

Bài tập thực hành-Khai thác dữ liệu-tuần 6

Phan Hồng Trâm - 21110414

May 2024

Mục lục

1	Trình bày tóm tắt lại phần code trong file báo cáo	2
---	--	---

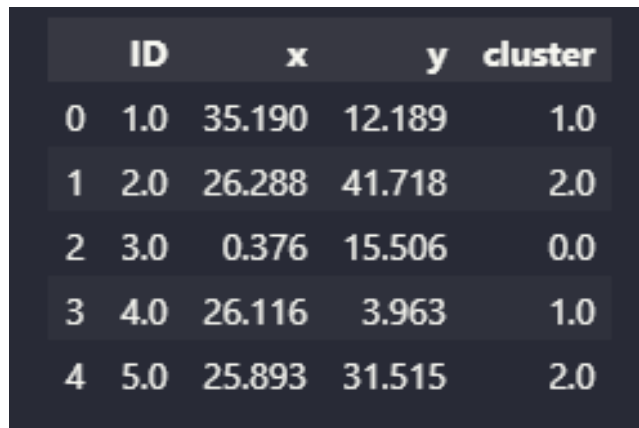
1 Trình bày tóm tắt lại phần code trong file báo cáo

Ta import thư viện:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

- Import thư viện và load data:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
blobs = pd.read_csv('data.csv')
colnames= list(blobs.columns[1:-1])
blobs = blobs.head(59)
blobs.head()
```



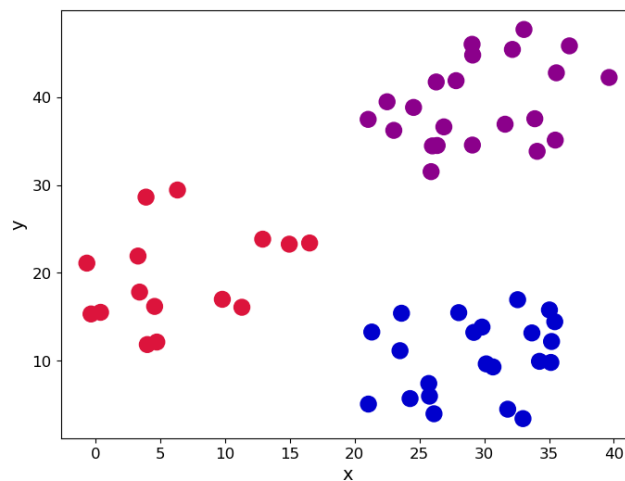
	ID	x	y	cluster
0	1.0	35.190	12.189	1.0
1	2.0	26.288	41.718	2.0
2	3.0	0.376	15.506	0.0
3	4.0	26.116	3.963	1.0
4	5.0	25.893	31.515	2.0

Hình 1: In ra 5 dòng đầu của dữ liệu

- Thiết lập trục quan và biểu đồ phân tán ban đầu:

```
customcmap = ListedColormap(['crimson', 'mediumblue', 'darkmagenta'])
fig, ax = plt.subplots(figsize= (8,6))
plt.scatter(x= blobs['x'], y= blobs['y'], s= 150,
            c= blobs['cluster'].astype('category'),
            cmap = customcmap)
ax.set_xlabel(r'x', fontsize= 14)
ax.set_ylabel(r'y', fontsize= 14)
plt.xticks(fontsize= 12)
plt.yticks(fontsize= 12)
plt.show()
```

- Bản đồ màu có tên customcmap được tạo bằng ListedColormap. Điều này xác định màu sắc được sử dụng để thể hiện các cụm khác nhau trong biểu đồ phân tán.



Hình 2: Biểu đồ phân tán ban đầu

- Bước 1 và 2: Xác định k và khởi tạo các tâm:

```
def initiate_centroids(k, dset):
    """
    Select k data points as centroids
    k: number of centroids
    dset: pandas dataframe
    """
    centroids = dset.sample(k)
    return centroids

np.random.seed(42)
k = 3
df = blobs[['x', 'y']]
centroids = initiate_centroids(k, df)
centroids
```

- Lấy ngẫu nhiên k điểm dữ liệu từ dset bằng `sample` và trả về chúng dưới dạng các centroid ban đầu.

	x	y
0	35.190	12.189
5	23.606	15.402
34	23.492	11.142

- Bước 3: Tính khoảng cách

```
def rsserr(a,b):
    """
    Calculate sum of square errors
    a and b are numpy arrays
    """
    return np.sum(np.square(a-b))

for i, centroid in enumerate(range(centroids.shape[0])):
    err = rsserr(centroids.iloc[centroid,:], df.iloc[36,:])
    print('Error for centroid {0}: {1:.2f}'.format(i, err))
```

- Hàm `rsserr`: tính tổng sai số bằng khoảng cách Manhattan giữa hai mảng NumPy (a và b).
- Sau đó ta dùng vòng lặp qua các centroid (indexed bởi i). Đối với mỗi centroid, hàm `rsserr` tính toán sai số giữa centroid đó và một điểm dữ liệu cụ thể rồi in ra kết quả.

```
Error for centroid 0: 28.31
Error for centroid 1: 29.51
Error for centroid 2: 33.88
```

- Bước 4: gán giá trị các tâm

```
def centroid_assignment(dset, centroids):
    """
    Given a dataframe 'dset' and a set of 'centroids', we assign each
    data point in 'dset' to a centroid.
    - dset - pandas dataframe with observations
    - centroids - pandas dataframe with centroids
    """
    k = centroids.shape[0]
    n = dset.shape[0]
    assignation = []
    assign_errors = []
    for obs in range(n):
        # Estimate error
        all_errors = np.array([])
        for centroid in range(k):
            err = rsserr(centroids.iloc[centroid,:], dset.iloc[obs,:])
            all_errors = np.append(all_errors, err)
        # Get the nearest centroid and the error
        nearest_centroid = np.where(all_errors==np.amin(all_errors))[0].tolist()[0]
        nearest_centroid_error = np.amin(all_errors)
        # Add values to corresponding list
        assignation.append(nearest_centroid)
        assign_errors.append(nearest_centroid_error)
    return assignation, assign_errors
```

- Hàm `centroid_assignment` thực hiện việc gán các điểm dữ liệu vào cụm gần nhất.
- Đầu vào:
 - * `dset`: Một DataFrame chứa các điểm dữ liệu.

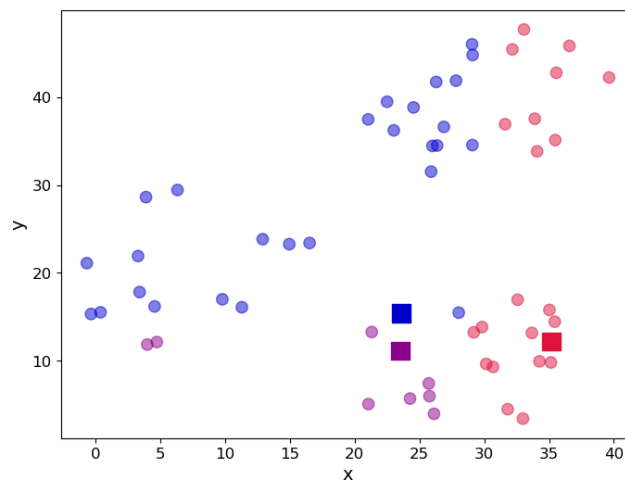
- * `centroids`: Một DataFrame chứa các tâm cụm (centroids).
 - Sau đó ta tính toán số lượng cụm (k) và số điểm dữ liệu (n) bằng cách lấy số lượng dòng của `centroids` để xác định k (cụm) và số lượng dòng của `dset` để xác định n (số điểm dữ liệu).
 - Ta khởi tạo 2 danh sách rỗng là `assignment` để lưu trữ nhãn cụm được gán cho mỗi điểm dữ liệu và `assign_errors` để lưu trữ lỗi (khoảng cách) của mỗi điểm dữ liệu với tâm cụm gần nhất.
 - Lặp qua từng điểm dữ liệu (`obs`) trong `dset`:
 - * khởi tạo mảng `all_errors` lưu trữ sai số ước lượng (Estimate error)
 - * Tính toán lỗi với tất cả các tâm cụm bằng cách sử dụng hàm `rsserr` để tính toán lỗi (khoảng cách Manhattan) giữa điểm dữ liệu `obs` với từng tâm cụm trong `centroids`. Kết quả tính toán lỗi được thêm vào danh sách `all_errors`.
 - Sử dụng `np.where` và `np.amin` để tìm ra chỉ số của tâm cụm có lỗi nhỏ nhất (gần nhất) với điểm dữ liệu `obs`. Chỉ số này chính là nhãn cụm được gán cho điểm dữ liệu `obs`. Trích xuất giá trị lỗi nhỏ nhất (gần nhất) với điểm dữ liệu `obs`.
 - Thêm chỉ số tâm cụm gần nhất vào danh sách `assignment`. Thêm giá trị lỗi nhỏ nhất vào danh sách `assign_errors`.
 - Trả về 2 danh sách `assignment` và `assign_errors`.
- Thêm cột gán tâm và sai số phát sinh để cập nhật biểu đồ phân tán biểu diễn các trọng tâm:

```
df['centroid'], df['error'] = centroid_assignment(df, centroids)
df.head()
```

	x	y	centroid	error
0	35.190	12.189	0	0.000
1	26.288	41.718	1	28.998
2	0.376	15.506	1	23.334
3	26.116	3.963	2	9.803
4	25.893	31.515	1	18.400

- Sau đó ta plot lại biểu đồ với centroids:

```
fig, ax = plt.subplots(figsize= (8,6))
plt.scatter(df.iloc[:,0], df.iloc[:,1], marker= 'o',
            c= df['centroid'].astype('category'),
            cmap= customcmap, s= 80, alpha= 0.5)
plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],
            marker= 's', s= 200, c= [0, 1, 2],
            cmap = customcmap)
ax.set_xlabel(r'x', fontsize= 14)
ax.set_ylabel(r'y', fontsize= 14)
plt.xticks(fontsize= 12)
plt.yticks(fontsize= 12)
plt.show()
```



Hình 3: Biểu đồ phân tán với các centroids

- Ta tính tổng sai số:

```
print('The total error is {0:.2f}'.format(df['error'].sum()))
```

The total error is 1061.82

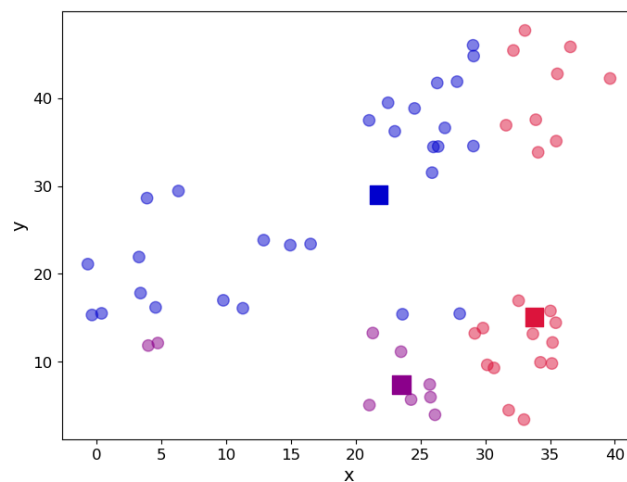
- Bước 5: cập nhật vị trí của k tâm bằng việc tính giá trị trung bình của các quan sát được gán cho mỗi tâm

```
centroids = df.groupby(['centroid']).agg('median').loc[:, colnames].reset_index(drop= True)
centroids
```

	x	y
0	33.7860	15.109
1	21.7635	29.021
2	23.4920	7.409

- Xem lại biểu đồ phân tán với vị trí của k tâm đã được cập nhật:

```
fig, ax = plt.subplots(figsize= (8,6))
plt.scatter(df.iloc[:,0], df.iloc[:,1], marker= 'o',
            c= df['centroid'].astype('category'),
            cmap= customcmap, s= 80, alpha= 0.5)
plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],
            marker= 's', s= 200,
            c= [0,1,2], cmap= customcmap)
ax.set_xlabel(r'x', fontsize= 14)
ax.set_ylabel(r'y', fontsize= 14)
plt.xticks(fontsize= 12)
plt.yticks(fontsize= 12)
plt.show()
```



- Bước 6: Lặp lại bước 3-5

```
def kmedians(dset, k = 2, tol = 1e-4):
    '''
    K-medians implementation for a
    'dset': Dataframe with observations
    'k': number of clusters, default = 2
    'tol': tolerance= 1E-4
    '''
    # Let us work in a copy, so we don't mess the original
    working_dset= dset.copy()
    # We define some variables to hold the error, the
    # stopping signal and a counter for the iterations
    err = []
    goahead = True
    j = 0

    # Step 2: Initiate clusters by defining centroids
    centroids = initiate_centroids(k, dset)

    while goahead:
        # Step 3+4: Assign centroids and calculate error
        working_dset['centroid'], j_err= centroid_assignment(working_dset, centroids)
        err.append(sum(j_err))

        # Step 5: Update centroid position
        centroids = working_dset.groupby('centroid').agg('median').reset_index(drop= True)
```

```
# Step 6: Restart the iteration
if j>0:
    # Is the error less than a tolerance (1E-4)
    if err[j-1]-err[j] <= tol:
        goahead = False

    j +=1

working_dset['centroid'], j_err = centroid_assignment(working_dset, centroids)
centroids = working_dset.groupby('centroid').agg('median').reset_index(drop = True)
return working_dset['centroid'], j_err, centroids
```

- Hàm `kmedians` nhận tham số đầu vào là DataFrame `dset`, `k` là số lượng cụm (mặc định là 2), `tol` là ngưỡng sai số (mặc định là `1e-4`).
- Hàm tạo một bản sao của DataFrame `dset` để tránh làm thay đổi dữ liệu gốc. Bản sao này được lưu trữ trong biến `working_dset`.
- Khởi tạo các biến:
 - * `err`: Danh sách rỗng để lưu trữ lỗi tổng thể sau mỗi lần lặp.
 - * `goahead`: Biến cờ để điều khiển vòng lặp (mặc định là `True`).
 - * `j`: Biến đếm số lần lặp.
- Hàm sử dụng hàm `intiate_centroids` để khởi tạo các tâm cụm ban đầu cho K-medians. Số lượng tâm cụm được lấy từ biến `k`. Các tâm cụm ban đầu được lưu trữ trong biến `centroids`.
- Vào vòng lặp:
 - * sử dụng hàm `centroid_assignment` để gán nhãn cụm cho từng điểm dữ liệu trong `working_dset`. Nhãn cụm được xác định dựa trên tâm cụm gần nhất (khoảng cách Manhattan nhỏ nhất) với điểm dữ liệu.
 - * Kết quả gán nhãn cụm được lưu trữ trong cột `centroid` của `working_dset`.
 - * Lỗi (khoảng cách) của mỗi điểm dữ liệu với tâm cụm gần nhất được lưu trữ trong danh sách `j_err`.
 - * Sau đó cập nhật tâm cụm bằng cách tính toán giá trị trung bình (median) của các thuộc tính cho mỗi cụm. Giá trị trung bình này được sử dụng để cập nhật vị trí của các tâm cụm. Các tâm cụm cập nhật được lưu trữ trong biến `centroids`.
 - * Ta kiểm tra điều kiện dừng: Hàm so sánh lỗi tổng thể hiện tại `err[j]` với lỗi tổng thể trước đó `err[j-1]`. Nếu sự khác biệt giữa hai lỗi nhỏ hơn hoặc bằng `tol`, biến cờ `goahead` được đặt thành `False` để thoát khỏi vòng lặp.
 - * Biến đếm `j` được tăng lên để theo dõi số lần lặp.
- Đầu ra là một tuple chứa ba yếu tố:
 - * `working_dset['centroid']`: Danh sách chứa nhãn cụm được gán cho từng điểm dữ liệu.
 - * `j_err`: Danh sách chứa lỗi (khoảng cách) của mỗi điểm dữ liệu với tâm cụm gần nhất.
 - * `centroids`: Một DataFrame chứa các tâm cụm (centroids) được cập nhật sau mỗi lần lặp.

- Thực thi hàm trên:

```
np.random.seed(42)
df['centroid'], df['error'], centroids = kmedians(df[['x','y']],3)
df.head()
```

	x	y	centroid	error
0	35.190	12.189	0	6.8565
1	26.288	41.718	1	6.3200
2	0.376	15.506	2	6.4780
3	26.116	3.963	0	10.4435
4	25.893	31.515	1	9.8390

- In ra vị trí của các tâm cuối cùng:

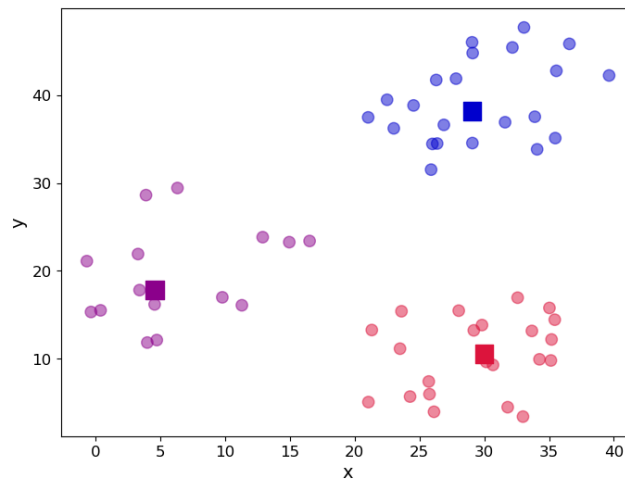
```
centroids
```

	x	y
0	29.9860	10.5365
1	29.0685	38.1785
2	4.5500	17.8100

Hình 4: Vị trí của tâm cuối cùng

- Xem lại biểu đồ phân tán:

```
fig, ax = plt.subplots(figsize= (8,6))
plt.scatter(df.iloc[:,0], df.iloc[:,1], marker= 'o',
            c= df['centroid'].astype('category'),
            cmap= customcmap, s= 80, alpha= 0.5)
plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],
            marker = 's', s= 200,
            c= [0,1,2], cmap= customcmap)
ax.set_xlabel(r'x', fontsize= 14)
ax.set_ylabel(r'y', fontsize= 14)
plt.xticks(fontsize= 12)
plt.yticks(fontsize= 12)
plt.show()
```



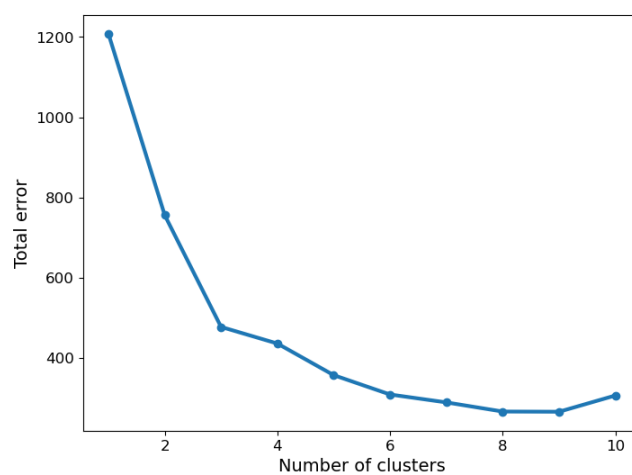
Hình 5: Biểu đồ phân tán sau khi đã cập nhật tâm cụm

- Sử dụng “elbow” để chỉ ra số cụm tối ưu:

```
err_total = []
n = 10
df_elbow = blobs[['x', 'y']]
for i in range(n):
    _, my_errs, _ = kmedians(df_elbow, i+1)
    err_total.append(sum(my_errs))

fig, ax = plt.subplots(figsize= (8,6))
plt.plot(range(1, n+1), err_total, linewidth= 3, marker = 'o')
ax.set_xlabel(r'Number of clusters', fontsize= 14)
ax.set_ylabel(r'Total error', fontsize= 14)
plt.xticks(fontsize= 12)
plt.yticks(fontsize=12)
plt.show()
```

- `err_total`: Danh sách rỗng để lưu trữ tổng lỗi theo số cụm.
- `n`: Số lượng lần lặp để thử nghiệm với các giá trị `k` (số cụm) khác nhau (được gán giá trị 10 trong đoạn code trên).
- `df_elbow`: Giả sử đây là DataFrame chứa các điểm dữ liệu cần phân cụm (chỉ lấy các cột 'x' và 'y').
- Ta lặp qua `n` lần, thử nghiệm với `n` giá trị `k` (số cụm) khác nhau:
 - * Sử dụng hàm `kmedians` để thực hiện thuật toán K-medians trên `df_elbow` với `i+1` cụm.
 - * Kết quả trả về của `kmedians` được lưu trữ trong ba biến: `_` (bỏ qua), `my_errs` (danh sách chứa lỗi (khoảng cách) của mỗi điểm dữ liệu với tâm cụm gần nhất), và `_` (bỏ qua).
 - * Sử dụng hàm `sum` để tính tổng các giá trị lỗi trong danh sách `my_errs` và thêm tổng lỗi này vào danh sách `err_total`.
- **Phân tích**: Biểu đồ Elbow Method sẽ là một đường cong giảm dần. Giá trị `k` (số cụm) được chọn là giá trị `k` gần điểm điểm chỗ của đường cong. Điểm chỗ là điểm mà đường cong bắt đầu giảm dần chậm lại, nghĩa là việc tăng thêm số cụm không mang lại cải thiện đáng kể về việc giảm lỗi. Dựa vào hình ta có thể thấy số `k` cụm tối ưu là 3.



Hình 6: Biểu đồ elbow thể hiện số cụm tối ưu