ASSIGNMENT 5 - CONFUSION MATRIX

19103112-SHUBHJOT SINGH

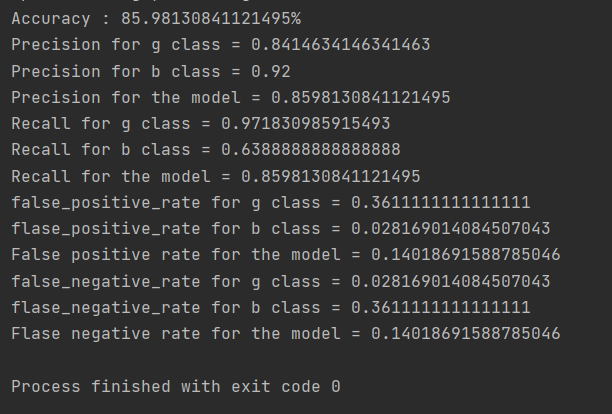
QUESTION:

Estimate the precision, recall, accuracy, false negative rate and false positive rate of K-NN classifier on the given data sets using k-fold cross validation.  
  
There are 3 different data sets (Ionosphere, Indian Liver Patient, Iris).

CODE for Ionosphere model:

import csv  
import random  
import operator  
import math  
import matplotlib.pyplot as plt  
testsetname=[]  
def loaddataset(filename,split,trainingset=[],testset=[]):  
 with open(filename,'r')as csvfile:  
 lines=csv.reader(csvfile)  
 dataset=list(lines)  
 for x in range(len(dataset)):  
 for y in range(34):  
 dataset[x][y]=float(dataset[x][y])  
 if random.random()<split:  
 trainingset.append(dataset[x])  
 else:  
 testset.append(dataset[x])  
 testsetname.append(dataset[x][-1])  
  
  
  
def euclidiandistance(instance1,instance2,length):  
 distance=0  
 for x in range(length):  
 distance+=pow((instance1[x]-instance2[x]),2)  
  
 return math.sqrt(distance)  
  
  
def getneighbor(trainingset,testinstance,k):  
 distance=[]  
 length=len(testinstance)-1  
 for x in range(len(trainingset)):  
 dist=euclidiandistance(testinstance,trainingset[x],length)  
 distance.append((trainingset[x],dist))  
 distance.sort(key=operator.itemgetter(1))  
 neighbors=[]  
 for x in range(k):  
 neighbors.append(distance[x][0])  
 return neighbors  
  
  
  
def getresponse(neighbors):  
 classvotes={}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classvotes:  
 classvotes[response]+=1  
 else:  
 classvotes[response]=1  
  
 sortedvotes=sorted(classvotes.items(),key=operator.itemgetter(1),reverse=True)  
 return sortedvotes[0][0]  
  
  
  
def getaccuracy(testset,predictions):  
 correct=0  
 for x in range(len(testset)):  
 if testset[x][-1] == predictions[x]:  
 correct+=1  
 return (correct/float(len(testset)))\*100.0  
  
def precision(testset,predictions,key):  
 TP=0  
 FP=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]==key:  
 TP+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]!=key and predictions[y]==key:  
 FP+=1  
 return (TP/(TP+FP),TP,FP)  
  
  
  
def recall(testset,predictions,key):  
 TP=0  
 FN=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]==key:  
 TP+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]==key and predictions[y]!=key:  
 FN+=1  
 return (TP/(TP+FN),TP,FN)  
  
  
def false\_positive\_rate(testset,predeictions,key):  
 FP=0  
 TN=0  
 for x in range(len(testset)):  
 if testset[x][-1]!=key and predictions[x]==key:  
 FP+=1  
  
  
 for y in range(len(testset)):  
 if testset[y][-1]!=key and predictions[y]!=key:  
 TN+=1  
 return (FP/(FP+TN),FP,TN)  
  
  
  
def false\_negative\_rate(testset,predictions,key):  
 FN=0  
 TP=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]!=key:  
 FN+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]==key and predictions[y]==key:  
 TP+=1  
 return (FN/(FN+TP),FN,TP)  
  
  
  
  
  
  
  
  
  
def plot\_graph(testset,predictions):  
 x\_axis1=[]  
 x\_axis2=[]  
 for i in range(len(predictions)):  
 x\_axis1.append(i)  
 for i in range(len(x\_axis1)):  
 x\_axis2.append(i)  
  
 plt.plot(x\_axis1,predictions,label="Predicted")  
 plt.plot(x\_axis1,testsetname,label="Expexted")  
 plt.legend()  
 plt.show()  
  
  
  
  
  
trainingset=[]  
testset=[]  
split=0.70  
loaddataset('space.csv',split,trainingset,testset)  
print('Train set : '+repr(len(trainingset)))  
print('test set : '+repr(len(testset)))  
  
#generate predictions  
predictions=[]  
k=5  
for x in range(len(testset)):  
 neighbors=getneighbor(trainingset,testset[x],k)  
 result=getresponse(neighbors)  
 predictions.append(result)  
 print('>predicted='+repr(result)+',actual='+repr(testset[x][-1]))  
accuracy=getaccuracy(testset,predictions)  
#plot\_graph(testset,predictions)  
print('Accuracy : '+repr(accuracy)+'%')  
  
  
#PRECISION  
preci\_g ,TP\_g,FP\_g= precision(testset,predictions,'g')  
preci\_b ,TP\_b,FP\_b= precision(testset,predictions,'b')  
TP=TP\_b+TP\_g  
FP=FP\_b+FP\_g  
print('Precision for g class = '+repr(preci\_g))  
print('Precision for b class = '+repr(preci\_b))  
print('Precision for the model = '+repr(TP/(TP+FP)))  
  
#RECALL  
  
recall\_g,TP\_g,FN\_g = recall(testset,predictions,'g')  
recall\_b,TP\_b,FN\_b= recall(testset,predictions,'b')  
TP=TP\_g+TP\_b  
FN=FN\_b+FN\_g  
print('Recall for g class = '+repr(recall\_g))  
print('Recall for b class = '+repr(recall\_b))  
print('Recall for the model = '+repr(TP/(TP+FN)))  
  
  
#FALSE\_POSITIVE\_RATE  
false\_positive\_g,FP\_g,TN\_g = false\_positive\_rate(testset,predictions,'g')  
false\_positive\_b,FP\_b,TN\_b = false\_positive\_rate(testset,predictions,'b')  
FP=FP\_b+FP\_g  
TN=TN\_b+TN\_g  
print('false\_positive\_rate for g class = '+repr(false\_positive\_g))  
print('flase\_positive\_rate for b class = '+repr(false\_positive\_b))  
print('False positive rate for the model = '+repr(FP/(TN+FP)))  
  
#FALSE\_NEGATIVE\_RATE  
false\_negative\_g,FN\_g,TP\_g= false\_negative\_rate(testset,predictions,'g')  
false\_negative\_b,FN\_b,TP\_b= false\_negative\_rate(testset,predictions,'b')  
FN=FN\_b+FN\_g  
TP=TP\_b+TP\_g  
print('false\_negative\_rate for g class = '+repr(false\_negative\_g))  
print('flase\_negative\_rate for b class = '+repr(false\_negative\_b))  
print('Flase negative rate for the model = '+repr(FN/(TP+FN)))

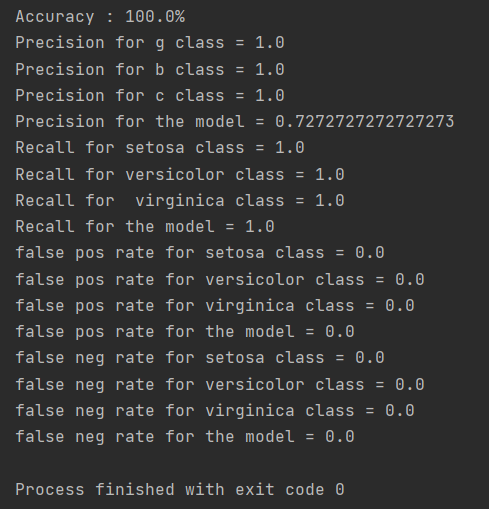
OUTPUT FOR IONOSPHERE MODEL:



CODE FOR IRIS DATASET MODEL:

import csv  
import random  
import operator  
import math  
import matplotlib.pyplot as plt  
testsetname=[]  
def loaddataset(filename,split,trainingset=[],testset=[]):  
 with open(filename,'r')as csvfile:  
 lines=csv.reader(csvfile)  
 dataset=list(lines)  
 for x in range(len(dataset)):  
 for y in range(4):  
 dataset[x][y]=float(dataset[x][y])  
 if random.random()<split:  
 trainingset.append(dataset[x])  
 else:  
 testset.append(dataset[x])  
 testsetname.append(dataset[x][-1])  
  
  
  
def euclidiandistance(instance1,instance2,length):  
 distance=0  
 for x in range(length):  
 distance+=pow((instance1[x]-instance2[x]),2)  
  
 return math.sqrt(distance)  
  
  
def getneighbor(trainingset,testinstance,k):  
 distance=[]  
 length=len(testinstance)-1  
 for x in range(len(trainingset)):  
 dist=euclidiandistance(testinstance,trainingset[x],length)  
 distance.append((trainingset[x],dist))  
 distance.sort(key=operator.itemgetter(1))  
 neighbors=[]  
 for x in range(k):  
 neighbors.append(distance[x][0])  
 return neighbors  
  
  
  
def getresponse(neighbors):  
 classvotes={}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classvotes:  
 classvotes[response]+=1  
 else:  
 classvotes[response]=1  
  
 sortedvotes=sorted(classvotes.items(),key=operator.itemgetter(1),reverse=True)  
 return sortedvotes[0][0]  
  
  
  
def getaccuracy(testset,predictions):  
 correct=0  
 for x in range(len(testset)):  
 if testset[x][-1] == predictions[x]:  
 correct+=1  
 return (correct/float(len(testset)))\*100.0  
  
def precision(testset,predictions,key):  
 TP=0  
 FP=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]==key:  
 TP+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]!=key and predictions[y]==key:  
 FP+=1  
 return (TP/(TP+FP),TP,FP)  
  
  
  
def recall(testset,predictions,key):  
 TP=0  
 FN=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]==key:  
 TP+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]==key and predictions[y]!=key:  
 FN+=1  
 return (TP/(TP+FN),TP,FN)  
  
  
def false\_positive\_rate(testset,predeictions,key):  
 FP=0  
 TN=0  
 for x in range(len(testset)):  
 if testset[x][-1]!=key and predictions[x]==key:  
 FP+=1  
  
  
 for y in range(len(testset)):  
 if testset[y][-1]!=key and predictions[y]!=key:  
 TN+=1  
 return (FP/(FP+TN),FP,TN)  
  
  
  
def false\_negative\_rate(testset,predictions,key):  
 FN=0  
 TP=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]!=key:  
 FN+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]==key and predictions[y]==key:  
 TP+=1  
 return (FN/(FN+TP),FN,TP)  
  
  
  
  
  
  
  
  
  
def plot\_graph(testset,predictions):  
 x\_axis1=[]  
 x\_axis2=[]  
 for i in range(len(predictions)):  
 x\_axis1.append(i)  
 for i in range(len(x\_axis1)):  
 x\_axis2.append(i)  
  
 plt.plot(x\_axis1,predictions,label="Predicted")  
 plt.plot(x\_axis1,testsetname,label="Expexted")  
 plt.legend()  
 plt.show()  
  
  
  
  
  
trainingset=[]  
testset=[]  
split=0.70  
loaddataset('iris.csv',split,trainingset,testset)  
print('Train set : '+repr(len(trainingset)))  
print('test set : '+repr(len(testset)))  
  
#generate predictions  
predictions=[]  
k=5  
for x in range(len(testset)):  
 neighbors=getneighbor(trainingset,testset[x],k)  
 result=getresponse(neighbors)  
 predictions.append(result)  
 print('>predicted='+repr(result)+',actual='+repr(testset[x][-1]))  
accuracy=getaccuracy(testset,predictions)  
#plot\_graph(testset,predictions)  
print('Accuracy : '+repr(accuracy)+'%')  
  
  
#PRECISION  
preci\_a,TP\_a,FP\_a= precision(testset,predictions,'setosa')  
preci\_b,TP\_b,FP\_b= precision(testset,predictions,'versicolor')  
preci\_c,TP\_c,FP\_c= precision(testset,predictions,'virginica')  
  
TP=TP\_a+TP\_b+TP\_c  
FP=FP\_a+FP\_b+TP\_c  
print('Precision for g class = '+repr(preci\_a))  
print('Precision for b class = '+repr(preci\_b))  
print('Precision for c class = '+repr(preci\_c))  
print('Precision for the model = '+repr(TP/(TP+FP)))  
  
#RECALL  
  
recall\_a,TP\_a,FN\_a= recall(testset,predictions,'setosa')  
recall\_b,TP\_b,FN\_b= recall(testset,predictions,'versicolor')  
recall\_c,TP\_c,FN\_c= recall(testset,predictions,'virginica')  
  
TP=TP\_a+TP\_b+TP\_c  
FN=FN\_a+FN\_b+FN\_c  
print('Recall for setosa class = '+repr(recall\_a))  
print('Recall for versicolor class = '+repr(recall\_b))  
print('Recall for virginica class = '+repr(recall\_c))  
print('Recall for the model = '+repr(TP/(TP+FN)))  
  
  
#FALSE\_POSITIVE\_RATE  
false\_pos\_a,FP\_a,TN\_a= false\_positive\_rate(testset,predictions,'setosa')  
false\_pos\_b,FP\_b,TN\_b= false\_positive\_rate(testset,predictions,'versicolor')  
false\_pos\_c,FP\_c,TN\_c= false\_positive\_rate(testset,predictions,'virginica')  
  
FP=FP\_a+FP\_b+FP\_c  
TN=TN\_a+TN\_b+TN\_c  
print('false pos rate for setosa class = '+repr(false\_pos\_a))  
print('false pos rate for versicolor class = '+repr(false\_pos\_b))  
print('false pos rate for virginica class = '+repr(false\_pos\_c))  
print('false pos rate for the model = '+repr(FP/(TN+FP)))  
  
#FALSE\_NEGATIVE\_RATE  
false\_neg\_a,FN\_a,TP\_a= false\_negative\_rate(testset,predictions,'setosa')  
false\_neg\_b,FN\_b,TP\_b= false\_negative\_rate(testset,predictions,'versicolor')  
false\_neg\_c,FN\_c,TP\_c= false\_negative\_rate(testset,predictions,'virginica')  
  
FN=FN\_a+FN\_b+FN\_c  
TP=TP\_a+TP\_b+TP\_c  
print('false neg rate for setosa class = '+repr(false\_neg\_a))  
print('false neg rate for versicolor class = '+repr(false\_neg\_b))  
print('false neg rate for virginica class = '+repr(false\_neg\_c))  
print('false neg rate for the model = '+repr(FN/(TP+FN)))

OUTPUT FOR IRIS DATASET MODEL:



CODE FOR LIVER DATASET MODEL:

import csv  
import random  
import operator  
import math  
import matplotlib.pyplot as plt  
testsetname=[]  
def loaddataset(filename,split,trainingset=[],testset=[]):  
 with open(filename,'r')as csvfile:  
 lines=csv.reader(csvfile)  
 dataset=list(lines)  
 for x in range(len(dataset)):  
 for y in range(9):  
 dataset[x][y]=float(dataset[x][y])  
 if random.random()<split:  
 trainingset.append(dataset[x])  
 else:  
 testset.append(dataset[x])  
 testsetname.append(dataset[x][-1])  
  
  
  
def euclidiandistance(instance1,instance2,length):  
 distance=0  
 for x in range(length):  
 distance+=pow((instance1[x]-instance2[x]),2)  
  
 return math.sqrt(distance)  
  
  
def getneighbor(trainingset,testinstance,k):  
 distance=[]  
 length=len(testinstance)-1  
 for x in range(len(trainingset)):  
 dist=euclidiandistance(testinstance,trainingset[x],length)  
 distance.append((trainingset[x],dist))  
 distance.sort(key=operator.itemgetter(1))  
 neighbors=[]  
 for x in range(k):  
 neighbors.append(distance[x][0])  
 return neighbors  
  
  
  
def getresponse(neighbors):  
 classvotes={}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classvotes:  
 classvotes[response]+=1  
 else:  
 classvotes[response]=1  
  
 sortedvotes=sorted(classvotes.items(),key=operator.itemgetter(1),reverse=True)  
 return sortedvotes[0][0]  
  
  
  
def getaccuracy(testset,predictions):  
 correct=0  
 for x in range(len(testset)):  
 if testset[x][-1] == predictions[x]:  
 correct+=1  
 return (correct/float(len(testset)))\*100.0  
  
def precision(testset,predictions,key):  
 TP=0  
 FP=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]==key:  
 TP+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]!=key and predictions[y]==key:  
 FP+=1  
 return (TP/(TP+FP))  
  
  
  
def recall(testset,predictions,key):  
 TP=0  
 FN=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]==key:  
 TP+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]==key and predictions[y]!=key:  
 FN+=1  
 return (TP/(TP+FN))  
  
  
def false\_positive\_rate(testset,predeictions,key):  
 FP=0  
 TN=0  
 for x in range(len(testset)):  
 if testset[x][-1]!=key and predictions[x]==key:  
 FP+=1  
  
  
 for y in range(len(testset)):  
 if testset[y][-1]!=key and predictions[y]!=key:  
 TN+=1  
 return (FP/(FP+TN))  
  
  
  
def false\_negative\_rate(testset,predictions,key):  
 FN=0  
 TP=0  
 for x in range(len(testset)):  
 if testset[x][-1]==key and predictions[x]!=key:  
 FN+=1  
  
 for y in range(len(testset)):  
 if testset[y][-1]==key and predictions[y]==key:  
 TP+=1  
 return (FN/(FN+TP))  
  
  
  
  
  
  
  
  
  
def plot\_graph(testset,predictions):  
 x\_axis1=[]  
 x\_axis2=[]  
 for i in range(len(predictions)):  
 x\_axis1.append(i)  
 for i in range(len(x\_axis1)):  
 x\_axis2.append(i)  
  
 plt.plot(x\_axis1,predictions,label="Predicted")  
 plt.plot(x\_axis1,testsetname,label="Expexted")  
 plt.legend()  
 plt.show()  
  
  
  
  
  
trainingset=[]  
testset=[]  
split=0.70  
loaddataset('liver.csv',split,trainingset,testset)  
print('Train set : '+repr(len(trainingset)))  
print('test set : '+repr(len(testset)))  
  
#generate predictions  
predictions=[]  
k=5  
for x in range(len(testset)):  
 neighbors=getneighbor(trainingset,testset[x],k)  
 result=getresponse(neighbors)  
 predictions.append(result)  
 print('>predicted='+repr(result)+',actual='+repr(testset[x][-1]))  
accuracy=getaccuracy(testset,predictions)  
#plot\_graph(testset,predictions)  
print('Accuracy : '+repr(accuracy)+'%')  
  
  
#PRECISION  
preci= precision(testset,predictions,2)  
print('Precision for model = '+repr(preci))  
  
  
#RECALL  
  
recall = recall(testset,predictions,2)  
print('Recall for model = '+repr(recall))  
  
  
#FALSE\_POSITIVE\_RATE  
false\_positive= false\_positive\_rate(testset,predictions,2)  
print('false\_positive\_rate for model= '+repr(false\_positive))  
  
#FALSE\_NEGATIVE\_RATE  
false\_negative= false\_negative\_rate(testset,predictions,2)  
print('false\_negative\_rate for model= '+repr(false\_negative))

OUTPUT FOR LIVER DATASET MODEL:

