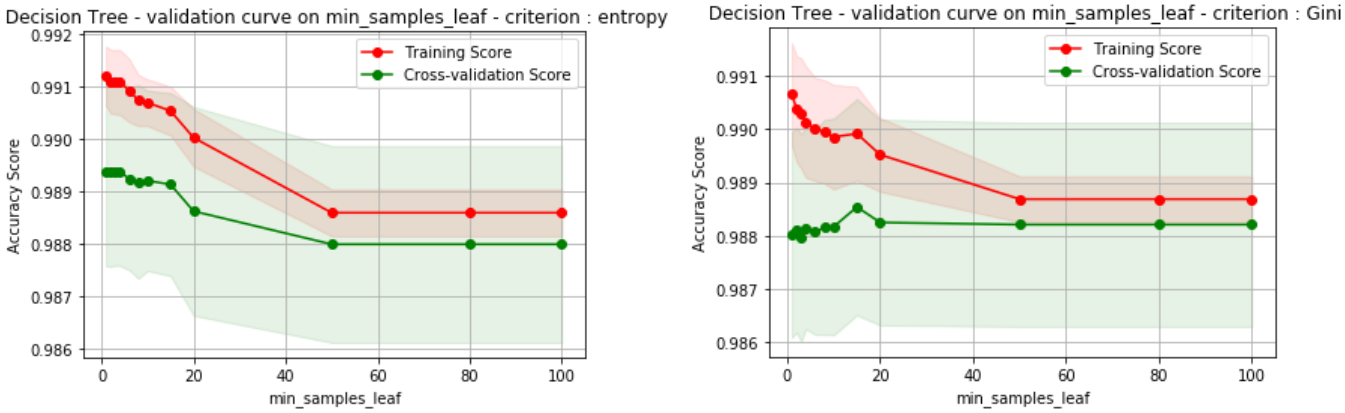

Figure-5: Decision Tree Validation Curve on Minimum Sample Leaf with Gini and Entropy Criterion

min_samples_leaf=1 is providing the best result. As the min_samples_leaf size increases, the model's performance is decreasing due to lack of splits. According to plots, we can say there's no significant difference between Gini and Entropy criterions among min_samples_leaf.

Boosting (AdaBoost Classifier):

I used AdaBoostClassifier for ensemble learner using my previous Decision Tree as weak learner. I made hyper-parameter optimization using below parameters:

{'n_estimators': [45,50,55], 'learning_rate': [0.22,0.23,0.24], 'algorithm' : ['SAMME','SAMME.R']}

After obtaining the best parameters, I experimented different learning rates and n_estimators.

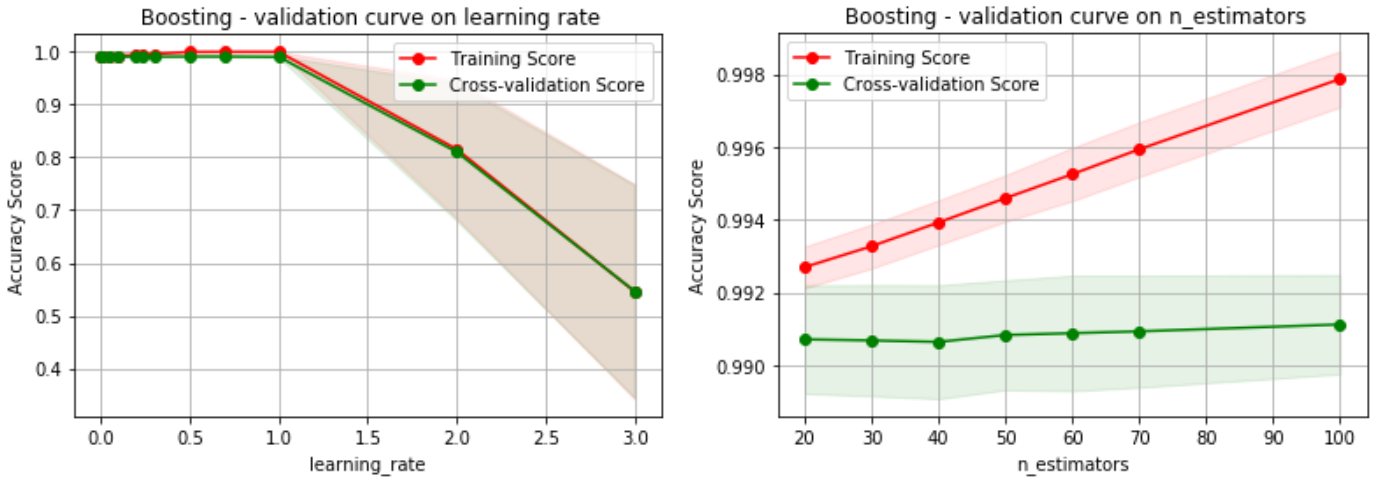

Figure-6: Boosting Validation Curve on Learning Rate and n_estimators.

Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators. I got the best result with learning_rate = 0.23 and n_estimators = 50.

Support Vector Classifier (Linear SVC):

I've used below parameters for hyper-parameter optimization.
{'dual':[True, False], 'fit_intercept': [True, False]}

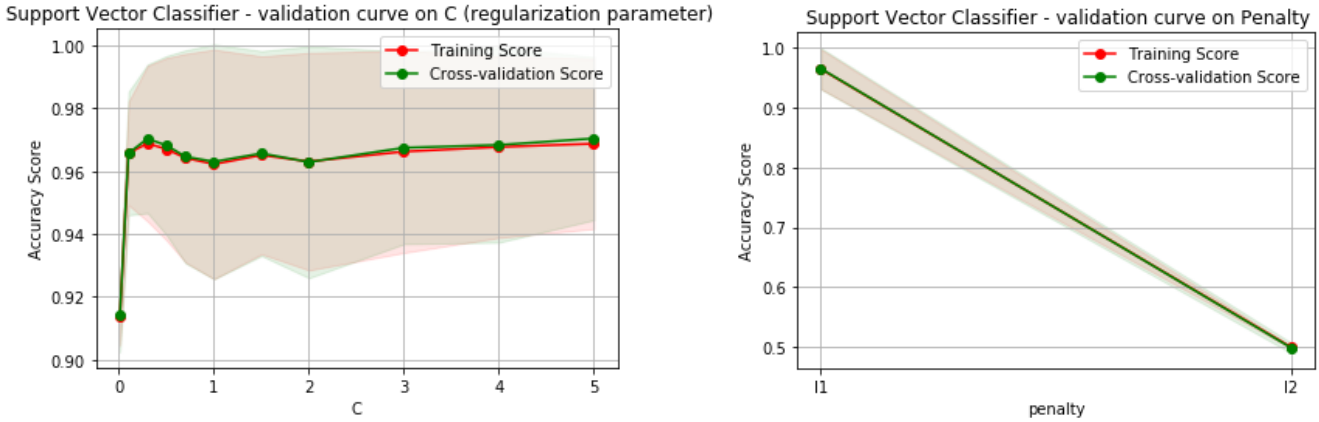

Figure-7: Support Vector Classifier Validation Curve on C (Regularization Parameter) and Penalty

During my experiments, I've used GridSearchCV to obtain the best parameters. But, for this experiment, the model was predicting all the target variables with the majority class' value. I thought about penalizing the model's complexity in order to eliminate some features which were negatively affecting model's performance. For this reason, I've added L1 regularization (Lasso) which uses absolute value of the coefficient and shrinking it towards 0 more aggressively comparing Ridge (L2). Figure-7 Penalty plot (right) is also showing the effect of L1 regularization. I obtained the best results with L1 penalty and C=0.1 .

Kernelized SVC :
I used below parameters for hyper-parameter optimization:

{ 'kernel': ['sigmoid'], 'degree':[0.0001,0.0005,0.0008], 'gamma' : ['scale','auto'], 'C': [1.4,1.45,1.50], 'shrinking':[True,False], 'tol':[0.011,0.015,0.016] }

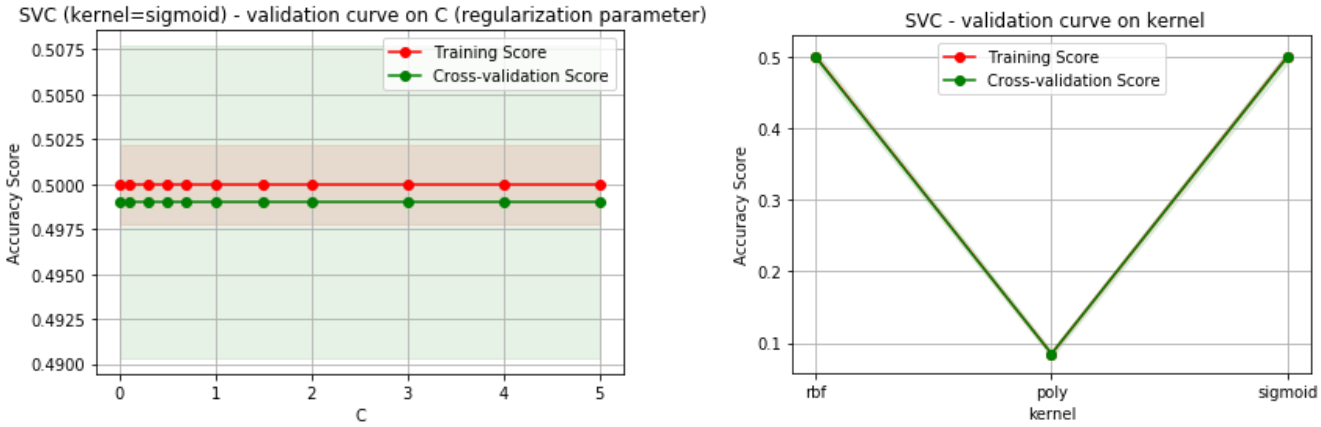

Figure-8: Kernelized Support Vector Classifier Validation Curve on C (Regularization Parameter) and Kernel before PCA
As we experimented before, regularization is the game changer for this dataset. Due to lack of regularization parameter, Kernelized SVC is not able to perform well on this data. So, I tried dimensionality reduction and tried again the performance of the model. I managed to obtain a better result but not outperforming Decision Tree, boosting and Linear SVC.

k-Nearest Neighbor Classifier:
I used below parameters for hyper-parameter optimization:
{'n_neighbors' : [4,5,6], 'weights' : ['uniform', 'distance'], 'algorithm': ['ball_tree','kd_tree','brute'], 'leaf_size':[1,2,3]}

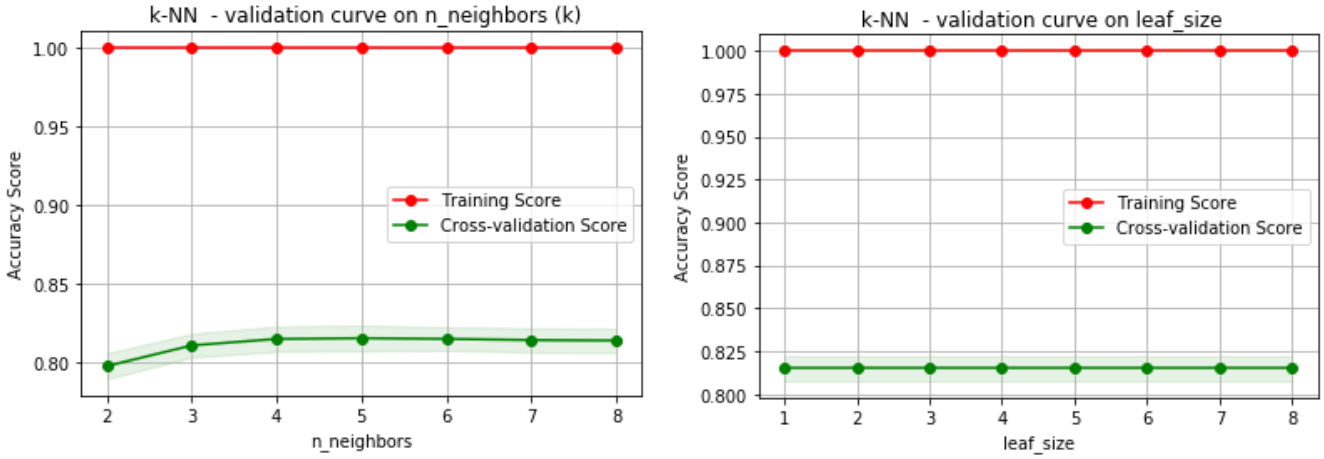

Figure-9: k-NN Classifier Validation Curve on n_neighbors(k) and leaf_size before PCA

The model was overfitting however I tuned the parameters. I tried eliminating some features by using dimensionality reduction (PCA) and applied k-NN again. Reducing dimensionality helped preventing overfitting but not increased the performance of the model.

Neural Networks:
I used below parameters for hyper-parameter optimization:

'hidden_layer_sizes' : [(8,), (16,), (32,), (64,),(100,),(128,)], 'activation': ['relu', 'logistic','identity','tanh'], 'solver':['lbfgs','sgd','adam'], 'alpha': [0.005, 0.001,0.0001], 'learning_rate':['constant','invscaling','adaptive'], 'learning_rate_init': sorted([(2**x)/500.0 for x in range(4)]+[0.000001]) }

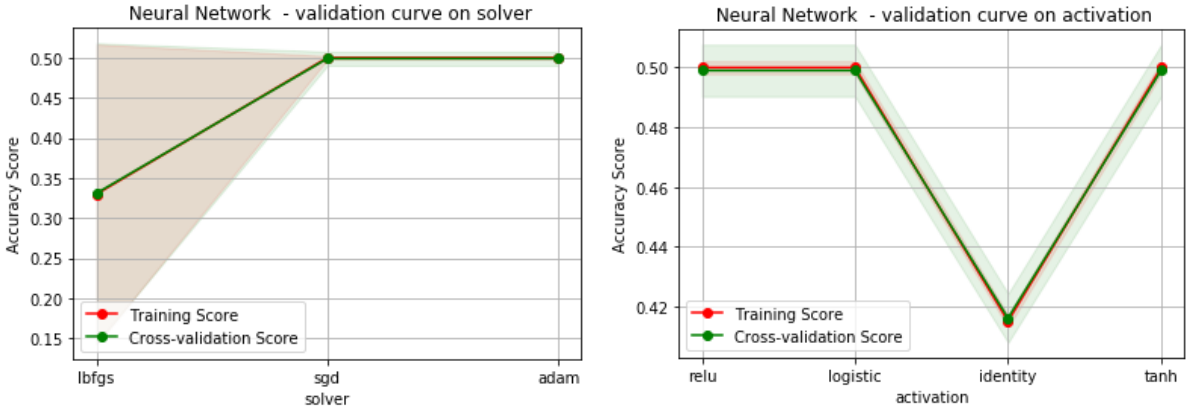

Figure-10: Neural Network Validation Curve on Solver and Activation Function

Figure-10 displays the performances of solvers and activation functions. For this dataset, I got the best results with “Adam” optimizer with “Relu” activation function.
With this model, I faced overfitting problem as well and managed to deal with it by applying dimensionality reduction with PCA.
After PCA, model’s performance increased but was not as good as Decision Tree, Boosting and Linear SVC.

Experiment-2 (Heart Disease UCI):
This dataset has less samples than my experiment-1 which made computations easier. Different from the experiment-1, I decided to use **recall** (TP/(TP+FN) as the evaluation metric due to the characteristic of the data. I've checked the correlation of the features as well in order to make sure, it'll be meaningful to apply machine learning models to extract the relationship between the features. After scaling the features, I started my experiments.

Below are the comparisons of my models' performances. I'll explain each model in detail separately.

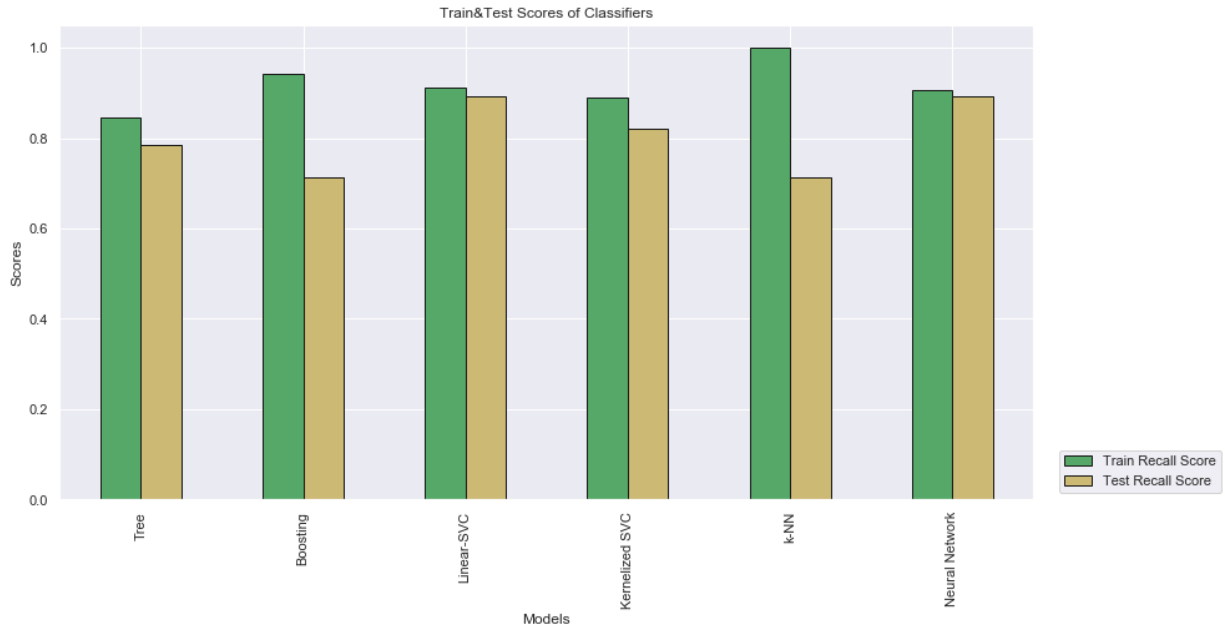

Figure-11: Train and Test Recall Scores of the Machine Learning Models

I got the best results with Linear SVC and Neural Network. Other models are overfitting which I believe due to small sample size. Of course, we should also check the process times in order to evaluate the models' performances.

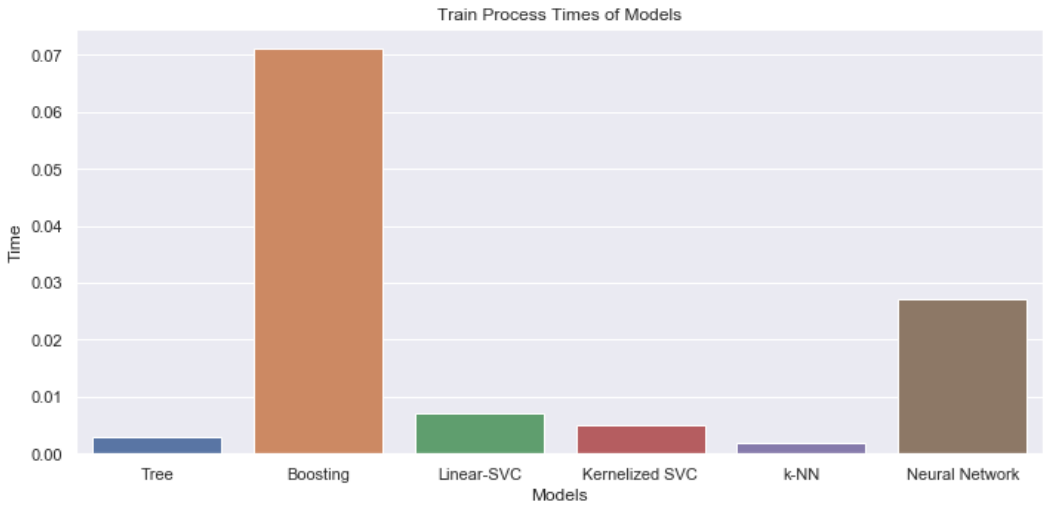

Figure-12: Train Process Times of Machine Learning Models

The high process time of Boosting and Neural Network is pretty normal because we're running the model iteratively within both of them. For boosting, the sub-samples of the weak learner (Decision Tree classifier) is being sequentially trained as per the number of estimators. Similarly, neural network is also running iteratively by back propagating the loss as per number of epochs.

The graphic representation of the models' process times might look like, there's a big difference between the models' performances but, the process times are between 0.002 and 0.07 seconds. Depending on the need (deployment, etc.) we can make a better evaluation about the speeds of the models.

As I explained above, due to the characteristics of the dataset, I've chosen "recall" as the evaluation metric. When we compare the recall scores of the models, we can see that Linear SVC and Neural Network are performing better. Decision Tree, Boosting and k-NN have overfitting. I believe, the main factor is the size of the dataset. We have 257 samples in train and 46 samples within the test set. With small size of samples, it's not easy for the models to train and predict. I've monitored different results with different trials using same parameters and models which I believe because of the small sample size as well especially with randomly initialized calculations.

Now, I'll explain my findings for each model.

Decision Tree Classifier:
I've used below parameters for hyper-parameter optimization and shifted accordingly as I reach edge values.
{'max_depth': [2, 3, 4, 5, 6], 'criterion': ['gini','entropy'], 'splitter' : ['best','random'], 'min_samples_split' : [2, 3, 4], 'min_samples_leaf' :[1, 2, 3], 'min_weight_fraction_leaf' : [0.0,0.01,0.02,0.03,0.04], 'max_features':[None, 'auto', 'sqrt', 'log2'], 'min_impurity_decrease' : [0.0, 0.02, 0.05]}

I used GridSearchCV with cross validation using 5 folds.

After obtaining best parameters, I fit the model to my train set and started making some experiments with some parameters. I used cross validation with 50 iterations to get smoother mean test and train score curves, each time with 20% data randomly selected as a validation set.
max_depth: The maximum depth of the tree

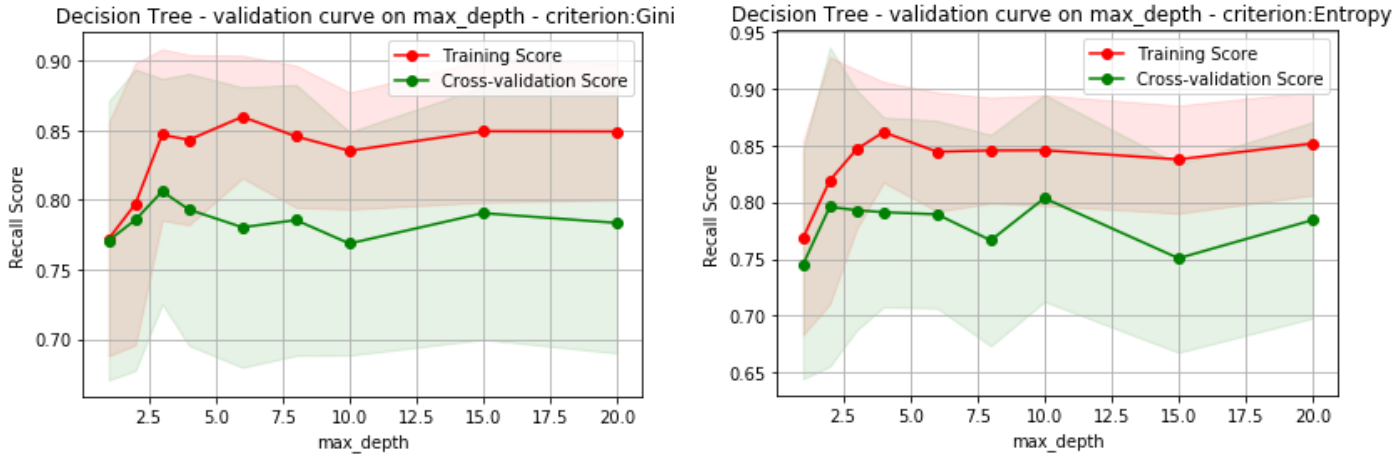

Figure-13: Decision Tree Validation Curve on Maximum Depth with Gini and Entropy Criterion

If we don't define a maximum depth, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. There's no significant difference between the Gini and Entropy criterions which are used calculating the splits. At maximum depth 3, they're providing the best results. With this sample, it's not easy to make an evaluation about the effect of max depth on overfitting.

min_samples_leaf: The minimum number of samples required to split an internal node

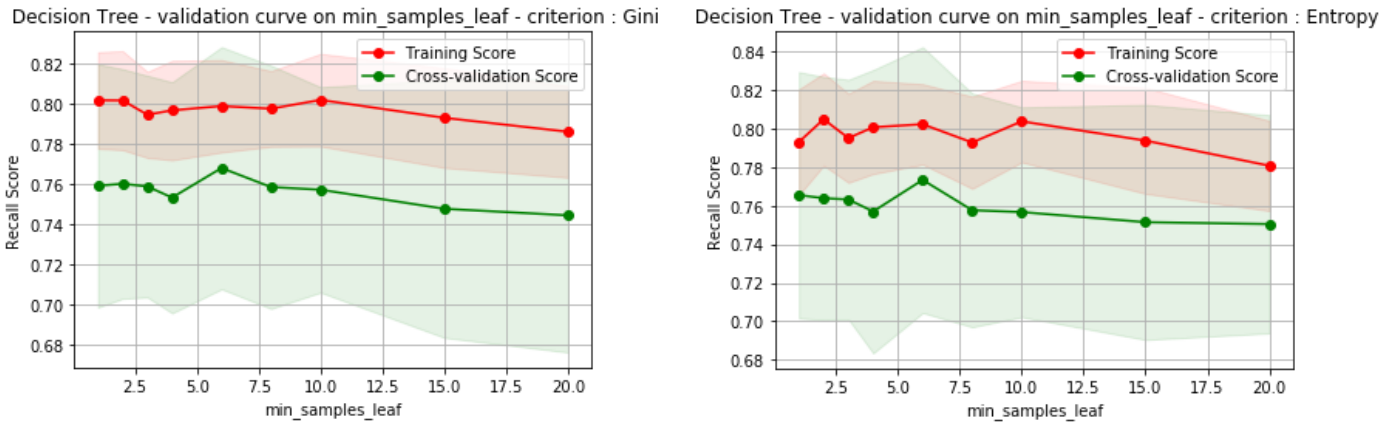

Figure-14: Decision Tree Validation Curve on Minimum Sample Leaf with Gini and Entropy Criterion

min_samples_leaf=2 is providing the best result. As the min_samples_leaf size increases, the model's performance is decreasing due to lack of splits. For this dataset, again it's not easy to make an evaluation about the effect of this parameter on overfitting. According to plots, we can say there's no significant difference between Gini and Entropy criterions among min_samples_leaf.

Boosting:

I used AdaBoostClassifier for ensemble learner using my previous Decision Tree as weak learner. I made hyper-parameter optimization using below parameters:
{'n_estimators': [36,35,36], 'learning_rate': [0.22,0.23,0.24], 'algorithm' : ['SAMME','SAMME.R']}
After obtaining the best parameters, I experimented different learning rates and n_estimators.

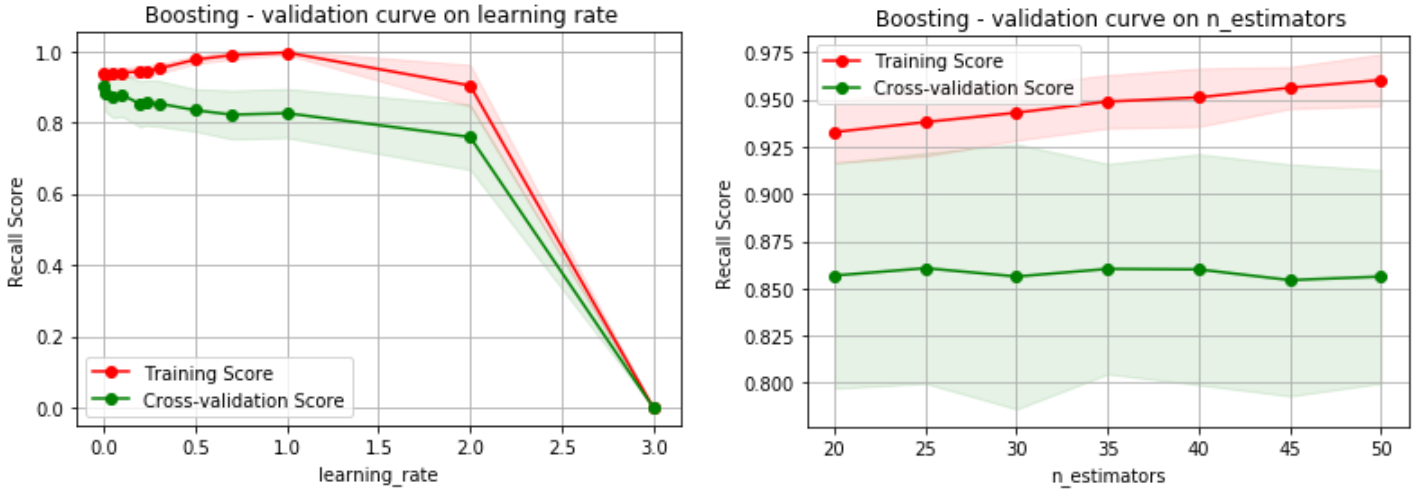

Figure-15: Boosting Validation Curve on Learning Rate and n_estimators.

Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators. I got the best results with learning_rate = 0.23 and n_estimators = 35.

Support Vector Classifier (Linear SVC):

I've used below parameters for hyper-parameter optimization.
{'dual':[True, False], 'tol':[0.097,0.098,0.0099], 'C': [2,3.7,3.8,4], 'fit_intercept': [True, False]}

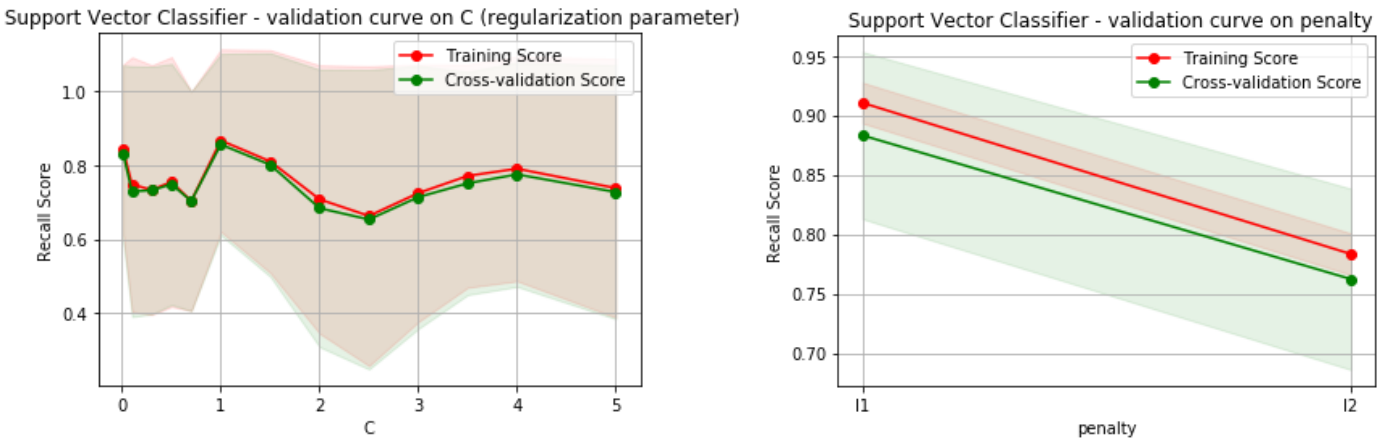

Figure-16: Support Vector Classifier Validation Curve on C (Regularization Parameter) and Penalty

The Linear Support Vector classifier is performing great with "L1" (Lasso) Regularization. There's no overfitting. L2 (ridge) regularization is not performing well. I got the best results with C=3 and L1 regularization.

Kernelized Support Vector Classifier:

I used below parameters for hyper-parameter optimization:
{' kernel: ['rbf','poly','sigmoid'], 'degree':[0.0001,0.0005,0.0008], 'gamma' : ['scale','auto'], 'C': [1.4,1.45,1.50], 'shrinking':[True,False],'tol':[0.011,0.015,0.016] }

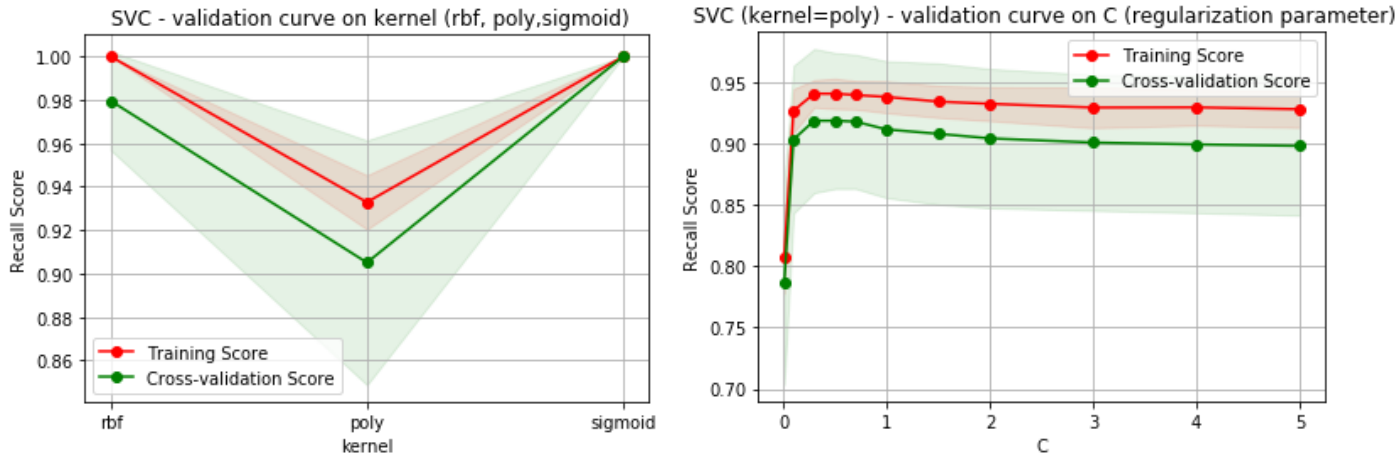

Figure-17: Kernelized Support Vector Classifier Validation Curve on C (Regularization Parameter) and Kernel

According to experiments, rbf and sigmoid kernels are performing well. According to cross validation and train scores sigmoid is the best, but rbf outperforming sigmoid on test data. The strength of the regularization is inversely proportional to C. The penalty is squared L2 penalty. I obtained the best results with rbf kernel and C= 1.7.

k-Nearest Neighbor Classifier:

I used below parameters for hyper-parameter optimization:
{'n_neighbors' : [4,5,6], 'weights' : ['uniform', 'distance'], 'algorithm': ['ball_tree','kd_tree','brute','auto'], 'leaf_size':[1,2,3] }

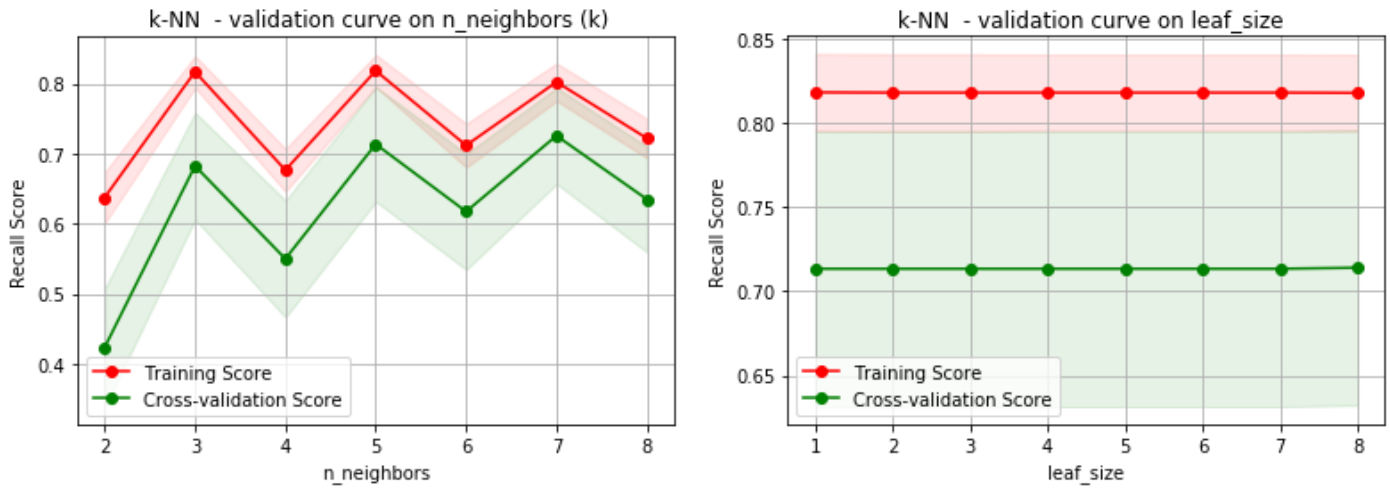

Figure-18: k-NN Classifier Validation Curve on n_neighbors(k) and leaf_size

The model is performing better with odd n_neighbors. And there's no significant effect of leaf_size on model's performance. I obtained the best results with n_neighbors = 5 and leaf_size=1

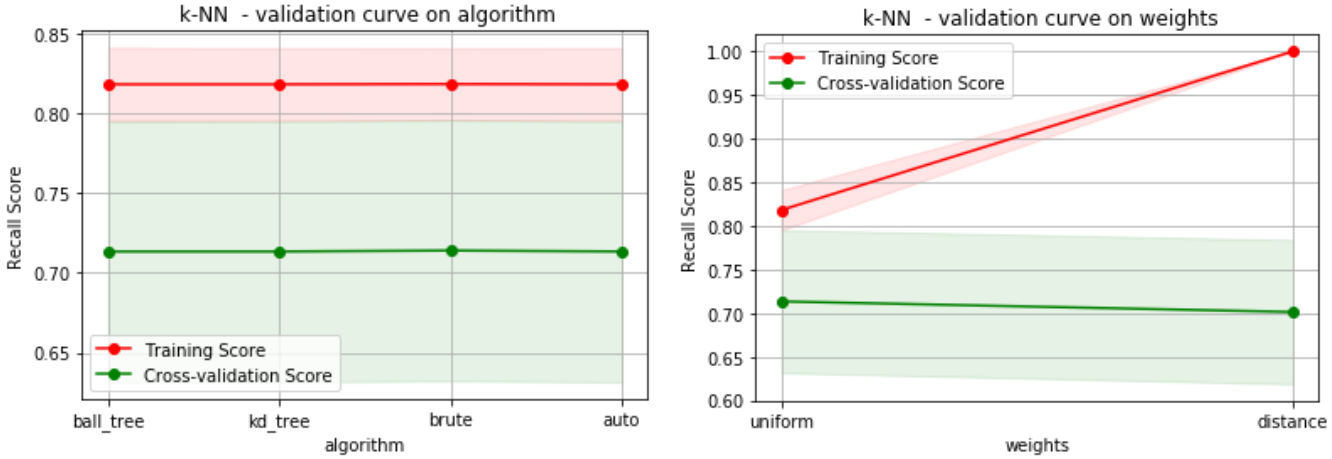

Figure-19: k-NN Classifier Validation Curve on algorithm and weights

The algorithms are used for computing the nearest neighbors. And the weight function is also used for prediction. With this dataset both of them don't have any significant effect on our model. Weights= distance is performing better with training set but on test data there's no difference between uniform and distance metrics.

Neural Networks:

I used below parameters for hyper-parameter optimization:

{'hidden_layer_sizes' : [(8,), (16,), (32,), (64,),(100,),(128,)], 'activation': ['relu', 'logistic','identity','tanh'], solver':['lbfgs','sgd','adam'], 'alpha': [0.005, 0.001,0.0001], 'learning_rate':['constant','invscaling','adaptive'], 'learning_rate_init': sorted([(2**x)/500.0 for x in range(4)]+[0.000001]) }

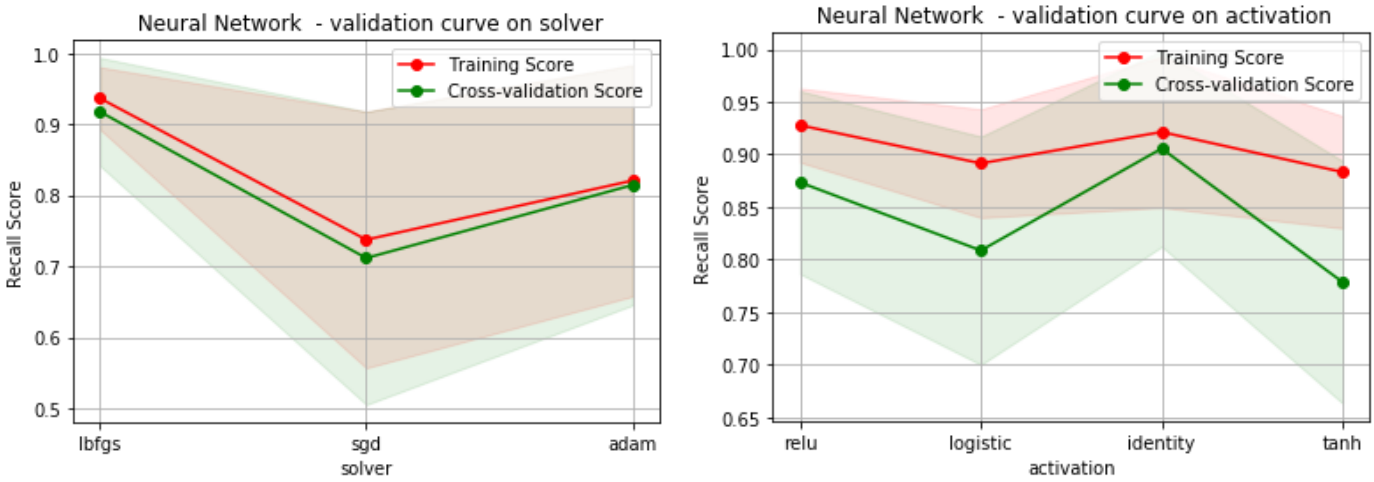

Figure-20: Neural Network Validation Curve on Solver and Activation Function

Figure-20 displays the performances of solvers and activation functions. For this dataset, I got the best results with Adam optimizer with Identity activation function.