
成 绩 _____

北京航空航天大學
BEIHANG UNIVERSITY

# 深度学习与自然语言处理大作业

# EM算法

| | |
|---|---|
| 院（系）名称 | 自动化科学与电气工程学院 |
| 专业名称 | 电子信息 |
| 学生学号 | ZY2103812 |
| 学生姓名 | 朱远哲 |
| 指导教师 | 秦曾昌 |

2022 年 4 月

# 目　　录

# 1 问题描述

一个袋子中三种硬币的混合比例为：s1, s2与1-s1-s2 (0<=si<=1), 三种硬币掷出正面的概率分别为：p, q, r。 （1）自己指定系数s1, s2, p, q, r，生成N个投掷硬币的结果（由01构成的序列，其中1为正面，0为反面），利用EM算法来对参数进行估计并与预先假定的参数进行比较。。

# 2 问题表达

代码的编写主要分为两个部分，第一个部分二项混合分布模型数据的生成，预先设置三种硬币的混合比例和相应掷出正面的概率共六个参数，采用二项混合模型生成数量满足要求的数据集。

第二个部分为EM算法编写的过程，采用迭代的方式估计参数隐变量，其基本想法是：若参数Θ 已知，则可根据训练数据推断出最优隐变量Z的值(E步) ;反之，若Z的值已知，则可方便地对参数Θ 做极大似然估计(M步)。利用编写的EM算法程序对第一步生成数据集的参数进行估计并对比准确性。

# 3 具体算法实现

## 3.1 数据集生成

数据的生成可以使用numpy中的random.binomial函数，依据相应的参数分别生成共10000个硬币投掷数据，将各类硬币投掷结果放入数组中，并利用shuffle函数对数组进行重新排列，得到最终的数据集。

## 3.2 EM 算法流程

最大期望算法（Expectation-Maximization algorithm, EM）是一类通过迭代进行极大似然估计的优化算法，通常作为牛顿迭代法的替代用于对包含隐变量或缺失数据的概率模型进行参数估计

EM算法分为E步和M步，E步求期望，对隐变量进行积分；M步求参数最大值。算法流程如下

（1） 取参数的初始值进行迭代

（2） E步：依据当前模型参数，计算分模型k对观测数据 $y_j$ 的响应度

$$\hat{\gamma}_{jk}=\frac{\alpha_k\phi(y_j|\theta_k)}{\sum_{k=1}^{K}\alpha_k\phi(y_j|\theta_k)},j=1,2,...,N;k=1,2,...,K$$

（3） M步：计算新一轮迭代的模型参数

$$\hat{\theta}_k=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}y_j}{\sum_{j=1}^{N}\hat{\gamma}_{jk}},k=1,2,...,K$$

$$\hat{\alpha}_k=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}}{\sum_{j=1}^{N}\hat{\gamma}_{jk}},k=1,2,...,K$$

（4） 重复第（2）步和第（3）步，直到收敛

## 3.3 EM 算法模型分析

由题目描述，数据集由二项混合分布生成，共有三种硬币的混合比例和相应掷出正面的概率共六个参数，$\theta=\{s_1,s_2,s_3,p,q,r\}$，带入上述算法流程可以得到相应步骤公式为

E步：

$$\hat{\gamma}_{1j}=\frac{s_1p^{y_j}(1-p)^{1-y_j}}{s_1p^{y_j}(1-p)^{1-y_j}+s_2q^{y_j}(1-q)^{1-y_j}+s_3r^{y_j}(1-r)^{1-y_j}},\ j=1,2,...,N;$$

$$\hat{\gamma}_{2j}=\frac{s_2q^{y_j}(1-q)^{1-y_j}}{s_1p^{y_j}(1-p)^{1-y_j}+s_2q^{y_j}(1-q)^{1-y_j}+s_3r^{y_j}(1-r)^{1-y_j}},\ j=1,2,...,N;$$

$$\hat{\gamma}_{3j}=1-\hat{\gamma}_{1j}-\hat{\gamma}_{2j}$$

M步：

$$\hat{s}_1=\frac{\sum_j^N\gamma_{1j}}{N},\quad \hat{s}_2=\frac{\sum_j^N\gamma_{2j}}{N},\quad \hat{s}_3=1-\hat{s}_1-\hat{s}_2$$

$$\hat{p}=\frac{\sum_j^N y_j\gamma_{1j}}{\sum_j^N\gamma_{1j}},\quad \hat{q}=\frac{\sum_j^N y_j\gamma_{2j}}{\sum_j^N\gamma_{2j}},\quad \hat{r}=\frac{\sum_j^N y_j\gamma_{3j}}{\sum_j^N\gamma_{3j}}$$

## 3.4 实验代码

（1） 数据生成

```
import numpy as np
import time

number = 10000
p1, p2, p3 = 0.2, 0.5, 0.4
number1, number2, number3 = int(number * 0.2), int(number * 0.5), number - int(number * 0.2) - int(number * 0.5)

data1 = np.random.binomial(1, p1, size=number1)
data2 = np.random.binomial(1, p2, size=number2)
data3 = np.random.binomial(1, p3, size=number3)
data0 = []
data0.extend(data1)
data0.extend(data2)
data0.extend(data3)
np.random.shuffle(data0)
data_arr = np.array(data0)
```

（2） E步

```python
def E_step(dataset, alpha1, alpha2, alpha3, theta1, theta2, theta3):
    gamma1 = alpha1 ** dataset * (1 - alpha1) ** (1 - dataset) * theta1
    gamma2 = alpha2 ** dataset * (1 - alpha2) ** (1 - dataset) * theta2
    gamma3 = alpha3 ** dataset * (1 - alpha3) ** (1 - dataset) * theta3
    sum0 = gamma1 + gamma2 + gamma3
    gamma1 = gamma1 / sum0
    gamma2 = gamma2 / sum0
    gamma3 = gamma3 / sum0
    return gamma1, gamma2, gamma3
```

（3） M步

```python
def M_step(dataset, gamma1, gamma2, gamma3):
    alpha1_new = np.sum(gamma1) / number
    alpha2_new = np.sum(gamma2) / number
    alpha3_new = np.sum(gamma3) / number
    theta1_new = np.dot(gamma1, dataset) / np.sum(gamma1)
    theta2_new = np.dot(gamma2, dataset) / np.sum(gamma2)
    theta3_new = np.dot(gamma3, dataset) / np.sum(gamma3)
    return alpha1_new, alpha2_new, alpha3_new, theta1_new, theta2_new, theta3_new
```

（4） 初始值设定及迭代

```python
if __name__ == '__main__':
    start = time.time()
    step = 0
    iter_num = 1000
    alpha01, alpha02, alpha03 = 0.1, 0.4, 0.5
    theta01, theta02, theta03 = 0.6, 0.7, 0.4
    while step < iter_num:
        step = step + 1
        gamma01, gamma02, gamma03 = E_step(data_arr, alpha01, alpha02, alpha03, theta01, theta02, theta03)
        alpha01, alpha02, alpha03, theta01, theta02, theta03 = M_step(data_arr, gamma01, gamma02, gamma03)
```

（5） 结果输出

```
---------------------------
the Parameters set is:
alpha1:0.2, alpha2:0.5, alpha3:0.3, theta1:0.2, theta2:0.5, theta3:0.4
---------------------------
the Parameters predict is:
alpha1:0.20, alpha2:0.44, alpha3:0.37, theta1:0.24, theta2:0.49, theta3:0.42
---------------------------
time span: 1.2524893283843994
```

# 4 运行结果

## 4.1 运行结果

预设参数：

| 硬币种类 | 掷出正面概率 | 数据占比 |
|---|---|---|
| 1 | 0.2 | 0.2 |
| 2 | 0.5 | 0.5 |
| 3 | 0.4 | 0.3 |

估计参数：1、生成数据量为10000个时：

| 硬币种类 | 掷出正面概率 | 数据占比 |
|---|---|---|
| 1 | 0.23 | 0.20 |
| 2 | 0.48 | 0.44 |
| 3 | 0.41 | 0.37 |

迭代次数设置为500次，程序运行时间为0.496 s

2、生成数据量为30000个时：

| 硬币种类 | 掷出正面概率 | 数据占比 |
|---|---|---|
| 1 | 0.23 | 0.20 |
| 2 | 0.49 | 0.44 |
| 3 | 0.41 | 0.37 |

迭代次数设置为500次，程序运行时间为1.770s

## 4.2 结果讨论与分析

从结果可以看出，通过EM算法得到了较好的估计效果，预设值不变的情况下，改变算法初始值，结果依旧可以收敛并得到较好的估计效果。增加生成数据量时，可以看出，估计结果有了一定程度上精度的提升，主要有两方面原因，一方面数据量增加时，有利于EM算法进行迭代求解，另一方面，由于数据生成时采用了二项分布随机数的方式，样本量的增加也会使样本方差和均值更加准确。但是程序每次运行，得到的结果都会不同，这也是所采用的数据集数据具有随机性所导致的。

# 5 个人总结与体会

本次大作业实现了EM算法在二项混合模型参数估计中的应用，通过本次程序编写，我对python编程变得更加熟悉，同时也学习了解了EM算法，对迭代式方法的理解更加深刻。在之前的学习中，概率模型中的变量都是观测变量，可以采用极大似然估计发或贝叶斯估计法对参数进行估计，通过本次大作业，学习了解了如果概率模型中含有隐变量的时候，该如何对模型参数进行估计，在学习中通过三硬币模型了解到何为概率模型中含有隐变量，通过算法的学习程序的编写，又让我有了更加深刻的理解，也通过大作业有了很大的收获。