基于K-Means与GMM模型实现小麦种子的聚类任务

1 问题描述

小麦是一种常见的农作物，因其具有良好的适应能力与物种多样性而成为当前主流的食用作物之一。由于小麦本身的物种多样性，不同的小麦种类能够适用于不同的下游产品的生产之中，为后续多样的小麦制品提供可靠的原材料。而除去本身有小麦自身基因所决定的不同种属，在传统的生物学分类方法中，通过X射线照射种子内核，分析其内核的不同特征，也能够有效的通过相关特征，分析总结出小麦类型。从而，通过分析X光照射下的小麦种子内核数据特征，我们能够区分出不同的小麦种子类别。

因此，在本文中，基于一个波兰的小麦种子数据集，通过K-Means与GMM聚类模型对小麦种子进行聚类任务，根据不同种类小麦种子之间的内核特征信息，对小麦种子进行聚类区分。同时，通过K-Means算法初始化GMM模型的参数，测试模型参数初始化对于模型聚类结果之间的关系。

数据集来源： UCI Machine Learning Repository: seeds Data Set

2 算法策略

2.1 特征归一化

由于数据中各类特征的数值各不相同，甚至部分特征数据之间存在有数量级之间的区别，这一现象会导致个参数之间方差与均值之间的不平衡。从而，为避免以上问题，我们对特征进行缩放，使其位于同一范围内。特征归一化方法如下：

$$Z = (X - \mu)/\sigma$$

其中，μ为样本均值,σ为样本的标准差。

2.2 K-Means

2.2.1 K-Means的原理与损失函数

原理：对于给定的样本集，按照样本之间的距离大小，将样本集划分为K个簇。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。

损失函数：假设簇划分为$(C_1, C_2, \dots C_k)$，在聚类过程中，最小化损失函数E，直到质心不再改变位置。损失函数为平方误差E，即

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2^2$$

其中，μ_i为c_i的均值向量,即为质心。表达式为：

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

2.2.2 K-Means伪代码

```
Input: 数据data,聚类簇数k，样本数为m

Do:从data中随机选择k个样本作为初始均值向量
Repeat：
    令C_i为空集（i=1,2,...,k）
    for j in 1,2,...,m ： do
        计算当前样本与各个均值向量之间的距离
        根据距离最近的均值向量确定当前样本的簇种类
        将样本划归对应的簇中
    end for
    for i in 1,2,...,k:
        计算新的均值变量
        if 均值向量改变：
            更新均值向量
        else:
            end for
End

Output:簇划分
```

2.3 GMM模型与EM算法

2.3.1 GMM模型

高斯混合模型是将一个事物分解为若干的基于高斯概率密度函数（正态分布曲线）形成的模型，适用于数据内部有多个类别，每个类别的数据呈现不同的分布的数据集。

假设混合高斯模型由K个高斯模型组成(即数据包含K个类)，则GMM的概率密度函数如下：

$$p(x) = \sum_{k=1}^{K} p(k) p(x \mid k) = \sum_{k=1}^{K} \pi_k N\left(x \mid \mu_k, \sum k\right)$$

其中，$p(x \mid k) = N\left(x \mid \mu_k, \sum k\right)$是第k个高斯模型的概率密度函数，可以看成选定第K个模型后，该模型产生x的概率。$P(k) = \pi k$是第k个高斯模型的权重，满足$\sum_{k=1}^{K} \pi_k = 1$。

设有样本集$Y = y_1, y_2, \dots, y_N, p\left(y_n \mid \mu, \Sigma\right)$是高斯分布的概率分布函数。假设样本的抽样是独立的，则似然函数为

$$L(\mu,\Sigma)=L\left(y_1,y_2,\ldots,y_N;\mu,\Sigma\right)=\prod_{n=1}^{N}p\left(y_n;\mu,\Sigma\right)$$

为最大化似然函数，对数化似然函数，得到对数似然函数为：

$$\ln L(\mu,\Sigma,\pi)=\sum_{n=1}^{N}\ln\sum_{k=1}^{K}\pi_k N\left(y_n\mid\mu_k,\Sigma_k\right)$$

最大化对数似然函数，对对数似然函数求导，估计模型参数(μ,Σ,π)。由于模型参数中存在有Σ参数无法通过求导计算优化参数，故使用EM算法解决缺失数据的参数估计问题。

2.3.2 EM算法求解GMM模型

引入隐变量r_{jk}，若取值为1，则第j个观测变量来自第k个高斯分量，反之则取0。则对于每一个观测数据y_i都对应一个向量变量$\Gamma_j=（r_{j1},\ldots,r_{jk}）$，其中，$\sum_{k=1}^{K}r_{jk}=1$则有

$$p\left(\Gamma_j\right)=\prod_{k=1}^{K}\alpha_k^{r_{jk}}$$

其中，K为GMM高斯分量的个数，α_k为第k个高斯分量的权值。

从而，对于观测数据y_i，在确认其属于哪个高斯分布之后（得到已知向量变量$\Gamma_j=（r_{j1},\ldots,r_{jk}）$的条件下测试数据的后验概率分布），服从概率分布为：

$$p\left(y_i\mid\Gamma_j;\theta\right)=\prod_{k=1}^{K}N(y_j\mid\mu_k,\Sigma_k)^{r_{jk}}$$

结合$\Gamma_j=（r_{j1},\ldots,r_{jk}）$的先验分布并对似然函数取对数，得到对数似然函数为：

$$\ln p\left(y,\Gamma_j;\theta\right)=\sum_{j=1}^{N}\sum_{k=1}^{K}\left(r_{jk}\ln\theta_k+r_{jk}\ln N\left(y_j\mid\mu_k,\Sigma_k\right)\right)$$

在引入隐变量之后，对高斯分布各参数进行估计。

首先，对$\ln N\left(y_j\mid\mu_k,\Sigma_k\right)$进行展开，得到：

$$\ln N\left(y_j\mid\mu_k,\Sigma_k\right)=-\frac{D}{2}\ln(2\pi)-\frac{1}{2}\ln|\Sigma_k|-\frac{1}{2}(y_j-\mu_k)^T\Sigma_k^{-1}\left(y_j-\mu_k\right)$$

假设已知隐变量r_{jk}的取值条件下，基于上述的似然函数对μ_k与Σ_k求偏导，得到：

$$\mu_k=\frac{\sum_{j=1}^{N}\sum_{k=1}^{K}r_{jk}y_j}{\sum_{j=1}^{N}\sum_{k=1}^{K}r_{jk}}$$

$$\Sigma_k=\frac{\sum_{j=1}^{N}\sum_{k=1}^{K}r_{jk}\left(y_j-\mu_k\right)\left(y_j-\mu_k\right)^T}{\sum_{j=1}^{N}\sum_{k=1}^{K}r_{jk}}$$

利用拉格朗日乘数法，结合似然函数与约束条件($\sum_{k=1}^{K}\alpha_k=1$)，得到：

$$\alpha_k=\frac{\sum_{j=1}^{N}r_{jk}}{-\lambda}$$

将上式的左右两边分别对k=1,2,...,K求和，得到$\lambda=-N$。带入上式：

$$\alpha_k=\frac{\sum_{j=1}^{N}r_{jk}}{N}$$

3 代码实现

3.1 数据集导入

利用$pandas$库中的文件操作函数，读入数据集。

```
import numpy as np
import pandas as pd

def make_data(file_path):
    df = pd.read_csv(file_path, sep='    ', header=None)
    df.columns = ['area', 'perimeter', 'compactness', 'kernel_len',
'kernel_width', 'asy_cof', 'k_groove_len', 'label']
    df = np.array(df)
    data = df[:, :-1]
    label = df[:, -1] - 1
    return data, label

train_data, train_label = make_data(
    r'.\seeds_dataset.txt')
```

3.2 特征归一化

对导入数据中的特征参数归一化，代码实现如下所示：

```
# 特征归一化
mu = []
std = []


def normalize(data):
    for i in range(0, data.shape[1] - 2):
        data[:, i] = ((data[:, i] - np.mean(data[:, i])) / np.std(data[:, i]))
        mu.append(np.mean(data[:, i]))
        std.append(np.std(data[:, i]))

#归一化特征参数
train_data = normalize(train_data)
```

3.3 K-Means

基于2.2中有关K-Means聚类算法的公式推导与伪代码，实现K-Means如下所示：

```
class kMeans:
    """
    本模型中使用k-means有两个用途：（1）比较与gmm模型之间的聚类准确性，（2）为gmm初始化参数
    """

    def __init__(self, data, epoch, k):
        self.data = data
        self.k = k
        self.epoch = epoch
        self.center = None

    def init_center(self):
```

```
        data_num, feature_num = self.data.shape
        self.center = np.zeros((self.k, feature_num))
        # 从数据集中随机确定K个点
        rand_id = np.random.randint(0, data_num, self.k)
        for i in range(self.k):
            self.center[i, :] = self.data[rand_id[i], :]
        return self.center

    def cluster_step(self, center):
        data_num, feature_num = self.data.shape
        temp_result = np.zeros(data_num)
        for i in range(data_num):
            min_dis = 10000
            for j in range(self.k):
                temp_dis = np.sum((self.data[i, :] - center[j, :]) ** 2)
                if temp_dis < min_dis:
                    min_dis = temp_dis
                    temp_result[i] = j
        return temp_result

    def renew_center(self, temp):
        data_num, feature_num = self.data.shape
        center = np.zeros((self.k, feature_num))
        for i in range(self.k):
            temp_class = np.where(temp == i)
            center[i, :] = (np.sum(self.data[temp_class, :], axis=1) /
len(temp_class[0])).ravel()
        return center

    def train(self):
        init_center = self.init_center()
        center = init_center
        final_result = []
        for i in range(self.epoch):
            final_result = self.cluster_step(center)
            center = self.renew_center(final_result)
        return final_result
```

3.4 GMM与EM算法

基于2.3中有关GMM聚类算法的公式推导与EM算法的参数优化过程，实现GMM+EM算法如下所示：

```
class GMM:
    """
    参数解释：k为高斯分布的个数（聚类数量）
    mean:高斯分布的均值向量
    cov：高斯分布的协方差矩阵
    data_dim:数据的特征数量
    data_num：数据的数量
    weight：高斯分布的初始权重
    k_init:是否需要用k-means初始化。
    高斯混合模型参数初始化中，可以选择用k-means初始化高斯分布，
    也可以随机生成高斯分布用于后续训练，默认随机初始化。
    """

    def __init__(self, data, Epoch, K, k_init=False, Delta=0.000001):
```

```
        self.Data = data
        self.Epoch = Epoch
        self.Data_num = data.shape[0]
        self.Feature_num = data.shape[1]
        self.K = K
        self.Delta = Delta
        self.k_init = k_init

        if not self.k_init:
            self.weight = np.random.rand(self.K)
            self.weight /= np.sum(self.weight)

            self.mean = []
            for i in range(self.K):
                temp_mean = np.random.rand(self.Feature_num)
                self.mean.append(temp_mean)

            self.cov = []
            for i in range(self.K):
                temp_cov = np.random.rand(self.Feature_num, self.Feature_num)
                self.cov.append(temp_cov)
        else:
            init_result = kMeans(self.Data, 1000, 3).train()
            store = defaultdict(list)
            for num_id, label in enumerate(init_result):
                store[label].append(num_id)
            self.weight = []
            self.mean = []
            self.cov = []
            for num_id in store.values():
                temp_data = self.Data[num_id]
                self.weight.append(len(num_id) / self.Data_num)
                self.mean.append(temp_data.mean(axis=0))
                self.cov.append(np.cov(temp_data.T))
            self.mean = np.array(self.mean)
            self.weight = np.array(self.weight)
            self.cov = np.array(self.cov)

    def get_guass(self, x, mean, cov):
        """
        这是自定义的高斯分布概率密度函数
        :param x: 输入数据
        :param mean: 均值数组
        :param cov: 协方差矩阵
        :return: x的概率
        """
        dim = np.shape(cov)[0]
        # cov的行列式为零时的措施
        covdet = np.linalg.det(cov + np.eye(dim) * 0.001)
        covinv = np.linalg.inv(cov + np.eye(dim) * 0.001)
        xdiff = (x - mean).reshape((1, dim))
        # 概率密度
        prob = 1.0 / (np.power(np.power(2 * np.pi, dim) * np.abs(covdet), 0.5))
* \
               np.exp(-0.5 * xdiff.dot(covinv).dot(xdiff.T))[0][0]
        return prob

    def train(self):
```

```
        temp_p = 0
        last_p = 1
        # guass_p表示第i个样本属于第k类高斯分布的概率
        guass_p = [np.zeros(self.K) for i in range(self.Data_num)]

        # 利用EM算法优化模型参数，终止条件（1）对数似然概率不再改变；（2）达到最大训练次数
        for i in range(self.Epoch):
            if abs(temp_p - last_p) < self.Delta:
                break
            else:
                last_p = temp_p
                # E-step:估计混合高斯模型中的参数
                for j in range(self.Data_num):
                    post_weight = [self.weight[k] * self.get_guass(self.Data[j],
self.mean[k], self.cov[k])
                                   for k in range(self.K)]
                    post_weight = np.array(post_weight)
                    guass_p[j] = post_weight / np.sum(post_weight)

                # M-step:最大化估计参数
                for k in range(self.K):
                    # 计算N个样本中属于第k类的数量，并更新高斯分布的概率、均值与协方差矩阵
                    num_k = np.sum([guass_p[q][k] for q in
range(self.Data_num)])
                    self.weight[k] = (1.0 * num_k) / self.Data_num
                    self.mean[k] = (1.0 / num_k) * np.sum([guass_p[q][k] *
self.Data[q] for q in range(self.Data_num)],
                                                          axis=0)
                    temp_diff = self.Data - self.mean[k]
                    self.cov[k] = (1.0 / num_k) * np.sum(
                        [guass_p[q][k] * temp_diff[q].reshape((self.Feature_num,
1)).dot(
                            temp_diff[q].reshape((1, self.Feature_num))) for q
in range(self.Data_num)], axis=0)

                # 计算此时的对数似然概率，若小于预设值Delta，则符合条件1，结束
                temp_p = []
                for n in range(self.Data_num):
                    cur_p = [np.sum(self.weight[k] *
self.get_guass(self.Data[n], self.mean[k], self.cov[k])) for k in
                             range(self.K)]
                    if cur_p != 0:
                        cur_p = np.log(np.array(cur_p))
                        temp_p.append(cur_p)
                temp_p = np.sum(temp_p)

        # 输出聚类结果
        for i in range(self.Data_num):
            guass_p[i] = guass_p[i] / np.sum(guass_p[i])
        result = [np.argmax(guass_p[i]) for i in range(self.Data_num)]
        return result
```

3.5 模型性能检验

对输出的聚类结果与数据的标签之间进行比较，计算相关标签的数据被聚类正确的比例，得到模型的聚类准确性。代码实现如下：

```
def acc(pred):
    pred1 = pred[:70]
    pred2 = pred[70:140]
    pred3 = pred[140:]
    t_1 = [np.sum(pred1 == 0), np.sum(pred1 == 1), np.sum(pred1 == 2)]
    t_2 = [np.sum(pred2 == 0), np.sum(pred2 == 1), np.sum(pred2 == 2)]
    t_3 = [np.sum(pred3 == 0), np.sum(pred3 == 1), np.sum(pred3 == 2)]
    t = np.array([t_1, t_2, t_3])
    temp = []
    print(t)
    for i in range(3):
        max_index = list(np.unravel_index(t.argmax(), t.shape))
        temp.append(max_index)
        t[max_index[0], :] = -1
        t[:, max_index[1]] = -1
    s = 0
    for i in temp:
        print(i)
        s += np.sum(pred[i[1]*70:(i[1]*70+70)] == i[0])
    s /= 210
    return s
```

4 模型表现

在小麦种子类型的聚类任务中，我们选择了不同品种小麦种子的区域、周长、压实度、籽粒长度、籽粒宽度、不对称系数、籽粒腹沟长度共7个特征信息以及类别数据（共三个类别），希望通过以上的X光显影的小麦内核特征信息完成小麦种子的聚类任务。

4.1 K-Means聚类表现

在K-Means模型的聚类任务的实现中，我们选择随机化初始参数，即随机化初始的聚类中心，探究随着训练轮数变化对模型聚类准确性的影响。在实验的过程中，设置训练轮数为[0, 150]，间隔数为15，训练模型。随着训练轮数的增加，K-Means模型聚类准确率与训练的轮数之间没有出现单调增加关系，而出现了上下剧烈波动的情况，最终在训练75轮之后，K-Means模型聚类准确率收敛，模型聚类小麦种子数据集的**准确率为0.935**。

推测在75轮前模型聚类准确率上下剧烈波动的原因可能是：K-Means模型选择的初始聚类中心为**随机选择**的样本点，在迭代次数较小的情况下，可能由于模型参数初始化的问题导致无法达到收敛。

4.2 GMM模型聚类表现

在GMM模型的聚类任务之中，我们选择随机化初始参数，即随机化初始高斯分布的概率、均值与协方差矩阵，探究随着训练轮数变化对模型聚类准确性的影响。在实验的过程中，设置训练轮数为[0, 300]，间隔数为25，训练模型。随着训练轮数的增加，GMM模型聚类准确率与训练的轮数之间没有出现单调增加关系，而出现了上下剧烈波动的情况，且直到最后也没有达到收敛的情况。模型聚类小麦种子数据集的**准确率为0.682**。

与4.1中的模型在训练轮数较少的情况相似的是，模型训练的准确率与训练轮数之间没有正相关，其最主要的原因还是：GMM模型中的高斯分布的初始化为随机初始化，导致模型的收敛与否直接取决于随机参数选择的好坏，以至于在长轮数训练之中模型的聚类准确性也没有达到收敛。此外，GMM模型的随机初始化参数较多，从而即使在较大的训练轮数之下也没有实现收敛。

4.3 基于K-Means初始化参数的GMM模型的聚类表现

鉴于4.2中GMM模型受到的参数初始化的显著影响，在初始化GMM的模型参数中，选择先用K-Means模型聚类数据，将K-Means的聚类结果作为GMM模型的参数初始化条件，优化GMM模型的参数选择。

在实验的过程中，选择K-Means模型先预训练100轮，将模型聚类结果传递给GMM模型初始化参数设置，设置训练轮数为[0, 300]，间隔数为50，训练模型。当采用K-Means模型优化GMM模型的参数之后，GMM模型的聚类准确率随着模型训练轮数的增加而增加，在训练50轮之后收敛，模型聚类小麦种子数据集的**准确率为0.964**。可以看到，在利用K-Means模型优化初始参数之后，GMM模型的稳定性增加，并快速达到了收敛，模型预测的准确率也有了极大的提升。

5 总结与展望

在本文中，基于一个波兰的小麦种子数据集，通过K-Means与GMM聚类模型对小麦种子进行聚类任务，根据不同种类小麦种子之间的内核特征信息，对小麦种子进行聚类区分。通过对模型的实验，可以得出以下结论：

- 在随机初始化模型参数的情况下，K-Means模型与GMM模型聚类准确率与训练的轮数之间没有出现单调增加关系，而出现了上下剧烈波动的情况。在达到一定的训练轮数之后，K-Means模型可以达到收敛，模型准确率为0.935；而GMM模型由于初始化的参数较K-Means更多，受到模型参数初始化的影响更大，模型较难达到收敛。
- 采用K-Means模型优化GMM模型的参数之后，GMM模型的聚类准确率随着模型训练轮数的增加而增加，GMM模型的稳定性增加，并快速达到了收敛，模型聚类小麦种子数据集的准确率为0.964，模型预测的准确率也有了极大的提升。这表明在机器学习的模型中，尤其是无监督模型中，参数的初始化选择与参数的设置直接影响到模型最终的性能好坏。

参考文献

[1]M. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, S. Zak, 'A Complete Gradient Clustering Algorithm for Features Analysis of X-ray Images', in: Information Technologies in Biomedicine, Ewa Pietka, Jacek Kawa (eds.), Springer-Verlag, Berlin-Heidelberg, 2010, pp. 15-24.