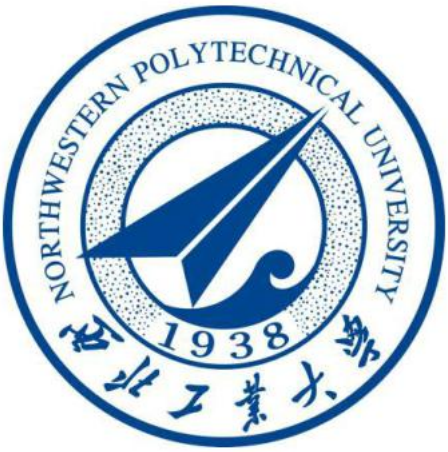
信 息 内 容 安 全 实 验 报 告



实验名称：	话题检测及分析
班级：	SC011701
姓名：	廖凯华 谢希 郑博文 裴嘉琨
学号：	2017302219 2017302220 2017302240 2017302242
指导教师：	杨黎斌
实验时间：	2020.4.9

摘要

随着当今社会中的社交媒体的快速增长，大量的信息呈现着爆炸式增长的趋势。社交媒体的便捷性、传播性、原创性使得大众的使用率急速上涨。但是同时随着大量的信息在网络上出现，对其中的信息内容进行提取整合是一件很重要的事情。因此我们主要选用 **K-Means 算法**和 **KNN 算法**实现针对文本的话题检测。在本份报告中我们会从相关已有文献入手，学习探究相关话题检测系统的原理和流程。并自主尝试使用 K-Means 算法和 KNN 算法分别对已有的文本数据库进行训练和检测。通过二维图像直观呈现了 K-Means 算法分簇结果和 KNN 模型构建过程。通过准确率的比较可以看出 KNN 算法是一种更好的选择。后期针对微博也进行了爬取数据的相关尝试。我们认为需要将多种的方法方式融合，取长补短，才可以得到一个较稳定的话题检测方法。

关键词：话题检测　K-Means 算法　KNN 算法

Abstract

With the rapid growth of social media in today's society, a large amount of information presents the trend of explosive growth. The convenience, dissemination and originality of social media make the usage rate of the public increase rapidly. But at the same time, as a large amount of information appears on the network, it is very significant to extract and integrate the information content. Therefore, we mainly use **K-MEANS** algorithm and **KNN** algorithm to realize **topic detection** for text. In this report, we will start with relevant existing literature to learn and explore the principle and process of related topic detection system, and try to use **K-MEANS** algorithm and **KNN** algorithm to train and detect the existing text database. The clustering results of **K-MEANS** algorithm and the process of **KNN** model construction are presented intuitively through two-dimensional images. The comparison of accuracy shows that **KNN** algorithm is a better choice. In the later stage, relevant attempts were made to crawl and fetch data for microblog. We believe that a stable topic detection method can be obtained only by integrating various methods and learning from each other.

Key Words: Topic detection, K-MEANS algorithm, KNN algorithm

目录

工作分配：

学号	姓名	工作分工
2017302219	廖凯华	K-Means、KNN 代码实现
2017302220	谢希	爬虫代码和相关文档
2017302240	郑博文	K-Means 实现和文档
2017302242	裴嘉琨	文档

1. 引言

随着近些年来各种众多的社交媒体的迅速发展，大量的新闻等不同的信息在社交媒体上进行着交互和出现。美国的 Twitter、中国的微博均是广大网民会选择使用的一种社交媒体网站，他们可以随时随地记录生活见闻、表达个人观点、关注亲友状态、了解最新时事等[1]。同时，因为这些社交媒体的便捷性、传播性、原创性使得大众的使用率急速上涨。其中微博的实时性，可以让大众可以快速及时的了解到全球各地发生的重大事件，从而可以看到不同的观点，并进行评论予以讨论。

但是同时随着大量的信息在网络上出现，对其中的信息内容进行提取整合是一件很重要的事情。不仅可以过滤无用信息，还可以提高对重要信息提取的效率。对海量的信息进行相关的检测和分类是一个很重要的事情。

本次的实验我们主要针对阅读相关文献，讨论关键新闻挖掘方法。通过 ***K-Means*** 算法和 ***KNN*** 算法分别实现一个可以对新闻报道进行分类的系统，并进行准确率比较。

2. 中文微博话题检测系统[2]

针对中国微博数据，实质上是一些列的独立的每条不超过 140 字的短文本。在文本中存在着一些特定的方式表示公共主体和用户间的互动关系。例如使用“@用户”的格式来表示提到某个用户，用“#主题#”表示参与某个特定主题的讨论。同时文本中还有会包含一些例如发送时间、来源、地理信息、发送者等附加属性。微博数据增长速度极快，话题分布也相对很分散。该系统主要是为了可以筛选出与现实中发生的新闻事件相关的话题。

话题检测主要包括了数据获取、预处理、分词和词频统计、主题词检测和话题聚类 5 步。

2.1 预处理

在获得到相关的微博新闻文本时，需要在被分词处理前进行预处理，减少噪声，屏蔽无关数据，提高检测的效率。主要有 3 部分处理规则：

(1) 忽略人数小于阈值 F 的用户。目的是为了取消掉僵尸用户或广告用户所发的信息，他们的信息法躲是信息少且噪声大。
(2) 忽略含有“@用户”格式的信息。该类信息大多是含有指定性的话题，更大的可能是交互式的对话形式。是新闻事件的可能性较少，所以忽略掉该类情况。
(3) 删除含有“#话题#”部分，该部分人为因素过大，会影响后面的词频的统计算法。

---

[1] 百度百科：https://baike.baidu.com/item/%E5%BE%AE%E5%8D%9A/79614?fr=aladdin

[2] 郑斐然, 苗夺谦, 张志飞, et al. 一种中文微博新闻话题检测的方法[J]. 计算机科学, 2012(01):144-147.

2.2 分词和词频统计

主要使用 ***ICTCLAS***[3]分词系统，该分词系统是中国科学院计算技术研究所在多年研究工作积累的基础上，所研制出的。其主要功能包括中文分词；词性标注；命名实体识别；新词识别；同时支持用户词典。***ICTCLAS*** 是当前世界上最好的汉语词法分析器。

通过该方法我们可以将预处理后的文本进行分词，不仅会有相关分词结果的词性，还有得到一个词向量。不同的词性的贡献程度是不同的，主要起作用的是名词和动词，所以只对这两种词性的词语进行下一步的操作。在进行统计词频时，我们需要设定一个时间段，对设定好的固定时间段内出现的文本进行分词和词频统计。在经过大量的实验会发现，分词的结果主要呈现长尾效应，我们主要选取前面高频的词语进行主题词检测，长尾部分将被删除。

2.3 主题词检测

由于主题词存在动态更新的特征，需要考虑两方面因素：1）检测词本身频率的变化。当发生相关事件时，相应词频会迅速升高。2）在时间窗内与最高词频数对比，标志着该词在热度的排行。

引入增长系数G_{ij}表示词语 i 在时间窗格（时间段）j 重点增长速度，计算公式为：

$$G_{ij} = \frac{F_{ij}}{\bar{F}_i} = \frac{F_{ij} \cdot K}{\sum_u^K F_{iu}} \quad (1)$$

F_{iu}是指词汇 i 在 u 时间窗内的出现频率，K 是回顾窗的大小。G_{ij}的值越大，则说明该词越有可能是突然出现的热议词。

为了可以更加合理选出最后的关键词，结合词频和增长速率，构造一个复合的权值进行衡量主题词的程度：

$$\omega_{ij} = log\, G_{ij+\alpha}\, log \frac{F_{ij}}{F_{max}} \quad (2)$$

其中F_{max}是指在时间窗内的最高词频。最后权值ω越大，则该词汇越有可能为关键词。α在公式中的主要作用是调节词频和增长速率之间的比重关系。在多次的实验后，认为其值在 1.0 到 1.5 之间结果最好。

通过该分类的方式，可以将前面时间段出现次数少但是在该时间段出现次数多词汇检测出，这些特征词汇作为新闻的关键主题词的可能性会较大。

2.4 主题词聚类

将得到的词依照距离进行聚类，主要的工作是寻找了一种能准确反映相关度的距离计算——上下文式的相似度，即凭借以往短句数据，计算新词与已有簇中任意词的条件概率，取最小值进行判断，很好解决了短词句中相似度获得的问题。

[3] 百度百科：https://baike.baidu.com/item/ICTCLAS/8609504?fr=aladdin

从而得到相关话题关键词的聚类，从而可以根据每一种聚类来表述相关的新闻话题。

通过对整个中文微博话题检测的过程分析，结合近些年改进后微博的特性，我们认为主要的难点和侧重点在以下的三个方面：

（1）微博平台反爬机制逐渐完善，很难获取原始数据。

（2）在主题词检测步骤中，需要划分时间窗，不同间隔对结果影响较大。

（3）在主题词聚类部分，计算词间条件概率需要多次遍历操作，需要适当处理。

3. 新闻话题挖掘流程

经过对《一种中文微博新闻话题检测的方法》论文的学习和我们小组成员的讨论，我们认为针对一个新闻话题的挖掘主要的流程如下：

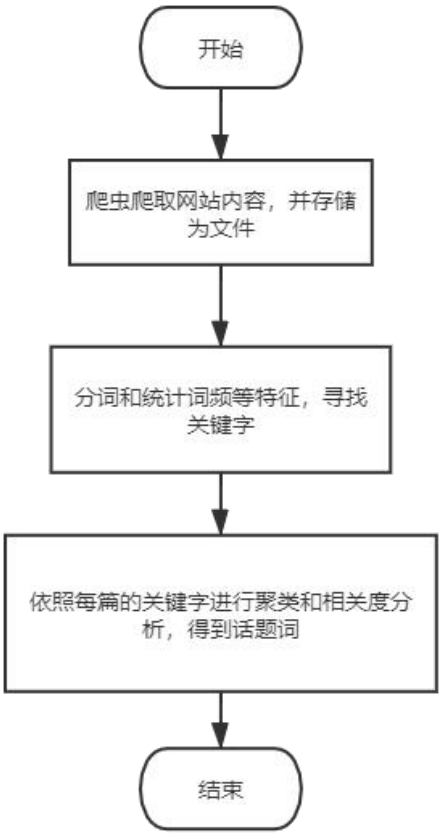


图 1. 新闻话题挖掘流程图

为了可以实现对关键词的新闻关键词提取，可以首先对需要收集的社交平台进行爬虫，爬取所有的网站内容，在针对相关的特性进行预处理从而储存为对应的文件。之后对已储存好的信息进行分词并统计词频特征工作从取得文本内容中的关键词。之后对得到的关键词进行聚类操作和相关度分析，从而得到聚类后的一簇一簇的关键词集合，之后进行整合得到话题词。

4. K-Means 实现话题分类

4.1 主要算法

4.1.1 TF-IDF 算法

TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF 加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级[4]。在该实验中调取相关库实现该算法。

4.1.2 K-Means 算法

K-means 算法采用迭代更新的思想,该算法的目标是根据输入的参数 k（k 表示需要将数据对象聚成几簇）,其基本思想为:首先指定需要划分的簇的个数 k 值,随机地选择 k 个初始数据对象作为初始聚类或簇的中心；然后计算其余的各个数据对象到这 k 个初始聚类中心的距离,并把数据对象划分到距离它最近的那个中心所在的簇类中；然后重新计算每个簇的中心作为下一次迭代的聚类中心.不断重复这个过程,直到各聚类中心不再变化时或者迭代达到规定的最大迭代次数时终止.迭代使得选取的聚类中心越来越接近真实的簇中心,所以聚类效果越来越好,最后把所有对象划分为 k 个簇[5]。

4.2 调用函数库

•（1）***sklearn***：开源的基于 python 语言的机器学习工具包。它通过 NumPy，SciPy 和 Matplotlib 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。
TfidfTransformer：用于统计 vectorizer 中每个词语的 TF-IDF 值。
CountVectorizer：将文本中的词语转换为词频矩阵。

•（2）***pandas***：是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的[6]。

•（3）***jieba***：基于 Trie 树结构实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图（DAG)；采用了动态规划查找最大概率路径, 找

[4] 百度百科：https://baike.baidu.com/item/tf-idf/8816134?fr=aladdin
[5] CSDN：https://blog.csdn.net/baidu_33566882/article/details/79886598
[6] 百度百科：https://baike.baidu.com/item/pandas

出基于词频的最大切分组合； 对于未登录词，采用了基于汉字成词能力的 HMM 模型[7]。

•（4）***Matplotlib***：是一个 Python 的 2D 绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形[8]。

4.3 分类预实现

我们在第一次分类预实现中先对每篇文章的主题词进行聚类，简单查看下结果，因此我们将其分为四个函数，各自的功能如下：
（1）trainfenci()：进行训练集分词与预处理；
（2）train()：进行 K-Means 聚类与模型建立；
（3）testfenci()：进行测试集分词与预处理；
（4）test()：利用训练过的模型进行测试。

4.3.1 训练集分词

```
def trainfenci():
    trainpath = '..\\话题检测\\训练集\\'
    resultpath = '..\\话题检测\\train.txt'
```

将训练集文本同一放于一个文件夹中，并在其目录下建立 6 类文件夹，存放各自的类别，将最终输出得到的训练集分词结果存于 train.txt 文件中。

```
i = 1
    while i<=22:                #针对 C4-Literature 类的文件进行一次读取
        with open(resultpath,'a') as rf:
            path = trainpath + 'C4-Literature\\' + 'Literature ('+str(i)+').txt'
            with open(path, 'r') as f:
                data = f.read()
            for keyword, weight in extract_tags(data,topK = 10, withWeight = True):
                print('%s %s' % (keyword, weight))
            #进行分词与计算每篇文本中的 tfidf 值
                rf.write(keyword+' ')
            rf.write('\n')
        i = i+1
    with open (resultpath,'a') as rf:
        rf.write('\n')
```

以上代码以 C4-Literature 为例，依次读取训练集中该类的每一个文本进行分词和计算 tfidf 操作，将选出的关键词放入训练集分词结果文本中。其余训练集中的各类文本同理可得。

[7] CSDN：https://blog.csdn.net/qq_37098526/article/details/88877798
[8] 百度百科：https://baike.baidu.com/item/Matplotlib/20436231?fr=aladdin

4.3.2 聚类与模型分析

```
def train():
    trainpath = '..\\话题检测\\train.txt'
    # file 文件类型的对象
    with open(trainpath,'r') as file:
        # 以列表的形式输出文本
        trainlines = list(file)
        #print(lines)
    train_list = trainlines
    #print(text_list)
    #需要进行聚类的文本集
    train_tfidf_matrix = tfidf_vectorizer.fit_transform(train_list)
    num_clusters = 6
    km_cluster = KMeans(n_clusters=num_clusters, max_iter=999999, n_init=1,
init='k-means++',n_jobs=1)
    trainresult = km_cluster.fit_predict(train_tfidf_matrix)
    print ("Train result: ", trainresult)
    print ('\n')
    print ("cluster_center: ")
    print (km_cluster.cluster_centers_)
    print ('\n')
    i = 1
    while i <= len(km_cluster.labels_):
        if i<10:
            print (i,'  ',km_cluster.labels_[i - 1])
        else:
            print (i,'  ',km_cluster.labels_[i - 1])
        i = i + 1
    print ('\n')
    print ("inertia:")
    print (km_cluster.inertia_)
    print ('\n')
    joblib.dump(km_cluster,'km.pkl')      #保存模型
```

num_clusters: 指定 K 的值
max_iter: 对于单次初始值计算的最大迭代次数
n_init: 重新选择初始值的次数
init: 制定初始值选择的算法
n_jobs: 进程个数，为-1 的时候是指默认跑满 CPU

4.3.3 测试集分词

我们对测试集的分词过程和训练集的过程类似，采取如下措施进行分词：

```
def testfenci():
    trainpath = '..\\话题检测\\测试集\\'
    resultpath = '..\\话题检测\\test.txt'
    i = 1
    while i<=10:
        with open(resultpath,'a') as rf:
            path = trainpath + 'C4-Literature\\' + 'Literature ('+str(i)+').txt'
            with open(path, 'r') as f:
                data = f.read()
            for keyword, weight in extract_tags(data,topK = 5, withWeight = True):
                print('%s %s' % (keyword, weight))
                rf.write(keyword+' ')
            rf.write('\n')
        i = i+1
    with open (resultpath,'a') as rf:
        rf.write('\n')
```

4.3.4 模型测试

我们加载在训练集中得到的模型，并用该模型对测试集进行评估，代码如下，具体操作和建立模型时的操作类似：

```
def test():
    testpath = '..\\话题检测\\test.txt' #测试集路径
    with open (testpath,'r') as file :
        testlines = list(file)
    test_list = testlines
    test_tfidf_matrix = tfidf_vectorizer.fit_transform(test_list)
    print(test_tfidf_matrix)
    km_cluster = joblib.load('km.pkl')      #加载模型
    testresult = km_cluster.fit_predict(test_tfidf_matrix)
    print ("Test result: ",testresult)
    print ("cluster_center: ")
    print (km_cluster.cluster_centers_)
    print ('\n')
```

（接下页）

（接上页）

```
    i = 1
    while i <= len(km_cluster.labels_):
        if i<10:
            print (i,'                ',km_cluster.labels_[i - 1])
        else:
            print (i,'               ',km_cluster.labels_[i - 1])
        i = i + 1
    print ('\n')
    print ("inertia:")
    print (km_cluster.inertia_)
    print ('\n')
```

4.3.5 预实现结果分析

训练集的分类如下图所示：

```
Train result:  [4 4 4 0 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 4 0 3 4 4 4 4 1 4 1 4 4 4 4 4 1 5
 4 4 4 4 1 1 5 4 4 4 4 1 4 1 4 4 4 1 4 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 3 3 3 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 5 3 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 2 3 3 3 3 3 2 3 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 4 3 3 2 2 3 3 3 3 3
 2 3 3 3 3 3 3 3 4 3 4 3 3 1 3 2 2 2 3 1 3 3 2 2 3 3 2 3 2 3 2 2 2 3 2 3 3
 3 3 3 3 3 3 0 4 3 4 2 3 3 3 3 3]
```

图 2. 训练集分类结果

从上图中我们简单地就可以看出，这次的预实现建立的模型并不理想，无法很明确地分析分出六类文章。同时，我们得到利用该模型的测试结果，很明显也不理想：

```
Test result:  [2 2 2 2 2 2 3 2 2 2 5 1 0 2 2 2 2 1 2 2 2 5 2 2 2 2 2 2 2 4 4 4 5 3 2 2 2
 3 2 2 3 3 2 5 2 2 2 2 2 2 2 2 2 2 5 2 2 2 2 2 2 1 2 2 2 5]
```

图 3. 测试集分类结果

通过第一次的预实现，我们发现对文章的主题（由 tfidf 值高的词构成的）进行分类并不能得到很好的效果，因此我们在正式的实现处理中，将每个文本的所有内容拿出来进行分类。

4.4 话题检测实现

4.4.1 提取文件

调取在训练集中已有文件进行训练，在获取相关文本形成列表时，如果当前某类的文件数过大，我们只选取其中的 50 个进行学习。

```
def getFiles():
    baseDir = './/database'
    firstDirList = [os.path.join(baseDir, f)  for f in os.listdir(baseDir)]  #获得一级目录列表
    filePath = []
    labelList = []  #原始标签列表，用于测试结果对比
    for file in firstDirList:  #处理二级目录，获得具体文件路径
        label = file.split('-')[0][13:]
        secondDirList = [os.path.join(file, f) for f in os.listdir(file)]
        # print(len(secondDirList))
        if len(secondDirList) > 100:
            secondDirList = secondDirList[:50]  #如果数量过多则取 50 个
        tmp = [label] * len(secondDirList)
        labelList.extend(tmp)
        filePath.extend(secondDirList)
    randnum = os.urandom(8)
    random.seed(randnum)
    random.shuffle(labelList) #以同样方式对路径和标签打乱
    random.seed(randnum)
    random.shuffle(filePath)
    return filePath, labelList
```

4.4.2 文本分词

在提取完相关的文本后，使用 jieba 库中的分词功能，对文本进行分词。为了提高后期文本检测的准确率，在分词时主要只保留中文字节。建立相关的停用词文档 stopwords.txt，并且字节若小于两个字节不给予考虑，这样可以减少无关词汇或符号对最终结果的影响。针对停用词文档中，主要包含：标点符号、数字、特殊符号、连词等词汇。

```
def segDepart(sentence):            # 对文档中的每一行进行中文分词
    print("正在分词")
    newSentence = re.sub(r'[^\u4e00-\u9fa5]', ' ', sentence)  #只保留中文
    sentenceDepart = jieba.cut(newSentence.strip())         # 创建一个停用词列表
    stopwords = [line.strip()
        for line in open('stopwords.txt', encoding='UTF-8').readlines()]
    outstr = " "                                            # 去停用词，拼接词为一行字符串
    for word in sentenceDepart:
        if word not in stopwords:
            if word != '\t':
                if len(word) >= 2:
                    outstr += word
                    outstr += " "
    return outstr
```

为了更好的后期调用和储存分词结果，专门创建新的文本 out.txt 进行储存。每一个文本的分词结果均单独保存于独立的一行上。

```
def saveInOne(path):
outputs = open("out.txt", 'w', encoding='UTF-8')
for filename in path:
inputs = open(filename, 'r', encoding='gb18030', errors='ignore')
for line in inputs:
lineSeg = segDepart(line)
outputs.write(lineSeg)
outputs.write('\n') #一篇文章的分词内容占一行
inputs.close()
outputs.close()
```

4.4.3 训练文本

该部分函数主要目的是根据刚才已经将词汇分好后的情况进行 TF-IDF 算法的计算，将训练文本分词后进行训练。将训练后的相关权重等数值形成的模型保存，等待 testfunc 的调用。相关较详细的代码注释如下：

```
def getModel():
    corpus = []
    #读取预料  一行预料为一个文档
    for line in open('out.txt', 'r', encoding='UTF-8').readlines():
        corpus.append(line.strip())

    #将文本中的词语转换为词频矩阵  矩阵元素 a[i][j]  表示 j 词在 i 类文本下的词频
vectorizer = CountVectorizer(min_df=10)

    #该类会统计每个词语的 tf-idf 权值
transformer = TfidfTransformer()

    #第一个 fit_transform 是计算 tf-idf  第二个 fit_transform 是将文本转为词频矩阵
tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))

    #获取词袋模型中的所有词语
word = vectorizer.get_feature_names()

    #将 tf-idf 矩阵抽取出来，元素 w[i][j]表示 j 词在 i 类文本中的 tf-idf 权重
weight = tfidf.toarray()
#打印特征向量文本内容
    resName = "Tfidf_Result.txt"
    result = codecs.open(resName, 'w', 'utf-8')    #不易出现编码问题
```

（接下页）

（接上页）

```
        for j in range(len(word)):
            result.write(word[j] + ' ')
        result.write('\n')

        #每类文本的 tf-idf 词语权重，第一个 for 遍历所有文本，第二个 for 便利某一类文本下的词语权重
        for i in range(len(weight)):
            for j in range(len(word)):
                result.write(str(weight[i][j]) + ' ')
            result.write('\n')
        result.close()
```

将每一个已知分类的文本进行 k-means 算法，进行相关文本的分簇，并且调用 Matplotlib 库画出分簇结果。

```
clf = KMeans(n_clusters=6) #已知可分为 6 类
        s = clf.fit(weight)
        joblib.dump(clf, 'km.pkl')
        #每个样本所属的簇
        label = []
        for i in range(len(clf.labels_)):
            label.append(clf.labels_[i])
        print(label)
        y_pred = clf.labels_
        pca = PCA(n_components=2)   #输出两维
        newData = pca.fit_transform(weight)   #载入 N 维
        xs, ys = newData[:, 0], newData[:, 1]
        #设置颜色（略）
        #设置类名（略）
        df = pd.DataFrame(dict(x=xs, y=ys, label=y_pred, title=corpus))
        groups = df.groupby('label')
        fig, ax = plt.subplots(figsize=(8, 5))   # set size
        ax.margins(0.02)
        for name, group in groups:
            ax.plot(
                group.x,
                group.y,
                marker='o',
                linestyle='',
                ms=10,
                label=cluster_names[name],
                color=cluster_colors[name],
```

可以看出通过 tf-idf 方法后在经过 k-means 算法得到的分类情况为：（不同的分类类别使用不同的颜色进行表述）

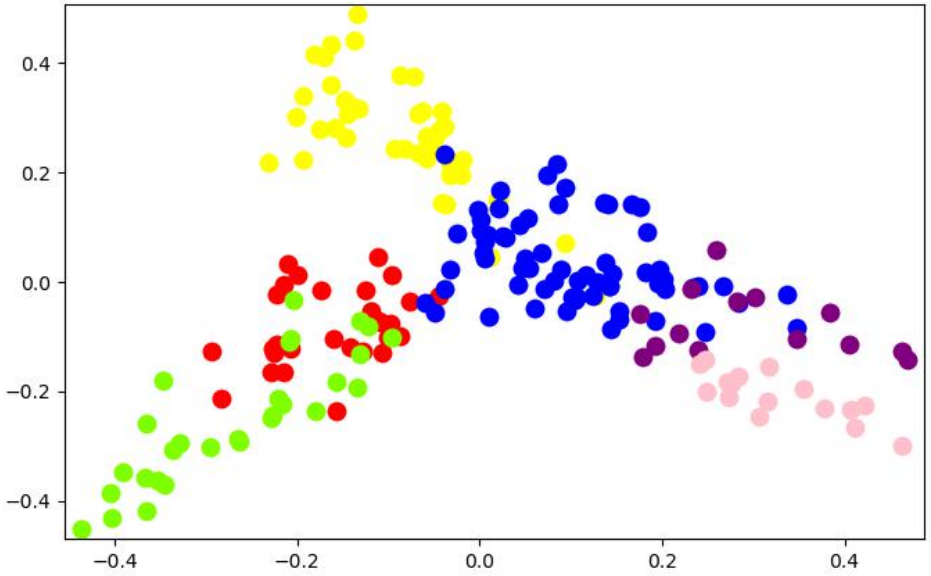


图 4. 训练集分簇结果图

4.4.4 测试文本分类

在通过训练后，选取部分的测试集内的文章来测试，调用前面训练时候已经得到的已有模型。

```
#读取相关测试集文件（略）
model_data = codecs.open('Tfidf_Result.txt', 'r', 'utf-8')
    model_output = model_data.readline()  #获取模型的第一行词信息，代表所有维度
    model_data.close()
    outputs.write(model_output)
    outputs.close()

    corpus = []

    #读取预料 一行预料为一个文档
    for line in open("test_out.txt", 'r', encoding='UTF-8').readlines():
        corpus.append(line.strip())

    #将文本中的词语转换为词频矩阵 矩阵元素 a[i][j] 表示 j 词在 i 类文本下的词频
    vectorizer = CountVectorizer(min_df=10)
```

（接下页）

（接上页）

```
    #该类会统计每个词语的 tf-idf 权值
    transformer = TfidfTransformer()

    #第一个 fit_transform 是计算 tf-idf  第二个 fit_transform 是将文本转为词频矩阵
    tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))

    #获取词袋模型中的所有词语
    word = vectorizer.get_feature_names()

    #将 tf-idf 矩阵抽取出来，元素 w[i][j]表示 j 词在 i 类文本中的 tf-idf 权重
    test_weight = tfidf.toarray()

    #  载入保存的模型
    clf = joblib.load('km.pkl')

    clf.fit_predict(test_weight)
    testRes = []
    for i in range(len(clf.labels_)):
        testRes.append(clf.labels_[i])
    return testRes
```

4.4.5 main 函数

使用并调用相关数据，并将相关需要保存的标签进行保存，便于后续的使用。

```
def main():
    files = np.load('files.npy')
    labels = np.load('labels.npy')
    trainData = files[:200]
    trainLabel = labels[:200]
    testData = files[200:]
    testLabel = labels[200:]
    theLabel = getModel()
    np.save('resTrainLabel.npy', theLabel)   #模型预测的训练集标签
    theLabel = testFunc(testData, testLabel)
    np.save('resTestLabel.npy', theLabel)   #模型预测的测试集标签
    np.save('testLabel.npy', testLabel)   #实际测试集标签
```

4.4.6 准确率计算

通过一系列的计算，我们已经对选取的测试集进行文本话题检测，并且选取一定数量的文本文章进行测试，对比实际情况，查看最后的准确率情况是怎样的。因为暂时未找到聚类标签与原有标签的自动对应关系，需要人工进行计算。其中：L1:模型聚类结果 L2：真实标签

```
def computeError(L1, L2):
    mydict = {'7': 1, '39': 5, '4': 3, '34': 0, '5': 2, '17': 4}
    #计算准确率
    rightCnt = 0

    for i in range(len(L2)): #列表元素一一对应
        tmp = mydict.get(L2[i])
        if tmp == L1[i]:
            rightCnt = rightCnt + 1
    accuracy = rightCnt / len(L2)
```

最终通过多次试验后发现准确率为：0.4444。

4.5 实验结果分析

根据整体的实验，我们得到针对已有的训练集和测试集得到的话题分类的结果并不理想，准确率只保持在 0.444，并不算高准确率。也就是在使用 k-Means 算法时，分簇的结果不理想，针对这种情况，我们针对出现这种情况进行分析：

① 算法对噪声和孤立点数据敏感,K-Means 算法将簇的质心看成聚类中心加入到下一轮计算当中,因此少量的该类数据都能够对平均值产生极大影响,导致结果的不稳定甚至错误。但是我们进行实际的实验时候，只是针对部分不会起决定性作用词汇编成停用词和长度为 1 的词语进行排除，针对词语的词性等其余的特性并未做更多的分析。

② 对初始聚类中心敏感,选择不同的聚类中心会产生不同的聚类结果和不同的准确率.随机选取初始聚类中心的做法会导致算法的不稳定性,有可能陷入局部最优的情况。

③ 无法发现任意簇,一般只能发现球状簇.因为 K-Means 算法主要采用欧式距离函数度量数据对象之间的相似度,并且采用误差平方和作为准则函数,通常只能发现数据对象分布较均匀的球状簇。

④ 文章特征使用 TF-IDF 可能不能很好反映文章本身特征，前后词袋不统一，造成新维度，准确率受到影响。

5. KNN 算法实现话题分类

5.1 KNN 算法

KNN 算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

5.2 主函数

在撰写 KNN 方式对文本进行检测分类时，前期工作与 K-Means 算法检测一致。对已有文本进行提取，通过 jieba 库进行分析，并将结果保存在 out.txt 文本文件中。详细的代码部分可以参考 4.4.1 和 4.4.2 部分。在预处理好文本后进行 KNN 相关操作如下：

5.2.1 寻找最佳主成分数量

寻找最佳主成分数量，还原度取适宜值。这里采用了 PCA 算法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。PCA 的工作就是从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。去除噪声和不重要的一些特征。

```
def findPCA(weight):
    candidate_components = range(10, 300, 30)
    explained_ratios = []
    for c in candidate_components:
        pca = PCA(n_components=c)
        X_pca = pca.fit_transform(weight)
        explained_ratios.append(np.sum(pca.explained_variance_ratio_))
                                                    #计算对原材料的还原度
    plt.figure(figsize=(10, 6), dpi=144)
    plt.grid()
    plt.plot(candidate_components, explained_ratios)
    plt.xlabel('Number of PCA Components')
    plt.ylabel('Explained Variance Ratio')
    plt.title('Explained variance ratio for PCA')
    plt.yticks(np.arange(0.5, 1.05, .05))
    plt.xticks(np.arange(0, 300, 20))
    plt.show()
```

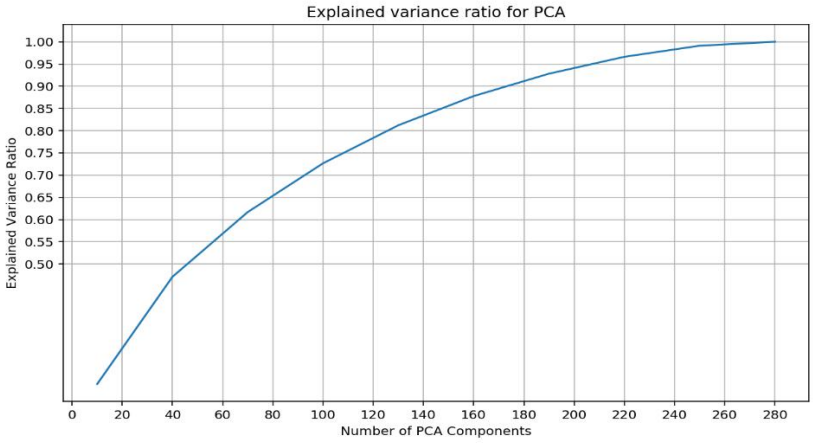


图 5. 主成分分析的解释方差比

这里取 90%处 x 值，过大数据噪点影响明显。根据上图 PCA 取值 160。

5.2.2 寻找 KNN 最佳参数

KNN 算法中，K 的值格外重要，设定 1 到 31 为备选的 K 取值。对可能存在的选值进行测试并计算相对应的准确率进行对比比较，确定 K 值。

```
def find_n_neighbors(x, y):
    k_range = range(1, 31)
    k_error = []
    #循环，取 k=1 到 k=31，查看误差效果
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        #cv 参数决定数据集划分比例，这里是按照 5:1 划分训练集和测试集
        scores = cross_val_score(knn, x, y, cv=6, scoring='accuracy')
        k_error.append(1 - scores.mean())
```

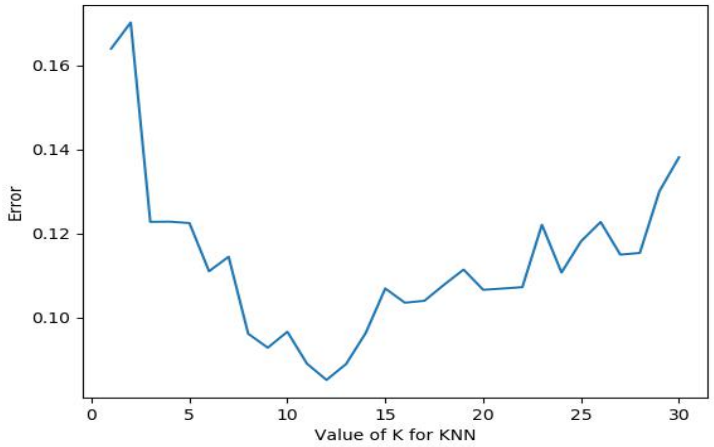


图 6. KNN 算法 K 值和错误率关系图

根据不同的 K 值制图，为了可以更加直观，将 K 值与对应的错误率在二维关系中展示。在图中，x 轴为 k 值，y 值为误差值。结合图中情况 K 取值 12。

5.2.3 训练及测试文本

根据前两步骤得到的图，分别选取 PCA=160、K=12。对文本先进行 KNN 模型的构建，之后将测试集内文本使用已有模型进行测试，可以得到相对应的测试文本类型和通过 KNN 预测得到的文本类型，计算得出最终的准确率。

```
def getModel(labels):
    corpus = []
    #读取预料  一行预料为一个文档
    for line in open('out.txt', 'r', encoding='UTF-8').readlines():
        corpus.append(line.strip())
    #将文本中的词语转换为词频矩阵  矩阵元素 a[i][j]  表示 j 词在 i 类文本下的词频
    vectorizer = CountVectorizer(min_df=10)
    #该类会统计每个词语的 tf-idf 权值
    transformer = TfidfTransformer()
    #第一个 fit_transform 是计算 tf-idf  第二个 fit_transform 是将文本转为词频矩阵
    tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))
    #将 tf-idf 矩阵抽取出来，元素 w[i][j]表示 j 词在 i 类文本中的 tf-idf 权重
    weight = tfidf.toarray()

    # findPCA(weight) ##  依据图像找到最佳 PCA 参数:160
    pca = PCA(n_components=160)
    all_pca = pca.fit_transform(weight)
    # find_n_neighbors(all_pca, labels) ##依据图像找到 KNN 最佳取值:12
    train_pca = all_pca[:200]
    test_pca = all_pca[200:]
    train_label = labels[:200]
    test_label = labels[200:]
     #定义一个 knn 分类器对象
    knn = KNeighborsClassifier(n_neighbors=12)
   #调用该对象的训练方法，主要接收两个参数：训练数据集及其样本标签
    knn.fit(train_pca, train_label)

    y_predict = knn.predict(test_pca)
    score=knn.score(test_pca,test_label,sample_weight=None)
    #输出原有标签
    print('y_predict = ')
    print(y_predict)
    #输出测试的结果
    print('y_test = ')
    print(test_label)
    #输出准确率
    print ('Accuracy:',score    )
```

某次测试后得到的结果如下图，可以看出准确率可以保持在 85%左右，相较 K-Mean 算法要高出很多。

```
y_predict =
['34' '7' '39' '5' '7' '5' '34' '39' '4' '4' '17' '4' '34' '5' '5' '4'
 '34' '34' '34' '17' '39' '5' '5' '17' '34' '39' '17' '4' '7' '39' '34'
 '4' '34' '5' '34' '39' '34' '7' '5' '4' '39' '7' '4' '39' '39' '34' '34'
 '5' '5' '4' '7' '4' '5' '5' '17' '7' '7' '39' '5' '34' '17' '17' '7' '5'
 '7' '7' '5' '5' '39' '5' '17' '17']
y_test =
['34' '7' '39' '5' '7' '5' '34' '39' '4' '4' '17' '4' '34' '5' '5' '5' '5'
 '4' '34' '17' '39' '5' '5' '17' '34' '39' '17' '4' '7' '39' '34' '5' '34'
 '34' '34' '39' '34' '7' '5' '5' '39' '4' '7' '39' '39' '34' '4' '5' '5'
 '4' '7' '34' '5' '5' '17' '7' '7' '39' '5' '34' '17' '17' '7' '5' '7' '7'
 '5' '5' '39' '5' '17' '17']
Accuracy: 0.8611111111111112
```

图 7. KNN 算法预测结果与准确率

5.3 实验结果分析

通过 KNN 算法对已有文本的话题检测分类，我们可以看出该方法下的准确率可以保持在 85%左右。相较于 K-Means 算法来看，针对话题检测，是一种准确率很高的算法。由于 KNN 方法主要靠周围有限的邻近的样本，有标签训练，而不是靠判别类域的方法来确定所属的类别，因此对于类域的交叉或重叠较多的待分类样本集来说，KNN 方法较其他方法更为适合。

6. 尝试爬取微博数据

在进行本次实验时，我们尝试去针对新浪微博网页上的数据进行爬取，但是实际工作情况并不理想。该部分将我们目前已实验程度进行展示，并对未来需要解决的问题进行分析。

微博是中国最早兴起的自媒体，积累了庞大的用户基础，各类明星动态和社会热点也都会在微博上第一时间呈现。当然还有一个更为重要的点：微博不同于 QQ 空间和微信朋友圈，它不需要添加他人好友才能阅览别人的动态，它可以无需上述繁琐的条件，直接对一个无关的人查看他的所有的个人信息。所以，微博成了一个唯一可以爬的社交媒体平台。

6.1 爬虫思路流程

在微博上，我们可以爬取的信息主要为：微博数据，个人信息，单条微博评论数据，社交关系数据。

微博在网站上目前有三个站点，分别为：https://weibo.cn/；https://m.weibo.com；https://weibo.com。第一和第三个站点是具有高级搜索功能的。其中三者爬虫的难度程度为：.cn< m.com< .com

图 8. 爬虫流程图

进行了部分实验，目前可以针对.cn 网站进行固定某个用户进行爬取相关的数据，但是目前没有实现高级搜索功能。实验结果如下图：

图 9. 爬取 https://weibo.cn/固定用户信息成功

图 10. 爬取 https://weibo.com/失败

6.2 尚待解决的问题和不足

1.只在 CN 上进行了实验，并未在更高难度上的.COM 上进行操作。

2..COM 上面的数字验证码（网上推荐使用 fuck-login）与图片汉字验证码尚未解决（尚未找到可行的处理方法）（在使用网上方法构建账号池的时候运行台出现 pkill 结果为乱码，疑是方法已过时）。

3.尚未实现高级搜索功能，只能对于特定用户进行爬取。

4.未实验多账号同时进行爬虫，只用单一账号的话容易被封。

7. 总结及展望

7.1 总结

通过分别对 KNN 算法和 KMeans 算法的准确度进行比较，我们发现 KNN 算法对文本的分类效果远远好于 KMeans 算法。因此，我们可以进一步分析并比较 KNN 算法与 KMeans 算法的优缺点：

7.1.1 KNN 算法优缺点

- 优点：
  （1）思想简单，理论成熟，既可以用来做分类也可以用来做回归；
  （2）可用于非线性分类；
  （3）训练时间复杂度为 O(n)；
  （4）准确度高，对数据没有假设，对 outlier 不敏感。
- 缺点：
  （1）计算量大；
  （2)样本不平衡问题(即有些类别的样本数量很多,而其它样本的数量很少)；
  （3）需要大量的内存；

7.1.2 K-Means 算法优缺点

- 优点：
  （1）原理比较简单，实现也是很容易，收敛速度快；
  （2）聚类效果较优；算法的可解释度比较强；
  （3）主要需要调参的参数仅仅是簇数 k。
- 缺点：
  （1）不同的初始聚类中心可能导致完全不同的聚类结果。算法速度依赖于初始化的好坏，初始质点距离不能太近；
  （2）如果各隐含类别的数据不平衡，比如各隐含类别的数据量严重失衡，或者各隐含类别的方差不同，则聚类效果不佳；
  （3）对噪音和异常点比较的敏感。
  （4）采用迭代方法，得到的结果只是局部最优。

7.2 K-Means 算法的改进[9]

针对 K-Means 算法的缺点，有许多现有的算法对其进行改进：

[9] https://www.cnblogs.com/eilearn/p/9046940.html

7.2.1 K-Means++算法

K-Means++算法对 K-Means 随机初始化质心的方法的优化策略如下：

a) 从输入的数据点集合中随机选择一个点作为第一个聚类中心

b) 对于数据集中的每一个点，计算它与已选择的聚类中心中最近聚类中心的距离

c) 选择一个新的数据点作为新的聚类中心，选择的原则是：距离较大的点，被选取作为聚类中心的概率较大

d) 重复 b 和 c 直到选择出 k 个聚类质心

e) 利用这 k 个质心来作为初始化质心去运行标准的 K-Means 算法。

7.2.2 二分-K 均值

二分-K 均值是为了解决 k-均值的用户自定义输入簇值 k 所延伸出来的自己判断 k 数目，针对初始聚类中心选择问题，其基本思路是：为了得到 k 个簇，将所有点的集合分裂成两个簇，从这些簇中选取一个继续分裂，如此下去，直到产生 k 个簇。

7.2.3 大样本优化 Mini Batch K-Means

也就是用样本集中的一部分的样本来做传统的 K-Means，这样可以避免样本量太大时的计算难题，算法收敛速度大大加快。当然此时的代价就是我们的聚类的精确度也会有一些降低。一般来说这个降低的幅度在可以接受的范围之内