08

Para saber mais: definindo os parâmetros do DBSCAN através do coeficiente de Silhueta

Assim como para o K-means e o Mean shift, é possível definir os melhores valores para os parâmetros `eps` e `min_samples`. Todavia, como são dois parâmetros, a visualização é um pouco mais difícil. Para vermos como fazer isso, podemos:

```
#20 valores para cada parâmetro.
faixa_min_samples = [i for i in range(1,100,5)]
faixa_eps = [i/10 for i in range(1,100,1)]
```

Precisaremos utilizar dois `for`. Um para variar o min_samples e o outro para variar o faixa_eps. Vamos criar um `for` dentro do outro de modo que para cada valor de min_samples possamos variar toda a faixa de eps. Outro ponto interessante ao qual devemos atentar é que a silhueta não pode ser calculada quando o número de clústeres é igual a 1 ou igual ao número de dados, no caso 178. No primeiro caso não existem vizinhos para comparar o grau de pertença ao grupo. No segundo caso cada cluster só possui um ponto, então, não é possível calcular a semelhança do mesmo com ele mesmo e com os demais. Para evitar esses problemas, então, colocaremos um `if`, ou "se", que basicamente avaliará se o número de labels é diferente de 1 e de 178. Se sim, ele calculará a silhueta. Para contar o número de labels, utilizaremos uma função chamada `counter` da biblioteca collections, que retorna um array onde a primeira posição é o valor e a segunda é o número de ocorrências ou o número de repetições.

```
agrupador.clusters_centers_
```

```
from collections import Counter
```

O número de objetos se encontra na primeira posição do array retornado pela função. Assim sendo, devemos fazer `Counter(labels)[0]`, que retornará o número de diferentes valores dentro do counter ou o número de labels diferentes.

```
valores_silhueta = []
eps_plot = []
min_samples_plot = []
for min_samples in faixa_min_samples:
  for eps in faixa_eps:
    labels = 0
    agrupador = DBSCAN(eps = eps, min_samples = min_samples, metric = 'manhattan')
    agrupador.fit_predict(df)
    labels = agrupador.labels_
    if(Counter(labels)[0] < len(df) and Counter(labels)[0] > 1): #porque não é possível calcula
    # silhueta para apenas  um cluster ou para número de clústeres igual
    # ao número de dados
      media_silhueta = silhouette_score(df, labels)
      valores_silhueta.append(media_silhueta)
      eps_plot.append(eps)
      min_samples_plot.append(min_samples)
```

Para desenhar o gráfico, iremos dessa vez adicionar nome aos eixos. Para fazer isso, utilizaremos o método update layout. Além disso, podemos marcar o ponto no qual o coeficiente de silhueta possui o maior valor, adicionando os atributos `marker_line_color="midnightblue"` e `marker_symbol['x']` para marcar um `x` na cor azul no ponto que dá o maior valor do coeficiente de silhueta.

```
ind = valores_silhueta.index(max(valores_silhueta))
#@title Default title text
fig = go.Figure()
fig.add_trace(go.Scatter3d(x=[min_samples_plot[ind]],
                           y = [eps_plot[ind]], z = [max(valores_silhueta)],
                           mode = 'markers',marker_line_color="midnightblue",
                            marker_symbol=['x']))
fig.add_trace(go.Scatter3d(x=min_samples_plot,
                           y = eps_plot, z = valores_silhueta,
                           mode = 'markers',
                          text = labels))
fig.update_layout(scene = dict(
                    xaxis_title='eps',
                    yaxis_title='min_samples',
                    zaxis_title='silhueta'))
fig.show()
```

O ponto com maior silhueta apresenta os melhores valores para eps e min_samples. Para saber de maneira numérica e não gráfica esses valores, é possível fazer:

```
# pega o índice do vamior valor do coeficiente de silhueta
ind = valores_silhueta.index(max(valores_silhueta))
# printa os valores de eps e min_samples para o maior valor de coeficiente de silhueta encontra
print(eps_plot[ind])
print(min_samples_plot[ind])
```

Por fim, o maior valor do coeficiente de silhueta é:

```
print(max(valores_silhueta))
```