import piplite

await piplite.install(['numpy'])

await piplite.install(['pandas'])

await piplite.install(['seaborn'])

# Pandas is a software library written for the Python programming language for data manipulation and analysis.

import pandas as pd

# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

import numpy as np

# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter function to plot data.

import matplotlib.pyplot as plt

#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

import seaborn as sns

# Preprocessing allows us to standarsize our data

from sklearn import preprocessing

# Allows us to split our data into training and testing data

from sklearn.model\_selection import train\_test\_split

# Allows us to test parameters of classification algorithms and find the best one

from sklearn.model\_selection import GridSearchCV

# Logistic Regression classification algorithm

from sklearn.linear\_model import LogisticRegression

# Support Vector Machine classification algorithm

from sklearn.svm import SVC

# Decision Tree classification algorithm

from sklearn.tree import DecisionTreeClassifier

# K Nearest Neighbors classification algorithm

from sklearn.neighbors import KNeighborsClassifier

def plot\_confusion\_matrix(y,y\_predict):

"this function plots the confusion matrix"

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y, y\_predict)

ax= plt.subplot()

sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

ax.set\_xlabel('Predicted labels')

ax.set\_ylabel('True labels')

ax.set\_title('Confusion Matrix');

ax.xaxis.set\_ticklabels(['did not land', 'land']); ax.yaxis.set\_ticklabels(['did not land', 'landed'])

plt.show()

from js import fetch

import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_2.csv"

resp1 = await fetch(URL1)

text1 = io.BytesIO((await resp1.arrayBuffer()).to\_py())

data = pd.read\_csv(text1)

URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_3.csv'

resp2 = await fetch(URL2)

text2 = io.BytesIO((await resp2.arrayBuffer()).to\_py())

X = pd.read\_csv(text2)

Y = data['Class'].to\_numpy()

data.head()

# students get this

transform = preprocessing.StandardScaler()

X = transform.fit\_transform(X)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 2)

Y\_test.shape

parameters ={'C':[0.01,0.1,1],

'penalty':['l2'],

'solver':['lbfgs']}

parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge

lr=LogisticRegression()

logreg\_cv = GridSearchCV(estimator = lr, param\_grid = parameters, cv = 10, scoring = 'accuracy', verbose = 2)

logreg\_cv.fit(X\_train,Y\_train)

print("tuned hpyerparameters :(best parameters) ",logreg\_cv.best\_params\_)

print("accuracy :",logreg\_cv.best\_score\_)

accuracy1 = logreg\_cv.score(X\_test, Y\_test)

accuracy1

yhat=logreg\_cv.predict(X\_test)

plot\_confusion\_matrix(Y\_test,yhat)

parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),

'C': np.logspace(-3, 3, 5),

'gamma':np.logspace(-3, 3, 5)}

svm = SVC()

svm\_cv = GridSearchCV(estimator = svm, param\_grid = parameters, cv = 10, scoring = 'accuracy', verbose = 2)

svm\_cv.fit(X\_train, Y\_train)

print("tuned hpyerparameters :(best parameters) ",svm\_cv.best\_params\_)

print("accuracy :",svm\_cv.best\_score\_)

accuracy2 = svm\_cv.score(X\_test, Y\_test)

accuracy2

yhat=svm\_cv.predict(X\_test)

plot\_confusion\_matrix(Y\_test,yhat)

parameters = {'criterion': ['gini', 'entropy'],

'splitter': ['best', 'random'],

'max\_depth': [2\*n for n in range(1,10)],

'max\_features': ['auto', 'sqrt'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10]}

tree = DecisionTreeClassifier()

tree\_cv = GridSearchCV(estimator = tree, param\_grid = parameters, cv = 10, scoring = 'accuracy', verbose = 2)

tree\_cv.fit(X\_train, Y\_train)

print("tuned hpyerparameters :(best parameters) ",tree\_cv.best\_params\_)

print("accuracy :",tree\_cv.best\_score\_)

tree\_cv.score(X\_test,Y\_test)

yhat = tree\_cv.predict(X\_test)

plot\_confusion\_matrix(Y\_test,yhat)

parameters = {'n\_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'algorithm': ['auto', 'ball\_tree', 'kd\_tree', 'brute'],

'p': [1,2]}

KNN = KNeighborsClassifier()

knn\_cv = GridSearchCV(estimator = KNN, param\_grid = parameters, cv = 10, scoring = 'accuracy', verbose = 2)

knn\_cv.fit(X\_train, Y\_train)

print("tuned hpyerparameters :(best parameters) ",knn\_cv.best\_params\_)

print("accuracy :",knn\_cv.best\_score\_)

knn\_cv.score(X\_test, Y\_test)

yhat = knn\_cv.predict(X\_test)

plot\_confusion\_matrix(Y\_test,yhat)

SVM, Linear Classification and KNN perform well, Decision Tree Classfier has more false positives.