LDA模型

1 任务描述

从给定的语料库中均匀抽取200个段落（每个段落大于500个词），每个段落的标签就是对应段落所属的小说。利用LDA（Latent Dirichlet Allocation）模型对于文本建模，并把每个段落表示为主题分布后进行分类。验证与分析分类结果。

2 算法原理

主题模型（Topic Model）是以非监督学习的方式对文档的隐含语义结构(latent semantic structure)进行聚类(clustering)的统计模型。

主题模型（Topic Model）是一种常用的文本挖掘工具，用于发现文本主体中的隐藏语义结构。每个文档都应该对应着一个或多个的主题（topic），而每个主题都会有对应的词分布，通过主题，就可以得到每个文档的词分布。依据这一原理，就可以得到主题模型的一个核心公式：

$$p\left(w_i \mid d_j\right)=\sum_{k=1}^{K} p\left(w_i \mid t_k\right) \times p\left(t_k \mid d_j\right)$$

其中w_i代表单词(word)；d_j代表文档(document)；t_k代表主题(topic)

在一个已知的数据集中，每个词和文档对应的$p\left(w_i \mid d_j\right)$都是已知的。而主题模型就是根据这个已知的信息，通过计算$p\left(w_i \mid t_k\right)$和$p\left(t_k \mid d_j\right)$的值，从而得到主题的词分布和文档的主题分布信息。而要得到这个分布信息，现在常用的方法就是LSA(LSI)和LDA。其中LSA主要是采用SVD的方法进行暴力破解，而LDA则是通过贝叶斯学派的方法对分布信息进行拟合。这里我们使用LDA算法。

利用LDA模型生成一篇文档的方式：

· 按照先验概率 $P(d_i)$ 选择一篇文档d_i从狄利克雷分布（即Dirichlet分布）α 中取样生成文档 d_i的主题分布 θ_i，换言之，主题分布 θ_i超参数为 α \alphaα 的Dirichlet分布生成。
· 从主题的多项式分布 θ_i中取样生成文档d_i第 j 个词的主题$z_{i,j}$。
· 从狄利克雷分布（即Dirichlet分布）β 中取样生成主题$z_{i,j}$对应的词语分布 $\phi z_{i,j}$，换言之，词语分布 $\phi z_{i,j}$由参数为 β 的Dirichlet分布生成。
· 从词语的多项式分布 $\phi z_{i,j}$中采样最终生成词语$w_{i,j}$。
根据文档反推其主题分布:

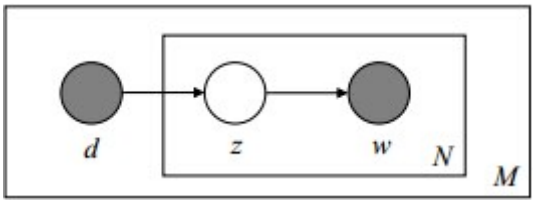

图1

图中被涂色的d、w表示可观测变量，未被涂色的z表示未知的隐变量；
从而可以根据大量已知的文档-词项信息，训练出文档-主题和主题-词项，如下公式所示：

$$p\left(w_i \mid d_j\right)=\sum_{k=1}^{K} p\left(w_j \mid z_k\right) \times p\left(z_k \mid d_i\right)$$

故得到文档中每个词的生成概率为：

$$
\begin{aligned}
P(d_i, w_j) &= P(d_i)P(w_j \mid d_i) \\
&= P(d_i)\sum_{k=1}^{K} P(w_j \mid z_k)P(z_k \mid d_i)
\end{aligned}
$$

由于可事先计算求出，而$P(w_j \mid z_k)$和$P(z_k \mid d_i)$未知，所以$\theta = (P(w_j \mid z_k), P(z_k \mid d_i))$就是我们要估计的参数（值），通俗点说，就是要最大化这个θ。

这里我们采用EM算法进行估计。

3 任务过程

首先在模型训练阶段对所有小说进行预处理，应用jieba进行分词，去除非中文和停用词的内容后，通过sklearn中的CountVectorizer进行词频计算，获得概率向量。

```
def preprocess(path):
    files = os.listdir(path)
    try:
        stopword_list = open(stop_file, encoding='utf-8')
    except:
        stopword_list = []
        print("error in stop_file")
    stop_list = []
    for line in stopword_list:
        line = re.sub(u'\n|\\r', '', line)
        stop_list.append(line)
    seg_list = []
    for file in files:
        position = path + '\\' + file
        with open(position, "r", encoding="ANSI") as f:
            data = re.sub(u"[^\u4e00-\u9fa5]", '', f.read())
        seg = " ".join(jieba.lcut(data, use_paddle=True, cut_all=False))
        seg_list.append(seg)
    vec = CountVectorizer(stop_words = stop_list)
    cnt = vec.fit_transform(seg_list)
    return cnt,vec
```

通过sklearn中的LatentDirichletAllocation和fit_transform构建lda模型。

在测试阶段，将每篇小说分别除以15以获取到以保证均匀的取到200+的段落，并判断每段是否大于500个词汇，同样的在去除每段的非中文和停用词后，对每段计算词频，获得概率向量。由于其中两篇的内容较少无法正常的分成每段大于500词的段落，因此在这里将《越女剑》去除。

```
def cut_para(path, min_char):
    with open(path, "r", encoding="ANSI") as f:
        file = f.read()
        n = len(file)
        paras = []
        tmp_char = ''
        for i, char in enumerate(file):
            if char != '\u3000':
                tmp_char += char
            if char == '\n':
                tmp_char = tmp_char[:-1]
                if len(tmp_char) >= min_char:
                    paras.append(tmp_char[0:min_char])
                    tmp_char = ''
```

```
                else:
                    continue
            if (i + 1) == n:
                if len(tmp_char) >= min_char:
                    paras.append(tmp_char[0:min_char])
                break
    f.close()
    paras = paras[::int(len(paras)/15)][0:15]
    return paras

def docs_gen(path, char_len, voc):
    files = os.listdir(path)
    try:
        stopword_list = open(stop_file, encoding='utf-8')
    except:
        stopword_list = []
        print("error in stop_file")
    stop_list = []
    for line in stopword_list:
        line = re.sub(u'\n|\\r', '', line)
        stop_list.append(line)
    seg_list = []
    for file in files:
        position = path + '\\' + file
        seg_list.extend(word_seg(cut_para(position, char_len)))
    vec = CountVectorizer(token_pattern = r"(?u)\b\w\w+\b",
                          max_features = 5000,
                          stop_words = stop_list,
                          max_df = 0.5,
                          vocabulary = voc)
    cnt = vec.fit_transform(seg_list)
    return cnt,vec
```

最后计算各个文档概率向量与各个小说的概率向量的相似程度即各个文档对应的主题，这里使用欧氏距离，即可判断各个文档属于哪篇小说，完成分类。由于段落是按通过计算分类正确的数量占总数的概率等到正确率。

```
def print_res2(res1, res2):
    match = 0
    for j, p2 in enumerate(res2):
        dis = []
        for i, p1 in enumerate(res1):
            dis.append(np.sqrt(np.sum((p1 - p2) ** 2)))
        match_idx = dis.index(min(dis))
        print(j, '→', match_idx)
        if (int(j / 15) == match_idx):
            match += 1
    print('正确率: %f %%' % (match / len(res2) * 100))
```

4 任务结果

使用以下14篇小说：

图2

设定获得的主题数为15，获得每个主题对应的8个最大概率词汇分别为：

```
Topic #0:
辨认出来 一十六年 温文守 催越 楚楚动人 杂耍 好要 乘人
Topic #1:
辨认出来 一十六年 温文守 催越 楚楚动人 杂耍 好要 乘人
Topic #2:
郭靖 黄蓉 洪七公 说道 欧阳锋 师父 黄药师 武功
Topic #3:
杨过 小龙女 说道 李莫愁 武功 心中 郭靖 黄蓉
Topic #4:
说道 令狐冲 一声 师父 武功 不知 弟子 心中
Topic #5:
陈家洛 张召重 说道 徐天宏 霍青桐 余鱼同 文泰来 乾隆
Topic #6:
辨认出来 一十六年 温文守 催越 楚楚动人 杂耍 好要 乘人
Topic #7:
韦小宝 说道 袁承志 皇上 皇帝 心想 康熙 一声
Topic #8:
胡斐 说道 程灵素 苗人凤 心中 两人 一声 袁紫衣
Topic #9:
遗著 凤家 骨灰坛 差使 青砖 强盗 大伙 勾结
Topic #10:
秦桧 故事 虬髯 不知 杨行密 高宗 皇帝 说道
Topic #11:
狄云 李文秀 水笙 师父 万震山 心中 丁典 说道
Topic #12:
辨认出来 一十六年 温文守 催越 楚楚动人 杂耍 好要 乘人
Topic #13:
张无忌 张翠山 谢逊 周芷若 赵敏 明教 教主 殷素素
Topic #14:
石破天 说道 雪山 丁当 白万剑 石清 帮主 武功
```

其中有几个topic的最大词汇可以让人很明显地分析出与该主题对应的小说，例如主题2对应《射雕英雄传》，主题3对应《神雕侠侣》，主题4对应《笑傲江湖》，主题5对应《书剑恩仇录》，主题7对应《鹿鼎记》，主题10对应《三十三剑客图》，主题13对应《倚天屠龙记》，主题14对应《侠客行》等，但还有近一半的主题与小说无法对应，而且有少数几个最大概率词汇一样的主题。

用 0-14 分别表示图2中的自上而下的15篇小说，将每篇小说对应概率最大的 Topic 打出：

```
Topic #0:
[]
Topic #1:
[]
Topic #2:
[5]
Topic #3:
[8]
Topic #4:
[3, 4, 9, 11, 13]
Topic #5:
[1]
Topic #6:
[]
Topic #7:
[7, 14]
Topic #8:
[12]
Topic #9:
[]
Topic #10:
[0]
Topic #11:
[6, 10]
Topic #12:
[]
Topic #13:
[]
Topic #14:
[2]
```

与我们人为分类的基本一致。分类的正确率达到了80.444444 %。其中又考虑到《飞狐外传》是《雪山飞狐》的前传，因此两本小说概率向量会很接近，成为一种干扰。