import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from matplotlib import rcParams

from matplotlib.cm import rainbow

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

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn import datasets, linear\_model

from sklearn.metrics import accuracy\_score, make\_scorer

import serial

import numpy as np

from sklearn.preprocessing import StandardScaler

import time

time.sleep(2)

# Open the serial port

ser = serial.Serial('COM7', 9600)

# Create a scaler to preprocess the data

scaler = StandardScaler()

def takeInput():

while True:

# Read data from the serial port

data = ser.readline().decode().strip()

print(data)

# Preprocess the data

## print('Enter folloing')

## print()

## v=input('TEMP:')

## x=input('HUMD:')

## y=input('RAIN:')

## z=input('MOI:')

##

## print('')

## data=str(v)+','+str(x)+','+str(y)+','+str(z)

##

if(data is not None):

X = np.array([data.split(',')], dtype=np.float32)

#X = scaler.transform(X)

print(X)

# Make a prediction

y\_pred = regr.predict(X)

print(y\_pred)

if y\_pred == 1:

print('Pump On')

ser.write("1".encode())

elif y\_pred == 0:

print('Pump off')

ser.write("2".encode())

data = pd.read\_csv("mydata.csv")

y = data['Result']

X = data.drop(['Result'], axis = 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 0)

knn\_classifier = KNeighborsClassifier(n\_neighbors = 4)

knn\_classifier.fit(X\_train, y\_train)

knn\_preds = knn\_classifier.predict(X\_test)

knn\_acc = accuracy\_score(y\_test, knn\_preds)

print("Accuracy with KNN: ", accuracy\_score(y\_test, knn\_preds))

svc\_clf = SVC(gamma='scale')

svc\_clf.fit(X\_train,y\_train)

svc\_preds = svc\_clf.predict(X\_test)

svc\_acc = accuracy\_score(y\_test, svc\_preds)

print("Accuracy with SVC: ", accuracy\_score(y\_test, svc\_preds))

regr = LogisticRegression(solver="liblinear").fit(X\_train,y\_train)

regr\_preds = regr.predict(X\_test)

regr\_acc = accuracy\_score(y\_test, regr\_preds)

print("Accuracy with LR: ", accuracy\_score(y\_test, svc\_preds))

from sklearn.metrics import accuracy\_score, precision\_score

import matplotlib.pyplot as plt

import numpy as np

# Generate some random binary classification data

y\_true = np.random.randint(0, 2, size=100)

y\_pred = np.random.randint(0, 2, size=100)

# Calculate accuracy and precision for different threshold values

thresholds = np.linspace(0, 1, num=101)

accuracy = []

precision = []

for t in thresholds:

y\_pred\_t = (y\_pred >= t).astype(int)

accuracy.append(accuracy\_score(y\_true, y\_pred\_t))

precision.append(precision\_score(y\_true, y\_pred\_t))

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import numpy as np

# Generate some random binary classification data

y\_true = np.random.randint(0, 2, size=100)

y\_pred = np.random.randint(0, 2, size=100)

# Compute confusion matrix

cm = confusion\_matrix(y\_true, y\_pred)

# Plot confusion matrix

plt.imshow(cm, cmap=plt.cm.Blues)

plt.colorbar()

plt.title('Confusion Matrix')

plt.xlabel('Predicted label')

plt.ylabel('True label')

plt.xticks(np.arange(2), ('Negative', 'Positive'))

plt.yticks(np.arange(2), ('Negative', 'Positive'))

plt.show()

# Plot accuracy-precision curve

plt.plot(precision, accuracy)

plt.xlabel('Precision')

plt.ylabel('Accuracy')

plt.title('Accuracy-Precision Curve')

plt.show()

takeInput()



