Introdução ao Reconhecimento de Padrões

Exercício 08: Clustering

Aluno: Giovanni Martins de Sá Júnior – 2017001850

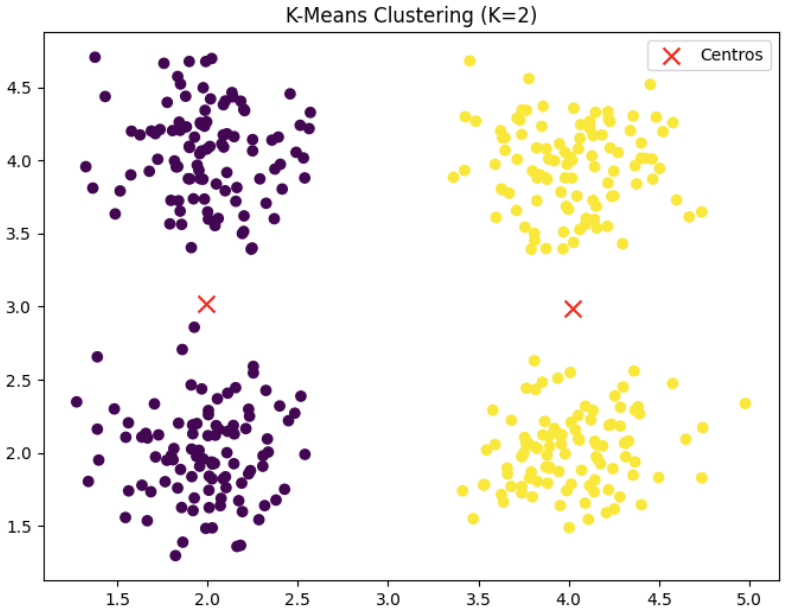
Parte 1: Algoritmo de K-médias

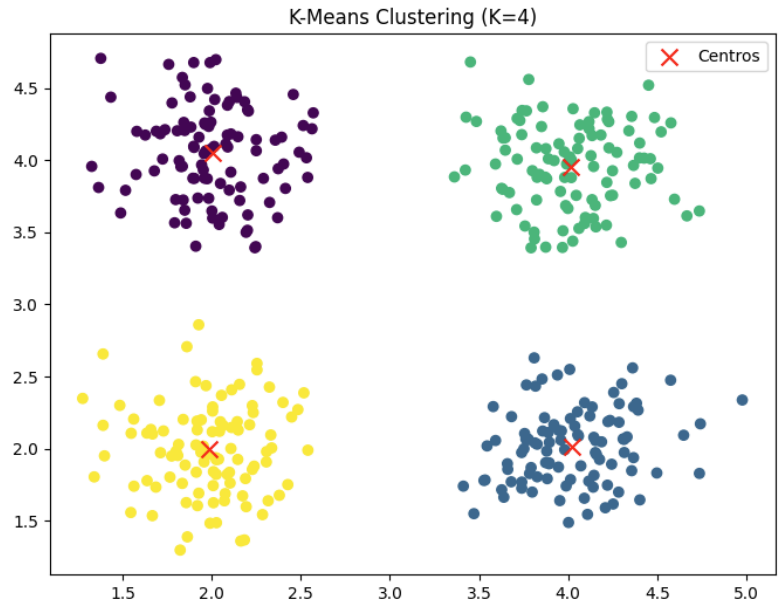
```
# Importação de Bibliotecas:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs


def k_means(data, k, max_iterations=100):

    # Passo 1: Definir o número k de clusters

    num_samples, num_features = data.shape

    # Passo 2: Escolher aleatoriamente k pontos de treinamento como centros
iniciais

    random_indices = np.random.choice(num_samples, k, replace=False)

    centers = data[random_indices]

    for _ in range(max_iterations):

        # Passo 3: Calcular a distância de todos os pontos para os centros
dos k-clusters

        distances = np.linalg.norm(data[:, np.newaxis] - centers, axis=2)

        # Atribuir cada ponto ao cluster mais próximo

        labels = np.argmin(distances, axis=1)

        # Passo 4: Calcular o novo centro dos clusters a partir da média dos
pontos

        new_centers = np.array([data[labels == i].mean(axis=0) for i in
range(k)])

        # Passo 5: Verificar se os centros convergiram

        if np.all(centers == new_centers):

            break

        else:

            centers = new_centers

    return centers, labels
```

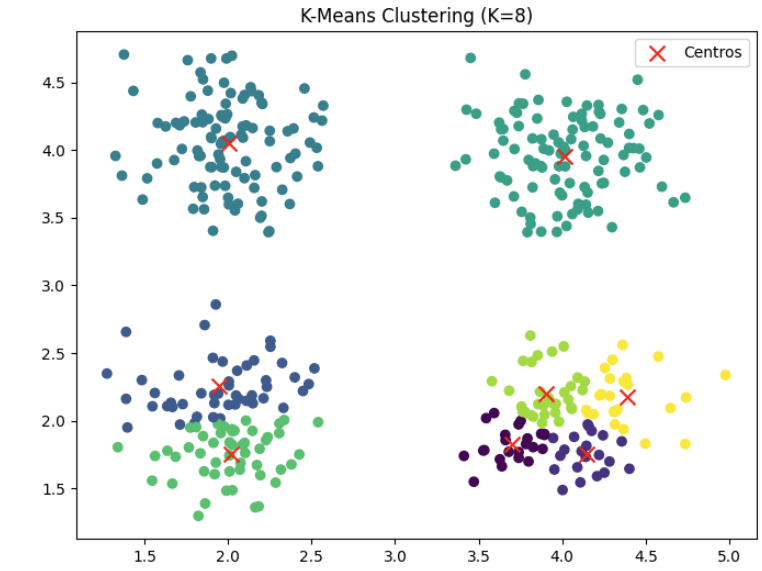
Parte 2: Aplicação do K-médias

Desvio Padrão = 0.3

```
# Função para criar uma distribuição gaussiana bidimensional com desvio
padrão (sd)

def create_gaussian_distribution(mean, sd, size):

    cov = np.array([[sd ** 2, 0], [0, sd ** 2]])

    return np.random.multivariate_normal(mean, cov, size)

# Criar quatro distribuições gaussianas

data1 = create_gaussian_distribution(mean=[2, 2], sd=0.3, size=100)

data2 = create_gaussian_distribution(mean=[4, 4], sd=0.3, size=100)

data3 = create_gaussian_distribution(mean=[2, 4], sd=0.3, size=100)

data4 = create_gaussian_distribution(mean=[4, 2], sd=0.3, size=100)

# Concatenar os dados

data = np.concatenate((data1, data2, data3, data4), axis=0)

# Valores de K a serem testados

k_values = [2, 4, 8]

# Executar o algoritmo de K-médias para diferentes valores de K

for k in k_values:

    centers, labels = k_means(data, k)

    # Plotar os grupos com cores diferentes

    plt.figure(figsize=(8, 6))

    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')

    plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=100,
label='Centros')

    plt.title(f'K-Means Clustering (K={k})')

    plt.legend()

    plt.show()
```

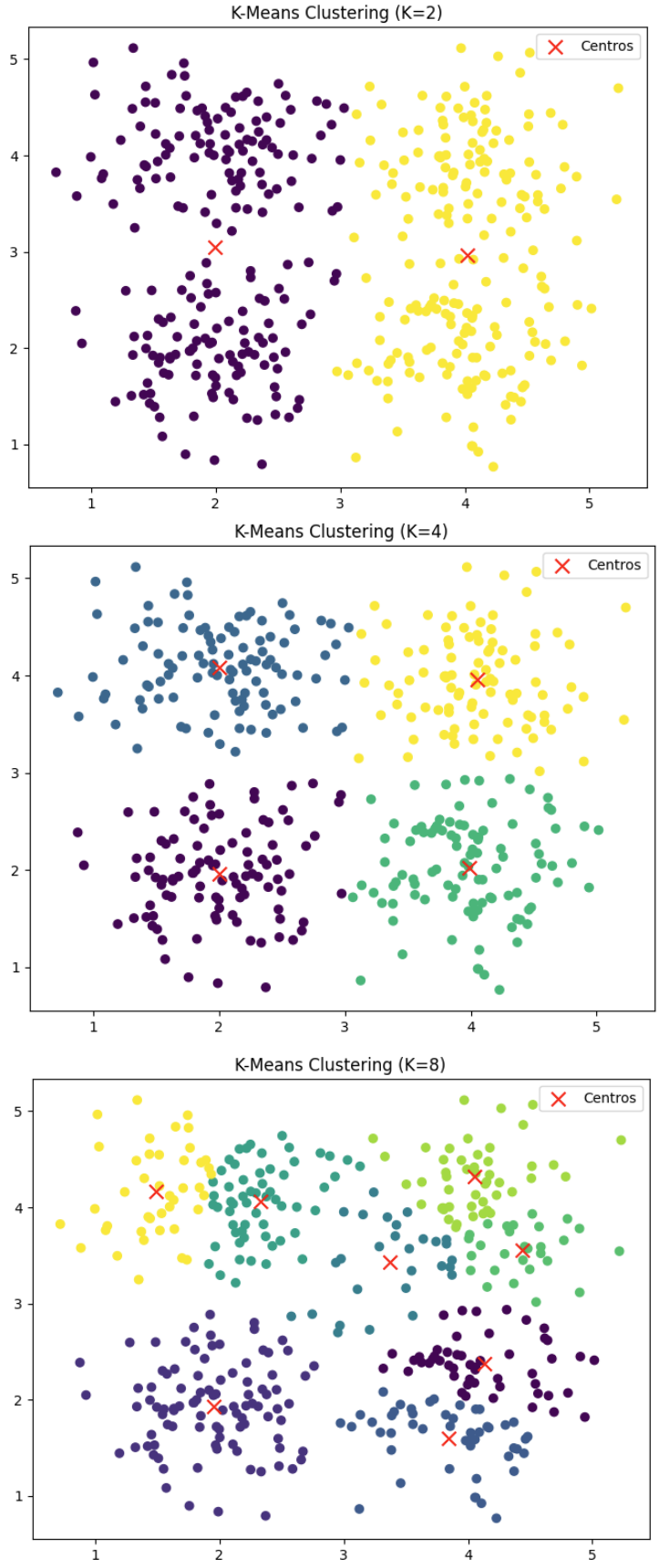

K-Means Clustering (K=2)
Centros
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


K-Means Clustering (K=4)
Centros
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


K-Means Clustering (K=8)
Centros
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

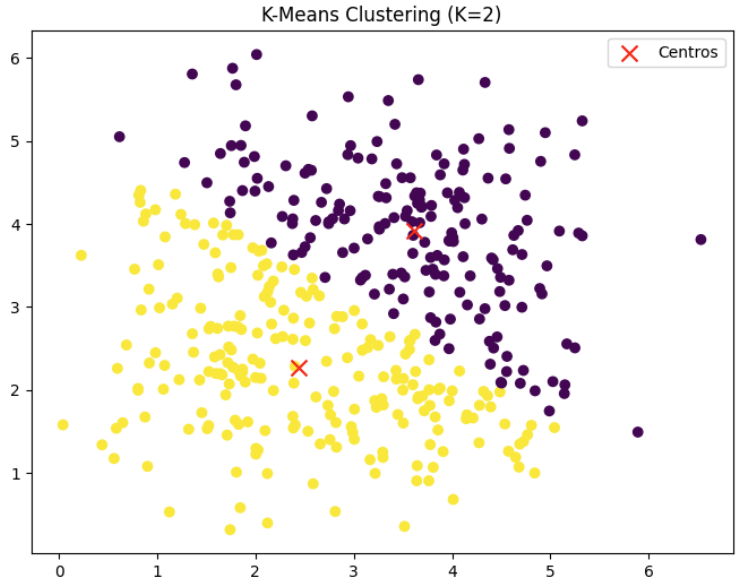
Desvio Padrão = 0.5

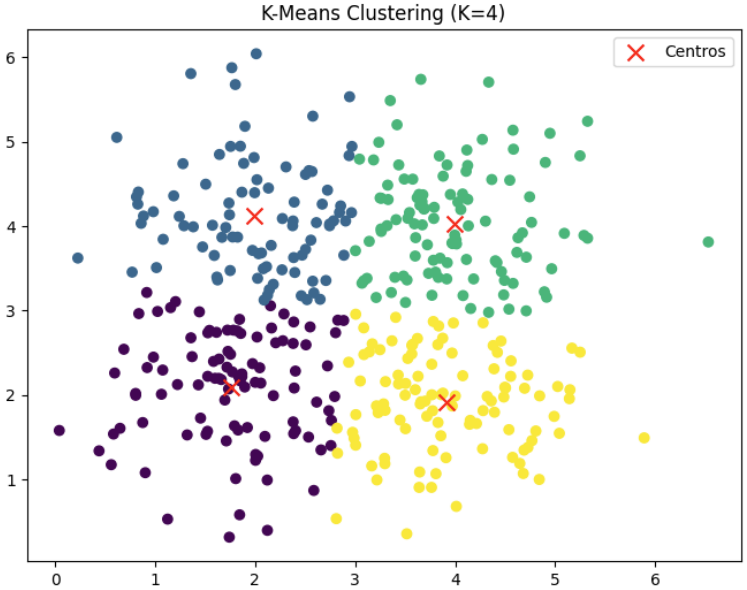
```
# Função para criar uma distribuição gaussiana bidimensional com desvio
padrão (sd)

def create_gaussian_distribution(mean, sd, size):

    cov = np.array([[sd ** 2, 0], [0, sd ** 2]])

    return np.random.multivariate_normal(mean, cov, size)

# Criar quatro distribuições gaussianas

data1 = create_gaussian_distribution(mean=[2, 2], sd=0.5, size=100)

data2 = create_gaussian_distribution(mean=[4, 4], sd=0.5, size=100)

data3 = create_gaussian_distribution(mean=[2, 4], sd=0.5, size=100)

data4 = create_gaussian_distribution(mean=[4, 2], sd=0.5, size=100)

# Concatenar os dados

data = np.concatenate((data1, data2, data3, data4), axis=0)

# Valores de K a serem testados

k_values = [2, 4, 8]

# Executar o algoritmo de K-médias para diferentes valores de K

for k in k_values:

    centers, labels = k_means(data, k)

    # Plotar os grupos com cores diferentes

    plt.figure(figsize=(8, 6))

    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')

    plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=100,
label='Centros')

    plt.title(f'K-Means Clustering (K={k})')

    plt.legend()

    plt.show()
```

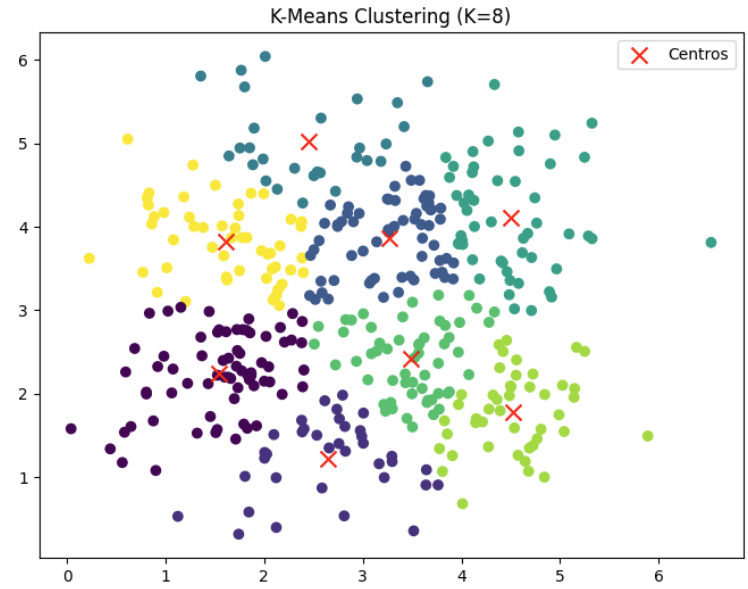

K-Means Clustering (K=2)
Centros
K-Means Clustering (K=4)
Centros
K-Means Clustering (K=8)
Centros

Desvio Padrão = 0.7

```
# Função para criar uma distribuição gaussiana bidimensional com desvio
padrão (sd)

def create_gaussian_distribution(mean, sd, size):

    cov = np.array([[sd ** 2, 0], [0, sd ** 2]])

    return np.random.multivariate_normal(mean, cov, size)

# Criar quatro distribuições gaussianas

data1 = create_gaussian_distribution(mean=[2, 2], sd=0.7, size=100)

data2 = create_gaussian_distribution(mean=[4, 4], sd=0.7, size=100)

data3 = create_gaussian_distribution(mean=[2, 4], sd=0.7, size=100)

data4 = create_gaussian_distribution(mean=[4, 2], sd=0.7, size=100)

# Concatenar os dados

data = np.concatenate((data1, data2, data3, data4), axis=0)

# Valores de K a serem testados

k_values = [2, 4, 8]

# Executar o algoritmo de K-médias para diferentes valores de K

for k in k_values:

    centers, labels = k_means(data, k)

    # Plotar os grupos com cores diferentes

    plt.figure(figsize=(8, 6))

    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')

    plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=100,
label='Centros')

    plt.title(f'K-Means Clustering (K={k})')

    plt.legend()

    plt.show()
```


K-Means Clustering (K=2)
Centros
0
1
2
3
4
5
6


K-Means Clustering (K=4)
Centros
0
1
2
3
4
5
6


K-Means Clustering (K=8)
Centros
0
1
2
3
4
5
6