"""

Implementation of Various classifiers using Scikit-learn and Implementing NaiveBayes from scratch

Below is our implementation and the data setup.

Input files - trainMatrixModified.txt, trainClasses.txt, testMatrixModified.txt, testClasses.txt

Output files - KNN.txt, SVM.txt, NB.txt,

               KNN-Graphs.jpg, SVM-Graphs.png, NB-Graphs.png,

               KNN-CM.png,SVM-CM.png, NB-CM.png

"""

import numpy as np

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score, precision\_score, f1\_score, \

    recall\_score

import math

def write\_metrics(pred, test, model):

    with open(model+'.txt', 'w') as wrt:

        wrt.write(f'Accuracy : {accuracy\_score(pred, test)}\n')

        wrt.write(f'Recall : {recall\_score(pred, test)}\n')

        wrt.write(f'Precision : {precision\_score(pred, test)}\n')

        wrt.write(f'Confusion Matrix : \n{confusion\_matrix(pred, test)}\n')

    metrics = [accuracy\_score(pred, test), recall\_score(pred, test), precision\_score(pred, test), f1\_score(pred, test)]

    headers = ['Accuracy', 'Recall', 'Precision', 'F1\_score']

    plt.figure()

    plt.bar(headers, metrics)

    # Add labels and title

    plt.xlabel('Metrics')

    plt.ylabel('Result')

    plt.title('Evaluation Metrics for '+model)

    plt.savefig(f'{model}\_EM.png')

    plt.figure(figsize=(8, 6))

    ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(pred, test)).plot()

    plt.savefig(model + '\_CM.png')

print('Reading files using numpy')

trainData\_Matrix = np.loadtxt('trainMatrixModified.txt').transpose()

trainData\_Classes = np.loadtxt('trainClasses.txt')

testData\_Matrix = np.loadtxt('testMatrixModified.txt').transpose()

testData\_Classes = np.loadtxt('testClasses.txt')

print('Started Classification for SVM')

classifier\_svm = SVC(kernel='linear').fit(trainData\_Matrix, trainData\_Classes[:, 1])

p\_svm = classifier\_svm.predict(testData\_Matrix)

write\_metrics(p\_svm,testData\_Classes[:, 1], 'SVM')

print('Started Classification for KNN')

classifier\_knn = KNeighborsClassifier(n\_neighbors=4).fit(trainData\_Matrix, trainData\_Classes[:, 1])

p\_knn = classifier\_knn.predict(testData\_Matrix)

write\_metrics(p\_knn,testData\_Classes[:, 1], 'KNN')

def TRAINMULTINOMIALNB(dataset\_train,labels\_train):

    samples\_WIN=np.count\_nonzero(labels\_train==0)

    samples\_Hock=np.count\_nonzero(labels\_train)

    WinConcatinates=[]

    HockConcatinates=[]

    i=0

    while i <labels\_train.size:

        if(labels\_train[i]==0.00):

            WinConcatinates.append(dataset\_train[i,:])

        else:

            HockConcatinates.append(dataset\_train[i,:])

        i+=1

    ColumnSumInWIn=np.sum(WinConcatinates,0)

    ColumnSumInHock=np.sum(HockConcatinates,0)

    return [((ColumnSumInWIn+1)/np.sum((ColumnSumInWIn+1))),((ColumnSumInHock+1)/np.sum((ColumnSumInHock+1)))],[(samples\_WIN/dataset\_train.shape[0]),(samples\_Hock/dataset\_train.shape[0])]

def APPLYMULTINOMIALNB(testDataSet,Conditional\_Probability\_Train,priorTrain):

    PriorTrainScoreWin=(math.log(priorTrain[0]))

    PriorTrainScoreHock=(math.log(priorTrain[1]))

    testPreds=[]

    for testDataSetRecord in testDataSet:

        TempVarWin=PriorTrainScoreHock

        TempVarHock=PriorTrainScoreWin

        for i in range(len(testDataSetRecord)):

            if(testDataSetRecord[i]>=1.0):

                TempVarHock+=math.log(Conditional\_Probability\_Train[0][i])

                TempVarWin+=math.log(Conditional\_Probability\_Train[1][i])

        testPreds.append(1 if TempVarWin>TempVarHock else 0)

    return testPreds

cond\_prob, p\_prob = TRAINMULTINOMIALNB(trainData\_Matrix, trainData\_Classes[:, 1])

p\_nb = APPLYMULTINOMIALNB(testData\_Matrix,cond\_prob,p\_prob)

write\_metrics(p\_nb,testData\_Classes[:, 1], 'NB')