第18章 概率潜在语义分析

1.概率潜在语义分析是利用概率生成模型对文本集合进行话题分析的方法。概率潜在语义分析受潜在语义分析的启发提出两者可以通过矩阵分解关联起来。

给定一个文本集合，通过概率潜在语义分析，可以得到各个文本生成话题的条件概率分布，以及各个话题生成单词的条件概率分布。

概率潜在语义分析的模型有生成模型，以及等价的共现模型。其学习策略是观测数据的极大似然估计，其学习算法是EM算法。

2.生成模型表示文本生成话题，话题生成单词从而得到单词文本共现数据的过程；假设每个文本由一个话题分布决定，每个话题由一个单词分布决定。单词变量w与文本变量d是观测变量话题变量z是隐变量。生成模型的定义如下： $P(T)=\prod_{(w,d)} P(w,d)^{n(w,d)}$

$P(w,d)=P(d)P(w|d)=P(d)\sum_{\alpha} P(z|d)P(w|z)$ 3.共现模型描述文本单词共现数据拥有的模式。共现模型的定义如下： $P(T)=\prod_{(w,d)} P(w,d)^{n(w,d)}$

$P(w,d)=\sum_{z\in Z} P(z)P(w|z)P(d|z)$

4.概率潜在语义分析的模型的参数个数是$O(M\cdot K+N\cdot K)$。现实中$K\ll M$，所以概率潜在语义分析通过话题对数据进行了更简洁地表示，实现了数据压缩。

5.模型中的概率分布$P(w|d)$可以由参数空间中的单纯形表示。M维参数空间中，单词单纯形表示所有可能的文本的分布，在其中的话题单纯形表示在K个话题定义下的所有可能的文本的分布。话题单纯形是单词单纯形的子集，表示潜在语义空间。

6.概率潜在语义分析的学习通常采用EM算法通过迭代学习模型的参数，$P(w|z)$ 和$P(z|d)$，而$P(d)$可直接统计得出。

概率潜在语义分析（probabilistic latent semantic analysis, PLSA）,也称概率潜在语义索引（probabilistic latent semantic indexing, PLSI）,是一种利用概率生成模型对文本集合进行话题分析的无监督学习方法。

模型最大特点是用隐变量表示话题，整个模型表示文本生成话题，话题生成单词，从而得到单词-文本共现数据的过程；假设每个文本由一个话题分布决定，每个话题由一个单词分布决定。

18.1.2 生成模型

假设有单词集合 $W=\{w_1,w_2,\ldots,w_M\}$， 其中M是单词个数；文本（指标）集合$D=\{d_1,d_2,\ldots,d_N\}$, 其中N是文本个数；话题集合$Z=\{z_1,z_2,\ldots,z_K\}$，其中K是预先设定的话题个数。随机变量 w 取值于单词集合；随机变量 d 取值于文本集合，随机变量 z 取值于话题集合。概率分布 $P(d)$、条件概率分布 $P(z|d)$、条件概率分布 $P(w|z)$ 皆属于多项分布，其中 $P(d)$ 表示生成文本 d 的概率，$P(z|d)$ 表示文本 d 生成话题 z 的概率，$P(w|z)$ 表示话题 z 生成单词 w 的概率。

每个文本 d 拥有自己的话题概率分布 $P(z|d)$，每个话题 z 拥有自己的单词概率分布 $P(w|z)$；也就是说**一个文本的内容由其相关话题决定，一个话题的内容由其相关单词决定。**

生成模型通过以下步骤生成文本·单词共现数据：
（1）依据概率分布 $P(d)$，从文本（指标）集合中随机选取一个文本 d , 共生成 N 个文本；针对每个文本，执行以下操作；
（2）在文本d 给定条件下，依据条件概率分布 $P(z|d)$, 从话题集合随机选取一个话题 z, 共生成 L 个话题，这里 L 是文本长度；
（3）在话题 z 给定条件下，依据条件概率分布 $P(w|z)$, 从单词集合中随机选取一个单词 w.

注意这里为叙述方便，假设文本都是等长的，现实中不需要这个假设。

生成模型中， 单词变量 w 与文本变量 d 是观测变量， 话题变量 z 是隐变量， 也就是说模型生成的是单词-话题-文本三元组合 (w,z,d)的集合， 但观测到的单词-文本二元组 （w,d）的集合， 观测数据表示为单词-文本矩阵 T的形式，矩阵 T 的行表示单词，列表示文本， 元素表示单词-文本对（w,d）的出现次数。

从数据的生成过程可以推出，文本-单词共现数据T的生成概率为所有单词-文本对(w,d)的生成概率的乘积：

$P(T)=\prod_{w,d}P(w,d)^{n(w,d)}$

这里 $n(w,d)$ 表示 (w,d)的出现次数，单词-文本对出现的总次数是 $N*L$。 每个单词-文本对（w,d）的生成概率由一下公式决定：

$P(w,d)=P(d)P(w|d)$

$=P(d)\sum_z P(w,z|d)$

$=P(d)\sum_z P(z|d)P(w|z)$

18.1.3 共现模型

$P(w,d)=\sum_{z\in Z}P(z)P(w|z)P(d|z)$

虽然生成模型与共现模型在概率公式意义上是等价的，但是拥有不同的性质。生成模型刻画文本-单词共现数据生成的过程，共现模型描述文本-单词共现数据拥有的模式。

如果直接定义单词与文本的共现概率 $P(w,d)$, 模型参数的个数是 $O(M*N)$, 其中 M 是单词数， N 是文本数。 概率潜在语义分析的生成模型和共现模型的参数个数是 $O(M*K+N*K)$, 其中 K 是话题数。 现实中 $K<<M$, 所以**概率潜在语义分析通过话题对数据进行了更简洁的表示，减少了学习过程中过拟合的可能性**。

算法 18.1 （概率潜在语义模型参数估计的EM算法）

输入： 设单词集合为 $W=\{w_1,w_2,\dots,w_M\}$, 文本集合为 $D=\{d_1,d_2,\dots,d_N\}$, 话题集合为 $Z=\{z_1,z_2,\dots,z_K\}$, 共现数据 $\{n(w_i,d_j)\}$, $i=1,2,\dots,M, j=1,2,\dots,N$;

输出： $P(w_i|z_k)$ 和 $P(z_k|d_j)$.

1. 设置参数 $P(w_i|z_k)$ 和 $P(z_k|d_j)$ 的初始值。
2. 迭代执行以下E步，M步，直到收敛为止。

E步：

$P(z_k|w_i,d_j)=\frac{P(w_i|z_k)P(z_k|d_j)}{\sum_{k=1}^K P(w_i|z_k)P(z_k|d_j)}$

M步：

$$P(w_i|z_k) = \frac{\sum_{j=1}^{N} n(w_i,d_j)P(z_k|w_i,d_j)}{\sum_{m=1}^{M}\sum_{j=1}^{N} n(w_m,d_j)P(z_k|w_m,d_j)}$$

```
$P(z_{k}|d_{j}) = \frac{\sum_{i=1}^{M}n(w_{i},d_{j})P(z_{k}|w_{i},d_{j})}
{n(d_{j})}$
```

习题 18.3

```
import numpy as np
```

```
X = [[0,0,1,1,0,0,0,0,0],
     [0,0,0,0,0,1,0,0,1],
     [0,1,0,0,0,0,0,1,0],
     [0,0,0,0,0,0,1,0,1],
     [1,0,0,0,0,1,0,0,0],
     [1,1,1,1,1,1,1,1,1],
     [1,0,1,0,0,0,0,0,0],
     [0,0,0,0,0,0,1,0,1],
     [0,0,0,0,0,2,0,0,1],
     [1,0,1,0,0,0,0,1,0],
     [0,0,0,1,1,0,0,0,0]]
X = np.asarray(X);X
```

```
array([[0, 0, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1],
       [1, 0, 0, 0, 0, 1, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1],
       [0, 0, 0, 0, 0, 2, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 1, 1, 0, 0, 0, 0]])
```

```
X.shape
```

```
(11, 9)
```

```
X = X.T;X
```

```
array([[0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [0, 1, 0, 0, 1, 1, 0, 0, 2, 0, 0],
       [0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0],
       [0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0]])
```

```
class PLSA:
    def __init__(self, K, max_iter):
        self.K = K
        self.max_iter = max_iter

    def fit(self, X):
        n_d, n_w = X.shape

        # P(z|w,d)
        p_z_dw = np.zeros((n_d, n_w, self.K))

        # P(z|d)
        p_z_d = np.random.rand(n_d, self.K)

        # P(w|z)
        p_w_z = np.random.rand(self.K, n_w)

        for i_iter in range(self.max_iter):
            # E step
            for di in range(n_d):
                for wi in range(n_w):
                    sum_zk = np.zeros((self.K))
                    for zi in range(self.K):
                        sum_zk[zi] = p_z_d[di, zi] * p_w_z[zi, wi]
                    sum1 = np.sum(sum_zk)
                    if sum1 == 0:
                        sum1 = 1
                    for zi in range(self.K):
                        p_z_dw[di, wi, zi] = sum_zk[zi] / sum1
```

```
            # M step

            # update P(z|d)
            for di in range(n_d):
                for zi in range(self.K):
                    sum1 = 0.
                    sum2 = 0.

                    for wi in range(n_w):
                        sum1 = sum1 + X[di, wi] * p_z_dw[di, wi, zi]
                        sum2 = sum2 + X[di, wi]

                    if sum2 == 0:
                        sum2 = 1
                    p_z_d[di, zi] = sum1 / sum2

            # update P(w|z)
            for zi in range(self.K):
                sum2 = np.zeros((n_w))
                for wi in range(n_w):
                    for di in range(n_d):
                        sum2[wi] = sum2[wi] + X[di, wi] * p_z_dw[di, wi, zi]
                sum1 = np.sum(sum2)
                if sum1 == 0:
                    sum1 = 1
                    for wi in range(n_w):
                        p_w_z[zi, wi] = sum2[wi] / sum1

        return p_w_z, p_z_d


# https://github.com/lipiji/PG_PLSA/blob/master/plsa.py
```

```
model = PLSA(2, 100)
p_w_z, p_z_d = model.fit(X)
```

```
p_w_z
```

```
array([[0.64238757, 0.05486094, 0.18905573, 0.24047994, 0.41230822,
        0.38136674, 0.81525232, 0.74314243, 0.32465342, 0.19798429,
        0.72010476],
       [0.6337431 , 0.79442181, 0.96755364, 0.22924392, 0.99367301,
        0.20277986, 0.40513752, 0.51164374, 0.73750246, 0.22300907,
        0.17339099]])
```

```
p_z_d
```

```
array([[7.14884177e-01, 2.85115823e-01],
       [5.38307075e-02, 9.46169293e-01],
       [1.00000000e+00, 3.40624611e-11],
       [1.00000000e+00, 1.12459358e-24],
       [1.00000000e+00, 5.00831891e-42],
       [1.66511004e-19, 1.00000000e+00],
       [1.00000000e+00, 8.02144289e-15],
       [1.04149223e-02, 9.89585078e-01],
       [5.96793031e-03, 9.94032070e-01]])
```

本章代码来源：https://github.com/hktxt/Learn-Statistical-Learning-Method

本文代码更新地址：https://github.com/fengdu78/lihang-code

中文注释制作：机器学习初学者公众号：ID:ai-start-com

配置环境：python 3.5+

代码全部测试通过。