哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： 实现 k-means 聚类方法和混合高斯模型

学号：1162100102
班级：1603104
姓名： 王晨懿

目录

一、实验目的

实现一个 k-means 算法和混合高斯模型，并且用 EM 算法估计模型中的参数。

二、实验要求及实验环境

实验要求：

测试：

用高斯分布产生 k 个高斯分布的数据（不同均值和方差）（其中参数自己设定）。

（1）用 k-means 聚类，测试效果；

（2)用混合高斯模型和你实现的 EM 算法估计参数，看看每次迭代后似然值变化情况，考察 EM 算法是否可以获得正确的结果（与你设定的结果比较）。

应用：

可以 UCI 上找一个简单问题数据，用你实现的 GMM 进行聚类。

实验环境：

Window10 64 位操作系统
Pycharm

二、设计思想（本程序中的用到的主要算法及数据结构）

k-means

算法原理

在多维空间中数据点的分组或聚类的问题中，假设我们有一个数据集 $D = \{x_1, \dots, x_n\}$，我们的目标是将数据集划分为 K 个类别 $C = \{C_1, \dots, C_K\}$ 。

令 μ_k 表示第 K 个聚类的中心，我们要找到数据点分别属于的聚类，以及一组向量 $\{\mu_k\}$ ，使得每个数据点和与它最近的向量 μ_k 之间的距离的平方和最小。

引入 $r_{nk} \in \{0,1\}$ ，其中 $k = 1, \dots, K$ ，采用"1-of-K" 的表示方式，表示数据点 x_n 属于 K 个聚类中的哪一个。

若将欧几里得距离作为距离度量，则可定义目标函数（失真度量）为：

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\left||x_n - \mu_k|\right|^2$$

目标是找到 $\{r_{nk}\}$ *和* $\{\mu_k\}$ 的值，使得 J 达到最小值。

我们采用一种迭代的方式，使得 J 达到最小值

①关于 r_{nk} 最小化 J ，保持 μ_k 固定

②关于 μ_k 最小化 J ，保持 r_{nk} 固定。

在步骤①中：

我们对每个 n 分别进行最优化，只要 k 的值使 $\left\|x_n - \mu_j\right\|^2$ 最小，我们就令 r_{nk} 为 1。

$$r_{nk} = \begin{cases} 1 & \text{如果} k = argmin_j \left\|x_n - \mu_j\right\|^2 \\ 0 & \text{其他情况} \end{cases}$$

在步骤②中：

令 J 关于 μ_k 的导数等于 0，则有

$$2\sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0$$

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

等价于

$$\mu_k = \frac{1}{|C_k|}\sum_{x \in C_k} x$$

算法流程如下：

输入： 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
聚类簇数 k.

过程：

从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$
repeat
  令 $C_i = \varnothing \ (1 \leqslant i \leqslant k)$
  for $j = 1, 2, \ldots, m$ **do**
    计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu}_i \ (1 \leqslant i \leqslant k)$ 的距离: $d_{ji} = \|\boldsymbol{x}_j - \boldsymbol{\mu}_i\|_2$;
    根据距离最近的均值向量确定 $\boldsymbol{x}_j$ 的簇标记: $\lambda_j = \arg\min_{i \in \{1,2,\ldots,k\}} d_{ji}$;
    将样本 $\boldsymbol{x}_j$ 划入相应的簇: $C_{\lambda_j} = C_{\lambda_j} \bigcup \{\boldsymbol{x}_j\}$;
  end for
  for $i = 1, 2, \ldots, k$ **do**
    计算新均值向量: $\boldsymbol{\mu}'_i = \frac{1}{|C_i|}\sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
    if $\boldsymbol{\mu}'_i \neq \boldsymbol{\mu}_i$ **then**
      将当前均值向量 $\boldsymbol{\mu}_i$ 更新为 $\boldsymbol{\mu}'_i$
    else
      保持当前均值向量不变
    end if
  end for
until 当前均值向量均未更新

输出： 簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

算法的实现

```
def k_means(data, k):
    """
    k-means算法
    :param data: 样本矩阵
    :param k: 类别数
    :return: 分类结果
    """
```

1. 首先初始化数据，对分类和均值向量进行初始化。我这里的均值向量的初始化是在 data 中随机选择 k 个点作为每个类的均值向量

```
N, m = data.shape
means = [data[random.randint(0, N - 1)] for i in range(k)]
for i in range(k):
    means[i] = 20 * np.random.random(2)
y = [-1] * N  # 初始化分类
update = True
C = [-1] * k
```

2. 然后进行迭代：

a) 关于 r_{nk} 最小化 J ，保持 μ_k 固定：

计算样本与各均值向量的距离，根据距离最近的均值向量确定样本 xi 的簇标记 yi，然后根据簇标记将样本划入响应的簇 C

这里的 my_min 函数返回最小值 d 及其索引 idx。

```
for i in range(len(data)):
    d, idx = my_min([dist(data[i], mean) for mean in means])
    y[i] = idx
for i in range(k):
    C[i] = [data[j] for j in range(N) if y[j] == i]  # 第j个样本属于第i类
```

b) 关于 μ_k 最小化 J ，保持 r_{nk} 固定：

根据分类计算新的均值向量。

如果没有任何均值向量更新，则退出循环。

```
for i in range(k):
    # 第i类中的样本C[j]
    if len(C[i]) == 0:
        continue
    sum = np.zeros(m)
    for j in range(len(C[i])):
        sum += C[i][j]
    new_mean = sum / len(C[i])  # 新的均值
    if abs(np.max(means[i]) - np.max(new_mean)) > math.exp(-10):
        means[i] = new_mean
        update = True  # 如果所有均值向量均未更新，则update=False，退出循环
```

3. 最后返回簇划分结果以及均值向量

```
return C, means
```

混合高斯模型

算法的原理

高斯混合模型可以看做是高斯分布的简单线性叠加，即

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

其中混合系数 π_k 满足

$$\pi_k = P(z_k=1) \qquad 0 <= \pi_k <= 1 \qquad \sum_{k=1}^{K} \pi_k = 1$$

K 维二元随机变量 z，采用‘1-of-K’的表示方法，其中一个特定的元素 z_k 等于 1，其余所有元素为 0，则有

$$p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

并且，给定 z 一个特定的值，x 的条件概率分布是一个高斯分布

$$p(\boldsymbol{x} \mid z_k = 1) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

也可以写成

$$p(\boldsymbol{x} \mid \boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

从而得到 x 的边缘概率分布，即对所有可能的 z 求和

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

用 $\gamma(z_k)$ 表示 $P(z_k = 1 \mid x)$ ，即观测到 x 后，z 的条件概率。$\gamma(z_k)$ 也可以被看做分量 k 对于“解释”观测值 x 的“责任”。则有

$$\begin{aligned} \gamma(z_k) \equiv p(z_k = 1 \mid \boldsymbol{x}) &= \frac{p(z_k = 1) p(\boldsymbol{x} \mid z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1) p(\boldsymbol{x} \mid z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_k)} \end{aligned}$$

若有数据集{x1, …, xN }，将其表示为 N×D 的矩阵 X。高斯混合模型的对数似然函数为

$$\ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

①令上式关于均值 μ_k 的导数等于 0，得到

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

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

其中 N_k 可以看做分配到聚类 k 的数据点的有效数量

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

②令其关于Σ_k 的导数等于 0，得到

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

每个数据点都有一个权值，权值等于对应的后延概率，分母为与对应分量相关联的数据点的有效数量。

③最后，我们关于混合系数 π_k 最大化对数似然函数。
这里我们考虑约束条件，要求混合系数之和为 1。
使用拉格朗日乘子法，最大化下式

$$\ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

令其关于 π_k 导数为 0 得

$$0 = \sum_{n=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda$$

整理得

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

第 k 个分量的混沌系数为 那个分量对于解释数据点的“责任”的平均值

为了寻找最大似然解，我们使用 EM 算法：

初始化均值 μ_k、协方差 $\boldsymbol{\Sigma}_k$ 、混合系数 π_k，计算对数似然函数的初始值

repeat:

E 步骤：

使用当前参数值计算“责任”

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

M 步骤：

1. 使用当前的“责任”重新估计参数

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

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

$$\pi_k^{\text{新}} = \frac{N_k}{N}$$

其中

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

2. 计算对数似然函数

$$\ln p(\boldsymbol{X} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

检查对数似然函数的收敛性。如果满足收敛的准则，break

算法的实现

```
def gmm_em(data, k):
    '''
    EM算法实现混合高斯模型
    :param data: 样本矩阵
    :param k: 类别数
    :return: 分类结果及各项参数
    '''
```

1. 初始化均值 μ_k、协方差Σ_k以及混合系数 π_k

```
n, m = data.shape
pi = [1 / k] * k
mu = [data[random.randint(0, n - 1)] for i in range(k)]
cov = [np.eye(m) for i in range(k)]
gamma = np.zeros((n, k))
pre = float('inf')
```

2. 进行迭代
   a) E 步骤
   使用当前的参数值计算责任 gamma
   这里的 gauss_prob() 用于计算高斯分布概率密度

```
# E 步骤
for i in range(n):
    temp = [pi[j] * gauss_prob(data[i], mu[j], cov[j]) for j in range(k)]
    sum_temp = sum(temp)
    for j in range(k):
        gamma[i][j] = temp[j] / sum_temp
```

   b) M 步骤
   使用当前的责任重新估计参数

```
# M 步骤
temp = [sum([gamma[i][j] for i in range(n)]) for j in range(k)]
for j in range(k):
    mu[j] = sum([gamma[i][j] * data[i] for i in range(n)]) / temp[j]
    cov[j] = sum([gamma[i][j] * np.dot((data[i] - mu[j]).reshape(m, 1), (data[i] - mu[j]).reshape(1, m))
                  for i in range(n)]) / temp[j]
    pi[j] = temp[j] / n
```

   c) 检查对数似然函数的收敛性
   比较前一次迭代和当前计算的对数似然函数的值，如果差值小于 math.exp(-10)，认为其收敛，退出迭代。

```python
# 检查对数似然函数的收敛性
log_lik_func = 0
for i in range(n):
    log_lik_func += math.log(sum([pi[j]*gauss_prob(data[i],mu[j],cov[j]) for j in range(k)]))
print(time, log_lik_func)
if abs(log_lik_func - pre) < math.exp(-10):
    break
else:
    pre = log_lik_func
```

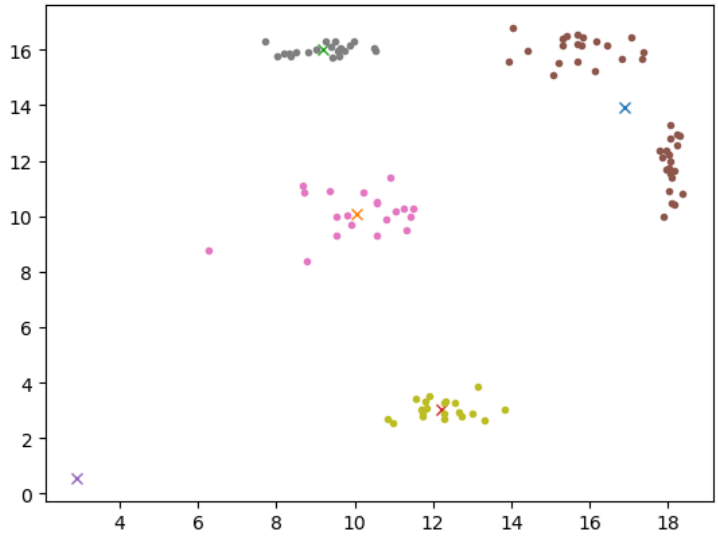
三、实验结果与分析

k-means 测试

高斯分布产生 k 个高斯分布的数据，每个分布有 n 个样本。
这里的高斯分布都是随机生成均值以及方差。最后返回样本集合。

```python
# 随机生成k类样本，每类样本有n个
def generate_data(k, n):
    x1, x2 = np.random.multivariate_normal([10, 10], [[1, 0], [0, 1]], n).T
    data = np.c_[np.array(x1).reshape(n, ), np.array(x2).reshape(n, )]
    for i in range(k - 1):
        mean = np.random.randint(2, high=20, size=2)
        cov = [[random.random(), 0], [0, random.random()]]
        x1, x2 = np.random.multivariate_normal(mean, cov, n).T
        X = np.c_[np.array(x1).reshape(n, ), np.array(x2).reshape(n, )]
        data = np.r_[data, X]
    return data
```

在这里我们取 k=5,2=20 进行测试，测试结果如下：



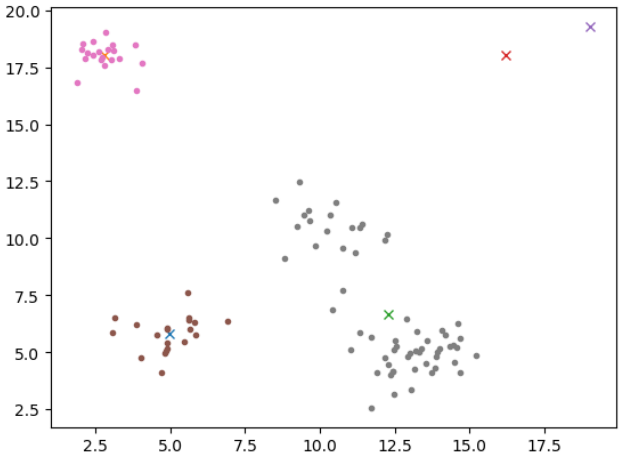

```
The size of C[0]:  40
The size of C[1]:  20
The size of C[2]:  20
The size of C[3]:  20
The size of C[4]:  0
```

图中原点表示样本，’X’ 表示均值向量。不同颜色表示不同聚类的划分结果。

我们可以观察到，五类样本被分成了 4 个簇。

不过尽管如此，输出结果还是比较让人满意，两类样本被归为一类，另外 3 类高斯分布的样本被很好地划分。这是可以理解的，这和初始值的选择也有一定的关系。

这只是典型测试样例之一。经过多次的测试我们可以观察到许多种情况。多数情况下 k-means 都能较好地完成聚类任务，个别情况下，性能有待提高（如下图所示，被分为了 3 类），这和生成数据的方法以及初始均值向量的选择有关

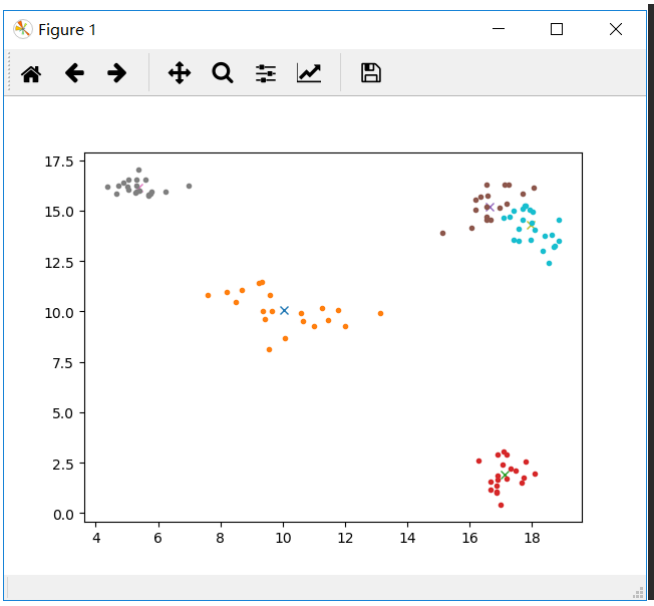


GMM 测试

人为生成的数据集

这里生成数据集的方法和在 K-means 中相同，生成 k 个高斯分布的数据，每个分布有 n 个样本。其中每个分布的均值和协方差都是随机生成。

生成的数据以及分类结果如下图所示：




```
The size of C[0]:  20
mu = [10.05201495 10.05818505]
cov = [[ 1.88382402 -0.52397044]
 [-0.52397044  0.72166995]]

The size of C[1]:  20
mu = [17.12360066  1.88712301]
cov = [[0.18301788 0.04222346]
 [0.04222346 0.47588764]]

The size of C[2]:  17
mu = [16.64822135 15.20612898]
cov = [[0.4147582  0.26421577]
 [0.26421577 0.51548418]]

The size of C[3]:  20
mu = [ 5.37172928 16.15388796]
cov = [[ 0.32186772 -0.02517387]
 [-0.02517387  0.10258152]]

The size of C[4]:  23
mu = [17.96395272 14.2789172 ]
cov = [[ 0.31726462 -0.29628386]
 [-0.29628386  0.85180788]]
```

可以看到，生成数据时，有两个高斯分布数据分布在了一起，但是尽管如此还是能很好地将其划分，从这一点来看，GMM 的性能要略优于 k-means 方法。不过这并不绝对。这和初值的选择有很大的关系。

UCI 数据集

这里使用的是 iris 数据集，其中包含 150 个样本，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。其中的一个种类与另外两个种类是线性可分离的，后两个种类是非线性可分离的。

测试结果如下图所示：

```
The size of C[0]:  50
mu = [5.006 3.418 1.464 0.244]
cov = [[0.121764 0.098292 0.015816 0.010336]
 [0.098292 0.142276 0.011448 0.011208]
 [0.015816 0.011448 0.029504 0.005584]
 [0.010336 0.011208 0.005584 0.011264]]

The size of C[1]:  45
mu = [5.9150259  2.77784881 4.20167053 1.29701236]
cov = [[0.27532052 0.09692843 0.18467938 0.05439964]
 [0.09692843 0.09264276 0.09113799 0.04299675]
 [0.18467938 0.09113799 0.20067534 0.06099778]
 [0.05439964 0.04299675 0.06099778 0.03200627]]

The size of C[2]:  55
mu = [6.54462282 2.94868952 5.47970156 1.98469899]
cov = [[0.38704635 0.09220754 0.30278578 0.06161751]
 [0.09220754 0.11033962 0.08427056 0.0560013 ]
 [0.30278578 0.08427056 0.32771779 0.0744624 ]
 [0.06161751 0.0560013  0.0744624  0.08576122]]
```

混合高斯模型的性能较好，基本分出了 3 个种类。然而这依赖于较好的初值。在此之前我尝试将均值设为随机，最终的生成结果经常会性能较差。

四、结论

k-means 测试结果较好，计算快算法简单。然而有些聚类可能重叠或者分布分散等原因会影响性能。

混合高斯模型是一种用法及其广泛的方法，可以模拟绝大部分概率分布现象，限制极少。然而其缺点极度依赖初值。使用者的经验判断可以影响模型的准确度

在 K-means 中，每个聚类中心的初始化都会影响聚类效果，同样的，GMM 对初值也很敏感。所以选择一个较好的初始值对两个算法都很重要。

GMM 依据的是数据点属于每个每类的概率，我们用最大似然方法去确定分类。我个人的看法是，k-means 是计算当前簇中所有元素的位置的均值，而 GMM 用概率进行描述数据点的分类，所以 GMM 要比 K-mean 性能更高。

五、参考文献

https://docs.scipy.org/doc/numpy/

https://en.wikipedia.org/wiki/K-means_clustering

https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm

七、附录：源代码（带注释）

见附件