深度学习与自然语言处理报告

19377354 渠海榕 Hairong Qu

qhrbh2001@163.com

摘要

本文是深度学习与自然语言处理第三次作业的报告，本次作业的要求是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，并利用 LDA 模型对文本进行建模，最终将每个段落表示为主题分布进行分类。本文通过实验验证了不同数量的主题个数对分类性能的影响，同时比较了以“词”和以“字”为基本单元下分类结果的差异。实验结果表明，主题个数对分类性能有一定的影响。当主题个数较少时，分类效果较差，而当主题个数过多时，分类效果也会下降。在实验中，当主题个数为 30 时，分类效果最佳。此外，以“词”和以“字”为基本单元进行分类的结果也有所不同。以“词”为基本单元的分类效果略优于以“字”为基本单元的分类效果。综上，本文通过实验验证了不同主题个数和基本单元对分类性能的影响，并得出了一些结论。这些结论对于后续文本分类的研究具有一定的参考意义。同时，本文也为利用 LDA 模型进行文本分类提供了一种可行的方法。

引言

本文主要介绍 LDA 算法和聚类算法在文本分类中的应用。LDA（Latent Dirichlet Allocation）是一种主题模型，它可以将文本数据中的每个文档表示为一组主题的概率分布，每个主题又表示为一组单词的概率分布。LDA 算法的基本思想是：将每个文档看作是若干个主题的混合，每个主题又看作是若干个单词的混合。LDA 算法的输入是文档集合，输出是每个文档的主题分布和每个主题的单词分布。

具体而言，LDA 算法首先随机地给每个单词分配一个主题，然后迭代地更新每个单词的主题和每个主题的单词，直到收敛为止。在每次迭代中，LDA 算法会计算每个单词属于每个主题的概率，以及每个文档属于每个主题的概率。这些概率可以用来表示每个文档的主题分布和每个主题的单词分布。

LDA 算法的优点是可以发现文本数据中的潜在主题，并且可以自动地将单词分配到主题中，从而减少了人工干预的成本。LDA 算法的应用非常广泛，包括文本分类、信息检索、推荐系统等领域。在文本分类中，聚类算法可以将相似的文本数据归为一类，从而实现文本的分类和分析。

在本文中，我们将利用 LDA 算法对文本数据进行主题建模，得到每个文本数据的主题分布情况，并将其作为特征向量进行聚类分析，从而实现文本数据的分类和分析。我们将通过比较不同主题个数的分类性能变化，以及以"词"和以"字"为基本单元下分类结果的差异来验证和分析分类结果。

原理

原理 1: LDA 模型

LDA（Latent Dirichlet Allocation）是一种主题模型，它可以将文档集合中的每篇文档表示成一组主题的分布，同时也将每个主题表示成一组单词的分布。LDA 的基本假设是，每篇文档都由多个主题组成，每个主题都由多个单词组成。LDA 通过对文档中的单词进行统计，来学习主题的分布和单词的分布。

具体地，LDA 模型假设一个文档d由多个主题z的混合组成，每个主题z又由多个单词w的分布组成。其中，主题的分布服从狄利克雷分布，单词的分布也服从狄利克雷分布。在 LDA 模型中，我们需要学习每个文档的主题分布θ_d，每个主题的单词分布ϕ_z，以及每个单词的主题z。

LDA 模型的生成过程如下：

1. 对于每个文档d，从主题分布θ_d中随机选择一个主题z；

2. 从主题z的单词分布ϕ_z中随机选择一个单词w；

3. 重复步骤 1 和步骤 2，生成文档d中的每个单词。

LDA 模型的参数学习可以使用 Gibbs 采样算法或变分推断算法等方法。通过学习得到的主题分布和单词分布，我们可以对文本进行主题建模和文本分类等任务。

LDA 模型的数学表达式如下：

假设有K个主题，V个单词，M篇文档，N_d表示文档d中的单词数，$w_{d,n}$表示文档d中

的第n个单词，$z_{d,n}$表示文档d中的第n个单词的主题，α和β分别表示主题分布和单词分布的超参数，则 LDA 模型的生成过程可以表示为：

$$\theta_d \sim \mathrm{Dir}(\alpha)$$
$$\phi_z \sim \mathrm{Dir}(\beta)$$
$$z_{d,n} \sim \mathrm{Multinomial}(\theta_d)$$
$$w_{d,n} \sim \mathrm{Multinomial}(\phi_{z_{d,n}})$$

其中，$\mathrm{Dir}(\cdot)$表示狄利克雷分布，$\mathrm{Multinomial}(\cdot)$表示多项式分布。

原理 2: Kmeans 聚类模型

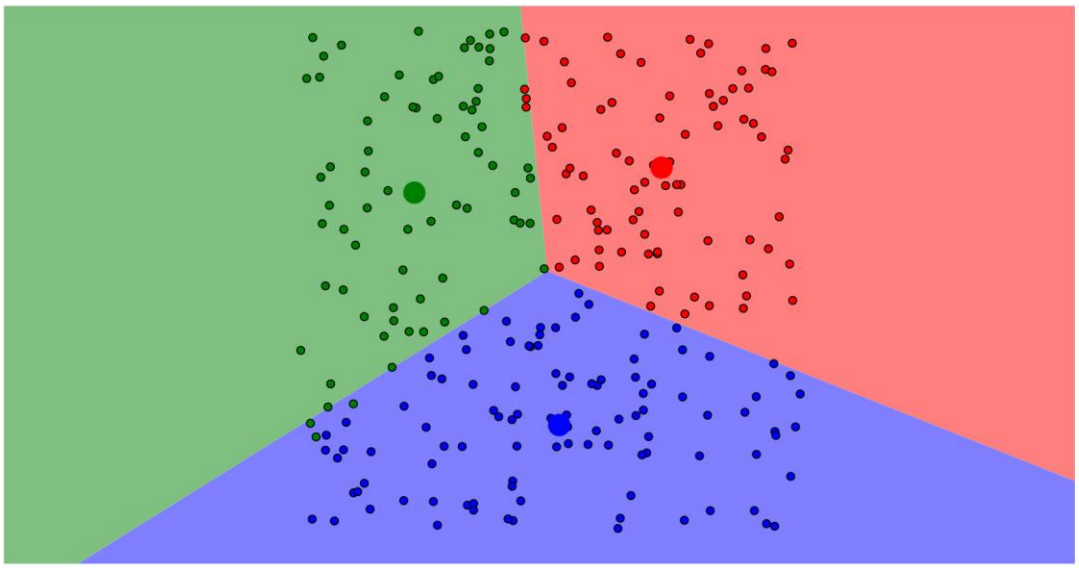

图 1 K-means 聚类模型

Kmeans 是一种基于距离的聚类算法，其主要思想是将数据集中的样本点分为 K 个簇，每个簇的中心点是簇中所有样本点的均值。Kmeans 算法的目标是最小化所有簇内样本点与簇中心点的距离之和。Kmeans 算法的流程如下：

1. 随机初始化 K 个簇的中心点
2. 对于每个样本点，计算其与 K 个簇中心点的距离，将其归为距离最近的簇
3. 对于每个簇，重新计算其中心点
4. 重复步骤 2 和步骤 3，直到簇中心点不再发生变化或达到最大迭代次数

Kmeans 算法的目标函数可以表示为：

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

其中，C_i表示第i个簇，μ_i表示C_i的中心点。

Kmeans 算法的优点是简单易懂、计算效率高。但是，Kmeans 算法需要预先指定簇的个数 K，且对初始中心点的选择敏感，容易收敛到局部最优解。

原理 3: Gibbs 采样求解 LDA

Gibbs 采样是一种基于马尔可夫链蒙特卡罗 (MCMC) 的方法，用于从后验分布中采样。LDA 中的 Gibbs 采样算法用于从联合分布 $p(z,w,\theta,\alpha,\beta)$ 中抽取样本。其中，z 是主题分布，w 是单词分布，θ 是文档主题分布，α 是主题分布的超参数，β 是单词分布的超参数。Gibbs 采样算法的核心思想是通过对联合分布中某个随机变量的条件分布进行采样，来逐步构建样本。

具体来说，Gibbs 采样算法需要进行以下步骤：

1. 初始化：对于每个文档中的每个单词，随机指定一个主题。

2. 对于每个单词，从联合分布中抽取一个新的主题。这里的抽样过程需要计算单词分布和主题分布的后验概率，并根据这些概率进行抽样。

3. 重复步骤 2，直到达到一定的迭代次数或收敛。

Gibbs 采样算法的优点在于可以处理高维数据，并且可以在不知道后验分布的具体形式的情况下进行抽样。LDA 中的 Gibbs 采样算法也是一种高效的方法，可以在大规模文本数据集上进行主题建模。

实验过程

过程 1: 段落生成

题目要求均匀地抽取每个段落大于 500 个词的 200 个段落， 每个段落的标签就是对应段落所属的小说。读取 inf.txt 文档可获取语料库内所有文章的标题，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，则一共能得到 208 个段落；对每篇文章分词之后，将一篇文章的总词数去掉停词后，接着把总词数除以 13，获取即每篇文章共有 13 个区间，所选取的段落为每个区间抽取前 500 个词。利用代码如下：

```
import os
import glob
import random
import jieba
from sklearn.cluster import KMeans
num_paragraphs = 200    # 抽取段落数
min_length = 500    # 每个段落最小长度
```

```
basepath = 'C:\\Users\\HUAWEI\\Desktop\\深度学习与自然语言处理\\第三次作业'  #文件夹位置
with open(basepath+'\\jyxstxtqj_downcc.com\\inf.txt','r',encoding='ANSI') as f:
    booklist = f.read().split(',') #读目录
with open(basepath+"\\cn_stopwords.txt","r",encoding = 'utf-8-sig') as f:
    stopwords = f.read().split() #读停用词
stopwords2 = [' ','wwwcrcom','www','cr173','com','□','说道','说','道','便','笑','=','txt','\n','本书来自 www.cr173.com 免费 txt 小说下载站','更多更新免费电子书请关注 www.cr173.com','\u3000','目录']
stopwords.extend(stopwords2)
content = []
for book in booklist:
    path = basepath+'\\jyxstxtqj_downcc.com\\'+ book +'.txt'
    with open(path, 'r', encoding='ANSI') as f:
        data_txt = f.read()  # 读取文本文件
    words = jieba.lcut(data_txt)  # 结巴分词
    words = [i for i in words if i not in stopwords]
    pos = int(len(words)//13)
    for i in range(13):
        data_temp = []
        data_temp = data_temp + words[i*pos:i*pos+500]
        content.append(data_temp)
```

过程 2: Gibbs 采样求解 LDA

代码如下:

```
import numpy as np
import random

doc_list=content
K=10
alpha=0.01
beta=0.1
num_iters=25
# 将列表中的元素转换为集合
word_dict = set(word for doc in doc_list for word in doc)
# 将列表中的元素转换为字典
word_dict = {word: i for i, word in enumerate(word_dict)}
# 将列表中的元素转换为 numpy 数组
doc_list = np.array([[word_dict[word] for word in doc] for doc in doc_list])
M = len(doc_list)  # 文档数
V = len(word_dict) # 词汇的大小
z = np.random.randint(0, K, size=(M, len(doc_list[0])))  # 主题分布
n_mz = np.zeros((M, K), dtype=int)  # 文档-主题分布
n_zv = np.zeros((K, V), dtype=int)  # 主题-词分布
```

```
    n_z = np.zeros(K, dtype=int)   # 主题数量
    for i in range(M):
        for j in range(len(doc_list[i])):
            k = z[i][j]
            n_mz[i][k] += 1
            n_zv[k][doc_list[i][j]] += 1
            n_z[k] += 1
    # Gibbs 采样
    for iter in range(num_iters):
        for i in range(M):
            for j in range(len(doc_list[i])):
                k = z[i][j]
                n_mz[i][k] -= 1
                n_zv[k][doc_list[i][j]] -= 1
                n_z[k] -= 1
                p = (n_mz[i] + alpha) * (n_zv[:, doc_list[i][j]] + beta) / (n_z + V * beta)
                # 将代码向量化，以提高计算速度
                k = np.random.choice(K, p=p / p.sum())
                z[i][j] = k
                n_mz[i][k] += 1
                n_zv[k][doc_list[i][j]] += 1
                n_z[k] += 1


    # 输出主题-词分布
    for k in range(K):
        print('Topic {}:'.format(k+1))
        top5_positions = np.argsort(n_zv[k])[::-1][:5]
        for v in top5_positions:
            print('{}:    {:.4f}'.format(list(word_dict.keys())[list(word_dict.values()).index(v)],
n_zv[k][v] / n_z[k]))
    print(n_mz)
```

设定分类主题为 10 时求解结果如下表所示:

表 1　设定主题为 10 类求解结果

Topic 1: 令狐冲: 0.0096 杨过: 0.0076 中: 0.0064 一个: 0.0057 仪琳: 0.0056	Topic 2: 麽: 0.0211 著: 0.0124 後: 0.0078 李文秀: 0.0072 中: 0.0071	Topic 3: 陈家洛: 0.0152 张召重: 0.0066 文泰来: 0.0051 徐天宏: 0.0049 众人: 0.0045	Topic 4: 张无忌: 0.0074 石破天: 0.0074 中: 0.0071 爷爷: 0.0060 听: 0.0060
Topic 5: 剑: 0.0113	Topic 6: 听: 0.0074	Topic 7: 韦小宝: 0.0224	Topic 8: 中: 0.0078

长剑: 0.0093 剑法: 0.0090 范蠡: 0.0075 一声: 0.0055	少女: 0.0071 一声: 0.0071 瞧: 0.0054 倒: 0.0051	剑士: 0.0081 青衣: 0.0073 皇帝: 0.0072 康熙: 0.0068	爹爹: 0.0066 见到: 0.0062 一个: 0.0062 事: 0.0056
Topic 9: 袁承志: 0.0135 中: 0.0095 曰: 0.0048 穆念慈: 0.0045 时: 0.0043	Topic 10: 见: 0.0112 听: 0.0096 中: 0.0083 一个: 0.0076 两人: 0.0067		

如果设定分类主题为 20，求解结果如下表所示:

表 2　topics 数量为 20 结果

Topic 1: 爹爹: 0.0143 青青: 0.0078 伯伯: 0.0065 法王: 0.0061 何红药: 0.0057	Topic 2: 令狐冲: 0.0294 仪琳: 0.0119 婆婆: 0.0098 田伯光: 0.0076 水蛭: 0.0073	Topic 3: 派: 0.0198 雪山: 0.0088 白万剑: 0.0071 弟子: 0.0054 林平之: 0.0054	Topic 4: 中: 0.0125 见: 0.0104 一声: 0.0088 袁承志: 0.0085 一个: 0.0082
Topic 5: 麽: 0.0424 著: 0.0218 後: 0.0158 李文秀: 0.0112 曹云奇: 0.0092	Topic 6: 李文秀: 0.0175 麽: 0.0169 苏普: 0.0123 苏鲁克: 0.0118 著: 0.0110	Topic 7: 张召重: 0.0137 举人: 0.0065 李沅芷: 0.0065 老师: 0.0059 欧阳克: 0.0052	Topic 8: 萧中慧: 0.0129 卓天雄: 0.0119 周威信: 0.0112 刀法: 0.0092 麽: 0.0090
Topic 9: 中: 0.0097 皇帝: 0.0060 李靖: 0.0051 天下: 0.0051 秦桧: 0.0048	Topic 10: 少林寺: 0.0080 大师: 0.0068 二人: 0.0068 蟾蜍: 0.0063 夫人: 0.0063	Topic 11: 听: 0.0134 中: 0.0121 见: 0.0074 想: 0.0071 之中: 0.0060	Topic 12: 曰: 0.0120 中国: 0.0098 阿青: 0.0076 中: 0.0069 杀: 0.0057
Topic 13: 陈家洛: 0.0243 文泰来: 0.0091 骆冰: 0.0065 群雄: 0.0062	Topic 14: 剑士: 0.0266 长剑: 0.0177 剑: 0.0157 青衣: 0.0125	Topic 15: 丁典: 0.0084 欧阳克: 0.0072 洪七公: 0.0072 狄云: 0.0066	Topic 16: 张无忌: 0.0171 中: 0.0056 欧阳锋: 0.0054 丐帮: 0.0051

红花: 0.0062	锦衫: 0.0086	黄眉僧: 0.0045	宋青书: 0.0045
Topic 17: 师父: 0.0091 中: 0.0083 杨过: 0.0076 听: 0.0065 没: 0.0063	Topic 18: 韦小宝: 0.0241 康熙: 0.0072 皇帝: 0.0065 一个: 0.0062 爷爷: 0.0058	Topic 19: 范蠡: 0.0361 少女: 0.0174 勾践: 0.0171 吴国: 0.0111 薛烛: 0.0108	Topic 20: 穆念慈: 0.0110 一个: 0.0101 老者: 0.0098 汉子: 0.0096 少女: 0.0087

如果设定分类主题为 5，求解结果如下表所示:

表 3 topics 数量为 5 结果

Topic 1: 听: 0.0104 中: 0.0064 一个: 0.0059 想: 0.0052 爹爹: 0.0049	Topic 2: 韦小宝: 0.0086 杨过: 0.0043 弟子: 0.0041 张无忌: 0.0040 杀: 0.0038	Topic 3: 麽: 0.0094 中: 0.0072 见: 0.0064 一个: 0.0056 走: 0.0053	Topic 4: 见: 0.0066 陈家洛: 0.0058 武功: 0.0054 听: 0.0032 中: 0.0030	Topic 5: 范蠡: 0.0085 中: 0.0073 剑士: 0.0071 袁承志: 0.0064 长剑: 0.0042

过程 3：对所求解的文本主题分布向量进行 Kmeans 聚类

代码如下:

```
from sklearn.cluster import KMeans

data = n_mz

# 聚类为 16 类

kmeans = KMeans(n_clusters=16)

kmeans.fit(data)

# 输出每个样本所属的类别

labels = kmeans.labels_

for i in range(len(labels)):

    print(labels[i],end = ' ')

    if (i+1)%13 == 0:

        print('')
```

对于 10 类 topics，得到的结果如下：

8 10 10 10 10 8 8 10 6 6 10 10 10

3 13 13 9 2 5 5 9 4 5 5 1 1

13 9 6 8 13 11 4 11 9 13 9 11 3

15 6 6 6 6 3 13 13 13 4 2 6 6

7 12 0 12 12 12 12 12 12 12 12 12 12

1 1 13 5 1 6 1 1 14 1 1 1 1

14 12 12 9 5 9 9 13 9 4 5 13 0

7 11 0 0 2 0 14 2 0 6 4 13 14

13 3 3 3 3 3 3 3 3 3 2 3 0

14 5 12 3 11 9 14 6 9 9 7 3 3

14 13 5 4 13 4 13 2 2 13 13 13 14

8 7 14 14 15 6 11 11 14 6 14 2 14

8 2 2 8 8 2 10 8 8 8 8 2 13

7 9 5 8 13 13 4 13 4 13 11 12 2

8 11 9 9 11 11 8 11 11 11 11 11 11

15 15 15 15 15 15 15 15 8 8 8 8 8

利用 Kmeans 将 208 个向量聚成 16 个类，结果中的每一行对应一个文本，它们理应是一个主题，从结果来看，第一个文本被归为第 10 类，之后每个文本分别被归为第 5，9，6，12，1，9，0，3，9，13，14，8，13，11，15，其中第 1 个，第 5 个，第 6 个，第 9 个，第

11 个，第 13 个，第 15 个和第 16 个文本的聚类结果更准确，可见它们的特征更加明显。

对于 20 类 topics，得到的结果如下:

11 5 5 5 5 12 12 5 5 5 5 5 5

6 11 11 6 7 11 7 2 14 6 6 9 9

11 10 6 6 10 0 2 0 11 11 11 6 2

3 3 3 6 6 13 8 8 6 6 3 3 3

9 7 7 7 7 7 7 7 15 9 7 7 11

9 9 0 9 9 9 9 9 0 9 9 2 9

11 5 5 7 11 11 1 1 0 7 12 1 5

4 1 3 3 3 3 3 3 14 3 3 3 6

0 2 2 0 2 2 2 2 0 0 11 0 3

11 15 11 15 6 6 6 5 3 2 6 15 14

11 13 13 7 13 6 6 14 14 13 6 6 14

11 15 15 15 3 15 13 1 13 15 15 3 15

11 12 11 12 12 3 12 12 12 12 12 10 11

11 11 13 2 1 1 1 1 1 6 1 11 1

10 10 10 10 10 10 10 10 10 10 10 10 10

8 8 8 4 4 4 4 4 4 4 4 8 8

对于 5 类 topics，得到的结果如下:

14 14 14 14 14 6 6 14 6 14 14 14 6

1 7 7 11 0 7 12 11 6 15 7 10 10

0 3 15 0 15 9 0 8 6 0 0 11 8

3 3 0 15 14 8 0 15 15 15 10 14 0

4 3 5 13 3 13 13 0 4 10 3 5 5

12 1 1 1 1 12 11 9 1 13 12 12 12

15 12 12 11 9 3 3 0 7 3 0 15 7

0 0 3 3 10 1 0 3 11 13 10 3 5

9 8 8 8 8 8 9 8 9 9 7 0 8

11 8 1 4 11 3 7 7 13 14 1 5 9

11 3 13 5 10 5 3 11 10 3 3 3 1

6 4 15 13 11 4 15 13 4 13 9 3 4

6 0 0 6 0 14 15 0 14 14 14 0 14

1 11 13 9 3 15 3 11 1 5 11 12 3

9 8 8 8 9 8 8 8 8 8 8 8 9

2 2 2 2 2 2 2 2 2 12 12 12 12

我们可以发现，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。分类主题数量为 20 时，得到的结果更加准确，分类主题数量为 5 时，得到的结果不准确，难以区分不同小说的差异。

过程 4：比较以“词”和以“字”为基本单元下分类结果的差异

表 4 以字作为单元分 10 个主题的结果表

Topic 1:	Topic 2:	Topic 3:	Topic 4:	Topic 5:
手: 0.0271	主: 0.0179	师: 0.0409	宝: 0.0161	马: 0.0270
中: 0.0237	胡: 0.0166	子: 0.0165	子: 0.0144	女: 0.0208
刀: 0.0193	家: 0.0146	令: 0.0143	韦: 0.0139	回: 0.0176
身: 0.0141	子: 0.0142	林: 0.0138	教: 0.0137	十: 0.0173
见: 0.0127	声: 0.0138	夫: 0.0135	天: 0.0137	中: 0.0142
Topic 6:	Topic 7:	Topic 8:	Topic 9:	Topic 10:
剑: 0.0800	子: 0.0274	时: 0.0185	中: 0.0151	李: 0.0178
士: 0.0254	声: 0.0192	天: 0.0149	十: 0.0138	文: 0.0174
国: 0.0176	汉: 0.0134	手: 0.0145	年: 0.0134	里: 0.0143
青: 0.0173	姑: 0.0110	石: 0.0144	袁: 0.0115	中: 0.0140
手: 0.0158	见: 0.0109	两: 0.0141	三: 0.0108	出: 0.0133

最终 kmeans 得到的结果如下:

12 4 0 0 0 0 0 0 4 0 0 0 0

11 14 4 14 8 3 3 15 14 14 14 15 3

4 13 8 1 1 1 1 1 8 1 4 6 9

2 2 4 15 13 1 13 13 13 14 2 6 15

12 7 7 7 7 7 7 7 7 15 7 15 15

12 3 7 3 3 15 15 11 9 11 11 11 11

12 15 15 4 13 14 14 14 13 13 4 4 15

12 14 6 8 13 6 6 13 3 6 13 13 13

4 9 9 1 3 9 9 9 12 15 2 13 9

12 6 4 4 13 6 6 6 6 13 6 11 4

12 7 6 9 6 6 6 6 6 6 6 6 3

12 8 8 8 2 13 1 8 8 8 8 2 7

12 3 1 1 3 13 9 1 3 14 14 1 1

12 14 3 3 6 9 6 7 6 7 1 1 1

5 5 5 5 8 13 5 5 13 14 5 5 5

10 10 10 10 10 2 2 10 10 2 2 10 2

我们可以发现，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。

结论

我们可以发现，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。

我们可以发现，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。