ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - TIN HỌC

BTLT TUẦN 8: KHAI THÁC DỮ LIỆU

Sinh viên thực hiện: Nguyễn Công Hoài Nam
Mã số sinh viên: 21280099

Ngày 2 tháng 6 năm 2024

MỤC LỤC

1. Cài đặt thuật toán Mahalanobis k-means

Import các thư viện cần thiết

```
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns
```

Các bước để cài đặt thuật toán **Mahalanobis K-means**
Bước 1: Khởi tạo các phân cụm (clusters) và gán chúng ứng với mỗi điểm dữ liệu
- **Mahalanobis k-means** cần các điểm dữ liệu được gán với các centroid từ ban đầu bởi vì nó cần chúng để tính ma trận hiệp sai cho các clusters để tính khoảng cách Mahalanobis
- Ở đây em sử dụng khoảng cách Euclidean để khởi tạo (thuật toán k-means)

```
def initiate_centroids_clusters(k,dset, random_state = 234):
    km=KMeans(n_clusters = k,max_iter = 1,random_state = random_state)

    # Lấy iter = 1 , nghĩa là chỉ khởi tạo bước lặp đầu tiên bằng thuật toán kmeans
    km.fit(dset)
    clusters=km.labels_

    centroids=pd.DataFrame(km.cluster_centers_,columns=km.feature_names_in_)

    return clusters, centroids
```

Bước 2: tính khoảng cách Mahalanobis từ các centroids đến các điểm dữ liệu và gán mỗi điểm dữ liệu ứng với centroid gần nhất

- Hàm tính khoảng cách Mahalanobis

$$D_{Maha}(X, \mu_r, \Sigma_r) = \sqrt{(X - \mu_r)\Sigma^{-1}(X - \mu_r)^T}$$

- X: mỗi điểm dữ liệu trong dataset (các hàng trong df)
- μ_r: tâm cụm r trong danh sách tâm cụm
- Σ_r: ma trận hiệp phương sai ứng với mỗi tâm cụm r

```
def mahalanobis_error(X, mu, sigma):
    ''''
    D(X,mu,sigma)^2 = (X-mu) * Sigma^-1 * (X - mu).T
    '''
    sigma_inv = np.linalg.inv(sigma)
    return np.sqrt((X - mu) @ sigma_inv @ (X - mu).T)
```

- Để tính được khoảng cách Mahalanobis ta cần ma trận hiệp phương sai, dưới đây là hàm để tính:

```
def calculate_covariance_matrices(dset,k):
    '''
    Trả về các cov_matrix của các cụm
    '''
    cov_matrices = []
    for centroid in range(k):
        cov_matrix = np.cov(dset.loc[dset['centroid']==centroid,['x','y']],rowvar=False)
```

```
        cov_matrices.append(cov_matrix)
    return cov_matrices
```

- Tính lại ma trận hiệp phương sai và khoảng cách Mahalanobis của các centroids đến từng điểm dữ liệu - Gán lại phân cụm cho các điểm dữ liệu ứng với phân cụm gần nhất

```
def centroid_assignation(dset, centroids):
    # Số  centroid
    k = centroids.shape[0]

    # Số samples trong tập dữ liệu
    n = dset.shape[0]

    assignation = []

    assign_errors = []

    cov_matrixs =[]

    # Tính ma trận hiệp phương sai cho từng cụm
    cov_matrices = calculate_covariance_matrices(dset,k)

    for obs in range(n):
        # List error cho từng centroid
        all_errors = np.array([])

        # Tính error cho từng centroid
        for centroid in range(k):
            err = mahalanobis_error(centroids.iloc[centroid,:], dset.iloc[obs, :2],
                cov_matrices[centroid])
            all_errors = np.append(all_errors, err)
        # Chọn centroid gần nhất
        nearest_centroid = np.where(all_errors == np.amin(all_errors))[0].tolist()[0]
        nearest_centroid_error = np.amin(all_errors)

        assignation.append(nearest_centroid)
        assign_errors.append(nearest_centroid_error)

    return assignation, assign_errors
```

Bước 3: lặp lại bước hai cho đến khi các phân cụm không còn thay đổi
- Hàm chính của thuật toán,

```
def mahalanobis_kmeans(dset, k = 2, tol = 1e-4, plot = False):
    # Tạo bản sao để tránh thay đổi data gốc
    working_dset = dset.copy()

    # Danh sách error cho từng bước lặp
    err = []

    # Biến điều kiện để tiếp tục vòng lặp
    goahead = True

    # Biến đếm số lần lặp
    j = 0

    # Gán giá trị khởi tạo cho centroids
    working_dset['centroid'],centroids=initiate_centroids_clusters(k,working_dset)
```

```
    while goahead:
        # Gán centroid và tính error cho các quan sát
        working_dset['centroid'], j_err = centroid_assignation(working_dset, centroids)

        # Tính tổng error và vẽ đồ thị
        err.append(sum(j_err))

        # Tính các centroid mới dựa trên các quan sát đã gán nhãn
        centroids = working_dset.groupby('centroid').agg('mean').reset_index(drop=True)

        if j > 0:
            if err[j-1] - err[j] <= tol:
                goahead = False

        j += 1

    if plot:
        fig, ax = plt.subplots(1, 1, figsize=(12, 8))
        show_plot(ax,working_dset,centroids)

    return working_dset['centroid'], j_err, centroids
```

- Để thể hiện kết quả cuối cùng ta viết thêm hàm để trực quan hoá các phân cụm

```
def show_plot(ax, df, centroids, title="After Mahalanobis K-means"):
    k = centroids.shape[0]
    scatter_points = ax.scatter(df.iloc[:, 0], df.iloc[:, 1], marker='o', c=df['centroid'].
        astype('category'), cmap='viridis', s=80, alpha=0.5)
    scatter_centroids = ax.scatter(centroids.iloc[:, 0], centroids.iloc[:, 1], marker='X', s
        =250, c=[i for i in range(k)], cmap='viridis', edgecolors='deeppink')
    ax.set_xlabel('x', fontsize=14)
    ax.set_ylabel('y', fontsize=14)
    ax.set_title(title, fontsize=16)
    ax.tick_params(axis='both', which='major', labelsize=12)

    legend_labels = [f'Centroid {i+1}: ({centroids.iloc[i, 0]:.2f}, {centroids.iloc[i, 1]:.2f
        })' for i in range(k)]
    legend_elements = scatter_centroids.legend_elements()[0]
    ax.legend(legend_elements, legend_labels, title='Centroids', loc='upper left')
```

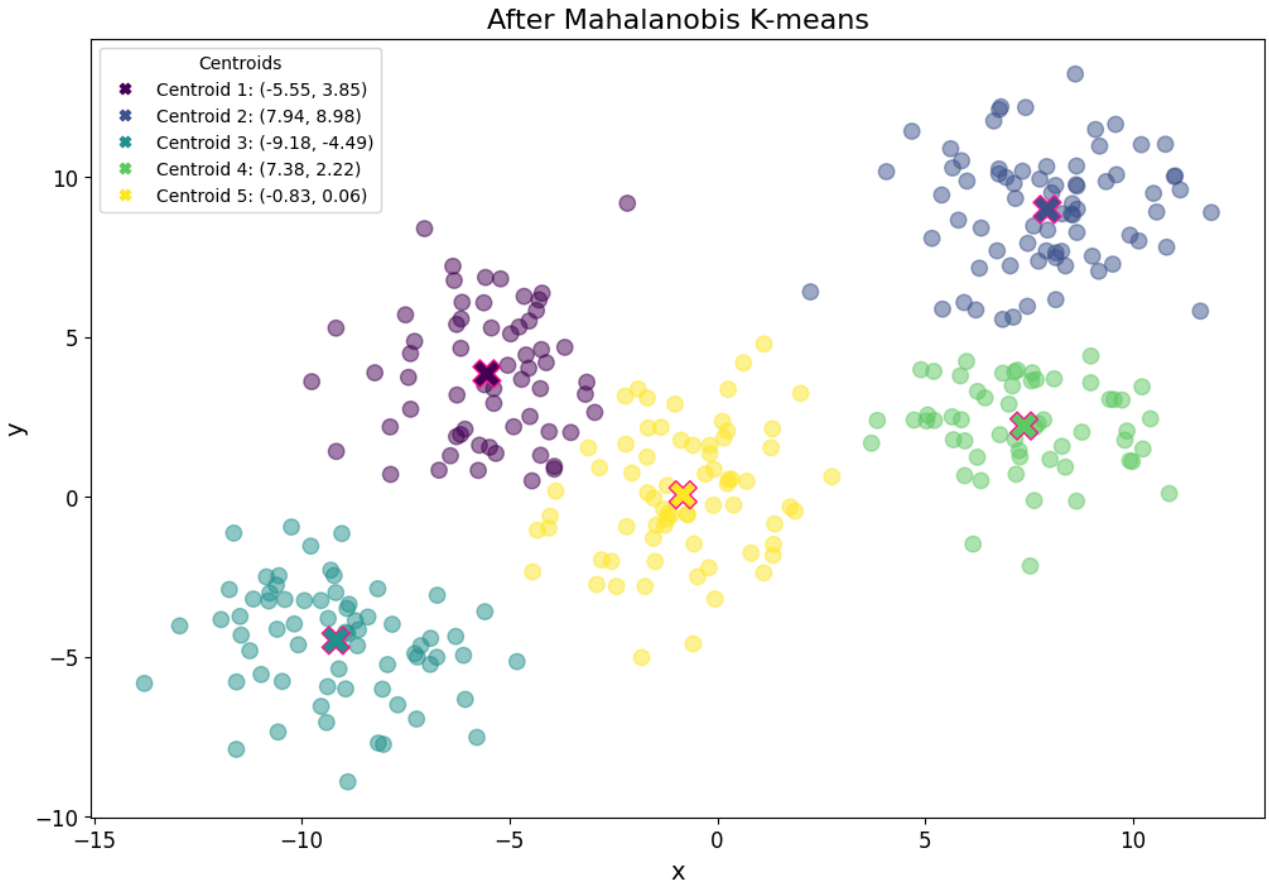
2. Kết quả

2.1. Kết quả thuật toán

Chạy thuật toán với data ban đầu (từ file pdf)

```
np.random.seed(234)
x,y=make_blobs(n_samples=330, centers=5,cluster_std=1.84)
df = pd.DataFrame({'x':x[:,0],'y':x[:,1]})
```

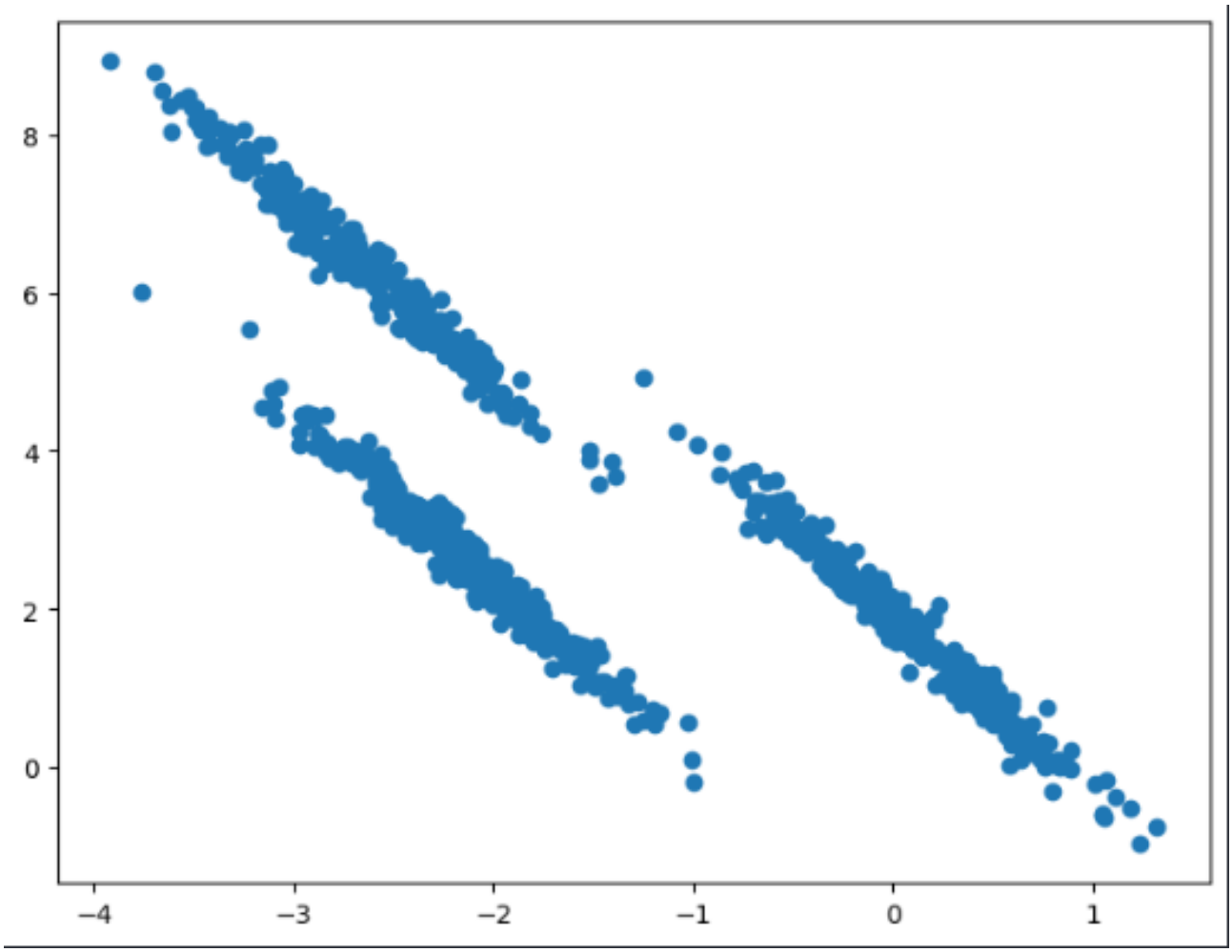
```
df['centroid'],df['error'],centroids=mahalanobis_kmeans(df,k=5, plot = True)
```



2.2. So sánh với GMM, K-means

Với những bộ dữ liệu có sự tương quan giữa các biến, việc tính toán chúng như những biến độc lập có thể dẫn đến sai số (thuật toán K-means)
- Do đó chúng ta cần những thuật toán quan tâm đến sự tương quan của các biến
- Đó chính là GMM và Mahanolobis K-means
Ta khởi tạo một bộ data có sự tương quan để thấy rõ sự khác biệt giữa các thuật toán

```
x, y = make_blobs(n_samples=1000, random_state=234,centers=3)
x = np.dot(x, [[0.2, -0.6], [-0.4, 0.8]])
df=pd.DataFrame({'x':x[:,0],'y':x[:,1]})

plt.figure(figsize=(8,6))
plt.scatter(x[:,0],x[:,1])
plt.show()
```



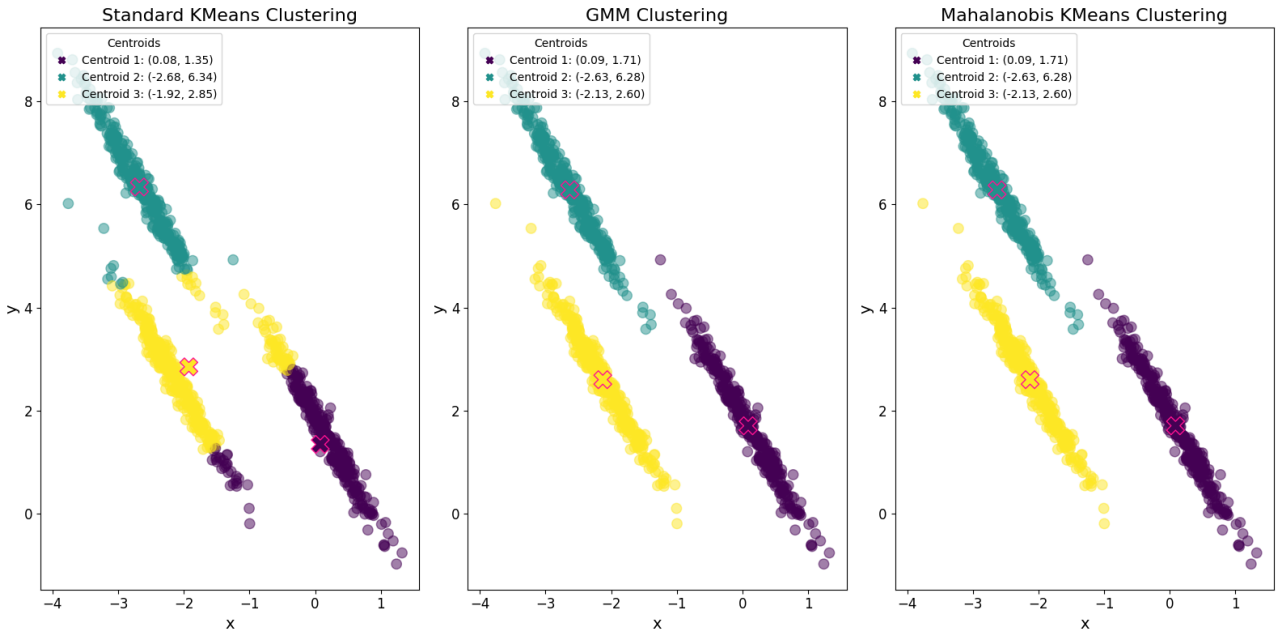
Tính toán GMM và K-means của thư viện Sklearn

```
# Apply standard KMeans
kmeans = KMeans(n_clusters=3, random_state=234)
df['centroid'] = kmeans.fit_predict(df[['x', 'y']])
centroids_kmeans = pd.DataFrame(kmeans.cluster_centers_, columns=['x', 'y'])

# Apply GMM
gmm = GaussianMixture(n_components=3, random_state=234)
df['centroid_gmm'] = gmm.fit_predict(df[['x', 'y']])
centroids_gmm = pd.DataFrame(gmm.means_, columns=['x', 'y'])
```

Vẽ biểu đồ trực quan kết quả

```
# Mahalanobis KMeans (assuming mahalanobis_kmeans is defined)
df['centroid_maha'], _, centroids_maha_kmeans = mahalanobis_kmeans(df[['x', 'y']], k=3, plot=
    False)

fig, axs = plt.subplots(1, 3, figsize=(16, 8))

# Plot for Standard KMeans
df['centroid'] = kmeans.fit_predict(df[['x', 'y']])
show_plot(axs[0], df[['x', 'y', 'centroid']], centroids_kmeans, title='Standard KMeans
    Clustering')

# Plot for GMM
```

```
df['centroid'] = df['centroid_gmm']  # Assign centroid_gmm to centroid for consistent plotting
show_plot(axs[1], df[['x', 'y', 'centroid']], centroids_gmm, title='GMM Clustering')

# Plot for Mahalanobis KMeans
df['centroid'] = df['centroid_maha']  # Assign centroid_maha to centroid for consistent plotting
show_plot(axs[2], df[['x', 'y', 'centroid']], centroids_maha_kmeans, title='Mahalanobis KMeans Clustering')

plt.tight_layout()
plt.show()
```



Có thể thấy GMM và Mahalanobis phân cụm tốt hơn nhiều so với K-means

3. Kết luận

Khi dữ liệu có tương quan giữa các biến, các phương pháp phân cụm như GMM (Gaussian Mixture Models) và Mahalanobis K-means thường vượt trội so với thuật toán K-means truyền thống.

GMM có khả năng mô hình hóa các cụm với hình dạng và kích thước khác nhau nhờ việc sử dụng các phân phối Gaussian. Hơn nữa, GMM có thể tính toán và mô hình hóa ma trận hiệp phương sai của dữ liệu, do đó có thể phát hiện các cấu trúc tương quan giữa các biến. Điều này cho phép GMM phân chia dữ liệu thành các cụm phức tạp hơn so với K-means.

Mahalanobis K-means sử dụng khoảng cách Mahalanobis tính đến cả phương sai và tương quan của các biến, do đó có thể phát hiện các cụm có hình dạng và kích thước khác nhau một cách hiệu quả hơn.

Tuy nhiên, với dữ liệu có các biến độc lập, K-means vẫn là sự lựa chọn hợp lý nhờ sự đơn giản và hiệu quả của thuật toán.

Vì vậy, việc lựa chọn thuật toán phân cụm phụ thuộc vào đặc điểm của dữ liệu, cần phải đánh giá ưu và nhược điểm của từng phương pháp để quyết định sử dụng phù hợp.