#Importing necessary Libraries

import numpy as np

import scipy.io.wavfile as wav

from python\_speech\_features import mfcc

import os

import pickle

import operator

#Defining the necessary functions for creating a dataset for KNN matching.

#Define a function to get the distance between feature vectors and find neighbors:

def distance(instance1 , instance2 , k ):

distance =0

mm1 = instance1[0]

cm1 = instance1[1]

mm2 = instance2[0]

cm2 = instance2[1]

#Method to calculate distance between two instances.

distance = np.trace(np.dot(np.linalg.inv(cm2), cm1))

distance+=(np.dot(np.dot((mm2-mm1).transpose() , np.linalg.inv(cm2)) , mm2-mm1 ))

distance+= np.log(np.linalg.det(cm2)) - np.log(np.linalg.det(cm1))

distance-= k

return distance

#This function returns a list of K nearest neighbours for any instance

#to be checked within a given dataset (dataset of features.)

def getNeighbors(trainingSet , instance , k):

distances =[]

for x in range (len(trainingSet)):

dist = distance(trainingSet[x], instance, k )+ distance(instance, trainingSet[x], k)

distances.append((trainingSet[x][2], dist))

distances.sort(key=operator.itemgetter(1))

neighbors = []

for x in range(k):

neighbors.append(distances[x][0])

return neighbors

#Identify the nearest neighbors:

def nearestClass(neighbors):

classVote = {}

for x in range(len(neighbors)):

response = neighbors[x]

if response in classVote:

classVote[response]+=1

else:

classVote[response]=1

sorter = sorted(classVote.items(), key = operator.itemgetter(1), reverse=True)

return sorter[0][0]

#Extract features from the data (audio files) and dump these features

#into a binary .dat file “my.dat”:

#directory = "C:/User/harat/Downloads/mgct"

f = open("my.dat" ,'wb')

i = 0

#Dataset creation : making a dat file were we get all the data about the audio files in a ".dat" file.

for folder in os.listdir():

#as we have 10 classes, we're starting the loop from 1 to 11

#so that we run the loop for total of 10 times, with each folder change (genre change), i (label) changes.

i += 1

if i==11 :

break

for file in os.listdir(folder):

#To read an Wav audio File in Python

(rate,sig) = wav.read(folder+"/"+file)

#MFCC is the feature we will use for our analysis,

#because it provides data about the overall shape of the audio frequencies.

mfcc\_feat = mfcc(sig,rate ,winlen=0.020, appendEnergy = False)

covariance = np.cov(np.matrix.transpose(mfcc\_feat))

mean\_matrix = mfcc\_feat.mean(0)

#making a feature typle that contains the mean matrix from mfcc as well as covariance,

#and last variable in the feature tuple is the label (where numbers correspond to particular genre)

feature = (mean\_matrix , covariance , i)

#This the the created dataset, which takes the input from specified path

#it then stores the feature as a .dat file which can be used later without having to train all the files over it.

pickle.dump(feature , f)

f.close()

#Loading the created dataset into a python readable object (list)

dataset = []

def loadDataset(filename):

with open("my.dat" , 'rb') as f:

while True:

try:

dataset.append(pickle.load(f))

except EOFError:

f.close()

break

loadDataset("my.dat")

#we have to convert the dataset from a list to np array.

dataset = np.array(dataset)

#type(dataset) ##uncomment this line to check the type of (dataset),

#Train and test split on the dataset:

#as the dataset contains features for all the audio files,

#we have to split that manually into train and test data

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

x\_train ,x\_test = train\_test\_split(dataset,test\_size=0.15)

#Make prediction using KNN and get the accuracy on test data:

leng = len(x\_test)

predictions = []

for x in range (leng):

predictions.append(nearestClass(getNeighbors(x\_train ,x\_test[x] , 8)))

#Define a function for model evaluation:

def getAccuracy(testSet, predictions):

#this is a variable to count total number of correct predictions.

correct = 0

for x in range (len(testSet)):

if testSet[x][-1]==predictions[x]:

correct+=1

return 1.0\*correct/len(testSet)

#Print accuracy using defined function

accuracy1 = getAccuracy(x\_test , predictions)

print(accuracy1)