Laboratório Virtual - Atividade 3

Disciplina	RDP – Reconhecimento de Padrões

Objetivos

A terceira atividade prática em laboratório virtual possui como objetivos principais:

- ✓ Realizar um problema de clusterização sobre dados reais;
- ✓ Aplicar o método Elbow para definir a quantidade ideal dos clusters;
- ✓ Visualizar os resultados para demonstrar o grau de separabilidade e a qualidade dos clusters

O dado de entrada contém amostras de carros e técnicas / preço e outras informações sobre eles. O objetivo deste problema é agrupar esses carros em clusters com base em seus atributos.

Abrindo o Projeto no Jupyter

Os arquivos do nosso projeto estão estruturados dentro das pastas da seguinte forma:

```
diretorio-projeto
    └ src
        └ ClusteringCars.ipynb
    └ data
        └ cars.csv
```

Nosso primeiro passo é abrir o arquivo ClusteringCars.ipynb para iniciar o desenvolvimento do nosso projeto. Vamos começar?

Auto - Data Collection

Os dados disponíveis identificam 8 atributos referentes a um modelo de carro diferente. Esse banco de dados, mantido pela Universidade de Irvine[1], designa, para

1 Database aberto pode ser acessado em: https://archive.ics.uci.edu/ml/datasets/Auto+MPG

cada modelo de carro diferente, consumo, número de cilindros, cilindrada, potência, peso, tempo de aceleração de 0 a 100, ano do modelo e o país de fabricação.

Fazendo a leitura do Dataset

Os dados disponíveis se encontram no formato csv. Para realizar a leitura dessa informação devemos criar uma nova aplicação e criar uma seção para a execução da tarefa:

```
01 import pandas as pd
02 import numpy as np
03
04 # realiza a leitura dos dados
05 dataset = pd.read_csv('../data/cars.csv')
06 # realizar a leitura dos valores de todas as colunas
07 X = dataset.iloc[:,:-1].values
08 # transforma a matriz em um dataframe
09 X = pd.DataFrame(X)
```

Transformando os dados não numéricos em numéricos e preenchendo espaços vazios

Antes de realizar o processo de clusterização devemos, mais uma vez, transformar os dados para fazer com que as informações categóricas sejam transformadas em valores numéricos. Para isso, podemos utilizar a função `to_numeric` presente na biblioteca pandas.

No entanto, essa função deve ser aplicada de forma independente sobre cada coluna. Então precisamos iterar sobre os dados realizando a operação coluna a coluna preenchendo cada dado não numérico com um valor numérico.

Caso a célula esteja vazia, valor nulo, devemos escolher um número para preenche-la. O valor numérico escolhido depende do contexto do problema. No nosso caso, como cada vetor de atributos não possui relação com nenhum outro, preencheremos as células vazias com o valor 0. Para isso utilizaremos a função `fillna()` e escolheremos o tipo int64

```
0  # lista com as colunas do DataFrame
9  columnlist = ['consumo', ' cilindros', ' cilindrada', 'potencia', ' peso', 'tempo-0-a-100',
1  'ano']
0  # atribui, a cada coluna, seu respectivo título através do atributo columns
1  X.columns = columnlist
1
1  # Tranforma os dados faltantes em dados numéricos. A função do pandas to_numeric
2  # No nosso caso, fazemos o mais simples que é preenchê-los com 0.
1  for i in range(0, len(columnlist)):
3      X[columnlist[i]] = pd.to_numeric(X[columnlist[i]], errors='coerce')
1                                                       .fillna(0)
4                                                       .astype(np.int64)
```

1 5 1 6 1 7 1 8 1 9 2 0	

Escolhendo o número de clusters

Para garantir que o KMeans apresente resultados coerentes precisamos escolher bem a quantidade de clusters. Para isso, utilizaremos o método Elbow. Para isso, faremos a seguinte operação:

1) Escolheremos arbitrariamente a quantidade de clusters que queremos analisar.
2) Rodaremos para cada uma das quantidades o algoritmo KMeans.
3) Coletaremos métricas de avaliação do desempenho do algoritmo. Sua silhueta, por exemplo
4) Escolheremos a quantidade de clusters mais adequada a resolução do problema

Para isso, importaremos o KMeans implementado na biblioteca sklearn.cluster. Criaremos, dentro do intervalo definido, modelos com diferentes quantidades de cluster. O comando para se criar esse modelo é expresso na linha 28. Nele definimos a quantidade de iterações "max_iter", o número de seeds a cada nova rodada "n_init", e se queremos utilizar alguma função para geração das seeds "random_state"

```
2  # importa o KMeans
1  from sklearn.cluster import KMeans
2  # vetor de resultados
2  wcss = []
2  # escolhemos de 2 a 5 clusters para análise
3  for i in range(2,5):
2      # criamos o modelo KMeans a ser testado
4      kmeans = KMeans(n_clusters=i,init='k-means++', max_iter=300, n_init=10,
2                                                                  random_state=0 )
5      # rodamos o KMeans sobre a base de dados
2      kmeans.fit(X)
6      # avaliamos a qualidade da clusterização realizada
2      wcss.append(kmeans.inertia_)
7
2
8
2
9
3
0
3
1
3
2
3
3
3
4
```

Após a criação do modelo podemos então aplicar o KMeans. Isso é feito pela função fit (linha 31). O resultado desse modelo pode ser obtido através de parâmetros de

qualidade como, por exemplo, a soma quadrática das distâncias de cada instância do cluster ao centróide (silhueta). Esse parâmetro é dado pela variável `kmeans.inertia_`.

Visualizando o resultado e aplicando o método Elbow

Após a realização da avaliação, precisamos plotar os resultados. Para isso utilizamos a biblioteca pyplot. Para facilitar a leitura criamos um gráfico de linha com os labels de cada eixo.

```
3  import matplotlib.pyplot as plt
5
3  # criamos o gráfico com os dados coletados
6  plt.plot(range(1,11),wcss)
3  plt.title('The Elbow Method')
7  plt.xlabel('Número de clusters')
3  plt.ylabel('WCSS')
8  plt.show()
3
9
4
0
4
1
4
2
```

Agrupamento

Agora que já definimos a quantidade de clusters basta rodar o algoritmo novamente com a quantidade escolhida de clusters. No nosso exemplo, utilizaremos 2 clusters (obviamente não é a quantidade ideal), criamos novamente o modelo (linha 46) e executamos o algoritmo com uma única diferença: utilizamos a função `fit_predict`. Essa função, além de clusterizar, realiza uma predição dos valores identificando a qual cluster cada amostra pertence (linha 49). Esses valores serão utilizados para visualizar o resultado.

```
4  # definimos a quantidade de clusters ideal
3  clusters = 2
4  # Aplicamos o KMeans com a quantidade de clusters determinados
4  kmeans = KMeans(n_clusters=clusters,init='k-means++',max_iter=300,n_init=10,
4                                                          random_state=0)
5  # realizamos a predição sobre a base de dados
4  y_kmeans = kmeans.fit_predict(X)
6
4
7
4
8
4
9
```

Visualização dos dados

Para concluir nossa tarefa é sempre importante uma visualização. Visualizar dados clusterizados é sempre um desafio pois as limitações impostas pela quantidade de dimensões que somos capazes de interpretar pode nos oferecer falsas impressões. Contudo, ainda sim é importante realiza-lo. Para essa tarefa devemos entender o resultado da fase anterior.

A estrutura `y_kmeans` (linha 49) armazena a predição de cluster para cada amostra na forma de um array. Isso significa que, para cada elemento, ele possui o índice do cluster ao qual ele pertence.

O primeiro passo, então é transformar o DataFrame em uma matriz que possa ser plotada (linha 51). Como queremos que esses dados possam ser identificados no gráfico precisamos de cores diferentes. Criamos, então, uma lista de cores para que cada cluster tenha sua identidade (linha 54).

Como utilizaremos um Scatterplot, teremos, à nossa disposição, dois eixos. Definimos então quais características desejamos visualizar no gráfico (linha 58 e 58). Se quisermos mudar a visualização dos dados podemos mudar quais os atributos desejamos ver.

Para plotar somente as amostras de um determinado cluster, utilizamos a capacidade do pandas em filtras os valores dentro de uma matriz. Fazemos isso quando identificamos uma condição para que a instância seja selecionada (linha 63: `R[y_kmeans == i, ...`). Essa condição faz com que somente as amostras que possuem o número de cluster igual a i sejam recuperadas. Assim, plotamos, cluster a cluster, seus valores sobre os gráficos.

Por fim, plotamos os centroides. O sklearn armazena as coordenadas dos centroides em uma estrutura separada identificada pela variável `kmeans.cluster_centers_` (linha 66). Essa variável armazena as coordenadas de cada centroide em cada um dos atributos do problema.

Assim, conseguimos visualizar nossos clusters!

```
50 # para plotar os dados transformamos o DataFrame em uma matriz de valores numéricos
51 R = X.as_matrix(columns=None)
52 # se quiser mais cores visite https://www.webucator.com/blog/2015/03/python-color-constants-module/
53 color=['red','green','blue','black','turquoise','violet','orange']
54 # determina quais as características serão plotadas no gráfico no caso a primeira coluna (0 - 'consumo') e a segunda (1 - 'cilindros')
55 x_axis = 0
56 y_axis = 1
57
```

```
61  # plota os clusters utilizando duas características do modelo
62  for i in range(0,clusters):
63      labelCluster = 'Cluster '+str(i+1)
64      plt.scatter(R[y_kmeans == i, x_axis], R[y_kmeans == i, y_axis],s=50,
65                                                          c=color[i],label=labelCluster)
66  # plota no gráfico
67  plt.scatter(kmeans.cluster_centers_[:,x_axis],kmeans.cluster_centers_[:,y_axis],
68                                                  s=300,c='yellow',label='Centroides')
69  plt.title('Clusterização dos modelos de carro')
70  plt.legend()
    plt.show()
```

Exercício

O teste de exemplo foi feito com 2 clusters e foi plotado com duas variáveis que não permitem visualizar como os clusters estão, de fato, separados. Refaça a clusterização considerando o resultado obtido com o método Elbow e escolha as características que demonstrem a melhor separação no gráfico.

Relatório: o relatório final da prática deve conter a escolha, e sua justificativa, da quantidade de clusters, os valores da silhueta de cada quantidade de clusters testada e o gráfico final com os atributos que melhor demostram a separação dos grupos.

Conclusão

Nessa segunda atividade aprendemos como podemos criar clusters. Entendemos como transformar os dados e escala-los para reduzir o impacto das amplitudes de cada atributo sobre o resultado final.