1. PHẦN MÔ HÌNH K- TRUNG BÌNH (K-MEANS MODEL)

1.1. Xây dựng chương trình từ numpy

Ví dụ 1. Nhắc lại ví dụ ở bài lý thuyết:

Chúng ta sẽ xét ví dụ nhân tạo với đầu vào là 3 tập ngẫu nhiên, mỗi tập N điểm được khởi tạo theo phân bố Gaussian (phân bố chuẩn) có 3 kỳ vọng (tâm cụm) xác định trước là means[1,:], means[2,:] và means[3,:]; có chung ma trận hiệp phương sai là cov[].

Trong đoạn mã gọi thư viện dưới đây, trước tiên, chúng ta cần numpy và matplotlib như trong các bài trước cho việc tính toán ma trận và hiển thị dữ liệu. Chúng ta cũng cần thêm thư viện scipy.spatial.distance để tính khoảng cách giữa các cặp điểm trong hai tập hợp một cách hiệu quả. Sau đó chúng ta tạo dữ liệu theo ý tưởng ở trên. Đoạn mã lệnh như dưới đây

```
# Gọi các thư viện cần thiết
# Ta tự xây dựng phần k-means nên sẽ không gọi sklearn

from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(11)

# Kỳ vọng và hiệp phương sai của 3 cụm dữ liệu
means = [[2, 2], [8, 3], [3, 6]]
cov = [[1, 0], [0, 1]]

# Số điểm mỗi cụm dữ liệu
N = 500

# Tạo các cụm dữ liệu qua phân bố chuẩn (Gaussian)
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)
# Tổng hợp dữ liệu từ các cụm
X = np.concatenate((X0, X1, X2), axis = 0)

# Số cụm = 3
K = 3

# Gán nhãn ban đầu cho các cụm, sau đó ta test model và so sánh
original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

Phương thức để hiển thị dữ liệu X lên mặt phẳng, ở đây sử dụng thông tin nhãn đã được gán ở phần trước trong tham đối label

```
def kmeans_display(X, label):
    K = np.amax(label) + 1
    X0 = X[label == 0, :]
    X1 = X[label == 1, :]
    X2 = X[label == 2, :]

    plt.plot(X0[:, 0], X0[:, 1], 'b^', markersize = 4, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'go', markersize = 4, alpha = .8)
    plt.plot(X2[:, 0], X2[:, 1], 'rs', markersize = 4, alpha = .8)

    plt.axis('equal')
    plt.plot()
```

```
    plt.show()
```

Dưới đây ta sẽ xây dựng một số hàm cần thiết cho mô hình K-means clustering

- kmeans_init_centers để khởi tạo các centers ban đầu.
- kmeans_asign_labels để gán nhán mới cho các điểm khi biết các centers.
- kmeans_update_centers để cập nhật các centers mới dữa trên dữ liệu vừa được gán nhãn.
- has_converged để kiểm tra điều kiện dừng của thuật toán.

Khởi tạo một bộ tâm cụm trên dữ liệu X với giả thiết có k cụm

```
def kmeans_init_centers(X, k):
    # randomly pick k rows of X as initial centers
    return X[np.random.choice(X.shape[0], k, replace=False)]
```

Ở đây chúng ta chọn các tâm cụm một cách ngẫu nhiên (miễn là các tâm cụm khác nhau). Các bạn hãy thử điều chỉnh bằng cách chọn K điểm xa nhau nhất theo phương pháp đã được trình bày trong phần lý thuyết.

Phương thức để gán cụm cho một điểm dữ liệu bằng cách tính khoảng cách từ điểm đó đến các tâm cụm, khoảng cách đến đâu ngắn nhất thì ta coi điểm hiện tại sẽ thuộc về cụm đó.

```
def kmeans_assign_labels(X, centers):
    # calculate pairwise distances btw data and centers
    D = cdist(X, centers)
    # return index of the closest center
    return np.argmin(D, axis = 1)
```

Phương thức để cập nhật lại tâm cụm sau mỗi bước lặp: Tâm cụm mới sẽ là trung bình cộng (theo tọa độ) của tất cả các điểm có trong cụm)

```
def kmeans_update_centers(X, labels, K):
    centers = np.zeros((K, X.shape[1]))
    for k in range(K):
        # collect all points assigned to the k-th cluster
        Xk = X[labels == k, :]
        # take average
        centers[k,:] = np.mean(Xk, axis = 0)
    return centers
```

Hàm để kiểm tra xem thuật toán có thực sự chạy (có hội tụ hay không) thông qua việc tâm cụm sau mỗi bước lặp có thay đổi hay không? Nếu tâm cụm không đổi nghĩa là thuật toán đã dừng (hội tụ) – tức là cần trả về TRUE. Kiểm tra cho tất cả các tâm cụm

```
def has_converged(centers, new_centers):
    # return True if two sets of centers are the same
    return (set([tuple(a) for a in centers]) ==
        set([tuple(a) for a in new_centers]))
```

Vòng lặp để thực hiện tất cả các bước trong thuật toán k-means cho đến khi thuật toán dừng

```
def kmeans(X, K):
    centers = [kmeans_init_centers(X, K)]
    labels = []
    it = 0
    while True:
        labels.append(kmeans_assign_labels(X, centers[-1]))
        new_centers = kmeans_update_centers(X, labels[-1], K)
        if has_converged(centers[-1], new_centers):
            break
```

```
        centers.append(new_centers)
        it += 1
    return (centers, labels, it)
```

Gọi và thực hiện phương pháp

```
(centers, labels, it) = kmeans(X, K)
print('Centers found by our algorithm:')
print(centers[-1])

kmeans_display(X, labels[-1])
```

1.2. Thực hiện bằng thư viện sklearn.

Để có thể đối sánh, chúng ta hãy thực hiện và hiển thị kết quả thu được bằng cách sử dụng thư viện scikit-learn.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0).fit(X)

print('Centers found by scikit-learn:')

print(kmeans.cluster_centers_)

pred_label = kmeans.predict(X)

kmeans_display(X, pred_label)
```

1.3.Ví dụ 2: Thực hiện phân cụm cho bộ dữ liệu chữ số viết tay

Hãy tìm lại phần thực hành đọc dữ liệu từ tệp chứa thông tin hình ảnh các chữ số viết tay và thực hiện việc phân cụm dữ liệu vào 10 cụm (10 chữ số viết tay) theo cách sau: Đọc 5000 mẫu dữ liệu từ phần training, sau đó thực hiện phân cụm bằng phương pháp k-means, tiếp theo hãy kiểm tra xem trong mỗi cụm, tỷ lệ có nhãn nào (từ 0 đến 9) là cao nhất. Sau đó đếm và in ra tỷ lệ các mẫu không thuộc nhãn đó nhưng được phân vào cùng một cụm với nhãn. Trong phần này ta sẽ sử dụng thư viện.

Phần import các thư viện cần thiết

```
import numpy as np
from mnist import MNIST
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import normalize
```

Xây dựng phương thức để hiện thị dữ liệu:

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


# This function visualizes filters in matrix A. Each column of A is a
# filter. We will reshape each column into a square image and visualizes
# on each cell of the visualization panel.
# All other parameters are optional, usually you do not need to worry
# about it.
# opt_normalize: whether we need to normalize the filter so that all of
# them can have similar contrast. Default value is true.
```

```
# opt_graycolor: whether we use gray as the heat map. Default is true.
# opt_colmajor: you can switch convention to row major for A. In that
# case, each row of A is a filter. Default value is false.
# source: https://github.com/tsaith/ufldl_tutorial

def display_network(A, m = -1, n = -1):
    opt_normalize = True
    opt_graycolor = True

    # Rescale
    A = A - np.average(A)

    # Compute rows & cols
    (row, col) = A.shape
    sz = int(np.ceil(np.sqrt(row)))
    buf = 1
    if m < 0 or n < 0:
        n = np.ceil(np.sqrt(col))
        m = np.ceil(col / n)


    image = np.ones(shape=(buf + m * (sz + buf), buf + n * (sz + buf)))

    if not opt_graycolor:
        image *= 0.1

    k = 0

    for i in range(int(m)):
        for j in range(int(n)):
            if k >= col:
                continue

            clim = np.max(np.abs(A[:, k]))

            if opt_normalize:
                image[buf + i * (sz + buf):buf + i * (sz + buf) + sz, buf + j * (sz +
buf):buf + j * (sz + buf) + sz] = \
                    A[:, k].reshape(sz, sz) / clim
            else:
                image[buf + i * (sz + buf):buf + i * (sz + buf) + sz, buf + j * (sz +
buf):buf + j * (sz + buf) + sz] = \
                    A[:, k].reshape(sz, sz) / np.max(np.abs(A))
            k += 1

    return image

def display_color_network(A):
    """
    # display receptive field(s) or basis vector(s) for image patches
    #
    # A         the basis, with patches as column vectors
    # In case the midpoint is not set at 0, we shift it dynamically
    :param A:
    :param file:
    :return:
    """
    if np.min(A) >= 0:
        A = A - np.mean(A)

    cols = np.round(np.sqrt(A.shape[1]))

    channel_size = A.shape[0] / 3
    dim = np.sqrt(channel_size)
    dimp = dim + 1
```

```
    rows = np.ceil(A.shape[1] / cols)

    B = A[0:channel_size, :]
    C = A[channel_size:2 * channel_size, :]
    D = A[2 * channel_size:3 * channel_size, :]

    B = B / np.max(np.abs(B))
    C = C / np.max(np.abs(C))
    D = D / np.max(np.abs(D))

    # Initialization of the image
    image = np.ones(shape=(dim * rows + rows - 1, dim * cols + cols - 1, 3))

    for i in range(int(rows)):
        for j in range(int(cols)):
            # This sets the patch
            image[i * dimp:i * dimp + dim, j * dimp:j * dimp + dim, 0] = B[:, i * cols
+ j].reshape(dim, dim)
            image[i * dimp:i * dimp + dim, j * dimp:j * dimp + dim, 1] = C[:, i * cols
+ j].reshape(dim, dim)
            image[i * dimp:i * dimp + dim, j * dimp:j * dimp + dim, 2] = D[:, i * cols
+ j].reshape(dim, dim)

    image = (image + 1) / 2

    return image
```

Chạy k-means cho 1000 mẫu dữ liệu đầu tiên

```
mndata = MNIST('../MNIST/')
mndata.load_testing()
X = mndata.test_images
X0 = np.asarray(X)[:1000,:]/256.0
X = X0

K = 10
kmeans = KMeans(n_clusters=K).fit(X)

pred_label = kmeans.predict(X)
```

Chọn hiển thị một số ảnh thuộc một số cụm

```
print(type(kmeans.cluster_centers_.T))
print(kmeans.cluster_centers_.T.shape)
A = display_network(kmeans.cluster_centers_.T, K, 1)

f1 = plt.imshow(A, interpolation='nearest', cmap = "jet")
f1.axes.get_xaxis().set_visible(False)
f1.axes.get_yaxis().set_visible(False)
plt.show()
# plt.savefig('a1.png', bbox_inches='tight')


# a colormap and a normalization instance
cmap = plt.cm.jet
norm = plt.Normalize(vmin=A.min(), vmax=A.max())

# map the normalized data to colors
# image is now RGBA (512x512x4)
image = cmap(norm(A))
import scipy.misc
scipy.misc.imsave('aa.png', image)
```

(Hai lệnh được đánh dấu cần kiểm tra có tương thích với thư viện hay không, nếu không hãy bỏ đi)

```
N0 = 20;
X1 = np.zeros((N0*K, 784))
X2 = np.zeros((N0*K, 784))

for k in range(K):
    Xk = X0[pred_label == k, :]

    center_k = [kmeans.cluster_centers_[k]]
    neigh = NearestNeighbors(N0).fit(Xk)
    dist, nearest_id  = neigh.kneighbors(center_k, N0)

    X1[N0*k: N0*k + N0,:] = Xk[nearest_id, :]
    X2[N0*k: N0*k + N0,:] = Xk[:N0, :]
```

Hiển thị kết quả

```
plt.axis('off')
A = display_network(X2.T, K, N0)
f2 = plt.imshow(A, interpolation='nearest' )
plt.gray()
plt.show()
```

Bài tập tự thực hành

Bài 1. Hãy áp dụng đoạn chương trình chúng ta tự xây dựng trong ví dụ 1, với dữ liệu là hình ảnh chữ số viết tay như trong ví dụ 2, chạy để xem xét kết quả.

Bài 2. Áp dụng mô hình trên cho bài tập phân loại ảnh chó-mèo (xem lại phần CNN), thử thực hiện phân cụm thành 02 cụm và kiểm tra kết quả.

2. PHẦN MÔ HÌNH TRỘN GAUSSIAN (GAUSSIAN MIXTURE MODEL)

2.1.Thuật toán EM - Xây dựng chương trình từ Numpy

Ví dụ 1: Dữ liệu 2D nhân tạo

Trong phần này chúng ta sử dụng dữ liệu nhân tạo với số chiều d = 2. Dữ liệu sẽ đươc tạo dưới dạng các cụm phân bố chuẩn, tuy nhiên ma trận hiệp phương sai sẽ được chọn để trục của cụm dữ liệu không song song với trục tọa độ.

```
def gen_data(k=3, dim=2, points_per_cluster=200, lim=[-20, 20]):
    '''
        Generates data from a random mixture of Gaussians in a given range. Will also
        plot the points in case of 2D. input:
            - k: Number of Gaussian clusters
            - dim: Dimension of generated points
            - points_per_cluster: Number of points to be generated for each cluster
            - lim: Range of mean values
    output:
            - X: Generated points (points_per_cluster*k, dim)
    '''
    x = []
    mean = random.rand(k, dim)*(lim[1]-lim[0]) + lim[0]
    for i in range(k):
        cov = random.rand(dim, dim+10)
        cov = np.matmul(cov, cov.T)
        _x = np.random.multivariate_normal(mean[i], cov, points_per_cluster)
        x += list(_x)
    x = np.array(x)
```

```
    if(dim == 2):
        fig = plt.figure()
        ax = fig.gca()
        ax.scatter(x[:,0], x[:,1], s=3, alpha=0.4)
        ax.autoscale(enable=True)
    return x
```

Phần khởi tạo cho thuật toán lặp

```
def __init__(self, k, dim, init_mu=None, init_sigma=None, init_pi=None, colors=None):
        '''
        Define a model with known number of clusters and dimensions.
        input:
            - k: Number of Gaussian clusters
            - dim: Dimension
            - init_mu: initial value of mean of clusters (k, dim)
                       (default) random from uniform[-10, 10]
            - init_sigma: initial value of covariance matrix of clusters (k, dim, dim)
                          (default) Identity matrix for each cluster
            - init_pi: initial value of cluster weights (k,)
                       (default) equal value to all cluster i.e. 1/k
            - colors: Color valu for plotting each cluster (k, 3)
                      (default) random from uniform[0, 1]
        '''
        self.k = k
        self.dim = dim
        if(init_mu is None):
            init_mu = random.rand(k, dim)*20 - 10
        self.mu = init_mu
        if(init_sigma is None):
            init_sigma = np.zeros((k, dim, dim))
            for i in range(k):
                init_sigma[i] = np.eye(dim)
        self.sigma = init_sigma
        if(init_pi is None):
            init_pi = np.ones(self.k)/self.k
        self.pi = init_pi
        if(colors is None):
            colors = random.rand(k, 3)
        self.colors = colors
```

Bước tính E:

- Theo công thức lý thuyết: $p(z_c|\mathbf{x}_n, \theta_t) = \frac{\pi_c N(\mu_{ct}, \Sigma_{ct}|\mathbf{x}_n)}{\sum_{k=1}^{C} \pi_k N(\mu_{kt}, \Sigma_{kt}|\mathbf{x}_n)}$ trong đó các ước lượng π_k, Σ_{kt} và μ_{kt} đều được lấy giá trị xấp xỉ có trong bước lặp t hiện tại.
- $N(\mu_{ct}, \Sigma_{ct}|\mathbf{x}_n)$ được tính bằng cách sử dụng hàm của thư viện numpy **multivariate_normal.pdf**

```
def e_step(self):
        '''
        E-step of EM algorithm.
        '''
        for i in range(self.k):
            self.z[:, i] = self.pi[i] * multivariate_normal.pdf(self.data,
mean=self.mu[i], cov=self.sigma[i])
        self.z /= self.z.sum(axis=1, keepdims=True)
```

Bước cực đại hóa (M-Step):

```
def m_step(self):
        '''
        M-step of EM algorithm.
```

```
        '''
        sum_z = self.z.sum(axis=0)
        self.pi = sum_z / self.num_points
        self.mu = np.matmul(self.z.T, self.data)
        self.mu /= sum_z[:, None]
        for i in range(self.k):
            j = np.expand_dims(self.data, axis=1) - self.mu[i]
            s = np.matmul(j.transpose([0, 2, 1]), j)
            self.sigma[i] = np.matmul(s.transpose(1, 2, 0), self.z[:, i] )
            self.sigma[i] /= sum_z[i]
```

Code đầy đủ có trong tệp Gaussian_Mix_EM.ipynb đính kèm.

Bài tập tự thực hành 1: Sử dụng đoạn code trên để áp dụng cho dữ liệu là phần đầu vào của tập dữ liệu hoa Iris (bỏ trường tên loại hoa). Sau khi phân cụm xong hãy đối sánh kết quả với các phân loại đúng.

Bài tập tự thực hành 2: Hãy sử dụng thuật toán K-means để phân cụm dữ liệu tự tạo đã có trong ví dụ 1 của phần này. So sánh và giải thích kết quả.

2.2. Sử dụng thư viện sk-learn

Ví dụ 2. Nhắc lại ví dụ ở bài lý thuyết:

Trong ví dụ này, chúng ta sử dụng dữ liệu là thông tin về chỉ số mua sắm và mức thu nhập có trong tệp dữ liệu shopping-data.csv. Tệp dữ liệu có 4 trường nhưng ta chỉ sử dụng 2 trường liên quan tới thu nhập và chỉ số mua sắm để tìm tương quan, các trường khác chỉ chứa thông tin ID nên ta bỏ qua.

Trong phần này ta sẽ sử dụng thư viện scikit-learn (sklearn) để thực hiện mô hình.

Các bước sẽ như sau:

(i) Đọc dữ liệu vào, chuẩn hóa (chú ý trong phần lý thuyết ta đã biết nếu kỳ vọng của các phân bố Gaussian là 0 thì tính toán sẽ tốt hơn); Ở trong code ta sẽ in ra một số thông tin của dữ liệu để xem đọc có đúng không.

(ii) Khởi tạo đối tượng của lớp mô hình trộn Gaussian thông qua hàm dựng

gm = GaussianMixture(n_components=K, covariance_type='full', random_state=0)

Ở đây số cụm được chia sẽ là n_component = K

(iii) Sau đó ta khớp dữ liệu của ta bằng đối tượng mô hình vừa tạo:

gm. fit(X_DATA)

(iv) Chúng ta vẽ dữ liệu ra dạng trực quan để có thể quan sát.

(v) Ở trong code dưới đây, chúng ta bổ sung thêm phần tìm xem K (số cụm nên dùng để chia dữ liệu) bao nhiêu là hợp lý nhất. Các làm của chúng ta ở đây đơn giản chỉ là thử với một dãy các giá trị số cụm, giá trị nào cho kết quả tốt nhất (trong số đó) thì sử dụng.

Dưới đây là các đoạn code:

Bước khai báo thư viện:

```
import numpy as np
import pandas as pd
import seaborn as sns
import itertools
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib.patheffects as PathEffects
from matplotlib.patches import Ellipse
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.mixture import GaussianMixture
# Thư viện chứa model Gaussian Mixture
```

Bước (i)

```
data = pd.read_csv("D:\\Teach_n_Train\\Machine
                       Learning\\slide\\Mixture_Model\\code
                                       \\shopping-data.csv",
                header=0,
                index_col=0)
print(data.shape)
data.head()

# Lấy ra thu nhập va điểm shopping
X = data.iloc[:, 2:4].values

# Chuẩn hoá dữ liệu
std = MinMaxScaler()
X_std = std.fit_transform(X)
print(X_std.shape)
```

Bước (ii) và bước (iii)

```
# Khởi tạo đối tượng mô hình GaussianMixture
gm = GaussianMixture(n_components=5,
                     covariance_type='full',
                     random_state=0)
gm.fit(X_std)
print('means: \n', gm.means_)
print('covariances: \n ', gm.covariances_)
```

Bước (v): Ta nêu phần chọn số K tốt nhất trước:

```
lowest_bic = np.infty
bic = []
n_components_range = range(1, 7)
# cv_types = ['spherical', 'tied', 'diag', 'full']
cv_types = ['full', 'tied']
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit Gaussian mixture theo phương pháp huấn luyện EM
        gmm = GaussianMixture(n_components=n_components,
                                        covariance_type=cv_type)
        gmm.fit(X_std)
        bic.append(gmm.bic(X_std))
        # Gán model có BIC scores thấp nhất là model tốt nhất
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

bic = np.array(bic)
color_iter = itertools.cycle(['navy', 'turquoise'])
clf = best_gmm
bars = []

# Vẽ biểu đồ BIC scores
plt.figure(figsize=(12, 8))
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + .2 * (i - 2)
    bars.append(plt.bar(xpos, bic[i * len(n_components_range):
                                  (i + 1) * len(n_components_range)],
                        width=.2, color=color))
plt.xticks(n_components_range)
```

```
plt.ylim([bic.min() * 1.01 - .01 * bic.max(), bic.max()])
plt.title('BIC score per model')
xpos = np.mod(bic.argmin(), len(n_components_range)) + .65 +\
    .2 * np.floor(bic.argmin() / len(n_components_range))
plt.text(xpos, bic.min() * 0.97 + .03 * bic.max(), '*', fontsize=14)
plt.xlabel('Number of components')
plt.legend([b[0] for b in bars], cv_types)
```

Bước (iv) – Hiển thị kết quả

```
def _plot_kmean_scatter(X, labels):
    '''
    X: dữ liệu đầu vào
    labels: nhãn dự báo
    '''
    # lựa chọn màu sắc
    num_classes = len(np.unique(labels))
    palette = np.array(sns.color_palette("hls", num_classes))

    # vẽ biểu đồ scatter
    fig = plt.figure(figsize=(12, 8))
    ax = plt.subplot()
    sc = ax.scatter(X[:,0], X[:,1], lw=0, s=40,
                                    c=palette[labels.astype(np.int)])

    # thêm nhãn cho mỗi cluster
    txts = []

    for i in range(num_classes):
        # Vẽ text tên cụm tại trung vị của mỗi cụm
        xtext, ytext = np.median(X[labels == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
        txts.append(txt)
    plt.title('t-sne visualization')
```

Cuối cùng: Gọi và thực hiện 2 phương thức chính: Thực hiện mô hình trộn Gaussian và phương thức hiển thị dữ liệu ra mặt phẳng 2 chiều để dễ quan sát.

```
labels = best_gmm.predict(X_std)
_plot_kmean_scatter(X_std, labels)
```

Chú ý với các ví dụ tiếp theo, chúng ta cần đọc dữ liệu đầy đủ và thiết lập vector dữ liệu X phù hợp (vì số chiều có thể lớn hơn). Sau đó trong phần hiển thị dữ liệu, chúng ta cần sử dụng mô hình phân tích thành phần chính (PCA) để giảm số chiều dữ liệu xuống còn 2 chiều để có thể hiển thị lên mặt phẳng.

2.3. **Ví dụ mở rộng.** Trong ví dụ này chúng ta sử dụng dữ liệu cho trong tệp đính kèm Sales_Transactions_Dataset_Weekly.csv hoặc tại link https://archive.ics.uci.edu/ml/machine-learning-databases/00396/ .

Mô tả dữ liệu: Dữ liệu chứa thông tin giao dịch bán hàng của 819 mã sản phẩm (Product ID) trong 819 dòng dữ liệu. Mỗi mã được đặc trưng bởi các thông tin: Mã (cột đầu); lượng giao dịch trong 52 tuần (week 0 – week 51), sau đó có thông tin chuẩn hóa của giao dịch các tuần (ta sử dụng thông tin này hoặc thông tin theo tuần – chỉ chọn 1 trong 2).

Yêu cầu: Hãy dựa vào ví dụ 1, mở rộng áp dụng với dữ liệu trên đây. Biết rằng số cụm tối ưu sẽ nằm trong khoảng từ 5 đến 15, hãy xác định số cụm và sau đó phân các mã sản phẩm thành các cụm dựa trên tiêu chí lượng giao dịch đã cho trong dữ liệu.