1. **Pen-and-paper**

**1)**

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

**2)**

**Uma imagem com texto

Descrição gerada automaticamente**

**Uma imagem com texto

Descrição gerada automaticamente**

**3)**

**Uma imagem com texto, quadro branco

Descrição gerada automaticamente**

**4)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X** | **0,0** | **0,1** | **0,2** | **0,3** | **0,4** | **0,5** | **0,6** | **0,7** | **0,8** | **0,9** | **1,0** |
|  | **FP** | **FP** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** |
|  | **FP** | **FP** | **FP** | **FP** | **FP** | **FP** | **FP** | **FP** | **TN** | **TN** | **TN** |
|  | **FP** | **FP** | **FP** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** |
|  | **FP** | **FP** | **FP** | **FP** | **FP** | **TN** | **TN** | **TN** | **TN** | **TN** | **TN** |
|  | **TP** | **TP** | **TP** | **TP** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** |
|  | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** |
|  | **TP** | **TP** | **TP** | **TP** | **TP** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** |
|  | **TP** | **TP** | **TP** | **TP** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** |
|  | **TP** | **TP** | **TP** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** | **FN** |
|  | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** | **TP** |
| **TruePositiveRate = TP/6** | **1** | **1** | **1** | **0,833** | **0,5** | **0,333** | **0,333** | **0,333** | **0,333** | **0,333** | **1** |
| **FalsePositiveRate = FP/4** | **1** | **1** | **0,75** | **0,5** | **0,5** | **0,25** | **0,25** | **0,25** | **0** | **0** | **1** |

**R:**

Para o calculo desta matriz calculámos a P (C=1 | y1, y2, y3, y4), para isto tivemos que calcular o denominador da probabilidade definida em 1).

Dado que y1, y2 e y3 e y4 são independentes e que y1, y3 e y4 tem distribuição normal, foi necessário calcular os parametros da distribuição normal de y1, y2 e y3:

-P (y1 = a1) = N (a1| media = 0,18(9), desvio padrão = 0,23309)

-P (y2 = a2, y3 = a3) = N (a2, a3 | medias = (0,15; 0,15), desvios padrões = (e00 = 0,122(7); e01 = e10 = 0,131(6); e11 = 0,209(4))

Podemos verificar que na coluna de threshold = 0,3, tem-se uma false positive rate de 0,5 e uma true positive rate de 0,833, daí a threshold = 0,3 ser a que mais otimiza a solução caso queira aumentar a TPR sem aumentar muito a FPR, olahando para as colunas 0,8 e 0,9 tem-se uma false positive rate de 0 e uma true positive rate de 0,333, o que se torna melhor opção caso se queira

1. **Programming and critical analysis**

**6)**

Ao calcular a accuracy do knn com a distância euclidiana usando um 10-fold cross validation obtemos as seguintes accuracies (para estas accuracies, usamos a formula :

K = 3

- Test data accuracy: 0,96922

- Train data accuracy: 0,99678

- |Train data accuracy – test data accuracy| = 0,02756

K = 5

- Test data accuracy: 0,97217

- Train data accuracy: 0,97511

- |Train data accuracy – test data accuracy| = 0,00294

K = 7

- Test data accuracy: 0,97515

- Train data accuracy: 0,97218

- |Train data accuracy – test data accuracy| = 0,00297

Visto que para k = 5 o valor(|Train data accuracy – test data accuracy|) é inferior ao valor para k=3 e k = 7, concluímos que a solução que apresenta menor risco de overfit é a de k= 5.

**7)**

Para k = 3, calculámos a accuracy com método naïve Bayes, usando de novo um 10-fold cross validation. Para isto calculámos as probabilidades condicionadas (P(y1 = 1|benign), P(y1 = 1 | malignant), P(y1 = 2 | benign), … , P(y9 = 10 | benign), P(y9 = 10 | malignant). Estes cálculos foram efectuados na função “calcProbTrain”.

Obtemos uma mean accuracy com o k = 3 de:

-0,96931

Através das duas listas de accuracies (knn e naïve bayes) testamos a hipotese nula: knn tem maior accuracy que naïve bayes.

Ao fazer este teste, obtemos um valor-p = 0,4957, daí concluirmos que não se pode rejeitar esta hipotese para nivéis de significancia usuais.

**8)**

**III. APPENDIX**

Paste your programming code here using Consolas 9pt or 10pt.

Use **highlighting** or colored text to facilitate the analysis by your faculty hosts.

# Grupo 117 Aprendizagem HomeWork 1

# Bernardo Castico ist196845

# Hugo Rita ist196870

import math

import numpy as np

import statistics

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

from sklearn.model\_selection import KFold

from operator import itemgetter

from scipy import stats

from matplotlib.patches import Rectangle

import matplotlib.pyplot as plt

Res = KFold(n\_splits=10, random\_state=117, shuffle=True)

#Atributes

atributes = ["Clump Thickness", "Cell Size Uniformity", "Cell Shape Uniformity", "Marginal Adhesion", "Single Epi Cell Size",

"Bare Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses"]

#Get data

def getVectors(vector, index):

    res = []

    for i in vector:

        res.append(i[index])

    return res

def divideData(data):

    benign = []

    malignant = []

    for i in data:

        if i[-1] == 1:

            benign += [i]

        else:

            malignant += [i]

    return [benign, malignant]

def getDataToMatrix(lines):

    realLines = []

    data = []

    toDelete = []

    for i in range(len(lines)):

        if i > 11:

            realLines += [lines[i]]

    for i in range(len(realLines)):

        for j in range(len(realLines[i])):

            if realLines[i][j] == "benign\n":

                realLines[i][j] = 1

            elif realLines[i][j] == "malignant\n":

                realLines[i][j] = 0

            elif realLines[i][j] == '?':

                toDelete += [i]

            else:

                realLines[i][j] = int(realLines[i][j])

    for i in range(len(realLines)):

        if i not in toDelete:

            data += [realLines[i]]

    return data

#Exercise 6

def knn(data, i, train, k, numberWrongs, numberRights):

    lowerIndexes = []

    kLower = []  #Stores the k lower euclidian distances in comparison to data[i]

    for j in train:

        distance = euclidianDistance(data[i], data[j])

        if len(kLower) < k:

            kLower += [[distance, j]]

        else:

            higher = 0            #Assume that the first element is the higher

            for q in range(1,k):

                if kLower[q][0] > kLower[higher][0]:

                    higher = q

            if kLower[higher][0] > distance:

                kLower[higher] = [distance,j]

    for q in range(len(kLower)):

        lowerIndexes += [data[kLower[q][1]][-1]]

    mode = statistics.mode(lowerIndexes)               #Calculates the mode using

    if mode == data[i][-1]:

        numberRights += 1

    else:

        numberWrongs += 1

    return [numberRights, numberWrongs]

def knn2(data, i, k, numberWrongs, numberRights):

    kLower = []

    lowerIndexes = []

    for j in range(len(data)):

        if j != i:

            distance = euclidianDistance(data[i], data[j])

            if len(kLower) < k:

                kLower += [[distance, j]]

            else:

                higher = 0

                for q in range(1, k):

                    if kLower[q][0] > kLower[higher][0]:

                        higher = q

                if kLower[higher][0] >= distance:

                    kLower[higher] = [distance, j]

    for q in range(len(kLower)):

        lowerIndexes += [data[kLower[q][1]][-1]]

    mode = statistics.mode(lowerIndexes)

    if mode == data[i][-1]:

        numberRights += 1

    else:

        numberWrongs += 1

    return[numberRights, numberWrongs]

def kFold1(data, k):

    results = [0,0]

    resultsAux1 = [0,0]

    accuracies1 = []

    resultsAux2 = [0,0]

    accuracies2 = []

    for train, test in Res.split(data):

        for i in test:

            resultsAux = knn(data, i, train, k, 0, 0)

            results[0] += resultsAux[0]

            results[1] += resultsAux[1]

            resultsAux1[0] += resultsAux[0]

            resultsAux1[1] += resultsAux[1]

        accuracies1 += [resultsAux1[0]/(resultsAux1[1]+resultsAux1[0])]

        resultsAux1[0] = 0

        resultsAux1[1] = 0

        for i in train:

            resultsAux = knn2(data, i, k, 0, 0)

            results[0] += resultsAux[0]

            results[1] += resultsAux[1]

            resultsAux2[0] += resultsAux[0]

            resultsAux2[1] += resultsAux[1]

        accuracies2 += [resultsAux2[0] / (resultsAux2[1] + resultsAux2[0])]

        resultsAux2[0] = 0

        resultsAux2[1] = 0

    return [accuracies1, accuracies2]

def euclidianDistance(ponto1, ponto2):

    distance = 0

    for i in range(0, 9):

        distance += (ponto1[i] - ponto2[i]) \*\* 2

    distance = math.sqrt(distance)

    return distance

# EXERCISE 7

def calcProbTrain(train, data):  #This function stores the probabilities from the data set in a matrix, where each line refers to a column

    probabilities = []

    malignantCounter = 0

    benignCounter = 0

    for k in range(0,9):

        probabilities += [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]

        for j in range(1,11):

            for i in train:

                if data[i][-1] == 0 and data[i][k] == j:  #P(j|Malignant)

                    probabilities[k][2\*(j-1)] += 1

                elif data[i][-1] == 1 and data[i][k] == j: #P(j|Benign)

                    probabilities[k][2\*(j-1)+1] += 1

    for i in train:

        if data[i][-1] == 0:

            malignantCounter += 1

        else:

            benignCounter += 1

    for i in range(len(probabilities)):

        for k in range(len(probabilities[i])):

            if k % 2 == 0:

                probabilities[i][k] = probabilities[i][k] / malignantCounter

            else:

                probabilities[i][k] = probabilities[i][k] / benignCounter

    return probabilities

def calculateBayesian(probabilities, index, data, train):

    n = len(train)

    probBenign = 0

    probMalignant = 0

    for i in train:

        if data[i][-1] == 0:

            probMalignant += 1

        else:

            probBenign += 1

    probMalignant = probMalignant/n

    probBenign = probBenign/n

    #Bayesian probability for class = malignant

    valueMalignant = probMalignant

    for k in range(9):

        valueMalignant \*= probabilities[k][2 \* data[index][k]-2]

    #Bayesian probability for class = benign

    valueBenign = probBenign

    for k in range(9):

        valueBenign \*= probabilities[k][2 \* data[index][k]-1]

    if valueBenign > valueMalignant and data[index][-1] == 1:

        return [1,0]

    elif valueMalignant > valueBenign and data[index][-1] == 0:

        return [1,0]

    else:

        return [0,1]

def kFoldEx7(data):

    result = [0,0]

    accuracies3 = []

    resultsAux3 = [0,0]

    for train, test in Res.split(data):

        probabilities = calcProbTrain(train, data)

        for i in test:

            resultAux = calculateBayesian(probabilities, i, data, train)

            result[0] += resultAux[0]

            result[1] += resultAux[1]

            resultsAux3[0] += resultAux[0]

            resultsAux3[1] += resultAux[1]

        accuracies3 += [resultsAux3[0]/(resultsAux3[1]+resultsAux3[0])]

        resultsAux3[0] = 0

        resultsAux3[1] = 0

    return accuracies3

# MAIN FUNCTION

def main():

    res = []

    accuracy1 = 0

    accuracy2 = 0

    accuracy3 = 0

    #GET DATA

    k = eval(input("k: "))

    with open("HW1.txt") as f:

        lines = f.readlines()

    for line in lines:

        tmp = line.split(',')

        res.append(tmp)

    data = getDataToMatrix(res)

    # EXERCISE 5

    benign, malignant = divideData(data)

    fig, \_ = plt.subplots(nrows=3, ncols=3, figsize=(10, 8))  # creating the histograms

    fig.tight\_layout(pad=4.0)

    axes = fig.axes #list with the axes

    fig.canvas.set\_window\_title('AP HW01 G132')  # Title of histogram

    colors = ['blue', 'red']

    # Titles of histograms

    for i in range(len(axes)):

        vectorsBenign = getVectors(benign, i)

        vectorsMalignant = getVectors(malignant, i)

        axes[i].title.set\_text(atributes[i])

        axes[i].set\_xlabel("Value")

        axes[i].legend(prop={'size': 9})

        axes[i].set\_ylabel("Counter")

        axes[i].hist([vectorsBenign, vectorsMalignant], 10, density=False, histtype='bar', color=colors,

                    label=["Benign", "Malignant"])

    plt.show()

    # EXERCISE 6

    results1 = kFold1(data, k)

    for i in results1[0]:

        accuracy1 += i

    accuracy1 /= 10

    for i in results1[1]:

        accuracy2 += i

    accuracy2 /= 10

    print("accuracy test: " + str(accuracy1))

    print("accuracy train: " + str(accuracy2))

    # EXERCISE 7

    results3 = kFoldEx7(data)

    for i in results3:

        accuracy3 += i

    accuracy3 /= 10

    print("accuracy bayes: " + str(accuracy3))

    pValue = stats.ttest\_ind(np.array(results1[0]), np.array(results3), alternative="less")

    print(pValue)

main()

**END**