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

E13 EM Algorithm (C++/Python)

17341137 Zhenpeng Song

December 6, 2019

Contents

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 20
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

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

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

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.

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

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

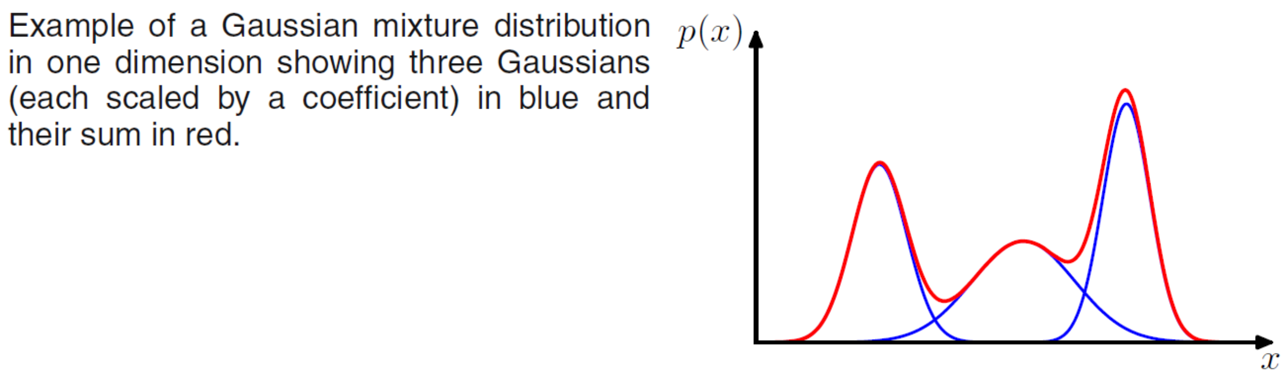

Example of a Gaussian mixture distribution in one dimension showing three Gaussians (each scaled by a coefficient) in blue and their sum in red.

Figure 1: Example of a Gaussian mixture distribution

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

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

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

which can also be written in the form

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

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

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

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

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

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

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

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: $\boldsymbol{\gamma}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. If necessary, you can plot the clustering results. Note that $\boldsymbol{\gamma}$ is essential for classifying.
- Please submit a file named `E13_YourNumber.pdf` and send it to `ai_201901@foxmail.com`

4 Codes and Results

4.1 Codes

```
# -*- coding: utf-8 -*-
"""
GMM-EM

 Classifier with Gaussian Mixture Model(GMM),
 using Expectation-Maximization algorithm(EM).

Name: Zhenpeng Song
ID:    17341137
Data:  Chinese Football Dataset wrt. 16 AFC football teams(2005-2018).
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# ===================================================================
# 1 LOAD DATA
# ===================================================================

##
# Data_loader
# @Func:    Load data from file
# @params: file_path, seperator
# @return: train_data, convert_Dict
##
def Data_loader(path, sep=','):
    train_data = pd.read_csv(path, sep=' ')
    # map id to country name
    id2country = {idx: train_data['Country'][idx] for idx in train_data.ind
    # dataframe -> ndarrary
    train_data = np.array(train_data.drop('Country', 1))
    # Strategies:
    # 1. Obeserving the data, I found that the smaller the score was,
    #    the better the team played.
    # 2. In order to lower the dimension from 7 to 2,
    #    I took the two average scores as the final evaluation
    #    features(one for World Cup and one for Asian Cup).
    # 3. As a result, the X_train dataset will be of shape(16, 2).
    X_train = np.hstack(((np.sum(train_data[:, 0:3]/4, axis=1).reshape(-1,
                         (np.sum(train_data[:, 4:]/3, axis=1).reshape(-1, 1
    return X_train, id2country

# =======================================================================
# 2 GMM Definition
# =======================================================================


class EM(object):

    # __init__
    # @Func:    Initiate the model.
    # @params: Dataset, class number to classify to, sigma_regularizer
    # @return: NO RETURN
    def __init__(self, Data, num_class, sigma_reg=1):
        self.Data = Data
        self.num_class = num_class
        self.params = self.Init_params(sigma_reg)

    # mvGaussian
    # @Func:    Calculate the multivariate Gaussian probability.
    # @params: Data x, Average value mu, Covariance sigma
    # @return: The multivariate Gaussian probability.
    def mvGaussian(self, x, mu, sigma):
        # import dot() for multiplication between matrices,
        # power() for power calculation,
        # inv() for inverse calculation of sigma,
        # det() for determinant calculation of sigma.
        from numpy import dot, power
        from numpy.linalg import inv, det

        # n is the dimension of the features space,
        # in this case, n = 2
        n = x.shape[1]
        # nume denotes for numerator and deno denotes for denominator,
        # corresponding to the equation.
        nume = np.exp(-0.5*dot(dot((x-mu), inv(sigma)), (x-mu).T))
        deno = power(2 * np.pi, n / 2) * power(det(sigma), 0.5)
        return nume / deno

    # Init_params
    # @Func:    Initialize the parameters.
    # @params:  sigma_regularizer
    # @return:  Initialized parameters.
    def Init_params(self, sigma_reg):
        # m is the number of samples, n is the demension of features
        m, n = self.Data.shape
        num_class = self.num_class

        # Parameters:
        # alpha: the mixing coef. in GMM
        # mu   : the average vector.
        # sigma: the covariance matrix.
        # gamma: Of shape(m, k), represent the probability for sample m
        #        to be of category k.
        alpha = np.array([1.0/num_class for i in range(num_class)])
        mu = np.array([self.Data[i*(m//num_class)] for i in range(num_class
        sigma = np.array([sigma_reg*np.eye(n) for i in range(num_class)])
        gamma = np.zeros((m, num_class))
        return alpha, mu, sigma, gamma

    # E_step
    # @Func:    E step of EM algorithm.
    # @params: NO Parms.
    # @return: NO RETURN
    def E_step(self):
        # m is the number of samples, n is the demension of features
        m, n = self.Data.shape
        num_class = self.num_class
        alpha, mu, sigma, gamma = self.params

        # Go through samples, calculate gamma
        # using bayes theory.
        for j in range(m):
            x = self.Data[j].reshape(1, n)
            deno = np.sum([alpha[i]*self.mvGaussian(x,mu[i],sigma[i]) for i
            for i in range(num_class):
                nume = alpha[i] * self.mvGaussian(x, mu[i], sigma[i])
                gamma[j][i] = nume / deno
        # Update parameters
        self.params = alpha, mu, sigma, gamma

    # M_step
    # @Func:    M step of EM algorithm.
    # @params: NO Parms.
    # @return: NO RETURN
    def M_step(self):
        # m is the number of samples, n is the demension of features
        m, n = self.Data.shape
        x = self.Data
        num_class = self.num_class
        alpha, mu, sigma, gamma = self.params

        # Regularizer to avoid sigma becoming singular.
        regularizer = 0.1 * np.eye(n)

        # sum up gamma as the denominator in following calculations
        sum_gamma = np.sum(gamma, axis=0)
        # Go through classes, update parameters step by step.
        for i in range(num_class):
            # Since numpy will defaultly
            # set (x, 1)-like ndarray to (x,),
            # which is known as 'rank 1 array'
            # and does not support numpy broadcast,
            # use reshape() to fix it.
            tmp_gamma = gamma[:, i].reshape(-1,1)

            # Parameters Updating
            # mu
            mu[i] = np.sum(tmp_gamma*x, axis=0) / sum_gamma[i]
            # sigma
            nume_sigma = 0
            for j in range(m):
                nume_sigma += gamma[j, i] * (x[j] - mu[i]).T * (x[j] - mu[i
            sigma[i] = nume_sigma / sum_gamma[i] + regularizer
            # alpha
            alpha[i] = sum_gamma[i] / m
        # Update parameters
        # (This step is not necessary since ndarray is passed by reference)
        self.params = alpha, mu, sigma, gamma

    # Norm_diff
    # @Func:   Calculate the norm-difference between two iters.
    # @params: pre: params in last iter., cur: params in current iter.
    # @return: the norm-difference.
    def Norm_diff(self, pre, cur):
        norm_diff = [np.linalg.norm(pre[i]-cur[i]) for i in range(len(pre))
        return np.average(norm_diff)

    # clusteror
    # @Func:   Cluster the dataset by gamma
    # @params: gamma
    # @return: The clustered result as a dict.
    def clusteror(self, gamma):
        m, n = self.Data.shape
        # Initialize C as the clustered dictionary
        # C: sample_id: (sample_id, sample_class)
        C = {}
        # Simply, use np.argmax() to
        # find the index of the largest element.
        for j in range(m):
            C[j] = (j, np.argmax(gamma[j]))
        return C

    # EM_method
    # @Func:   EM algorithm.
    # @params: Epsilon as a threshold for norm-difference,
    #          MaxIter to limit the max epochs.
    # @return: Clustered result as a dictionary, final mu.
    def EM_method(self, Epsilon=5, MaxIter=500):
        # Track the params for debugging
        pre_params = []
        pre_params.append([attr.copy() for attr in self.params])
        norm_diff = np.inf

        while (norm_diff > Epsilon and MaxIter):
            MaxIter -= 1
            self.E_step()
            self.M_step()
            norm_diff = self.Norm_diff(pre_params[-1], list(self.params))
            pre_params.append([attr.copy() for attr in self.params])
            # print(norm_diff)
        return self.clusteror(self.params[-1]), self.params


# ============================================================================
# 3 Utilities
# ============================================================================

def plotter(ClusterDict, Centers, Data, convert_dict, num_class):
    colorset = ['red', 'blue', 'yellow', 'black', 'green']
    color = {i:colorset[i] for i in range(num_class)}
    fig = plt.figure(figsize=(9,6))
    ax = fig.add_subplot(111)
    for i in Centers:
        ax.scatter(i[0], i[1], s=100, c='green', marker='D')
    for i in ClusterDict:
        ax.scatter(Data[i][0], Data[i][1], color=color[ClusterDict[i][1]])
        ax.annotate(convert_dict[ClusterDict[i][0]],
                    xy=(Data[i][0], Data[i][1]),
                    xytext=(Data[i][0]+0.1, Data[i][1]+0.1))

    plt.title('Clustered Result')
    plt.xlim(10, 45)
    plt.ylim(0, 20)
    ax.set_xlabel('Appearance in WorldCup')
    ax.set_ylabel('Appearance in AsianCup')
    plt.show()
```

```python


def displayParams(params):
    print("="*10+"Parameters:"+"="*10)
    print("Mu:")
    for idx, i in enumerate(params[1]):
        print("Mu[{:d}]: ".format(idx), i)
    print("Sigma:")
    for idx, i in enumerate(params[2]):
        print("Sigma[{:d}]:\n".format(idx), i)
    print("Gamma:")
    for idx, i in enumerate(params[3]):
        print("Gamma[{:2d}]: ".format(idx), i)


# =========================================================================
# 4 Run the model
# =========================================================================

FILE_PATH = './E13_dataset.csv'
SEP = ' '
NUM_CLASS = 3
SIGMA_REG = 1000
EPSILON = 5

if __name__ == '__main__':
    X_train, id2country = Data_loader(FILE_PATH, SEP)
    Em = EM(X_train, NUM_CLASS, SIGMA_REG)
    ClusteredDict, params = Em.EM_method(EPSILON)
    plotter(ClusteredDict, params[1], X_train, id2country, NUM_CLASS)
    displayParams(params)
```

For a clearer version, please refer to ‘src/GMM_EM.py‘

4.2 Results

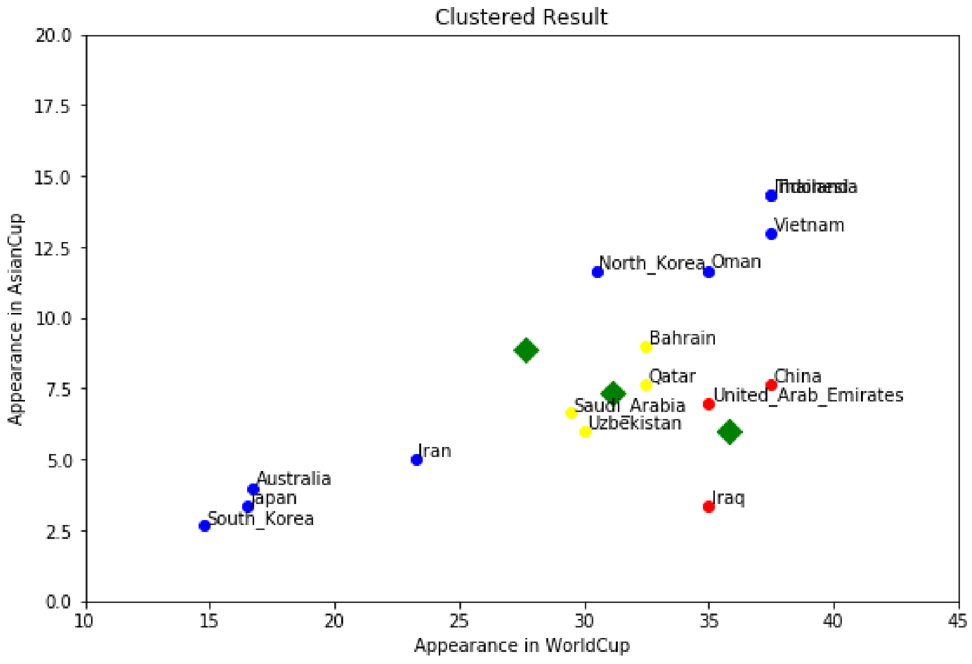
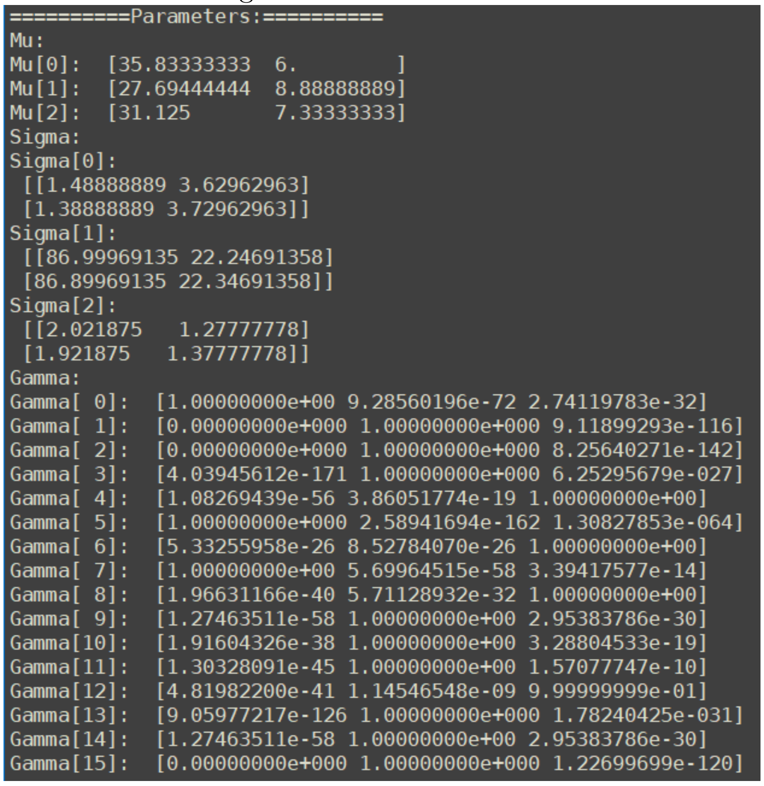


Figure 2: Clustered Result



```
==========Parameters:==========
Mu:
Mu[0]:  [35.83333333  6.        ]
Mu[1]:  [27.69444444  8.88888889]
Mu[2]:  [31.125       7.33333333]
Sigma:
Sigma[0]:
 [[1.48888889 3.62962963]
 [1.38888889 3.72962963]]
Sigma[1]:
 [[86.99969135 22.24691358]
 [86.89969135 22.34691358]]
Sigma[2]:
 [[2.021875   1.27777778]
 [1.921875   1.37777778]]
Gamma:
Gamma[ 0]:  [1.00000000e+00 9.28560196e-72 2.74119783e-32]
Gamma[ 1]:  [0.00000000e+000 1.00000000e+000 9.11899293e-116]
Gamma[ 2]:  [0.00000000e+000 1.00000000e+000 8.25640271e-142]
Gamma[ 3]:  [4.03945612e-171 1.00000000e+000 6.25295679e-027]
Gamma[ 4]:  [1.08269439e-56 3.86051774e-19 1.00000000e+00]
Gamma[ 5]:  [1.00000000e+000 2.58941694e-162 1.30827853e-064]
Gamma[ 6]:  [5.33255958e-26 8.52784070e-26 1.00000000e+00]
Gamma[ 7]:  [1.00000000e+00 5.69964515e-58 3.39417577e-14]
Gamma[ 8]:  [1.96631166e-40 5.71128932e-32 1.00000000e+00]
Gamma[ 9]:  [1.27463511e-58 1.00000000e+00 2.95383786e-30]
Gamma[10]:  [1.91604326e-38 1.00000000e+00 3.28804533e-19]
Gamma[11]:  [1.30328091e-45 1.00000000e+00 1.57077747e-10]
Gamma[12]:  [4.81982200e-41 1.14546548e-09 9.99999999e-01]
Gamma[13]:  [9.05977217e-126 1.00000000e+000 1.78240425e-031]
Gamma[14]:  [1.27463511e-58 1.00000000e+00 2.95383786e-30]
Gamma[15]:  [0.00000000e+000 1.00000000e+000 1.22699699e-120]
```

Figure 3: Final Parameters