Agrupamento

Introdução

Até agora, todos os modelos que vimos no curso podem ser classificados como modelos supervisionados de mineração de dados. Eles começam com um conjunto de dados rotulados, e os usam como base para fazer previsões sobre novos dados, não rotulados.

Hoje vamos ver um modelo chamado agrupamento, ou clusterização. Este modelo, ao contrário dos demais que já vimos, é um exemplo de aprendizado não supervisionado, em que nós trabalhamos com dados completamente não rotulados.

A ideia por trás do modelo é que, quando olhamos para uma fonte de dados, é natural que eles, de alguma forma, formem agrupamentos. Um conjunto de dados que mostre onde milionários moram no Rio provavelmente possui agrupamentos em lugares como Leblon ou Lagoa. Um conjunto de dados que mostre espectadores de TV ou de um serviço de streaming provavelmente possuem agrupamentos em torno da idade desses espectadores (nos EUA, a audiência de pessoas entre 18 e 34 é particularmente relevante para decisões de publicidade ou sobre quais programas são mantidos e quais são cancelados).

Diferente dos problemas que temos visto, geralmente não há um agrupamento "correto". Um esquema de agrupamento alternativo pode agrupar pessoas entre 18 e 34 com "moram sozinhos", outros podem agrupar com "moram com os pais". Nenhum esquema é necessariamente mais correto - pelo contrário, cada um é melhor no que diz respeito à sua própria métrica "quão bom são os agrupamentos?"

Além disso, os agrupamentos não se rotulam sozinhos. Temos que fazer isso vendo os dados contidos em cada um.

Algoritmo K-means

Um dos algoritmos de agrupamento mais comum é o "K-means", ou k-médias. Nesse algoritmo, os dados são agrupados em torno de k grupos, onde cada grupo possui mais ou menos a mesma variância entre os dados. O número de grupos é fornecido pelo usuário.

Para aplicarmos esse modelo, assumimos as seguintes premissas:

- Primeiro, que os agrupamentos são convexos, ou seja, possuem um escopo "fechado", como se todos os dados estivessem distribuídos num círculo, por exemplo;
- Segundo, que todas as features são igualmente escaladas, ou seja, não temos dados da ordem de milhões e outros na ordem de unidades. Caso não estejam, precisamos aplicar recursos para escalá-los, como já fizemos em outras aulas;

- Terceiro, que os agrupamentos são balanceados, ou seja, possuem aproximadamente o mesmo número de observações.

O modelo funciona da seguinte forma:

- Ele escolhe aleatoriamente K números de centróides dentre as amostras como centros iniciais;
- Para cada amostra, calcula a distância Euclidiana entre a amostra e o centróide;
- Designa cada amostra para o agrupamento mais próximo;
- Tira a média de todos os pontos em cada agrupamento e reposiciona os centroides;
- Mede a distância entre os centróides antigos e os novos;
- Caso essa distância seja diferente de 0, repete os primeiros passos até que essa

Aplicação

Vamos aplicar esse modelo na base de dados iris, do sklearn.

Preparação dos dados

```
%matplotlib inline

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

plt.style.use("ggplot")
```

O `ggplot` é um estilo de gráfico que é adaptado de um outro estilo desenvolvido para R.

```
from sklearn.datasets import load_iris

iris = load_iris()
```

Agora vamos converter esses dados para um DataFrame no pandas. Os dados do DF serão obtidos com `iris.data`, enquanto que os nomes das colunas serão através de `iris.feature_names`.

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2

Já trabalhamos com esse conjunto de dados antes, mas vamos rever algumas informações.

```
df.shape
```

```
(150, 4)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

Vamos ver se temos dados nulos ou ausentes.

```
df.isnull().sum()
```

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
```

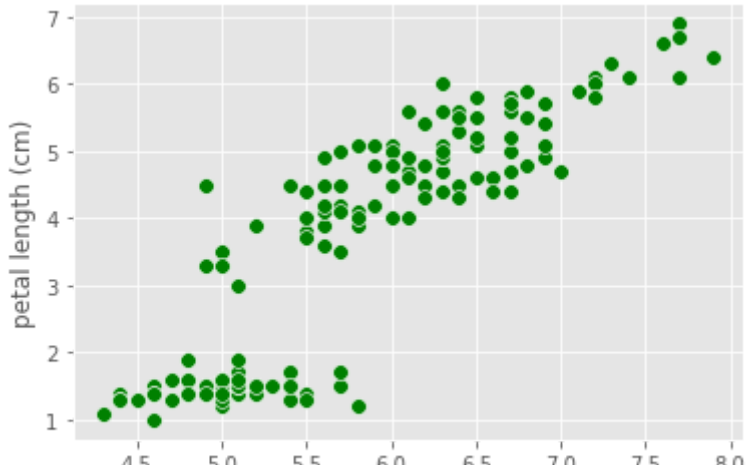
Visualização dos dados

Agora vamos criar um gráfico de dispersão entre *sepal length* e *petal length*. Vamos usar apenas dois dos atributos por conta da limitação na visualização em 2D.

```
sns.scatterplot(
    x="sepal length (cm)",
    y="petal length (cm)",
    data=df,
    s=60,
    color="green"
)

# Se quisermos salvar a imagem
# plt.savefig("scatterplot.png", dpi=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa29498fb50>
```



Vamos agora agrupar esses dados baseado nas suas similaridades.

Modelagem

```
from sklearn.cluster import KMeans

data = df[["sepal length (cm)", "petal length (cm)"]].values
model = KMeans(n_clusters=3, random_state=10)
model.fit(data)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=10, tol=0.0001, verbose=0)
```

Análise dos resultados

```
centroids = model.cluster_centers_
centroids
```

```
array([[6.83902439, 5.67804878],
       [5.00784314, 1.49215686],
       [5.87413793, 4.39310345]])
```

O resultado indicou um array bidimensional que consiste nas coordenadas dos centros dos três agrupamentos.

Podemos agora usar a função `predict` para identificar os clusters de cada amostra.

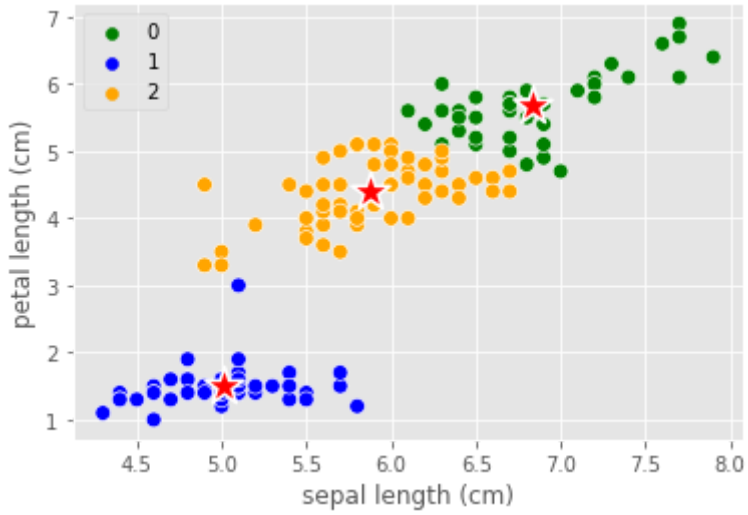
```
labels = model.predict(data)
```

Visualizar os resultados

```
sns.scatterplot(x="sepal length (cm)", y="petal length (cm)", data=df, s=60, hue=labels, palette=["green'
sns.scatterplot(x=centroids[:, 0], y=centroids[:, 1], marker="*", s=400, color="red")
```

```
# Se quisermos salvar a imagem
# plt.savefig("scatterplot.png", dpi=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa2949063d0>
```



Encontrando o número ótimo de agrupamentos

Um dos maiores desafios em modelos não supervisionados é avaliar o desempenho do modelo. Uma má escolha no número de agrupamentos pode diminuir o desempenho do modelo. Um método bem comum para auxiliar na escolha desse número é o *Elbow method*.

Esse método leva em consideração a soma dos erros quadrados, que já vimos quando falamos das regressões. Essa soma, quando considerada apenas dentro de cada agrupamento, é chamada de inércia do agrupamento, e pode ser obtida pelo atributo `inertia_`. Por exemplo, no nosso caso, que escolhemos `k = 3`:

```
model.inertia_
```

```
53.809978644106934
```

Baseado nessa inércia, podemos montar um gráfico para apoiar, chamado de *elbow plot*, ou gráfico cotovelo, para estimar um número ótimo de agrupamentos para cada tarefa. O objetivo é identificar um `k` tal que a inércia comece a diminuir mais devagar se continuarmos aumentando o `k`.

Nas próximas linhas vamos implementar esse gráfico.

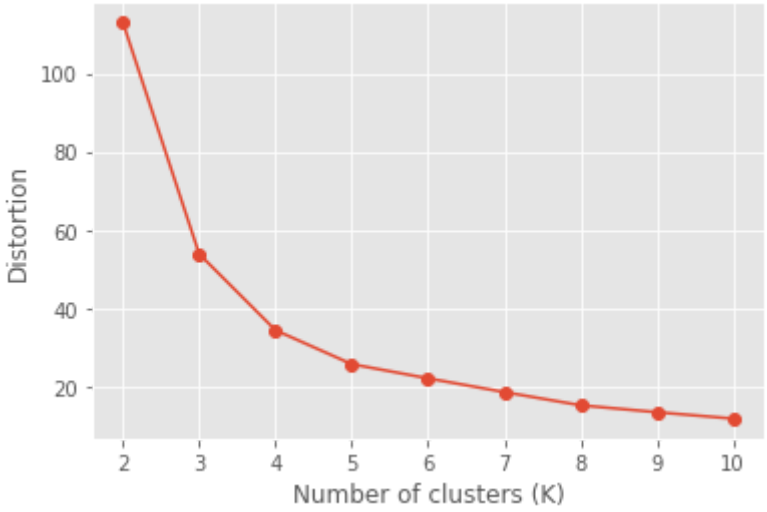
```
distortions = []

for k in range(2, 11):
    kmeans_model = KMeans(n_clusters=k, random_state=10)
    kmeans_model.fit(data)
    distortions.append(kmeans_model.inertia_)

plt.plot(range(2, 11), distortions, marker="o")
```

```
plt.xlabel("Number of clusters (K)")
plt.ylabel("Distortion")
# plt.savefig("elbow plot.png")
```

```
Text(0, 0.5, 'Distortion')
```



Podemos ver do gráfico que a inércia diminui bem rápido até `k = 4`, porém a partir daí a queda do valor começa a diminuir.