

****

信 息 学 院

《社交网络技术与应用》

期末大作业 项目报告

**题 目 微博数据分析——元宇宙**

**组 员 11920192203642 袁佳哲**

**34520192201612 谢健祥**

**30220192201014 程昊天**

**34520192201617 徐荪睿**

## 实验步骤

### 数据爬取

##### 爬虫代码

代码：

导入模块

import time

import datetime

import re

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

import xlwt

driver = webdriver.Chrome()

登录

def LoginWeibo(username, password):

try:

# 输入用户名/密码登录

print('准备登陆Weibo.cn网站...')

driver.get("http://login.sina.com.cn/")

elem\_user = driver.find\_element\_by\_name("username")

elem\_user.send\_keys(username) # 用户名

elem\_pwd = driver.find\_element\_by\_name("password")

elem\_pwd.send\_keys(password) # 密码

elem\_sub = driver.find\_element\_by\_xpath("//input[@class='W\_btn\_a btn\_34px']")

elem\_sub.click() # 点击登陆 因无name属性

try:

# 输入验证码

time.sleep(15)

elem\_sub.click()

except:

# 不用输入验证码

pass

# 获取Coockie 推荐资料：http://www.cnblogs.com/fnng/p/3269450.html

print('Crawl in ', driver.current\_url)

print('输出Cookie键值对信息:')

for cookie in driver.get\_cookies():

print(cookie)

for key in cookie:

print(key, cookie[key])

print('登陆成功...')

except Exception as e:

print("Error: ", e)

finally:

print('End LoginWeibo!\n')

获取搜索结果：

def GetSearchContent(key):

driver.get("http://s.weibo.com/")

print('搜索热点主题：')

time.sleep(5)

# 输入关键词并点击搜索

item\_inp = driver.find\_element\_by\_xpath("//input[@node-type='text']")

item\_inp.send\_keys(key)

item\_inp.send\_keys(Keys.RETURN) # 采用点击回车直接搜索

time.sleep(5)

# 获取搜索词的URL，用于后期按时间查询的URL拼接

current\_url = driver.current\_url

current\_url = current\_url.split('&')[0]

global start\_stamp

global page

# 需要抓取的开始和结束日期，可根据你的实际需要调整时间

# start\_date = datetime.datetime(2022, 1, 11)

start\_date = datetime.datetime(2022, 3, 17)

end\_date = datetime.datetime(2022, 3, 18)

delta\_date = datetime.timedelta(days=1)

# 每次抓取一天的数据

start\_stamp = start\_date

end\_stamp = start\_date + delta\_date

global outfile

global sheet

outfile = xlwt.Workbook(encoding='utf-8')

while end\_stamp <= end\_date:

page = 1

# 每一天使用一个sheet存储数据

sheet = outfile.add\_sheet(start\_stamp.strftime("%Y-%m-%d-%H"))

initXLS()

# 通过构建URL实现每一天的查询

url = current\_url + '&typeall=1&suball=1&timescope=custom:' + start\_stamp.strftime(

"%Y-%m-%d-%H") + ':' + end\_stamp.strftime("%Y-%m-%d-%H") + '&Refer=g'

driver.get(url)

handlePage() # 处理当前页面内容

start\_stamp = end\_stamp

end\_stamp = end\_stamp + delta\_date

获取页面按钮结果：

def **handlePage**():

*while* True:

*# 之前认为可能需要sleep等待页面加载，后来发现程序执行会等待页面加载完毕*

*# sleep的原因是对付微博的反爬虫机制，抓取太快可能会判定为机器人，需要输入验证码*

        time.**sleep**(1)

*# 先行判定是否有内容*

*if* **checkContent**() *and* page <= 50:

**print**("getContent")

**getContent**()

*# 先行判定是否有下一页按钮*

*if* **checkNext**():

*# 拿到下一页按钮*

                next\_page\_btn = driver.**find\_element\_by\_css\_selector**("#pl\_feedlist\_index > div.m-page > div > a.next")

                next\_page\_btn.**click**()

*else*:

**print**("no Next")

*break*

*else*:

**print**("no Content")

*break*

判断页面加载完成后是否还有内容

*# 判断页面加载完成后是否有内容*

def **checkContent**():

*# 有内容的前提是有“导航条”？错！只有一页内容的也没有导航条*

*# 但没有内容的前提是有“pl\_noresult”*

*try*:

        driver.**find\_element\_by\_xpath**("//div[@class='card card-no-result s-pt20b40']")

        flag = False

*except*:

        flag = True

*return* flag

判断是否有下一页按钮

# 判断是否有下一页按钮

def checkNext():

try:

driver.find\_element\_by\_css\_selector("#pl\_feedlist\_index > div.m-page > div > a.next")

flag = True

except:

flag = False

return flag

判断是否有展开全文按钮

*# 判断是否有展开全文按钮*

def **checkqw**():

*try*:

        driver.**find\_element\_by\_xpath**(".//div[@class='content']/p[@class='txt']/a")

        flag = True

*except*:

        flag = False

*return* flag

初始化excel

*# 在添加每一个sheet之后，初始化字段*

def **initXLS**():

    name = ['博主昵称', '微博认证', '微博内容','发布时间', '转发', '评论','赞']

    global row

    global outfile

    global sheet

    row = 0

*for* i *in* range(**len**(name)):

        sheet.write(row, i, name[i])

    row = row + 1

    outfile.**save**("./微博数据0317.xls")

写入excel

*# 将dic中的内容写入excel*

def **writeXLS**(*dic*):

    global row

    global outfile

    global sheet

*for* k *in* *dic*:

*for* i *in* range(**len**(*dic*[k])):

            sheet.write(row, i, *dic*[k][i])

        row = row + 1

    outfile.**save**("./微博数据0317.xls")

获取页面内容

# 在页面有内容的前提下，获取内容

def getContent():

    # 寻找到每一条微博的class

    try:

        nodes = driver.find\_elements\_by\_xpath("//div[@class='card-wrap']/div[@class='card']")

    except Exception as e:

        print(e)

    # 在运行过程中微博数==0的情况，可能是微博反爬机制，需要输入验证码

    if len(nodes) == 0:

        input("请在微博页面输入验证码！")

        url = driver.current\_url

        driver.get(url)

        getContent()

        return

    dic = {}

    global page

    print(start\_stamp.strftime("%Y-%m-%d-%H"))

    print('页数:', page)

    page = page + 1

    print('微博数量', len(nodes))

    for i in range(len(nodes)):

        dic[i] = []

        try:

            BZNC = nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='txt']").get\_attribute("nick-name")

        except:

            BZNC = ''

        print('博主昵称:', BZNC)

        dic[i].append(BZNC)

        try:

            #WBRZ = nodes[i].find\_element\_by\_xpath(".//div[@class='info']/div/a[contains(@title,'微博')]").get\_attribute('title') # 若没有认证则不存在节点

            WBRZ = nodes[i].find\_element\_by\_xpath(".//div[@class='avator']/a/span[@class = 'woo-icon-wrap woo-avatar-icon']").get\_attribute(

                'title')  # 若没有认证则不存在节点

        except:

            WBRZ = ''

        print('微博认证:', WBRZ)

        dic[i].append(WBRZ)

        # 判断展开全文和网页链接是否存在

        try:

            nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='txt']/a[@action-type='fl\_unfold']").is\_displayed()

            flag = True

        except:

            flag = False

        # 获取微博内容

        try:

            if flag:

                nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='txt']/a[@action-type='fl\_unfold']").click()

                time.sleep(1)

                # WBNR = nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[2]").text.replace("\n","")

                WBNR = nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@node-type='feed\_list\_content\_full']").text.replace("\n","")

                # 判断发布位置是否存在

                try:

                    nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='txt']/a/i[@class='wbicon']").is\_displayed()

                    flag = True

                except:

                    flag = False

                # 获取微博发布位置

                try:

                    if flag:

                        pattern = nodes[i].find\_elements\_by\_xpath(".//div[@class='content']/p[2]/a[i[@class='wbicon']]")

                        if isinstance(pattern,list):

                            text = [p.text for p in pattern]

                            FBWZ = [loc for loc in [re.findall('^2(.\*$)', t) for t in text] if len(loc) > 0][0][0]

                        else:

                            text = pattern.text

                            FBWZ = re.findall('^2(.\*$)',text)[0]

                    else:

                        FBWZ = ''

                except:

                    FBWZ = ''

            else:

                WBNR = nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='txt']").text.replace("\n","")

                # 判断发布位置是否存在

                try:

                    nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='txt']/a/i[@class='wbicon']").is\_displayed()

                    flag = True

                except:

                    flag = False

                # 获取微博发布位置

                try:

                    if flag:

                        pattern = nodes[i].find\_elements\_by\_xpath(".//div[@class='content']/p[@class='txt']/a[i[@class='wbicon']]")

                        if isinstance(pattern,list):

                            text = [p.text for p in pattern]

                            FBWZ = [loc for loc in [re.findall('^2(.\*$)', t) for t in text] if len(loc) > 0][0][0]

                        else:

                            text = pattern.text

                            FBWZ = re.findall('^2(.\*$)',text)[0]

                    else:

                        FBWZ = ''

                except:

                    FBWZ = ''

        except:

            WBNR = ''

        print('微博内容:', WBNR[:5])

        dic[i].append(WBNR)

        # print('发布位置:', FBWZ)

        # dic[i].append(FBWZ)

        try:

            FBSJ = nodes[i].find\_element\_by\_xpath(".//div[@class='content']/p[@class='from']/a[1]").text

        except:

            FBSJ = ''

        print('发布时间:', FBSJ)

        dic[i].append(FBSJ)

        try:

            ZF\_TEXT = nodes[i].find\_element\_by\_xpath(".//a[@action-type='feed\_list\_forward']").text

            # ZF\_TEXT = nodes[i].find\_element\_by\_xpath(".//div[@class='content']").text

            if ZF\_TEXT == '转发':

                ZF = 0

            else:

                ZF = int(ZF\_TEXT.split(' ')[0])

        except:

            ZF = 0

        print('转发:', ZF)

        dic[i].append(ZF)

        # print('\n')

        try:

            PL\_TEXT = nodes[i].find\_element\_by\_xpath(".//a[@action-type='feed\_list\_comment']").text

            # ZF\_TEXT = nodes[i].find\_element\_by\_xpath(".//div[@class='content']").text

            if PL\_TEXT == '评论':

                PL = 0

            else:

                PL = int(PL\_TEXT.split(' ')[0])

        except:

            PL = 0

        print('评论:', PL)

        dic[i].append(PL)

        # print('\n')

        try:

            Z\_TEXT = nodes[i].find\_element\_by\_xpath(".//span[@class='woo-like-count']").text

            # ZF\_TEXT = nodes[i].find\_element\_by\_xpath(".//div[@class='content']").text

            if Z\_TEXT == '赞':

                Z = 0

            else:

                Z = int(Z\_TEXT.split(' ')[0])

        except:

            Z = 0

        print('赞:', Z)

        dic[i].append(Z)

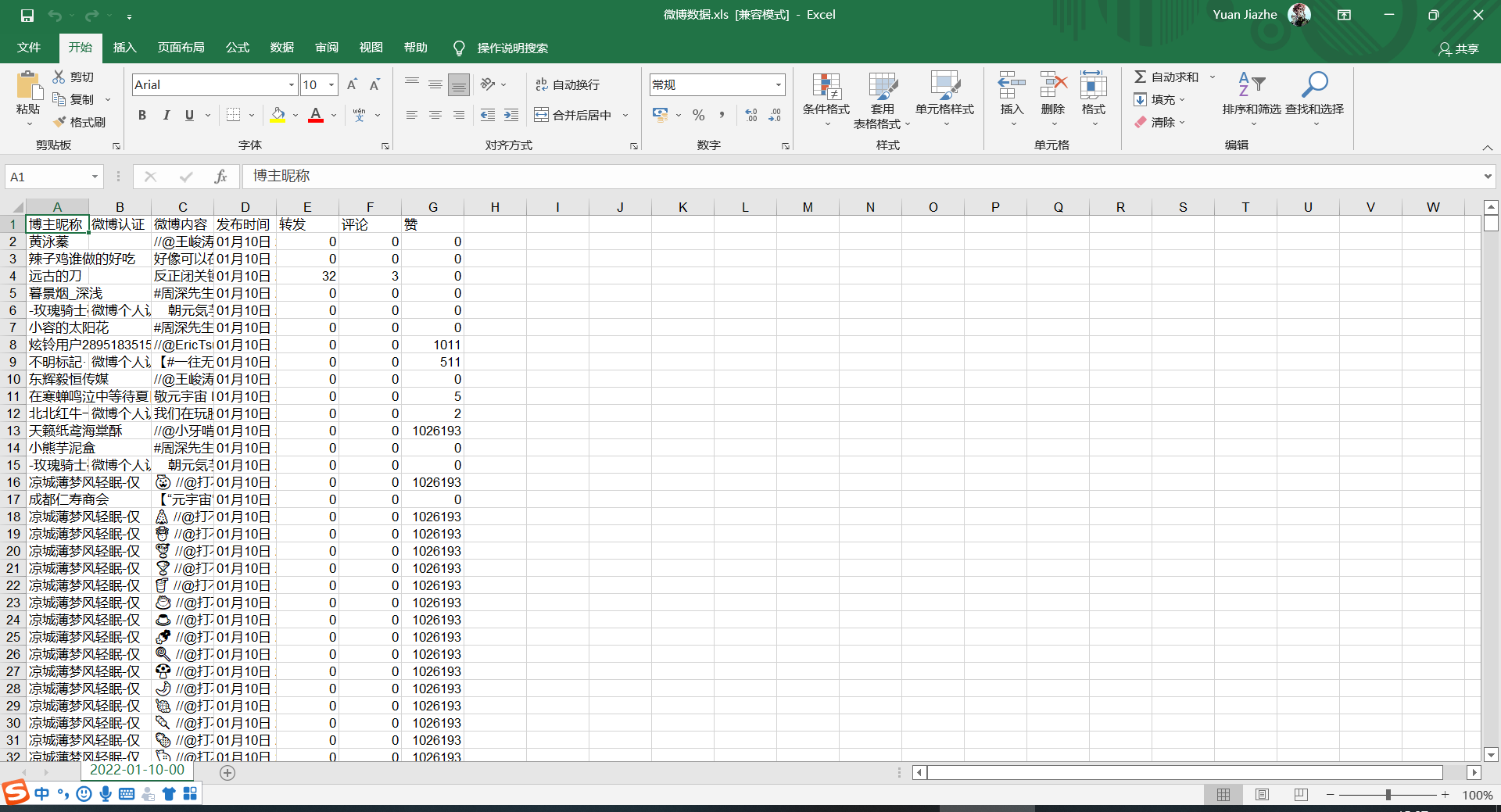
        print('\n')

    # 写入Excel

    writeXLS(dic)

输出：





### 数据预处理与存储

数据整合（将数据整合成一个csv）

import os

import pandas as pd

import warnings

from tqdm import tqdm

warnings.filterwarnings('ignore')

raw\_dataframe = pd.DataFrame()

def update\_file(file\_path):

    global raw\_dataframe

    print("\ncurrent file：", file\_path)

    excel\_file = pd.ExcelFile(file\_path)

    sheet\_name = excel\_file.sheet\_names

    for i in tqdm(sheet\_name):

        df\_tmp = pd.read\_excel(file\_path, sheet\_name=i)

        if raw\_dataframe.empty:

            raw\_dataframe = df\_tmp

        else:

            raw\_dataframe = pd.concat([raw\_dataframe, df\_tmp], axis=0, ignore\_index=True)

    print("file %s read completed" % file\_path)

def walk\_dir(dir\_path):

    for curr, dirs, files in os.walk(dir\_path):

        for file in files:

            if not file.endswith(".xlsx"):

                continue

            fpath = os.path.join(curr, file)

            update\_file(fpath)

# 数据预处理

def preHandle(df):

    df['微博内容'] = df['微博内容'].str.replace(r'[^\w]+', '')

    df.drop\_duplicates(['微博内容'], keep='last', inplace=True)

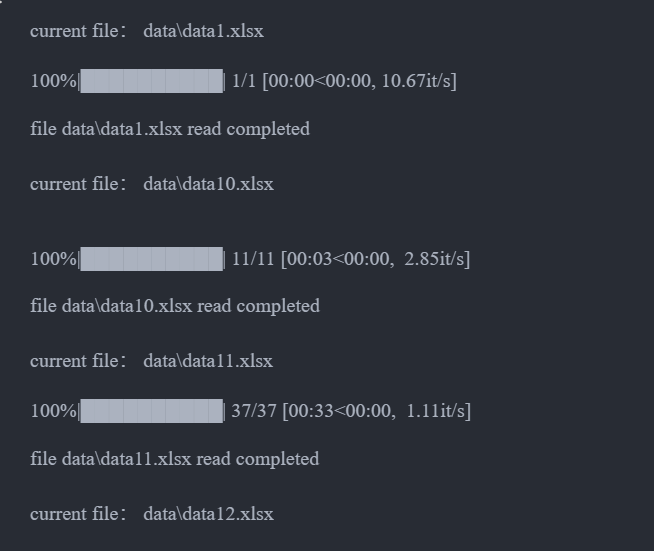
    df.reset\_index(drop=True, inplace=True)

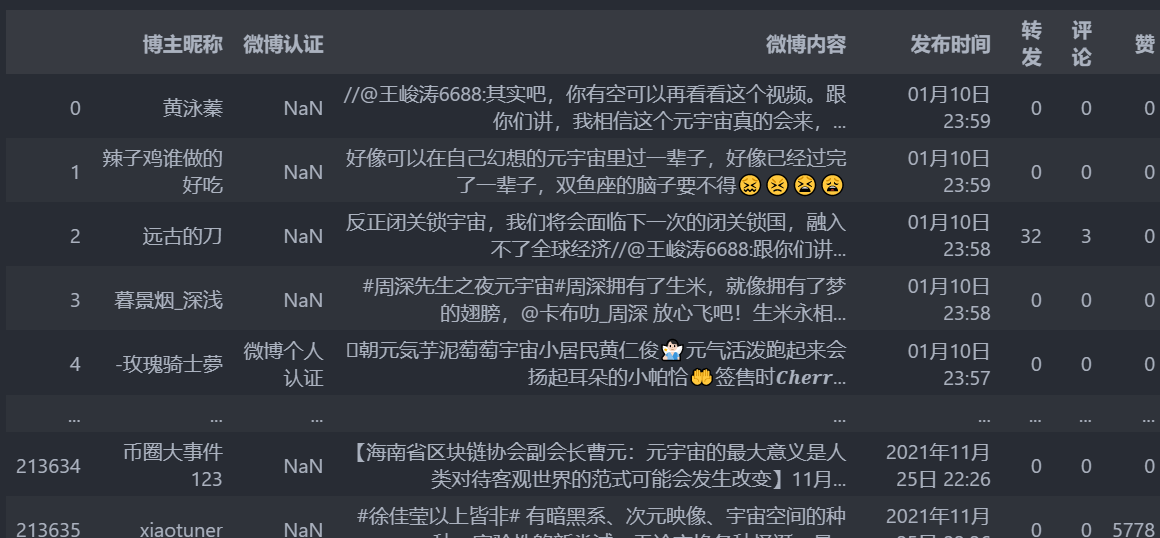
walk\_dir("data")

print("ready to preprocess")

raw\_dataframe

输出结果：





数据导出：

raw\_dataframe.rename(columns={'博主昵称': 'weibo\_username', '微博认证': 'weibo\_verification', '微博内容': 'weibo\_content', '发布时间': 'submit\_time', '转发': 'forward\_counts', '评论': 'comments\_counts', '赞': 'points\_times'}, inplace=True)

raw\_dataframe.head(10)

raw\_dataframe.to\_csv("rawdata.csv", index=0, encoding='utf-8')

print("export successfully")



数据预处理：

此处先去除非中文字符，然后根据字符串匹配去重。这里我们不选择进行句间重复词处理，因为这样做可能影响之后的词语重要性分析

def preHandle(df):

    df['weibo\_content'] = df['weibo\_content'].str.replace(u'[^\u4e00-\u9fa5]', '')

    df.drop\_duplicates(['weibo\_content'], keep='last', inplace=True)

    df.reset\_index(drop=True, inplace=True)

preHandle(raw\_dataframe)

raw\_dataframe.to\_csv("rawdata\_deduplicate\_replcae.csv", index=0, encoding='utf-8')

raw\_dataframe.head()

输出结果：



分词+去停用词：

import jieba

from tqdm import tqdm

data = raw\_dataframe['weibo\_content'].values

jieba.load\_userdict('userdict.txt')

seg\_data = []

words = []

stopwords = []

seg\_posts=[]

for word in open('stop\_words.txt','r',encoding='utf-8'):

    stopwords.append(word.strip())

for line in tqdm(data):

    seglist = jieba.cut(line, cut\_all=False)

    tmp = ''

    l = []

    for word in seglist:

        if word not in stopwords:

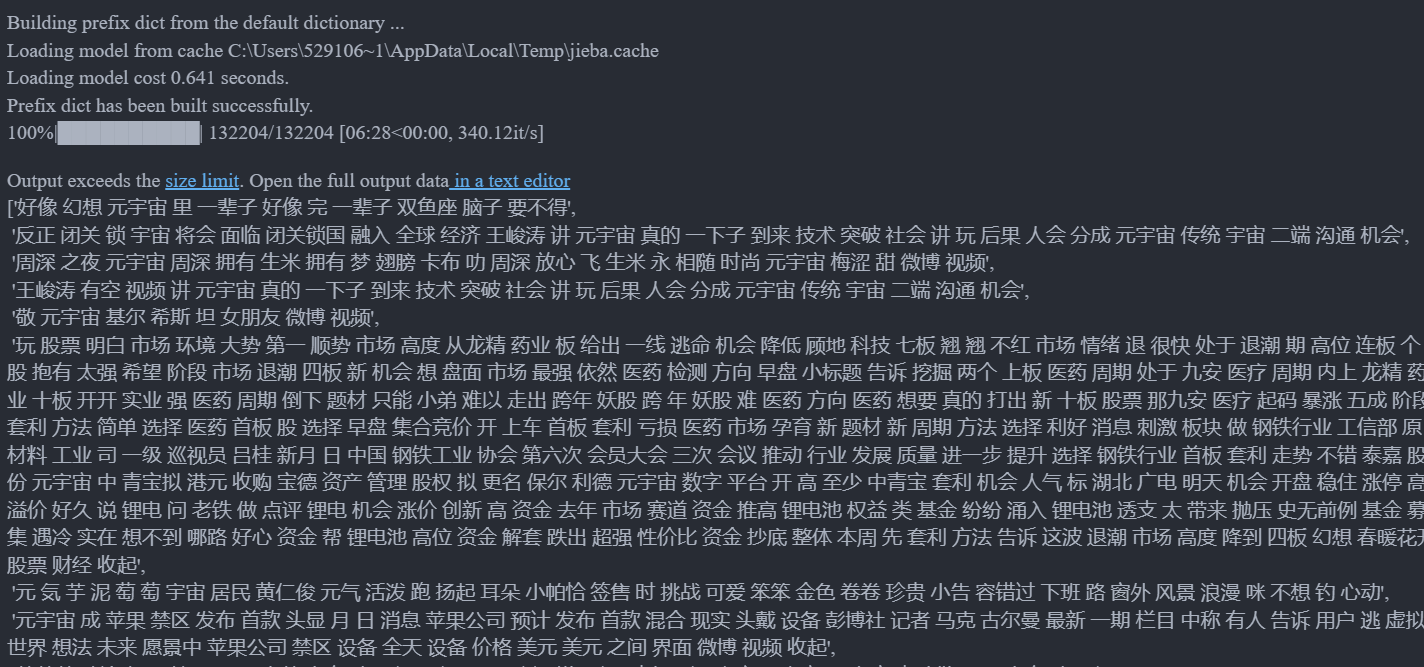
            l.append(word)

        tmp = ' '.join(l)

    seg\_data.append(tmp)

seg\_data

输出结果：



分词结果保存：

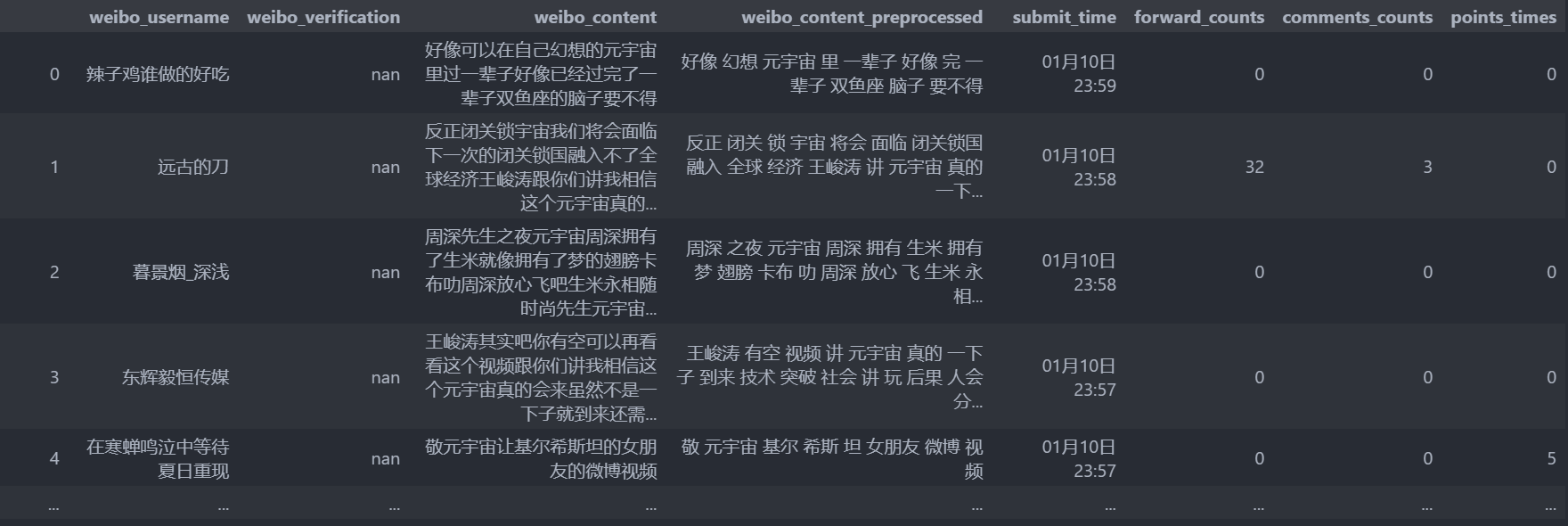
# raw\_dataframe['weibo\_content\_preprocessed'] = seg\_data

raw\_dataframe.insert(3, 'weibo\_content\_preprocessed', seg\_data)

raw\_dataframe.to\_csv('rawdata\_reprocessed.csv', index=0, encoding='utf-8')

raw\_dataframe

输出结果：



数据存储：

我们使用mysql进行数据存储，因为mysql功能强大、操作便捷，且容易和之后的Django进行对接

环境导入

import os

import pandas as pd

import warnings

from tqdm import tqdm

import pymysql

from sqlalchemy import create\_engine

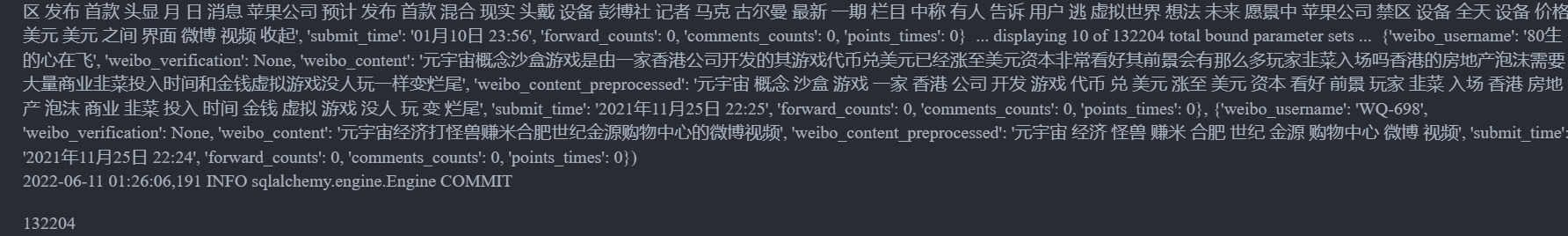
warnings.filterwarnings('ignore')

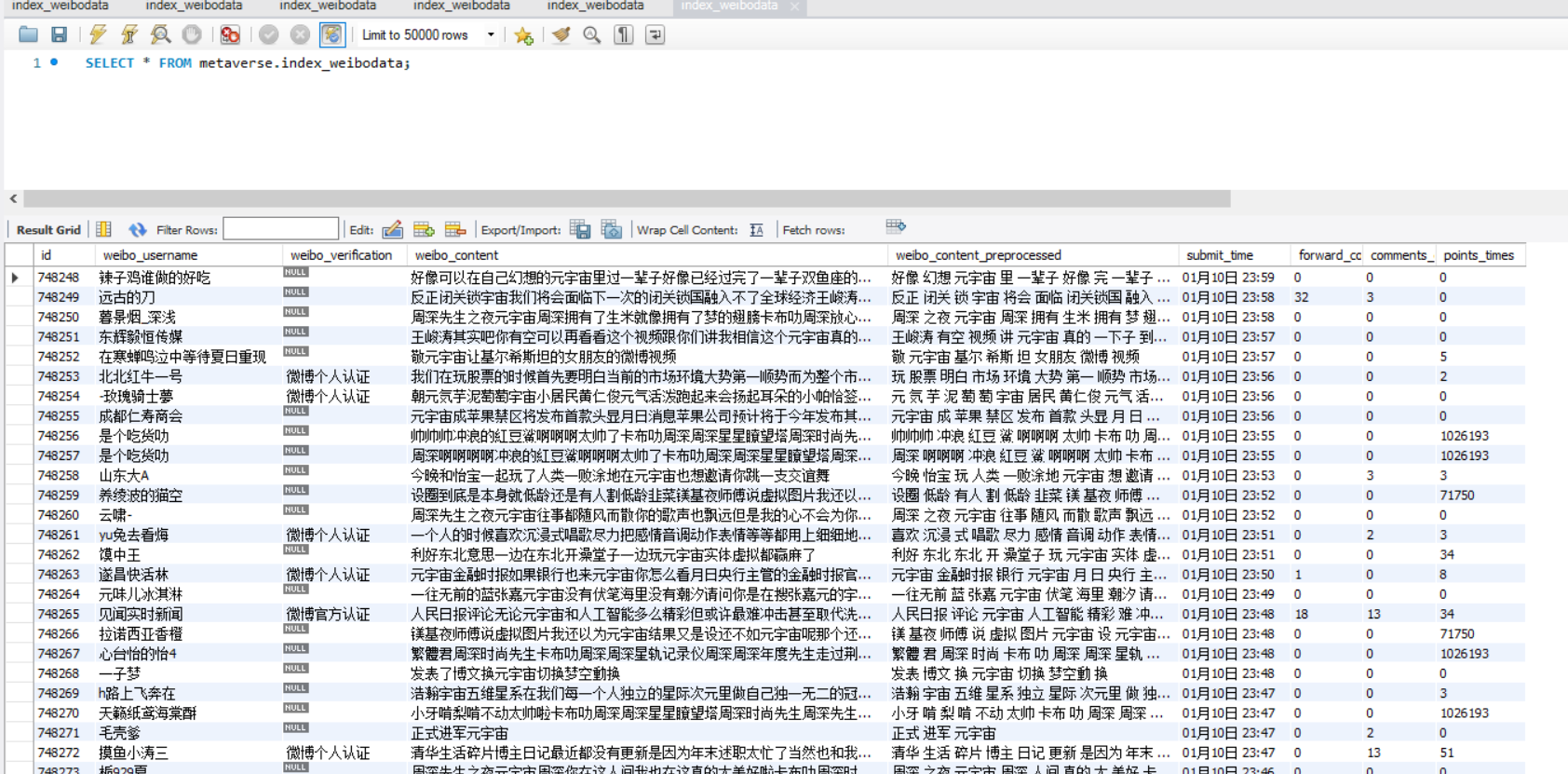
使用to\_sql将数据导入数据库

engine = create\_engine('mysql+pymysql://root:qq6666774@localhost:3306/metaverse', echo=True)

raw\_dataframe.to\_sql('index\_weibodata', engine, index=False, if\_exists='append')

输出结果：





### 词云图

该部分的处理流程是，将数据集中的“weibo\_content\_preprocessed”特征列利用jiaba的jieba.analyse. textrank函数进行关键词分析。但是由于textrank算法是对微博的文本进行分析，无法将博主认证信息、转评赞数据等纳入考虑范围，所以我们还对总体数据中每个词的词频进行了统计，并利用自己设计的评分算法对数据进行分析，提取出关键词。最后我们对两种方法得到的词云图结果进行了分析与对比。

一、使用jieba.analyse. textrank进行关键词分析

步骤1：导入必要的库：

import pandas as pd

import jieba.analyse

import jieba.posseg as psg

from os import path

import matplotlib.pyplot as plt

from wordcloud import WordCloud

步骤2：读入文件与查看：

raw\_df = pd.read\_csv(r'rawdata\_reprocessed.csv')

raw\_df.head()

输出：

图形用户界面, 文本

描述已自动生成

步骤3：使用jieba.analyse. textrank统计关键词：

我们设置了允许的词性为 副词、名词、地名、人名、非语素词(将元宇宙词等加入自定义词典之后，他们分词之后的词性为x)、动名词。这些词最能表现出数据的主题。

其中userdic.txt中的内容为：

文本

描述已自动生成

jieba.load\_userdict('userdic.txt')

text = ' '.join(map(str,raw\_df['weibo\_content\_preprocessed'].values))

keywords = jieba.analyse.textrank(text,topK=50,withWeight=True,allowPOS=['d','n','ns','nr','x','vn'])

print(keywords)

输出：

文本

描述已自动生成

步骤4：绘制词云图：

words\_frequence = {x[0]:x[1] for x in keywords}

backgroud\_Image=plt.imread(r'METAVERSE3.jpg')

# 若是有中文的话，font\_path ='simsun.ttc'必须添加，不然会出现方框，不出现汉字

# simsun.ttc为汉字编码文件，可以从本地windows系统找一个汉字编码文件上传， 如C:\\Windows\Fonts下有许多汉字编码文件

wordcloud = WordCloud(font\_path ='simsun.ttc',repeat=True,mask=backgroud\_Image,background\_color='white',width=2000,height=1000,colormap='cool')

wordcloud = wordcloud.fit\_words(words\_frequence)

wordcloud.to\_file('wordcloud\_textrank.png')

plt.imshow(wordcloud)

输出：

白板上的字

中度可信度描述已自动生成

二、进行词频统计与使用自定义评分算法进行关键词分析：

步骤1：导入必要的库

import pandas as pd

import jieba.analyse

import math

import numpy as np

import jieba.posseg as pseg

from tqdm import tqdm

from os import path

import matplotlib.pyplot as plt

from wordcloud import WordCloud

import jieba

读入数据与查看：数据与使用textrank进行关键词提取的数据相同

raw\_df = pd.read\_csv(r'rawdata\_reprocessed.csv')

raw\_df.head()

输出：

电脑屏幕的截图

描述已自动生成

步骤2：定义之后使用的函数：

(1)auth\_map函数用于根据用户的认证信息返回权重

def auth\_map(x):

if x == '微博个人认证':

return 3

elif x == '微博官方认证':

return 9

else:

return 1

(2)get\_score函数根据各种信息计算这条博文的权重，具体为：

认证：普通用户博文：1分，个人认证博文：3分，官方认证博文：9分

转评赞数据：ceil(1xlog(点赞数+1))+ceil(3xlog(评论数+1))+ceil(5xlog(转发数+1)) 分

认证信息与转评赞数据得分相加即为此条博文的权重。

这样设置的原因为：

1. 我们统计了数据中各种认证信息的数量，普通用户、个人认证用户、官方认证大概呈9：3：1的比例，故设置 普通用户博文：1分，个人认证博文：3分，官方认证博文：9分

文本

描述已自动生成

1. 我们查看了转评赞数据的平均值，发现转发与评论数据平均值相差不大，而点赞数据比他们俩高出了几个数量级。由于点赞数据获取相对容易，所以我们给他设置了很低的权重，而评论数据获取较为困难，出现评论说明博文已经有了一定的传播性，所以我们将其设置为为ceil(3xlog(评论数+1))（+1为了防止评论数为0出现错误）；而转发数据就意味着博文已经在大范围内传播，能极大增加博文内的单词的曝光度，所以我们给了他最高的权重。

文本

描述已自动生成

get\_score函数先计算每一项的分数，再将他们相加得到最终的分数：

def get\_score(df):

score1 = df['points\_times'].map(float).map(lambda x:math.ceil(math.log10(x+1)))

score2 = df['comments\_counts'].map(float).map(lambda x:3\*math.ceil(math.log10(x+1)))

score3 = df['forward\_counts'].map(float).map(lambda x:5\*math.ceil(math.log10(x+1)))

score4 = df['weibo\_verification'].map(auth\_map)

score = score1+score2+score3+score4

return score

(3)允许的词性与之前相同，也同样需要加载自定义的词典, pos\_filter函数用于过滤掉不在允许词性列表中的单词。

pos\_allow=['d','n','ns','nr','x','vn']

jieba.load\_userdict('./userdic.txt')

def pos\_filter(line:str,pos\_list):

words = pseg.cut(str(line))

ret\_words = []

for word,pos in words:

if word.strip() is not '' and pos in pos\_list :

ret\_words.append(word)

return ret\_words

步骤3：开始统计每个单词的词频以及单词的重要性:：

遍历每一条博文，计算其权重，过滤掉不符合要求的词性的词之后，博文中出现的单词获得与博文权重相同的分数。遍历完所有博文之后，单词的分数即为词的重要性，分数越高，说明此在重要的博文中出现的越多(出现在官方博文或者转评赞很多的博文)，重要性越高。

词频统计则对在出现次数dict中+1即可。

word\_count = dict()

keywords = dict()

contents = raw\_df['weibo\_content\_preprocessed'].values

for line,score in tqdm(zip(contents,scores),total=len(scores)):

# words = str(line).split(' ')

words = pos\_filter(line,pos\_allow)

for word in words:

if keywords.get(word) is None:

keywords[word] = score

else:

keywords[word] = keywords[word]+score

if word\_count.get(word) is None:

word\_count[word] = 1

else:

word\_count[word] = word\_count[word]+1

步骤4：对关键词按照得分降序排序，选取前50个，即得到重要性最高的前50个关键词：

sorted\_kw\_dic = sorted(keywords.items(),key=lambda x:x[1],reverse=True)

words\_frequence = dict()

for i in range(50):

words\_frequence[sorted\_kw\_dic[i][0]] = sorted\_kw\_dic[i][1]

print(sorted\_kw\_dic[i])

输出：

图片包含 表格

描述已自动生成

步骤5：绘制词云图对关键词进行可视化：

backgroud\_Image=plt.imread(r'./METAVERSE3.jpg')

# 若是有中文的话，font\_path ='simsun.ttc'必须添加，不然会出现方框，不出现汉字

# simsun.ttc为汉字编码文件，可以从本地windows系统找一个汉字编码文件上传， 如C:\\Windows\Fonts下有许多汉字编码文件

wordcloud = WordCloud(font\_path ='simsun.ttc',repeat=True,mask=backgroud\_Image,background\_color='white',width=2000,height=1000,colormap='cool')

wordcloud = wordcloud.fit\_words(words\_frequence)

wordcloud.to\_file('wordcloud\_myscoring.png')

plt.imshow(wordcloud)

输出：

白板上的字

中度可信度描述已自动生成

步骤6：对词频字典按照出现次数降序进行排序，选取频次最高的30个词进行可视化：

print(len(word\_count))

sorted\_wc\_dic = sorted(word\_count.items(),key=lambda x:x[1],reverse=True)

x\_word = []

y\_count = []

for i in range(30):

x\_word.append(sorted\_wc\_dic[i][0])

y\_count.append(sorted\_wc\_dic[i][1])

print(sorted\_wc\_dic[i])

输出：

文本

描述已自动生成

绘制柱状图对词频数据进行可视化：

def autolabel(rects):

for rect in rects:

height = rect.get\_height()

plt.text(rect.get\_x()+rect.get\_width()/2.-0.35, 1.01\*height, '%s' % int(height))

plt.figure(figsize=(20,10))

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号 #有中文出现的情况，需要u'内容'

rects = plt.bar(x\_word,y\_count,width=0.5,color="c")

plt.xlabel("word")

plt.ylabel("frequency")

plt.yscale('symlog')

plt.ylim(10000,250000)

autolabel(rects)

输出：

图表, 直方图

描述已自动生成

从词频统计柱状图中可以看出，‘元宇宙’的词频一骑绝尘，达到了20多万次，是第二多的‘数字’的约六倍，这可能是因为我们爬取微博时设置的关键词即为‘元宇宙’，这样每一条微博中都包含了至少一个‘元宇宙’，使得‘元宇宙’的词频最大。随后出现次数较多的‘数字’、‘板块’、‘市场’等也与元宇宙相关，从中可以看出元宇宙的金融属性较强，科技含量也比较高。

三、textrank与自定义重要性得到的词云图分析对比：

Textrank得到的词云图：

白板上的字

中度可信度描述已自动生成

自定义重要性得到的词云图：

白板上的字

中度可信度描述已自动生成

在词云图中二者共有的关键词几乎相同：元宇宙、数字、板块、公司、科技、游戏、世界、概念、股份、投资等等都是二者共有的，他们大多数也都出现在了词频统计数量较多的榜单之中，可见词频高的词与关键词有很强的相关性。

词云图中的关键词如数字、科技、链接等时数据的主题有了很强的科技色彩。这也符合元宇宙是利用科技手段进行链接与创造，与现实世界映射与交互的虚拟世界的定义。而概念、世界、游戏、现实等也暗暗透露出元宇宙现在还只停留在概念层面、 并未真正的实现，游戏可能是现阶段的一个发展方向。板块、公司、投资、股票、指数等词则将视线转向金融领域，似乎元宇宙的出现让资本市场掀起了一股浪潮。

### 聚类

在选择算法之前，我们先进行了小部分数据的测试，发现Kmeans聚类的效果不如LDA模型训练出来的效果好。

因此在该部分，实现了两种分类算法，其一是LDA，另一是BTM

LDA处理流程：中文分词、去停用词、单词向量化、LDA模型主题提取

1. LDA算法:

步骤1：导入依赖：

import **os**

import **pandas** as **pd**

import **re**

import **jieba**

import **jieba**.**posseg** as **psg**

from **sklearn**.**feature\_extraction**.**text** import **TfidfVectorizer**, **CountVectorizer**

from **sklearn**.**decomposition** import **LatentDirichletAllocation**

import **numpy** as **np**

import **pyLDAvis**

import **pyLDAvis**.**sklearn**

from **wordcloud** import **WordCloud**

import **matplotlib**.**pyplot** as **plt**

步骤2：导入数据：

# 路径的设置和文件读取

output\_path = r'Output'

file\_path = r'Data\rawdata\_reprocessed.csv'

stop\_file = r'Data\stop\_words.txt'

**jieba**.load\_userdict(r'Data\userdic.txt')

data = **pd**.**read\_csv**(file\_path).**astype**(**str**)

tfidf\_data = data.**copy**()

count\_data = data.**copy**()

data.**head**()

将数据进行LDA训练：

# 定义中文分词函数

def **chinese\_word\_cut**(mytext):

    try:

        stopword\_list = **open**(stop\_file,encoding ='utf-8')

    except:

        stopword\_list = []

**print**("error in stop\_file")

    stop\_list = []

    flag\_list = ['n','nz','vn']

    for line in stopword\_list:

        line = **re**.**sub**(u'\n|\\r', '', line)

        stop\_list.**append**(line)

    word\_list = []

    #jieba分词

    seg\_list = **jieba**.cut(mytext)

    for seg\_word in seg\_list:

        # 只保留中文

        word = **re**.**sub**(u'[^\u4e00-\u9fa5]','',seg\_word.word)

        find = 0

        for stop\_word in stop\_list:

            if stop\_word == word or **len**(word)<2:     #this word is stopword

                    find = 1

                    break

        if find == 0 and seg\_word.flag in flag\_list:

            word\_list.**append**(word)

    return (" ").**join**(word\_list)

data["content"] = data.weibo\_content.apply(**chinese\_word\_cut**)

data.**to\_csv**('weibi\_data\_keywords.csv')

# 定义打印主题词函数

def **print\_top\_words**(model, feature\_names, n\_top\_words):

    tword = []

    for topic\_idx, topic in **enumerate**(model.components\_):

        # print("Topic #%d:" % topic\_idx)

        topic\_w = " ".**join**([feature\_names[i] for i in topic.argsort()[:-n\_top\_words - 1:-1]])

        tword.**append**(topic\_w)

**print**(topic\_w)

    return tword

#提取1000个特征词语

n\_features = 1000

# 根据单词出现的次数向量化

count\_tf\_vectorizer = CountVectorizer(strip\_accents = 'unicode',max\_features=n\_features,stop\_words='english',max\_df = 0.5,min\_df = 10)

# 根据TFIDF算法向量化

tfidf\_tf\_vectorizer = TfidfVectorizer(strip\_accents = 'unicode',max\_features=n\_features,stop\_words='english',max\_df = 0.5,min\_df = 10)

count\_tf = count\_tf\_vectorizer.fit\_transform(data.content)

tfidf\_tf = tfidf\_tf\_vectorizer.fit\_transform(data.content)

n\_topics = 5

count\_lda = LatentDirichletAllocation(n\_components=n\_topics, max\_iter=50,learning\_method='batch',learning\_offset=50,random\_state=0)

tfidf\_lda = LatentDirichletAllocation(n\_components=n\_topics, max\_iter=50,learning\_method='batch',learning\_offset=50,random\_state=0)

# ##### 以下是训练代码，运行时间较久

count\_lda.fit(count\_tf)

tfidf\_lda.fit(tfidf\_tf)

# 输出每个主题对应词语

n\_top\_words = 25

count\_tf\_feature\_names = count\_tf\_vectorizer.get\_feature\_names()

count\_topic\_word = **print\_top\_words**(count\_lda, count\_tf\_feature\_names, n\_top\_words)

tfidf\_tf\_feature\_names = tfidf\_tf\_vectorizer.get\_feature\_names()

tfidf\_topic\_word = **print\_top\_words**(tfidf\_lda, tfidf\_tf\_feature\_names, n\_top\_words)

# 输出每个文本对应的主题

count\_topics=count\_lda.transform(count\_tf)

tfidf\_topics=tfidf\_lda.transform(tfidf\_tf)

count\_topic = []

for t in count\_topics:

    count\_topic.**append**("Topic #"+**str**(**list**(t).**index**(**np**.**max**(t))))

count\_data['概率最大的主题序号']=count\_topic

count\_data['每个主题对应概率']=**list**(count\_topics)

count\_data.**to\_excel**(output\_path+"data\_topic\_count.xlsx",index=False)

tfidf\_topic = []

for t in tfidf\_topics:

    tfidf\_topic.**append**("Topic #"+**str**(**list**(t).**index**(**np**.**max**(t))))

tfidf\_data['概率最大的主题序号']=tfidf\_topic

tfidf\_data['每个主题对应概率']=**list**(tfidf\_topics)

tfidf\_data.**to\_excel**(output\_path+"data\_topic\_tfidf.xlsx",index=False)

**pyLDAvis**.enable\_notebook()

pic = **pyLDAvis**.sklearn.prepare(count\_lda, count\_tf, count\_tf\_vectorizer)

**pyLDAvis**.display(pic)

**pyLDAvis**.save\_html(pic, output\_path+'count\_lda\_pass'+**str**(n\_topics)+'.html')

**pyLDAvis**.display(pic)

**pyLDAvis**.enable\_notebook()

pic = **pyLDAvis**.sklearn.prepare(tfidf\_lda, tfidf\_tf, tfidf\_tf\_vectorizer)

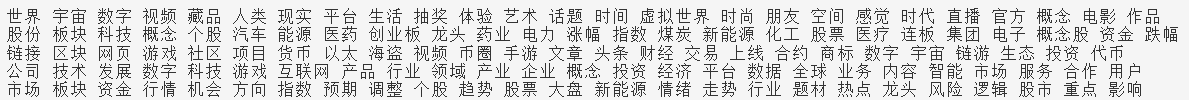
**pyLDAvis**.display(pic)

**pyLDAvis**.save\_html(pic, output\_path+'tfidf\_lda\_pass'+**str**(n\_topics)+'.html')

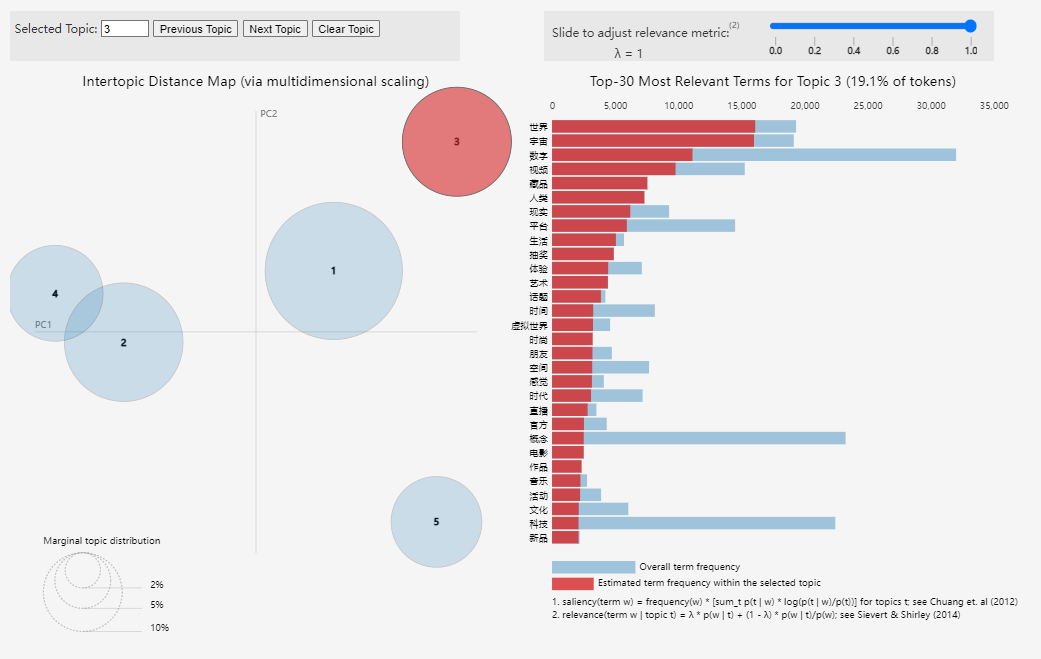
**pyLDAvis**.display(pic)

结果显示：

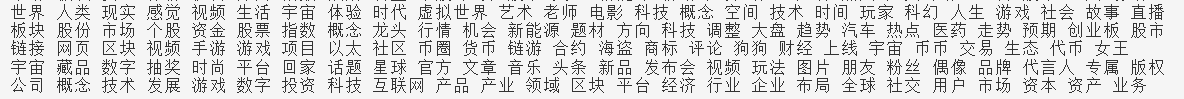
使用词频向量化的结果：

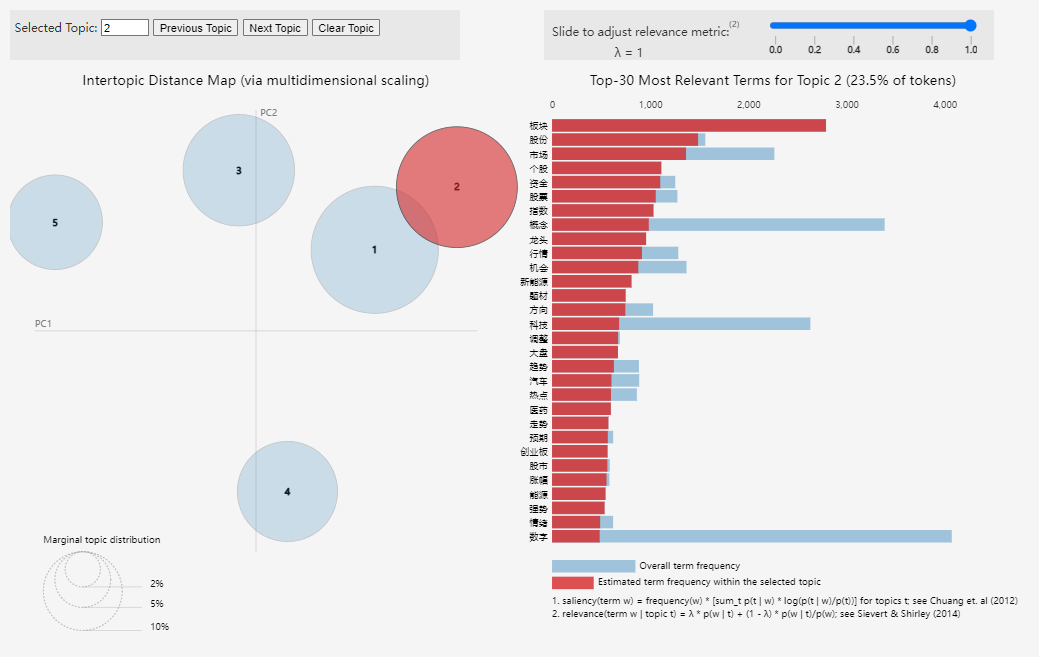


可以看到各个主题的侧重点各有不同，但每个主题之间仍有重复词，猜测是因为原始数据的相关性本来就较高导致的。



使用TFIDF向量化的结果：





可以看到两个向量化方法所聚出的5个类别分配都比较均匀，均在20%左右，且从坐标图来看，各主题之间的重合度是相对较低的，由于我们微博是短文本，故从理论上以及以往实验的经验，结合本次实验的结果，可以认为在此任务中tfidf的效果要更好一些，接下来就是对tfidf向量化的LDA聚类结果进行分析：

可视化工具使用的是pyLDAvis库

左侧显示的是类别映射到二维空间中的坐标，右侧则是该类中排名前30的主题词汇

右上可以选择阈值，来筛选那些和该类相关度在给定范围内的词（即该词出现的文本被分到该类的比例）由于本次实验的数据是根据同一关键词爬取的，因此我们有理由认为某类独有的词才更能反映该类的含义（因为大方向都是与元宇宙相关）

结果分析：

第一类：从互联网、技术、风口、布局等词可以看出词类主题是对元宇宙的出现给互联网行业带来的影响

第二类：股份、股票、指数、股市等词可以看出应该是元宇宙对于股市的冲击和影响

第三类：人类、显示、生活、艺术、电影、人生等主题词都是与人的生活习习相关，故此类应该是元宇宙对于人们生活的影响或人们对元宇宙的畅想

第四类：第四类的关键词教混杂，猜测应该是一些误爬的数据或者与元宇宙相关不大的数据，因为与其他主题毫不相关，因此被划分到了同一类

第五类：也是有很多毫不相关的词，但不难从排名第三的词汇抽奖得知，这个主题很可能是人们转发的抽奖信息。

1. BTM

BTM和LDA一样，均为提取文本主题的模型，他与LDA的主要差别在于，LDA算法使用单词进行训练而BTM算法使用单词两两组合生成的Biterm进行训练；这也使得BTM算法比LDA算法更适合短文本的主题聚类分析，因为两两组合Biterm能使训练的材料大大增多。

通过设定主题数，BTM算法返回数据集中每一个主题的占比以及各个主题中的关键词。

由于代码组件过多，详情请移步提交材料的BTM代码部分。

步骤1：导入必要的库：

import time

from Model import \*

import sys

import indexDocs

import topicDisplay

import os

步骤2：进行函数的定义：

def BTM(argvs):

if(len(argvs)<4):

usage()

else:

if (argvs[0] == "est"):

K = argvs[1]

W = argvs[2]

alpha = argvs[3]

beta = argvs[4]

n\_iter = argvs[5]

save\_step = argvs[6]

docs\_pt = argvs[7]

dir = argvs[8]

print("===== Run BTM, K="+str(K)+", W="+str(W)+", alpha="+str(alpha)+", beta="+str(beta)+", n\_iter="+str(n\_iter)+", save\_step="+str(save\_step)+"=====")

clock\_start = time.time()

model = Model(K, W, alpha, beta, n\_iter, save\_step)

model.run(docs\_pt,dir)

clock\_end = time.time()

print("procedure time : "+str(clock\_end-clock\_start))

else:

usage()

步骤3：进行BTM训练：

def main():

mode = "est"

K = 5 #主题数

alpha = 0.5 #控制主题生成的参数

beta = 0.5 #控制Biterm生成的参数

n\_iter = 15 #迭代次数

save\_step = 100

W = None

# pycharm

output\_dir = "../output/"

input\_dir = "../sample-data/"

model\_dir = output\_dir + "model/" # 模型存放的文件夹

voca\_pt = output\_dir + "voca.txt" # 生成的词典

dwid\_pt = output\_dir + "doc\_wids.txt" # 每篇文档由对应的序号单词组成

# doc\_pt = input\_dir + "test.dat" #输入的文档

doc\_pt = input\_dir + "weibo\_content\_preprocessed.txt" # 输入的文档

print("=============== Index Docs =============")

# W生成的词典

W = indexDocs.run\_indexDocs(['indexDocs', doc\_pt, dwid\_pt, voca\_pt])

print("W : " + str(W))

argvs = []

argvs.append(mode)

argvs.append(K)

argvs.append(W)

argvs.append(alpha)

argvs.append(beta)

argvs.append(n\_iter)

argvs.append(save\_step)

argvs.append(dwid\_pt)

argvs.append(model\_dir)

argvs.append(voca\_pt)

print("=============== Topic Learning =============")

BTM(argvs)

print("================ Topic Display =============")

topicDisplay.run\_topicDicplay(['topicDisplay', model\_dir, K, voca\_pt])

topicDisplay.perplexity(argvs)

if \_\_name\_\_ == "\_\_main\_\_":

main()

其中训练使用的weibo\_content\_preprocessed.txt中的内容为：

图片包含 日历

描述已自动生成

BTM输出：

图片包含 散点图

描述已自动生成

在BTM算法得到的结果中，主题1占据了大部分份额，达到了72.3%，主题1的关键词聚焦在科技金融领域，其余四个并无可明显总结出的主题，聚类效果一般；且使用BTM聚类的结果并不好进行可视化，所以我们放弃了这个聚类方案。

算法选择依据：最终，根据聚类的分布，聚类效果等综合评定，选定的算法是使用tfidf进行单词向量化的LDA算法。

### 情感分析

在该部分，处理的流程：直接使用Snownlp库进行情感分析

词性标注：（简单演示）

wb = data[0:4]['weibo\_content']

for s in wb:

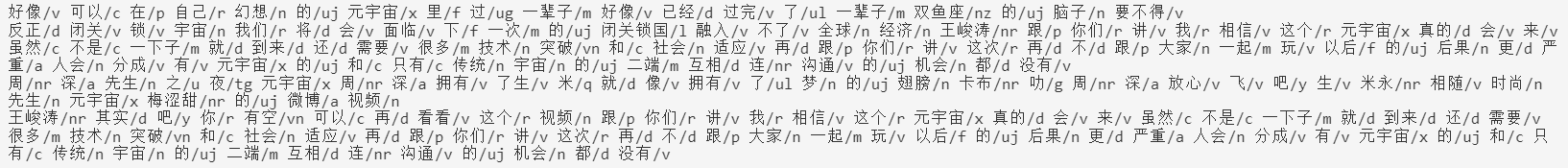
    wordLists = psg.cut(s)

    for w in wordLists:

        print(w,end=' ')

    print()

标注结果：



利用SnowNLP库进行情感分析：

comments = data['weibo\_content']

sentimentlist = []

for comment in comments:

    s = SnowNLP(comment)

    sentimentlist.append(s.sentiments)

#    print(s.sentiments)

#可视化结果

plt.hist(sentimentlist, bins=np.arange(0,1,0.01),facecolor='b')

plt.xlabel('Probability')

plt.ylabel('Quantity')

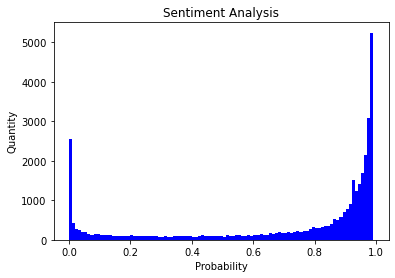
plt.title('Sentiment Analysis')

plt.show()

将情感分布以图展示：

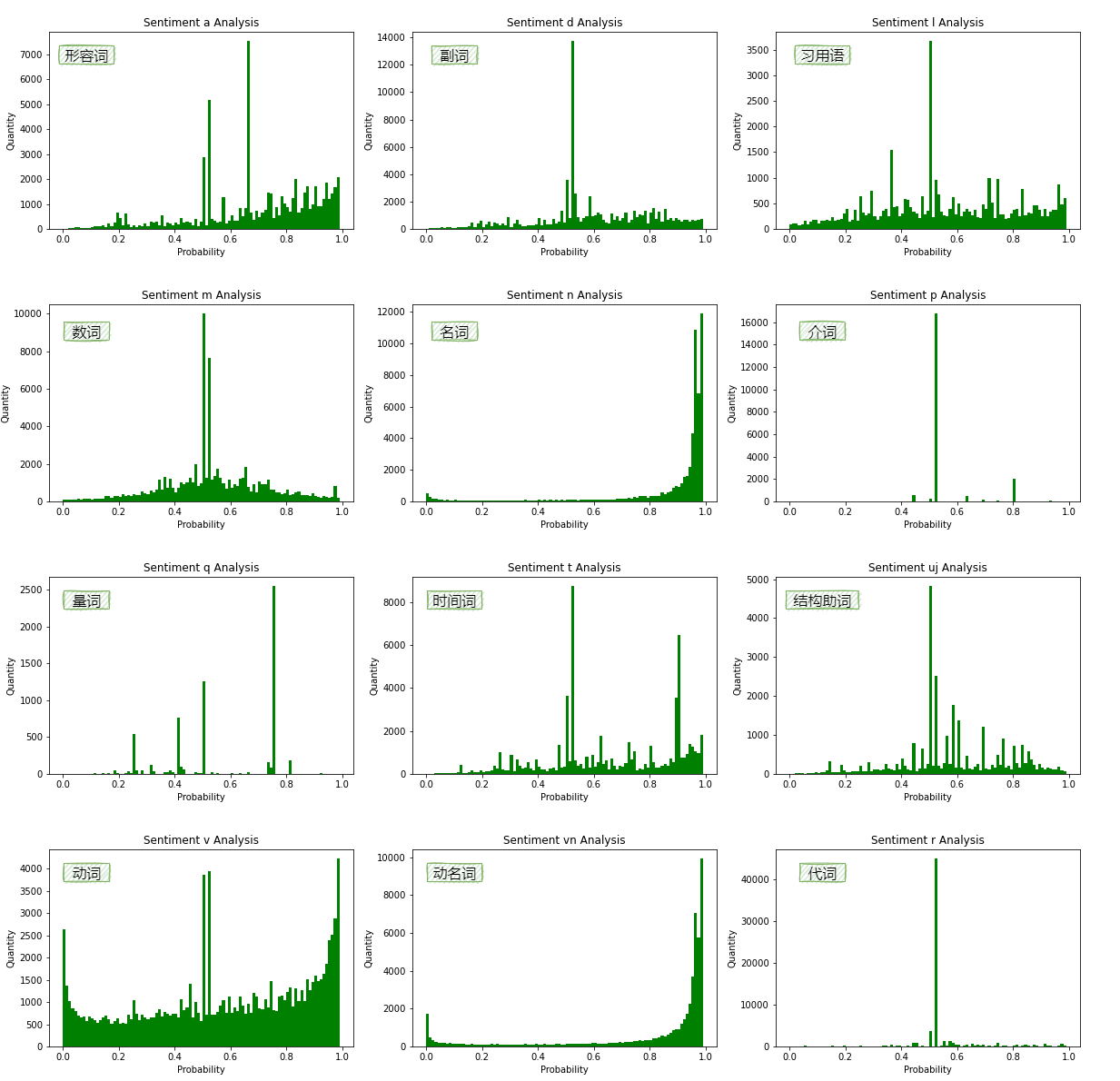
整体的情感分析结果图：

可以看到该话题下大多数微博都是正向的，也有不少微博是负面的，感情色彩趋于两极化。推测是元宇宙话题下大家大多都进行了对元宇宙的畅想，因此中性微博较少

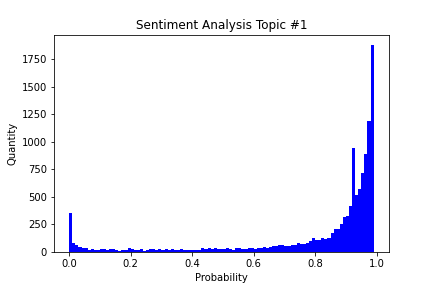


不同词性的情感分布

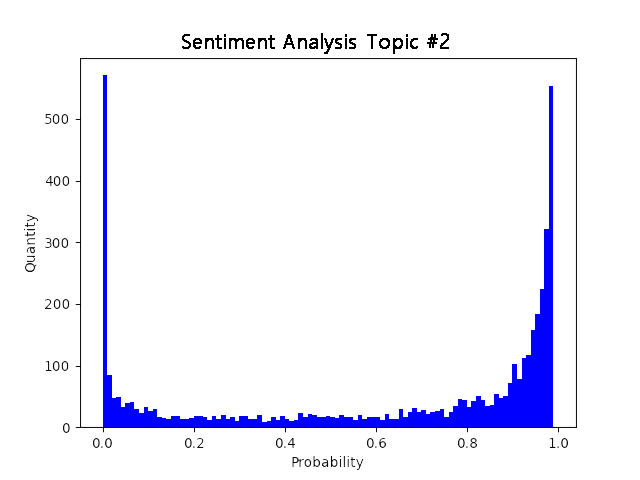
分别将句子中的每个词性提取出来进行单独的情感分析，发现



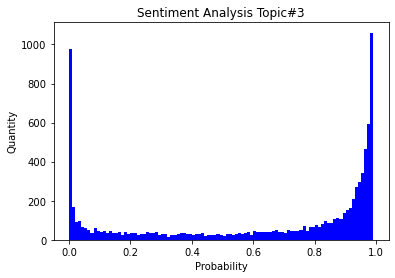
根据聚类结果进行情感分析：



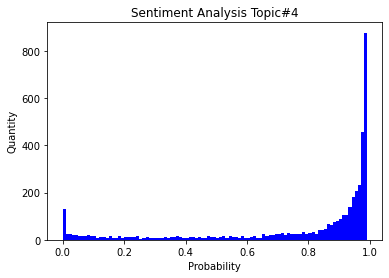
主题一下的微博情感色彩更加积极一些，应该是认为元宇宙对互联网行业的影响是乐观的



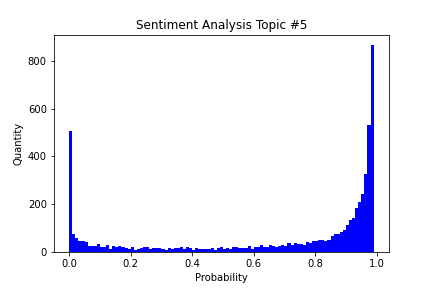
主题二的情感则是两极分化，且积极和消极数量近似，说明元宇宙给股市带来的影响有好有坏



主题三的情感与主题二类似，喜忧参半，说明人们对元宇宙改变生活这一看法褒贬不一



主题四是虚拟货币相关，推测元宇宙将虚拟货币带上了另一个高潮



主题五应该是与抽奖相关的数据，比较杂乱（或者说是前四类聚剩下的），故情感分析与整体的情感分析非常相似

选择snownlp的原因：

Snownlp是在电商评论的数据集上训练出来的，电商评论和微博相似度比较高，因此可以使用Snownlp对微博进行情感分析

### 知识抽取和可视化

在该部分我们实现了文本的命名实体识别和实体关系抽取以及它们的可视化。

命名实体识别部分我们采用的hanlp和基于BiLSTM等模型实现的jiagu工具，关系提取部分我们就用的是jiagu工具。前端用的html页面显示。

知识抽取代码：

from django.http import HttpResponse

from django.shortcuts import render

import jiagu

from pyhanlp import \*

import os

# 接收请求数据

def getText(request):

    request.encoding='utf-8'

    ctx ={}

    text = request.POST['q']

    ctx['input'] =text

    #hanlp 命名实体识别

    segment = HanLP.newSegment()

    hanlp = segment.seg(text)

    print(hanlp)

    hanlpNer = {}

    hPname = []

    hLname = []

    hIname = []

    for i in hanlp:

        strN = str(i)

        res = strN.split("/")

        hanlpNer[res[0]] = res[1]

    for key in hanlpNer:

        temp = hanlpNer.get(key)

        if temp.startswith('nr'):

            hPname.append(key)

        if temp.startswith('ns'):

            hLname.append(key)

        if temp.startswith('ni') or temp.startswith('nt'):

            hIname.append(key)

    ctx['hPname'] = hPname

    ctx['hLname'] = hLname

    ctx['hIname'] = hIname

    #jiagu

    words = jiagu.seg(text) # 分词

    print(words)

    pos = jiagu.pos(words) # 词性标注

    print(pos)

    ner = jiagu.ner(words) # 命名实体识别

    print(ner)

    knowledge = jiagu.knowledge(text) #关系抽取

    print(knowledge)

    relationExtraction = []

    for i in knowledge:

        strR = ''

        strR = strR+str(i[0])+'的'+str(i[1])+'是'+str(i[2])

        relationExtraction.append(strR)

    dictionary = dict(zip(words,ner))

    Pname = []

    Lname = []

    Iname = []

    for key in dictionary:

        if dictionary.get(key) == 'B-PER' or dictionary.get(key) == 'B-IER':

            Pname.append(key)

        if dictionary.get(key) == 'B-LOC' or dictionary.get(key) == 'I-LOC':

            Lname.append(key)

        if dictionary.get(key) == 'B-ORG' or dictionary.get(key) == 'I-ORG':

            Iname.append(key)

    ctx['Pname'] = Pname

    ctx['Lname'] = Lname

    ctx['Iname'] = Iname

    if request.POST:

        ctx['text'] = request.POST['q']

    ctx['views\_list'] =  relationExtraction

    return render(request, "get\_form.html", ctx)

前端界面代码：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>命名实体识别和关系抽取</title>

<style>

    body{background:#CCC;}

    div{background:#FFF;

        color:#09F;

        padding:15px;

        margin:60px auto 0px;

        width:600px;

        font-family:"微软雅黑";

        box-shadow:10px 10px 15px black  ;}

    h2{text-align:center;}

    hr{width:80%;

        border:1px solid  #09F ;

        margin-top:-5px;

    }

    input{margin:10px}

    li{margin:10px;}

    input[type="text"] ,input[type="tel"]

    { width:100px;

        border:0px;

        border-bottom:1px solid;    }

    button{background:#09F;

        color:white;

        width:100px;

        height:40px;

        font-size:16px;

        font-weight:bold;

        font-family:"微软雅黑";

        margin:5% 39%;}

    button:hover{background:#0CF;}

</style>

</head>

<body>

    <div>

    <h2>命名实体识别和关系抽取</h2>

    <form action="/getText/" method="post">

        {% csrf\_token %}

        <label>文档：</label><input type="text" name="q" value="{{ input }}">

        <input type="submit" value="提交">

    </form>

    <label>文档内容：</label><p>{{ text }}</p>

    <label>命名实体识别：</label>

    <ul>

        <label>人名：</label>

        <li>jiagu:

            {% if Pname %}

            {% for Pn in Pname %}

            <a>{{ Pn }}</a>

            {% endfor %}

            {% else %}

            无

            {% endif %}

        </li>

        <li>hanlp:

            {% if hPname %}

            {% for Pn in hPname %}

            <a>{{ Pn }}</a>

            {% endfor %}

            {% else %}

            无

            {% endif %}

        </li>

        <label>地名：</label>

        <li>jiagu:

            {% if Lname %}

            {% for Pn in Lname %}

            <a>{{ Pn }}</a>

            {% endfor %}

            {% else %}

            无

            {% endif %}

        </li>

        <li>hanlp:

            {% if hLname %}

            {% for Pn in hLname %}

            <a>{{ Pn }}</a>

            {% endfor %}

            {% else %}

            无

            {% endif %}

        </li>

        <label>机构名：</label>

        <li>jiagu:

            {% if Iname %}

            {% for Pn in Iname %}

            <a>{{ Pn }}</a>

            {% endfor %}

            {% else %}

            无

            {% endif %}

        </li>

        <li>hanlp:

            {% if hIname %}

            {% for Pn in hIname %}

            <a>{{ Pn }}</a>

            {% endfor %}

            {% else %}

            无

            {% endif %}

        </li>

    </ul>

    <label>关系抽取：</label>

    <ul>

        {% for view in views\_list %}

            <li>{{ view }}</li>

        {% endfor %}

    </ul>

    </div>

</body>

</html>

运行演示：





分析：我们可以从结果上看到，hanlp命名实体识别无法将中职联公司识别为机构名，而jiagu工具的命名实体识别中的地名识别不好，无法将一些区和镇划分为地名。关系抽取结果则基本能将句子的大概意思表达出来。 jiagu是用的ownthink开源的知识图谱训练数据，目前只能使用百科的描述进行测试。其他的诸如OpenNRE、DeepKE等工具在虚拟机上部署的，所以这边没有做出效果对比图。

前端部分

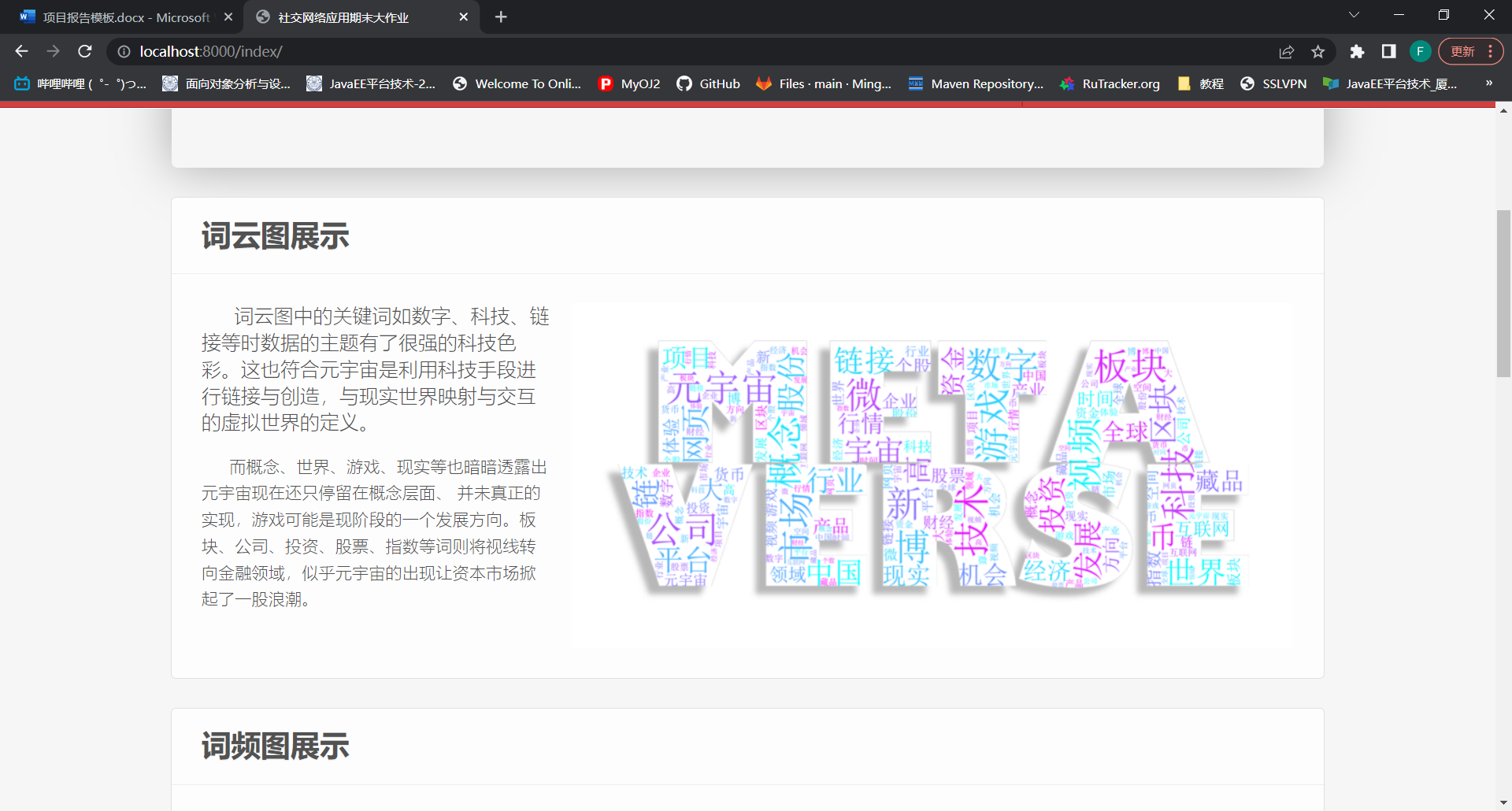
前端使用html+css+js+django搭建，同时搭配mysql进行使用

（注：如果需要运行该项目，需安装mysql，然后更改settings.py中的数据库配置，比如用户名、密码等，之后需要使用django进行migrate建表，然后在建立好的数据库（metaverse）的index\_weibodata表中的weibo\_content属性后新增属性weibo\_content\_reprocessed，格式为LONGTEXT。之后运行代码中的read3.ipynb进行数据导入（read3.ipynb中需要修改create\_engine方法中的数据库连接字符串）。准备工作完成后，runserver启动服务器，然后使用localhost:8000/index进入主页）

主页（点击不同按钮可进行跳转）



前端展示词云图



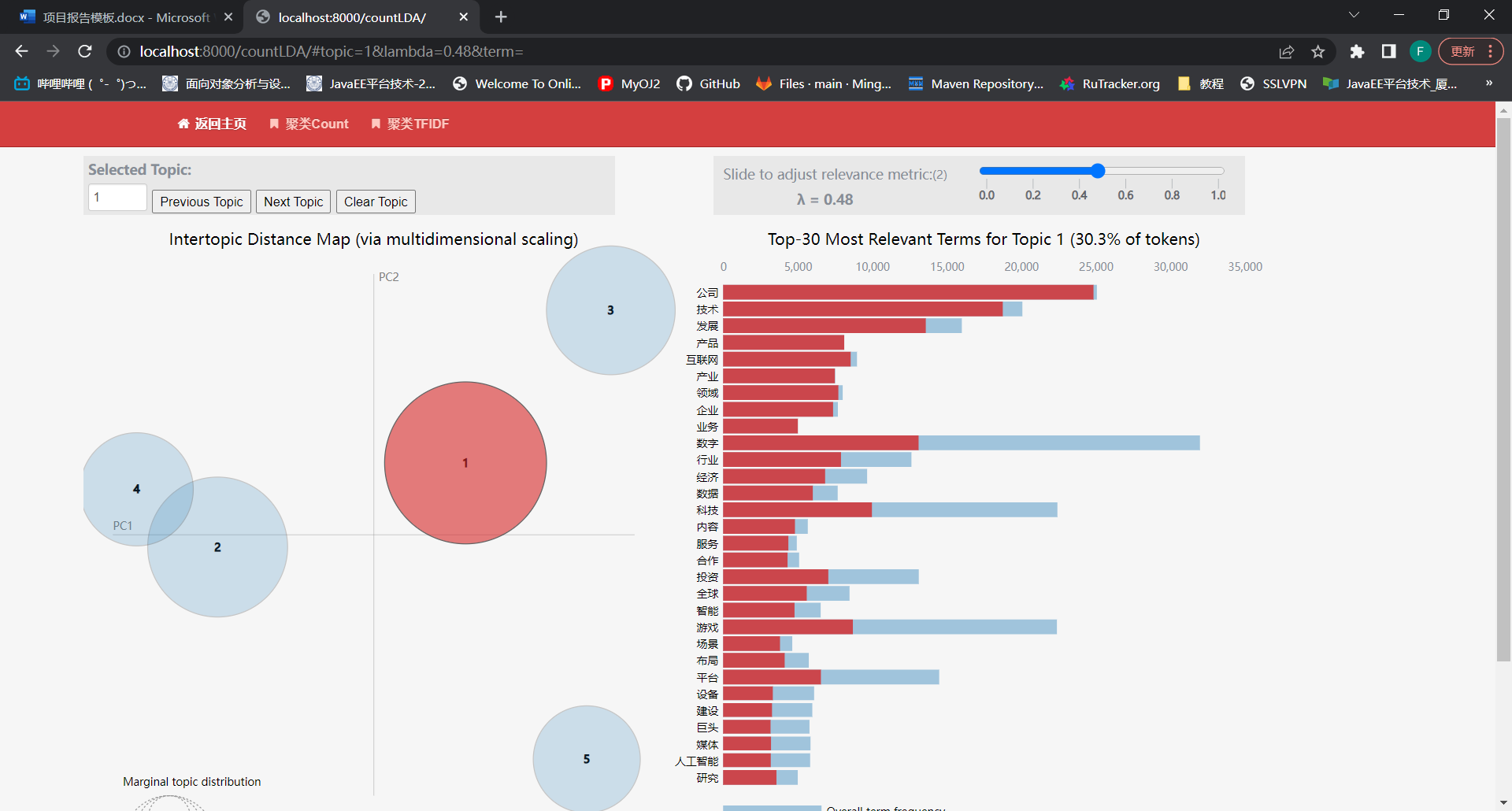
前端展示词频图



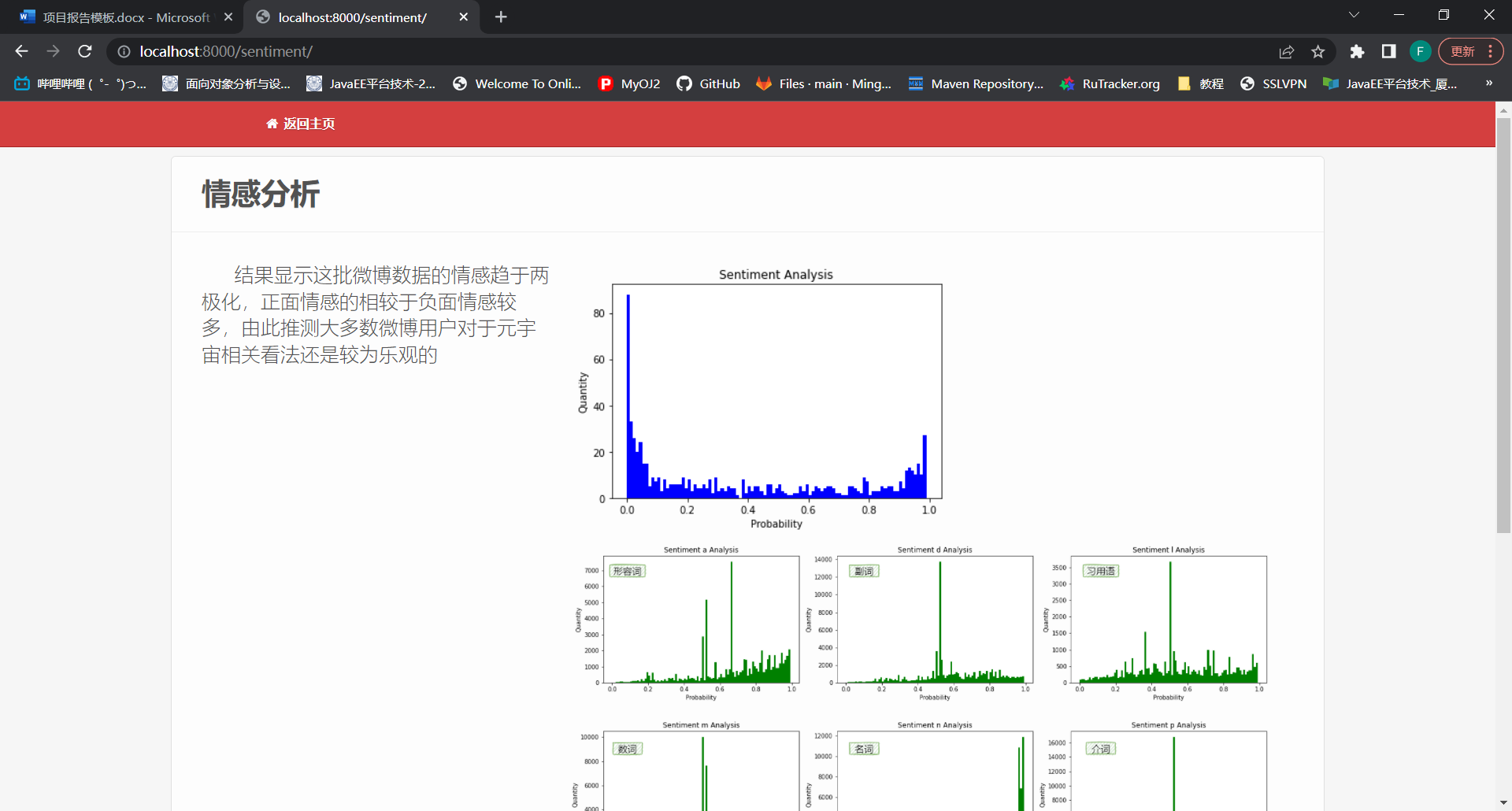
前端展示部分微博数据（可分属性排序、搜索、显示缩略微博内容、设置每页展示数量）



前端展示聚类效果



前端展示情感分析



前端展示知识抽取

* 
* 

## 任务分工

请描述清楚各自的分工和完成情况

谢健祥：负责关键词提取分析、词云图绘制、利用BTM算法进行聚类分析

袁佳哲：负责数据爬取、数据预处理、数据存储、前端可视化

徐荪睿：负责数据爬取、使用LDA算法进行聚类分析、情感分析

程昊天：负责数据爬取、知识抽取及其可视化

请每个成员各自描述实现自己任务过程种遇到的难点和解决方法

谢健祥： 在关键词提取的时候微博的权重应充分考虑认证信息与转评赞数据等，各个信息之间的权重不好平衡，经过数据的统计分析与少部分样本的测试之后得到各个部分合适的权重；在进行BTM分析时，用于数据量大，模型运行时间非常长，而且经常因电脑配置不行而崩溃，最终只采样了约20000条数据的分析，作为参考。

袁佳哲： 数据存储一开始尝试使用Django自带的sqlite，但是使用起来非常不方便，主要是没法查看数据，所以选择使用mysql存储。原本的数据导入尝试使用先导出为csv，然后使用cmd命令操作mysql进行导入，但因为数据量过大，会导致cmd直接崩溃。之后发现dataframe自带to\_sql功能，只需进行相关配置即可。

程昊天： 本来想用Bert-BiLSTM-CRF命名实体识别模型，来训练相关数据来进行预测，但由于电脑配置不行，所以直接采用现成的工具。后面的关系抽取也是直接采用jiagu的工具。

徐荪睿： 聚类结果中不同的主题之间会有重复的词语，相当影响对聚类结果的分析，使用pyLDAvis可视化库进行可视化后，适当调节词语和主题的相关度阈值后重复的词语减少了许多。

本想使用困惑度分析来对聚类结果进行一个评价，但结果非常不尽如人意，基本上是乘上升的趋势，分的类数越多，效果越差，因此放弃困惑度这个度量，而直接根据实验结果进行分析评价。

请阐述项目过程中遇到的团队协作问题和解决方法

因需要的数据量过大，且微博有反爬机制，所以一台电脑难以胜任爬取10万条数据的任务。因此我们选择将选用的时间段分为四段，在四台电脑上同时进行。之后将爬取的数据进行收集，再使用代码进行汇总。