School of Data and Computer Science, Sun Yat-sen University

E13 EM Algorithm (C++/Python)

19214808 Yikun Liang

2019 年 12 月 5 日

目录

1 Chinese Football Dataset

The following Chinese Football Dataset has recored the performance of 16 AFC football teams between 2005 and 2018.

```
Country  2006WorldCup  2010WorldCup  2014WorldCup  2018WorldCup  2007AsianCup  2011AsianCup
    2015AsianCup
China       50          50          50          40          9           9           5
Japan       28          9           29          15          4           1           5
South_Korea             17          15          27          19          3           3           2
Iran        25          40          28          18          5           5           5
Saudi_Arabia            28          40          50          26          2           9           9
Iraq        50          50          40          40          1           5           4
Qatar       50          40          40          40          9           5           9
United_Arab_Emirates                50          40          50          40          9           9           3
Uzbekistan              40          40          40          40          5           4           9
Thailand                50          50          50          40          9           17          17
Vietnam     50          50          50          50          5           17          17
Oman        50          50          40          50          9           17          9
Bahrain     40          40          50          50          9           9           9
North_Korea             40          32          50          50          17          9           9
Indonesia               50          50          50          50          9           17          17
Australia               16          21          30          30          9           2           1
```

The scoring rules are below:

- For the FIFA World Cup, teams score the same with their rankings if they enter the World Cup; teams score 50 for failing to entering the Asia Top Ten; teams score 40 for entering the Asia Top Ten but not entering the World Cup.
- For the AFC Asian Cup, teams score the same with their rankings if they finally enter the top four; teams score 5 for entering the top eight but not the top four, and 9 for entering the top sixteen but not top eight; teams score 17 for not passing the group stages.

We aim at classifying the above 16 teams into 3 classes according to their performance: the first-class, the second-class and the third-class. In our opinion, teams of Australia, Iran, South Korea and Japan belong to the first-class, while the Chinese football team belongs to the third-class.

2 EM

2.1 The Gaussian Distribution

The Gaussian, also known as the normal distribution, is a widely used model for the distribution of continuous variables. In the case of a single variable x, the Gaussian distribution can be written in the form

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\} \tag{2.1.1}$$

where μ is the mean and σ^2 is the variance.

For a D-dimensional vector $\mathbf{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\mathbf{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\mathbf{\Sigma}|^{1/2}}\exp\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}\} \tag{2.1.2}$$

where $\boldsymbol{\mu}$ is a D-dimensional mean vector, $\mathbf{\Sigma}$ is a $D \times D$ covariance matrix, and $|\mathbf{\Sigma}|$ denotes the determinant of $|\mathbf{\Sigma}|$.

2.2 Mixtures of Gaussians

2.2.1 Introduction

While the Gaussian distribution has some important analytical properties, it suffers from significant limitations when it comes to modelling real data sets. Consider the example shown in Figure 1. This is known as the ‘Old Faithful’ data set, and comprises 272 measurements of the eruption of the Old Faithful geyser at Yel-lowstone National Park in the USA. Each measurement comprises the duration of the eruption in minutes (horizontal axis) and the time in minutes to the next eruption (vertical axis). We see that the data set forms two dominant clumps, and that a simple Gaussian distribution is unable to capture this structure, whereas a linear superposition of two Gaussians gives a better characterization of the data set.

图 1: Example of a Gaussian mixture distribution

Such superpositions, formed by taking linear combinations of more basic distributions such as Gaussians, can be formulated as probabilistic models known as *mixture distributions.* In Figure 1 we see that a linear combination of Gaussians can give rise to very complex densities. By using a sufficient number of Gaussians, and by adjusting their means and covariances as well as the coefficients in the linear combination, almost any continuous density can be approximated to arbitrary accuracy.

We therefore consider a superposition of K Gaussian densities of the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k,\mathbf{\Sigma}_k) \tag{2.2.1}$$

which is called a mixture of Gaussians. Each Gaussian density $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k,\mathbf{\Sigma}_k)$ is called a component of the mixture and has its own mean $\boldsymbol{\mu}_k$ and covariance $\mathbf{\Sigma}_k$.

The parameters π_k in (2.2.1) are called *mixing coefficients.* If we integrate both sides of (2.2.1) with respect to $\mathbf{x}$, and note that both $p(\mathbf{x})$ and the individual Gaussian components are normalized, we obtain

$$\sum_{k=1}^{K} \pi_k = 1. \tag{2.2.2}$$

Also, the requirement that $p(\mathbf{x}) \geq 0$, together with $\mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k) \geq 0$, implies $\pi_k \geq 0$ for all k. Combining this with condition (2.2.2) we obtain

$$0 \leq \pi_k \leq 1. \tag{2.2.3}$$

We therefore see that the mixing coefficients satisfy the requirements to be probabilities.

From the sum and product rules, the marginal density is given by

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(k)p(\mathbf{x}|k) \tag{2.2.4}$$

which is equivalent to (2.2.1) in which we can view $\pi_k = p(k)$ as the prior probability of picking the k^{th} component, and the density $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = p(\mathbf{x}|k)$ as the probability of $\mathbf{x}$ conditioned on k. From Bayes' theorem these are given by

$$\gamma_k(\mathbf{x}) = p(k|\mathbf{x}) = \frac{p(k)p(\mathbf{x}|k)}{\sum_l p(l)p(\mathbf{x}|l)} = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_l \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}. \tag{2.2.5}$$

The form of the Gaussian mixture distribution is governed by the parameters π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, where we have used the notation $\boldsymbol{\pi} = \{\pi_1, ..., \pi_K\}$, $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k\}$ and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, ..., \boldsymbol{\Sigma}_K\}$. One way to set the values of there parameters is to use maximum likelihood. From (2.2.1) the log of the likelihood function is given by

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\} \tag{2.2.6}$$

where $X = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$. One approach to maximizing the likelihood function is to use iterative numerical optimization techniques. Alternatively we can employ a powerful framework called expectation maximization (EM).

2.2.2 About Latent Variables

We now turn to a formulation of Gaussian mixtures in terms of discrete *latent* variables. This will provide us with a deeper insight into this important distribution, and will also serve to motivate the expectation-maximization (EM) algorithm.

Recall from (2.2.1) that the Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2.2.7}$$

Let us introduce a K-dimensional binary random variable $\mathbf{z}$ having a 1-of-K representation in which a particular element z_k is equal to 1 and all other elements are equal to 0. The values of z_k therefore satisfy $z_k \in \{0, 1\}$ and $\Sigma_k z_k = 1$, and we see that there are K possible states for the vector $\mathbf{z}$ according to which element is nonzero. We shall define the joint distribution $p(\mathbf{x}, \mathbf{z})$ in terms of a marginal distribution $p(\mathbf{z})$ and a conditional distribution $p(\mathbf{x}|\mathbf{z})$. The marginal distribution over $\mathbf{z}$ is specified in terms of the mixing coefficients π_k, such that

$$p(z_k = 1) = \pi_k \tag{2.2.8}$$

where the parameters $\{\pi_k\}$ must satisfy

$$0 \leq \pi_k \leq 1 \tag{2.2.9}$$

together with

$$\sum_{k=1}^{K} \pi_k = 1 \tag{2.2.10}$$

in order to be valid probabilities. Because $\mathbf{z}$ uses a 1-of-K representation, we can also write this distribution in the form

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}. \tag{2.2.11}$$

Similarly, the conditional distribution of $\mathbf{x}$ given a particular value for $\mathbf{z}$ is a Gaussian

$$p(\mathbf{x}|z_k = 1) = (\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{2.2.12}$$

which can also be written in the form

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} p(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k)^{z_k}. \tag{2.2.13}$$

The joint distribution is given by $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, and the marginal distribution of $\mathbf{x}$ is then obtained by summing the joint distribution over all possible states of $\mathbf{z}$ to give

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{2.2.14}$$

where we have made use of (2.2.12) and (2.2.13). Thus the marginal distribution of $\mathbf{x}$ is a Gaussian mixture of the form (2.2.7). If we have several observations $\mathbf{x_1}, ..., \mathbf{x_N}$, then, because we have represented the marginal distribution in the form $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$, it follows that for every observed data point $\mathbf{x}_n$ there is a corresponding latent variable $\mathbf{z}_n$.

We have therefore found an equivalent formulation of the Gaussian mixture involving an explicit latent variable. It might seem that we have not gained much by doing so. However, we are now able to work with the joint distribution $p(\mathbf{x}, \mathbf{z})$ instead of the marginal distribution $p(\mathbf{x})$, and this will lead to significant simplifications, most notably through the introduction of the expectation-maximization (EM) algorithm.

Another quantity that will play an important role is the conditional probability of $\mathbf{z}$ given $\mathbf{x}$. We shall use $\gamma(z_k)$to denote $p(z_k = 1|\mathbf{x})$, whose value can be found using Bayes' theorem

$$\gamma(z_k) = p(z_k = 1|\mathbf{x}) = \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\mathbf{x}|z_j = 1)} = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{\Sigma}_j)} \tag{2.2.15}$$

We shall view π_k as the prior probability of $z_k = 1$, and the quantity $\gamma(z_k)$ as the corresponding posterior probability once we have observed $\mathbf{x}$. As we shall see later, $\gamma(z_k)$ can also be viewed as the responsibility that component k takes for ‘explaining’ the observation $\mathbf{x}$.

2.3 EM for Gaussian Mixtures

Initially, we shall motivate the EM algorithm by giving a relatively informal treatment in the context of the Gaussian mixture model.

Let us begin by writing down the conditions that must be satisfied at a maximum of the likelihood function. Setting the derivatives of $\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to the means $\boldsymbol{\mu}_k$ of the Gaussian components to zero, we obtain

$$0 = -\sum_{n=1}^{n} \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{\gamma(z_{nk})} \sum_k (\mathbf{x}_n - \boldsymbol{\mu}_k) \tag{2.3.1}$$

Multiplying by $\boldsymbol{\Sigma}_k^{-1}$ (which we assume to be nonsingular) and rearranging we obtain

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{2.3.2}$$

where we have defined

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}). \tag{2.3.3}$$

We can interpret N_k as the effective number of points assigned to cluster k. Note carefully the form of this solution. We see that the mean $\boldsymbol{\mu}_k$ for the k^{th} Gaussian component is obtained by taking a weighted mean of all of the points in the data set, in which the weighting factor for data point $\mathbf{x}_n$ is given by the posterior probability $\gamma(z_{nk})$ that component k was responsible for generating $\mathbf{x}_n$.

If we set the derivative of $\ln(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to $\boldsymbol{\Sigma}_k$ to zero, and follow a similar line of reasoning, making use of the result for the maximum likelihood for the covariance matrix of a single Gaussian, we obtain

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \tag{2.3.4}$$

which has the same form as the corresponding result for a single Gaussian fitted to the data set, but again with each data point weighted by the corresponding posterior probability and with the denominator given by the effective number of points associated with the corresponding component.

Finally, we maximize $\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to the mixing coefficients π_k. Here we must take account of the constraint $\sum_{k=1}^{K} \pi_k = 1$. This can be achieved using a Lagrange multiplier and maximizing the following quantity

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda(\sum_{k=1}^{K} \pi_k - 1) \tag{2.3.5}$$

which gives

$$0 = \sum_{n=1}^{N} \frac{\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2.3.6}$$

where again we see the appearance of the responsibilities. If we now multiply both sides by π_k and sum over k making use of the constraint $\sum_{k=1}^{K} \pi_k = 1$, we find $\lambda = -N$. Using this to eliminate λ and rearranging we obtain

$$\pi_k = \frac{N_k}{N} \tag{2.3.7}$$

so that the mixing coefficient for the k^{th} component is given by the average responsibility which that component takes for explaining the data points.

2.4 EM Algorithm

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters (comprising the means and covariances of the components and the mixing coefficients).

1. Initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
2. **E step**. Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2.4.1}$$

3. **M step**. Re-estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{2.4.2}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{new})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})^{\mathrm{T}} \tag{2.4.3}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{2.4.4}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}). \tag{2.4.5}$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\} \tag{2.4.6}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

3 Tasks

- Assume that score vectors of teams in the same class are normally distributed, we can thus adopt the Gaussian mixture model. Please classify the teams into 3 classes by using EM algorithm. If necessary, you can refer to page 430-439 in the book `Pattern Recognition and Machine Learning.pdf` and the website `https://blog.csdn.net/jinping_shi/article/details/59613054` which is a Chinese translation.
- You should show the values of these parameters: γ, μ and Σ. If necessary, you can plot the clustering results. Note that γ is essential for classifying.
- Please submit a file named `E13_YourNumber.pdf` and send it to `ai_201901@foxmail.com`

4 Codes and Results

本次实现了两个算法，一个使用了 scikit-learn 包提供的 api，另一个使用 EM 算法进行计算。本文结尾部分提供了两种方法的对比。

首先是 scikit 的实现。代码如下：

```python
# -*- coding: utf-8 -*-
#  使用EM算法解算GGM  EM算法采用scikit-learn包提供的api
#  数据集：《机器学习》--西瓜数据4.0   :文件watermelon4.txt

from sklearn import mixture
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
import pandas

pca = PCA(n_components = 2)

# 预处理数据
def loadData(filename):
    data = []
    with open(filename, 'r') as rfile:
        for line in rfile.readlines():
            list = line.split()
            list.remove(list[0])
            data.append(list)

    data.remove(data[0])
    newdata = []
    for list in data:
        newlist = []
        for i in list:
            newlist.append(int(i))
        newdata.append(newlist)
    dataMat = np.array(newdata)
    #为方便后面画图展示，此处将数据进行降维到二维。训练时也使用二维数据。
    #降维使用sklearn包的pca方法。
    dataMat = pca.fit_transform(dataMat)
    return dataMat



def test_GMM(dataMat, components=3,iter = 100,cov_type="full"):
    clst = mixture.GaussianMixture(n_components=n_components,max_iter=iter,
                                                covariance_type=cov_type)
    clst.fit(dataMat)
    predicted_labels =clst.predict(dataMat)
    return clst.means_,predicted_labels     # clst.means_返回均值


#在属性空间中绘图表示结果
def showCluster(dataMat, k, centroids, clusterAssment):
    numSamples, dim = dataMat.shape
    if dim != 2:
        print("Sorry! I can not draw because the dimension of your data is not 2!")
        return 1

    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']
    if k > len(mark):
        print("Sorry! Your k is too large!")
        return 1

        # draw all samples
    for i in range(numSamples):
        markIndex = int(clusterAssment[i])
        plt.plot(dataMat[i, 0], dataMat[i, 1], mark[markIndex])

    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']
    # draw the centroids
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize=12)

    plt.show()


if __name__=="__main__":
    dataMat = loadData('football.txt')
    n_components = 3
    iter=100
    cov_types = ['spherical', 'tied', 'diag', 'full']
    centroids,labels = test_GMM(dataMat,n_components,iter,cov_types[3])
    print(labels)
    showCluster(dataMat, n_components, centroids, labels)  # 这里labels维度改变了，注意
                                                    修改showCluster方法
```

下面展示的是 EM 算法。

```python
# -*- coding: utf-8 -*-
# 高斯混合模型  使用EM算法解算
# 数据集：《机器学习》--西瓜数据4.0   :文件watermelon4.txt
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# 预处理数据
def loadData(filename):
    dataSet = []
    fr = open(filename)
    for line in fr.readlines():
        curLine = line.strip().split(' ')
        fltLine = list(map(float, curLine))
        dataSet.append(fltLine)
    return dataSet

#将数据库进行归一化
def normalization(datingDatamat):
   max_arr = datingDatamat.max(axis=0)
   min_arr = datingDatamat.min(axis=0)
   ranges = max_arr - min_arr
   norDataSet = np.zeros(datingDatamat.shape)
   m = datingDatamat.shape[0]
   norDataSet = datingDatamat - np.tile(min_arr, (m, 1))
   norDataSet = norDataSet/np.tile(ranges,(m,1))
   return norDataSet

#降维后变为2维
pca = PCA(n_components = 2)

# 预处理数据
def myloadData(filename):
    data = []
    with open(filename, 'r') as rfile:
        for line in rfile.readlines():
            list = line.split()
            list.remove(list[0])
            data.append(list)

    data.remove(data[0])
    newdata = []
    for list in data:
        newlist = []
        for i in list:
            newlist.append(int(i))
        newdata.append(newlist)
    dataMat = np.array(newdata)
    #矩阵使用pca进行降维，方便后面进行计算和展示属性空间
    dataMat = pca.fit_transform(dataMat)
    return dataMat


# 高斯分布的概率密度函数
def prob(x, mu, sigma):
    n = np.shape(x)[1]
    #求样本x到均值的马氏距离
    expOn = float(-0.5 * (x - mu) * (sigma.I) * ((x - mu).T))
    #计算高斯概率密度
    divBy = pow(2 * np.pi, n / 2) * pow(np.linalg.det(sigma), 0.5)  # np.linalg.det 计算
                                                                    矩阵的行列式
    return pow(np.e, expOn) / divBy

# EM算法
def EM(dataMat, maxIter=50):
    m, n = np.shape(dataMat)
    # 1.初始化各高斯混合成分参数
    alpha = [1 / 3, 1 / 3, 1 / 3]   # 1.1初始化 alpha1=alpha2=alpha3=1/3
    #mu = [dataMat[5, :], dataMat[21, :], dataMat[26, :]] # 1.2初始化 mu1=x6,mu2=x22,mu3
                                                                    =x27
    mu = [dataMat[0, :], dataMat[1, :], dataMat[6, :]]  # 1.2初始化 mu1=(China),mu2=(
                                                                    Japan),mu3=(Qatar)
    sigma = [np.mat([[0.1, 0], [0, 0.1]]) for x in range(3)]    # 1.3初始化协方差矩阵
    gamma = np.mat(np.zeros((m, 3)))
    for i in range(maxIter):
        #遍历整个数据库
        for j in range(m):
            sumAlphaMulP = 0
            for k in range(3):
                gamma[j, k] = alpha[k] * prob(dataMat[j, :], mu[k], sigma[k]) # 4.计算混
                                                                    合成分生成的后验概率，即
                                                                    gamma
                sumAlphaMulP += gamma[j, k]
            for k in range(3):
                gamma[j, k] /= sumAlphaMulP #计算得到gamma后验概率
        sumGamma = np.sum(gamma, axis=0)

        #遍历每个类，计算每个类的均值mu和sigma与混合系数
        for k in range(3):
            mu[k] = np.mat(np.zeros((1, n)))
            sigma[k] = np.mat(np.zeros((n, n)))
            for j in range(m):
                mu[k] += gamma[j, k] * dataMat[j, :]
            mu[k] /= sumGamma[0, k] #  7.计算新均值向量
            for j in range(m):
                sigma[k] += gamma[j, k] * (dataMat[j, :] - mu[k]).T *(dataMat[j, :] - mu[k])
            sigma[k] /= sumGamma[0, k]  # 8. 计算新的协方差矩阵
            alpha[k] = sumGamma[0, k] / m   # 9. 计算新混合系数
            # print(mu)
    return gamma


# init centroids with random samples
def initCentroids(dataMat, k):
    numSamples, dim = dataMat.shape
    centroids = np.zeros((k, dim))
    for i in range(k):
        index = int(np.random.uniform(0, numSamples))
        centroids[i, :] = dataMat[index, :]
    return centroids


#高斯聚类的主函数
def gaussianCluster(dataMat):
    m, n = np.shape(dataMat)
    centroids = initCentroids(dataMat, m)  ## step 1: init centroids
    clusterAssign = np.mat(np.zeros((m, 2)))
    gamma = EM(dataMat)
    #计算得到每一个example对应的后验概率，取最大值对应下标即为对应的分类结果。
    for i in range(m):
        # amx返回矩阵最大值，argmax返回矩阵最大值所在下标
        clusterAssign[i, :] = np.argmax(gamma[i, :]), np.amax(gamma[i, :])  # 15.确定x的簇标记lambda
        ## step 4: update centroids
    for j in range(m):
        pointsInCluster = dataMat[np.nonzero(clusterAssign[:, 0].A == j)[0]]
        centroids[j, :] = np.mean(pointsInCluster, axis=0)  # 计算出均值向量
    return centroids, clusterAssign


#展示聚类的函数
def showCluster(dataMat, k, centroids, clusterAssment):
    numSamples, dim = dataMat.shape
    if dim != 2:
        print("Sorry! I can not draw because the dimension of your data is not 2!")
        return 1

    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']
    if k > len(mark):
        print("Sorry! Your k is too large!")
        return 1

        # draw all samples
    for i in range(numSamples):
        markIndex = int(clusterAssment[i, 0])
        plt.plot(dataMat[i, 0], dataMat[i, 1], mark[markIndex])

    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']
    # draw the centroids
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize=12)

    plt.show()

if __name__=="__main__":
    dataMat = np.mat(loadData('watermelon4.txt'))
    dataMat = np.mat(myloadData('football.txt'))
    #为统一输入矩阵，此处对矩阵进行归一化。
    dataMat = normalization(dataMat)
    #print(dataMat)

    centroids, clusterAssign = gaussianCluster(dataMat)
    print(clusterAssign)
    showCluster(dataMat, 3, centroids, clusterAssign)
```

实验结果如下:

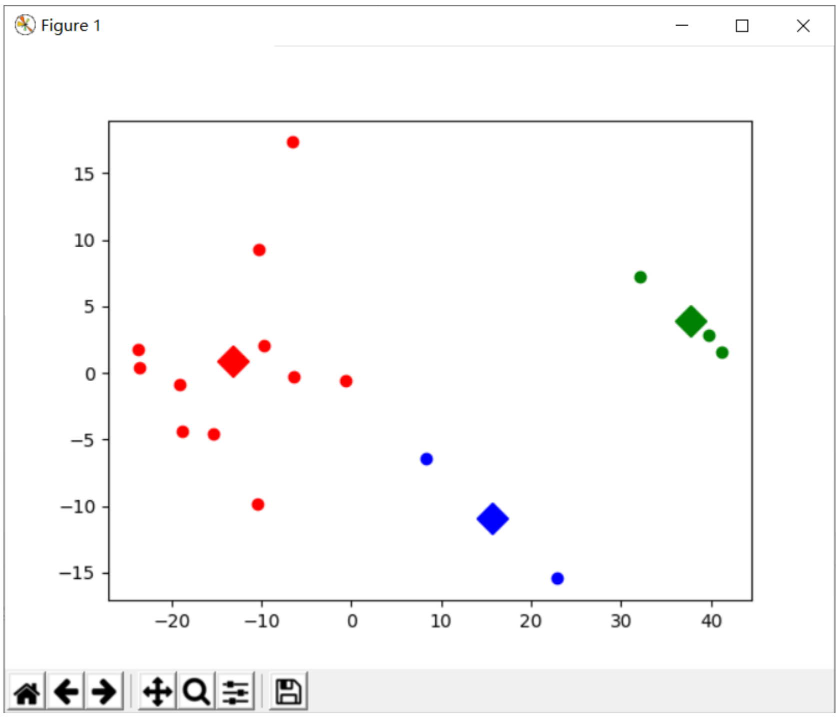
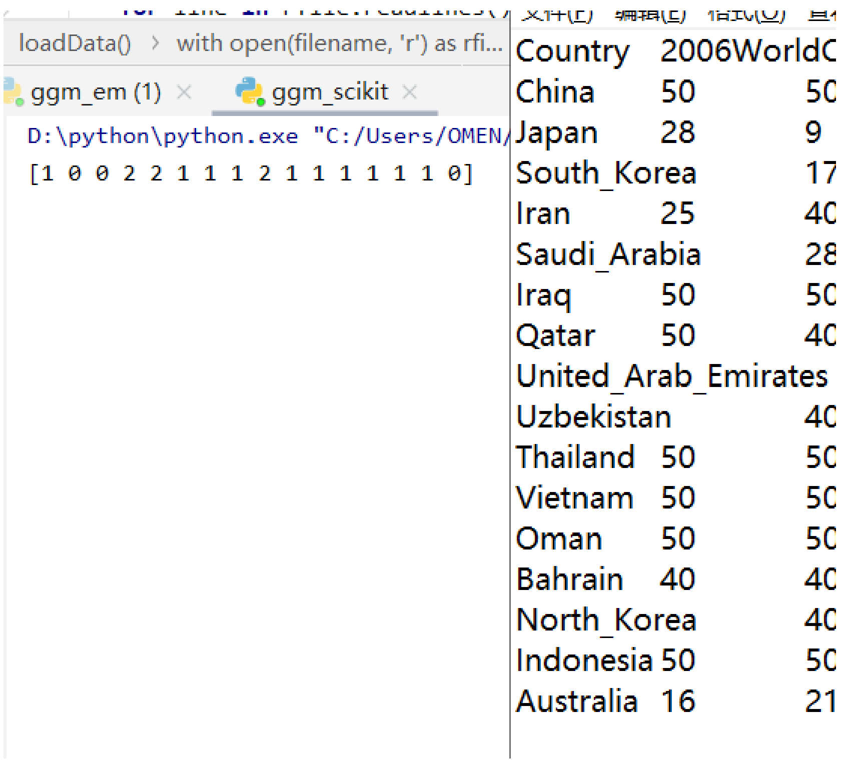


图 2: scikit



```
D:\python\python.exe "C:/Users/OMEN/
[1 0 0 2 2 1 1 1 2 1 1 1 1 1 1 0]
```

```
Country     2006WorldC
China       50      50
Japan       28      9
South_Korea         17
Iran        25      40
Saudi_Arabia        28
Iraq        50      50
Qatar       50      40
United_Arab_Emirates
Uzbekistan          40
Thailand    50      50
Vietnam     50      50
Oman        50      50
Bahrain     40      40
North_Korea         40
Indonesia 50        50
Australia   16      21
```

图 3: labels prediceted by scikit

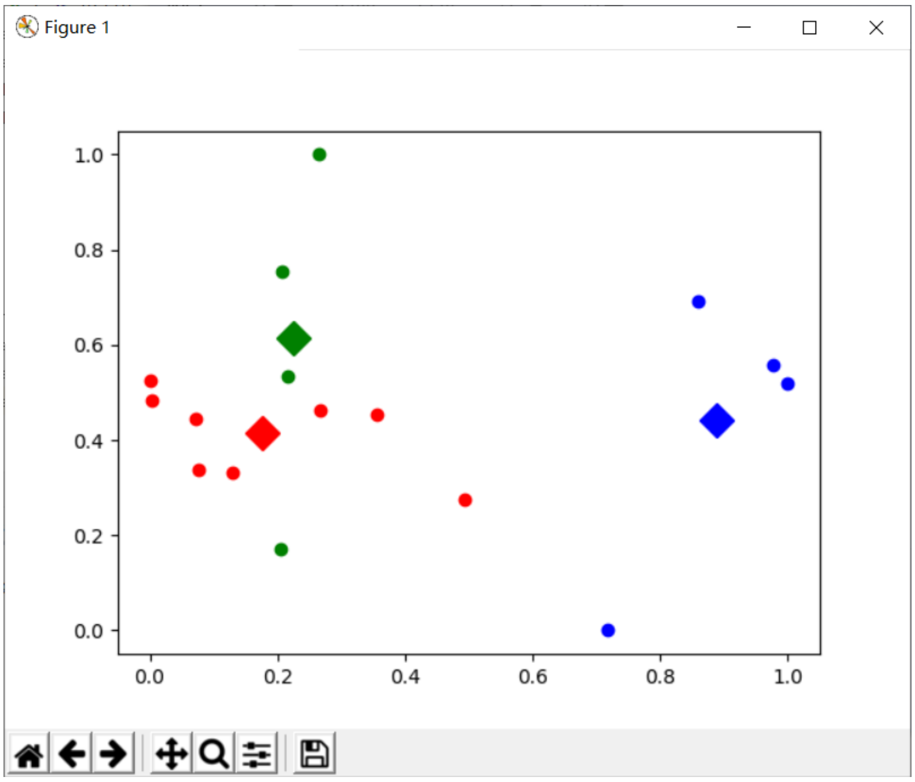
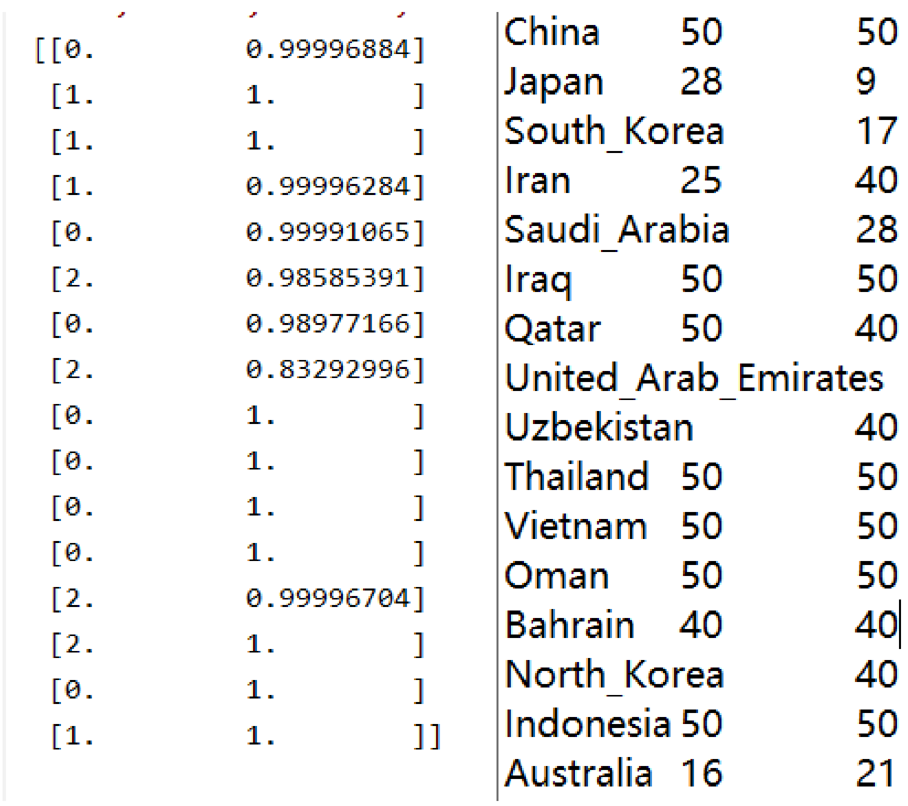


图 4: EM



```
[[0.          0.99996884]
 [1.          1.        ]
 [1.          1.        ]
 [1.          0.99996284]
 [0.          0.99991065]
 [2.          0.98585391]
 [0.          0.98977166]
 [2.          0.83292996]
 [0.          1.        ]
 [0.          1.        ]
 [0.          1.        ]
 [0.          1.        ]
 [2.          0.99996704]
 [2.          1.        ]
 [0.          1.        ]
 [1.          1.        ]]
```

China	50	50
Japan	28	9
South_Korea		17
Iran	25	40
Saudi_Arabia		28
Iraq	50	50
Qatar	50	40
United_Arab_Emirates		
Uzbekistan		40
Thailand	50	50
Vietnam	50	50
Oman	50	50
Bahrain	40	40
North_Korea		40
Indonesia	50	50
Australia	16	21

图 5: labels predicted by EM