Machine Learning: Assignment 1

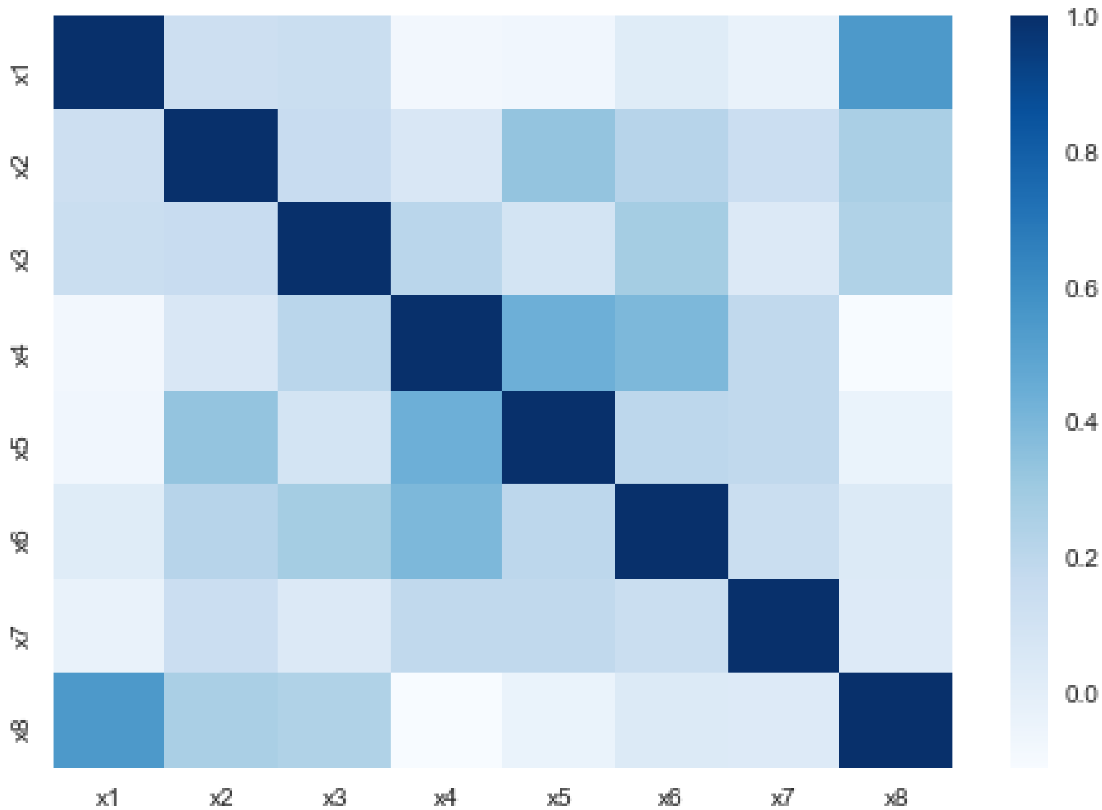
```
import numpy as np
import pandas as pd
import time
import gc
import random
from sklearn.model_selection import cross_val_score, GridSearchCV,
cross_validate, train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, f1_score
from sklearn.datasets import make_classification
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import validation_curve
from sklearn.neural_network import MLPClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from yellowbrick.model_selection import LearningCurve, ValidationCurve
from sklearn.preprocessing import OneHotEncoder

#Random State
rs = 614
```

1. Data Import, leansing Setup and helper functions

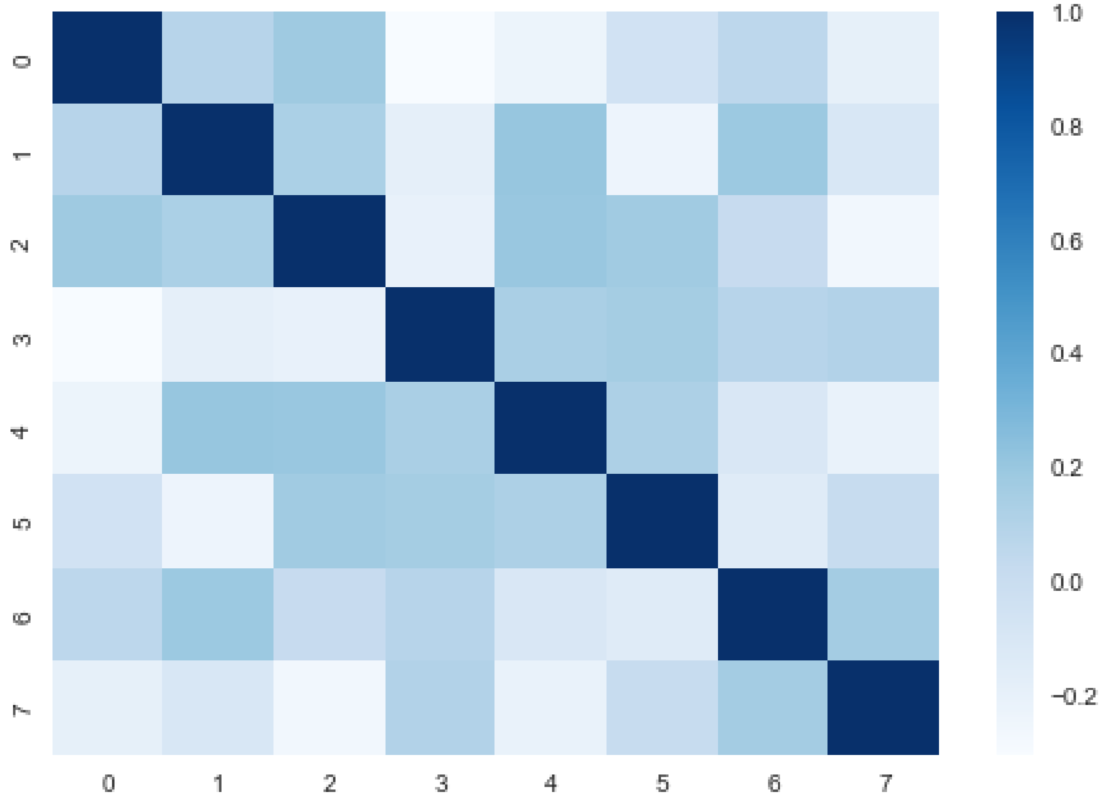
```
class Data():
    def dataAllocation(self,path):
        df = pd.read_csv(path)
        x_data = df.iloc[:, :-1]
        y_data = df.iloc[:, -1 ]
        return x_data,y_data
        # X, y = make_classification(n_samples=500, n_features=5,
n_informative=5, n_redundant=0, random_state=rs)
        # return X, y

    def syntheticData(self):
        return make_classification(n_samples=2000, n_features=8,
n_informative=8, n_redundant=0, random_state=rs)

    def trainSets(self,x_data,y_data):
        x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
```

```
test_size = 0.3, random_state = rs, shuffle = True)
        return x_train, x_test, y_train, y_test


data = 'data/pima-indians-diabetes.csv'
dataset = Data()
x1_data,y1_data = dataset.dataAllocation(data)
x1_train, x1_test, y1_train, y1_test = dataset.trainSets(x1_data,y1_data)
scaler = StandardScaler()
scaled_x1_train = scaler.fit_transform(x1_train)
scaled_x1_test = scaler.transform(x1_test)

x2_data,y2_data = dataset.syntheticData()
x2_train, x2_test, y2_train, y2_test = dataset.trainSets(x2_data,y2_data)
scaler = StandardScaler()
scaled_x2_train = scaler.fit_transform(x2_train)
scaled_x2_test = scaler.transform(x2_test)

x_data = [x1_data, x2_data]
y_data = [y1_data, y2_data]
x_test = [x1_test, x2_test]
y_test = [y1_test, y2_test]
x_train = [x1_train, x2_train]
y_train = [y1_train, y2_train]
scaled_x_train = [scaled_x1_train, scaled_x2_train]
scaled_x_test = [scaled_x1_test, scaled_x2_test]


sizes = np.linspace(0.3, 1.0, 10)
print("Heatmap for Features")
data_corr = sns.heatmap(pd.DataFrame(x_data[0]).corr(), cmap='Blues')
```

```
Heatmap for Features
```




```
data_corr = sns.heatmap(pd.DataFrame(x_data[1]).corr(), cmap='Blues')
```




```
def fakeTunedData(i):
    fakeData = [{'max_depth': 9, 'min_samples_leaf': 1}, {'C': 0.01,
'kernel': 'linear'}, {'n_neighbors': 3}, {'hidden_layer_sizes': 81,
'learning_rate_init': 0.04},{'max_depth': 3, 'n_estimators': 90}]
    return 72+i, fakeData[i]

def learningCurvePlot(tuned, data_n):
    v1 = LearningCurve(tuned, scoring='f1_weighted', train_sizes=sizes,
n_jobs=4)
    v1.fit(x_data[data_n],y_data[data_n])
    v1.show()
```

2. Decision Tree Classifier

```
class DTClassifier():

    def trainTest(self, data_n):
        df = []
        for i in range(16):
            for j in range(20):
                dt_clf = DecisionTreeClassifier(max_depth=i+1,
min_samples_leaf=j+1)
                dt_clf.fit(x_train[data_n], y_train[data_n])
```

```
                y1_predict_train = dt_clf.predict(x_train[data_n])
                y1_predict_test = dt_clf.predict(x_test[data_n])
                df.append([i+1, j+1, "Train", accuracy_score(y_train[data_n],
y1_predict_train)])
                df.append([i+1, j+1, "Test", accuracy_score(y_test[data_n],
y1_predict_test)])

        return pd.DataFrame(df, columns=["Depth", "Leaf Size", "Sample Type",
"F1 Score" ])

    def hyperParameterTuning(self, data_n):
        param_grid = {'max_depth': range(1, 21), 'min_samples_leaf': range(1,
20)}
        tuned = GridSearchCV(estimator = DecisionTreeClassifier(random_state
= rs), param_grid = param_grid, cv=10)
        tuned.fit(x_train[data_n], y_train[data_n])
        return tuned.best_score_, tuned.best_params_

    def dataExplorePlot(self, df):
        g = sns.FacetGrid(df, hue="Sample Type", col="Depth", height=2,
col_wrap=4)
        g.map(sns.lineplot, "Leaf Size", "F1 Score" )

    def showLearningCurve(self, best_params, data_n):
        tuned = DecisionTreeClassifier(max_depth=best_params['max_depth'],
min_samples_leaf=best_params['min_samples_leaf'], random_state=rs)
        learningCurvePlot(tuned, data_n)
```

3. Support Vector Machine

```
class SupportVectorMachine():
    def trainTest(self, data_n):
        cs = [x/10000 for x in [1, 10, 100, 1000, 10000, 100000, 1000000]]
        df = []
        for c in cs:
            for k in ["linear", "sigmoid"]:
                model = SVC(kernel = k, C=c)
                model.fit(scaled_x_train[data_n],y_train[data_n])
                y1_predict_train = model.predict(scaled_x_train[data_n])
                y1_predict_test = model.predict(scaled_x_test[data_n])
                df.append([k, c, accuracy_score(y_test[data_n],
y1_predict_test)])
        return pd.DataFrame(df, columns=["Kernel", "C", "Accuracy"])

    def hyperParameterTuning(self, data_n):
        param_grid = {'C': [x/10000 for x in [1, 10, 100, 1000, 10000,
100000, 1000000]],
            'kernel': ["linear", "sigmoid"]}
        svm_tune = SVC(gamma = "auto")
```

```
        svm_cv = GridSearchCV(estimator = svm_tune, param_grid = param_grid, 
n_jobs=5, return_train_score=True)
        svm_cv.fit(scaled_x_train[data_n], y_train[data_n])
        best_score = svm_cv.best_score_
        return best_score, svm_cv.best_params_

    def dataExplorePlot(self, df):
        g = sns.FacetGrid(df, col="Kernel",  height=4)
        g.map(sns.pointplot, "C", "Accuracy")

    def showLearningCurve(self, best_params, data_n):
        tuned = SVC(kernel=best_params['kernel'], C=best_params['C'])
        learningCurvePlot(tuned, data_n)
```

4. KNN

```
class KNN():
    def trainTest(self, data_n):
        df = []
        for i in range(20):
            model = KNeighborsClassifier(n_neighbors= i+1)
            model.fit(x_train[data_n], y_train[data_n])
            y1_predict_test = model.predict(x_test[data_n])
            df.append([i+1, accuracy_score(y_test[data_n], y1_predict_test)])
        return pd.DataFrame(df, columns= ["Neighbors", "Accuracy"])

    def hyperParameterTuning(self, data_n):
        tuned = GridSearchCV(KNeighborsClassifier(), {"n_neighbors" : 
range(1, 21)})
        tuned.fit(x_train[data_n], y_train[data_n])
        return tuned.best_score_, tuned.best_params_

    def dataExplorePlot(self, df):
        sns.lineplot(data=df, x="Neighbors", y="Accuracy")

    def showLearningCurve(self, best_params, data_n):
        tuned = KNeighborsClassifier(n_neighbors=best_params['n_neighbors'])
        learningCurvePlot(tuned, data_n)
```

5. Neural Network

```
class NN():

    def trainTest(self, data_n):
        df = []
        for i in [0.01, 0.02, 0.04, 0.08, 0.1]:
            model = MLPClassifier(max_iter=300, learning_rate_init=i)
```

```
            model.fit(scaled_x_train[data_n],y_train[data_n])
            y1_predict_train = model.predict(scaled_x_train[data_n])
            y1_predict_test = model.predict(scaled_x_test[data_n])
            df.append([i, accuracy_score(y_test[data_n], y1_predict_test)])
        return pd.DataFrame(df, columns=["Learning Rate", "Accuracy"])

    def hyperParameterTuning(self, data_n):
        param_grid = {
            'hidden_layer_sizes': [x**2 for x in range(2, 11)],
            'learning_rate_init': [0.01, 0.02, 0.04, 0.08, 0.1]
        }
        tuned = GridSearchCV(MLPClassifier(max_iter=200), param_grid =
param_grid, cv=10)
        tuned.fit(scaled_x_train[data_n], y_train[data_n])
        return tuned.best_score_, tuned.best_params_

    def dataExplorePlot(self, df):
        sns.lineplot(data=df, x="Learning Rate", y="Accuracy")

    def showLearningCurve(self, best_params, data_n):
        tuned = MLPClassifier(max_iter=50,
learning_rate_init=best_params['learning_rate_init'],
hidden_layer_sizes=best_params['hidden_layer_sizes'])
        learningCurvePlot(tuned, data_n)
```

6. Boost

```
class Boost():
    def trainTest(self, data_n):
        df = []
        for i in range(50, 251, 10):
            model = GradientBoostingClassifier(n_estimators=i, max_depth=3,
min_samples_leaf=10)
            model.fit(scaled_x_train[data_n],y_train[data_n])
            y1_predict_train = model.predict(scaled_x_train[data_n])
            y1_predict_test = model.predict(scaled_x_test[data_n])
            df.append([i, accuracy_score(y_test[data_n], y1_predict_test)])
        return pd.DataFrame(df, columns=["Estimators", "Accuracy"])

    def hyperParameterTuning(self, data_n):
        param_grid = {'n_estimators': range(50, 151, 40), 'max_depth':
range(2, 4)}
        tuned = GridSearchCV(GradientBoostingClassifier(), param_grid, cv=10)
        tuned.fit(scaled_x_train[data_n], y_train[data_n])
        return tuned.best_score_, tuned.best_params_

    def dataExplorePlot(self, df):
        sns.lineplot(data=df, x="Estimators", y="Accuracy")
```

```
    def showLearningCurve(self, best_params, data_n):
        tuned =
GradientBoostingClassifier(n_estimators=best_params['n_estimators'],
max_depth=best_params['max_depth'], min_samples_leaf=100)
        learningCurvePlot(tuned, data_n)


classifiers = [DTClassifier(), SupportVectorMachine(), KNN(), NN(), Boost()]
# classifiers = [KNN()]

classifiers_name = ["Decision Tree", "Support Vector Machine", "K-Nearest
Neighbors", "Neural Network", "Gradient Boosting"]

i = 0

for classifier in classifiers:
    name = classifiers_name[i]
    print(str(i+1)+".", name, end="\n\n")
    df1 = classifier.trainTest(0)
    df2 = classifier.trainTest(1)

    #best_score, best_params = fakeTunedData(i)
    best_score1, best_params1 = classifier.hyperParameterTuning(0)
    best_score2, best_params2 = classifier.hyperParameterTuning(1)
    print(str(i+1)+".1","Hyperparameters Exploration", end="\n\n")
    print("For this project,", name, "will be hypertuned by adjusting:", [x
for x in best_params1], end=". ")
    print("The following charts show how the accuracy is affected when the
hyperparamter(s) are changed for both dataset 1 and dataset 2:")

    classifier.dataExplorePlot(df1)
    plt.figure()
    classifier.dataExplorePlot(df2)
    plt.figure()



    print("\n\n")
    print(str(i+1)+".2","Hypertuning", end="\n\n")
    print("GridSearchCV was performed for", name, "classifier." , end="\n\n")
    print("Dataset 1 Results:")
    for key in best_params1:
        print("The optimal value of", key, "was", best_params1[key], end = ".
")
    print("Likewise the accuracy of", name, "classifier", "was", best_score1,
"when the optimized hyperparameter(s) value(s) were used.")
    classifier.showLearningCurve(best_params1, 0)

    print("Dataset 2 Results:")
```

```
    for key in best_params2:
        print("The optimal value of", key, "was", best_params2[key], end = ".
")
    print("Likewise the accuracy of", name, "classifier", "was", best_score2,
"when the optimized hyperparameter(s) value(s) were used.")
    classifier.showLearningCurve(best_params2, 1)
    print()
    print()

    i = i + 1
```

1. Decision Tree

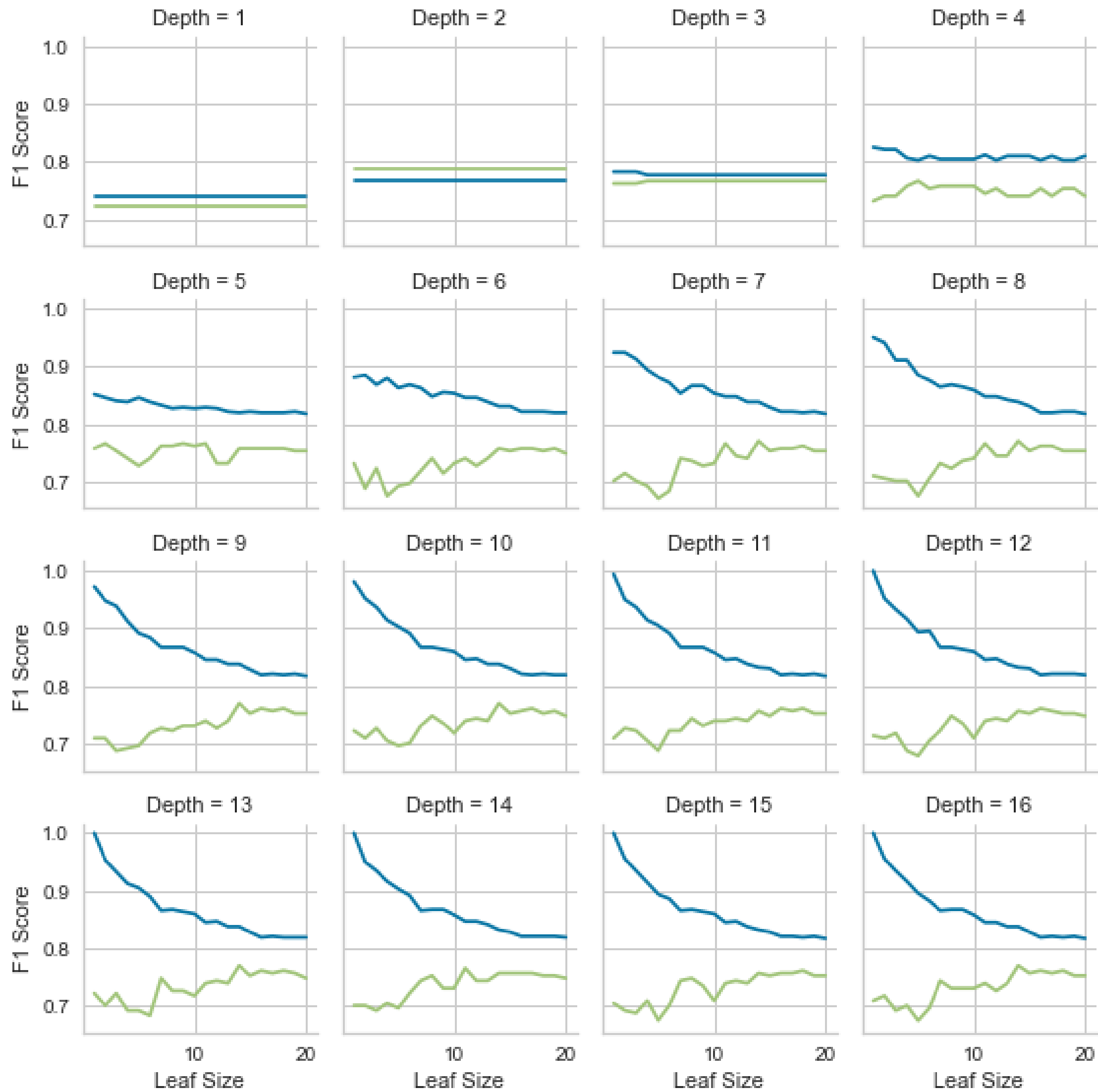
1.1 Hyperparameters Exploration

For this project, Decision Tree will be hypertuned by adjusting: ['max_depth', 'min_samples_leaf']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:

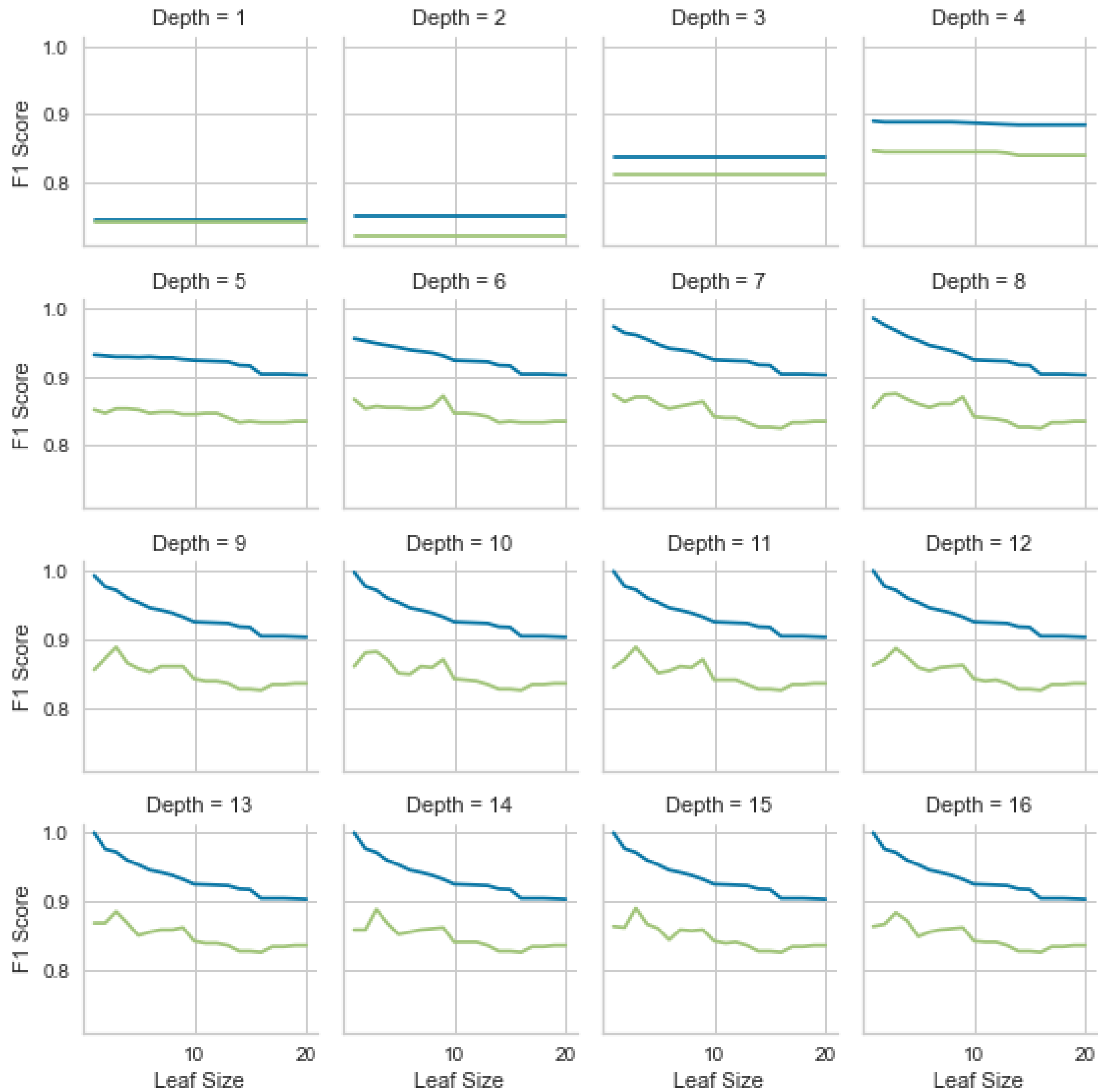
1.2 Hypertuning

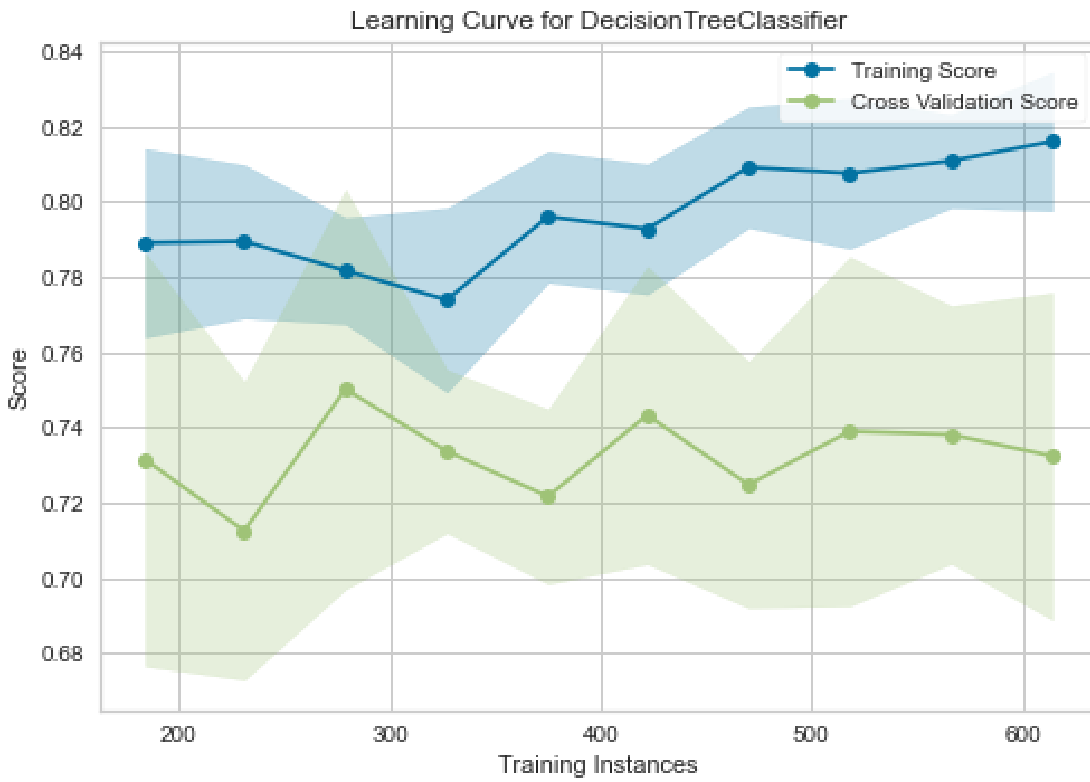
GridSearchCV was performed for Decision Tree classifier.

Dataset 1 Results:
The optimal value of max_depth was 6. The optimal value of min_samples_leaf was 17. Likewise the accuracy of Decision Tree classifier was 0.7635918937805731 when the optimized hyperparameter(s) value(s) were used.

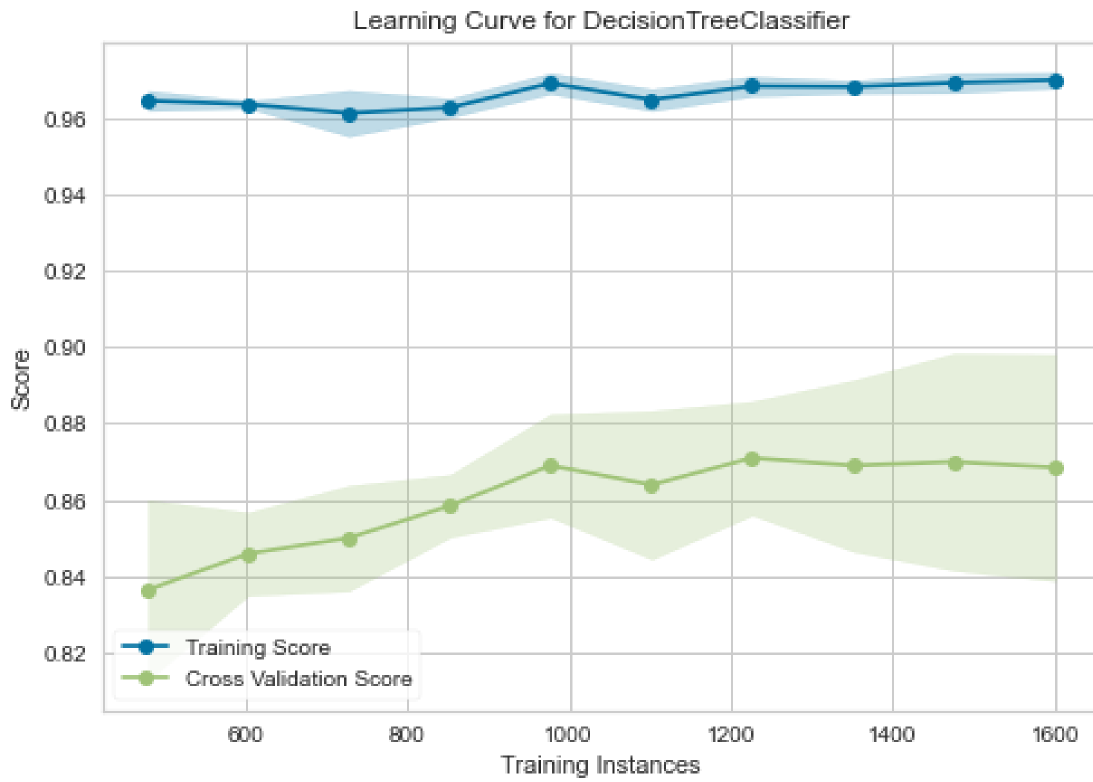


<Figure size 576x396 with 0 Axes>


Depth = 1
Depth = 2
Depth = 3
Depth = 4
Depth = 5
Depth = 6
Depth = 7
Depth = 8
Depth = 9
Depth = 10
Depth = 11
Depth = 12
Depth = 13
Depth = 14
Depth = 15
Depth = 16
F1 Score
1.0
0.9
0.8
10
20
Leaf Size




```
Dataset 2 Results:
The optimal value of max_depth was 10. The optimal value of min_samples_leaf
was 3. Likewise the accuracy of Decision Tree classifier was
0.8764285714285714 when the optimized hyperparameter(s) value(s) were used.
```



2. Support Vector Machine

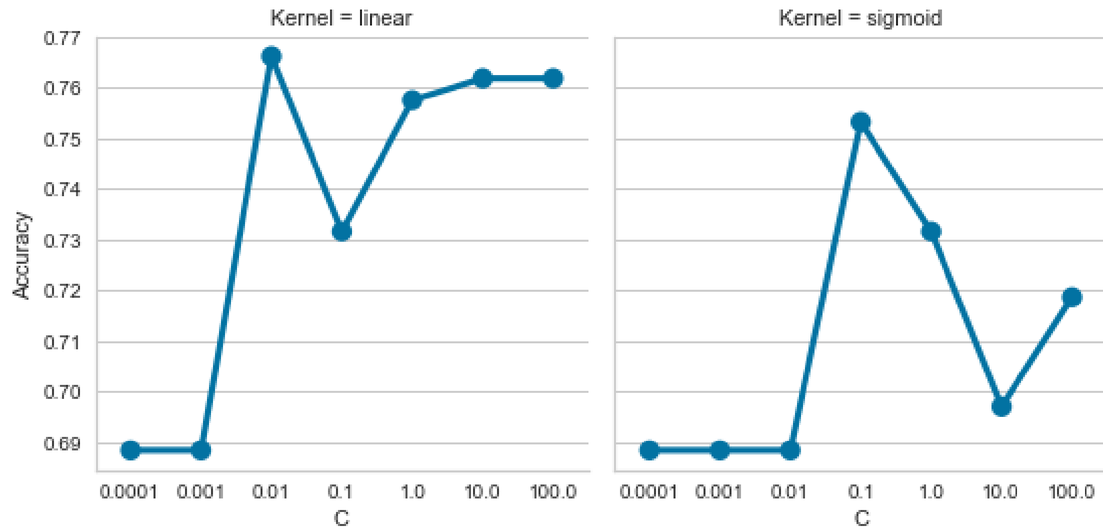
2.1 Hyperparameters Exploration

For this project, Support Vector Machine will be hypertuned by adjusting: ['C', 'kernel']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:

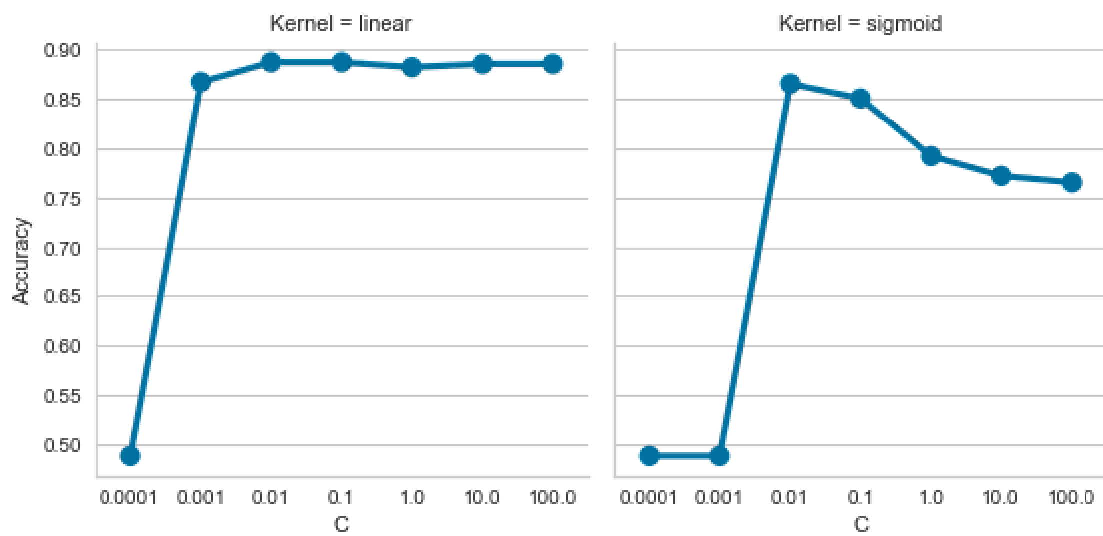
2.2 Hypertuning

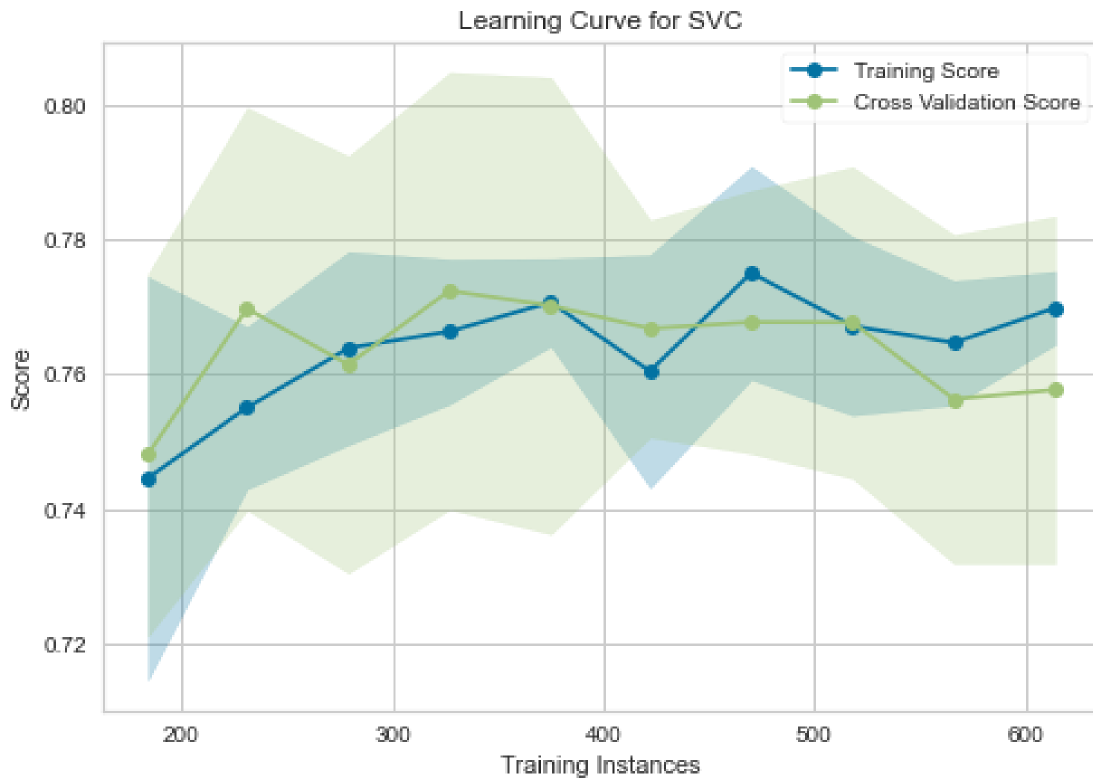
GridSearchCV was performed for Support Vector Machine classifier.

Dataset 1 Results:
The optimal value of C was 1.0. The optimal value of kernel was linear.
Likewise the accuracy of Support Vector Machine classifier was 0.7820526133610246 when the optimized hyperparameter(s) value(s) were used.

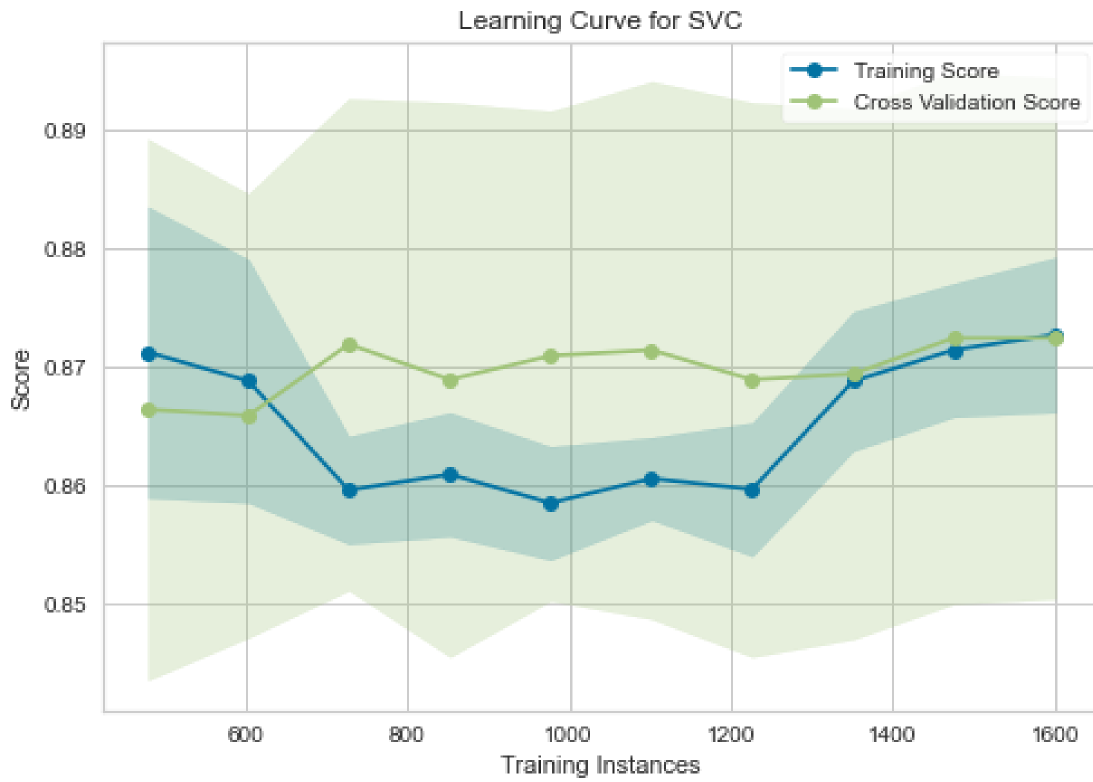


<Figure size 576x396 with 0 Axes>





Dataset 2 Results:
The optimal value of C was 100.0. The optimal value of kernel was linear. Likewise the accuracy of Support Vector Machine classifier was 0.8635714285714287 when the optimized hyperparameter(s) value(s) were used.



3. K-Nearest Neighbors

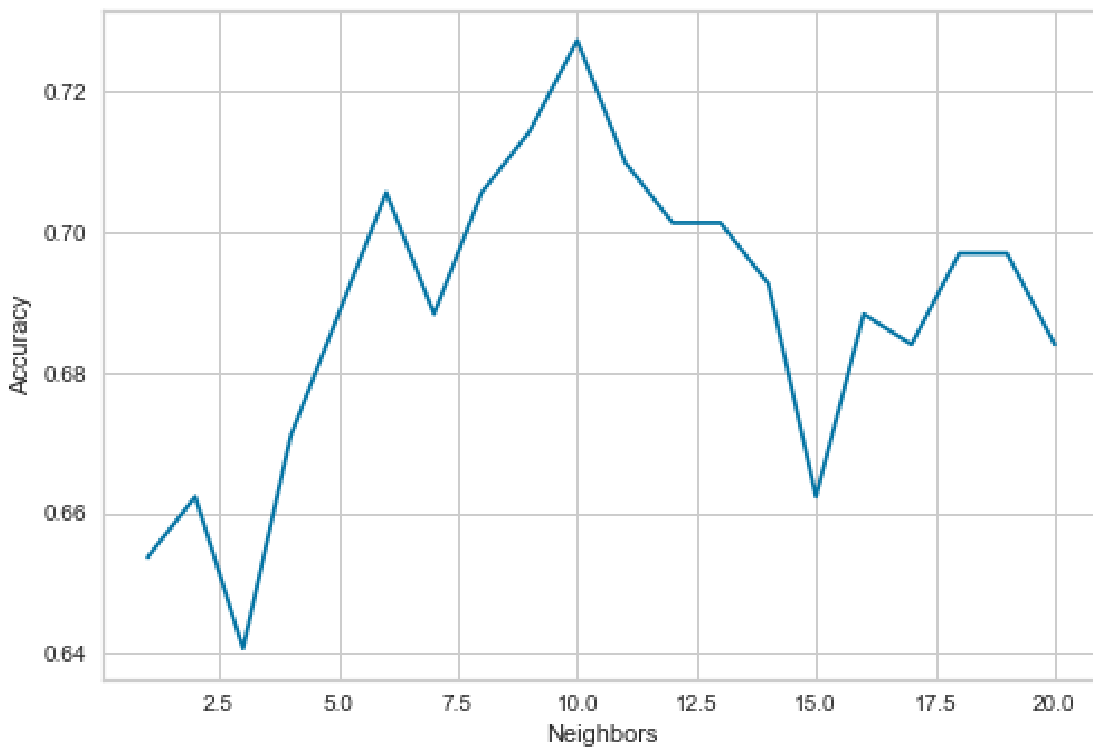
3.1 Hyperparameters Exploration

For this project, K-Nearest Neighbors will be hypertuned by adjusting: ['n_neighbors']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:

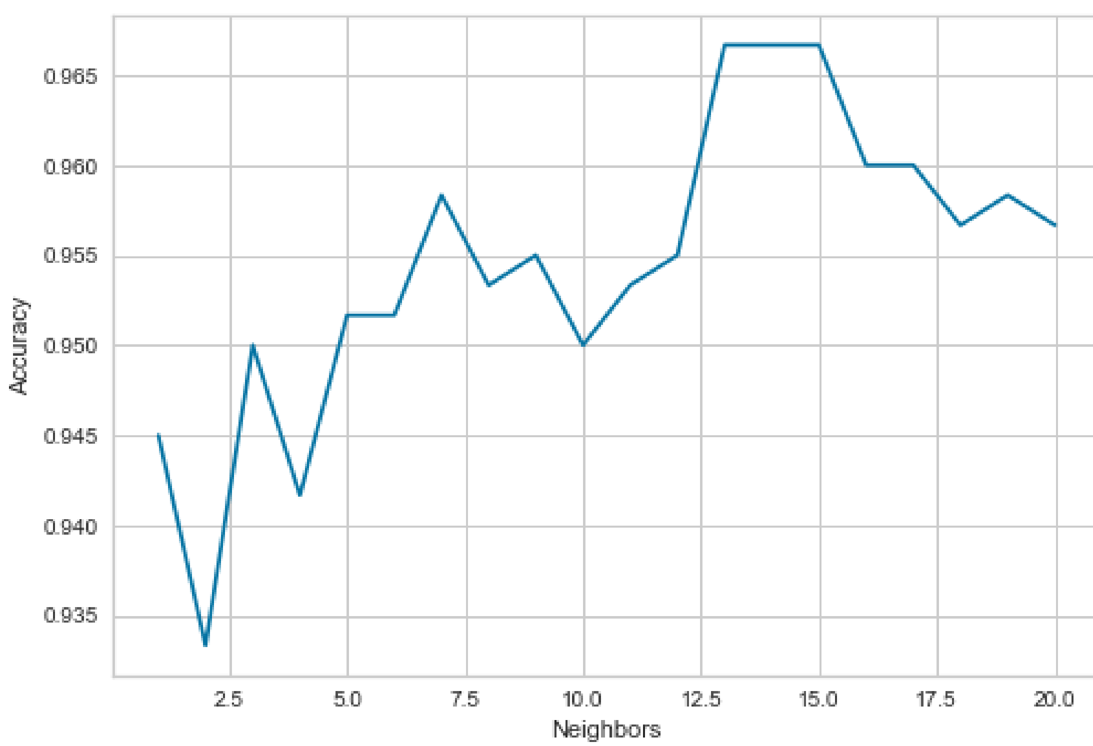
3.2 Hypertuning

GridSearchCV was performed for K-Nearest Neighbors classifier.

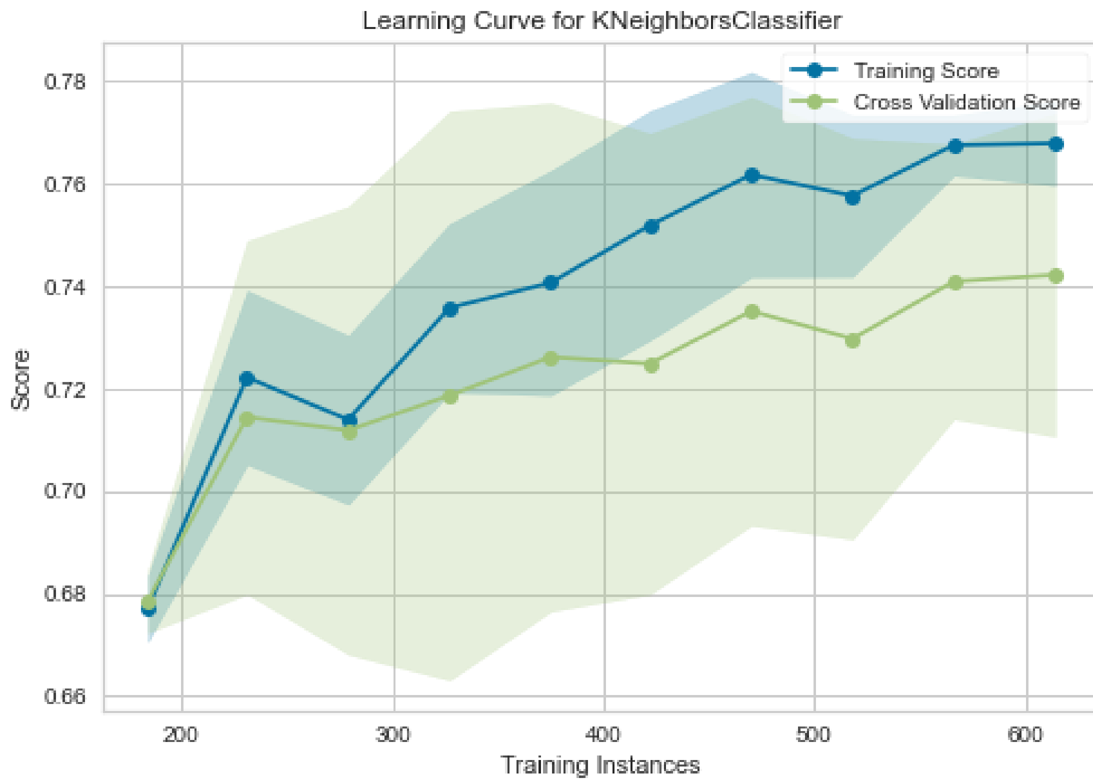
Dataset 1 Results:
The optimal value of n_neighbors was 12. Likewise the accuracy of K-Nearest Neighbors classifier was 0.7690377293181032 when the optimized hyperparameter(s) value(s) were used.



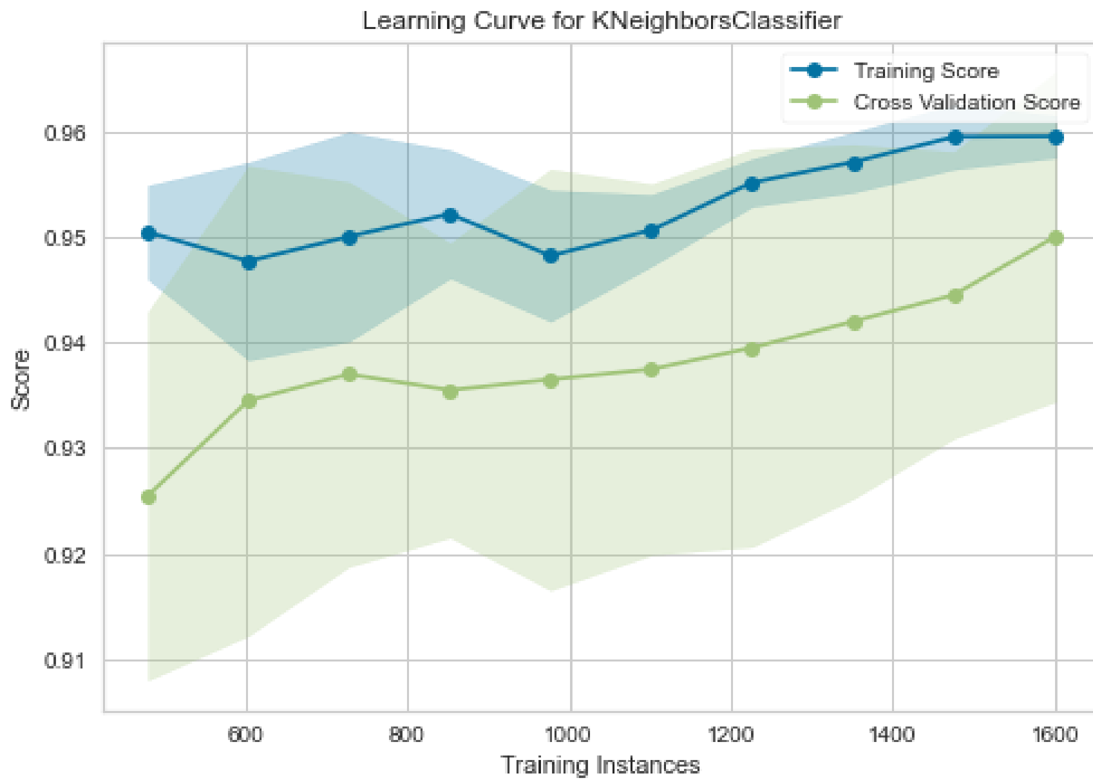
0.72
0.70
0.68
0.66
0.64
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Neighbors



0.965
0.960
0.955
0.950
0.945
0.940
0.935
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Neighbors



Dataset 2 Results:
The optimal value of n_neighbors was 11. Likewise the accuracy of K-Nearest Neighbors classifier was 0.9464285714285714 when the optimized hyperparameter(s) value(s) were used.



4. Neural Network

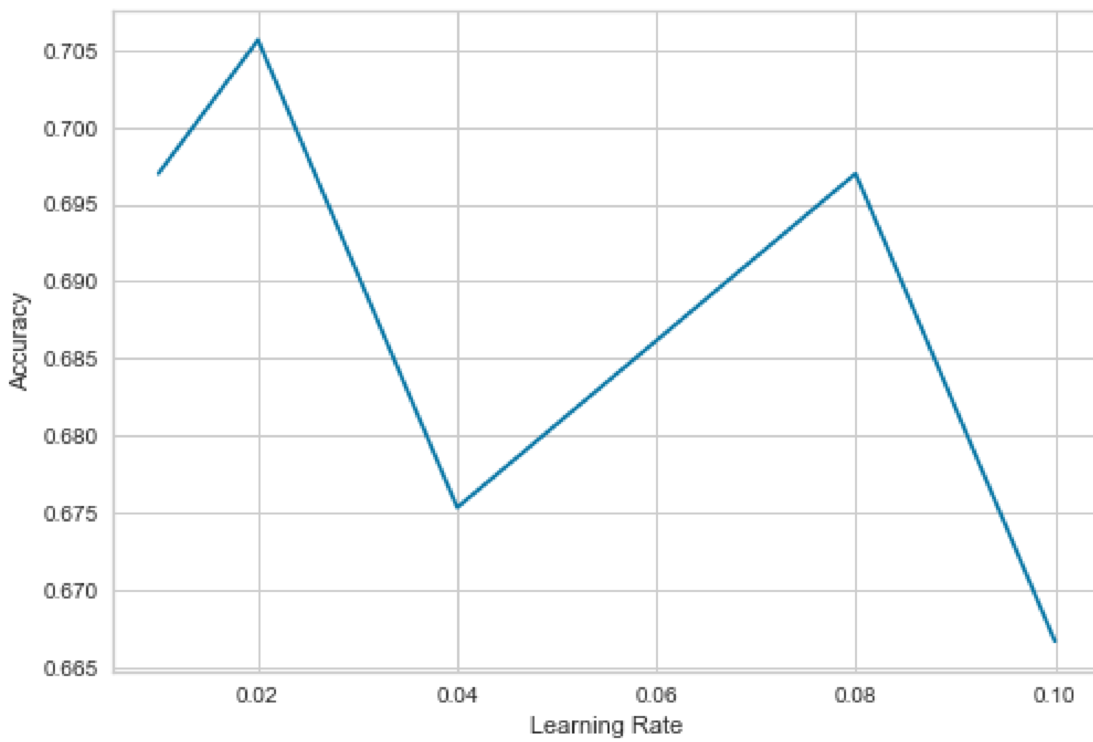
4.1 Hyperparameters Exploration

For this project, Neural Network will be hypertuned by adjusting: ['hidden_layer_sizes', 'learning_rate_init']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:

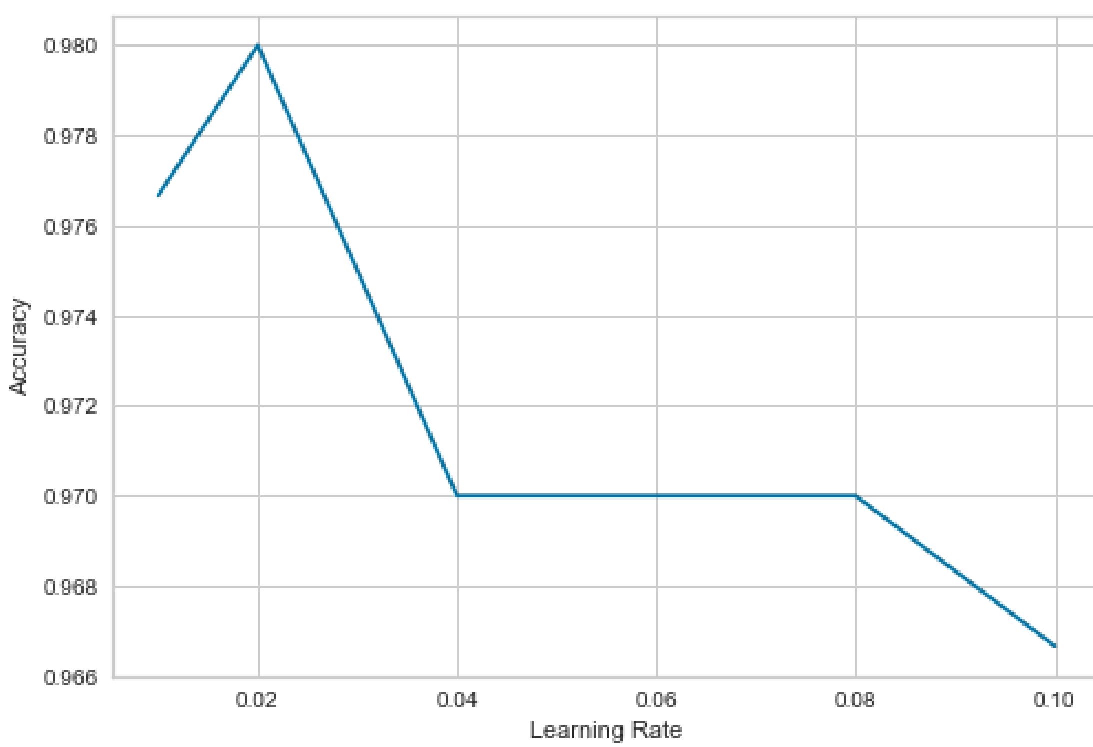
4.2 Hypertuning

GridSearchCV was performed for Neural Network classifier.

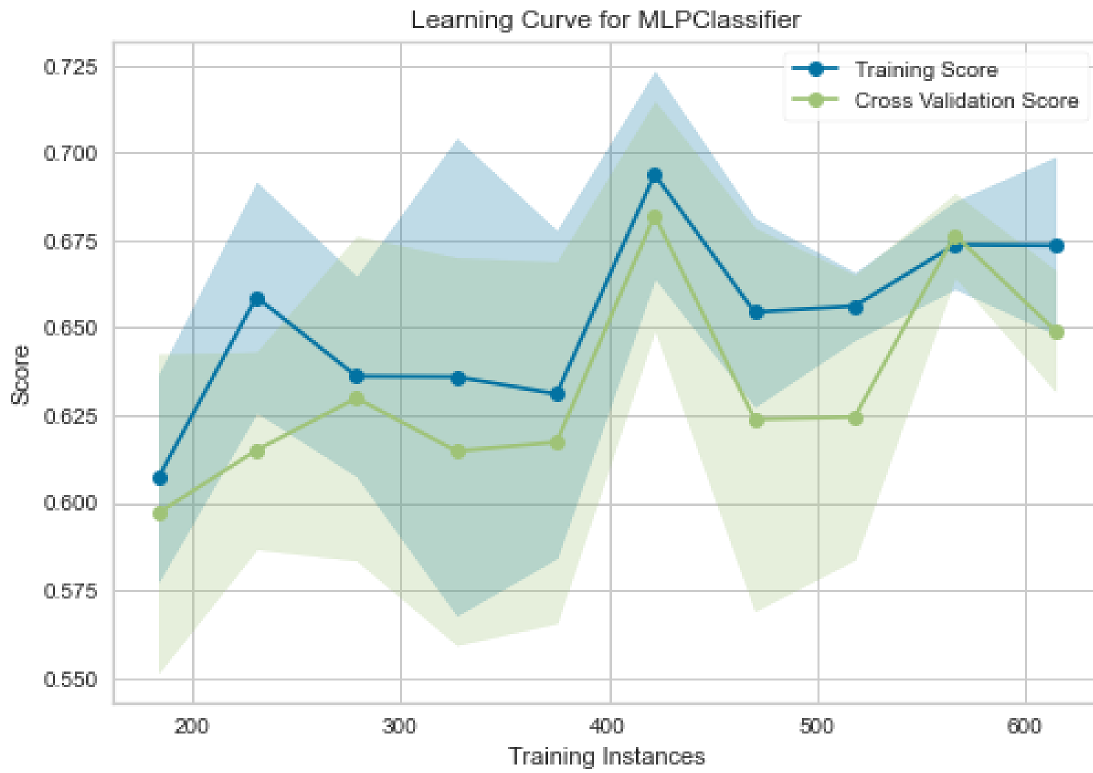
Dataset 1 Results:
The optimal value of hidden_layer_sizes was 9. The optimal value of learning_rate_init was 0.01. Likewise the accuracy of Neural Network classifier was 0.7841020265548566 when the optimized hyperparameter(s) value(s) were used.



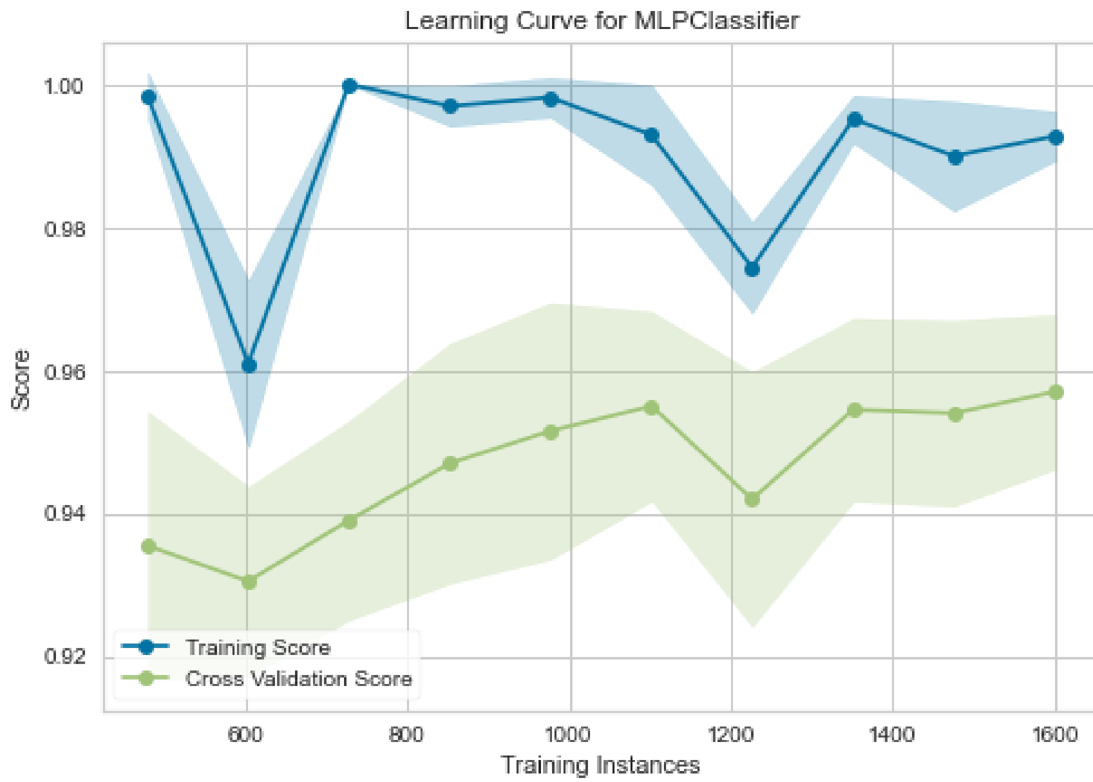
Accuracy
0.705
0.700
0.695
0.690
0.685
0.680
0.675
0.670
0.665
0.02
0.04
0.06
0.08
0.10
Learning Rate



Accuracy
0.980
0.978
0.976
0.974
0.972
0.970
0.968
0.966
0.02
0.04
0.06
0.08
0.10
Learning Rate



Dataset 2 Results:
The optimal value of hidden_layer_sizes was 64. The optimal value of learning_rate_init was 0.04. Likewise the accuracy of Neural Network classifier was 0.9592857142857142 when the optimized hyperparameter(s) value(s) were used.



5. Gradient Boosting

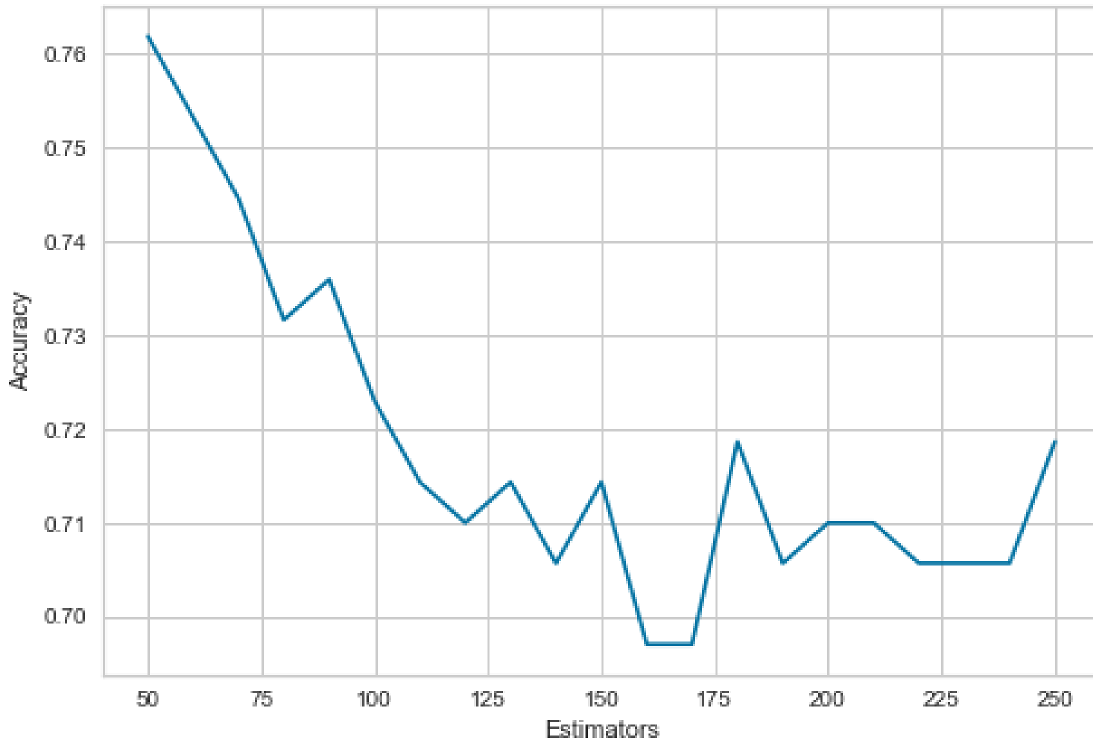
5.1 Hyperparameters Exploration

For this project, Gradient Boosting will be hypertuned by adjusting: ['max_depth', 'n_estimators']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:

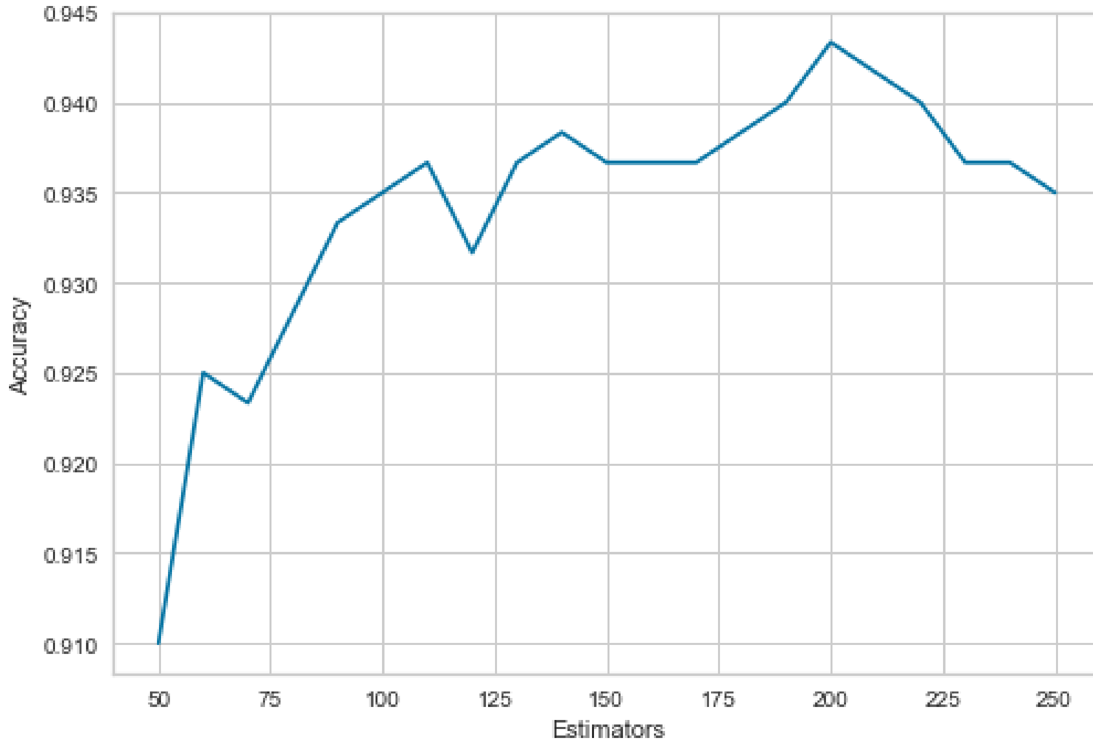
5.2 Hypertuning

GridSearchCV was performed for Gradient Boosting classifier.

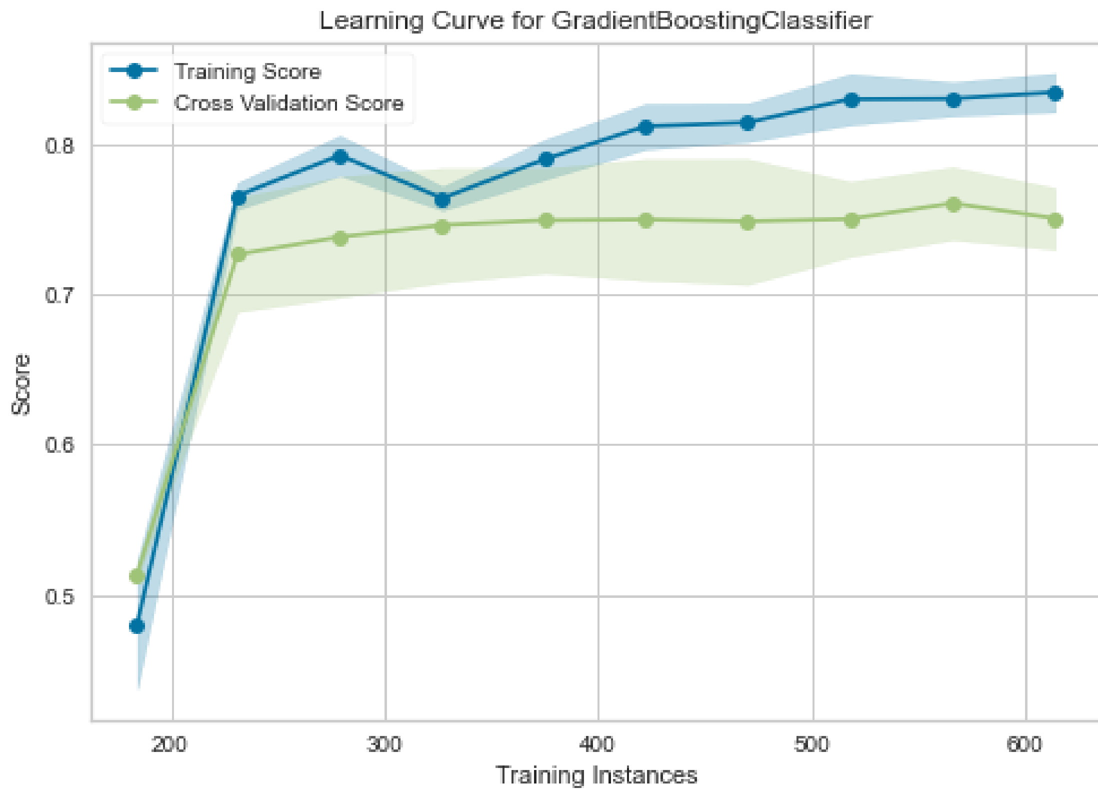
Dataset 1 Results:
The optimal value of max_depth was 3. The optimal value of n_estimators was 130. Likewise the accuracy of Gradient Boosting classifier was 0.7841020265548567 when the optimized hyperparameter(s) value(s) were used.



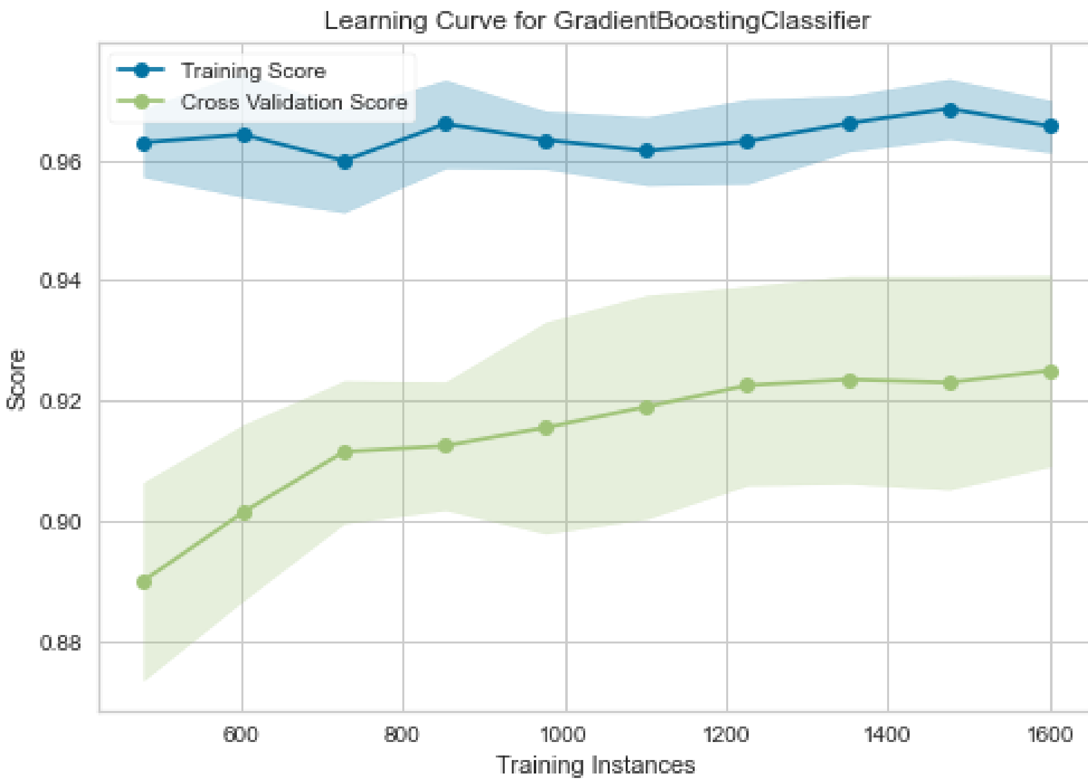
Accuracy
Estimators
0.76
0.75
0.74
0.73
0.72
0.71
0.70
50
75
100
125
150
175
200
225
250



Accuracy
Estimators
0.945
0.940
0.935
0.930
0.925
0.920
0.915
0.910
50
75
100
125
150
175
200
225
250



Dataset 2 Results:
The optimal value of max_depth was 3. The optimal value of n_estimators was 130. Likewise the accuracy of Gradient Boosting classifier was 0.9178571428571429 when the optimized hyperparameter(s) value(s) were used.



Learning Curve for GradientBoostingClassifier
Training Score
Cross Validation Score
Score
0.96
0.94
0.92
0.90
0.88
600
800
1000
1200
1400
1600
Training Instances