

自然语言处理 NLP 作业③

文本聚类 Text clustering

姓名：罗福杰

学号：3120305208

班级：S0078

目录

1 作业要求.....	3
2 聚类的基本概念.....	3
2.1 聚类算法的基本概念.....	3
2.2 常用的聚类算法和评价指标.....	3
2.2.1 K-means 聚类算法.....	3
2.2.2 Canopy 聚类算法.....	5
2.2.3 聚类的常用评价指标.....	6
2.3 文本聚类算法的流程.....	6
3 文本聚类的具体实现过程.....	7
3.1 读取文本信息：	8
3.2 文本信息预处理：	10
3.3 计算 TF-IDF：	12
3.4 K-means 聚类的实现：	15
3.5 聚类的结果和分析：	16
3.5.1 实验一的结果分析和改进.....	16
3.5.2 实验二的结果分析和改进.....	17
3.5.3 实验三的结果和分析.....	17
参考内容：	19

1 作业要求

- 1) This assignment uses the same data of “Text classification”.
- 2) The difference is that you should assume the class labels for all documents are unseen when you do clustering.
- 3) You should provide model evaluation results and discuss the reasons of the results.

2 聚类的基本概念

2.1 聚类算法的基本概念

聚类分析算法通常意义上来讲就是通过相关的计算方法来处理我们得到的需要被分类的数据集的过程。按照相对应的计算方法和判别准则进行相似性计算。得到相似性后，利用相关的方法和判别准则将目标数据集划分为指定的类别。

聚类分析想要达成的目标就利用我们计算得到的样本之间的相似度对数据进行分类。一般情况下我们经常使用的聚类分析方法的步骤主要分为：数据预处理，聚类方法的选择，聚类性能的评价，选取最优结果这几步。通过这几步的操作我们可以把具有不同相似度的数据分类到不同的簇中。而且聚类分析方法是一种探索性的分析方法，当我们在进行具体的操作时我们不必事先给出一个给定的具体分类标准，它是一种**无监督**的算法。

2.2 常用的聚类算法和评价指标

2.2.1 K-means 聚类算法

K-means 算法是聚类分析中被常用到的算法。它是采用样本之间的距离作为聚类指标进行相关的聚类分析活动。它是一种典型的**无监督**的聚类方法。因此不需要进行人工专家的标记，在很大程度上减少了聚类的成本。正是由于这些原因 K-means 方法的适用性比其它的聚类方法要好，而且它的改进潜力很大，可以在很多方面对其进行改进。

K 均值聚类算法是很常用的算法。下面是 K-means 方法的计算流程来对其进行简单描述。主要有如下几步：

- 1) 我们选取数据空间中的 K 个样本作为初始中心，其中每个对象代表一个聚类中心；
- 2) 对于样本集合中的数据，通过计算它们与这些聚类中心的距离来判定它们的归属。我们按距离最近的准则，把它们分到距离它们最近的聚类中心所对应的类；
- 3) 更新聚类中心，将每个类别中所有对象所对应的均值作为该类别的聚类中心，计算目标函数的值，然后继续放入样本计算它们的距离继续按照最近距离进行分类，并重新计算聚类中心；
- 4) 重复步骤 2 然后对每次迭代进行判断聚类中心和目标函数的值是否发生改变，如果聚类中心不再发生变化，或满足一定的迭代条件后，输出聚类的结果。

K-means 聚类算法的流程图如图 2.1 所示：

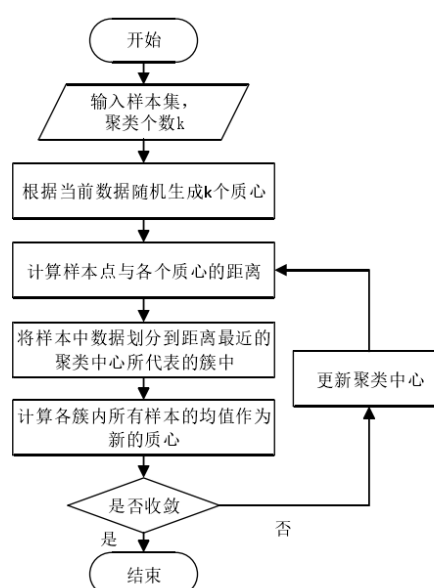


图 2.1 K-means 聚类算法的流程图

K-Means 算法主要分为数据划分以及聚类中心的更新这两个过程，聚类中心的变化小于预设阈值是 K-Means 算法结束的重要标志。数据划分的过程是通过相应的距离度量指标，计算数据对象与聚类中心的距离，并将其划分至距离最近的类别中。聚类中心更新的过程是通过计算各类别的均值来替换原来的聚类中心

点。

K-Means 算法是通过持续迭代聚类过程,使得到的相同类别内数据之间的距离较小,不同类别内数据之间距离较大。其作为一种较为常用的聚类算法,有原理简单、易于实现、具有伸缩性等优点,但是因为 K-Means 算法对初始点敏感,初始点的随机选取容易导致聚类耗时严重或聚类结果不满足需求。

2.2.2 Canopy 聚类算法

Canopy 算法是一种简单、快速的聚类算法,Canopy 算法示意图如图 2.2 所示。

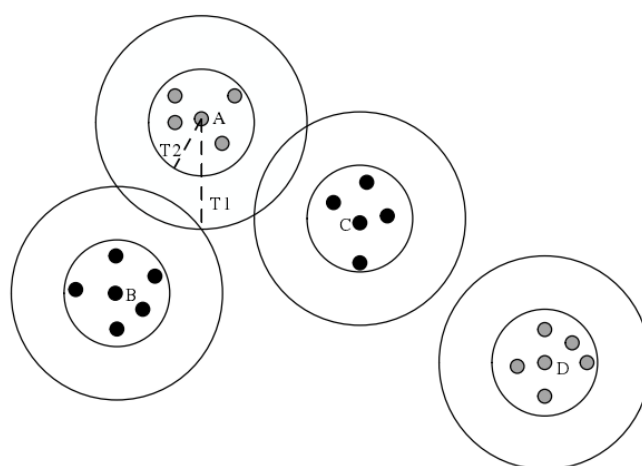


图 2.2 Canopy 算法示意图

基本思想是:采用合适的距离度量方式来计算目标数据集中的数据对象与 canopy 中心的距离,如果距离的值大于阈值,说明该数据对象与该 canopy 中的数据差异较大,将该数据作为新的 canopy 中心。若果距离值小于阈值,说明比较相似,将该数据加入该 canopy 中心所表示的 canopy 中。

Canopy 算法流程图如图 2.3 所示。

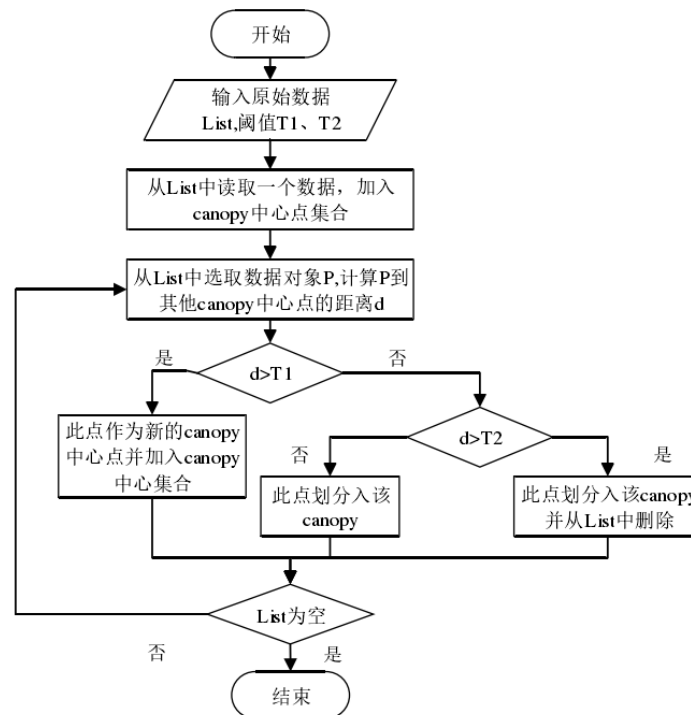


图 2.3 Canopy 算法流程图

2.2.3 聚类的常用评价指标

聚类算法的评价指标可以划分为**内部指标**和**外部指标**。

当数据集中没有已经分好的类别或没有人工事先对其分类, 那么关于聚类结果的评估是基于数据自身聚类结果的, 内部评价指标认为类内样本相似度越高, 不同类别的类内样本的相似度越低的聚类模型的聚类效果较好, 轮廓系数、DVI 系数、DB 系数等是较为常用的内部评价指标。

外部指标指当已知数据预分类别时, 可以利用这些可用外部信息来比较聚类结果和外部划分准则的匹配度。常用的外部评价指标为精确率、召回率、F-Measure 值。

2.3 文本聚类算法的流程

文本聚类的一般流程是首先对文本数据进行文本预处理, 然后选择合适的聚类算法对经过文本预处理后所得的结构化数据进行聚类。主要流程图如图 2.4 所示:

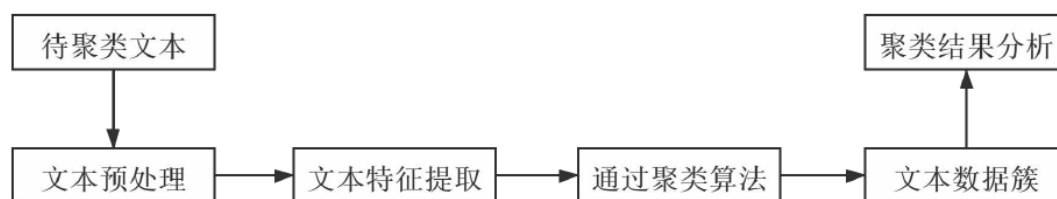


图 2.4 文本聚类流程图

首先需要收集得到初始文本数据集，这一步骤中的文本数据均为非结构化的初始文本数据集，其后的两个环节“文本预处理”和“文本特征提取”可以合称为“文本表示”阶段。初始文本数据集在经过“文本表示”操作后转化为适用于聚类算法的结构化数据。

文本预处理阶段的主要操作如下：

- 1) 分词：将输入的文本信息进行切分；
- 2) 去停用词：停用词是指不附带或附带很少信息的功能词，如“an”，“that”等词语，为了减小存储数据所需空间和提高运行效率的目的，常常需要建立一个停用词表，在“文本预处理”阶段将停用词去掉再进行“文本特征提取”操作。
- 3) 词性规范化：对同一个单词的不同形态表示进行归并操作。

文本特征提取阶段的主要操作是：将文本数据进行向量表示并提取主要的特征，常用的方法有：

- 1) 词频向量法：构建语料库，然后将每个文档转化为向量表示；
- 2) 特征频率-逆文档频率法（TF-IDF）：计算每个文档的 TF-IDF 矩阵，再进行特征提取；
- 3) 神经网络模型等方法。

3 文本聚类的具体实现过程

步骤：

1. 导入文本数据
2. 分词
3. 导入停用词表

4. 去除停用词，文本数据清洗
5. 将每个样本（每篇文章）转换为 TFIDF 向量
 - 5.1 导入 `tfidfvectorizer` 模块，
 - 5.2 实例化 `tfidfvectorizer` 模块，设置最大的特征数 `max_features`, 前 `n` 个 TFIDF 最大的词作为特征；
 - 5.3 训练 `tfidfvectorizer` 模块，使用 `.fit(words)` 函数，输入的数据是用空格分词后的数据；
 - 5.4 将文本数据转换成 TFIDF 向量
6. 构建聚类模型（
 - 6.1 导入聚类模块 `KMEANS`，
 - 6.2 实例化聚类模块 `KMEANS`, 设置 `K` 值
 - 6.3 训练聚类模型， 用 `.fit(特征数据)`
 - 6.4 查看聚类结果 `labels_`
 - 6.5 聚类结果分析

3.1 读取文本信息：

- 1) 给定所有文章所在的磁盘路径，本实验的文章路径是：

`root_path = 'C:/Users/admin/Desktop/mini_newsgroups'`

- 2) 编程实现读取每篇文章的路径，得到路径名和文章所属类别的两个列表 `list`;


```
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.mideast/77392
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.mideast/77813
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.mideast/77815
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176869
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176878
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176881
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176884
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176886
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176895
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176904
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176916
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176926
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176930
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176951
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176956
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176982
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176983
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176984
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176986
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/176988
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/177008
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178301
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178309
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178318
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178327
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178337
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178341
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178349
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178360
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178361
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178368
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178382
C:/Users/admin/Desktop/mini_newsgroups/talk.politics.misc/178389
```

3) 读取每篇文章的内容:

代码如下:

```
"""读一个文件的内容"""
def read_one_text_file(one_text_path):
    with open(one_text_path, 'r', encoding='utf-8', errors='ignore') as f:
        one_text_info = f.read()
    return one_text_info
```

4) 将所有内容赋值给字典 news,

其中关键字 key: dict_keys(['data', 'target_names', 'target']), 里面是每篇文章的具体内容 text 和所属的类别, 以及类别的编号;

获取所有内容的具体实现过程如下:

```
"""获取所有 news 的信息, 并放在一个字典 news 里面"""
def get_news_info(root_path):
    class_20_names, files_list = get_files_name_list(root_path)
    news = {"data": [], "target_names": [], "target": []}
    # 对于每个类别的每一个文件, 添加到 news 字典里
    for i in range(0, len(class_20_names)):
        for j in range(0, len(files_list[i])):
            news["data"].append(read_one_text_file(files_list[i][j]))
            news["target_names"].append(class_20_names[i])
            news["target"].append(i)
    return news
```

3.2 文本信息预处理:

- 1) 读信息：这一步读取了完整的信息，包含了文本的所有信息，如下面所示：

```
In [36]: runfile('E:/study/My_Code/Spyder/NLP_TextClass/textclasser04_2.py', wdir='E:/study/My_Code/Spyder/NLP_TextClass')
-----
Path: cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!fs7.ece.cmu.edu!europa.eng.gtefsd.com!howland.reston.ans.net!usc!
sdd.hp.com!sgiblab!sgigate!odin!fido!solntze.wpd.sgi.com!livesey
From: livesey@solntze.wpd.sgi.com (Jon Livesey)
Newsgroups: alt.atheism
Subject: Re: An Anecdote about Islam
Date: 5 Apr 1993 23:32:28 GMT
Organization: sgi
Lines: 15
Distribution: world
Message-ID: <1pqfic$9s2@fido.asd.sgi.com>
References: <1pd5nr$89n@sjl.gov> <113689@bu.edu> <168A4AB7F.I3150101@dbstu1.rz.tu-bs.de> <114127@bu.edu>
NNTP-Posting-Host: solntze.wpd.sgi.com

In article <114127@bu.edu>, jaeger@buphy.bu.edu (Gregg Jaeger) writes:
|>
|> I don't understand the point of this petty sarcasm. It is a basic
|> principle of Islam that if one is born muslim or one says "I testify
|> that there is no god but God and Mohammad is a prophet of God" that,
|> so long as one does not explicitly reject Islam by word then one _must_
|> be considered muslim by all muslims. So the phenomenon you're attempting
|> to make into a general rule or psychology is a direct odds with basic
|> Islamic principles. If you want to attack Islam you could do better than
|> than to argue against something that Islam explicitly contradicts.

Then Mr Mozumder is incorrect when he says that when committing
bad acts, people temporarily become atheists?

jon.
```

- 2) 去掉头信息：头信息包含了较多的无用信息，真正处理的应该是文本的正文，所以：将源文本进行切分，遇到第一个含两个分行的切分一次，去掉头信息；用以下代码：

```
text_list = text.split('\n\n',1)
text = text_list[-1]
```

得到的去掉头信息的文本结果是：

```
In article <114127@bu.edu>, jaeger@buphy.bu.edu (Gregg Jaeger) writes:
|>
|> I don't understand the point of this petty sarcasm. It is a basic
|> principle of Islam that if one is born muslim or one says "I testify
|> that there is no god but God and Mohammad is a prophet of God" that,
|> so long as one does not explicitly reject Islam by word then one _must_
|> be considered muslim by all muslims. So the phenomenon you're attempting
|> to make into a general rule or psychology is a direct odds with basic
|> Islamic principles. If you want to attack Islam you could do better than
|> than to argue against something that Islam explicitly contradicts.

Then Mr Mozumder is incorrect when he says that when committing
bad acts, people temporarily become atheists?

ion.
```

- 3) 英文分词：在这一步中去掉了文本中的多余的符号和信息，并且将文本处理成一个单词列表 words_list:

```

text = text.replace("\n\n","").replace("\n","").replace(">","").replace('.', '')
r='[!\"#$%&\'()*+_-./:;<=>?@[\\]^`{|}~]+'
text = re.sub(r,' ',text)
text = text.replace("  ",').replace(" ',')
text = text.lower()
words_list = text.split(' ')
print("-----")
print(words_list)
print("")

```

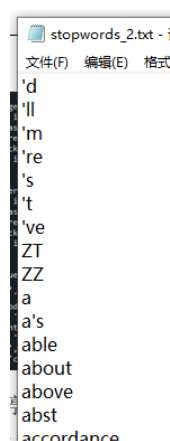
得到的结果是：

```

['in', 'article', '114127', 'buedu', 'jaeger', 'buphybuedu', 'gregg', 'jaeger', 'writes', 'i', 'don', 't', 'understand', 'the',
'point', 'of', 'this', 'petty', 'sarcasm', 'it', 'is', 'a', 'basic', 'principle', 'of', 'islam', 'that', 'if', 'one', 'is',
'born', 'muslim', 'or', 'one', 'says', 'i', 'testify', 'that', 'there', 'is', 'no', 'god', 'but', 'god', 'and', 'mohammad',
'is', 'a', 'prophet', 'of', 'god', 'that', 'so', 'long', 'as', 'one', 'does', 'not', 'explicitly', 'reject', 'islam', 'by',
'word', 'then', 'one', 'must', 'be', 'considered', 'muslim', 'by', 'all', 'muslims', 'so', 'the', 'phenomenon', 'you', 're',
'attempting', 'to', 'make', 'into', 'a', 'general', 'rule', 'or', 'psychology', 'is', 'a', 'direct', 'odds', 'with', 'basic',
'islamic', 'principles', 'if', 'you', 'want', 'to', 'attack', 'islam', 'you', 'could', 'do', 'better', 'than', 'than', 'to',
'argue', 'against', 'something', 'that', 'islam', 'explicitly', 'contradictsthen', 'mr', 'mozumder', 'is', 'incorrect', 'when',
'he', 'says', 'that', 'when', 'committingbad', 'acts', 'people', 'temporarily', 'become', 'atheists', 'jon']

```

- 4) 去除停用词: 构建停用词表, 本实验构造了一个停用词表 stopwords.txt 文件, 收录了 902 个停用词, 每个停用词占一行, 并且可以继续追加, 如下图所示。



```

stopwords_2.txt -
文件(F) 编辑(E) 格式
'd
'll
'm
're
's
't
've
ZT
ZZ
a
a's
able
about
above
abst
accordance

```

去掉停用词的代码：

```

'''去除停用词'''
def remove_stop_words(words_list, stop_words_path):
    stop_words_list = []
    words_clean_list = []
    f = open(stop_words_path, 'r')
    lines = f.readlines()
    for line in lines:
        stop_words_list.append(line.strip('\n'))
    for word in words_list:
        if word in stop_words_list:
            continue
        words_clean_list.append(word)
    return words_clean_list

```

去掉停用词后的结果对比：

```
[ 'in', 'article', 'buedu', 'jaeger', 'buphybuedu', 'gregg', 'jaeger', 'writes', 'i', 'don', 't', 'understand', 'the', 'point',
  'of', 'this', 'petty', 'sarcasm', 'it', 'is', 'a', 'basic', 'principle', 'of', 'islam', 'that', 'if', 'one', 'is', 'born',
  'muslim', 'or', 'one', 'says', 'i', 'testify', 'that', 'there', 'is', 'no', 'god', 'but', 'god', 'and', 'mohammad', 'is', 'a',
  'prophet', 'of', 'god', 'that', 'so', 'long', 'as', 'one', 'does', 'not', 'explicitly', 'reject', 'islam', 'by', 'word', 'then',
  'one', 'must', 'be', 'considered', 'muslim', 'by', 'all', 'muslims', 'so', 'the', 'phenomenon', 'you', 're', 'attempting', 'to',
  'make', 'into', 'a', 'general', 'rule', 'or', 'psychology', 'is', 'a', 'direct', 'odds', 'with', 'basic', 'islamic',
  'principles', 'if', 'you', 'want', 'to', 'attack', 'islam', 'you', 'could', 'do', 'better', 'than', 'than', 'to', 'argue',
  'against', 'something', 'that', 'islam', 'explicitly', 'contradictsthen', 'mr', 'mozumder', 'is', 'incorrect', 'when', 'he',
  'says', 'that', 'when', 'committingbad', 'acts', 'people', 'temporarily', 'become', 'atheists', 'jon']

[ 'article', 'buedu', 'jaeger', 'buphybuedu', 'gregg', 'jaeger', 'writes', 'understand', 'petty', 'sarcasm', 'basic',
  'principle', 'islam', 'born', 'muslim', 'testify', 'god', 'god', 'mohammad', 'prophet', 'god', 'explicitly', 'reject', 'islam',
  'word', 'considered', 'muslim', 'muslims', 'phenomenon', 'attempting', 'rule', 'psychology', 'direct', 'odds', 'basic',
  'islamic', 'principles', 'attack', 'islam', 'argue', 'islam', 'explicitly', 'contradictsthen', 'mozumder', 'incorrect',
  'committingbad', 'acts', 'people', 'temporarily', 'atheists', 'jon']
```

- 5) 得到所有文章预处理后的结果：以上是针对同一个文章 text 做的预处理，现在需要对所有的文章进行预处理、去除停用词等操作，所以通过 for 循环来编程实现

3.3 计算 TF-IDF:

文本预处理完成后，开始计算 TF-IDF 的值， $TF - IDF = TF \times IDF$ ，词频 TF 和逆文档频率 IDF 的具体定义如下：

$$\text{词频 } TF = \frac{\text{某个词在该文章出现的次数}}{\text{该文章的总词数}}$$

$$\text{逆文档频率 } IDF = \log \left(\frac{\text{语料库的文档总数}}{\text{包含该词的文档数} + 1} \right)$$

- 1) 构建语料库：里面包含 2000 篇已经处理完成的文章中包含的词语。具体的代码如下：

```
"""构建语料库，是一个list"""
def make_words_dataset(words_two_list):
    words_dataset_list = []
    for i in range(0, len(words_two_list)):
        print("一共有 2000 个文档，正在处理第: ", i+1, " 个文档")
        for word in words_two_list[i]:
            if word not in words_dataset_list:
                words_dataset_list.append(word)
    print("*****Done!*****")
    print("语料库中的单词总数是: ", len(words_dataset_list))
    print("语料库中前 100 个单词: \n", words_dataset_list[:100])
    return words_dataset_list
```

结果如下：（由此可见，语料库中共有 51135 个单词）

```

一共有 2000 个文档, 正在处理第: 1992 个文档
一共有 2000 个文档, 正在处理第: 1993 个文档
一共有 2000 个文档, 正在处理第: 1994 个文档
一共有 2000 个文档, 正在处理第: 1995 个文档
一共有 2000 个文档, 正在处理第: 1996 个文档
一共有 2000 个文档, 正在处理第: 1997 个文档
一共有 2000 个文档, 正在处理第: 1998 个文档
一共有 2000 个文档, 正在处理第: 1999 个文档
一共有 2000 个文档, 正在处理第: 2000 个文档
*****Done!*****
语料库中的单词总数是: 51135
语料库中前100个单词:
['apr', 'harderccr', 'pidaong', 'bob', 'mcgwier', 'writes', 'hate', 'economic', 'terrorism', 'political',
'correctness', 'worse', 'policy', 'effective', 'approach', 'donating', 'organizing', 'directly', 'indirectly',
'supports', 'gay', 'rights', 'issues', 'boycott', 'funding', 'scouts', 'reconcile', 'apparent', 'contradiction', 'rob',
'strom', 'watsonibmcom', 'ibm', 'mill', 'river', 'road', 'box', 'yorktown', 'heights', 'ny', 'kmr', 'pocwruedu',
'keith', 'ryan', 'people', 'questions', 'rarely', 'answer', 'themnope', 'answered', 'question', 'posed',
'multipletimeskeith', 'livesey', 'solntzewpdsgicom', 'jon', 'motto', 'stay', 'default', 'cutoff', 'exact', 'surely',
'notion', 'account', 'population', 'makeup', 'talking', 'arguingthat', 'interpreted', 'offensive', 'larger', 'portion',
'thepopulation', 'ago', 'bobbe', 'viceicotekcom', 'robert', 'beauchaine', 'capital', 'punishment', 'wrong', 'shooting',
'analogy', 'continue', 'drive', 'car', 'realizing', 'sooner', 'killed', 'automobile', 'accident', 'result', 'driving',
'killing', 'sufficient', 'evidence', 'conclude', 'kill', 'lifetimeyes', 'causeor']

```

- 2) 根据 TF-IDF 的计算公式，编程实现计算。将计算结果赋值给 news 的新键值对，便于后续聚类时使用。代码和示意图具体如下：

```

'''包含该词的文档数'''
def num_include_this_word(words_two_list, word):
    num_text_include_this_word = 0
    for i in range(0, len(words_two_list)): #对于每篇文章
        if word in words_two_list[i]:
            num_text_include_this_word += 1
    return num_text_include_this_word

'''计算 TF-IDF'''
def compute_tf_idf(words_two_list):
    all_text_tfidf_list = []
    for i in range(0, len(words_two_list)): #对每一篇文章
        print("*****")
        print("一共有 2000 个文档, 正在处理第: ", i+1, " 个文档")
        this_text_word_num = len(words_two_list[i])
        #将这篇文章里不重复的元素挑选出来
        this_text_word_no_repeat = list(set(words_two_list[i]))
        this_text_tfidf_numpy = np.zeros([3, len(this_text_word_no_repeat)])
        print(this_text_word_num, len(this_text_word_no_repeat))
        for j in range(0, len(this_text_word_no_repeat)): #对于每一个词
            word = this_text_word_no_repeat[j]
            #计算 TF
            print("当前的词: ", word)
            this_word_num = words_two_list[i].count(word)
            print("这个词在这篇文章中出现的次数: ", this_word_num)
            tf_this_word = this_word_num / this_text_word_num
            this_text_tfidf_numpy[1][j] = tf_this_word
            #计算 IDF
            num_text_include_this_word = num_include_this_word(words_two_list, word)
            print("这个词在所有文章中出现的次数: ", num_text_include_this_word)
            print("")
            idf_this_word = math.log(len(words_two_list), (num_text_include_this_word+1))
            this_text_tfidf_numpy[2][j] = idf_this_word
            this_text_tfidf_numpy[0][j] = tf_this_word * idf_this_word
        print(this_text_tfidf_numpy)
        all_text_tfidf_list.append(this_text_tfidf_numpy)
    return all_text_tfidf_list

```

当前的词: personal
这个词在这篇文章中出现的次数: 1
这个词在所有文章中出现的次数: 62

当前的词: surprisingly
这个词在这篇文章中出现的次数: 1
这个词在所有文章中出现的次数: 7

当前的词: killed
这个词在这篇文章中出现的次数: 2
这个词在所有文章中出现的次数: 60

当前的词: risk
这个词在这篇文章中出现的次数: 2
这个词在所有文章中出现的次数: 19

当前的词: writes
这个词在这篇文章中出现的次数: 1
这个词在所有文章中出现的次数: 1041

当前的词: solely
这个词在这篇文章中出现的次数: 1
这个词在所有文章中出现的次数: 13

当前的词: tired
这个词在这篇文章中出现的次数: 1

```
[[1.56801510e-02 3.12415507e-02 3.16064080e-02 4.33716850e-02
 9.34896232e-03 2.46167363e-02 2.46167363e-02 2.01825053e-02
 3.33853947e-02 1.66926973e-02 3.12415507e-02 4.03650105e-02
 2.53279770e-02 2.69548479e-02 5.67547062e-02 2.98229608e-02
 9.37246520e-02 1.67816081e-02 9.37246520e-02 5.91336716e-02
 1.87449304e-02 5.91336716e-02 2.70925003e-02 4.68623260e-02
 4.03650105e-02 4.68623260e-02 9.37246520e-02 4.68623260e-02
 1.37166926e-02 3.62576447e-02 7.88448955e-02 1.87449304e-02
 4.14384297e-02 9.37246520e-02 3.62576447e-02 1.69681544e-02
 1.87449304e-02 2.10171744e-02 1.30357356e-02 9.37246520e-02
 9.37246520e-02 1.91006199e-02 1.78593648e-02 5.30560255e-02
 2.34311630e-02 2.76956559e-02 3.33853947e-02 7.49797216e-02
 1.44732103e-02 9.37246520e-02 9.37246520e-02 1.56801510e-02
 1.57409409e-02 3.00938771e-02 3.57187296e-02 9.37246520e-02
 2.61438305e-02 9.37246520e-02 9.37246520e-02 3.62576447e-02
 1.37681453e-02 9.37246520e-02 2.24763400e-02 3.12415507e-02
 2.82139316e-02 1.71674743e-02 5.91336716e-02 2.82139316e-02
 2.12677510e-02 9.37246520e-02 2.46167363e-02 2.53279770e-02
 9.37246520e-02 2.29297869e-02 2.46406159e-02 9.37246520e-02
 9.37246520e-02 2.95668358e-02 1.59324051e-02 2.70925003e-02
 1.84226706e-02 1.84226706e-02 5.91336716e-02 2.94036305e-02
 1.60683044e-02 2.53279770e-02 5.91336716e-02 4.03650105e-02
 2.70925003e-02 4.68623260e-02 9.37246520e-02 4.68623260e-02
 1.63627712e-02 4.03650105e-02]
[8.54700855e-03 8.54700855e-03 1.70940171e-02 1.70940171e-02
 8.54700855e-03 8.54700855e-03 8.54700855e-03 8.54700855e-03
 8.54700855e-03 8.54700855e-03 8.54700855e-03 8.54700855e-03
 8.54700855e-03 1.70940171e-02 2.56410256e-02 1.70940171e-02
 8.54700855e-03 8.54700855e-03 8.54700855e-03 8.54700855e-03
 8.54700855e-03 8.54700855e-03 8.54700855e-03 8.54700855e-03]
```

一共有 2000 个文档, 正在计算第: 1 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 2 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 3 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 4 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 5 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 6 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 7 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 8 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 9 个文档的 TF-IDF
一共有 2000 个文档, 正在计算第: 10 个文档的 TF-IDF

3) 在每篇文章中, 选取 TF-IDF 值最大的十位作为该文章的特征值, 核心代码和结果如下:

```
#将 TF-IDF 为前十的元素取出来,作为特征
idx = np.argsort(this_text_tfidf_numpy[0], -10)[-10:]
this_text_tfidf_numpy_10 = this_text_tfidf_numpy[0][idx]
for p in idx:
    this_text_word_no_repeat_10.append(this_text_word_no_repeat[p])
all_text_tfidf_list_10.append(this_text_tfidf_numpy_10)
all_text_no_repeat_list_10.append(this_text_word_no_repeat_10)
```

一共有 2000 个文档, 正在计算第: 9 个文档的 TF-IDF

这篇文章所有的tf-idf:
[0.03394676 0.0547552 0.15040521 0.04907113 0.10266753 0.05199314
0.12884232 0.05611833 0.08491503 0.05611833 0.04725773 0.03636942
0.23838661 0.30081042 0.13299253 0.10121014 0.12884232 0.22867233
0.30081042 0.12545696 0.05716808 0.09152078 0.04725773 0.30081042
0.09222053 0.15040521 0.05199314 0.04576039 0.30081042]
这篇文章排名前十的tf-idf值:
[0.12884232 0.13299253 0.30081042 0.22867233 0.30081042 0.30081042
0.15040521 0.23838661 0.15040521 0.30081042]

一共有 2000 个文档, 正在计算第: 10 个文档的 TF-IDF

这篇文章所有的tf-idf:
[0.06412739 0.02209421 0.0828545 0.09619109 0.03639878 0.05366365
0.07442359 0.07442359 0.03750664 0.0828545 0.14884717 0.07442359
0.03781496 0.06412739 0.0828545 0.04924176 0.0828545 0.0320637
0.03781496 0.09451772 0.05791281 0.0352385 0.0828545]
这篇文章排名前十的tf-idf值:
[0.07442359 0.07442359 0.0828545 0.0828545 0.0828545 0.0828545
0.98451772 0.09619109 0.0828545 0.14884717]

一共有 2000 个文档, 正在计算第: 11 个文档的 TF-IDF

3.4 K-means 聚类的实现:

K-means 聚类的思想：先随机选择 k 个聚类中心，把集合里的元素与最近的聚类中心聚为一类，得到一次聚类，再把每一个类的均值作为新的聚类中心重新聚类，迭代 n 次得到最终结果。

- 1) 初始化聚类中心：由上面的步骤可以得到所有文章的 TF-IDF 向量，首先随机选取一个点作为聚类点，然后计算离这个点最远的一个点为第二个初始点，再选取离这两个都最远的点作为第三个聚类点，以此类推，得到 20 个要分类的聚类初始点。核心代码和结果如下：

```
def distance(e1,e2):
    return np.linalg.norm(e1 - e2)
# arr 中距离 a 最远的元素，用于初始化聚类中心
def farthest(k_arr, arr):
    f = np.zeros([1,10]); max_d = 0
    for e in arr: d = 0
        for i in range(k_arr.__len__()):
            d = d + distance(k_arr[i],e)
            if d > max_d: max_d = d; f = e
        return f
def cluster_k_means(k,x_list):
    point_num = len(x_list)
    arr = np.array(x_list) #初始化聚类中心和聚类容器
    r = np.random.randint(point_num)
    k_arr = np.array([arr[r]]) # 随机选一个当作聚类中心
    cla_arr = [[]]
    for i in range(k-1):
        d = farthest(k_arr,arr)
        k_arr = np.concatenate([k_arr,np.array([d])])
        cla_arr.append([])
    print("初始化的聚类中心: ")
    for e in k_arr:
        print(e)
```

```
初始化的聚类中心:
[0.27674558 0.27674558 0.27674558 0.27674558 0.43863137 0.43863137
0.43863137 0.43863137 0.43863137 0.43863137]
[0.04122475 0.07802977 0.04679843 0.04122475 0.05201985 0.04122475
0.04122475 0.04122475 0.04122475 0.04122475]
[0.15615508 0.27414461 0.27414461 0.36552614 0.54828921 0.54828921
0.54828921 1.09657843 0.34593198 0.54828921]
[0.07442359 0.07442359 0.0828545 0.0828545 0.0828545 0.0828545
0.98451772 0.09619109 0.0828545 0.14884717]
[0.26745815 0.26745815 0.26745815 0.26745815 0.26745815 0.26745815
0.34556387 0.33749461 0.53491631 1.06983261]
[0.16614825 0.16614825 0.16614825 0.16614825 0.16614825 0.16614825
0.52413936 0.16614825 0.99688948 0.25709966]
[0.04122475 0.07802977 0.04679843 0.04122475 0.05201985 0.04122475
0.04122475 0.04122475 0.04122475 0.04122475]
[0.15615508 0.27414461 0.27414461 0.36552614 0.54828921 0.54828921
0.54828921 1.09657843 0.34593198 0.54828921]
[0.07442359 0.07442359 0.0828545 0.0828545 0.0828545 0.0828545
0.98451772 0.09619109 0.0828545 0.14884717]
[0.26745815 0.26745815 0.26745815 0.26745815 0.26745815 0.26745815
0.34556387 0.33749461 0.53491631 1.06983261]
[0.15615508 0.27414461 0.27414461 0.36552614 0.54828921 0.54828921
```

- 2) 循环迭代：编写循环迭代的主程序，每一轮循环过程中，都要计算当前的特征点离哪个聚类中心最近，然后重新聚成新的类，接着更新聚类中心点，直到满足迭代条件为止。其核心代码和运行结果如下：

```
cla_temp = cla_arr #迭代聚类
```

```

for i in range(n): # 迭代 n 次
    print("一共有 ",n," 次迭代，现在正进行第 ",i+1," 轮迭代...")
    for e in arr: # 把集合里每一个元素聚到最近的类
        ki = 0 # 假定距离第一个中心最近
        min_d = distance(e, k_arr[ki])
        for j in range(1,k_arr.__len__()):
            if distance(e, k_arr[j]) < min_d:
                min_d = distance(e, k_arr[j]);ki = j
        cla_temp[ki].append(e)
    for k in range(k_arr.__len__()): # 迭代更新聚类中心
        if n-1 == i:
            break
        k_arr[k] = means(cla_temp[k])
        cla_temp[k] = []

```

```

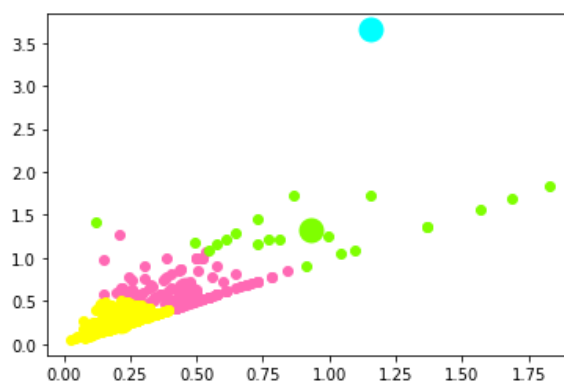
一共有 10 次迭代，现在正进行第 1 轮迭代...
一共有 10 次迭代，现在正进行第 2 轮迭代...
一共有 10 次迭代，现在正进行第 3 轮迭代...
一共有 10 次迭代，现在正进行第 4 轮迭代...
一共有 10 次迭代，现在正进行第 5 轮迭代...
一共有 10 次迭代，现在正进行第 6 轮迭代...
一共有 10 次迭代，现在正进行第 7 轮迭代...
一共有 10 次迭代，现在正进行第 8 轮迭代...
一共有 10 次迭代，现在正进行第 9 轮迭代...
一共有 10 次迭代，现在正进行第 10 轮迭代...

```

3.5 聚类的结果和分析：

3.5.1 实验一的结果分析和改进

如果将每篇文章的维度压缩为 2 维，处理的数据是四个类别的 400 篇文章，进行 4 分类，将运行结果可视化出来，如下图所示：



由上图可知，聚类出来的效果很差，因为模型没法对异常点进行处理，只能默认该异常点单独成为一类文章，所以采取这种方式有待改进：主要是将文章提取为两个点会损失大量的信息，因此应该增加维度，然后再进行实验。

3.5.2 实验二的结果分析和改进

由实验一结果,应该增加每篇文章的维度再进行分析 and 实验,取每篇文章 TF-IDF 的最大的前十个组成一个新的十维特征向量,然后再进行分析和比较,得到的前三类的聚类结果如下:

[illegible]

由实验二的分类结果不均衡（第二类的文章很少）可知，应该是异常点导致的这种情况，原因与改进：

- 1) 对比 TF-IDF 的计算公式，应该是特征值的词频或逆文档频率较高造成的，即可能是停用词去除不完整导致的，或者词性规则化不完整导致的；
- 2) 另外一方面，该文本数据集中，有些文本内容很少（去除停用词后，仅 2-3 个词，然后该文章的特征向量填 0 补充），这样会影响模型的效果；
- 3) 在特征提取的过程中，可以尝试选择使用词频向量法进行表示和实验。

3.5.3 实验三的结果和分析

以上存在的问题都是继续需要改进的地方，下面使用库函数的方法进行实验和分析，调用 `sklearn` 中的文本特征提取的算法 `TfidfVectorizer` 和聚类函数算法 `KMeans()`，对文本数据进行聚类。核心代码如下：

```
tfidf_stop_vec = TfidfVectorizer(analyzer='word', stop_words='english')
x_tfidf_stop_train = tfidf_stop_vec.fit_transform(x_text)
km_model = KMeans(n_clusters=20)    # 实例化构造聚类器
km_model.fit(x_tfidf_stop_train)    # 聚类，传入所有特征数据
label_pred = km_model.labels_      # 获取聚类的标签
print("-----")
print(label_pred)
print("len:",len(label_pred))
print("前一百个样本聚类的结果:",label_pred[:100])
```

实验过程中打印了前一千个文本数据通过聚类得到的结果：

```
前1000样本聚类的结果: [11 2 10 3 10 9 9 9 3 9 5 6 18 0 6 9 1
3 9 17 0 5 18 7 11 9 10 5 9 19 6 9 9 9 13 12 3 6 15 9 12
10 9 14 17 17 5 9 6 9 13 9 15 15 9 6 3 9 9 11 10 9 13 9 9
3 9 6 18 12 12 11 5 9 19 8 3 1 9 9 3 9 11 3 5 14 9 0 10
9 15 0 9 5 6 16 9 13 9 9 10 6 12 9 9 15 9 14 3 19 15 13 9
9 3 9 0 3 9 3 6 9 11 3 18 9 9 6 15 9 5 9 1 19 11 1 15
1 12 15 13 9 1 9 9 3 9 0 9 9 6 0 11 9 9 9 9 15 10 19 9
19 9 1 15 9 3 6 9 15 13 7 9 19 3 1 3 14 7 8 9 9 9 6 3
9 12 12 9 12 9 10 9 9 15 9 14 19 9 9 9 8 6 3 9 9 19 9 12
3 13 6 4 9 9 12 6 18 6 9 3 3 5 9 9 3 17 19 12 9 9 9 3
3 6 12 10 15 13 9 19 9 15 9 15 9 16 1 3 9 9 9 11 0 15 17 9
9 9 10 9 6 18 5 5 6 6 0 9 9 9 9 9 15 9 9 1 8 3 15 1
15 9 6 9 6 19 9 6 18 15 4 9 12 9 9 17 9 9 13 9 9 14 15 13
5 13 9 9 12 6 9 9 9 4 9 9 9 1 8 4 2 15 15 9 3 2 0 9
9 11 18 12 12 9 11 9 9 9 0 18 15 9 9 9 18 6 0 3 14 11 12 11
5 0 3 14 3 9 11 15 9 18 9 3 9 11 10 9 0 15 15 9 3 4 19 9
9 12 18 13 9 19 5 9 17 13 9 9 0 9 15 9 18 6 9 18 9 13 3 9
9 15 7 18 13 11 9 9 6 9 6 9 9 9 9 9 15 15 9 9 9 4 3 14
4 15 9 9 3 17 3 9 17 9 15 9 15 3 9 9 15 4 6 9 9 9 9 3
0 9 15 8 1 6 9 18 11 9 6 19 9 1 5 12 13 9 1 15 0 9 13 15
8 9 9 9 9 9 8 3 9 15 6 3 9 9 7 11 3 9 12 11 9 11 18 9
1 5 19 9 11 9 9 3 9 7 9 9 12 11 18 15 6 9 3 12 9 9 9 6
9 9 13 17 11 18 1 3 6 9 19 9 9 9 9 9 10 18 3 3 15 2 3 19
9 9 9 6 9 5 9 9 9 1 9 9 11 12 1 6 9 10 13 7 3 19 6 9
9 9 9 9 0 9 11 2 6 9 11 10 15 18 9 17 6 19 3 9 9 9 0 9
3 11 12 9 2 4 1 0 9 9 11 9 18 13 19 13 12 9 13 9 6 14 9 9
15 0 9 6 19 12 15 9 18 13 18 9 3 9 3 13 9 11 9 19 1 9 16 15
15 13 11 10 11 5 9 9 0 9 15 8 9 5 9 5 9 4 9 9 5 9 19 19
6 10 0 3 9 4 0 9 6 3 19 5 18 9 18 1 14 9 17 9 1 9 9 9
9 3 17 6 9 3 9 6 1 13 12 13 9 9 9 9 9 9 8 9 3 3 17 9
```

聚类的指标通过下面的参数表征：通常有

- 1) 调整兰德系数：Adjusted Rand Index: 在[-1, 1]之间，值越大意味着聚类结果与真实情况越吻合。从广义的角度来将，ARI 是衡量两个数据分布的吻合程度的；
- 2) 互信息：用来衡量两个数据分布的吻合程度，在[-1, 1]之间，值越大意味着聚类结果与真实情况越吻合；
- 3) 同质性（homogeneity）：每个类里只包含单个类的成员；
- 4) 完整性（completeness）：给定类的所有成员都分配给同一个群集。

完整性具体的结果是：

```
评价指标：兰德指数
0.12152341897909116

评价指标：互信息
0.4242748985048062

评价指标：同质性homogeneity, 每个群集只包含单个类的成员。
0.3819219308598172

评价指标：完整性completeness: 给定类的所有成员都分配给同一个群集
0.4838785181793694
```

由结果可知，聚类得到的效果一般，与上一次的作业相比，效果较差。这一方面是无监督聚类算法的缺陷，另一方面，可能是文本数据的预处理过程中的不足。无论如何，这些都是未来探索和提高的方向。

参考内容：

- [1] 陈威. 基于 K-means 算法的文本文档主题聚类分析[D].兰州大学, 2020.
- [2] 邹鋆. 基于聚类算法的文本挖掘研究[D]. 电子科技大学, 2020.
- [3] 林红静. 基于 K-means 的微博短文本聚类算法研究[D].海南大学, 2016.
- [4] https://blog.csdn.net/qg_37509235/article/details/82925781
- [5] <https://www.jb51.net/article/187770.htm>
- [6] <https://blog.csdn.net/ustbbsy/article/details/80960652>