Projeto 1 - Códigos

1 Parte A

Para a elaboração desse projeto, criamos uma classe chamada *PatterRec*. Todas as operações foram feitas a partir dos métodos implementados nessa classe. A seguir, acompanhamos os códigos referentes a todos os passos, incluindo as bibliotecas importadas.

```

    from matplotlib import markers
    import numpy as np
    import matplotlib.pyplot as plt
    import os
    from pathlib import Path ##To create a new folder

    from random import random, randrange
    from random import gauss

    import cv2


```

Listing 1: Importação das bibliotecas

1.1 Leitura e extração de dados

Funções para e Leitura dos arquivos:

```

    def initTextReader(self, path, ext):

    self.path = path
    self.ext = ext
    self.searchDirectory(path, ext)
    self.sortFiles()

  def searchDirectory(self, path, ext):

    self.filesList = []

    for file in os.listdir(path):
      if file.endswith(ext):
        self.filesList.append(os.path.join(path, file))

  def sortFiles(self):
    number = np.empty(len(self.filesList))
```

```
19
20      for i in range(len(self.filesList)):
21        name = self.filesList[i].split("/")[-1].strip(self.ext)
22        index = i
23        number[i] = int(name.split(" ")[-1])
24
25      sorted = np.argsort(number)
26      filesAux = []
27      for i in sorted:
28        filesAux.append(self.filesList[i])
29
30      self.filesList = filesAux
```

Listing 2: Leitura de arquivos

Função para extração dos dados de cada um dos arquivos.

```
1
2       def extractData(self):
3       ##Intended to be used with files that have
4       ##"x" values on the first column and "y" values
5       ##on the second column.
6
7       self.x = []
8       self.y = []
9
10      for file in self.filesList:
11        new = open(file, "r")
12        data = new.readlines()
13        data.pop(0) ##data 0 is 'x'
14
15        xAux = []
16        yAux = []
17
18        for i in range(len(data)):
19          data[i].replace("\n", "")
20
21          a, b = data[i].split(' ')
22          a = a.strip('"')
23          a = float(a)
24          b = float(b)
25          xAux.append(a)
26          yAux.append(b)
27        self.x.append(xAux)
28        self.y.append(yAux)
29
30
```

Listing 3: Extração de dados

1.2 Geração de gráficos em barras e histogramas

Funções para criação dos gráficos em barras conjuntos e separados respectivamente.

```
1
2       def plotBarGraphs(self):
3
4     fig, axs = plt.subplots(5,4, figsize=(50, 50), facecolor='white',
    edgecolor='k')
5     fig.subplots_adjust(hspace = 1, wspace=0.5)
6     axs = axs.ravel()
7
8     for i in range(len(self.x)):
9       name = self.filesList[i].split("/")[-1].strip(self.ext).replace("_", "
    ")
10      axs[i].bar(self.x[i], self.y[i])#, color='blue'
11      axs[i].set_xlabel("Indice Medida")
12      axs[i].set_ylabel("Valor Aferido")
13      axs[i].set_title(name)
14    plt.show()
15
```

Listing 4: Geração de gráficos de barra conjuntos

```
1
2     def generateBarGraphs(self):
3     ###Creating a new folder to put the barGraphs
4     barGraphsPath = self.path + "BarGraphs"
5     Path(barGraphsPath).mkdir(parents=True, exist_ok=True)
6
7     plt.figure(figsize=(1920/96, 1080/96), dpi=96)
8
9     for i in range(len(self.x)):
10
11      name = self.filesList[i].split("/")[-1].strip(self.ext)
12      figName = barGraphsPath + "/" + name + ".png"
13
14
15      plt.xlabel("Observacao")
16      plt.ylabel("Medida")
17      plt.title("Arquivo " + name)
18      plt.bar(self.x[i], self.y[i], color='blue')
19      plt.savefig(figName)
20      plt.clf()
21
```

Listing 5: Geração de gráficos em barras isolados

Funções para criação dos histogramas conjuntos e separados respectivamente.

```
1
2    def plotHistograms(self):
3
4    fig, axs = plt.subplots(5,4, figsize=(50, 50), facecolor='white',
     edgecolor='k')
5    fig.subplots_adjust(hspace = 1, wspace=0.5)
6    axs = axs.ravel()
7
8    for i in range(len(self.x)):
9      name = self.filesList[i].split("/")[-1].strip(self.ext).replace("_", "
     ")
10     axs[i].hist(self.y[i])#, color='blue'
11     axs[i].set_xlabel("Observacao")
12     axs[i].set_ylabel("Valor")
13     axs[i].set_title(name)
14   plt.show()
15
16
```

Listing 6: Geração de histogramas conjuntos

```
1
2    def generateHistograms(self):
3
4    histograms = self.path + "Histograms"
5    Path(histograms).mkdir(parents=True, exist_ok=True)
6
7    plt.figure(figsize=(1920/96, 1080/96), dpi=96)
8
9    for i in range(len(self.x)):
10
11     name = self.filesList[i].split("/")[-1].strip(self.ext)
12     figName = histograms + "/" + name + ".png"
13
14     plt.xlabel("Observacao")
15     plt.ylabel("Valor")
16     plt.title("Arquivo " + name)
17     plt.hist(self.y[i])
18     plt.savefig(figName)
19     plt.clf()
20
21
```

Listing 7: Geração de histogramas isolados

1.3 Estatística

Funções extração de dados estatísticos, modelagem e visualização.

```
  def meanAndStd(self):
  self.yMean = [ ]
  self.yStd = [ ]
  for i in range(len(self.x)):
    self.yMean.append(np.mean(self.y[i]))
    self.yStd.append(np.std(self.y[i]))

  s1 = 0
  s2 = 0
  d1 = 0
  d2 = 0
  for i in range(len(self.x)):
    if (i <= 10):
      s1 += self.yMean[i]
      d1 += self.yStd[i]
    else:
      s2 += self.yMean[i]
      d2 += self.yStd[i]
  #Mean of each subset
  self.mean1 = s1/10
  self.mean2 = s2/10
  self.std1 = d1/10
  self.std2 = d2/10

```

Listing 8: Medidas de média e desvio padrão

```
  def generateUniformData(self, l=-0.5, h=0.5, r=500, n=10):
  self.uniformDistrib = []
  for i in range(n):
    self.uniformDistrib.append(np.random.uniform(low=l, high=h, size=(1,r)
 )[0])

def generateGaussianData(self, m=0, d=1, r=500, n=10):
  self.gaussianDistrib = []

  for i in range(n):
    mat =  np.zeros(r)

    for i in range(r):
      value = gauss(m, d)
      mat[i] = value
    self.gaussianDistrib.append(mat)
  print(self.gaussianDistrib)


```

Listing 9: Geradores de dados sugeridos no projeto

Função para escrever os dados gerados seguindo o mesmo padrão dos dados fornecidos, para que um novo objeto da classe possa ser inicializado com os novos dados.

```
1
2     def writeGeneratedData(self):
3     path = "/home/eu/AnaliseEReconhecimento/Aula1/DadosGerados/"
4     Path(path).mkdir(parents=True, exist_ok=True)
5     for i in range(len(self.uniformDistrib)):
6       file = open(path+"gen_ "+str(i)+".txt", "w")
7       file.write('"'+"x"+'"'+"\n")
8       for j in range(len(self.uniformDistrib[i])):
9         file.write('"'+str(j)+'"' + " " + str(self.uniformDistrib[i][j])+ "\
    n")
10      file.close()
11
12    for i in range(len(self.gaussianDistrib)):
13      file = open(path+"gen_ "+str(i + 10)+".txt", "w")
14      file.write('"'+"x"+'"'+"\n")
15      for j in range(len(self.gaussianDistrib[i])):
16        file.write('"'+str(j)+'"' + " " + str(self.gaussianDistrib[i][j])+ "
    \n")
17      file.close()
18
19
```

Listing 10: Escrevendo os dados gerados

Plot das medidas estatísticas.

```
1
2       def plotMeanStd(self):
3
4     self.xGeneral = []
5     for i in range(1, len(self.x)+1):
6       self.xGeneral.append(i)
7
8
9     fig, ax1 = plt.subplots(figsize=(1920/96, 1080/96), dpi=96)
10
11    ax2 = ax1.twinx()
12    ax1.scatter(self.xGeneral, self.yMean, color='green')
13    ax2.scatter(self.xGeneral, self.yStd, color ='blue')
14
15    ax1.set_xlabel('Arquivos')
16    ax1.set_ylabel('Media', color='g')
17    ax2.set_ylabel('Desvio', color='b')
18    ax1.set_xticks(np.arange(min(self.xGeneral), max(self.xGeneral)+1, 1.0))
19    ax1.set_yticks(np.arange(-1, 1.5, 0.5))
```

```
    ax2.set_xticks(np.arange(min(self.xGeneral), max(self.xGeneral)+1, 1.0))
    ax1.axvline(10.5, color="orange")

    plt.show()


```

Listing 11: Plotando as medidas estatísticas realizadas

1.4 Imagens

Geração de imagens a partir das distribuições uniforme e gaussiana em 2 dimensões

```

def generateImage(self):
    mat =  np.zeros((200, 200))
    for i in range(200):
      for j in range(200):
        value = gauss(0, 1)
        mat[i][j] = value
    cv2.imshow("randGauss", mat)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    sampl = np.random.uniform(low=-0.5, high=0.5, size=(200,200))
    cv2.imshow("uniformRand", sampl)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

```

Listing 12: Geração das imagens

2 Parte B

Nessa parte do projeto, continuamos com a classe elaborada na parte A e adicionamos métodos específicos requeridos por essa etapa.

2.1 Gerador de autômatos

Implementamos um gerador de autômatos genéricos que recebe a matriz estocástica, o número de passos e se o autômato apresenta valores binários ou não e nos retorna um vetor com os respectivos valores do passeio.

```

    def genericAutomato(self, n, mat, bin=0): #mat must be a numpy mXm array
    automata = np.zeros(n)
```

```
    automata[0] = 0 # we must start at the node zero
    shape = mat.shape

    ##Is it a squared matrix?
    if (shape[0] == shape[1]):
      nodes = shape[0]
    else:
      print(shape[0])
      print(shape[1])
      print("Not a squared matrix")
      return 1


    if (bin==0):
      ##Building the signal based on the stochastic matrix
      for i in range(1, n): #n steps on the automato
        value = np.random.rand(1)
        current = int(automata[i-1])
        probs = mat[:, current]

        sumPrb = 0

        for j in range(nodes):

          sumPrb = sumPrb + probs[j]

          if(sumPrb >= value):

            automata[i] = j

            break

    else:
      for i in range(1, n): #binary automato
        value = np.random.rand(1)
        current = int(automata[i-1])
        probs = mat[:, current]

        sumPrb = 0

        for j in range(nodes):

          sumPrb = sumPrb + probs[j]

          if(sumPrb >= value):

            if (j%2 == 0):
              automata[i] = 0
```

```
52              else:
53                automata[i] = 1
54
55              break
56      return automata
57
```

Listing 13: Gerador de autômatos genéricos

2.2 Apresentação em uma dimensão do autômato gerado

Para a apresentação visual do autômato gerado, utilizamos a biblioteca *openCV* para criação de figuras a partir do valor do autômato em um certo passo.

```
1
2       def visualizeAutomata(self, automata):
3       mat = np.zeros((100, len(automata)))
4       zeros = 0
5       ones = 0
6       for i in range(len(automata)):
7         if(automata[i] == 0):
8           mat[:,i] = 0
9           zeros += 1
10        else:
11          mat[:,i] = 1
12          ones += 1
13
14      cv2.imshow("automata", mat)
15      cv2.waitKey(0)
16      cv2.destroyAllWindows()
17
```

Listing 14: Visualização em uma dimensão do automato gerado

Para os autômatos que não apresentam resultado binário, implementamos uma função que apresenta os resultados em um *stem plot*.

```
1
2       def plotStem(self, automata):
3       xAux = []
4       for i in range(len(automata)):
5         xAux.append(i)
6
7       plt.stem(xAux, automata)
8       plt.xlabel("passo")
9       plt.ylabel("valor")
10      plt.show()
```

```
11
12
```

Listing 15: Stem plot de autômatos com valores não binários

2.3 Apresentação em duas dimensões do autômato gerado

Para visualização em 2D, criamos a função que mostra as três diferentes apresentações do autômato que foram implementada.

```
1
2     def visualize2DAutomata(self, automata):
3     mat = np.zeros((len(automata), len(automata)))
4     zeros = 0
5     ones = 0
6     for i in range(len(automata)):
7       if(automata[i] == 0):
8         mat[:,i] = 0
9         mat[i,:] = 0
10
11      else:
12        mat[:,i] = 1
13        mat[i,:] = 1
14
15
16    cv2.imshow("automata1", mat)
17
18    mat2 = np.zeros((len(automata), len(automata)))
19    zeros = 0
20    ones = 0
21    for i in range(len(automata)):
22      if(automata[i] == 0):
23        # mat[:,i] = 0
24        # mat[i,:] = 0
25        pass
26      else:
27        mat2[:,i] = 1
28        mat2[i,:] = 1
29
30
31    cv2.imshow("automata2", mat2)
32    cv2.waitKey(0)
33    cv2.destroyAllWindows()
34
35
```

Listing 16: Visualização em duas dimensões

2.4 Densidade de 1's

Para o cálculo e visualização da densidade de 1's a partir da frequência relativa de diferentes número de passos, temos as seguintes funções:

```
    def relFreq(self, automata, target):
    c = 0
    for i in range(len(automata)):
      if (automata[i] == target):
        c+=1
    return c/len(automata)
```

Listing 17: Cálculo da frequência relativa

```
    def density(self, a, b, c):

    self.freqa = []
    self.freqb = []
    self.freqc = []

    for i in (200, 500, 750, 1000, 2000):
      autoa = self.genericAutomato(i, a)
      autob = self.genericAutomato(i, b)
      autoc = self.genericAutomato(i, c)
      self.freqa.append(self.relFreq(autoa, 1))
      self.freqb.append(self.relFreq(autob, 1))
      self.freqc.append(self.relFreq(autoc, 1))

      self.plotRelFreq()
    plt.show()


```

Listing 18: Densidade de 1's

```
    def plotRelFreq(self):

    fig = plt.gcf()
      #fig.suptitle("f")


    sns.distplot(self.freqa, rug=False, hist=False, color="black")
    sns.distplot(self.freqb, rug=False, hist=False, color="yellow")
    sns.distplot(self.freqc, rug=False, hist=False, color="red")


```

```
13      plt.xlim(0,1)
14      plt.xlabel("F")
15      plt.ylabel("Densidade")
16
17
```

Listing 19: Plot das densidades calculadas