Clustering com K-means: Uma Visão Geral

Társila Samille

March 25, 2024

O que é Clustering?

- Clustering é uma técnica de análise de dados que agrupa pontos semelhantes em clusters.
- Facilita a identificação de padrões e tendências nos dados.
- K-means é um dos algoritmos mais populares nesse contexto.

Como funciona o K-means?

1. Inicialização: Escolha aleatória de K centróides.
2. Atribuição: Atribua cada ponto ao cluster com o centróide mais próximo.
3. Atualização: Recalcule os centróides como a média dos pontos em cada cluster.
4. Repetição: Repita até a convergência dos centróides.

Algoritmo K-means

```python
import numpy as np
from numpy.linalg import norm

class Kmeans:
    '''Implementing Kmeans algorithm.'''

    def __init__(self, n_clusters, max_iter=100,
    random_state=123):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.random_state = random_state

    def initializ_centroids(self, X):
        np.random.RandomState(self.random_state)
        random_idx = np.random.permutation(X.shape[0])
        centroids = X[random_idx[:self.n_clusters]]
        return centroids
```

Algoritmo K-means

```python
class Kmeans:

    def compute_centroids(self, X, labels):
        centroids = np.zeros((self.n_clusters, X.shape[1]))
        for k in range(self.n_clusters):
            centroids[k, :] = np.mean(X[labels == k, :],
    axis=0)
        return centroids

    def compute_distance(self, X, centroids):
        distance = np.zeros((X.shape[0], self.n_clusters))
        for k in range(self.n_clusters):
            row_norm = norm(X - centroids[k, :], axis=1)
            distance[:, k] = np.square(row_norm)
        return distance

    def find_closest_cluster(self, distance):
        return np.argmin(distance, axis=1)
```

Algoritmo K-means

```
class Kmeans:

    def compute_sse(self, X, labels, centroids):
            distance = np.zeros(X.shape[0])
            for k in range(self.n_clusters):
                distance[labels == k] = norm(X[labels == k]
    - centroids[k], axis=1)
            return np.sum(np.square(distance))
```

Algoritmo K-means

```
class Kmeans:
    def fit(self, X):
            self.centroids = self.initializ_centroids(X)
            for i in range(self.max_iter):
                old_centroids = self.centroids
                distance = self.compute_distance(X,
    old_centroids)
                self.labels = self.find_closest_cluster(
    distance)
                self.centroids = self.compute_centroids(X,
    self.labels)
                if np.all(old_centroids == self.centroids):
                    break
            self.error = self.compute_sse(X, self.labels,
    self.centroids)
```

Algoritmo K-means

```python
class Kmeans:

    def predict(self, X):
        distance = self.compute_distance(X, self.centroids)
        return self.find_closest_cluster(distance)
```

K-means in action

K-Means Clustering GIF.

Algoritmo K-means

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
data = np.random.rand(100, 2) * 10
# Define the number of clusters (k)
k = 3

# Create a KMeans object with k clusters
kmeans = cluster.KMeans(n_clusters=k, random_state=0)

# Fit the KMeans model to the data
kmeans.fit(data)

cluster_labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

Em quais áreas o K-means pode ser aplicado?

- Segmentação de mercado
- Análise de imagens
- Agrupamento de documentos
- Recomendação de produtos
- Detecção de fraudes
- E muito mais!

Como avaliar a qualidade dos clusters?

Métodos

- Método do Cotovelo
- Análise de Silhueta
- Índice de Davies-Bouldin
- Índice de Calinski-Harabasz

Como avaliar a qualidade dos clusters?

Método do Cotovelo (Elbow Method)

Prós:

- Fácil de entender e implementar.
- Fornece uma medida visual para determinar o número ideal de clusters.

Contras:

- Às vezes, pode ser subjetivo decidir o ponto de inflexão no gráfico.

Cálculo:

Soma dos Quadrados das Distâncias Intra-cluster

Interpretação:

- Identifique o ponto onde a curva começa a nivelar-se.
- O número ideal de clusters é geralmente escolhido nesse ponto.

Elbow Method

```
# Run the Kmeans algorithm and get the index of data points clusters
sse = []
list_k = list(range(1, 10))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(X_std)
    sse.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(6, 6))
plt.plot(list_k, sse, '-o')
plt.xlabel(r'Number of clusters *k*')
plt.ylabel('Sum of squared distance');
```

Elbow Method

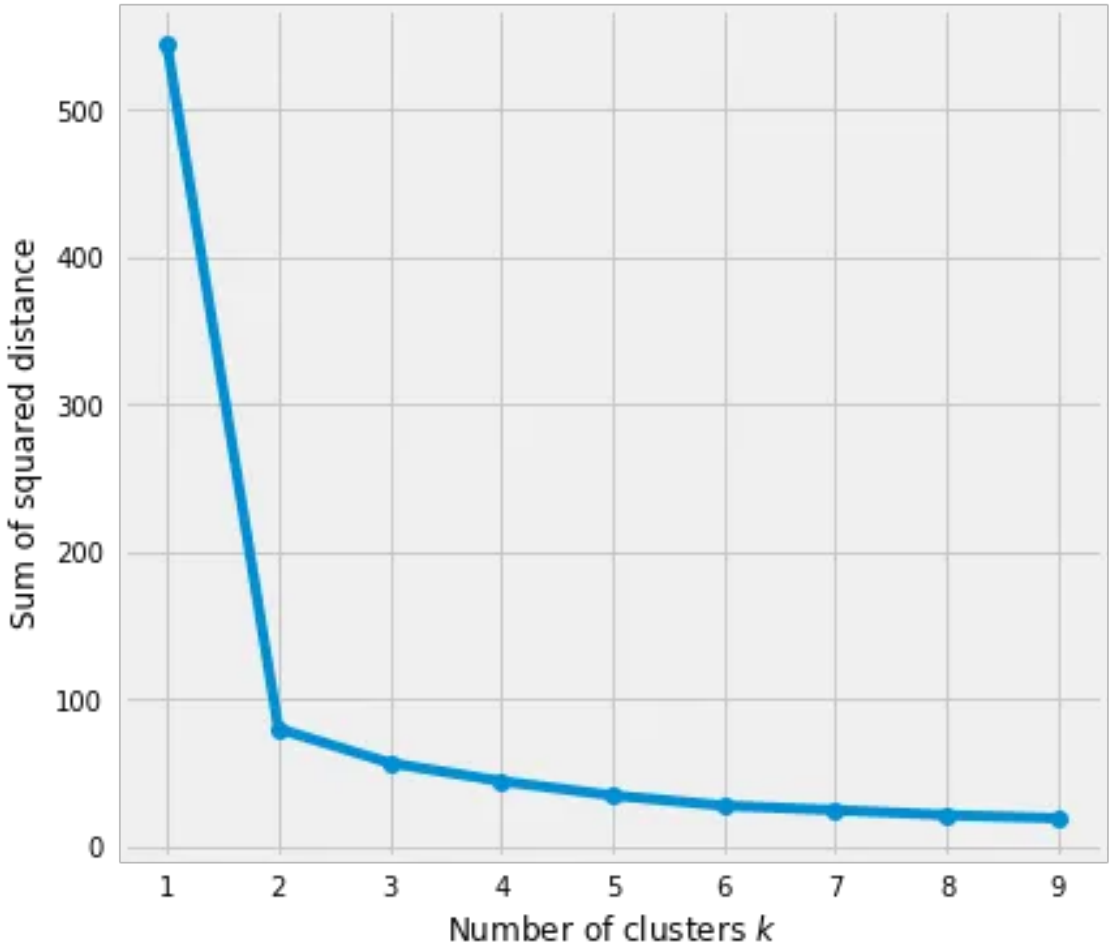

Figure: Elbow Method

Como avaliar a qualidade dos clusters?

Análise de Silhueta (Silhouette Analysis)

Prós:

- Fornece uma medida de quão bem cada objeto se encaixa em seu próprio cluster.
- Valores próximos de $+1$ indicam uma boa separação entre clusters.

Contras:

- Não fornece uma indicação direta do número ideal de clusters.

Como avaliar a qualidade dos clusters?

Análise de Silhueta (Silhouette Analysis)

Cálculo:

- Calcule a média das distâncias de x^i para todos os outros pontos no mesmo cluster.
- Calcule a distância média de x^i para todos os pontos no cluster mais próximo.

$$\text{Coeficiente de Silhueta} = \frac{b^i - a^i}{\max(a^i, b^i)}$$

Interpretação:

- Valores próximos de $+1$ são desejáveis, indicando que os pontos estão bem agrupados.

Silhouette Analysis

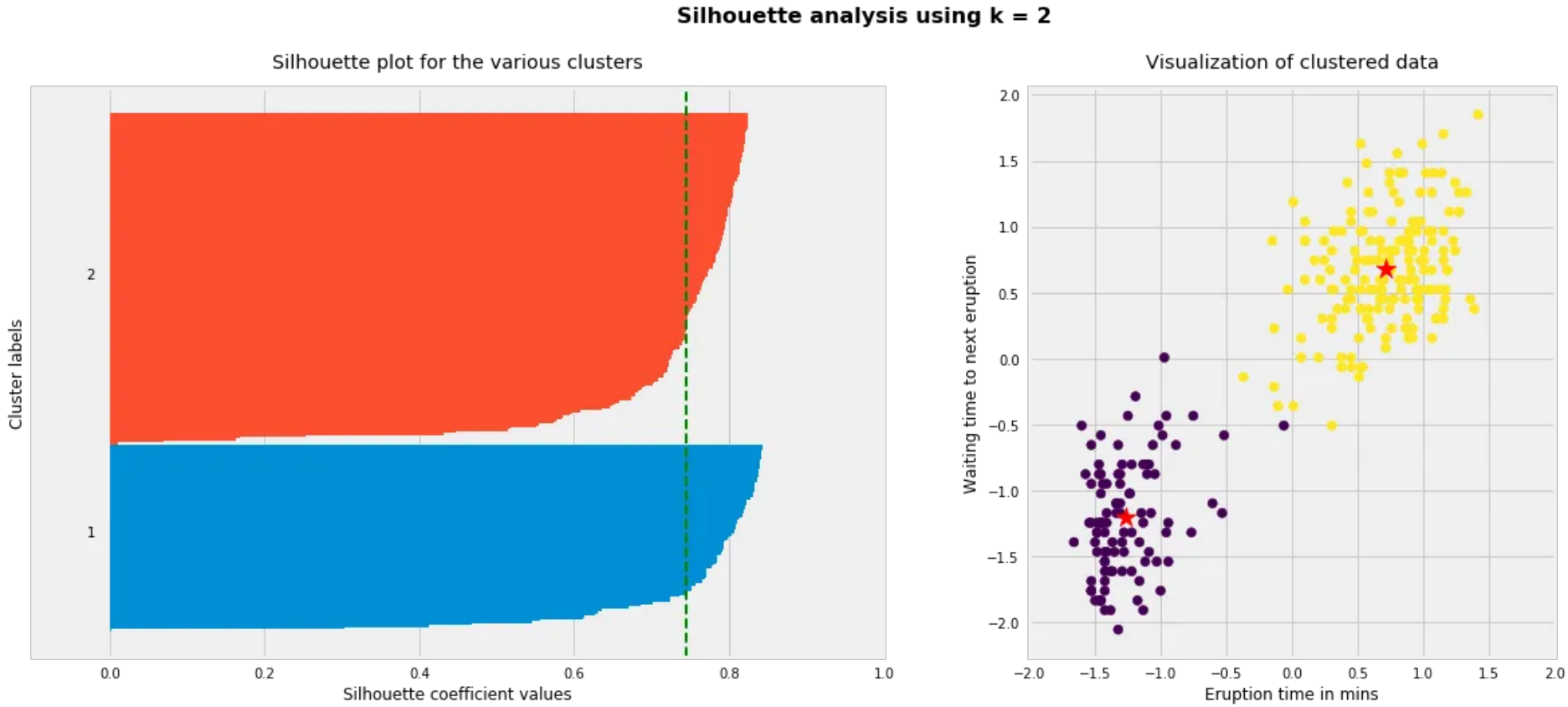

Figure: Silhouette Analysis

Silhouette Analysis

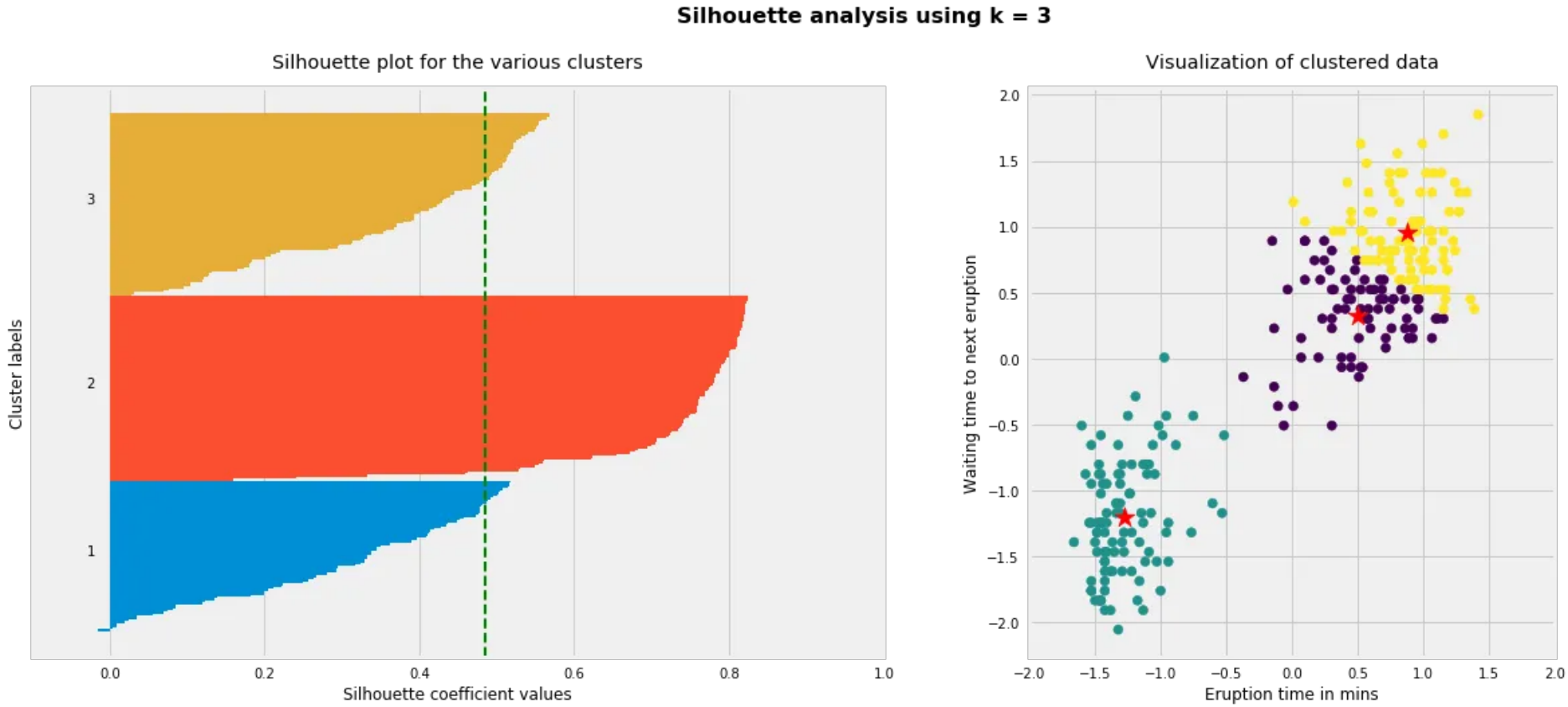

Figure: Silhouette Analysis

Silhouette Analysis

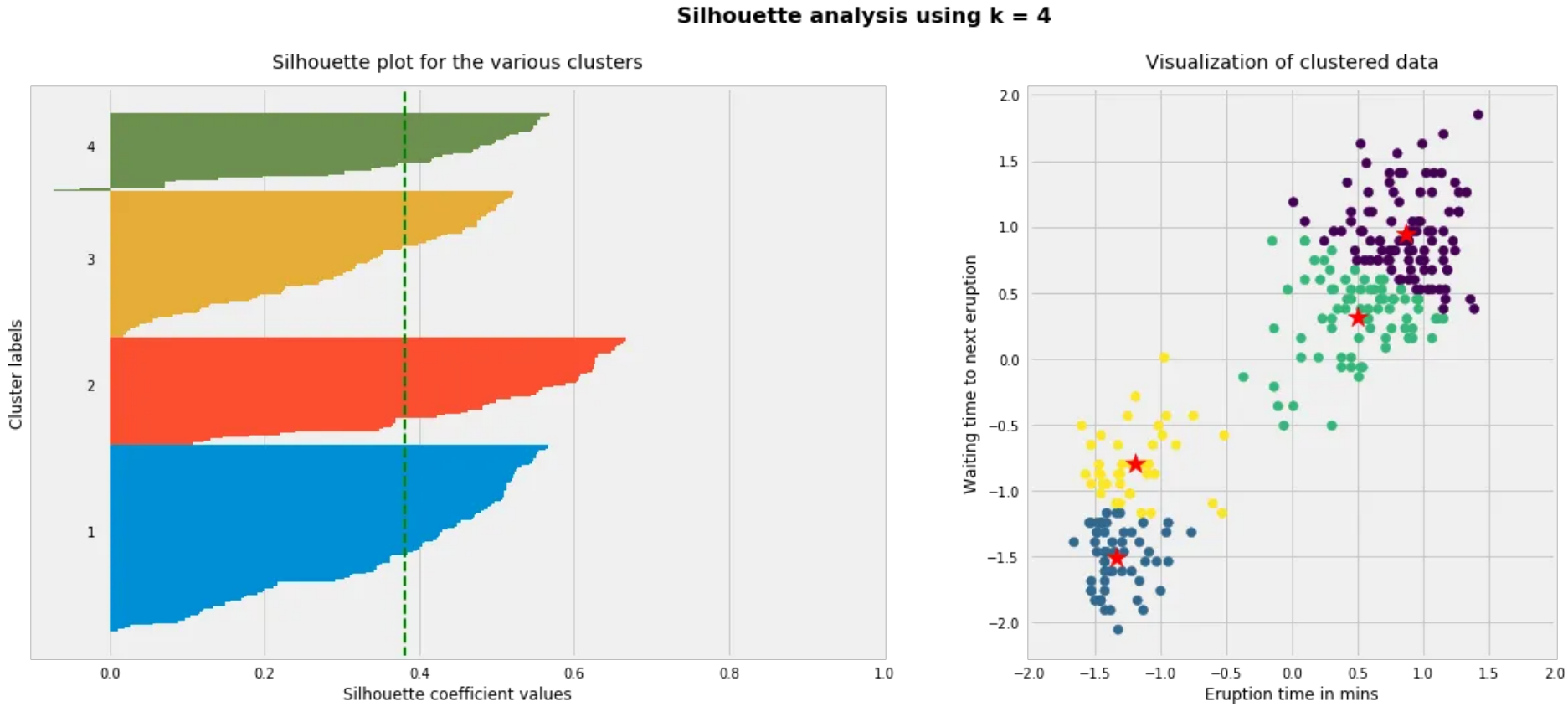

Figure: Silhouette Analysis

Como avaliar a qualidade dos clusters?

Índice de Davies-Bouldin (Davies-Bouldin Index)

Prós:

- Fornece uma medida da separação entre clusters, considerando também a dispersão dentro de cada cluster.

Contras:

- Não é tão intuitivo como outros métodos.

Cálculo:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right)$$

Interpretação:

- Quanto menor o índice, melhor a separação entre os clusters.

Como avaliar a qualidade dos clusters?

Índice de Calinski-Harabasz (Calinski-Harabasz Index)

Prós:

- Fornece uma medida de quão bem os clusters estão separados uns dos outros.

Contras:

- Sensível ao número de clusters.

Como avaliar a qualidade dos clusters?

Índice de Calinski-Harabasz (Calinski-Harabasz Index)

Cálculo:

$$CH(k) = \frac{B(k)}{(k-1)} \cdot \frac{N-k}{W(k)}$$

onde:

- $CH(k)$ é o índice de Calinski-Harabasz para um agrupamento com k clusters.
- $B(k)$ é a dispersão (variância) entre clusters para o agrupamento.
- $W(k)$ é a dispersão (variância) dentro de clusters para o agrupamento.
- N é o número total de pontos de dados.
- k é o número de clusters.

Interpretação:

- Valores mais altos indicam uma melhor separação entre os clusters.

Quais são os pontos fortes e fracos do K-means?

Vantagens

- Simples e eficiente
- Fácil de interpretar
- Escalável para grandes conjuntos de dados

Desvantagens

- Sensibilidade à inicialização dos centróides
- Dependente do número de clusters pré-definido (K)
- Pode não funcionar bem com clusters de formas irregulares

Resumo e Considerações Finais

- O K-means é um algoritmo versátil e útil para diversas aplicações.
- É importante entender suas vantagens e desvantagens para usá-lo eficazmente.
- A escolha do número de clusters e a avaliação da qualidade dos clusters são etapas importantes.

Referências

- DABBURA, Imad. **K-Means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks**. Disponível em: <https://medium.com/geekculture/k-means-clustering-how-it-works-finding-the-optimum-numbe Acesso em: [19 mar 2024].