PLSA理论与实践

PLSA又称为概率潜在语义分析，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。该模型最大的特点是加入了主题这一隐变量，文本生成主题，主题生成单词，从而得到单词-文本共现矩阵。本文将对包含物理学、计算机科学、统计学、数学四个领域的15000条文献摘要的数据集（保存在`Task-Corpus.csv`中）使用PLSA算法进行处理。

一、算法推导

1.1 E-steps

设单词集合为$w_i(i=1,\cdots,M)$，其中M为单词数；文本集合为$d_j(j=1,\cdots,N)$，其中N为文本数；主题集合为$z_k(k=1,\cdots,K)$，其中K为主题数。对给定的文本，主题的分布是一个有K个选项的多项分布，因此参数个数为$N\times K$，设参数矩阵为Λ。对给定的主题，单词的分布是一个有M个选项的多项分布，因此参数个数为$K\times M$，设参数矩阵为Θ。一般来说$K\ll M$，这就避免了模型的过拟合。

如果主题未知，根据全概率公式有

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

因此非完全数据（主题未知）的似然函数为

$$L(\Theta,\Lambda|X)=p(X|\Theta)=\prod_{i=1}^{M}\prod_{j=1}^{N}(p(d_j)\sum_{k=1}^{K}p(w_i|z_k)p(z_k|d_j))^{n(w_i,d_j)} \tag{2}$$

对数似然为

$$\log L(\Theta,\Lambda|X)=\sum_{i=1}^{M}\sum_{j=1}^{N}n(w_i,d_j)\log(p(d_j)\sum_{k=1}^{K}p(w_i|z_k)p(z_k|d_j)) \tag{3}$$

对数似然中包含求和的对数，因此难以处理。

如果主题已知，文章d_j出现单词w_i的概率为

$$p(w_i,d_j)=p(d_j)p(w_i|z_k)p(z_k|d_j) \tag{4}$$

因此完全数据的似然函数为

$$L(\Theta|X)=\prod_{i=1}^{M}\prod_{j=1}^{N}(p(d_j)p(w_i|z_k)p(z_k|d_j))^{n(w_i,d_j)} \tag{5}$$

对数似然为

$$\log L(\Theta|X)=\sum_{j=1}^{N}n(w_i,d_j)\log(p(d_j)p(w_i|z_k)p(z_k|d_j)) \tag{6}$$

Q函数为对数似然$\log L(\Theta|X)$在后验分布$p(z_k|w_i,d_j)$下的期望

$$
\begin{aligned}
Q &= \sum_{k=1}^{K} p(z_k|w_i,d_j) \sum_{i=1}^{M}\sum_{j=1}^{N} n(w_i,d_j)\log(p(d_j)p(w_i|z_k)p(z_k|d_j)) \\
&= \sum_{i=1}^{M}\sum_{j=1}^{N} n(w_i,d_j) \sum_{k=1}^{K} p(z_k|w_i,d_j)\log(p(d_j)p(w_i|z_k)p(z_k|d_j))
\end{aligned} \tag{7}
$$

其中后验概率

$$
p(z_k|w_i,d_j) = \frac{p(w_i|z_k)p(z_k|d_j)}{\sum_{k=1}^{K} p(w_i|z_k)p(z_k|d_j)} \tag{8}
$$

1.2 M-step

$p(w_i|z_k), p(z_k|d_j)$满足约束条件

$$
\sum_{i=1}^{M} p(w_i|z_k) = 1, k = 1,\cdots,K \tag{9}
$$

$$
\sum_{k=1}^{K} p(z_k|d_j) = 1, j = 1,\cdots,N \tag{10}
$$

引入拉格朗日函数

$$
J = Q + \sum_{k=1}^{K} r_k(1 - \sum_{i=1}^{M} p(w_i|z_k)) + \sum_{j=1}^{N} \rho_j(1 - \sum_{k=1}^{K} p(z_k|d_j)) \tag{11}
$$

$$
\frac{\partial J}{\partial p^*(w_i|z_k)} = \sum_{j=1}^{N} \frac{n(w_i,d_j)p(z_k|w_i,d_j)}{p(w_i|z_k)} - r_k = 0 \tag{12}
$$

因此

$$
r_k p^*(w_i|z_k) = \sum_{j=1}^{N} n(w_i,d_j)p(z_k|w_i,d_j) \tag{13}
$$

对i求和，就有

$$
r_k = \sum_{i=1}^{M}\sum_{j=1}^{N} n(w_i,d_j)p(z_k|w_i,d_j) \tag{14}
$$

$$
p^*(w_i|z_k) = \frac{\sum_{j=1}^{N} n(w_i,d_j)p(z_k|w_i,d_j)}{\sum_{i=1}^{M}\sum_{j=1}^{N} n(w_i,d_j)p(z_k|w_i,d_j)} \tag{15}
$$

同理

$$
p^*(z_k|d_j) = \frac{\sum_{j=1}^{N} n(w_i,d_j)p(z_k|w_i,d_j)}{\sum_{i=1}^{M} n(w_i,d_j)} \tag{16}
$$

$(6)(13)(14)$三式共同构成PLSA算法的迭代公式。

二、算法实现

用python实现PLSA算法。首先对数据集先做预处理。对给定的文本进行分词，利用wordnet语料库将同义词进行替换（例如单复数不同的词需要替换成同一个词），并将停用词排除（停用词表在网上下载，参见作业中的`stopwords.dic`文件）。然后对全体文本构成的单词集合进行词频统计，构建词频矩阵$n(w_i, d_j)$。这一部分用到了python的`nltk`包。核心代码如下。

```
    words = set()
    word_counts = []
    for document in documents:
        seglist = word_tokenize(document)
        wordlist = []
        for word in seglist:
            synsets = wordnet.synsets(word)
            if synsets:
                syn_word = synsets[0].lemmas()[0].name()
                if syn_word not in stopwords:
                    wordlist.append(syn_word)
            else:
                if word not in stopwords:
                    wordlist.append(word)
        words = words.union(wordlist)
        word_counts.append(Counter(wordlist))
    word2id = {words:id for id, words in enumerate(words)}
    id2word = dict(enumerate(words))

    N = len(documents) # number of documents
    M = len(words) # number of words
    X = np.zeros((N, M))
    for i in range(N):
        for keys in word_counts[i]:
            X[i, word2id[keys]] = word_counts[i][keys]
```

然后根据$(6)(13)(14)$三式进行PLSA算法的编写。注意到这三个式子都可以写成矩阵的形式，提高运算效率。同时注意到这三个式子都和分子成正比，因此可以计算出份子再除以归一化常数即可。E-step的代码如下。

```
def E_step(lam, theta):
    # lam: N * K, theta: K * M, p = K * N * M
    N = lam.shape[0]
    M = theta.shape[1]
    lam_reshaped = np.tile(lam, (M, 1, 1)).transpose((2,1,0)) # K * N * M
    theta_reshaped = np.tile(theta, (N, 1, 1)).transpose((1,0,2)) # K * N * M
    temp = lam @ theta
    p = lam_reshaped * theta_reshaped / temp
    return p
```

M-step的代码如下。

```python
def M_step(p, X):
    # p: K * N * M, X: N * M, lam: N * K, theta: K * M
    # update lam
    lam = np.sum(p * X, axis=2) # K * N
    lam = lam / np.sum(lam, axis=0) # normalization for each column
    lam = lam.transpose((1,0)) # N * K

    # update theta
    theta = np.sum(p * X, axis=1) # K * M
    theta = theta / np.sum(theta, axis=1)[:, np.newaxis] # normalization for each row

    return lam, theta
```

计算对数似然的代码如下。

```python
def LogLikelihood(p, X, lam, theta):
    # p: K * N * M, X: N * M, lam: N * K, theta: K * M
    res = np.sum(X * np.log(lam @ theta)) # N * M
    return res
```

用随机数初始化Θ, Λ以避免落入局部最优。设定最大迭代次数为200。对数似然的阈值为10。当相邻两次对数似然的差小于阈值或者达到最大迭代次数时停止迭代。如果计算对数似然时报错，说明某个参数被舍入到0，此时也需要停止迭代。

三、结果分析

由于笔记本电脑的内存有限，从所给数据集中随机抽取1000篇文本进行实验。设定主题数为4。某次实验的结果如下。构建的字典中包含11342个单词。字典保存在`dictionary.json`文件中。

程序在迭代152次后停止。可以看到对数似然确实在不断上升。

```
iter 126, loglikelihood=-641492.8640626435
iter 127, loglikelihood=-641456.545512969
iter 128, loglikelihood=-641416.4457939145
iter 129, loglikelihood=-641377.2670346699
iter 130, loglikelihood=-641338.0566469812
iter 131, loglikelihood=-641300.1142771629
iter 132, loglikelihood=-641263.2922759751
iter 133, loglikelihood=-641226.539450907
iter 134, loglikelihood=-641189.3085311342
iter 135, loglikelihood=-641158.233029981
iter 136, loglikelihood=-641133.8620199539
iter 137, loglikelihood=-641112.1104276278
iter 138, loglikelihood=-641091.6884679872
iter 139, loglikelihood=-641069.0936236759
iter 140, loglikelihood=-641047.413835524
iter 141, loglikelihood=-641027.970127103
iter 142, loglikelihood=-641013.1648921494
iter 143, loglikelihood=-640998.6415058394
iter 144, loglikelihood=-640974.9665844033
iter 145, loglikelihood=-640949.5683991287
iter 146, loglikelihood=-640930.7281174071
iter 147, loglikelihood=-640916.0869649214
iter 148, loglikelihood=-640902.282806176
iter 149, loglikelihood=-640888.0247734097
iter 150, loglikelihood=-640874.6005312797
iter 151, loglikelihood=-640862.9150480253
iter 152, loglikelihood=-640852.9399332189
```

每个文本的主题分布保存在`DocTopicDistribution.csv`文件中。每个主题的单词分布保存在`TopicWordDistribution.csv`文件中。每个主题中出现概率最高的9个单词保存在`topics.txt`文件中，如下图所示。可以看到出现概率最高的单词分别为`astatine, network, Associate_in_Nursing, algorithm`，分别对应了物理学、计算机科学、统计学、数学四个领域。这证明了PLSA方法的有效性。

```
['astatine', 'beryllium', 'consequence', 'galaxy', 'star', 'Associate_in_Nursing', 'information_technolo
['network', 'learning', 'data', 'propose', 'Associate_in_Nursing', 'beryllium', 'information_technology'
['Associate_in_Nursing', 'system', 'astatine', 'consequence', 'magnetic', 'propose', 'information_techno
['algorithm', 'Associate_in_Nursing', 'function', 'beryllium', 'consequence', 'information_technology',
```

项目开源

本项目开源在[kungfu-crab/PLSA: A python implementation for PLSA(Probabilistic Latent Semantic Analysis) using EM algorithm. (github.com)](), 仅作为学习交流使用，禁止转载与抄袭。

参考文献

[1] Hofmann, T. (1999). Probabilistic Latent Semantic Analysis. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (pp. 289-296). Morgan Kaufmann Publishers Inc.