Identificação automática de linhas em gráficos com algorítmos de Machine Learning

Antonio Rodrigues Neto

Agosto, 2021

Este projeto visa construir um algoritmo na linguagem python para identificar automaticamente os dados plotados como linha em gráficos obtidos de imagens no formato PNG. Para isso, é utilizada a técnica não supervisionada de Machine Lenarning denomiada DBSCAN. Esta técnica tem como objetivo o agrupamento de observações da amostra de treino com base em densidade. Assim, objetiva-se que cada série do gráfico contido na imagem seja obtida como um grupo do DBSCAN.

1 Input data

A entrada de dados no código deve realizar basicamente três tarefas:

- Importar a imagem do gráfico em PNG. Para isso, é utilizada função *mpimg* da biblioteca *matplotlib*, a qual vai obter em formato numérico os dados de cores de cada pixel da imagem PNG.
- Extrair as informações da imagem e agrupá-las em um dataset. A extração das informações requer o entedimento do posicionamento dos dados de cada pixel (guardando posição cartesiana xy de cada um) dentro do objeto *img*. Além disso, é necessário inverter os dados das posições y, visto que a leitura da imagem se dá de cima para baixo;
- Setar os parâmetros do DBSCAN. Ambos os parametros *eps* (raio) e *p-min* (número mínimo de pontos de um cluster) são setados após avaliações de tentativa e erro.

Segue abaixo o código que realiza as tarefas do Input data, bem como a importação das bibliotecas e funções necessárias:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
```

```
3     import matplotlib.image as mpimg
4     import pandas as pd
5     from sklearn.cluster import DBSCAN
6     from sklearn.preprocessing import scale
7
8
9     img=mpimg.imread('t2.PNG')
10
11    eps=0.15
12    pmin=25
13
14    # Obtendo dados da imagem
15
16    vec_x=[]
17    vec_y=[]
18    vec_r=[]
19    vec_g=[]
20    vec_b=[]
21    vec_alpha=[]
22
23    for i in range(len(img)):
24        for j in range(len(img[i])):
25            a=img[i,j]
26            grava=True
27            if (a[0]==1.0 and a[1]==1.0 and a[2]==1.0):
28                grava=False
29            if (grava):
30                vec_x.append(float(j))
31                vec_y.append(float(i))
32                vec_r.append(a[0])
33                vec_g.append(a[1])
34                vec_b.append(a[2])
35                vec_alpha.append(a[3])
36
37    ## Invertendo valores de y
38
39    y_max=float(len(img))
40    print(vec_y[5],vec_y[500],vec_y[1000])
41
42    for i in range(len(vec_y)):
43      vec_y[i] = y_max - vec_y[i]
```

```
44
45      # Colocando dados em data_frame
46      data = {'R':vec_r,'G':vec_g, 'B':vec_b,'alpha':vec_alpha,'x':vec_x,'y':vec_y}
47      df = pd.DataFrame (data, columns = ['R','G','B','alpha','x','y'])
```

2 Processamento do DBSCAN

O processamento do algoritmo DBSCAN é realizado com base em seis variáveis: componente R da cor, componente G da cor, componente B da cor, parâmetro α da cor, posição x e posição y. Todas já estão presentes no dataset *df* do *pandas*. Lembrando que é necessário realizar padronização com a função *scale*, visto que os limites das variáveis são diferentes, o que prejudica nos cálculos de distância e densidade.

Assim, pode-se aplicar a função *DBSCAN* da biblioteca *sklearn.cluster*. Os cluster obtidos são então plotados na tela, para que o usuário avalie visualmente os resultados obtidos.

Neste momento, a correta avaliação das séries requer a entrada manual de algumas informações pelo usuário, estão são:

- Identificação de quais clusters são as séries desejadas, pois os eixos x e y também são clusterizados e estes devem ser identificados pelo usuário no plot dos clusters.
- Os valores limite dos eixos x e y para que a transformação do espaço de pixels para o espaço real seja feita corretamente. Vale mencionar aqui que, por hipótese, o código assume que ambos os eixos iniciam no valor 0.

Com estas informações, o código deve obter quais pixels pertencem à cada uma da séries desejadas, bem como suas posições cartesianas. As posições devem ser transformadas para valores xy reais com base nos valores limites dos eixos informados pelo o usuário. Para isso, basta aplicar uma simples transformação linear de coordenadas.

Segue abaixo o código que realiza as tarefas de etapa de processamento:

```
1       ## Aplicando dbscan com x,y e cores
2       data1 = scale(df)
3       X = np.column_stack((data1[:,0],data1[:,1],data1[:,2],data1[:,3],data1[:,4],data1[:,5]))
4
5       clustering = DBSCAN(eps=eps, min_samples=pmin).fit(X)
6       labels = clustering.labels_
7
8       # Number of clusters in labels, ignoring noise if present.
9       n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
```

```
    n_noise_ = list(labels).count(-1)

    print("Number of obtained clusters:",n_clusters_)

    ## Fazendo dataframe com os labels
    data1 = {'R':vec_r,'G':vec_g, 'B':vec_b, 'alpha':vec_alpha, 'labels':labels ,'x':vec_x,'y':vec_y}
    df_cluster = pd.DataFrame (data1, columns = ['R','G','B','alpha','labels','x','y'])
    df_cluster1 = df_cluster[df_cluster['labels']!=-1]
    print(df_cluster1.head())
    print(df_cluster1.tail())

    ## Plotando grupos da clusterizacao
    groups = df_cluster1.groupby('labels')
    for name, group in groups:
        plt.plot(group['x'], group['y'], marker="o", linestyle="", label=name)
    plt.legend()
    plt.show()

    ## Getting the axis from user
    x,y = input("Digite qual o grupo contem os eixos x e y:").split()
    x = int(x)
    y = int(y)

    ## Getting series data from user
    ler = True
    series=[]
    while (ler):
      a=input("Digite qual grupo corresponde uma serie do grafico: (-1 para parar)")
      a=int(a)
      if (a==-1):
        ler = False
      else:
        series.append(a)

    ## Getting axis limits
    max_x,max_y = input("Digite valor maximo dos eixos x e y:").split()
    max_x=float(max_x)
    max_y=float(max_y)

    # Pegando eixos
```

```
    axis = df_cluster1[(df_cluster1['labels']==x) | (df_cluster1['labels']==y)]
    orig_x=min(axis.x)
    orig_y=min(axis.y)

    max_x_pixel=max(axis.x)
    max_y_pixel=max(axis.y)

    # Pegando Series
    s_graph=[]
    for i in range(len(series)):
      s_graph.append(df_cluster1[df_cluster1['labels']==series[i]])

    ## Calculando x_real e y_real
    for i in range(len(series)):
      s_graph[i]['x_real'] = (max_x/(max_x_pixel-orig_x))*(s_graph[i]['x']-orig_x)
      s_graph[i]['y_real'] = (max_y/(max_y_pixel-orig_y))*(s_graph[i]['y']-orig_y)
```

3 Pós-Processamento / Saída de dados

Assim, o código já tem todas as informações necessárias para realizar o pós-processamento dos dados e posterior Output data.

Nessa etapa, é setado o número de pontos (*n_points*) que será avaliado em cada uma das séries identificadas pelo usuário na etapa anterior. O intervalo em x de cada série é então dividido em um número determinado de intervalos $\Delta\bar{x}$ correspondente à *n_points*. Cada ponto $\bar{x}$ avaliado é setado como o valor central de cada intervalo $\Delta\bar{x}$. O valor $\bar{y}_i$ de cada ponto para cada série i é determinado pela média de todos os pontos da série que correspondam à $\bar{x}$.

Com estas informações, é construída uma string *output* que vai receber todas informações à serem escritas no arquivo TXT de saída de dados. A saida de dados consiste de uma matriz contendo os valores $\bar{x}$ e os valores $\bar{y}_i$ obtidos para cada um das séries i.

Segue abaixo o código que realiza as tarefas desta etapa:

```
    ##
    ##  CRIANDO SAIDA DE DADOS
    ##
    n_points=50

    # Obtendo valores de x_barra
    x_points = np.arange(0, max_x+max_x/n_points, max_x/(n_points))
```

```
    x_values=[]
    for i in range(n_points):
      x_values.append(x_points[i]+(x_points[i+1]-x_points[i])/2.0)

    # Obtendo valores de y_barra para cada serie
    y_values=[]
    for serie in range(len(series)):
      y_local=[]
      for i in range(n_points):
        dentro = s_graph[serie].x_real.between(x_points[i],x_points[i+1])
        y_local.append(np.mean(s_graph[serie][dentro].y_real))
      y_values.append(y_local)

    # Criando string de saida de dados
    output=""
    output+='x     '
    for i in range(len(series)):
      output+='    Serie_{}    '.format(series[i])
    output+='\n'

    for i in range(n_points):
      output+='{:.6f}   '.format(x_values[i])
      for j in range(len(series)):
        if (not np.isnan(y_values[j][i])):
          output+='{:12f}  '.format(y_values[j][i])
        else:
          output+='     -        '
      output+='\n'

    # Gravando saida de dados em um arquivo TXT
    saida = open("grafico.txt","w")
    saida.write(output)
    saida.close()


    print("End of code")
    pass
```

4 Exemplos de aplicação

4.1 Exemplo 1: Gráfico com série única

No primeiro exemplo é gerado um gráfico no Excel com uma série única que representa a função $f(x) = x^2 + 5$. A imagem importada pelo código é ilustrada na Fig. 1.

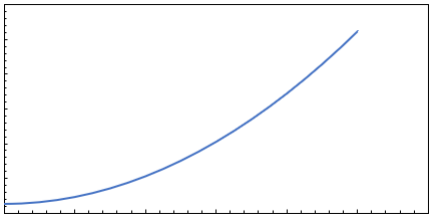

Figure 1: Imagem importada no Exemplo 1.

Os parâmetros utilizados na clusterização são $eps = 0.15$ e $p\_min = 25$. Os clusters obtidos pelo DBSCAN são ilustrados na Fig. 2. Vale citar que os valores máximos dos eixos x e y são, respectivamente, 15 e 120. Estes valores também devem ser informados pelo usuário.

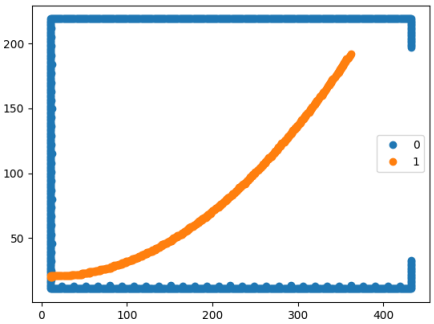

Figure 2: Cluster obtidos no Exemplo 1.

Assim, o usuário seta que o grupo 0 são os eixos xy e o grupo 1 é a série desejada. A Fig. 3 ilustra os resultados obtidos pela clusterização (pontos da série "Clusterizado") em comparação com o gráfico original (série "y_analtico").

Pode-se concluir a partir da Fig. 3 que os resultados obtidos pelo algorítmo foram excelentes em relação a referência. Todos os pontos calculados caíram exatamente em cima da série original, conforme o esperado.

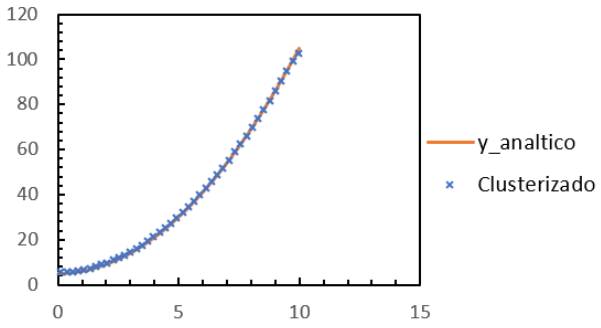

Figure 3: Resultados obtidos no Exemplo 1.

4.2 Exemplo 2: Gráfico com múltiplas séries

No segundo exemplo é gerado um gráfico no Excel com três diferentes séries que representam as seguintes funções:

- $f_1(x) = 100x + 10$, com $0 \leq x \leq 15$
- $f_2(x) = x^3$, com $0 \leq x \leq 15$
- $f_3(x) = 1000 \sin x + 500x$, com $1 \leq x \leq 15$

A imagem importada pelo código é ilustrada na Fig. 4.

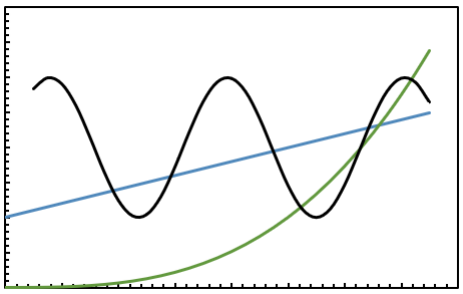

Figure 4: Imagem importada no Exemplo 2.

Os parâmetros utilizados na clusterização deste exemplo são os mesmos do primeiro exemplo ($eps = 0.15$ e $p\_min = 25$). Os clusters obtidos pelo DBSCAN são ilustrados na Fig. 5. Vale citar que os valores máximos dos eixos x e y foram informados manualmente pelo usuário.

Com base na Fig. 5, o usuário deve informar que os clusters 0 e 4 correspondem aos eixos xy, enquanto que os clusters 1, 2 e 3 são séries desejadas. Assim, os resultados obtidos pelo algoritmo são ilustrados na Fig. 6 através das séries "Cluster Serie_1", "Cluster Serie_2" e "Cluster Serie_3" no gráfico.

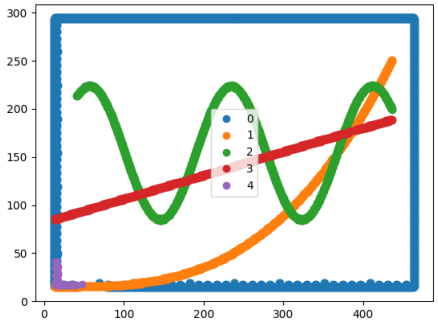

Figure 5: Clusters obtidos no Exemplo 2.

A Fig. 6 permite concluir que os resultados deste exemplo foram também satisfatório. Todos os pontos obtidos para as três séries coincidiram exatamente com as funções analíticas originais. Vale observar ainda que, mesmo existindo o cruzamento entre linhas de diferentes séries em diversos pontos, o algoritmo foi capaz de contornar essa dificuldade e o resultado não foi afetado.

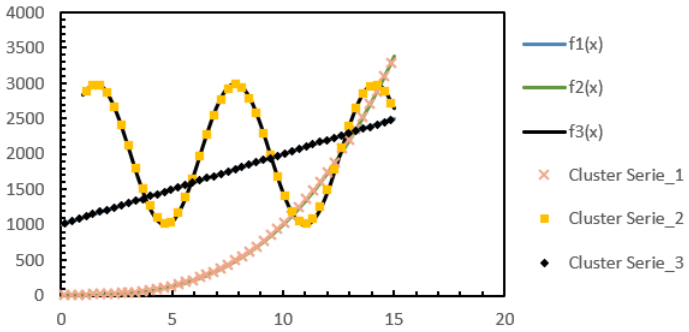

Figure 6: Resultados obtidos no Exemplo 2.

5 Conclusão

Os resultados apresentados concluem que o algoritmo desenvolvido nesse projeto cumpriu satisfatoriamente seu objetivo de obter automaticamente os pontos de séries de um gráfico a partir da imagem em PNG. Ambos os exemplos encontraram convergência facilmente do DBSCAN utilizando os mesmos parâmetros. O algoritmo se mostrou ainda robusto ao conseguir trabalhar com diversas séries no mesmo gráfico e que contem cruzamentos entre diferentes linhas.

Obviamente o algoritmo desenvolvido possui uma série de limitações que poderiam ser atacadas em oportunidades futuras. Como limitações e hipóteses simplificadoras assumidas, destacam-se:

- A necessidade do usuário informar quais os clusters se referem aos eixos xy do gráfico para que estes sejam excluídos da resposta final. Esta tarefa poderia ser realizada automaticamente pelo código ao

buscar por relações matemáticas entre os valores $\bar{y}_i$ obtidos pelos clusters que indicassem qual deles se refere aos eixos;

- A necessidade do usuário informar os valores máximos dos eixos, considerando que ambos iniciam no zero. Um algoritmo de identificação de números na imagem poderia ser aplicado para identificar esses valores, permitindo que o PNG inicial contivesse os labels dos eixos;

- A dificuldade em lidar com gráficos que possuam legenda dentro da área de plotagem. Essa limitação poderia ser contornada ao programar funções que permitissem identificar no PNG a área da legenda e retirar os pixels referentes à ela da clusterização.

Assim, conclui-se que este trabalho aplica com sucesso uma técnica não supervisionada de Machine learning em uma aplicação gráfica. O projeto final encontra-se com algumas limitações e hipóteses simplificadoras, porém este se mostra robusto e com bastante potencial. Tais limitações poderiam ser vencidas futuramente, caso haja interesse no desenvolvimento mais profundo deste projeto.