机器学习第三次

郭英明

2020 年 11 月 22 日

1

1.1. 推导三硬币模型的 EM 算法中隐变量后验分布的计算公式以及参数更新公式。

模型中参数为 $\theta = (\pi, p, q)$；随机变量 $X, Z \in \{0,\ 1\}$，其中 X 表示观测变量结果，Z 表示隐变量结果// **隐变量的后验分布：**

$$P(x, z|\theta) = \pi^z (1-\pi)^{1-z} p^{zx} (1-p)^{z(1-x)} q^{(1-z)x} (1-q)^{(1-z)(1-x)}$$
$$P(x|\theta) = \pi p^x (1-p)^{1-x} + (1-\pi) q^x (1-q)^{1-x}$$

根据贝叶斯公式：

$$P(z|x, \theta) = \frac{P(x,z|\theta)}{\sum\limits_z P(z|\theta) P(x|z,\theta)}$$

带入公式有：

$$P(z|x, \theta) = \frac{\pi^z (1-\pi)^{1-z} p^{zx} (1-p)^{z(1-x)} q^{(1-z)x} (1-q)^{(1-z)(1-x)}}{\pi p^x (1-p)^{1-x} + (1-\pi) q^x (1-q)^{1-x}}$$

参数更新公式：

EM 算法中 t 步时，

$$Q^{(t)}(z) = P(z|x, \theta^{(t)})$$

M 步中，

$$
\begin{aligned}
\theta^{(t+1)} &= \arg\max_{\theta} J(\theta, Q^{(t)}(z)) \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{z} P(z|x,\theta) log(P(x,z|\theta)) \\
&= \arg\max_{\theta} \sum_{i=1}^{n} (\frac{\pi^{(t)(p^{(t)x_i})(1-p^{(t)})^{1-x}} log(\pi p_x (1-p)^{1-x})}{P(x|\theta^{(t)})} \\
&+ \frac{(1-\pi^{(t)})(q^{(t)x})(1-q^{(t)})^{1-x} log((1-\pi) q^x (1-q)^{1-x})}{P(x|\theta^{(t)})})
\end{aligned}
$$

令

$$
\begin{aligned}
a^{(t)} &= \frac{\pi^{(t)}(p^{(t)})^x (1-p^{(t)})^{1-x}}{\pi^{(t)}(p^{(t)})^{1-x} + (1-\pi^{(t)})(q^{(t)})^x (1-q^{(t)})^{1-x}} \\
b^{(t)} &= \frac{\pi^{(1-(t)}(q^{(t)})^x \ (1-q^{(t)})^{1-x}}{\pi^{(t)}(p^{(t)})^{1-x} + (1-\pi^{(t)})(q^{(t)})^x (1-q^{(t)})^{1-x}} \\
F(\theta) &= \sum_{i=1}^{n} a \log(\pi p^x (1-p)^{1-x}) + b \log((1-\pi) q^x (1-q)^{1-x})
\end{aligned}
$$

$$
\theta^{(t+1)} = \arg\max_{\theta} \sum_{i=1}^{n} (a^t \log(\pi p^x (1-p)^{1-x}) + b^t \log((1-\pi) q^x (1-q)^{1-x}))
$$

易得 $F(\theta)$ 是关于 π, p, q 的凹函数，对 π, p, q 进行一阶求导有

$$
\begin{aligned}
\pi &= \frac{1}{n} \sum_{i=1}^{n} a^{(t)} \\
p^{t+1} &= \frac{\sum_{i=1}^{n} a^{(t)} x}{\sum_{i=1}^{n} a^{(t)}} \\
q^{t+1} &= \frac{\sum_{i=1}^{n} b^{(t)} x}{\sum_{i=1}^{n} b^{(t)}}
\end{aligned}
$$

1.2. 推导高斯混合模型参数更新公式。
似然函数：

$$
\begin{aligned}
L(\theta) &= \prod_{i=1}^{n} P(x|\theta) \\
&= \prod_{i=1}^{n} \sum_{z} P(x,z|\theta)
\end{aligned}
$$

由 Jensen 不等式有

$$
\begin{aligned}
LL(\theta) = &\sum_{i=1}^{n} \log \sum_{z} P(x,z|\theta) \\
&\sum_{i=1}^{n} \log(\sum_{z} Q(z) \frac{P(x,z|\theta)}{Q(z)}) \\
&\geq \sum_{i=1}^{n} \sum_{z} Q(z) \log(\frac{P(x,z|\theta)}{Q(z)})
\end{aligned}
$$

令 $J(\theta, Q(z)) = \sum_{i=1}^{n} \sum_{z} Q(z) \log(\frac{P(x,z|\theta)}{Q(z)}), J(\theta, Q(z))$ 为 $Q(z)$ 的下界，则可通过优化 $j(\theta, Q(z))$ 优化 $LL(\theta)$
E 步：

$$
Q^t(z) = P(z=k|x,\theta^t) = \gamma^t
$$

M 步：

$$
\begin{aligned}
\theta^{\theta+1} &= \arg\max_{\theta} J(\theta, Q^{(t)}(z)) \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma^t (\log(P(z=k|\theta)) + \log(x|z=k,\theta)) \\
&= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma^t \frac{1}{2} \log|\sum_{k}| + \frac{1}{2}(x-u)^T \sum_{k}^{-1}(x-u) - log\alpha
\end{aligned}
$$

令

$$G(\theta) = \gamma^t \frac{1}{2} \log |\sum_k| + \frac{1}{2}(x-u)^T \sum_k^{-1}(x-u) - log\alpha$$

$$\frac{\partial G}{\partial u} = 0$$

有

$$u^{t+1} = \frac{\sum_{i=1}^n \gamma^t x}{\sum\limits_{i=1}^n \gamma^t}$$

令

$$\frac{\partial G}{\partial \sum\limits_k} = 0$$

有

$$\sum_k^{t+1} = \frac{\sum\limits_{i=1}^n \gamma^t (x-u^{t+1})(x-u^{t+1})^T}{\sum\limits_{i=1}^n \gamma^t}$$

其中 $u^{t+1} = \frac{\sum\limits_{i=1}^n \gamma^t x}{\sum\limits_{i=1}^n \gamma^t}$ 优化问题可以转化为

$$\max_{\alpha_1,\cdots,\alpha_k} \sum_{i=1}^n \sum_{k=1}^K \gamma^t log\alpha_k$$

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

写出拉格朗日函数：

$$\sum_{i=1}^n \sum_{k=1}^K \gamma^t log\alpha_k - \lambda \sum_{k=1}^K (\alpha_k - 1)$$

对参数求导最后可得：

$$\alpha_k^{t+1} = \frac{1}{n}\sum_{i=1}^{n}\gamma^t, k = 1, 2, \cdots, K$$

2 高斯混合模型-EM

2.1 自己实现

```

# -*- coding: utf-8 -*-

import numpy as np
from tqdm import tqdm
from sklearn.mixture import GaussianMixture

# EM算法指定的类别个数
class_number = 3
#EM算法的迭代次数
 iterations  = 50

#正态分布的概率密度函数
def pdf(x,mu,sigma):
    x = np.mat(x).reshape(2,1)
    mu = np.mat(mu).reshape(2,1)
    sigma = np.mat(sigma).reshape(2,2)
    coef = 1/(np.sqrt((2*np.pi)**len(x)*np.linalg.det(sigma)))
    exp = np.exp(-0.5*(x-mu).T*sigma.I*(x-mu))
    prob = coef*exp
    return prob[0,0]

#生成2*2的协方差矩阵
def sys_matrix():
    #主对角线元素
```

```python
    ele = np.random.uniform(5,20,2)
    #不在对角线上的元素一定比对角线上的小
    b = np.random.uniform(0,min(ele),1)
    arr = np.array([[ele[0], b[0]],[b[0], ele[1]]])
    return arr

#给定的3组参数
mean1 = [3,1]
cov1 = [[1,-0.5],[-0.5,1]]

mean2 = [8,10]
cov2 = [[2,0.8],[0.8,2]]

mean3 = [12,2]
cov3 = [[1,0],[0,1]]

#各自生成300个样本
values1 = np.random.multivariate_normal(mean1, cov1, 300)
values2 = np.random.multivariate_normal(mean2, cov2, 300)
values3 = np.random.multivariate_normal(mean3, cov3, 300)
#拼接成900个样本
values = np.r_[values1,values2,values3]


#随机生成初始值
mean_list = [np.random.uniform(1,3,2) for i in range(class_number)]
sigma_list = [sys_matrix() for i in range(class_number)]
alpha = np.random.uniform(1,100,class_number).reshape(1,class_number)
alpha = alpha/np.sum(alpha)



for k in tqdm(range(iterations)):
```

```
    #计算新类别所属概率
    '''
    gamma是一个矩阵，每一行表示特定的样本分别归属某一总体的概率，
    例如gamma[0,0]表示第一个样本，隶属第一个总体概率，注意此处第一个总体是名义上的第一个总体
    不是生成样本时的第一个总体
    '''
    log_likehood = 0
    gamma = np.zeros((1,class_number))
    #遍历每一个样本，求分别归属每一类的概率
    for i in range(len(values)):
        #pdf是上文中定义的求多维高斯分布的概率密度函数的函数
        temp = np.array([pdf(values[i ,:], mean_list[j], sigma_list[j]) for j in range(class_number)]).r
        #计算 被除数，本质上是两个矩阵相乘（求和后相加等价于矩阵相乘）
        dividee = np.dot(alpha,temp.T)
        log_likehood += np.log(dividee)
        '''除数 等价于对应元素相乘，使用numpy时，如果数据类型为np.array则*表示对应元素相乘
          此时如果要做矩阵乘法，则使用np.dot(),如果是np.mat类型，则*表示矩阵乘法
        '''
        divider = alpha*temp
        #对于np.array而言，四则运算都是逐元素执行的，此处的除法也不例外
        temp = divider/dividee
        gamma = np.r_[gamma,temp]
    gamma = gamma[1:,:]
    #gamma按列求平均,即可得到新的alpha的估计值
    new_alpha = np.mean(gamma,axis = 0).reshape(1,class_number)

    #计算新均值
    #计算新均值时，被除数是gamma按列求和后的值
    dividee = np.sum(gamma,axis = 0).reshape(1,class_number)
    #计算新均值时，除数 是 gamma的每一列与样本做矩阵乘法（公式中的求和并相加可以转义为矩阵
    divider =([np.dot(values.T,gamma[:,j].reshape(len(values),1)) for j in range(class_number)])
    #逐个元素执行除法，并整形成与mean_list相同的形状
    new_mean = [(divider[i]/dividee[0, i]).reshape(2) for i in range(class_number)]
```

```
    
    #计算新方差
    # dividee = np.sum(gamma,axis = 0).reshape(1,class_number)
    #将900个样本分别与三个类别的均值相减，并存储在列表内
    diff = [values - mean_list[j].reshape(1,2) for j in range(class_number)]
    divider = []
    #此处是为了求900组差值自身矩阵相乘形成的2*2矩阵
    for i in range(class_number):
        #取出diff中的一个元素，注意每个元素是一个900*2的矩阵
        temp = diff[i]
        #每个元素自身相乘，形成2*2的矩阵，同时乘以gamma矩阵的对于元素，后续只要直接做求和处
        temp_matrix = np.array([temp[j,:].reshape(2,1)*temp[j,:].reshape(1,2)*gamma[j,i] for j in ran
        #矩阵求和，并将累和后的矩阵作为元素 追加到 divider列表中
        divider.append(np.sum(temp_matrix,axis = 0))
    # 逐元素做除法，得到新估计的方差
    new_sigma = [divider[i]/dividee[0,i] for i in range(class_number)]
    #更新参数
    mean_list = new_mean
    alpha = new_alpha
    sigma_list = new_sigma

#下面是结果输出的代码

print('\n自行实现的EM算法：')
print('*'*20)
print('pi:',alpha)
print('*'*20)
print('mu:')
for m in mean_list:
    print(m)
print('*'*20)
print('sigma:')
for s in sigma_list:
```

```
    print(s)

print('*'*20)
AIC = -2*log_likehood + 2*(6*class_number-1)
BIC = -2*log_likehood + np.log(len(values))*(6*class_number-1)

print('AIC:',AIC[0,0])
print('BIC:',BIC[0,0])
print('*'*20)
print('\n\n\n')
```

2.2 调库实现

```
gmm=GaussianMixture(n_components=class_number,covariance_type='full', random_state=0)
gmm.fit(values)
gmm_mean = gmm.means_
gmm_cov= [gmm.covariances_[i] for i in range(class_number)]
gmm_alpha = gmm.weights_
print('sklearn包中的EM算法')
print('*'*20)
print('pi:',gmm_alpha)
print('*'*20)
print('mu:')
for i in range(class_number):
    print(gmm_mean[i])
print('*'*20)
print('sigma:')
for s in gmm_cov:
    print(s)
print('*'*20)
print('AIC:',gmm.aic(values))
print('BIC',gmm.bic(values))
print('\n\n\n')
```

```
print('*'*20)
print('真实值(顺序未必与预测值对应！！！！)')
print('*'*20)
print('pi:')
print ([1/3,1/3,1/3])
print('*'*20)
print('mu:')
print(np.array(mean1))
print(np.array(mean2))
print(np.array(mean3))
print('*'*20)
print('cov:')
print(np.array(cov1))
print(np.array(cov2))
print(np.array(cov3))
```

2.3 result

```
自行实现的EM算法:
********************
pi: [[0.03627287 0.29639534 0.66733178]]
********************
mu:
[8.58801627 9.83306019]
[ 7.92581048 10.08471055]
[7.51694744 1.53375755]
********************
sigma:
 [[ 1.62382139 -0.57830739]
  [-0.57830739 0.27434739]]
[[2.08002679 0.82778084]
  [0.82778084 1.64933572]]
[[21.15162901   1.91057065]
```

```
 [ 1.91057065  1.20876102]]
********************
AIC: 8422.59899397279
BIC: 8504.239704949303
********************




sklearn包中的EM算法
********************
pi: [0.33333333 0.33333322 0.33333344]
********************
mu:
[3.02835786 1.05542054]
[ 7.99570386 10.04771047]
[12.00688688  2.00465087]
********************
sigma:
[[ 0.94503528 -0.40985998]
 [-0.40985998  0.93637324]]
[[2.07140619  0.66802496]
 [0.66802496 1.54829122]]
[[ 1.09251519 -0.0214873 ]
 [-0.0214873   1.00524051]]
********************
AIC: 7352.490815632364
BIC 7434.131526608877




********************
```

```
真实值(顺序未必与预测值对应！！！！)
********************
pi:
[0.3333333333333333, 0.3333333333333333, 0.3333333333333333]
********************
mu:
[3 1]
[ 8 10]
[12  2]
********************
cov:
[[ 1.  -0.5]
 [-0.5  1. ]]
[[2.   0.8]
 [0.8  2. ]]
[[1  0]
 [0  1]]
```