# Machine Learning: Assignment 1

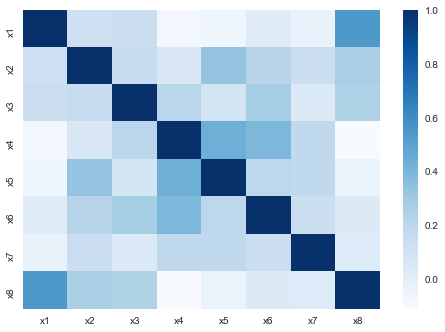
import numpy as np  
import pandas as pd  
import time  
import gc  
import random  
from sklearn.model\_selection import cross\_val\_score, GridSearchCV, cross\_validate, train\_test\_split  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, f1\_score  
from sklearn.datasets 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  
import time  
  
#Random State  
rs = 614

# 1. Data Import, leansing Setup and helper functions

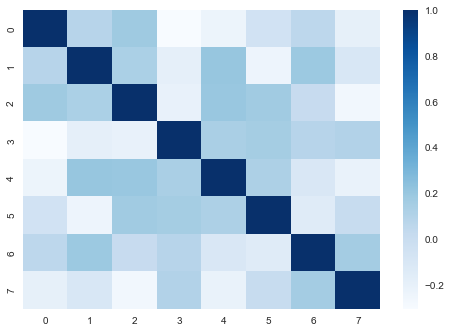
class Data():  
 def dataAllocation(self,path):  
 df = pd.read\_csv(path)  
 print(df.shape)  
 x\_data = df.iloc[:, :-1]  
 y\_data = df.iloc[:, -1 ]  
 return x\_data,y\_data  
 # X, y = make\_classification(n\_samples=500, n\_features=5, n\_informative=5, n\_redundant=0, random\_state=rs)  
 # return 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')

(768, 9)  
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(20):  
 dt\_clf = DecisionTreeClassifier(max\_depth=i+1, criterion='entropy', random\_state=rs)  
 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, "Depth", "Train", f1\_score(y\_train[data\_n], y1\_predict\_train)])  
 df.append([i+1, "Depth", "Test", f1\_score(y\_test[data\_n], y1\_predict\_test)])  
  
 for i in range(20):  
 dt\_clf = DecisionTreeClassifier(min\_samples\_leaf=i+1, criterion='entropy', random\_state=rs)  
 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, "Leaf Size", "Train", f1\_score(y\_train[data\_n], y1\_predict\_train)])  
 df.append([i+1, "Leaf Size", "Test", f1\_score(y\_test[data\_n], y1\_predict\_test)])  
  
  
  
 return pd.DataFrame(df, columns=["N", "Paramter Type", "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)  
   
 start\_time = time.time()  
 tuned.fit(x\_train[data\_n], y\_train[data\_n])  
 training\_time = time.time() - start\_time  
 start\_time = time.time()  
 tuned.predict(x\_test[data\_n])  
 testing\_time = time.time() - start\_time  
 return tuned.best\_score\_, tuned.best\_params\_, training\_time, testing\_time  
  
 def dataExplorePlot(self, df):  
 g = sns.FacetGrid(df, hue="Sample Type", col="Paramter Type", height=2, col\_wrap=4)  
 g.map(sns.lineplot, "N", "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(["Train", k, c, accuracy\_score(y\_train[data\_n], y1\_predict\_train)])  
 df.append(["Test", k, c, accuracy\_score(y\_test[data\_n], y1\_predict\_test)])  
 return pd.DataFrame(df, columns=["Type", "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")  
 tuned = GridSearchCV(estimator = svm\_tune, param\_grid = param\_grid, n\_jobs=5, return\_train\_score=True)  
   
 start\_time = time.time()  
 tuned.fit(scaled\_x\_train[data\_n], y\_train[data\_n])  
 training\_time = time.time() - start\_time  
 start\_time = time.time()  
 tuned.predict(scaled\_x\_test[data\_n])  
 testing\_time = time.time() - start\_time  
 return tuned.best\_score\_, tuned.best\_params\_, training\_time, testing\_time  
   
   
 def dataExplorePlot(self, df):  
 g = sns.FacetGrid(df, hue="Type", col="Kernel", height=4)  
 g.map(sns.pointplot, "C", "Accuracy")  
 g.add\_legend()  
   
 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])  
 y1\_predict\_train = model.predict(x\_train[data\_n])  
 df.append(["Train", i+1, accuracy\_score(y\_train[data\_n], y1\_predict\_train)])  
 df.append(["Test", i+1, accuracy\_score(y\_test[data\_n], y1\_predict\_test)])  
  
  
 return pd.DataFrame(df, columns= ["Type", "Neighbors", "Accuracy"])  
   
 def hyperParameterTuning(self, data\_n):  
 tuned = GridSearchCV(KNeighborsClassifier(), {"n\_neighbors" : range(1, 21)})  
   
 start\_time = time.time()  
 tuned.fit(x\_train[data\_n], y\_train[data\_n])  
 training\_time = time.time() - start\_time  
 start\_time = time.time()  
 tuned.predict(x\_test[data\_n])  
 testing\_time = time.time() - start\_time  
 return tuned.best\_score\_, tuned.best\_params\_, training\_time, testing\_time  
  
 def dataExplorePlot(self, df):  
 sns.lineplot(data=df, x="Neighbors", y="Accuracy", hue="Type")  
   
 def showLearningCurve(self, best\_params, data\_n):  
 tuned = KNeighborsClassifier(n\_neighbors=best\_params['n\_neighbors'])  
 learningCurvePlot(tuned, data\_n)

# 5. Neural Network

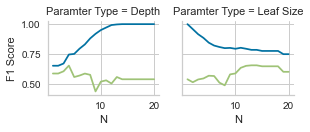
class NN():  
  
 def trainTest(self, data\_n):  
 df = []  
 for i in [x/100000 for x in [1, 10, 100, 1000, 10000, 100000]]:  
 model = MLPClassifier(max\_iter=300, learning\_rate\_init=i)  
 model.fit(scaled\_x\_train[data\_n],y\_train[data\_n])  
 y1\_predict\_train = model.predict(scaled\_x\_train[data\_n])  
 y1\_predict\_test = model.predict(scaled\_x\_test[data\_n])  
 df.append(["Train", "learning\_rate\_init", i, accuracy\_score(y\_train[data\_n], y1\_predict\_train)])  
 df.append(["Test", "learning\_rate\_init", i, accuracy\_score(y\_test[data\_n], y1\_predict\_test)])  
   
 for i in [x\*\*2 for x in range(2, 11)]:  
 model = MLPClassifier(max\_iter=300, hidden\_layer\_sizes=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(["Test", "hidden\_layer\_sizes", i, accuracy\_score(y\_test[data\_n], y1\_predict\_test)])  
 df.append(["Train", "hidden\_layer\_sizes", i, accuracy\_score(y\_train[data\_n], y1\_predict\_train)])  
  
 return pd.DataFrame(df, columns=["Sample Type", "Paramter Type", "X", "Accuracy"])  
   
 def hyperParameterTuning(self, data\_n):  
 param\_grid = {  
 'hidden\_layer\_sizes': [x\*\*2 for x in range(2, 11)],  
 'learning\_rate\_init': [x/100000 for x in [1, 10, 100, 1000, 10000, 100000]]  
 }  
 tuned = GridSearchCV(MLPClassifier(max\_iter=200), param\_grid = param\_grid, cv=10)  
   
 start\_time = time.time()  
 tuned.fit(scaled\_x\_train[data\_n], y\_train[data\_n])  
 training\_time = time.time() - start\_time  
 start\_time = time.time()  
 tuned.predict(scaled\_x\_test[data\_n])  
 testing\_time = time.time() - start\_time  
 return tuned.best\_score\_, tuned.best\_params\_, training\_time, testing\_time  
  
 def dataExplorePlot(self, df):  
 sns.pointplot(data=df[df["Paramter Type"] == "learning\_rate\_init"], x="X", y="Accuracy", hue="Sample Type" )  
 plt.figure()  
 sns.pointplot(data=df[df["Paramter Type"] == "hidden\_layer\_sizes"], x="X", y="Accuracy", hue="Sample Type" )  
   
   
 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

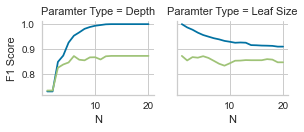
class Boost():  
 def trainTest(self, data\_n):  
 df = []  
 for i in range(1, 251, 10):  
 model = GradientBoostingClassifier(n\_estimators=i, max\_depth=3, min\_samples\_leaf=10)  
 model.fit(scaled\_x\_train[data\_n],y\_train[data\_n])  
 y1\_predict\_train = model.predict(scaled\_x\_train[data\_n])  
 y1\_predict\_test = model.predict(scaled\_x\_test[data\_n])  
 df.append(["Train", i, accuracy\_score(y\_train[data\_n], y1\_predict\_train)])  
 df.append(["Test", i, accuracy\_score(y\_test[data\_n], y1\_predict\_test)])  
 return pd.DataFrame(df, columns=["Sample Type", "Estimators", "Accuracy"])  
   
 def hyperParameterTuning(self, data\_n):  
 param\_grid = {'n\_estimators': range(1, 251, 10)}  
 tuned = GridSearchCV(GradientBoostingClassifier(max\_depth=3, min\_samples\_leaf=10), param\_grid, cv=10)  
   
 start\_time = time.time()  
 tuned.fit(scaled\_x\_train[data\_n], y\_train[data\_n])  
 training\_time = time.time() - start\_time  
 start\_time = time.time()  
 tuned.predict(scaled\_x\_test[data\_n])  
 testing\_time = time.time() - start\_time  
 return tuned.best\_score\_, tuned.best\_params\_, training\_time, testing\_time  
   
 def dataExplorePlot(self, df):  
 sns.lineplot(data=df, x="Estimators", y="Accuracy", hue="Sample Type")  
  
 def showLearningCurve(self, best\_params, data\_n):  
 tuned = GradientBoostingClassifier(n\_estimators=best\_params['n\_estimators'], max\_depth=3, min\_samples\_leaf=10)  
 learningCurvePlot(tuned, data\_n)

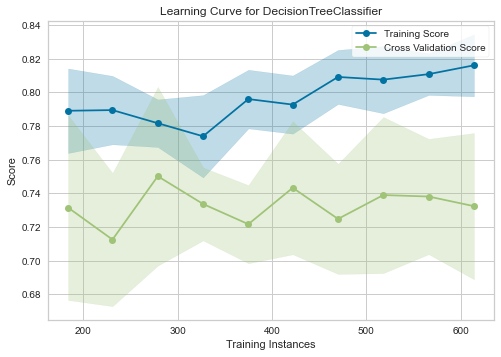
classifiers = [DTClassifier(), SupportVectorMachine(), KNN(), NN(), Boost()]  
#classifiers = [KNN()]  
  
classifiers\_name = ["Decision Tree", "Support Vector Machine", "K-Nearest Neighbors", "Neural Network", "Gradient Boosting"]  
  
i = 0  
  
accuracies = []  
train\_times = []  
test\_times = []  
  
for classifier in classifiers:  
 name = classifiers\_name[i]  
 print(str(i+1)+".", name, end="\n\n")  
 df1 = classifier.trainTest(0)  
 df2 = classifier.trainTest(1)  
 classifier.dataExplorePlot(df1)  
 plt.figure()  
 classifier.dataExplorePlot(df2)  
 plt.figure()  
  
 #best\_score, best\_params = fakeTunedData(i)  
 best\_score1, best\_params1, training\_time1, testing\_time1 = classifier.hyperParameterTuning(0)  
 best\_score2, best\_params2, training\_time2, testing\_time2 = classifier.hyperParameterTuning(1)  
 accuracies.append(["Dataset 1", best\_score1])  
 accuracies.append(["Dataset 2", best\_score2])  
 train\_times.append(["Dataset 1", training\_time1])  
 train\_times.append(["Dataset 2", training\_time2])  
 test\_times.append(["Dataset 1", testing\_time1])  
 test\_times.append(["Dataset 2", testing\_time2])  
 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:")  
   
 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.

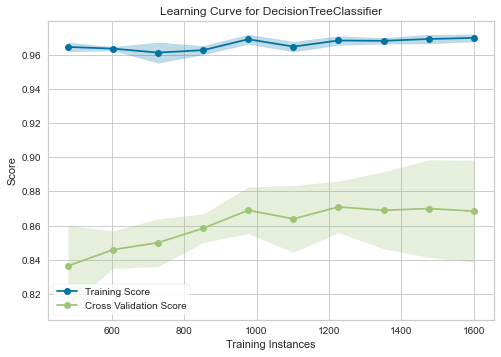


<Figure size 576x396 with 0 Axes>

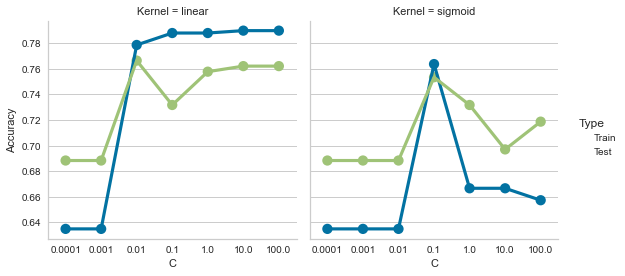




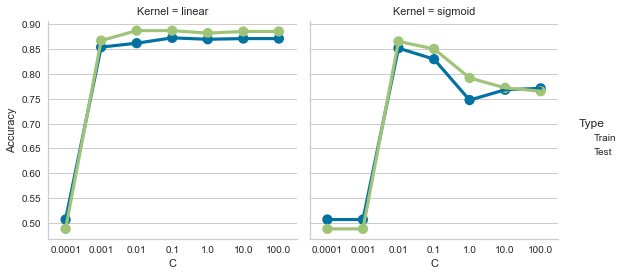
Dataset 2 Results:  
The optimal value of max\_depth was 10. The optimal value of min\_samples\_leaf was 3. Likewise the accuracy of Decision Tree classifier was 0.8764285714285714 when the optimized hyperparameter(s) value(s) were used.



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.

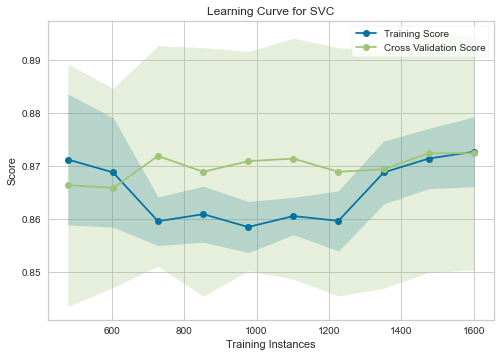


<Figure size 576x396 with 0 Axes>

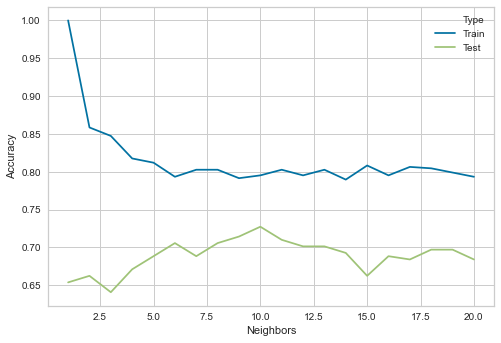


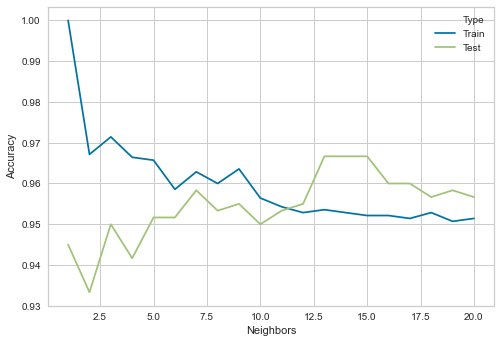


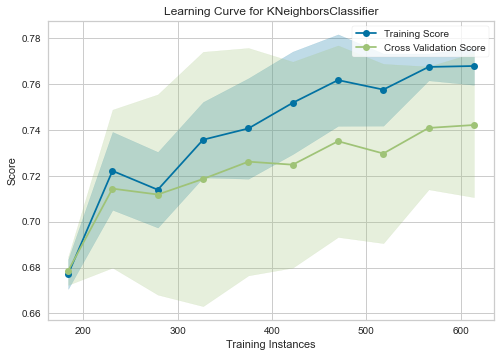
Dataset 2 Results:  
The optimal value of C was 100.0. The optimal value of kernel was linear. Likewise the accuracy of Support Vector Machine classifier was 0.8635714285714287 when the optimized hyperparameter(s) value(s) were used.



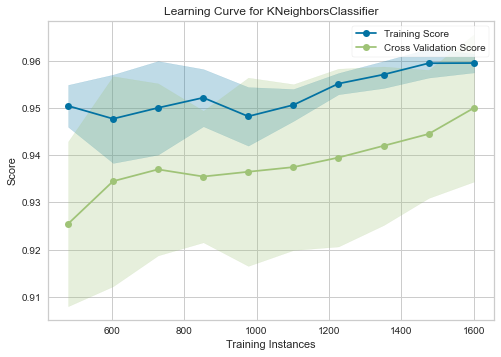
3. K-Nearest Neighbors  
  
3.1 Hyperparameters Exploration  
  
For this project, K-Nearest Neighbors will be hypertuned by adjusting: ['n\_neighbors']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:  
  
  
  
3.2 Hypertuning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



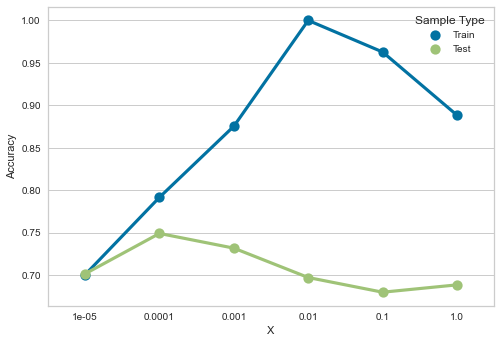


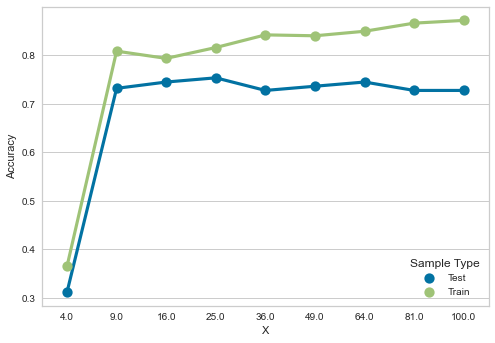


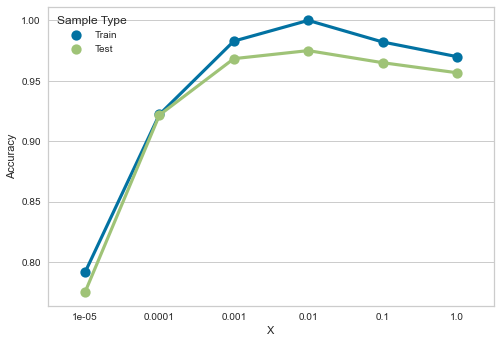
Dataset 2 Results:  
The optimal value of n\_neighbors was 11. Likewise the accuracy of K-Nearest Neighbors classifier was 0.9464285714285714 when the optimized hyperparameter(s) value(s) were used.

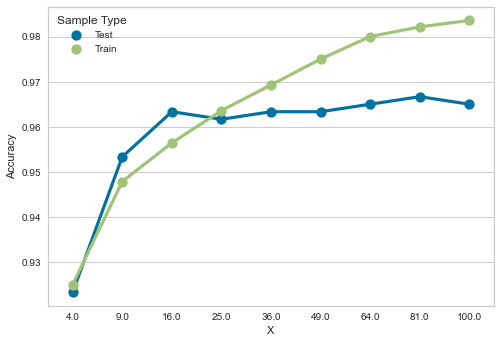


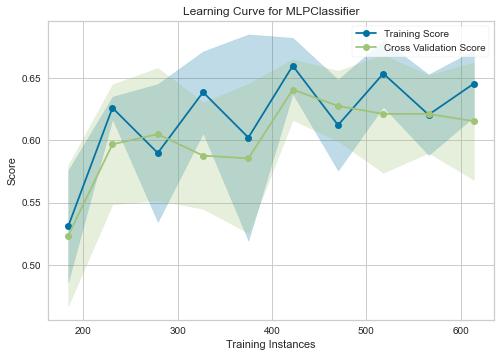
4. Neural Network  
  
4.1 Hyperparameters Exploration  
  
For this project, Neural Network will be hypertuned by adjusting: ['hidden\_layer\_sizes', 'learning\_rate\_init']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:  
  
  
  
4.2 Hypertuning  
  
GridSearchCV was performed for Neural Network classifier.  
  
Dataset 1 Results:  
The optimal value of hidden\_layer\_sizes was 25. The optimal value of learning\_rate\_init was 0.001. Likewise the accuracy of Neural Network classifier was 0.7841719077568134 when the optimized hyperparameter(s) value(s) were used.



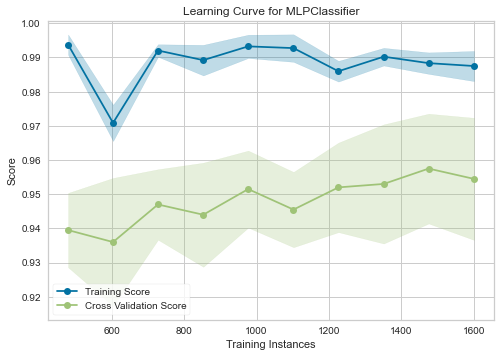




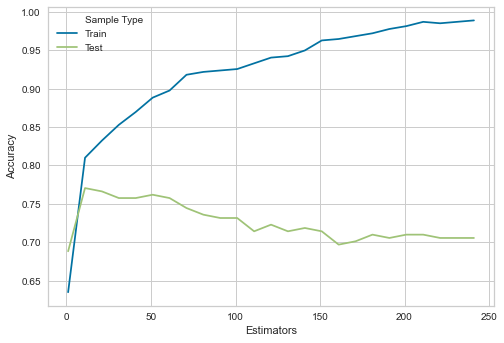


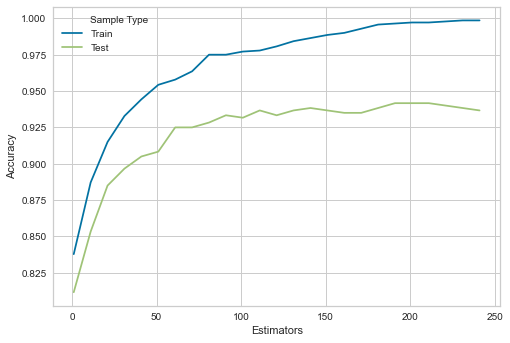


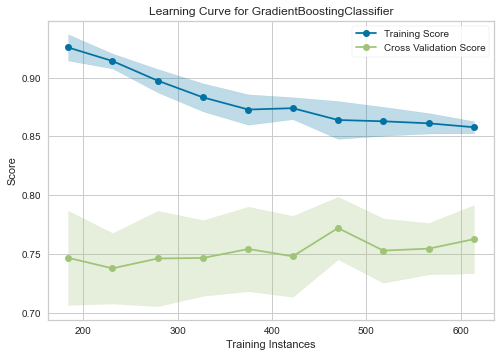
Dataset 2 Results:  
The optimal value of hidden\_layer\_sizes was 100. The optimal value of learning\_rate\_init was 0.01. Likewise the accuracy of Neural Network classifier was 0.9614285714285714 when the optimized hyperparameter(s) value(s) were used.



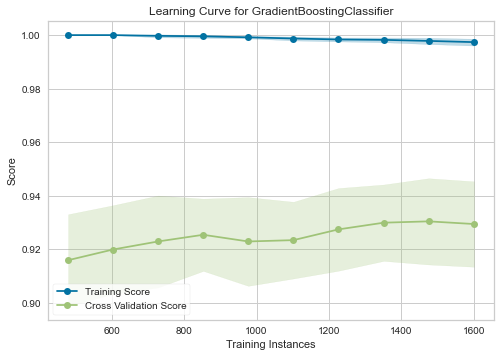
5. Gradient Boosting  
  
5.1 Hyperparameters Exploration  
  
For this project, Gradient Boosting will be hypertuned by adjusting: ['n\_estimators']. The following charts show how the accuracy is affected when the hyperparamter(s) are changed for both dataset 1 and dataset 2:  
  
  
  
5.2 Hypertuning  
  
GridSearchCV was performed for Gradient Boosting classifier.  
  
Dataset 1 Results:  
The optimal value of n\_estimators was 51. Likewise the accuracy of Gradient Boosting classifier was 0.7784765897973445 when the optimized hyperparameter(s) value(s) were used.







Dataset 2 Results:  
The optimal value of n\_estimators was 241. Likewise the accuracy of Gradient Boosting classifier was 0.927142857142857 when the optimized hyperparameter(s) value(s) were used.



train\_times

[['Dataset 1', 12.758336067199707],  
 ['Dataset 2', 18.173854112625122],  
 ['Dataset 1', 1.5860681533813477],  
 ['Dataset 2', 2.237246036529541],  
 ['Dataset 1', 0.6618459224700928],  
 ['Dataset 2', 1.1341099739074707],  
 ['Dataset 1', 119.77273082733154],  
 ['Dataset 2', 290.4396331310272],  
 ['Dataset 1', 45.655099868774414],  
 ['Dataset 2', 116.97686910629272]]

df\_accuracy = pd.DataFrame(accuracies, columns=["Dataset", "Accuracy %"])  
df\_train\_time = pd.DataFrame(train\_times, columns=["Dataset", "Time (s)"])  
df\_test\_time = pd.DataFrame(test\_times, columns=["Dataset", "Time (s)"])

df\_accuracy.head()

Dataset Accuracy %  
0 Dataset 1 0.763592  
1 Dataset 2 0.876429  
2 Dataset 1 0.782053  
3 Dataset 2 0.863571  
4 Dataset 1 0.769038

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

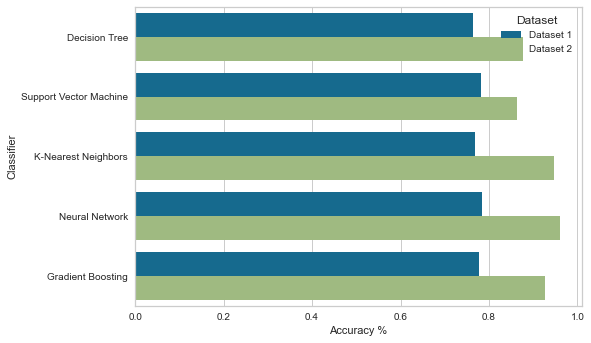
df\_accuracy['Classifier'] = names  
df\_train\_time['Classifier'] = names  
df\_test\_time['Classifier'] = names

df\_accuracy.head()

Dataset Accuracy % Classifier  
0 Dataset 1 0.763592 Decision Tree  
1 Dataset 2 0.876429 Decision Tree  
2 Dataset 1 0.782053 Support Vector Machine  
3 Dataset 2 0.863571 Support Vector Machine  
4 Dataset 1 0.769038 K-Nearest Neighbors

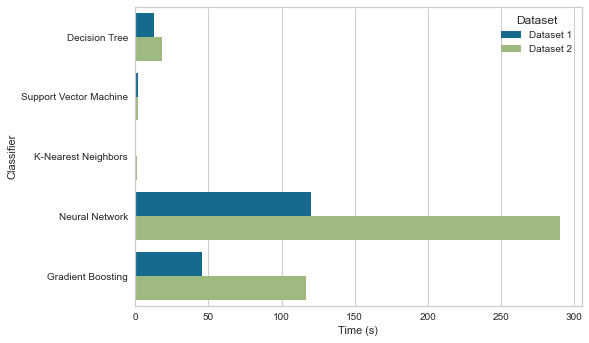
sns.barplot(data=df\_accuracy, y="Classifier", x="Accuracy %", hue="Dataset")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fe42748cc70>



sns.barplot(data=df\_train\_time, y="Classifier", x="Time (s)", hue="Dataset")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fe4276a3fa0>



sns.barplot(data=df\_test\_time, y="Classifier", x="Time (s)", hue="Dataset")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fe42799f0d0>

