【NLP作业-03】LDA模型

杨思捷 ZY2103533

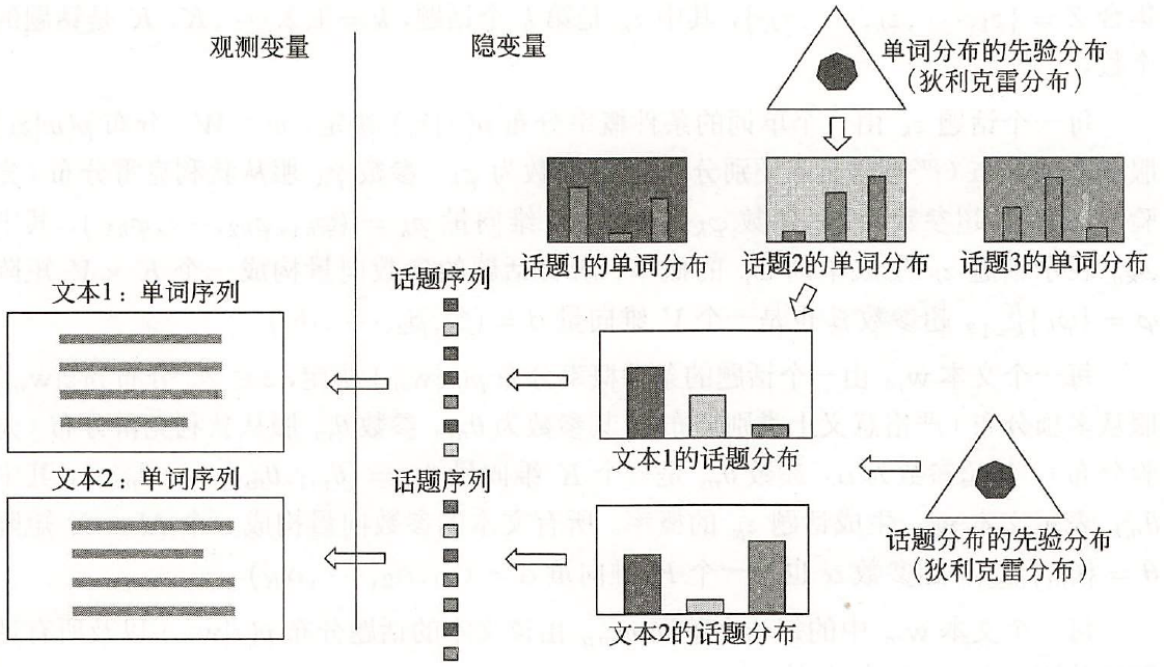
1. 引言

本次作业使用LDA模型对金庸小说进行分析，提取出了高频词，并利用SVM通过文本参数对文本进行分类。

2. LDA模型

潜在狄利克雷分布（Latent Dirichlet Allocation，LDA）是文本是文本集合的生成概率模型。模型假设话题由单词的多项分布表示，文本由话题的多项分布表示，单词分布和话题分布的先验分布都是狄利克雷分布。文本内容的不同是由于它们的话题分布不同。

LDA的文本生成过程如下图



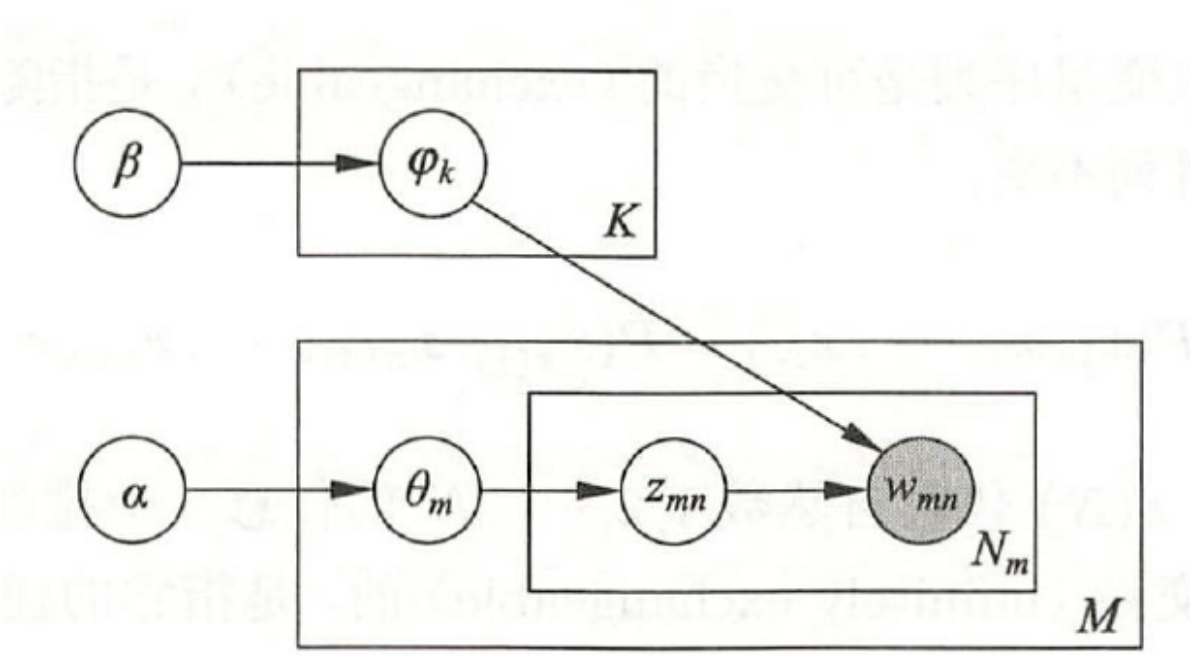
数学描述如下：

LDA一般需要使用三个集合，首先是单词集合$W = \{w_1, \cdots, w_i, \cdots, w_V\}$，其中$w_i$是第$i$个单词。其次是文本集合$W = \{\mathbf{w}_1, \cdots, \mathbf{w}_m, \cdots, \mathbf{w}_M\}$，其中$\mathbf{w}_i$是第$i$个文本。文本$\mathbf{w}_m$是一个单词序列，$\mathbf{w}_m = (w_{m1}, \cdots, w_{mn}, \cdots, w_{mN_m})$。最后是话题集合$Z = \{z_1, \cdots, z_k, \cdots, z_K\}$

每一个话题z由一个单词的条件概率分布$p(w|z)$决定，$w \in W$。分布$p(w|k)$服从多项分布，其参数为φ_k。参数φ_k是一个V维向量$\varphi_k = (\varphi_{k1}, \cdots, \varphi_{kV})$，服从狄利克雷分布，其超参数为$\beta$。其中$\varphi_{kv}$表示话题$z_k$生成单词$w_v$的概率所有话题的参数向量构成一个$K \times V$的矩阵，超参数$\beta$也是一个$V$维向量，$\beta = (\beta_1, \cdots, \beta_V)$

每一个文本$\mathbf{w}$由一个话题的条件概率分布$p(z|\mathbf{w})$决定，$z \in Z$。分布$p(z|\mathbf{w})$服从多项分布，其参数为θ_m。参数θ_m是一个K维向量$\theta_m = (\theta_{m1}, \cdots, \theta_{mK})$，服从狄利克雷分布，其超参数为$\alpha$。其中$\theta_{mk}$表示文本$\mathbf{w}_m$生成话题$z_k$的概率所有话题的参数向量构成一个$M \times K$的矩阵，超参数$\alpha$也是一个$K$维向量，$\alpha = (\alpha_1, \cdots, \alpha_K)$

LDA的概览图如下，每一个文本$\mathbf{w}_m$中的每一个单词w_{nm}由该文本的话题分布$p(z|\mathbf{w}_m)$以及所有话题的单词分布$p(w|z_k)$决定。



3. 吉布斯抽样

LDA的参数估计是一个复杂的最优化问题，很难精确求解，一般只能近似求解。常用的近似求解方法为吉布斯抽样，吉布斯抽样的实现简单但是迭代次数可能会比较多。

LDA模型的学习通常采用收缩的吉布斯抽样方法，基本想法是，通过对隐变量θ和φ积分，得到边缘概率分布，$p(\mathbf{w},\mathbf{z}|\alpha,\beta)$，其中变量$\mathbf{w}$是可观测的，变量$\mathbf{z}$是不可观测的；对后验概率分布$p(\mathbf{z}|\mathbf{w},\alpha,\beta)$进行吉布斯抽样，得到后验概率分布的样本集合；再利用这个样本集合对参数θ和φ进行估计，最终得到LDA模型的参数估计。

这部分的算法流程参考了李航老师的统计学习方法20.3章中的算法20.2

算法 20.2（LDA 吉布斯抽样算法）

输入：文本的单词序列 $\mathbf{w}=\{\mathbf{w}_1,\cdots,\mathbf{w}_m,\cdots,\mathbf{w}_M\}$，$\mathbf{w}_m=(w_{m1},\cdots,w_{mn},\cdots,w_{mN_m})$；

输出：文本的话题序列 $\mathbf{z}=\{\mathbf{z}_1,\cdots,\mathbf{z}_m,\cdots,\mathbf{z}_M\}$，$\mathbf{z}_m=(z_{m1},\cdots,z_{mn},\cdots,z_{mN_m})$ 的后验概率分布 $p(\mathbf{z}|\mathbf{w},\alpha,\beta)$ 的样本计数，模型的参数 φ 和 θ 的估计值；

参数：超参数 α 和 β，话题个数 K。

（1）设所有计数矩阵的元素 n_{mk}，n_{kv}，计数向量的元素 n_m，n_k 初值为 0；

（2）对所有文本 $\mathbf{w}_m$，$m=1,2,\cdots,M$

对第 m 个文本中的所有单词 w_{mn}，$n=1,2,\cdots,N_m$

（a）抽样话题 $z_{mn}=z_k\sim \mathrm{Mult}\left(\frac{1}{K}\right)$；

增加文本-话题计数 $n_{mk}=n_{mk}+1$，

增加文本-话题和计数 $n_m=n_m+1$，

增加话题-单词计数 $n_{kv}=n_{kv}+1$，

增加话题-单词和计数 $n_k=n_k+1$；

（3）循环执行以下操作，直到进入燃烧期

对所有文本 $\mathbf{w}_m$，$m=1,2,\cdots,M$

对第 m 个文本中的所有单词 w_{mn}，$n=1,2,\cdots,N_m$

（a）当前的单词 w_{mn} 是第 v 个单词，话题指派 z_{mn} 是第 k 个话题；

减少计数 $n_{mk}=n_{mk}-1$，$n_m=n_m-1$，$n_{kv}=n_{kv}-1$，$n_k=n_k-1$；

（b）按照满条件分布进行抽样

$$p(z_i|\mathbf{z}_{-i},\mathbf{w},\alpha,\beta)\propto\frac{n_{kv}+\beta_v}{\sum_{v=1}^{V}(n_{kv}+\beta_v)}\cdot\frac{n_{mk}+\alpha_k}{\sum_{k=1}^{K}(n_{mk}+\alpha_k)}$$

得到新的第 k' 个话题，分配给 z_{mn}；

（c）增加计数 $n_{mk'}=n_{mk'}+1$，$n_m=n_m+1$，$n_{k'v}=n_{k'v}+1$，$n_{k'}=n_{k'}+1$；

（d）得到更新的两个计数矩阵 $N_{K\times V}=[n_{kv}]$ 和 $N_{M\times K}=[n_{mk}]$，表示后验概率分布 $p(\mathbf{z}|\mathbf{w},\alpha,\beta)$ 的样本计数；

（4）利用得到的样本计数，计算模型参数

$$\theta_{mk}=\frac{n_{mk}+\alpha_k}{\sum_{k=1}^{K}(n_{mk}+\alpha_k)}$$

$$\varphi_{kv}=\frac{n_{kv}+\beta_v}{\sum_{v=1}^{V}(n_{kv}+\beta_v)}$$

通过计算困惑度来评估LDA的参数估计效果

$$perplexity=exp(-\frac{\sum\log\left(p(w)\right)}{N})$$

4. 分类

使用SVM对一系列段落进行分类

5. 编程实现及结果分析

5.1. 段落选取及预处理

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。

此处的处理是选定字数最多的5本小说，分别为鹿鼎记，天龙八部，笑傲江湖，倚天屠龙记，神雕侠侣，然后对五本小说约以2000字为长度进行分段，每本小说选取40段总共选取200段个进行下一步的处理。

这部分的代码实现对应`gen_para.py`

```
# 预处理函数，将原始文本处理为断句断好的列表
def getCorpus(text_raw):
    text_raw = re_preprocess.sub("",text_raw)
    punctuationL =["\t","\n","\u3000","\u0020","\u00A0"," "]
    for i in punctuationL:
        text_raw = text_raw.replace(i,"")
    text_raw = text_raw.replace(", ",",。")
    corpus = text_raw.split("。")
    return corpus
```

5.2. LDA学习和吉布斯抽样

输入一系列文本单词序列，通过`jieba`库进行分词，并去除一些无意义的停用词。为每个词分配一个唯一的id，并通过词的id将语料库重新表达。

```
    def gen_dict(self,documentL):
        word2id_dict = {}
        id2word_dict = {}
        docs = []
        cnt_document = []
        cnt_word_id = 0

        for document in documentL:
            segList = jieba.cut(document)
            for word in segList:
                word = word.strip()
                if len(word) > 1 and word not in stopWordL:
                    if word in word2id_dict:
                        cnt_document.append(word2id_dict[word])
                    else:
                        cnt_document.append(cnt_word_id)
                        word2id_dict[word] = cnt_word_id
                        id2word_dict[cnt_word_id] = word
                        cnt_word_id += 1
            docs.append(cnt_document)
            cnt_document = []
        self.docs, self.word2id_dict, self.id2word_dict = docs, word2id_dict,
id2word_dict
        self.num_doc = len(self.docs)
        self.num_word = len(self.word2id_dict)
```

然后根据产生的字典生成对应的参数向量，并进行随机初始化

```
    def param_init(self):
        # # 各文档的词在各主题上的分布数目
        self.ndz = np.zeros([self.num_doc,self.num_topic]) + alpha
        # # 词在主题上的分布数
        self.nzw = np.zeros([self.num_topic,self.num_word]) + beta
        # # 每个主题的总词数
        self.nz = np.zeros([self.num_topic]) + self.num_word*beta
        # theta = np.zeros([num_doc,num_topic])
        self.phi = np.zeros([self.num_topic,self.num_word])


        for d, doc in enumerate(docs):
            zCurrentDoc = []
            for w in doc:
                self.pz = np.divide(np.multiply(self.ndz[d, :], self.nzw[:, w]),
self.nz)
                z = np.random.multinomial(1, self.pz / self.pz.sum()).argmax()
                zCurrentDoc.append(z)
                self.ndz[d, z] += 1
                self.nzw[z, w] += 1
                self.nz[z] += 1
            self.Z.append(zCurrentDoc)
```

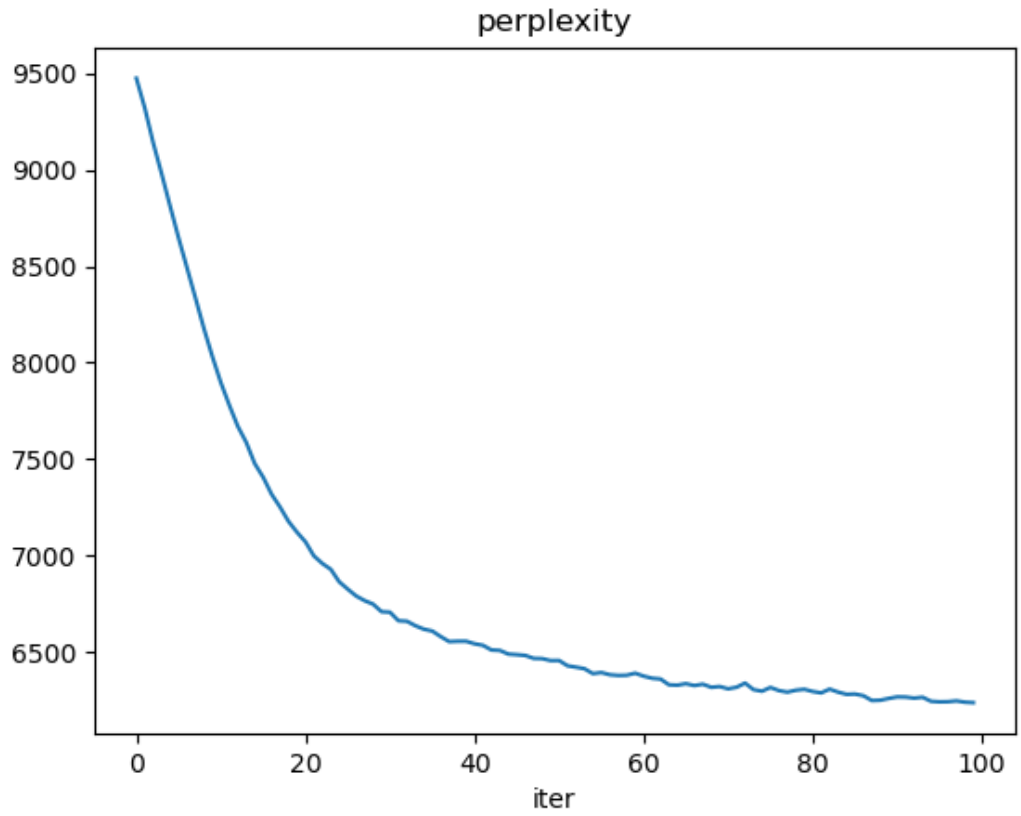
进行吉布斯采样并更新参数向量

```
    # gibbs采样
    def gibbs_sampling(self):
        # 为每个文档中的每个单词重新采样topic
        for d, doc in enumerate(self.docs):
            for index, w in enumerate(doc):
                z = self.Z[d][index]
                # 将当前文档当前单词原topic相关计数减去1
                self.ndz[d,z] -= 1
                self.nzw[z,w] -= 1
                self.nz[z] -= 1
                # 重新计算当前文档当前单词属于每个topic的概率
                self.pz = np.divide(np.multiply(self.ndz[d,:], self.nzw[:,w]),
self.nz)
                # 按照计算出的分布进行采样
                z = np.random.multinomial(1, self.pz / self.pz.sum()).argmax()
                self.Z[d][index] = z
                # 将当前文档当前单词新采样的topic相关计数加上1
                self.ndz[d, z] += 1
                self.nzw[z, w] += 1
                self.nz[z] += 1
        self.theta = [(self.ndz[i]+alpha)/(len(self.docs[i])+num_topic*alpha) for
i in range(self.num_doc)]
        self.phi = [(self.nzw[i]+beta)/(self.nz[i]+self.num_word*beta) for i in
range(num_topic)]
```

5.3. LDA结果分析

此处的处理的结果来自鹿鼎记，天龙八部，笑傲江湖，倚天屠龙记，神雕侠侣等5本小说，每本小说有40个分段总计200个分段的的文本。

可以看出，经过60步迭代后困惑度基本趋于收敛。



各主题的前十高频词如下：

Topic1	杨过	小龙女	李莫愁	法王	蒙古	郭靖	周伯通	黄蓉	金轮法王	陆无双
Topic2	虚竹	少林寺	和尚	少林	师兄	鸠摩智	僧人	喇嘛	方丈	武功
Topic3	教主	兄弟	二人	微笑	公孙止	裘千尺	女儿	刘一舟	性命	大哥
Topic4	令狐冲	岳不群	恒山	剑法	盈盈	师妹	岳灵珊	长剑	林平之	华山派
Topic5	少女	段誉	女子	姑娘	爹爹	张翠山	段正淳	夫人	王语嫣	木婉清
Topic6	说道	一个	师父	一声	心中	突然	不敢	心想	便是	只见
Topic7	韦小宝	公主	皇上	太后	皇帝	康熙	侍卫	吴三桂	说道	台湾
Topic8	萧峰	丐帮	刘正风	群雄	乔峰	弟子	魔教	将军	今日	盟主
Topic9	武功	剑法	一招	长剑	高手	这位	女郎	大师	乃是	一拳
Topic10	张无忌	谢逊	赵敏	灭绝师太	明教	胡青牛	不知	教主	周芷若	魔教

根据对金庸小说的印象，这些高频词可以体现出主题的内容。

5.4. SVM分类

通过$\theta_m = (\theta_{m1}, \cdots, \theta_{mK})$进行分类，代码如下：

```
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier

from sklearn import metrics
import numpy as np
```

```
data_np = np.loadtxt("theta_000.csv",delimiter=",")

# print(data_np.shape)
label = []
for i in range(5):
    label = label + [i]*40

X_train, X_test, y_train, y_test = train_test_split(data_np, label, test_size=.2,
random_state=10)
# 训练模型
model = OneVsRestClassifier(svm.SVC(kernel='linear', probability=True))

clt = model.fit(X_train, y_train)


y_test_pred = clt.predict(X_test)
ov_acc = metrics.accuracy_score(y_test_pred, y_test)
print("overall accuracy: %f" % (ov_acc))
print("==========================================")
acc_for_each_class = metrics.precision_score(y_test, y_test_pred, average=None)
print("acc_for_each_class:\n", acc_for_each_class)
print("==========================================")
avg_acc = np.mean(acc_for_each_class)
print("average accuracy:%f" % (avg_acc))
```

5.5. SVM分类结果分析

对不同迭代次数的θ_m进行分类，分类结果如下：

迭代次数	鹿鼎记	天龙八部	笑傲江湖	倚天屠龙记	神雕侠侣	总体准确度
10	77.78%	60.00%	33.33%	50.00%	100.00%	60.00%
20	100.00%	85.71%	66.67%	66.67%	85.71%	80.00%
30	100.00%	88.89%	85.71%	77.78%	100.00%	90.00%
60	100.00%	83.33%	85.71%	100.00%	100.00%	92.50%
90	100.00%	90.91%	85.71%	100.00%	100.00%	95.00%

可以看出，随着迭代次数增加，总体准确度有所提高。但是天龙八部和笑傲江湖的区分度还有一定差异，原因可能在于停用词不够全面，例如上面的Topic6的结果是许多常用词，不能体现主题的具体内容，可以考虑加入更多的停用词进行限制。

6. 参考资料

https://zhuanlan.zhihu.com/p/31470216

李航. 统计学习方法 第二版〔M〕.北京：清华大学出版社 2019.127