# 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

# 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 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'  
rs = 614  
dataset = Data()  
x\_data,y\_data = dataset.dataAllocation(data)  
x\_train, x\_test, y\_train, y\_test = dataset.trainSets(x\_data,y\_data)  
scaler = StandardScaler()  
scaled\_x\_train = scaler.fit\_transform(x\_train)  
scaled\_x\_test = scaler.transform(x\_test)  
sizes = np.linspace(0.3, 1.0, 10)  
# print("Heatmap for Features")  
# data\_corr = sns.heatmap(pd.DataFrame(x\_train).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):  
 visualizer = LearningCurve(tuned, scoring='f1\_weighted', train\_sizes=sizes, n\_jobs=4)  
 visualizer.fit(x\_data,y\_data)  
 visualizer.show()

# 2. Decision Tree Classifier

class DTClassifier():  
  
 def trainTest(self):  
 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, y\_train)  
   
 y\_predict\_train = dt\_clf.predict(x\_train)  
 y\_predict\_test = dt\_clf.predict(x\_test)  
 df.append([i+1, j+1, "Train", f1\_score(y\_train, y\_predict\_train)])  
 df.append([i+1, j+1, "Test", f1\_score(y\_test, y\_predict\_test)])  
  
 return pd.DataFrame(df, columns=["Depth", "Leaf Size", "Sample Type", "F1 Score" ])  
  
 def hyperParameterTuning(self):  
 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, y\_train)  
 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):  
 tuned = DecisionTreeClassifier(max\_depth=best\_params['max\_depth'], min\_samples\_leaf=best\_params['min\_samples\_leaf'], random\_state=rs)  
 learningCurvePlot(tuned)

# 3. Support Vector Machine

class SupportVectorMachine():  
 def trainTest(self):  
 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,y\_train)  
 y\_predict\_train = model.predict(scaled\_x\_train)  
 y\_predict\_test = model.predict(scaled\_x\_test)  
 df.append([k, c, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns=["Kernel", "C", "Accuracy"])  
   
 def hyperParameterTuning(self):  
 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, y\_train)  
 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):  
 tuned = SVC(kernel=best\_params['kernel'], C=best\_params['C'])  
 learningCurvePlot(tuned)

# 4. KNN

class KNN():  
 def trainTest(self):  
 df = []  
 for i in range(20):  
 model = KNeighborsClassifier(n\_neighbors= i+1)  
 model.fit(x\_train, y\_train)  
 y\_predict\_test = model.predict(x\_test)  
 df.append([i+1, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns= ["Neighbors", "Accuracy"])  
   
 def hyperParameterTuning(self):  
 tuned = GridSearchCV(KNeighborsClassifier(), {"n\_neighbors" : range(1, 21)})  
 tuned.fit(x\_train, y\_train)  
 return tuned.best\_score\_, tuned.best\_params\_  
  
 def dataExplorePlot(self, df):  
 sns.lineplot(data=df, x="Neighbors", y="Accuracy")  
   
 def showLearningCurve(self, best\_params):  
 tuned = KNeighborsClassifier(n\_neighbors=best\_params['n\_neighbors'])  
 learningCurvePlot(tuned)

# 5. Neural Network

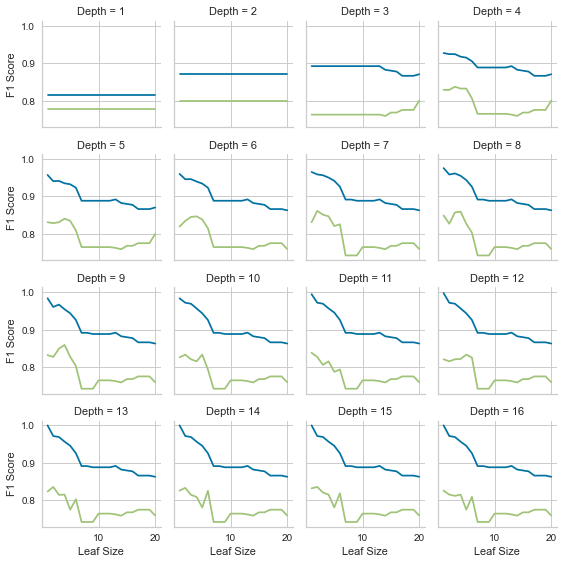
class NN():  
  
 def trainTest(self):  
 df = []  
 for i in [0.01, 0.02, 0.04, 0.08, 0.1]:  
 model = MLPClassifier(max\_iter=300, learning\_rate\_init=i)  
 model.fit(scaled\_x\_train,y\_train)  
 y\_predict\_train = model.predict(scaled\_x\_train)  
 y\_predict\_test = model.predict(scaled\_x\_test)  
 df.append([i, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns=["Learning Rate", "Accuracy"])  
   
 def hyperParameterTuning(self):  
 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=50), param\_grid = param\_grid, cv=10)  
 tuned.fit(scaled\_x\_train, y\_train)  
 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):  
 tuned = MLPClassifier(max\_iter=50, learning\_rate\_init=best\_params['learning\_rate\_init'], hidden\_layer\_sizes=best\_params['hidden\_layer\_sizes'])  
 learningCurvePlot(tuned)

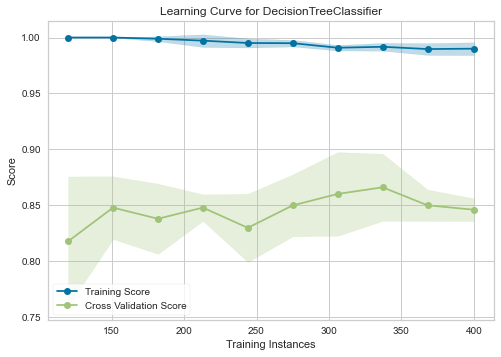
# 6. Boost

class Boost():  
 def trainTest(self):  
 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,y\_train)  
 y\_predict\_train = model.predict(scaled\_x\_train)  
 y\_predict\_test = model.predict(scaled\_x\_test)  
 df.append([i, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns=["Estimators", "Accuracy"])  
   
 def hyperParameterTuning(self):  
 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, y\_train)  
 return tuned.best\_score\_, tuned.best\_params\_  
   
 def dataExplorePlot(self, df):  
 sns.lineplot(data=df, x="Estimators", y="Accuracy")  
  
 def showLearningCurve(self, best\_params):  
 tuned = GradientBoostingClassifier(n\_estimators=best\_params['n\_estimators'], max\_depth=best\_params['max\_depth'], min\_samples\_leaf=100)  
 learningCurvePlot(tuned)

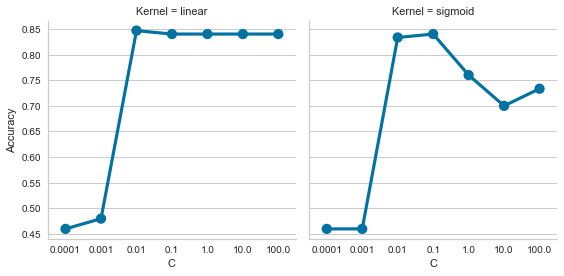
classifiers = [DTClassifier(), SupportVectorMachine(), KNN(), NN(), Boost()]  
classifiers\_name = ["Decision Tree", "Support Vector Machine", "K-Nearest Neighbors", "Neural Network", "Gradient Boosting"]  
  
i = 0  
  
for classifier in classifiers:  
 name = classifiers\_name[i]  
 print(str(i+1)+".", name, end="\n\n")  
 df = classifier.trainTest()  
 best\_score, best\_params = fakeTunedData(i)  
 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\_params], end=". ")  
 print("The following chart shows how the accuracy is affected when the hyperparamter(s) are changed:")  
  
 classifier.dataExplorePlot(df)  
 plt.figure()  
 #best\_score, best\_params = classifier.hyperParameterTuning()  
 #print(best\_score, best\_params)  
 print("\n\n")  
 print(str(i+1)+".2","Hypertuning", end="\n\n")  
 print("GridSearchCV was performed for", name, "classifier" , end=". ")  
 for key in best\_params:  
 print("The optimal value of", key, "was", best\_params[key], end = ". ")  
 print("Likewise the accuracy of", name, "classifier", "was", best\_score, "when the optimized hyperparameter(s) value(s) were used.")  
 classifier.showLearningCurve(best\_params)  
 print()  
 print()  
  
 i = i + 1

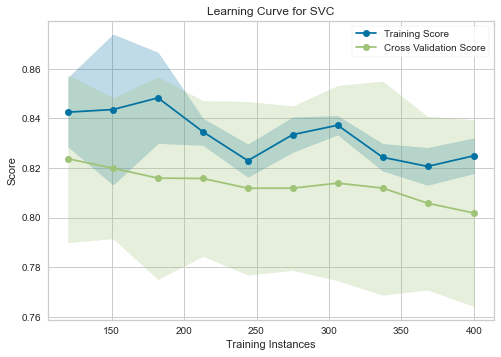
1. Decision Tree  
  
1.1 Hyperparameters Exploration  
  
For this project, Decision Tree will be hypertuned by adjusting: ['max\_depth', 'min\_samples\_leaf']. The following chart shows how the accuracy is affected when the hyperparamter(s) are changed:  
  
  
  
1.2 Hypertuning  
  
GridSearchCV was performed for Decision Tree classifier. The optimal value of max\_depth was 9. The optimal value of min\_samples\_leaf was 1. Likewise the accuracy of Decision Tree classifier was 72 when the optimized hyperparameter(s) value(s) were used.



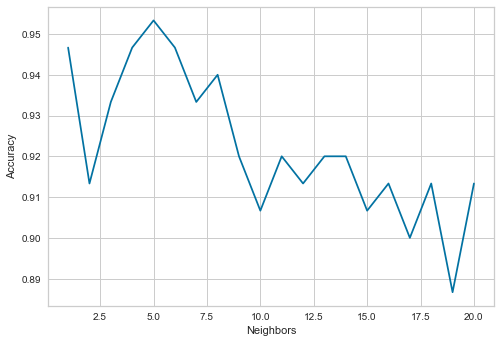


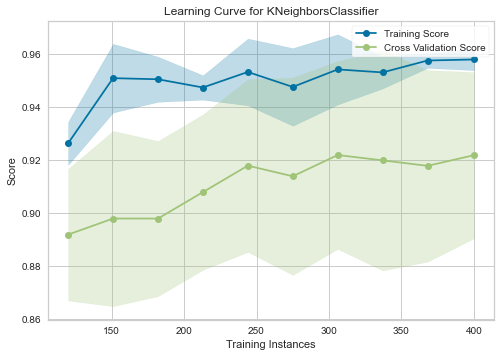
2. Support Vector Machine  
  
2.1 Hyperparameters Exploration  
  
For this project, Support Vector Machine will be hypertuned by adjusting: ['C', 'kernel']. The following chart shows how the accuracy is affected when the hyperparamter(s) are changed:  
  
  
  
2.2 Hypertuning  
  
GridSearchCV was performed for Support Vector Machine classifier. The optimal value of C was 0.01. The optimal value of kernel was linear. Likewise the accuracy of Support Vector Machine classifier was 73 when the optimized hyperparameter(s) value(s) were used.



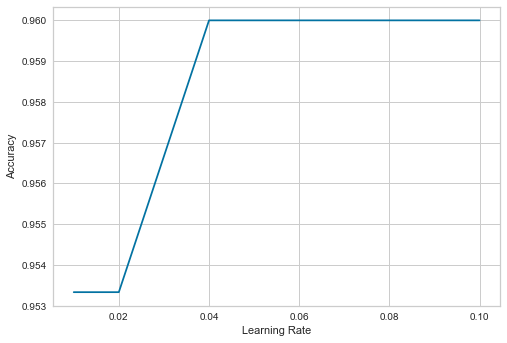


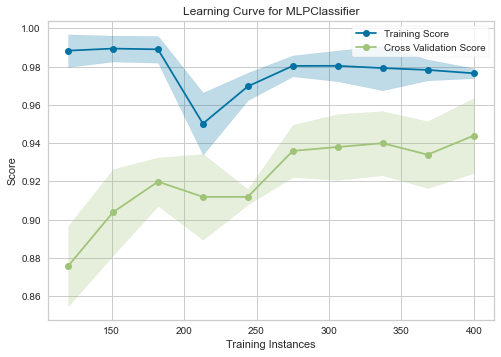
3. K-Nearest Neighbors  
  
3.1 Hyperparameters Exploration  
  
For this project, K-Nearest Neighbors will be hypertuned by adjusting: ['n\_neighbors']. The following chart shows how the accuracy is affected when the hyperparamter(s) are changed:  
  
  
  
3.2 Hypertuning  
  
GridSearchCV was performed for K-Nearest Neighbors classifier. The optimal value of n\_neighbors was 3. Likewise the accuracy of K-Nearest Neighbors classifier was 74 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 chart shows how the accuracy is affected when the hyperparamter(s) are changed:  
  
  
  
4.2 Hypertuning  
  
GridSearchCV was performed for Neural Network classifier. The optimal value of hidden\_layer\_sizes was 81. The optimal value of learning\_rate\_init was 0.04. Likewise the accuracy of Neural Network classifier was 75 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 chart shows how the accuracy is affected when the hyperparamter(s) are changed:  
  
  
  
5.2 Hypertuning  
  
GridSearchCV was performed for Gradient Boosting classifier. The optimal value of max\_depth was 3. The optimal value of n\_estimators was 90. Likewise the accuracy of Gradient Boosting classifier was 76 when the optimized hyperparameter(s) value(s) were used.

