DLNLP_2023_Homework3 王彪--ZY2203114

作业三：从上面链接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，（1）在不同数量的主题个数下分类性能的变化；（2）以"词"和以"字"为基本单元下分类结果有什么差异

1. LDA模型

LDA是一款基于Dirichlet分布的概率主题模型，运用了概率论和词袋模型等知识。从作业内容分析推导这一模型，数据集为金庸的16本小说。本文从小说中随机抽取M个段落，即得到M篇文档，对应第m个文档中有n_m个词，假定每篇文档中的词都是从一系列主题中选择出来的，每个主题下有对应的词，文档中一个词$w_m n$的产生要经历两个步骤，首先从该文档的主题分布中采样一个主题，然后在这个主题对应的词分布中采样一个词。不断重复直到m个文档都完成上述过程。

随机生成过程的两个步骤都符合Dirichlet分布：

- 首先，抽主题：α 是某文档主题的Dirichlet分布的超参数，采样得到文档m的主题分布 θ_m：$(p_1, p_2, \ldots, p_k)$，k是主题个数，采样出某一主题，得到文档m第n个词的主题编号Z_{mn}；
- 然后，抽主题下的词：β是某主题的词语的Dirichlet分布的超参数，采样得到所有主题词语的分布ϕ，得到主题1$(p_{11}, p_{12}, \ldots, p_{1v})$，.....主题k$(p_{k1}, p_{k2}, \ldots, p_{kv})$，从第一步主题编号$Zmn$对应的主题分布中抽样得到词语$W_{mn}$.

  这一过程中，w_{mn}是可以观察到的已知变量，α, β是跟据经验给定的先验参数，其他变量Z_{mn}, θ, ϕ都是未知变量，需要根据观察到的变量学习，这里使用**Gibbs Sampling**算法，其运行方式为，每次抽取概率向量的一个维度，未定其他维度的变量，采样当前维度的值，不断迭代，直到收敛。初始时，随机给文本中的每个词分配主题$z^{(0)}$，然后统计每个主题z下词出现的概率，及每个文档m下出现主题z的数量，以及每个主题下词的总量，据此估计排除当前词的主题分布，然后根据这个分布为该词采样新主题，最终每个文档的主题分布θ_m，每个主题的词分布 收敛，算法停止。

2. 分类模型

本实验选用支持向量机对文本段落进行多分类，核函数选择线性核。

3. 代码

3.1 数据准备

预处理代码与第一次作业类似，略。

200个段落的选择代码如下：

```
def get_segment(file,len,flag):
    seg = []
    with open(file, 'r') as f:
        txt_list = f.readline().split(',')
        file_path = txt_list[random.randint(0, len(txt_list)-1)] + '.txt'
        split_words = get_single_corpus(DATA_PATH + file_path, flag)
        begin = random.randint(0, len(split_words)-len-1)
        seg.extend(split_words[begin:begin+len])
    return file_path, seg
```

3.2 LDA模型

3.2.1 初始化

```
def initialize():
    for f_idx, segment in enumerate(segments):
        temp_word2topic = []
        for w_idx, word in enumerate(segment):
            init_topic = random.randint(0, topic_num-1)
            file2topic[f_idx, init_topic] += 1
            topic2word[init_topic, word] += 1
            topic_count[init_topic] += 1
            temp_word2topic.append(init_topic)
        word2topic.append(temp_word2topic)
```

3.2.2 困惑度计算

```
def compute_perplexity(): # 计算困惑度
    file_count = np.sum(file2topic, 1)
    # print(file_count)
    count = 0
    perplexity = 0
    for f_idx, segment in enumerate(segments):
        for word in segment:
            perplexity = (perplexity + np.log(((topic2word[:, word] / topic_count) *
                                               (file2topic[f_idx, :] / file_count[f_idx])).sum()))
            count += 1

    return np.exp(perplexity / (-count))
```

3.2.3 gibbs_sample

```
def gibbs_sample():
    global file2topic
    global topic2word
    global topic_count
    new_file2topic = np.zeros([file_num, topic_num])
    new_topic2word = np.zeros([topic_num, word_num])
    new_topic_count = np.zeros([topic_num])
    for f_idx, segment in enumerate(segments):
        for w_idx, word in enumerate(segment):
            old_topic = word2topic[f_idx][w_idx]
```

```
            p = np.divide(np.multiply(file2topic[f_idx, :], topic2word[:, word]), topic_count)
            new_topic = np.random.multinomial(1, p/p.sum()).argmax()
            word2topic[f_idx][w_idx] = new_topic
            new_file2topic[f_idx, new_topic] += 1
            new_topic2word[new_topic, word] += 1
            new_topic_count[new_topic] += 1
    file2topic = new_file2topic
    topic2word = new_topic2word
    topic_count = new_topic_count
```

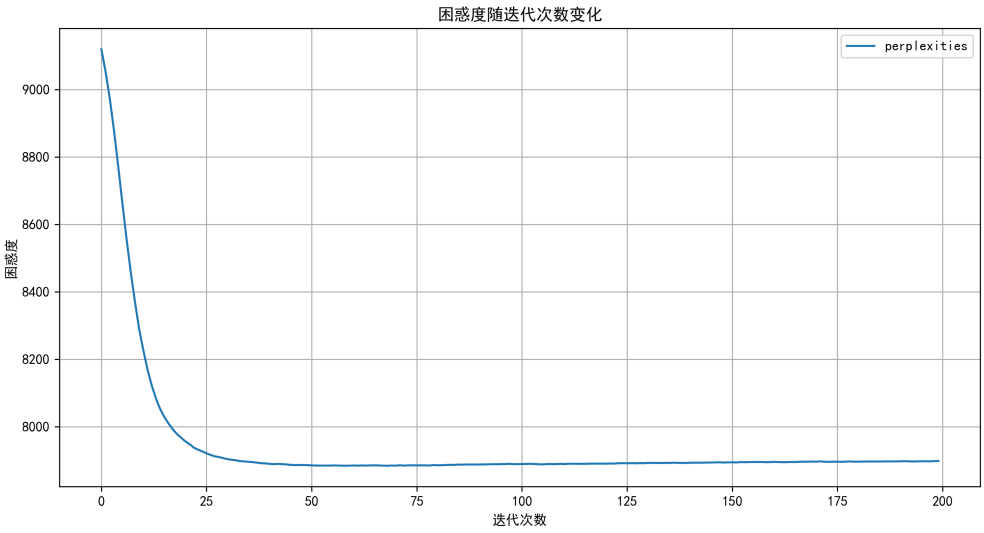
3.2.4 画图

```
def drawfig(x,perplexities,name): # 画图
    plt.rcParams["figure.figsize"] = (12, 6)
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.title('困惑度随迭代次数变化')
    plt.xlabel('迭代次数')
    plt.ylabel('困惑度')
    plt.grid()
    plt.plot(x, perplexities, label='perplexities')
    plt.legend()
    plt.savefig(name, dpi=300)
```

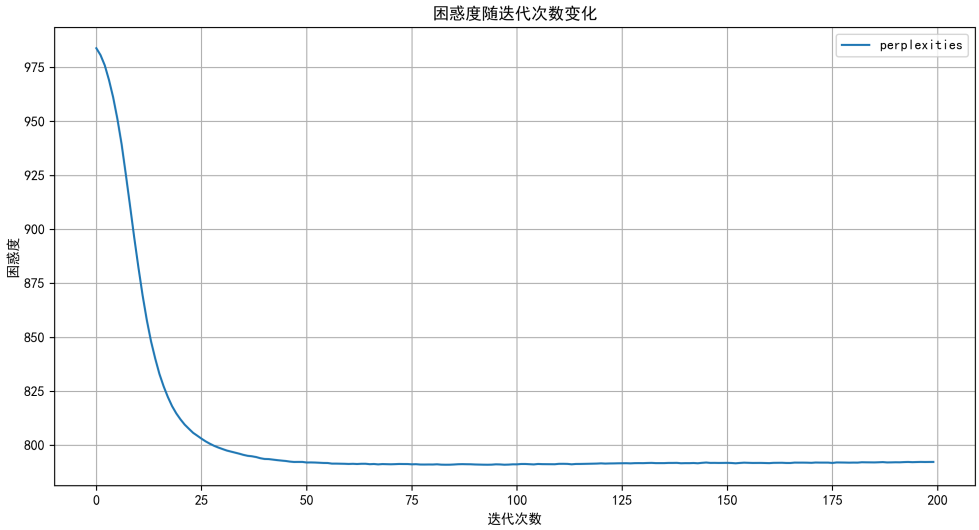
4.实验结果

4.1 16本小说

- 分词:



- 分字符



分词模式下，对每个主题下的词根据概率进行降序排列，部分结果展示如下（分字符不做展示）：

```
主题0的高频词为：
石破天:0.000719  道:0.000559  孩子:0.000551 不知:0.000496 没:0.000454  知道:0.000454 想:0.000395  说:0.000383  武林中:0.000328  这位:0.000320
--------------------------
主题1的高频词为：
心中:0.000723 爹爹:0.000694 见:0.000673  一个:0.000673 两人:0.000589 突然:0.000559 快:0.000475  两个:0.000454 伸手:0.000416 站:0.000416
--------------------------
主题2的高频词为：
中:0.001051  众人:0.000820 死:0.000454  住:0.000433  大叫:0.000425 下去:0.000366 有人:0.000324 出:0.000307  话:0.000294  周伯通:0.000265
--------------------------
主题3的高频词为：
著:0.000790  麽:0.000774  李文秀:0.000669  走:0.000664  老人:0.000589 汉子:0.000580 没:0.000509  说:0.000496  只见:0.000479 师父:0.000467
--------------------------
主题4的高频词为：
道:0.001333  袁承志:0.001030  青青:0.000467 知道:0.000408 问:0.000400  不能:0.000349 生死:0.000320 徐天宏:0.000311  话:0.000311  罢:0.000307
--------------------------
主题5的高频词为：
 :0.003031  道:0.001829  麽:0.000837  杨过:0.000711 甚:0.000669  郭靖:0.000664 便:0.000471  夫妇:0.000458 武功:0.000454 说:0.000416
```

可以看到，心中，爹爹，孩子，夫妇，武功，杨过，郭靖，师父等是高频词，效果还不错。

- 分词模型下SVM训练测试结果：

```
overall accuracy:0.700000
----------------------
accuracy for each class: [0.69230769 1.         1.         0.75       0.66666667 1.
 0.         0.58823529 0.73333333 0.77777778 0.75       0.64
 0.625      0.88888889 0.71428571 0.58823529]
----------------------
average accuracy:0.713421

overall accuracy:0.475000
```

```
----------------------
accuracy for each class: [0.5        0.         0.         1.
0.71428571 0.
 0.         0.25       0.         0.         0.         0.33333333
 0.4        0.5        1.         0.6       ]
----------------------
average accuracy:0.331101
```

- 分字符模型下SVM训练测试结果：

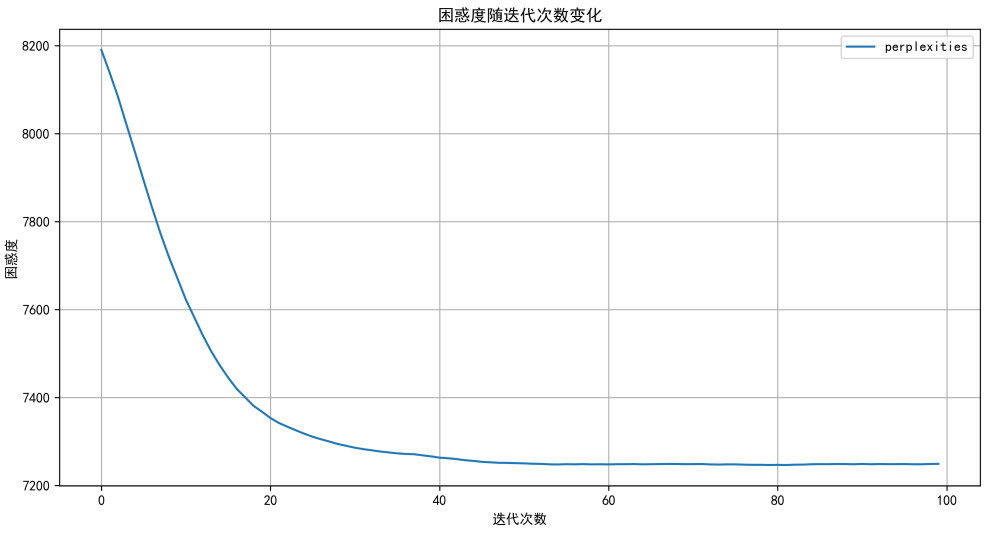
```
overall accuracy:0.768750
----------------------
accuracy for each class: [0.66666667 0.75       1.         1.
0.62068966 1.
 0.83333333 0.66666667 1.         1.         0.83333333 0.73333333
 1.         0.75       0.83333333 0.8       ]
----------------------
average accuracy:0.842960

overall accuracy:0.500000
----------------------
accuracy for each class: [1.         0.85714286 0.         0.         0.
    0.5
 1.         0.33333333 1.         0.         1.         0.
 0.25       0.        ]
----------------------
average accuracy:0.424320
```

16分类下SVM在测试集的表现非常差。

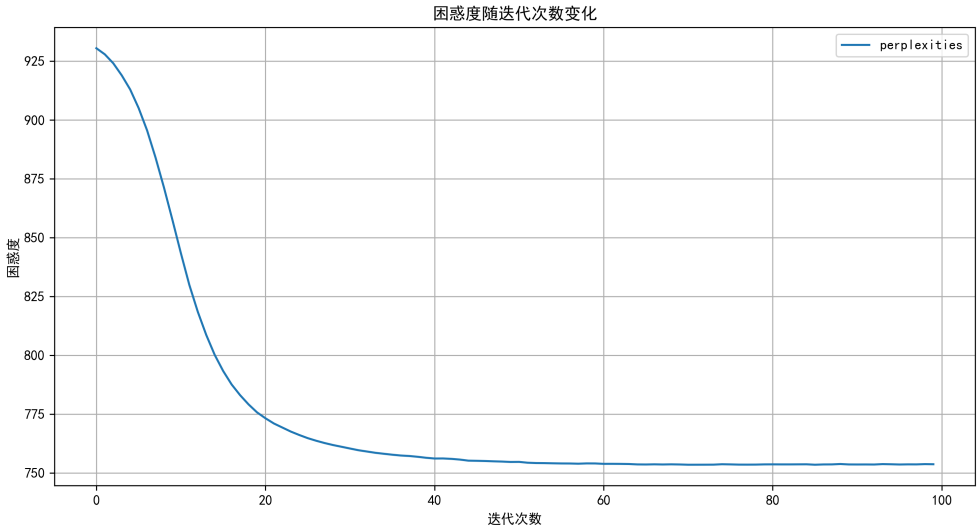
4.2 5本小说

降低分类难度，选择5本小说。

- 分词



- 分字符



分词模式下，对每个主题下的词根据概率进行降序排列，部分结果展示如下（分字符不做展示）：

```
主题0的高频词为：
萧峰:0.000606 一声:0.000450 伸手:0.000433 却是:0.000408 想:0.000374 起来:0.000362 怎地:0.000349 金花婆婆:0.000332 杀:0.000328 有人:0.000324
--------------------------
主题1的高频词为：
道:0.003566 说:0.000782 难道:0.000471 罢:0.000450 没:0.000433 二人:0.000416 不肯:0.000353 定:0.000324 两位:0.000320 一句:0.000320
--------------------------
主题2的高频词为：
道:0.003036 便:0.001846 说:0.001472 说道:0.000669 弟子:0.000555 这位:0.000391 和尚:0.000391 包不同:0.000391 想:0.000379 星宿:0.000379
--------------------------
主题3的高频词为：
郭靖:0.001985 黄蓉:0.001535 两人:0.000808 欧阳锋:0.000799 洪七公:0.000669 一个:0.000547 吃:0.000534 功夫:0.000526 黄蓉道:0.000463 转身:0.000379
--------------------------
主题4的高频词为：
中:0.000871 慕容复:0.000702 丐帮:0.000589 段誉:0.000475 帮主:0.000459 王语嫣:0.000429 心下:0.000404 段:0.000395 一个:0.000379 杨:0.000374
--------------------------
主题5的高频词为：
师父:0.001653 武功:0.000694 黄药师:0.000589 中:0.000505 心中:0.000505 梅超风:0.000429 弟子:0.000404 点:0.000370 少林寺:0.000341 欧阳克:0.000332
```

可以看出，许多我们耳熟能详的词作为高频词被选了出来，效果非常好。

- 分词模型下SVM训练测试结果：

```
overall accuracy:0.793750
----------------------
accuracy for each class: [0.81818182 0.83333333 0.7755102  0.70588235
0.80645161]
----------------------
average accuracy:0.787872

overall accuracy:0.625000
----------------------
accuracy for each class: [0.85714286 1.         0.36842105 1.         0.6
    ]
----------------------
average accuracy:0.765113
```

测试集的准确率达到0.765.

- 分字符模型下SVM训练测试结果：

```
overall accuracy:0.925000
----------------------
accuracy for each class: [0.91428571 0.96666667 0.90322581 0.96428571
0.88888889]
----------------------
average accuracy:0.927471

overall accuracy:0.925000
----------------------
accuracy for each class: [0.9        1.         0.91666667 1.         0.8
    ]
----------------------
average accuracy:0.923333
```

测试集的准确率达到0.923，效果比分词模型更佳.

5. 结论

LDA模型能够较好地解决一词多义和多词一意的问题，实验说明了LDA的有效性，并通过SVM进行了验证，针对金庸小说，有效的分类通常是一些有密切关系的人物，或者有关联的动作等。在分类类别非常多时，效果比较差，而在5分类时，效果很好，且分字符模式的效果优于分词模式。