实验四：基于混合高斯模型的二分类

PB17051065 刘逸飞

实验目标

使用混合高斯模型解决二分类问题。利用给定的训练集训练高斯模型，再使用测试集测试训练效果。每个样本的特征是三维的，两类分别命名A、B，且测试时认为两个类的先验概率是相同的。

训练GMM模型使用EM迭代算法，EM算法的初始数值可以使用 K-means 聚类算法，分别对A、B类数据的训练样本进行聚类，并使用聚类样本的统计参数对A、B类的混合高斯模型进行初始化。

实验使用不同次数的EM算法训练模型，观察模型的准确率。此外还应对比不同混合数量对GMM模型准确率的影响情况。

实验原理

GMM混合高斯模型

高斯混合模型是一种生成模型，是多个高斯模型 $C_1, C_2, C_3 \dots$ 构成的（$N(X|\mu_k, \Sigma_k)$ 代表在对应正态分布中的样本的概率）。它可以看作一个随机变量的值是有 k 的概率由 C_k 对应的高斯分布产生的；当然也可以看作由多个高斯分布拟合成的一个任意的随机变量分布：

$$P(X) = \sum_{k=1}^{K} p_k N(X|\mu_k, \Sigma_k)$$

$$s.t. \sum_{k=1}^{K} p_k = 1$$

现在设观察的样本集：$X=(X_1,X_2,\ldots,X_n)$，对应隐变量集：$Z=(Z_1,Z_2,\ldots,Z_n)$，这里的隐变量是用于描述样本附带的某些没有观测到的属性的。例如，在GMM模型里，$Z1$ 表示样本 X_1 是由具体哪一个高斯分布产生的，可以是 $C_1,C_2,C_3\ldots$ 中的任何一个，这是没有观测到的。观测样本的完整集可以表示为：$(X,Z)=((X_1,Z_1),(X_2,Z_2),\ldots,(X_3,Z_3))$

（每个组合完整代表了一个样本，但应当注意样本是有多个特征的，即 X_1,Z_1 都是向量，表示它们取得某个值时用小写代表 $X_1=x_1$）

GMM的参数估计的解析解是不可行的。通过极大似然估计参数列表 $\theta=\{p_1,\mu_1,\Sigma_1,p_2,\mu_2,\Sigma_2\ldots\}$ ，可以得到：

$$\hat{\theta}_{ML}=arg\max_{\theta}\sum_{i=1}^{N}\log\sum_{k=1}^{K}p_kN(X|\mu_k,\Sigma_k)$$

它的极值点无法得到解析解，因为第二个求和号在 log 内部。

EM算法

EM算法是一种迭代算法，被用来代替求解上述参数估计问题

$$\theta^{(t+1)}=arg\max_{\theta}E_{z|x,\theta^{(t)}}\left[\log P(X,Z|\theta)\right]$$

$$Q(\theta,\theta^{(t)})\overset{def}{=}E_{z|x,\theta^{(t)}}\left[\log P(X,Z|\theta)\right]$$

记：

$$\theta^{(t+1)}=arg\max_{\theta}Q(\theta,\theta^{(t)})$$

（E step）下面我们带入GMM模型，得到GMM的迭代公式。简化上述 Q 表达式：

$$\begin{aligned}Q(\theta,\theta^{(t)})&=E_{z|x,\theta^{(t)}}\left[\log P(X,Z|\theta)\right]\\&=\sum_{Z}P(Z|X,\theta^{(t)})\log P(X,Z|\theta)\\&=\sum_{Z}\prod_{i=1}^{N}P(Z_i|X_i,\theta^{(t)})\log\prod_{i=1}^{N}P(X_i,Z_i|\theta)\\&=\sum_{Z}\prod_{i=1}^{N}P(Z_i|X_i,\theta^{(t)})\sum_{i=1}^{N}\log P(X_i,Z_i|\theta)\end{aligned}$$

取第二个连加号内的任一项：

$$
\begin{aligned}
R_i &= \sum_{Z} \prod_{i=1}^{N} P(Z_i|X_i,\theta^{(t)}) \cdot \log P(X_1,Z_1|\theta) \\
&= \sum_{Z_1,Z_2,\ldots} \prod_{i=2}^{N} P(Z_i|X_i,\theta^{(t)}) \cdot P(Z_1|X_1,\theta^{(t)}) \log P(X_1,Z_1|\theta) \\
&= \sum_{Z_1} P(Z_1|X_1,\theta^{(t)}) \log P(X_1,Z_1|\theta) \sum_{Z_2,Z_3,\ldots} \prod_{i=2}^{N} P(Z_i|X_i,\theta^{(t)}) \\
&= \sum_{Z_1} P(Z_1|X_1,\theta^{(t)}) \log P(X_1,Z_1|\theta)
\end{aligned}
$$

最后得到：

$$
Q(\theta,\theta^{(t)}) = \sum_{i=1}^{N} \sum_{Z_i} P(Z_i|X_i,\theta^{(t)}) \log P(X_i,Z_i|\theta)
$$

其中，根据高斯混合模型可以很容易得到下式。并且里面的 $p_{Z_i}, \mu_{z_i}, \Sigma_{z_i}$ 都是每次迭代中等待优化的参数，即 $\theta^{(t)} = \{p_1, \mu_1, \Sigma_1, p_2, \mu_2, \Sigma_2 \ldots\}$

$$
P(X_i,Z_i|\theta) = p_{z_i} N(X|\mu_{z_i},\Sigma_{z_i})
$$

（M step）下面我们计算 $P(Z_i|X_i,\theta^{(t)})$，它是在**给定混合模型参数**下的条件概率。在GMM模型中使用贝叶斯定理：

$$
\begin{aligned}
P(X_i|\theta^{(t)}) &= \sum_{k=1}^{K} p_k^{(t)} N(X_i|\mu_k^{(t)},\Sigma_k^{(t)}) \\
P(X_i,Z_i|\theta^{(t)}) &= P(Z_i|\theta^{(t)}) P(X_i|Z_i,\theta^{(t)}) = p_{Z_i}^{(t)} N(X_i|\mu_{Z_i}^{(t)},\Sigma_{Z_i}^{(t)}) \\
P(Z_i|X_i,\theta^{(t)}) &= \frac{P(X_i,Z_i|\theta^{(t)})}{P(X_i|\theta^{(t)})} = \frac{p_{Z_i}^{(t)} N(X_i|\mu_{Z_i}^{(t)},\Sigma_{Z_i}^{(t)})}{\sum_{k=1}^{K} p_k^{(t)} N(X_i|\mu_k^{(t)},\Sigma_k^{(t)})}
\end{aligned}
$$

将上面得到的两个条件概率带入到 Q 中

$$
\begin{aligned}
Q(\theta,\theta^{(t)}) &= \sum_{i=1}^{N} \sum_{Z_i} P(Z_i|X_i,\theta^{(t)}) \log P(X_i,Z_i|\theta) \\
&= \sum_{i=1}^{N} \sum_{Z_i} \frac{p_{Z_i}^{(t)} N(X_i|\mu_{Z_i}^{(t)},\Sigma_{Z_i}^{(t)})}{\sum_{k=1}^{K} p_k^{(t)} N(X_i|\mu_k^{(t)},\Sigma_k^{(t)})} \log[p_{z_i} N(X|\mu_{z_i},\Sigma_{z_i})]
\end{aligned}
$$

下面开始求解迭代参数的估计值，注意上式中分数的一项是不带任何待预测参数的，即之前被记为 $P(Z_i|X_i,\theta^{(t)})$ 的项。使用拉格朗日乘数法优化约束下的 p_k，对 μ_{z_i}, Σ_{z_i} 求极值。由带约束的拉格朗日优化得到：

$$
p_k^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} P(Z_i = C_k|X_i,\theta^{(t)})
$$

其中 $P(Z_i = C_k|X_i, \theta^{(t)})$ 使用贝叶斯定理求解：

$$P(Z_i = C_k|X_i, \theta^{(t)}) = \frac{P(Z_i = C_k|\theta^{(t)})P(X_i|Z_i = C_k, \theta^{(t)})}{P(X_i|\theta^{(t)})}$$

以上几个概率都是可求的，$P(X_i|\theta^{(t)})$ 是在当前迭代参数下产生样本的概率，$P(Z_i = C_k|\theta^{(t)})$ 可以直接从当前迭代参数 $\theta^{(t)}$ 中得到，即 $p_k^{(t)} = P(Z_i = C_k|\theta^{(t)})$。$P(X_i|Z_i = C_k, \theta^{(t)})$ 为第k个高斯子分布下 X_i 的分布，由 $\mu_k^{(t)}, \Sigma_k^{(t)}$ 确定。

同理，对 Q 求导可以得到均值和协方差的局部最优解。

$$\mu_k^{(t+1)} = \frac{\sum_{i=1}^{N} X_i P(Z_i = C_k|X_i, \theta^{(t)})}{\sum_{i=1}^{N} P(Z_i = C_k|X_i, \theta^{(t)})}$$

$$\Sigma_k^{(i+1)} = \frac{\sum_{i=1}^{N} (X_i - \mu_k^{(i+1)})(X_i - \mu_k^{(i+1)})^T P(Z_i = C_k|X_i, \theta^{(t)})}{\sum_{i=1}^{N} P(Z_i = C_k|X_i, \theta^{(t)})}$$

K-means 聚类算法

K-means 是聚类算法的一种，同样运用了迭代优化的思路。其基本过程是：先任意将样本划分为目标聚类数个聚类中，然后重复进行下面迭代操作：

1. 对于每个聚类，计算样本的质心点（欧氏距离最小点）并标记。
2. 对于每个样本，将其重新划分为距离它欧氏距离最近的中心对应的聚类中。

K-means 算法的收敛性很好。但其结果依赖于聚类划分的起始方法。本次实验采用随机初始划分的方式，随机产生初始聚类中心。

实验代码说明

实验代码有以下文件：

`Kmeans_display` 展示K-means 算法的迭代效果

`Kmeans_init` 使用K-means对GMM模型初始化

`TestPlatform` EM算法效果测试

`main` EM算法

下面仅说明EM算法核心部分的代码，其余代码均有完整注释。

混合高斯分类器 `GaussianMixClassifier` 的定义如下。记N为训练样本数量，M为样本属性数量（本次实验为3），C为分类数量（本次实验为2），K为混合高斯模型的混合数量。

`Means` 是一个大小为 C 的列表，每个对象同样为大小为 K 的列表，K 列表每个对象为大小为 M 的 array，记录了高斯模型的均值。

`Covar,pro` 同理，分别记录模型的协方差矩阵和GMM的模型构成参数（即每个高斯分布占比）

```
    def __init__(self):
        # denote: N:Num of samples  M:Num of attributes  K:
Num of mix models
        # Means: the mean of distribute
        # Covar: the Covariance of distribute
        # Pro: the ratio of a single gaussian component
        # Rpoch: iteration times
        self.Means = None # C list - K list - M array
        self.Covar = None # C list - K list - M*M array
        self.Pro = None # C list - k list
        self.K_MulNum = 0 # K
        self.M_AttrNum = 3 # M=3
        self.C_ClassNum = 2 # C=2
        self.Epoch = 0
```

分类器有以下方法：

```
def Initalise(self,means,covar,pro,k,epoch=3)
# 初始化分类器
def GaussianCal(self, attr, means, covar)
# 计算高斯分布概率
def MixGaussianCal(self, attr, tag)
# 计算高斯混合模型概率
def SingleIter(self,dataset,tagset)
# EM算法单次迭代
def SingleIter_TrainSingleType(self,dataset,tag)
# EM算法在A类模型或B类模型单次迭代
def TrainModel(self,dataset,tagset)
# 训练模型
def TestSamples(self,dataset,labelset)
# 测试模型
```

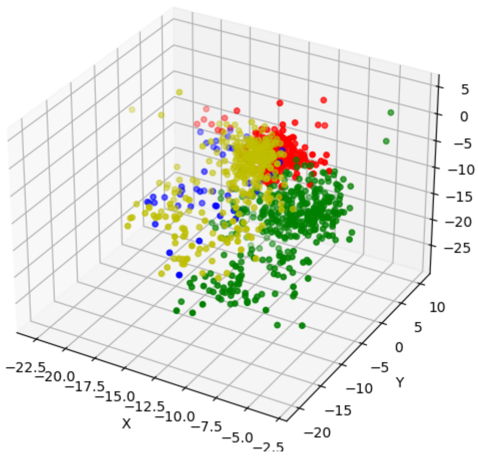
`SingleIter_TrainSingleType` 函数如下，`Pro_nk` 即为 $P(Z_i = C_k|X_i,\theta^{(t)})$ 的计算值。通过 $P(Z_i = C_k|X_i,\theta^{(t)})$ 求和得到 $p_k^{(t+1)} = \frac{1}{N}\sum_{i=1}^{N} P(Z_i = C_k|X_i,\theta^{(t)})$ ，在程序中记录为 `Pro_Sum` ，之后就可以使用上述推导的估计公式分别对 `self.Pro,self.Means,self.Covar` 进行估计。

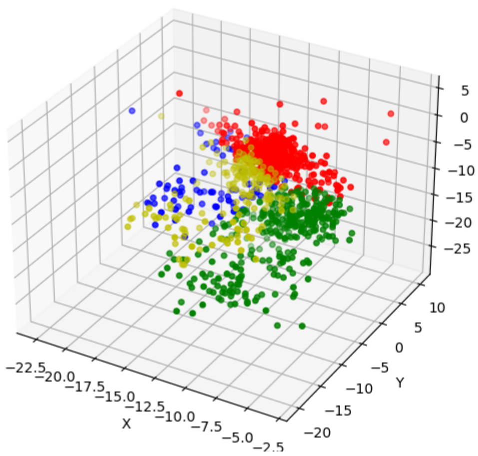
```
    def SingleIter_TrainSingleType(self,dataset,tag):
        # training mixture model for A (tag = 1) or B(tag = 2)
        # dataset: [ [],[] ]
        Pro_nk = []
```

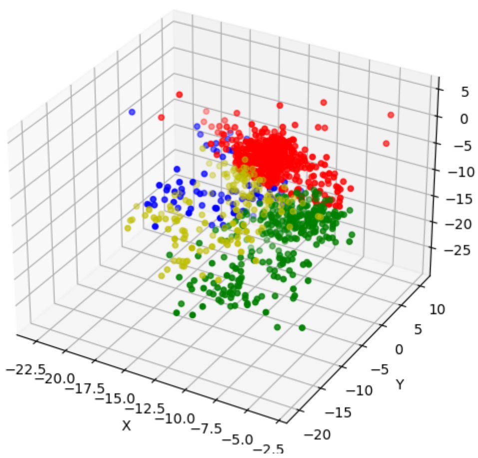
```
        for n in range(len(dataset)):
            Pro_nk_TempRow = []    # K list
            for k in range(self.K_MulNum):
                Pro_nk_TempRow.append(self.Pro[tag-1][k] *
self.GaussianCal(dataset[n], self.Means[tag-1][k],
self.Covar[tag-1][k] ) / self.MixGaussianCal(dataset[n], tag)
)
            Pro_nk = Pro_nk + [Pro_nk_TempRow]
        Pro_Sum = list(sum( np.array(Pro_nk)))   #  K list

        # estimate Pro
        self.Pro[tag-1] = list( np.array(Pro_Sum) / len(
dataset ) )  # type  K array

        # estimate Mean
        MeansTemp = []
        for k in range(self.K_MulNum):
            FracSum = np.zeros([self.M_AttrNum])
            for i in range(len(dataset)):
                FracSum = FracSum + np.array( dataset[i] ) *
Pro_nk[i][k]
            MeansTemp = MeansTemp + [FracSum / Pro_Sum[k]]
        self.Means[tag-1] = MeansTemp

        # estimate Cov
        CovarTemp = []
        for k in range(self.K_MulNum):
            FracSum =
np.zeros([self.M_AttrNum,self.M_AttrNum])
            for i in range(len(dataset)):
                FracSum = FracSum + np.multiply(
(np.array(dataset[i]) - self.Means[tag-1]
[k]).reshape(-1,1), (np.array(dataset[i]) - self.Means[tag-1]
[k]) ) * Pro_nk[i][k]
            CovarTemp = CovarTemp + [FracSum / Pro_Sum[k]]
        self.Covar[tag-1] = CovarTemp

        print("train",chr(64+tag),"finish")
```

实验结果

聚类初始化 先通过 `Kmeans_display.py` 展示 K-means 的聚类结果，下面是A 类训练数据设置聚类数量为4时，分别进行1，2，3次迭代后聚类的结果。





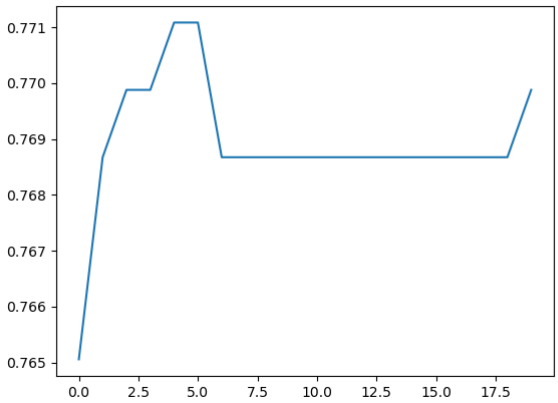


EM算法对GMM训练　在 `main.py` 中对GMM模型进行训练并检测测试集准确率，下面的例子给出了高斯混合数为4，进行2次迭代的结果：

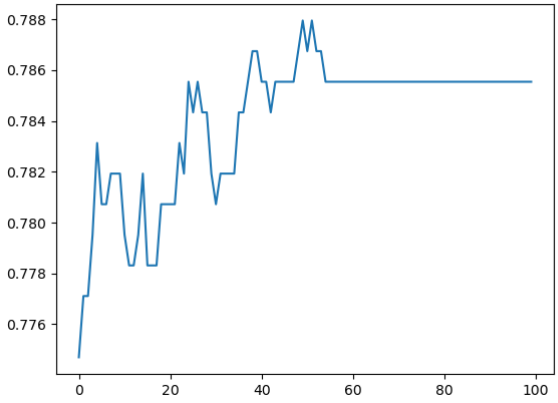
```
Mixture NUM: 4 Epoch: 2
loading trainset samples: 1891
set ClassA num: 1045
set ClassB num: 846
start training model...
---------- start ----------
round: 1
train A finish
train B finish
---------- round:  1  train finish ----------
round: 2
train A finish
train B finish
---------- round:  2  train finish ----------
test on given set...
load samples: 830
test Set:  830 correct: 638 Cor Rate: 0.7686746987951807

进程已结束，退出代码为 0
```

迭代次数对模型准确率的影响　固定GMM模型混合数，在 `TestPlatform` 中对同一个初始值初始化的GMM模型进行迭代，每次迭代后计算正确率，再继续迭代至收敛，得到在测试集的正确率如下：

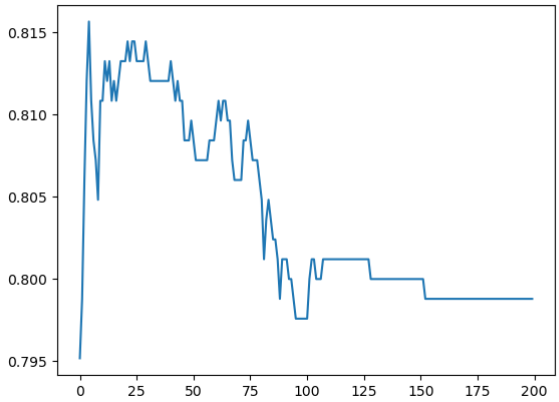
M=2时：



M=4时：



M=8时：



不同混合数对正确率的影响 根据上述测试结果，在不同混合数下，设置不同的迭代次数。多次使用不同的初值检测正确率结果如下：

M	$TEST1$	$TEST2$	$TEST3$
2	77.3%	76.6%	77.1%
4	78.8%	80.0%	79.2%
8	81.5%	80.0%	81.9%

对比与总结

- 混合数量对正确率有较明显的影响。GMM模型的特点是理论上可以模拟任何概率分布，可以认为混合数越高，越能贴合实际概率分布。模型的准确度自然越高。
- 关于迭代次数的影响：

  M=2时，在迭代过程中的正确率变化不大（< 1%），并且在测试集上的正确率表现为先稳定提升，到达某个最高点后正确率反而下降。这解释为M=2时模型较为简单，收敛速度很快。并且随着迭代次数的上升出现了过拟合现象，即在测试集表现下降而在训练集表现良好。5次迭代内一般就能得到最优的表现能力，正确率约为77%。

  M=4时，模型在很长时间后才出现收敛，这是因为模型更为复杂，需要更长时间才能收敛。同时达到的测试效果自然比M=2时更好，迭代次数对模型正确率的提升也更高。多次测试后结果为60次迭代左右正确率达到最高，正确率约为79%。

  M=8时，经过多次测试结果为50次左右正确率达到最高，正确率约为81.5%。
- EM算法的迭代次数受初始值的影响很大。初始值不理想对模型迭代次数要求高。从不同混合数对正确率的影响的结果可以看出，不同的初始值实行EM迭代，迭代次数随正确率的变化很大，难以确定最优时的迭代次数。