k-means

Objetivo

O problema de agrupamento (*clustering* em inglês) consiste em particionar um conjunto $x_1, x_2, \ldots, x_n$ de pontos em $\mathbb{R}^d$ em k partes[10]. Podemos ver uma partição como uma função

$$\sigma : [n] \to [k]$$

que associa para x_i à $\sigma(i)$-ésima parte.

Uma vez que temos uma partição σ podemos obter o centro de cada grupo (ou *cluster*) fazendo

$$\mu_j = \frac{1}{|\{i : \sigma(i) = j\}|} \sum_{i:\sigma(i)=j} x_i \text{ para todo } j \in [k].$$

Para o problema de k-means queremos encontrar $\sigma^\star$ tal que

$$\sigma^\star \in \arg\min_{\sigma:[n]\to[k]} \sum_{i=1}^{n} \|x_i - \mu_{\sigma(i)}\|_2^2.$$

Algoritmo clássico

O algoritmo clássico do *k-means*, também conhecido como *naive k-means*, é uma abordagem iterativa para encontrar uma solução aproximada para o problema de *clustering*. Começamos fixando um número k de partes. Após a inicialização, o algoritmo alterna dois passos principais: atribuição de pontos aos *clusters* e atualização dos centros dos *clusters*.

(i) **Inicialização**: para inicializar o algoritmo escolhemos algum particionamento σ aleatoriamente (veja exemplo na Figura 9).

(ii) **Atribuição de Pontos aos Clusters**: Dada uma partição atual σ, o primeiro passo é atribuir cada ponto x_i ao *cluster* cujo centro é o mais próximo, ou seja:

$$\sigma(i) = \arg\min_{j\in[k]} \|x_i - \mu_j\|_2^2.$$

[10] Normalmente k é muito menor que n.

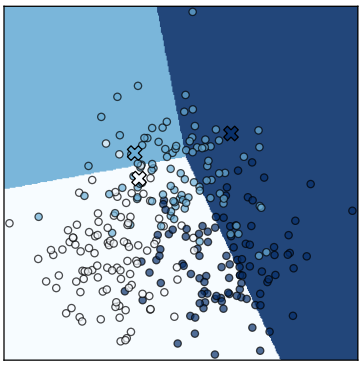

Figura 9: Exemplo de inicialização do *k-means*, no exemplo temos 3 classes e $k = 3$.

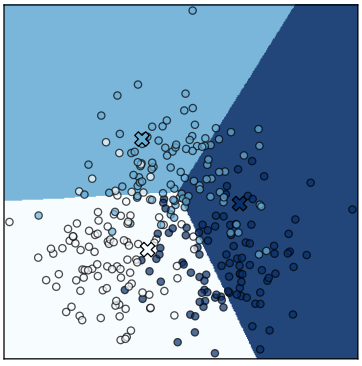

Figura 10: O mesmo exemplo anterior, mas agora após uma iteração com um passo de atribuição e um de atualização de centros.

Isso significa que cada ponto é associado ao *cluster* cujo centro é o mais próximo em termos de distância Euclidiana.

(iii) **Atualização dos Centros dos Clusters**: Uma vez que os pontos foram atribuídos aos *clusters*, o próximo passo é recalcular os centros dos *clusters* com base na nova partição. Os novos centros são dados por:

$$\mu_j = \frac{1}{|\{i : \sigma(i) = j\}|} \sum_{i:\sigma(i)=j} x_i \quad \text{para todo } j \in [k].$$

Esse processo de atribuição de pontos (i) e atualização dos centros (ii) é repetido iterativamente até que não haja alterações significativas nas atribuições dos pontos ou até que um número máximo de iterações seja atingido. O exemplo da Figura 9 é continuado nas Figuras 10 e 11 e mostra o algoritmo se estabilizando com a passagem das iterações.

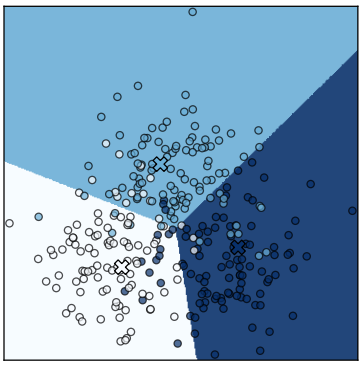

Figura 11: Resultado final após algumas iterações.

A escolha da inicialização dos centros dos clusters pode afetar o desempenho do algoritmo. Uma prática comum é inicializar os centros aleatoriamente a partir dos dados ou utilizando algum método heurístico. O algoritmo pode convergir para um mínimo local, e a escolha do número de clusters k também é crítica. Diversas variações e melhorias foram propostas para lidar com essas questões.

Um exemplo com código

Agora veremos como o *k-means* se comporta em um conjunto de dados não artificial. Os dados que usaremos serão imagens de dígitos $(0, 1, 2, 3, \ldots, 9)$ escritos à mão e digitalizados como imagens em escala de cinza (16 bits) com $8x8$ pixels.

Podemos então imaginar que essa base de dados é composta de matrizes $8x8$ cujas entradas são inteiros entre 0 e 15. Mas também podemos ver essas matrizes como um único vetor de 64 coordenadas. Nesse caso, a distância entre duas imagens é simplesmente a norma Euclideana da diferença entre essas imagens.

O Código 1 importa as bibliotecas, carrega os dados e permitem visualizar um dígito, que é apresentado na Figura 12. Na Figura 13 é possível observar como a norma Euclideana pode ser utilizada para obter uma distância entre duas representações de dígitos. A diferença entre um dígito zero e o dígito um foi de 59.6, já entre o dígito zero e outro dígito zero foi de 23.7.

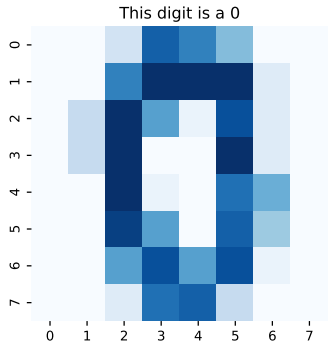

Figura 12: Saída do Código 1.

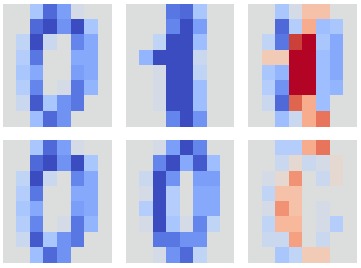

Figura 13: Exemplo de diferença entre dois dígitos. Nas duas linhas a imagem da direita é a diferença das outras duas.

```
# Import libs
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
```

```python

from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder

# Set a nice figure size
plt.figure(figsize=(4, 4))

# Load data
data, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data.shape, np.unique(labels).size

# Show a random digit
i = np.random.choice(n_samples)

sns.heatmap(data[i].reshape(8,8), cbar=False, cmap="Blues")
plt.title(f"This digit is a {labels[i]}")
plt.show()
```

Código 1: Importando bibliotecas e visualizando um dígito aleatório

Após executar o *k-means* podemos associar para cada cluster uma classe tomando a classe como a classe mais comum no cluster. O Código 2 faz isso e produz uma matriz relacionando para cada dígito os clusters em que suas representações aparecerem (em proporção)[11].

```python
# Apply kmeans
kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4)
kmeans.fit(data)

# Assign cluster labels
encoder = OneHotEncoder(sparse_output=False, categories='auto')
oh_kmeans_clusters = encoder.fit_transform(kmeans.labels_.reshape(-1, 1))
oh_labels = encoder.fit_transform(labels.reshape(-1, 1))
confusion = oh_labels.T @ oh_kmeans_clusters

# Plot digits vs clusters
sns.heatmap(confusion[:,np.argmax(confusion, axis=1)], cbar=False, cmap="
    ↪ Blues")
plt.title("Digits vs clusters")
plt.xlabel("Digit")
plt.ylabel("Cluster")
plt.show()
```

Código 2: Executando o k-means e visualizando digitos e clusters

Por fim, o Código 3 produz a Figura 15 usando o método de redução de dimensionalidade PCA[12].

```python
# Apply PCA to data and cluster centers
reduced_data = PCA(n_components=2).fit_transform(
    np.vstack([data, kmeans.cluster_centers_])
)

# Create a scatter plot of the 2D PCA result
data_points = pd.DataFrame({
    "PC1": reduced_data[:-10,0],
    "PC2": reduced_data[:-10,1],
    "Digit": labels
})
```

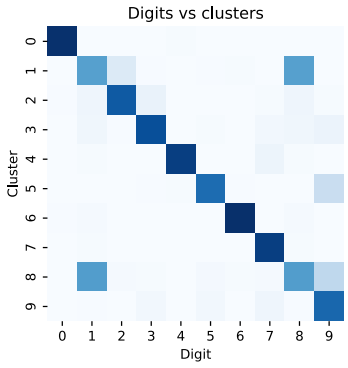

Figura 14: Saída do Código 2.

[11] O que explica que o cluster do dígito 8 tenha uma quantidade significativa de dígitos 1 ou 9? E que existam alguns noves na classe do dígito 5?

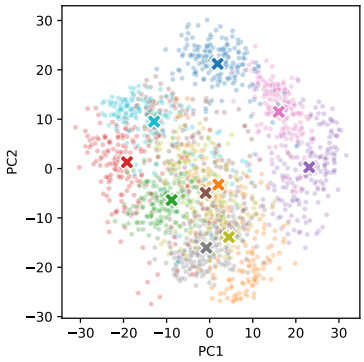

Figura 15: Saída do Código 3.

[12] O PCA busca achar uma transformação linear entre o espaço original, no caso $\mathbb{R}^{64}$, e o espaço final, no caso $\mathbb{R}^2$, que distorça o mínimo possível as distancias no espaço original. Permitindo visualizar os dados.

```
cluster_centers = pd.DataFrame({
    "PC1": reduced_data[-10:,0],
    "PC2": reduced_data[-10:,1],
    "Digit": np.argmax(confusion, axis=0)
})

sns.scatterplot(data_points, x="PC1", y="PC2", hue="Digit", palette="tab10",
    ↪  marker="o", s=15, legend=False, alpha=.3)
sns.scatterplot(cluster_centers, x="PC1", y="PC2", hue="Digit", palette="
    ↪ tab10", marker="X", s=120, legend=False)
plt.show()
```

Código 3: Visualização em 2 dimensões