深度学习与自然语言处理第三次作业

李文雯 ZY2203106

作业要求

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 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。

主要方法

M1: LDA 描述

LDA（Linear Discriminant Analysis），是一种文档主题生成模型，，它可以将文档中每篇文档的主题按照概率分布的形式给出。也称为一个三层贝叶斯概率模型，包含词、主题和文档三层结构。所谓生成模型，就是说，我们认为一篇文章的每个词都是通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语”这样一个过程得到。文档到主题服从多项式分布，主题到词服从多项式分布。

LDA 是一种非监督机器学习技术，可以用来识别大规模文档集（document collection）或语料库（corpus）中潜藏的主题信息。它采用了词袋（bag of words）的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。但是词袋方法没有考虑词与词之间的顺序，这简化了问题的复杂性，同时也为模型的改进提供了契机。每一篇文档代表了一些主题所构成的一个概率分布，而每一个主题又代表了很多单词所构成的一个概率分布。

LDA 的核心思想是寻找到最佳的投影方法，将高维的样本投影到特征空间(feature space)，使得不同类别间的数据“距离”最大，而同一类别内的数据“距离”最小。

M2: LDA 模型生成

首先定义文章集合为 Doc,文章主题集合为 Topic，Doc 中的每个文档 doc 可以看作为一个单词序列<w1,w2,….wn>，其中 wi 表示为第 i 个单词，doc 共有 n 个单词。

Doc 中的所有不同单词组成一个集合 Voc，LDA 模型以文档集合 Doc 作为输入，最终训练出两个结果向量，k 表示 Topi 词，m 表示 Voc 中包含的词语数量。

对每个 Doc 中对应到不同 Topic 的概率θd=<$p_{t1},\ldots,p_{tk}$>,其中 Pt_i 表示 doc 对应 Topic 中第 i 个 Topic 词的概率。其中 $p_{t_i} = \frac{n_{t_i}}{n}$，$n_{t_i}$ 表示 doc 中对应的第 i 个 Topic

的词的数目，n 表示 doc 中所有词的总数。

对每个 Topic 中的 Topic，生成不同单词的概率 $\varphi_t = <p_{w_1},\ldots,p_{w_m}>$，其中 p_{w_i} 表示 t 生成 Voc 中的第 i 个单词的概率。

其中 $p_{w_i} = \frac{N_{w_i}}{N}$，其中 N_{w_i} 表示对应到 Topic 的 VOC 中第 i 个单词的数目，N 表示所有对应到 Topic 的单词总数。

LDA 的核心公式如下：

$p(w|d) = p(w|t) * p(t|d)$

公式以 Topic 作为中间层，通过当前的 θ_d 和 φ_t 给出了文档 d 中出现单词 w 的概率。其中的 p(t| d) 可通过 θ_d 计算得到，p(w| t) 利用 φ_t 计算得到。因此利用当前的 θ_d 和 φ_t 我们可以为一个文档中的单词计算它对应任意一个 Topic 时的 p(w| d) 值，然后根据这些结果来更新这个词对应的 Topic。相对应的，如果这个更新改变了这个单词所对应的 Topic 值，反过来也会影响 θ_d 和 φ_t。

M3：算法流程

step1：数据处理

所用的数据集为金庸先生的 16 本小说，首先对数据集进行预处理：1.删除开头的无关信息；2.删除文中的标点符号；3.删除隐藏符号（换行符、分页符等）；4.删除所有中文停词。

在分词模式下，用 Python 中的中文分词组件 Jieba 的精确分词模式对预处理后的数据集进行分词；在分字模式下，将每个字试做分的词。

对给定语料库进行分析可知，语料库内共有 16 篇文章，从每一篇文章内抽取 13 个段落，共有 208 个段落；对每篇文章分词或分字之后，将一篇文章的总词数除以 13，即每篇文章共有 13 个区间，所选取的段落为每个区间抽取前 500 个词。

对训练集和测试集数据预处理代码如下，二者区别为训练集中选取的各段落中的第 0-500 个单词，测试集中选取的是第 501-1000 个单词。

训练集：

```
def read_data(path, mode):  # 读取语料内容
    content = []
    con_list = []
    names = os.listdir(path)
    r = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@，。?★、…【】《》？“”‘’！
[\\]^_`{|}~「」『』（）]+'
    for name in names:
        con_temp = []
        novel_name = path + '\\' + name
        with open(novel_name, 'r', encoding='ANSI') as f:
            corpus = f.read()
            corpus = re.sub(r, '', corpus)
            corpus = corpus.replace('\n', '')
            corpus = corpus.replace('\u3000', '')
            corpus = corpus.replace('本书来自 www.cr173.com 免费 txt 小说下载
```

```
站\n 更多更新免费电子书请关注 www.cr173.com', '')
            if mode == 'token':
                con = jieba.lcut(corpus)
                con_list = list(con)
            elif mode == 'char':
                con_list = list(corpus)
                con=corpus
            pos = int(len(con)//13) ####16 篇文章，分词后，每篇均匀选取 13 个
500 词段落进行建模
            for i in range(13):
                con_temp = con_temp + con_list[i*pos:i*pos+500]
            content.append(con_temp)
        f.close()
    return content, names
```

测试集：

```
    def read_data_test(path,mode):  # 读取语料内容
    content = []
    con_list = []
    names = os.listdir(path)
    r = u'[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@，。?★、…【】《》？“”‘’！
[\\]^_`{|}~「」『』（）]+'
    for name in names:
        con_temp = []
        novel_name = path + '\\' + name
        with open(novel_name, 'r', encoding='ANSI') as f:
            corpus = f.read()
            corpus = re.sub(r, '', corpus)
            corpus = corpus.replace('\n', '')
            corpus = corpus.replace('\u3000', '')
            corpus = corpus.replace('本书来自 www.cr173.com 免费 txt 小说下载
站\n 更多更新免费电子书请关注 www.cr173.com', '')
            if mode == 'token':
                con = jieba.lcut(corpus)
                con_list = list(con)
            elif mode == 'char':

                con_list = list(corpus)
                con=corpus

            pos = int(len(con)//13) ####16 篇文章，分词后，每篇均匀选取 13 个
500 词段落进行建模
            for i in range(13):
                con_temp = con_temp + con_list[i*pos+501:i*pos+1000]
            content.append(con_temp)
```

```
        f.close()
    return content, names
```

step2：模型训练

首先进行初始化，每篇文章的每个词语随机赋予一个初始的 Topic 值，然后分别统计每篇文章的总词数、每篇文章的词频、每个 Topic 的总词数、每个 Topic 的词频；再计算每个 topic 被选中的概率，然后进行迭代，训练模型，具体步骤如下所示：

```
    [data_txt, files] = read_data("金庸小说集",mode="char")
Topic_All = []  # 每篇文章中的每个词来自哪个 topic
Topic_count = {}  # 每个 topic 有多少词
Topic_fre0 = {}; Topic_fre1 = {}; Topic_fre2 = {}; Topic_fre3 = {};
Topic_fre4 = {}; Topic_fre5 = {}; Topic_fre6 = {}; Topic_fre7 = {};
Topic_fre8 = {}; Topic_fre9 = {}; Topic_fre10 = {}; Topic_fre11 = {};
Topic_fre12 = {}; Topic_fre13 = {}; Topic_fre14 = {}; Topic_fre15 = {}
# 每个 topic 的词频
Doc_count = []  # 每篇文章中有多少个词
Doc_fre = []  # 每篇文章有多少各个 topic 的词
i = 0
for data in data_txt:
    topic = []
    docfre = {}
    for word in data:
        a = random.randint(0, len(data_txt)-1)  # 为每个单词赋予一个随机初始
topic
        topic.append(a)
        if '\u4e00' <= word <= '\u9fa5':
            Topic_count[a] = Topic_count.get(a, 0) + 1  # 统计每个 topic 总
词数
            docfre[a] = docfre.get(a, 0) + 1  # 统计每篇文章的词频
            exec('Topic_fre{}[word]=Topic_fre{}.get(word, 0) +
1'.format(i, i))  # 统计每个 topic 的词频
    Topic_All.append(topic)
    docfre = list(dict(sorted(docfre.items(), key=lambda x: x[0],
reverse=False)).values())
    Doc_fre.append(docfre)
    Doc_count.append(sum(docfre))  # 统计每篇文章的总词数
    # exec('print(len(Topic_fre{}))'.format(i))
    i += 1
Topic_count = list(dict(sorted(Topic_count.items(), key=lambda x: x[0],
reverse=False)).values())

Doc_fre = np.array(Doc_fre)  # 转为 array 方便后续计算
Topic_count = np.array(Topic_count)  # 转为 array 方便后续计算
Doc_count = np.array(Doc_count)  # 转为 array 方便后续计算
```

```
# print(Doc_fre)
# print(Topic_count)
# print(Doc_count)

Doc_pro = []  # 每个topic被选中的概率
Doc_pronew = []  # 记录每次迭代后每个topic被选中的新概率
for i in range(len(data_txt)):
    doc = np.divide(Doc_fre[i], Doc_count[i])
    Doc_pro.append(doc)
Doc_pro = np.array(Doc_pro)
print(Doc_pro)
stop = 0  # 迭代停止标志
loopcount = 1  # 迭代次数
while stop == 0:
    i = 0
    for data in data_txt:
        top = Topic_All[i]
        for w in range(len(data)):
            word = data[w]
            pro = []
            topfre = []
            if '\u4e00' <= word <= '\u9fa5':
                for j in range(len(data_txt)):
                    exec('topfre.append(Topic_fre{}.get(word,
0))'.format(j))  # 读取该词语在每个topic中出现的频数
                pro = Doc_pro[i] * topfre / Topic_count  # 计算每篇文章选中
各个topic的概率乘以该词语在每个topic中出现的概率，得到该词出现的概率向量
                m = np.argmax(pro)  # 认为该词是由上述概率之积最大的那个topic
产生的
                Doc_fre[i][top[w]] -= 1  # 更新每个文档有多少各个topic的词
                Doc_fre[i][m] += 1
                Topic_count[top[w]] -= 1  # 更新每个topic的总词数
                Topic_count[m] += 1
                exec('Topic_fre{}[word] = Topic_fre{}.get(word, 0) -
1'.format(top[w], top[w]))  # 更新每个topic该词的频数
                exec('Topic_fre{}[word] = Topic_fre{}.get(word, 0) +
1'.format(m, m))
                top[w] = m
        Topic_All[i] = top
        i += 1
    # print(Doc_fre, 'new')
    # print(Topic_count, 'new')
    if loopcount == 1:  # 计算新的每篇文章选中各个topic的概率
        for i in range(len(data_txt)):
```

```
            doc = np.divide(Doc_fre[i], Doc_count[i])
            Doc_pronew.append(doc)
        Doc_pronew = np.array(Doc_pronew)
    else:
        for i in range(len(data_txt)):
            doc = np.divide(Doc_fre[i], Doc_count[i])
            Doc_pronew[i] = doc
    # print(Doc_pro)
    # print(Doc_pronew)
    if (Doc_pronew == Doc_pro).all():  # 如果每篇文章选中各个topic的概率不再变化，则认为模型已经训练完毕
        stop = 1
    else:
        Doc_pro = Doc_pronew.copy()
    loopcount += 1
print(Doc_pronew)  # 输出最终训练的到的每篇文章选中各个topic的概率
print(loopcount)  # 输出迭代次数

print('模型训练完毕！')
```

step3：模型测试

仍然在对应的 16 部小说中选择段落作为测试集，在之前训练集中选取的是 208 个段落中的第 0-500 个单词，因此在测试集中选取第 501-1000 单词形成测试段落，最终形成待分类文章。

```
[test_txt, files] = read_data_test("金庸小说集",mode="char")
Doc_count_test = []  # 每篇文章中有多少个词
Doc_fre_test = []  # 每篇文章有多少各个topic的词
Topic_All_test = []  # 每篇文章中的每个词来自哪个topic
i = 0
for data in test_txt:
    topic = []
    docfre = {}
    for word in data:
        a = random.randint(0, len(data_txt) - 1)  # 为每个单词赋予一个随机初始topic
        topic.append(a)
        if '\u4e00' <= word <= '\u9fa5':
            docfre[a] = docfre.get(a, 0) + 1  # 统计每篇文章的词频
    Topic_All_test.append(topic)
    docfre = list(dict(sorted(docfre.items(), key=lambda x: x[0], reverse=False)).values())
    Doc_fre_test.append(docfre)
    Doc_count_test.append(sum(docfre))  # 统计每篇文章的总词数
```

```
    i += 1

Doc_fre_test = np.array(Doc_fre_test)
Doc_count_test = np.array(Doc_count_test)
# print(Doc_fre_test)
# print(Doc_count_test)
Doc_pro_test = []  # 每个topic被选中的概率
Doc_pronew_test = []  # 记录每次迭代后每个topic被选中的新概率
for i in range(len(test_txt)):
    doc = np.divide(Doc_fre_test[i], Doc_count_test[i])
    Doc_pro_test.append(doc)
Doc_pro_test = np.array(Doc_pro_test)
# print(Doc_pro_test)
stop = 0  # 迭代停止标志
loopcount = 1  # 迭代次数
while stop == 0:
    i = 0
    for data in test_txt:
        top = Topic_All_test[i]
        for w in range(len(data)):
            word = data[w]
            pro = []
            topfre = []
            if '\u4e00' <= word <= '\u9fa5':
                for j in range(len(data_txt)):
                    exec('topfre.append(Topic_fre{}.get(word,
0))'.format(j))  # 读取该词语在每个topic中出现的频数
                pro = Doc_pro_test[i] * topfre / Topic_count  # 计算每篇文
章选中各个topic的概率乘以该词语在每个topic中出现的概率，得到该词出现的概率向量
                m = np.argmax(pro)  # 认为该词是由上述概率之积最大的那个topic
产生的
                Doc_fre_test[i][top[w]] -= 1  # 更新每个文档有多少各个topic
的词
                Doc_fre_test[i][m] += 1
                top[w] = m
        Topic_All_test[i] = top
        i += 1
    print(Doc_fre_test, 'new')
    if loopcount == 1:  # 计算新的每篇文章选中各个topic的概率
        for i in range(len(test_txt)):
            doc = np.divide(Doc_fre_test[i], Doc_count_test[i])
            Doc_pronew_test.append(doc)
        Doc_pronew_test = np.array(Doc_pronew_test)
    else:
```

```
        for i in range(len(test_txt)):
            doc = np.divide(Doc_fre_test[i], Doc_count_test[i])
            Doc_pronew_test[i] = doc
    # print(Doc_pro_test)
    # print(Doc_pronew_test)
    if (Doc_pronew_test == Doc_pro_test).all():  # 如果每篇文章选中各个
topic 的概率不再变化，则认为训练集已分类完毕
        stop = 1
    else:
        Doc_pro_test = Doc_pronew_test.copy()
    loopcount += 1
print(Doc_pronew)
print(Doc_pronew_test)
print(loopcount)
print('测试集测试完毕！')
```

得到测试结果后，则需要根据这些概率结果，区分每篇待分类文章究竟来自哪一本小说，此处采用的是欧式距离的方式，即比较待分类文章与已知的小说，两者对于各个 Topic 的概率向量之间的距离最近，即认为是来自该本小说。

```
    for k in range(len(test_txt)):
    pro = []
    for i in range(len(data_txt)):
        dis = 0
        for j in range(len(data_txt)):
            dis += (Doc_pro[i][j] - Doc_pro_test[k][j])**2  #计算欧式距离
        pro.append(dis)
    m = pro.index(min(pro))
    result.append(m)
print(result)
```

实验结果

本次应用了 LDA 模型对语料进行了模型训练与测试，实验结果如下表所示：

Table 1：以字为基本单元的分类结果

主题个数	正确分类个数	错误分类个数	准确率	平均准确率
16	160	48	76.9%	78.5%
	164	44	78.8%	
	166	42	79.8%	
30	188	20	90.4%	89.57%
	185	23	88.9%	
	186	22	89.6%	
50	194	14	93.3%	94.23%
	198	10	95.2%	
	196	12	94.2%	

Table 2：以词为基本单元的分类结果

主题个数	正确分类个数	错误分类个数	正确率	平均正确率
16	108	100	51.92%	49.84%
	97	111	46.63%	
	106	102	50.96%	
30	145	65	69.71%	72.12%
	155	53	74.52%	
	150	58	72.12%	
50	170	38	81.73%	83.65%
	179	29	86.06%	
	173	35	83.17%	

结论

1. 从表中可以发现，在不同数量的主题个数下，分类性能有所变化。当主题个数较少时，分类结果可能比较模糊，难以区分不同小说之间的差异。当主题个数较多时，分类结果可能过于细致,难以将不同小说归为同一类别。因此,我们需要根据具体情况选择合适的主题个数。
2. 对比表 1 和表 2，以"词"和以"字"为基本单元下的分类结果有所不同。以"字"为基本单元下的分类结果可能更加细致，能够更好地反映不同小说之间的差异。但是，以"字"为基本单元下的分类结果可能也更加复杂，需要更多的主题个数来进行区分。因此，我们需要根据具体情况选择合适的基本单元和主题个数。

参考文档

[1] https://blog.csdn.net/shzx_55733/article/details/116280982?spm=1001.2014.3001.5501

[2] (57 条消息）深度学习与自然语言处理第三次作业——LDA 段落主题分布问题 lda 主题分布_荤荤大端荤荤的博客-CSDN 博客