Projeto 2 - Códigos

1 Método de classificação pelos k-vizinhos

Elaboramos uma rotina responsável pela leitura dos arquivos utilizando a biblioteca *pandas*. Abaixo, podemos encontrar as bibliotecas utilizadas nesse projeto.

```
1
2     from matplotlib.colors import Normalize
3     import matplotlib.pyplot as plt
4     import numpy as np
5     import pandas as pd
6     import itertools
7     from math import sqrt
8     from pathlib import Path
9
10
```

Listing 1: Importação das bibliotecas

Rotina para leitura dos arquivos com a biblioteca *pandas*:

```
1
2     colNames = ["X0", "X1", "X2", "X3", "Category"]
3     dataFrame = pd.read_csv("/home/eu/AnaliseEReconhecimento/Projeto2/
    irisDataset/iris.data", header=None, names=colNames)
4     groups = dataFrame.groupby("Category")
5
```

Listing 2: Leitura de arquivos

O segundo passo é a criação dos gráficos com *scatterplot*. Para isso, necessitamos de um vetor que contenha todos as configurações 2 a 2 dos parâmetros de nosso arquivo. Tal variação de parâmetros é conseguida a partir do seguinte:

```
1
2     permutNames = ["0", "1", "2", "3"]
3     product = pd.DataFrame(list(itertools.product(permutNames, repeat=2)))
4     p = product.to_numpy()
5     p = p.astype(int)
6
7
```

Listing 3: Produto te dos parâmetros de nossa base

Com tal vetor em mãos, implementamos uma função que mostra todos os *scatterplots*.

```
def plotProduct(df, permutNames, groups):
    '''
    This function is used to plot the product of all data present in our dataset

    - args -

    df: a pandas dataframe that has all our data
    permutNames: list with the names of each axes
    groups: pandas dataframe grouped by a certain column ("category" in our case)

    - return -

    none
    '''

    ###Creating the product (2x2) of all names to be plotted
    product = pd.DataFrame(list(itertools.product(permutNames, repeat=2)))

    ###Creating pyplot figure with 16 graphs
    fig, axs = plt.subplots(4,4, figsize=(30, 30))
    fig.subplots_adjust(hspace = 0.30, wspace=0.15)

    axs = axs.ravel()

    ###Using the product dataframe, plotting the graphs with all data
    for i in range(len(product)):
        one = "X"+str(product.loc[i][0])
        two = "X"+str(product.loc[i][1])
        for name, group in groups:
            axs[i].plot(group[one], group[two], marker="o", linestyle="", label=name)
            axs[i].set_xlabel(one)
            axs[i].set_ylabel(two)
            axs[i].legend()#loc="upper right"
    plt.show()
```

Listing 4: Geração de gráficos da variação dos parâmetros 2 a 2

```
data = dataFrame.to_numpy()
```

Listing 5: Transformação de nossos dados para um vetor do numpy

Por fim, para a implementação do métodos dos k-vizinhos, construímos duas outras funções

auxiliares para a divisão dos objetos do arquivo das flores em um vetor de testes e outro de treinamento e uma função que calcula a distância euclidiana entre dois pontos.

```
1
2     def split(data):
3         train =[]
4         test =[]
5
6         for i in range(len(data)):
7             if (i%2==0):
8                 test.append(data[i])
9             else:
10                train.append(data[i])
11
12        return([train, test])
13
14
```

Listing 6: Divisão de dados

```
1
2     def calcDistance(x0, y0, x1, y1):
3         return sqrt((x0-x1)**2+(y0-y1)**2)
4
5
```

Listing 7: Distância euclidiana

Para o método dos k-vizinhos, implementamos a seguinte função:

```
1
2     def k_neighbors(data, indexes):
3     ##using odd indexes as train and even as test
4
5     newData = split(data)
6     train, test = newData[0], newData[1]
7     confusionMatrix = []
8
9     ##For combinations of parameters
10    for i in indexes:
11        dist = []
12        guessList = []
13        ##Chose one from the test list
14        for j in range(len(test)):
15
16            ##calculate all the distances between the chosen one from the
    test list to all from the train list
17            for m in range(len(train)):
```

```
                #distance = sqrt((train[j][i[0]]-test[m][i[0]])**2+(train[j
    ][i[1]]-test[m][i[1]])**2)
                distance = calcDistance(train[j][i[0]],  train[j][i[1]],
    test[m][i[0]], test[m][i[1]])
                dist.append([distance, j, m, i])

            ##find the smallest distance
            sortedDist = sorting(dist, 0)

            ##true category |  guessed one
            guess = [test[j][4], train[sortedDist[0][2]][4]]

            if(guess[0] == guess[1]):
                guessList.append([guess, "True"])
            else:
                guessList.append([guess, "False"])



        mat = np.matrix([[0, 0, 0], [0, 0, 0], [0, 0, 0]])

        for q in range(len(guessList)):


            if(guessList[q][0][0] == "Iris-setosa"):

                if (guessList[q][0][1] == "Iris-setosa"):
                    mat[0,0]+=1

                if (guessList[q][0][1] == "Iris-versicolor"):
                    mat[1,0]+=1

                if (guessList[q][0][1] == "Iris-virginica"):
                    mat[2,0]+=1

            if(guessList[q][0][0] == "Iris-versicolor"):

                if (guessList[q][0][1] == "Iris-setosa"):
                    mat[0,1]+=1

                if (guessList[q][0][1] == "Iris-versicolor"):
                    mat[1,1]+=1

                if (guessList[q][0][1] == "Iris-virginica"):
                    mat[2,1]+=1

            if(guessList[q][0][0] == "Iris-virginica"):

```

```
                if (guessList[q][0][1] == "Iris-setosa"):
                    mat[0,2]+=1

                if (guessList[q][0][1] == "Iris-versicolor"):
                    mat[1,2]+=1

                if (guessList[q][0][1] == "Iris-virginica"):
                    mat[2,2]+=1


        # norm = np.linalg.norm(mat)
        mat = mat/25
        confusionMatrix.append([mat, i])


    plotMatrix(confusionMatrix)

```

Listing 8: Método dos k-vizinhos

Para criação das matrizes de confusão, criamos a função *plotMatrix()* chamada ao fim da função dos k-vizinhos.

```

    def plotMatrix(confMatInt):
        confmat = "/home/eu/AnaliseEReconhecimento/Projeto2/" + "confMats"
        Path(confmat).mkdir(parents=True, exist_ok=True)

        fig = plt.figure()#figsize=(1920/96, 1080/96), dpi=96
        #ax = fig.add_subplot(111)
        names = ["setosa", "versicolor", "virginica"]

        for i in range(len(confMatInt)):

            name = str(confMatInt[i][1])
            figName = confmat + "/" + name + ".png"

            plt.xlabel("Verdadeiro")
            plt.ylabel("Classificado")
            plt.title("Parametros " + name)
            a = plt.imshow(confMatInt[i][0], vmin=0, vmax=1)
            fig.colorbar(a)
            plt.xticks(ticks=[0, 1, 2], labels=names)
            plt.yticks(ticks=[0, 1, 2], labels=names)
            for (j, k), z in np.ndenumerate(confMatInt[i][0]):
                plt.text(k, j, '{:0.1f}'.format(z), ha='center', va='center'
    ,
                bbox=dict(boxstyle='round', facecolor='white', edgecolor='
    0.3'))
```

```
25              plt.savefig(figName)
26              plt.clf()
27
28
```

Listing 9: Geração das matrizes de confusão