微博文本挖掘关键技术研究与实现*

李陶深 于斐钥

广西大学计算机与电子信息学院，南宁，530004

摘 要 数据挖掘技术是人们分析数据和处理数据的有效手段。针对微博数据的分析和挖掘需求，本文研究微博文本挖掘关键技术，提出对微博文本进行挖掘的技术方案。在该方案中，首先利用 Python 语言的爬虫技术，从微博账户相关网站和微博网站采集数据；然后对采集到的数据人工判断，把积极语句逐条复制到指定文件；接着去掉不需要的字符，并利用 TF-IDF 词频向量化对采集到的的数据进行预处理；然后对数据进行降维，对降维后的数据集进行默认交叉验证划分和五层折叠交叉验证；最后通过逻辑回归（Logistic Regression）、朴素贝叶斯(Naive Bayes)、支持向量机（SVM）、K 最近邻（K-Nearest Neighbor，KNN）等常用的分类器，对预处理后的微博文本数据集进行分类。

关键字 微博文本，数据挖掘，Python，分类器

Research on Key Technologies of Microblog Text Mining

Li Taoshen Yu Feiyao

School of Computer, Electronics & Information
Guangxi University
Nanning 530004,China
tshli@gxu.edu.cn

Abstract—Data mining technology is an effective means for people to analyze and process data. Aiming at the analysis and mining requirements of microblog data, this paper studies the key technologies of microblog text mining, and puts forward the technical scheme of microblog text mining. Firstly, the crawler technology of Python language is used to collect data from microblog account related websites and microblog. Secondly, the collected data is manually divide into negative statements and positive statements, and copy them to the specified file respectively. Thirdly, unnecessary characters are removed, and the collected data is preprocessed by using TF-IDF word frequency vectorization. Fourthly, the data dimension is reduced, and the data set obtained after dimension reduction is carried on default cross validation and five layer folding cross validation. Finally, logistic regression, naive Bayes, support vector machine (SVM), k-nearest neighbor(KNN) and other classifiers are used to classify the preprocessed micro blog text data set.

Key words—microblog text, data mining，Python，classifier

1 引 言

微博是现下国内互联网最大交流平台之一，它所产生的庞大数据背后蕴藏着巨大的商业价值和社会价值。人们试图对庞大的微博数据进行各种微博语料分析，以获取所需的字段或者有用信息。

文本挖掘技术是人们分析数据和处理数据的有效手段，它可以从文本语料库中发现、检索和提取信息。更具体地讲，文本挖掘结合了自然语言处理、人工智能、信息检索和数据挖掘等技术，以帮助理解复杂的书面分析处理系统。目前，文本挖掘的准确性和处理复杂问题的能力稳步提高。微博文本挖掘可以分为微博的数据获取、文本的语义挖掘方法以及文本挖掘的应用 3 部分[1]。微博的数据获取技术大致可以分为基于 API 的获取技术、基于网络爬虫的获取技术和基于网络数据流的获取技术[1]。文献[2]利用新浪官方提供的数据 API 接口，从新浪网站上获取所需的数据。文献[3]利用网络爬虫技术，设计了一个基于 Python 的新浪微博爬虫系统，通过使用 scrapy 多线程爬虫框架，实现对微博网页的实时数据爬取。文献[4]通过特定的应用协议识别，实时在网络中抓取正在传输的数据包，并进行数据抽取还原。文献[5]和[6]研究并提出了多策略融合的微博数据采集方案，能够比较高效、稳定地获取微博数据。基于 API 的数据获取技术比较简单，效率较高，但是数据抓取不全面，实时获取的效果不好。基于网络爬虫的数据获取技术获取数据比较全，但是需要身份验证。基于网络数据流的数据获取技术不受身份验证限制，可以容易获取的指定区域的网页数据，但是对硬件和软件的要求较高，不容易实现。

*基金资助：本文得到得到广西科技计划项目（桂科 AD20297125）和广西高等教育本科教学改革工程项目一般项目(2020JGA116)资助.

微博文本的语义挖掘方法包括微博文本预处理、微博挖掘处理 2 部分。针对微博文本数据具有不规范、口语化、碎片化等特点，微博文本预处理的主要任务是利用分词、去停用词、特征提取等手段去除微博中的噪音信息[1]。微博挖掘处理任务是分类与聚类、热点话题检测、情感分析等。文献[7]研究了中文微博中的情感表达特征，提出了一种微博文本情绪显性特征的多策略集成分析法。文献[8]提出了一种能够同时考虑关键词和发生时间的微博文本数据挖掘算法。文献[9]将高效用模式挖掘算法引入微博文本处理中，研究了微博文本突发话题检测方法。文献[10]运用一种称之为 MBUT-LDA 的主题建模方法去挖掘微博数据的主题，提高了微博文本挖掘的准确度。

目前，有关微博文本的语义挖掘方法的研究成果不是太多，还需要做比较深入和广泛的研究。本文对微博文本挖掘关键技术进行研究，提出对微博文本进行挖掘的技术方案。

2 关键技术研究与实现

如前所述，微博文本挖掘关键技术主要包括微博文本数据的获取、数据预处理、特征标注与选择、微博数据进行主题分类、对分类结果进行评估。下面介绍本文对这些关键技术的设计考虑。

2．1 微博文本数据的获取与标注

在微博文本数据挖掘技术研究与实现时，首先要考虑确定好微博文本数据集。这里涉及的技术有数据获取与标注。本文的数据集是通过爬虫程序，从指定的浏览器微博首页上爬取 3 万多的微博文本数据。爬取得到的用户微博内容放在 csv 表格中，然后手工分辨用户发布的微博内容，一条条判断，将消极语句复制粘贴到 neg60000.txt 中，将积极语句复制粘贴到 pos60000.txt 中，最终处理的结果是：消极语句 1.5 万条左右，积极语句 1.5 万条左右。

具体的操作介绍如下。

（1） 指定目标爬虫的网页 url

因为普通的 web 版微博网页的请求被 js 加密过，并且有接口要考虑。为了简化获取方式，本文采用了微博 m 站即 m.weibo.cn 作为目标 url。

具体实现时，调用 selenium 中的 webdriver 包实现。为了控制网页行为代替鼠标操控，先用 webdriver.Firefox()打开浏览器，且定义为 driver，之后的主要功能通过 driver 来实现。然后使用 driver.get(url)输入链接并加载，同时通过 js 控制鼠标把页面拉到位置值为 3 万的地方。driver.execute_script(js)调用 js，将页面拖到底使得最大限量的评论显示出来，代码如下所示。

```
from selenium import webdriver   #控制鼠标行为
import time   #休眠、记录当前时间
inport xlwt    #写入文件用
url = ‘https://m.weibo.com

driver   =   webdriver.firefox(executable_path=r’C:/progrom Files (x86)/Mozilla Firefox/geckdriver.exe’    #  打开浏览器
driver.get(10)   #输入链接并加载
time.sleep       #休眠一下
js="var q=document.Document Element.scrollTop =30000"
```

Selenium 库中定位的方式有 id 定位、name 定位、class 定位等，此处使用 driver.find_ element_by_class_ name("more_txt").click()，通过 class_name 定位、重复点击“查看更多”，来使全部的评论都显示出来。其中 class 定位：find_element_by_class_name(self, name) 通过查找网页的 class 元素属性，只要查找的部分具有 class 属性，就可以通过 class_name 进行查找。但是，需要注意的是，在一个页面中可能有多个具有相同 class_name 的元素，因此该方法相对适合于页面上唯一具有 class 属性的元素。

（2） 数据的爬取

通过点击开发者工具，可以手动寻找 class_name 并获取所需的网页。根据已经全部显示的需要爬的内容，可以爬取每一条评论用户昵称、评论内容、评论用户的链接等，用户昵称、评论内容、评论用户等通过调用 driver.find_element_by_xpath 获取，其中 xpath 定位使用 find_element_by_xpath(self, xpath)，xpath 从网页复制获取，如图 1 所示。实现代码如图 2 所示。

图 1 获取 XPath

爬取评论并储存列表的实现代码如下所示。

```
for i in rang(10000):   #循环读取微博评论
  tem=[]    #临时存储列表
  k=str(i+1)   #转换类型，字符串合并
  #a: 评论用户昵称+评论内容，通过 xpath 从网页获取
  a=driver.find_element_by_xpath(“/html/body/div[1]/div/
  a_1=a.split(u“:”)[0]   #截取评论用户昵称
  a_2=a[a.find(u“:”)+1:len(a)]   #截取用户评论内容
  #d: 评论用户 url 地址，通过 xpath 获取
  d=driver.find_element_by_xpath(“/html/body/div[1]/div/
```

（3） 获取数据的存储

对于获取的数据，最后通过调用 pd.DataFrame.to_csv 将评论存储列表 res_PL 写入 csv 文件中。写入的代码如下所示，其中通过 columns_name 定义表头。

```
colume_name=[u'评论用户昵称', u'评论内容',u'评论用户链接', u'评论用户 ID',u'评论时间']    #定义表头
#定义表的内容
res_tem=pd.DataFrame(colume=colums_name,data=res_PL)
#写入表的内容
res_tem.to_csv(u"C:\\Users\\Administrator\\Desktop\\评论用户数据.csv",encoding='utf_8_sig)
```

2．2 数据预处理

数据预处理（Data Preprocessing）是指在对数据进行分类以前，先对原始数据进行必要的清理转换、特征选择和提取等一系列处理工作，达到挖掘算法进行知识获取研究所要求的最低规范和标准[11]。

数据挖掘的对象是数据集。现实中采集到的原始数据存在的问题有：较为分散，不够完整；噪声较多，数据随机性答；来源广泛，收集和处理起来比较困难。所以采集到的数据一般都不能满足数据挖掘算法的要求，必须对数据进行预处理，以提高数据质量，符合规范和要求。

（1） 导入数据并标注

处理思路是：把neg60000.txt里的数据集逐行导入到neg=[]列表中，再建立一个DataFrame({})型的二维数据框，将含有消极语句的neg列表赋给DataFrame({})型的二维数据框表格中的text竖列，并全部标注为0，即表格中lab全部标为0。积极数据集的处理同上。具体的实现介绍如下。

对于消极语句数据集，使用f = open('情感分析60000/neg60000.txt')打开消极语句数据集txt文件，建立一个neg = []列表，通过使用f.readline()逐行读取消极语料集中的语句，用neg.append(line)把读到的一行行语句添加进neg列表中，最后用f.close()关闭微博文本消极数据集。实现代码如下。

```
f=open('情感分析60000/neg60000.txt')
neg=[]
line=f.readline()
neg.append(line)
while line:
    line=f.readline()
    neg.append(line)
f.close()
```

对于积极语句数据集，使用f = open('情感分析60000/pos60000.txt')打开积极语句数据集txt文件，建立一个pos = []列表，通过使用f.readline()一行行读取积极语料集中的语句，用pos.append(line)把读到的一行行语句添加进pos列表中，最后用f.close()关闭微博文本消极数据集。实现代码如下。

```
f=open('情感分析60000/pos60000.txt')
pos=[]
line=f.readline()
pos.append(line)
while line:
    line=f.readline()
    pos.append(line)
f.close()
```

建立消极语句和积极语句数据列表后，调用pandas库的DataFrame({})函数，构建两个DataFrame型的二维数据框df1、df2。将neg[]列表全部放入df1表格中的df1['text']里，即全部放入表格中text那一栏的竖列里，并将df1['lab']即表格lab栏，全部赋值为0。将pos[]列表全部放入df2表格中的df2['text']里，即全部放入表格中text那一栏的竖列里，并将df2['lab']即表格lab栏，全部赋值为1。

得到含有标注和微博文本语句的df1、df2二维数据框后，调用pandas库中的concat()函数，通过pd.concat([df1,df2])来纵向合并df1、df2数据框为一个数据框df。然后用df.index = range(df.shape[0])语句对df数据框中每行数据进行重新排序，使得df数据框里的index标签不再是乱的。然后再从sklearn.utils库中引入shuffle函数，通过df = shuffle(df)来避免过拟合、打乱df数据框中的数据顺序，实现代码如下。

```
df1=pd.DataFrame({})
df1[‘text’]=neg
df1[‘lab’]=0
df2=pd.DataFrame({})
df2[‘text’]=pos
df2=[‘lab’]=1
df=pd.concat({})
df.index=rang([df1,df2])
df=shuffle(df)
```

（2） 清洗数据

清洗数据时，本文将文本转录为包括标点符号在内的符号条件，同时排除了非原创帖子包括广告帖子和转帖，在转录过程中不进行任何更改完成的转录工作。清洗数据时使用了停用词表去掉停用词。本文还通过删除text中数据有缺失的行和通过正则表达式和过滤重复行等三种方法来进一步清洗数据，实现对清洗数据方面的优化。具体的做法如下：

① 为了清洗数据、删除在二维数据框df中有缺失值的行，本文调用了pandas库中dataframe函数的dropna()方法，通过df.dropna(subset=['text'], inplace=True)可以删除df数据框中text中有缺失值（为NaN）的行。代码如下所示。

```
df.dropna(subset=[‘text’],inplace=True)
df.index=rang(df.shape[0])
```

② 为了清洗数据、过滤在二维数据框df中有重复行的目的，本文调用了pandas库中dataframe函数的

drop_duplicates()方法，通过df.drop_duplicates(subset=['text'], inplace= True)去除df二维数据框text那一竖列里有值重复的行。再通过df.index = range(df.shape[0])重新排序，以便使df数据框中index不是乱的。代码如x下所示。

```
df.drop_duplicates(subset=[‘text’],inplace=True)
df.index=rang(df.shape[0])
```

③ 正则表达式更常见的用途之一是查找所有匹配的字符串，并用不同的字符串替换它们。为此，本文使用sub方法提供一个替换值，可以是字符串或函数，也可以是要处理的字符串。Re.sub的功能是用给定的替换内容替换匹配模式的子字符串。具体操作时，通过导入python内置的re模块来使用正则表达式，调用re模块里的sub()方法去除与正则表达式含义相同的text里的字符、字符串或符号。正则表达式里使用'\'作为转义字符正则表达式，诸如[a-z0-9A-Z]：表示一个属于a-z或0-9或A-Z的字符、\s：不可见的字符包括空格、换行符。具体的实现代码如下所示。

```
text=re.sub(“[\s+\.\!\/_,$%^*(+\”\’]+|[+--！ ，。 ？ 、~@#%.....&*()]+|[\d+]}[a-zA-Z+]|@|・|【|】”,“”,text)
text=text.replace('-',' ')
```

在上面的代码中，replace函数是pandas包中适用于批量替换情况的函数，使用Replace(old_text，new_text)替换函数，把text = text.replace('-','')括号左边的字符串替换成括号右边的。在这里，直接替换成中间不含空格的''，实际的操作效果是把replace('-','')括号左边的全部去除，以达到数据清洗的目的。

④ 去停用词。微博文本中存在大量无意义的词语，这些词语在人们发在微博平台文本中表达自己所思所想，但并不参与文本意义的表达。这些词被称为停用词，是指在文本中出现频率高但对研究没有意义的词，如“和”、“得”、“在”、“介”，以及一些使用频率过高的词，如“我”、“啊”、“吧”。微博文本中用户发微博口语化很严重，所以去除停用词是文本预处理阶段非常重要的一步。本文从网上下载了开源的哈工大停用词txt文档，用它对文本text进行过滤，以避免分词后干扰。哈工大停用词txt文档的停用词有：同、同一、同时、同样、后来、后面、向、向着、吓、吗、否则、吧、吧哒、吱、呀、呃、呕，等等。

处理停用词的过程大致如下：通过调用open("停用词.txt",encoding='utf8')方法打开停用词文档；通过file.readlines()方法读取文档中全部的行，并把读出的内容写入到stopword[]列表中；通过在jieba分词的循环时，逐个拿jieba分好的词与停用词stopword列表中的词进行比对，把不在stopword列表里的词放入jieba分词过程中；最后返回的text里。实现的代码如下所示。

```
file=open("停用词.txt",encoding='utf8')
stopwird=[line.lstrip().rstrip() for line in file.readlines()]
file.close()

for k in jieba.cut(text):
  if ('\u4e00' <= k <='\u9fa5' and k not in stopwird) :
    all_text.append(k)
return ' '.join(all_text)
```

2．3 jieba 分词

单词是文本的最小单位。在英语句子中，单词之间有空格作为分隔符，汉语单词的连续性使得汉语分词成为文本分析中必须处理的一个关键步骤。分词是指将一个连续的中文字符串序列分成几个独立单词的过程。在Python语言中有很多分词工具，包括盘古分词、牙哈分词、jieba分词、清华图拉克分词等，它们的基本用法差不多。本文采用的分词方法是jieba，它是一种结合了基于规则和基于统计方法的中文分词工具。jieba分词提供三种模式:精简模式、全模式、搜索引擎模式。

本文选用jieba分词的精简模式进行分词，调用的函数是jieba.cut()，其中参数cut_call的默认值为false。下面通过2个示例来观察精简模式的分词结果，分别是[今天天气,真,好]和[我要,毕业,我要,毕业,我,想,及格]，精简模式与另外两种模式全模式、搜索引擎模式的代码和分词结果对比如图2、图3所示。

```
import jieba
seg='我要毕业我要毕业我想及格'
print(','.join(jieba.cut(seg)))
print(','.join(jieba.cut(seg,cut_all=True)))
print(','.join(jieba.cut_for_search(seg)))
```

```
我要,毕业,我要,毕业,我,想,及格
我,要,毕业,我,要,毕业,我,想,及格
我要,毕业,我要,毕业,我,想,及格
```

图 2 jieba 分词的三种分词表现示例 1

```
import jieba
seg='今天天气真好'
print(','.join(jieba.cut(seg)))
print(','.join(jieba.cut(seg,cut_all=True)))
print(','.join(jieba.cut_for_search(seg)))
```

```
今天天气,真,好
今天,今天天气,天天,天气,真好
今天,天天,天气,今天天气,真,好
```

图 3 jieba 分词的三种分词表现示例 2

编程实现时，首先通过import引入jieba包，由def定义一个名称为seg的函数，seg函数引入df二维数据框中的text；接下来使用jieba.cut(text)进行分词，并且使用for循环同时用正则表达式提取中文文本和去除停用词；最后将分词结果通过append(分词结果)添加到all_text=[]列表中，在seg函数结尾return返回到seg(x)

中。例如，去停用词与正则去符号与jieba分词的代码如下。

```
def seg(text)
 text=re.sub("[\s+\.\!\/_,$%^*(+\"\']+|[+--！ ，。 ？ 、~@#%.....&*()]+|[\d+]}[a-zA-Z+]|@|·|【|】”,“”,text)
 text=text.replace('-',' ')
 all_text=[]
 for k in jieba.cut(text):
  if ('\u4e00' <= k <='\u9fa5' and k not in stopwird) :
   all_text.append(k)
 return ' '.join(all_text)
```

为了调用上述定义的seg函数，本文使用了pandas库中的apply(lambda x:seg(x))函数，形成一个闭环，使用语句df['text'] = df['text'].apply(lambda x:seg(x))可以在这一步生成分词了。其中lambda的输入参数x是整个df['text']，输出是冒号后的seg(x)，输出一般是根据表达式(expression)计算得到的值或者是自定义函数，这里是上述定义的seg函数得到的all_text列表。Apply与lambda的关系是当想让方程作用在一维的向量上时，常常使用apply与lambda合用完成。

为了规整df数据框中lab的数据类型，调用pandas库中的astype('int64')函数，规定lab数据类型为int64。调用的代码如下：

```
df['lab'] = df['lab'].apply(lambda x:seg(x))
```

图4是分词后的df数据框里的前几行数据。

```
df.head()
```

	text	lab
0	怒 北京 熊麦 宠物医院 抵制 怒 怒 怒 小 胖子 北京 熊麦 宠物医院	0
1	好 想要 枚 红色 神器 泪	0
2	回复 红山区 省 道旁 一个 物流园 赤峰市 区有 几公里 经停 宽敞 管它 远近 住	1
3	右下角 偏大 左上角 米袋 旁 麻将桌 好 酷 黑夜 彩虹 衰 一家子 都 不 彼得 非 旁...	0
4	温度 风度 太 开心	1

图 4 分词后的df数据框（部分）

2．4 TF-IDF

TF-IDF是文本挖掘的文本特征提取任务中广泛使用的技术，是一种统计方法，用于评估单词对文档集或语料库中文档的重要性。如果文档中某些词在文档中出现的频率很高，并且很少出现在其他文档中，则认为具有良好的类别区分能力。

本文使用TF-IDF进行文本特征提取，并对文本数据进行特征值化，其中用到的参数如max_df=0.8是根据本文的数据集调整获得。具体实现时，是通过两个函数对原始的df数据框中的text数据进行TF-IDF词频特征向量化，即CountVectorizer .fit_transform()和TfidfTransformer .fit_transform()。

为了将语料库抽象为向量，本文采用向量空间模型将文本表示为由实数值分量组成的向量，每个分量对应一个单词，相当于将文本表示为空间中的一个点，单词矩阵的大小对应于向量的维度。处理的结果就是将语料库转换为TF-IDF向量表示。

特征提取是利用已有的特征参数构造一个低维的特征空间，将原始特征中包含的有用信息映射到少数特征上，忽略冗余的无关信息。本文的特征提取是通过映射(或变换)将原始特征转化为较少的新特征。

具体的处理步骤如下：

(1) 首先从sklearn.feature_extraction.text中引入CountVectorizer,TfidfTransformer类，然后再实例化类CountVectorizer，即实例化一个特征提取转换器，将文本转化为特征。为了具体地看到此处出来的结果，可以通过CountVectorizer类的get_feature_names()方法看一下提取的这些特征的名字，此处返回的是列表，如图5。此时将CountVectorizer的参数max_df设为0.8,意义为去除最频繁的词；min_df设为5，意义为忽略少于5个文档中出现的词。其中，min_df可以设置为范围在[0.0, 1.0]的float，也可以设置为没有范围限制的int，其默认值为1.0，含义为小于某一限度的词频忽略。实验中测试了不同的值0.01、0.1、3、5、10等，最终结果认为5较好。max_df = 0.50表示“忽略出现在50％以上文档中的术语”、max_df = 25表示“忽略超过25个文档中出现的术语”。综合实验结果表明，在三个参数值分别为0.5、0.8、25的情况 下，本文的数据集在参数为0.8时处理的效果较好且合理。

(2) 使用CountVectorizer类中的fit_transform()方法，将已转换为df['text'].values.tolist()列表型的text数据，并将文本中的词语转换为词频矩阵。图6显示了词频矩阵的实例。调用这个方法后得到的结果设为counts，它的意义是词频，即词在文本中出现的次数。

在图6的输出结果中，括号内的第1个数字表示文档的序号，第2个数字表示词的序号，第3个数字就是词频（TF），即在整个文档中出现的次数。例如，最后一个例子（32134，4042）1的意思是：该词位于序号为32134的文档，该文档中这个词的序号是4042，在整个文档中该词出现了1次。

(1) 从sklearn.feature_extraction.text中引入CountVectorizer，TfidfTransformer类，再实例化TfidfTransformer。使用TfidfTransformer类中的fit_transform(counts)方法，可以将上面步骤的counts词频计数矩阵转换为标准的tf-idf表示，即根据词频信息生成TF-IDF向量,具体实现时，本文把tf-idf向量化结果设为x_。为了更直观，图7给出了print(x_)结果，print(x_.shape)的结果为（32135,11768）。

```
print(counter.get_feature_names())
```

```
['心思', '心怡', '心急', '心情', '心惊胆战', '心想', '心想事成', '心意', '心愿', '心旷神怡', '心泉', '心灵', '心爱', '心理', '心理学',
'心生', '心疼', '心痛', '心目', '心碎', '心经', '心肝', '心脏', '心脏病', '心语', '心跳', '心酸', '必去', '必备', '必定', '必将', '必
有', '必点', '必看', '必胜', '必胜客', '必需', '忍不住', '忍住', '忍受', '忍心', '忍者', '忏悔', '志芯', '志强', '志愿', '志愿者', '志
梅', '志远', '忘不了', '忘带', '忘记', '忙活', '忙碌', '忠实', '忠诚', '忧伤', '忧郁', '忧郁症', '快乐', '快到', '快去', '快快', '快
意', '快成', '快报', '快来', '快板', '快点', '快点儿', '快要', '快讯', '快递', '快速', '快门', '念书', '念叨', '忽悠', '忽然', '忽略',
'忽视', '怀孕', '怀念', '怀抱', '怀旧', '怀疑', '怀里', '态度', '怎么回事', '怎么着', '怎能', '怒放', '怒转', '怒骂', '怕怕', '怕死',
'思密达', '思念', '思思', '思想', '思维', '思考', '思路', '急得', '急死了', '急诊', '急诊科', '急需', '性价比', '性别', '性感', '性
格', '性能', '性质', '怪不得', '怪兽', '怪诞', '总会', '总共', '总决赛', '总动员', '总厨', '总有', '总有一天', '总理', '总监', '总算',
'总经理', '总统', '总编辑', '总能', '总裁', '总部', '恋上', '恋人', '恋爱', '恐怕', '恐怖', '恐怖片', '恐惧', '恐惧症', '恐龙', '恒
大', '恢复', '恨不得', '恨少', '恨死', '恩恩', '恩施', '恩爱', '恭候', '恭喜', '恭喜发财', '恭祝', '恭贺', '恰好', '恳请', '恶劣', '恶
心', '恶性循环', '恶意', '恶搞', '恶梦', '恶毒', '恶行', '恶魔', '悄悄', '悄悄的', '悄然', '悉尼', '悟空', '悠悠', '悠游', '悠然', '悠
着点', '悠闲', '患者', '悦悦', '悦旅', '您好', '您老', '悬崖', '悬念', '悲伤', '悲剧', '悲哀', '悲壮', '悲惨', '悲愤', '悲痛', '悲观',
'情书', '情人', '情人节', '情侣', '情况', '情商', '情怀', '情意', '情感', '情报', '情景', '情歌', '情结', '情绪', '情绪化', '情节',
'情调', '情谊', '惆怅', '惊人', '惊吓', '惊呆', '惊喜', '惊心', '惊心动魄', '惊恐', '惊悚', '惊现', '惊艳', '惊讶', '惊醒', '惊闻',
'惊险', '惊魂', '惋惜', '惦记', '惦记着', '惨不忍睹', '惨剧', '惨无人道', '惨烈', '惨痛', '惩罚', '惬意', '惭愧', '想不到', '想不起
来', '想像', '想到', '想家', '想开', '想当初', '想当年', '想得到', '想念', '想想', '想来', '想法', '想点', '想瘦', '想着', '想要', '想
见', '想象', '想象力', '想起', '想过', '愉快', '愉悦', '意义', '意味', '意味着', '意境', '意外', '意大利', '意想不到', '意淫', '意犹未
尽', '意见', '意识', '愚人', '愚人节', '愚蠢', '感人', '感兴趣', '感冒', '感到', '感动', '感受', '感受一下', '感叹', '感同身受', '感
性', '感恩', '感恩节', '感悟', '感情', '感慨', '感染', '感激', '感激不尽', '感脚', '感觉', '感触', '感言', '感谢', '感谢上帝', '感谢
您', '愤怒', '愤青', '愿望', '慈善', '慈悲', '慈祥', '慎重', '慢慢', '慰问', '慷慨', '憋着', '憔悴', '憧憬', '懂事', '懂得', '懒人',
```

图 5 文本转化为特征名字（部分）

```
print(counts)
```

```
  (0, 2108)     2
  (0, 4019)     2
  (0, 5601)     1
  (0, 9333)     1
  (1, 5247)     1
  (1, 8569)     1
  (1, 8929)     1
  (2, 10)       1
  (2, 3005)     1
  (2, 4069)     1
  (2, 8990)     1
  (3, 90)       1
  (3, 2561)     1
  (3, 4879)     2
  (3, 4898)     1
  (3, 6115)     1
  (3, 11719)    2
  (4, 4737)     1
  (4, 7514)     1
  (5, 88)       1
  (5, 348)      2
  (5, 938)      1
  (5, 963)      1
  (5, 1245)     2
  (5, 1490)     2
  :     :
```

```
(32126, 7066) 1
(32126, 8171) 2
(32126, 9338) 3
(32126, 10712)        1
(32126, 11386)        1
(32127, 8454) 1
(32128, 3005) 1
(32129, 3746) 1
(32130, 9455) 1
(32130, 11075)        1
(32131, 3837) 1
(32131, 4254) 1
(32131, 6288) 1
(32131, 7379) 1
(32132, 563)  1
(32132, 1458) 1
(32132, 3832) 1
(32132, 5273) 1
(32132, 9310) 1
(32132, 10315)        1
(32132, 10906)        1
(32133, 471)  1
(32134, 2534) 1
(32134, 3447) 1
(32134, 4042) 1
```

图 6 语料库中词语的词频

```
print(x_)
```

```
  (0, 9333)     0.29802691407020704
  (0, 5601)     0.31960695642652764
  (0, 4019)     0.8199999816729695
  (0, 2108)     0.3696368243964486
  (1, 8929)     0.5665490217206482
  (1, 8569)     0.6514085711932046
  (1, 5247)     0.5046672957141211
  (2, 8990)     0.6628347302085893
  (2, 4069)     0.6521757780709337
  (2, 3005)     0.2412271527872479
  (2, 10)       0.2777162863157948
  (3, 11719)    0.6060699508590187
  (3, 6115)     0.2289319039189551
  (3, 4898)     0.2998524024385828
  (3, 4879)     0.550445376779066
  (3, 2561)     0.29694694093511
  (3, 90)       0.3149449733484112
  (4, 7514)     0.8788088026252869
  (4, 4737)     0.4771740651254104
  (5, 10804)    0.36941522610596994
  (5, 8904)     0.20288133600625272
  (5, 8859)     0.3348401514634616
  (5, 7592)     0.2096639449792961
  (5, 7326)     0.253279345522037
  (5, 6449)     0.19647002214510997
  :     :
```

```
(32126, 3012) 0.3021868679203128
(32126, 2505) 0.12455313288839738
(32126, 1691) 0.4155429200659845
(32126, 1660) 0.3933910870363576
(32126, 602)  0.1254249865806036
(32127, 8454) 1.0
(32128, 3005) 1.0
(32129, 3746) 1.0
(32130, 11075)        0.7187041383533607
(32130, 9455) 0.6953160155740364
(32131, 7379) 0.4736452345182123
(32131, 6288) 0.49664064499648586
(32131, 4254) 0.5970286742311162
(32131, 3837) 0.41540946510822074
(32132, 10906)        0.4008252243533315
(32132, 10315)        0.4781739113382639
(32132, 9310) 0.4532061855070493
(32132, 5273) 0.289442981588278
(32132, 3832) 0.3254173715449585
(32132, 1458) 0.37541982777953414
(32132, 563)  0.27327504655888496
(32133, 471)  1.0
(32134, 4042) 0.4912251866340626
(32134, 3447) 0.7950936170632008
(32134, 2534) 0.35570206089041245
```

图 7 print(x_)输出 TF-IDF 向量化的数据集

TF-IDF过程的实现代码如下：

```
counter=countVectorizer(max_df=0.8,min_df=5)
counts=Counter.fit_transform(df['text'].values.tolist())
tfidfer=TfidfTransform()
x_=tfidfer.fit_transform(counts)|
word=counter.gei_feature_names()
y=df['lab']
```

(4) 将df数据框的lab积极消极标注设为y，即y = df['lab']。y与x_ = tfidfer.fit_ transform(counts)就是本文数据集处理到现在的原始数据，之所以称之为原始数据是因为后面有些分类器需要降维，那时候还需要对x_和y进一步处理。

2．5 降维

Scikit-learn 提供了可实现降维的 TruncatedSVD 类，TruncatedSVD 的创建必须指定所需的特征数或所要选择的成分数，本文指定为100。通过调用 fit() 函数来拟合该变换，然后再通过调用 transform() 函数将其应用于原始矩阵。实现代码如下：

```
dtpca=decomposition.TruncatedSVD(n_components=100)
x=dtpca.fit_transform(x_)
```

使用 SVD 降维是因为原始数据 x_是稀疏矩阵，所以不适合用 PCA 降维，PCA 降维时发现出错。将 tf-idf 向量化结果进行 x_降维后的结果如图 9 所示。

2．6 交叉验证

为了防止模型仅在已知数据集的表现较好，在未知数据集上的表现较差，需要确保模型的泛化能力，亦即对新鲜样本的适应能力。只有保证模型的泛化能力，模型的构建才有意义。因此，交叉验证在整个建模流程中显得尤为重要。如果不对数据集进行处理，而仅是用含有标签的已知数据训练模型会得到很高分数，但却失效于对未知数据的预测，这种情况称为“过拟合”。过拟合的出现表明模型未学习到数据中的本质规律，造成模型的预测能力较差，因此，如何避免模

型的过拟合是值得关注且必须解决的问题。

```
print(x)
```

```
[[ 1.23580765e-02  6.41558595e-03  7.05314697e-03 ...  1.08537874e-03
  -3.80832226e-03  3.03163915e-03]
 [ 4.26715563e-03  2.94411295e-03  7.62984144e-04 ...  2.89649505e-02
   1.89722742e-02  1.05014973e-02]
 [ 2.19763683e-02  2.00436795e-02  1.06633098e-02 ... -2.77951780e-03
   3.29420042e-03 -5.11661467e-03]
 ...
 [ 5.53138453e-03  4.16021693e-03  3.98449995e-03 ... -3.65216421e-03
  -5.76489984e-03 -6.33462810e-03]
 [ 4.67385238e-05  3.59377398e-05  3.30911976e-05 ... -2.99904850e-04
  -1.33641987e-04  7.30892752e-04]
 [ 1.28027924e-02  1.37648619e-02  9.48455916e-03 ... -8.46481598e-03
  -1.21168982e-02 -3.02469228e-03]]
```

图 9 降维后的数据集 x

在scikit -learn中，可以使用训练集/测试集拆分和交叉验证的方法避免该种情况的出现，如图10所示，将数据集进行训练集/测试集拆分，在训练集上进行交叉验证后得到最佳模型参数，从而在测试集上得到该模型的评分。

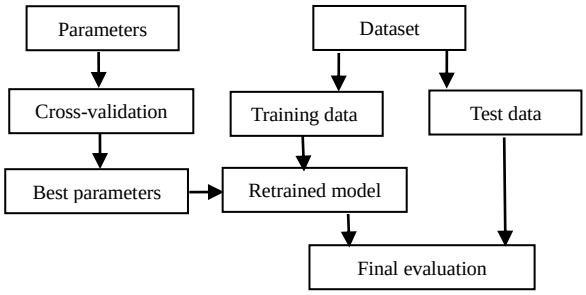

图 10 交叉验证示意图

交叉验证是机器学习中建立模型和验证模型参数的常用方法。所谓交叉验证，顾名思义就是重复使用数据，将样本数据划分为不同的训练集和测试集。训练集用于对模型进行训练，测试集用于对模型的预测进行评价。在此基础上，可以得到多个不同的训练集和测试集，一个训练集中的一个样本下一次可能会变成测试集中的一个样本，这就是“交叉”。将数据集划分为训练数据集和测试数据集，其中训练数据集用于训练模型，而测试数据集用于评估模型的泛化能力，训练学习模型的目的是选出泛化能力最强的模型。

文献[12]对朴素贝叶斯(NB)、支持向量机（SVM）、K最近邻（KNN）、XGboost、随机森林等五种分类器都进行了五折交叉验证，但是该文献的数据集过小，没有考虑分类器调参时间问题。本文所用的数据集较大，必须考虑运行时间。我们通过实验结果发现，五折交叉验证并不能使模型的准确率达到最高，这是因为k折越大，建模和调参需要花费的时间越多，且考虑本文的数据集较大，所以本文不使用文献[1]的五折交叉验证，转为使用默认交叉验证，因为默认交叉验证的运行速度更快，耗费时间更少，适合本文这种较大数据集的情况。

2．7 模型的具体选择与调参

本文在文献[1]研究的基础上，部分使用了网格调参，如XGboost。没有全部使用网格调参的原因是：已经在数据降维的情况下，调参SVM时长超过10小时，cpu过热设备不允许，故中断网格调参。下面介绍各个分类器的调参方式。

（1）朴素贝叶斯分类器

在参数alpha同为默认值1.0的情况下，本文用SKlearn 工具包中的多项式朴素贝叶斯（Multinomial NB）和伯努利朴素贝叶斯（BernoulliNB）算法分别生成一个多项式朴素贝叶斯分类器和一个伯努利朴素贝叶斯分类器。实验结果表明，对未降维的数据集，使用BernoulliNB分类器精度更高，准确率为0.79。

（2）KNN分类器

为了在KNN算法中寻找最好的K，本文首先从sklearn.neighbors中导入类KNeighbors Classifier，在循环内通过KNeighborsClassifier(n_neighbors=k)继承一个类建立KNN分类器；然后再通过fit()函数训练模型，通过测试集得出模型评分。在不同K值下循环找到模型评分最高的模型的K值，则此K值为最好的K距离。

（3）支持向量机分类器（SVM）

本文是通过GridsearchCV(svc, parameters2, cv=5, scoring='accuracy')算法进行网格调参的，其中的参数kernel在四种核linear、poly、rbf、sigmod里选择；参数c通过调用numpy库里的linspace函数构建一个起始为0.1、终值为20、差为默认值50的等差数列；参数gamma也是通过调用numpy库里的linspace函数构建一个起始为0.1、终值为20、差为20的等差数列。在等差数列中通过网格调参选择最优模型参数，同时此处将cv设为5，进行五折交叉验证。如果运行十个小时以上，硬件不支持继续跑下去，则中断调参。

（4）Xgboost分类器

采用网格调参算法GridSearchCV获取最佳参数。数据集使用的是train_test_split()方法划分的TF-IDF向量化的原始数据x_和y。即使用的降维和默认交叉验证划分后的数据集。具体的操作步骤如下：

首先调用XGBClassifier()建立一个默认参数的xgboost分类器。参数gamma的选择范围调用了numpy库中的linspace函数。

其次使用网格调参算法GridSearchCV，得到参数max_depth在数据1、2、3中3最优。通过模型的best_params_得到最佳参数结果。

3 结束语

本文对微博文本挖掘关键技术进行研究，提出了对微博文本进行挖掘的技术方案。在该方案中，首先利用 Python 语言的爬虫技术，从微博账户相关网站和微博网站采集数据，并对采集到的数据人工判断，将消极语句和积极语句分别、逐条地人工复制粘贴到指定的文件中。接着利用 jieba 分词、正则表达式匹配不需要的字符，利用 TF-IDF 词频向量化对数据进行预处理。然后利用 TruncatedSVD 对数据进行降维，利用 train_test_split 进行数据集的默认交叉验证划分，利用 cross_val_predict 进行数据集的五层折叠交叉验证。最后，通过选择某个分类器对预处理后的微博文本数据集进行分类。

参 考 文 献

[1] 余容,李光强,尹健.微博文本挖掘研究综述[J].情报探索. 2017,(5):97-103

[2] 黄延炜,刘嘉勇.新浪微博数据获取技术研究[J].信息安全与通信保密. 2013, (6):71-73,76

[3] 罗咪.基于 Python 的新浪微博用户数据获取技术[J].电子世界. 2018,(5): 138-139

[4] 游翔,葛卫丽.微博数据获取技术及展望[J].电子科技. 2014,27(10):123-126, 132

[5] 王培名,陈兴蜀,王海舟.多策略融合的微博数据获取技术研究[J].山东大学学报(理学版). 2019,54(5):28-36,43

[6] 张辉，周成祖.多策略融合的微博数据获取技术探究[J].网络安全技术与应用. 2020,(12): 72-74

[7] 戴天翔,岑鑫,柳珺文.基于文本挖掘的微博文本情绪分析技术研究[J].科技资讯. 2017,15(7):209-212

[8] 陈兰兰,胡细玲.基于多视角聚类模型的微博文本数据挖掘算法研究[J].科技通报. 2017,33(11):129-132

[9] 欧阳双.基于高效用模式挖掘的研究[D].武汉：武汉大学，2018

[10] 张明生,邓少灵.基于 MBUT-LDA 主题模型的微博文本挖掘研究[J].电子商务. 2019,(7):70-71

[11] 王振武.大数据挖掘与应用[M].北京:清华大学出版社, 2017

[12] P Wang，Y Yan，Y Si，et al.Classification of Proactive Personality: Text Mining Based On Weibo Text and Short-answer Questions Text[J].IEEE Access, 2020,8: 97370 - 97382