

B站历史弹幕词频统计及情感分析

小组成员

杨奎、徐小龙、张一锋、付文韬、杨采杭

成员分工

成员	分工
杨奎	1.爬取B站历史弹幕，生成数据集； 2.整合并修改PPT内容，撰写演讲稿，讲解PPT； 3.整合并补充项目报告的内容。
徐小龙	1.对收集到的数据集进行词频统计、情感分析及可视化，并且将模型部署到hadoop + spark 平台上运行； 2.撰写项目报告的具体实现过程部分。
张一锋	1.查找资料，辅助参与技术实现； 2.制作PPT。
付文韬	1.查找资料，辅助参与技术实现； 2.制作PPT。
杨采杭	测试模型

具体实现过程

1 爬取B站视频弹幕

(1) 爬取网页源码

A 获取BVID

`https://www.bilibili.com/video/BV1bR4y1f7P2/?
vd_source=676aec372d3f6bb58e9057eed3ee9cd4`

视频的 bvid 即 url 上的 BV1bR4y1f7P2

B 设置请求头

伪装成正常的HTTP请求

注意: SESSDATA 与 document.cookie 会时常更新

```
# F12-应用-Cookie-http://www.bilibili.com-SESSDATA 进行获取
SESSDATA = "b1d7a454%2C1684483318%2C4fd22%2Ab2"

# F12-Console-输入document.cookie 进行获取
cookie = 'buvid3=9808386F-D905-70E8-073C-A9DAA2C287AF59067infoc; ...'

#用户代理可以在任意一个请求的请求头中查看

cookie += f";SESSDATA={SESSDATA}"
headers = {
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36", #用户代理
    "cookie": cookie
}
```

C 获取CID

```
bvid = BV1bR4y1f7P2
url = f"https://api.bilibili.com/x/web-interface/view/detail?bvid={bvid}" #当前
网页的源代码
response = requests.get(url, headers=headers)
response.encoding = "utf-8"
data = response.json() #转换为json格式
#当前页面有多个视频
cids = [dic["cid"] for dic in data["data"]["view"]["pages"]] #cid列表
#当前页面只有一个视频
cid = data["data"]["view"]["pages"][0]["cid"] #cid
```

在 json 数据中 得到 cid = 895476403

D 获取实时弹幕

```
#下一步的oid即上一步我们得到的cid=895476403
oid = 895476403
url = f"https://api.bilibili.com/x/v1/dm/list.so?oid={oid}" #查看实时弹幕网页
```

E 获取历史弹幕

```
oid = 895476403
#获取2022-11-19这一天的历史弹幕
url = f"https://api.bilibili.com/x/v2/dm/web/history/seg.so?type=1&oid={oid}&date=2022-11-19"
response = requests.get(url, headers=headers)
DATA = response.content      #.text 得到文本, .content得到二进制
```

通过这种方式抓包得到的数据是乱码，实际上这不是加密，是使用 **protobuf** 序列化后的文件

- 如果只需要得到弹幕内容，可以使用正则匹配
- 如果要得到弹幕的所有信息，则需要对文件进行逆序列化

基本方法：首先得到B站弹幕的 proto 定义，再使用 protoc 进行编译，获得一个.py文件，然后导入使用

(2) 逆序列化

A 查看B站的弹幕proto定义：

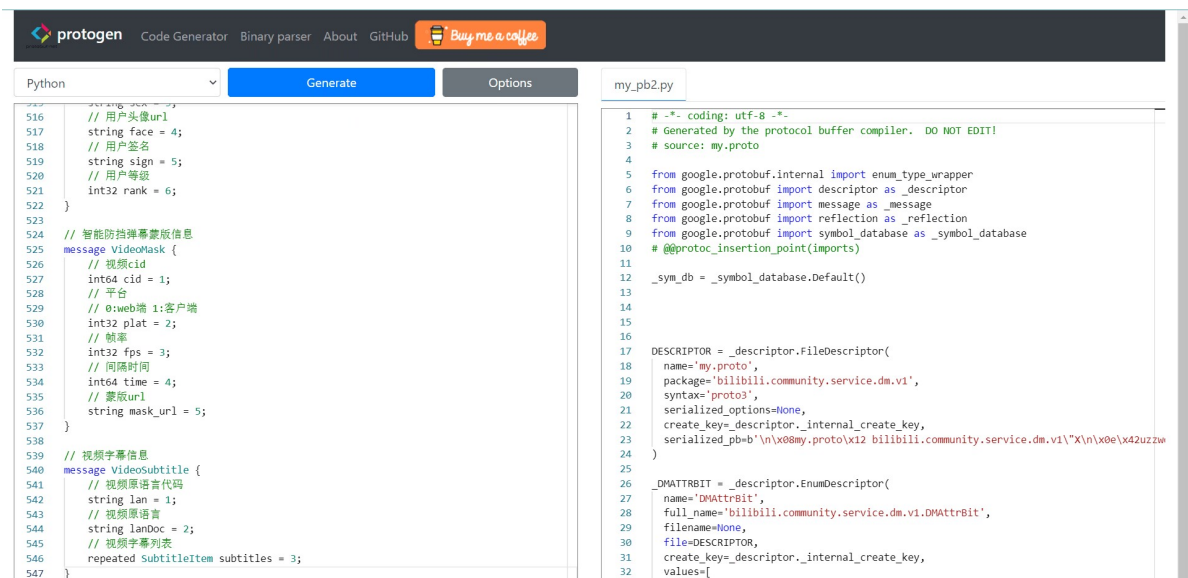
```
https://github.com/SocialSisterYi/bilibili-API-collect/blob/bb437d2012e6291b38c78d42755db9d836d4975f/grpc_api/bilibili/community/service/dm/v1/dm.proto
```

B 进入在线 proto 编译网站

```
https://protogen.marcgrave11.com/#
```

C 编译

将B站的 proto 定义复制到该网站，选择语言为 Python，点击 Generate 后右边得到的就是我们想要的



The screenshot displays the Protogen online proto compiler interface. The left pane shows the input .proto file content for 'dm.proto', including fields for user info, video info, and subtitles. The right pane shows the generated Python code, which imports protobuf modules and defines the corresponding Python classes and descriptors for the .proto file.

D 提取代码

复制右边的代码，在自己的项目里面创建一个py文件，粘贴进去

注意：你创建的文件名必须以_pb2.py结尾！（例如：bili_pb2.py）

E 逆序列化

对 `respon.content` 进行逆序列化

安装相关库：

```
pip install protobuf=3.19.0
```

 #pip安装，注意带上版本号，最新版本暂时存在bug无法使用

使用：

```
import bili_pb2
from google.protobuf import text_format

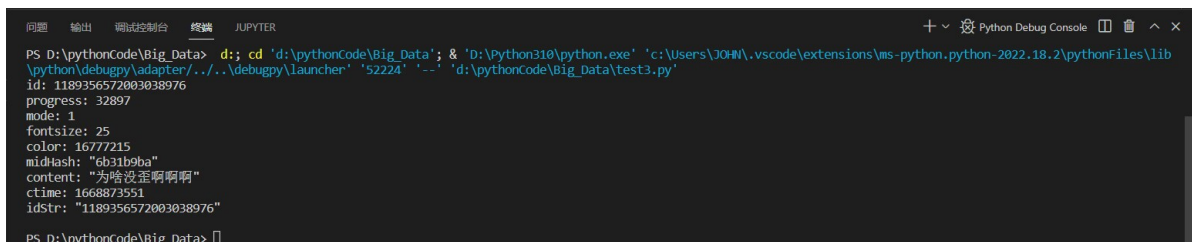
my_seg = bili_pb2.DmSegMobileReply()
my_seg.ParseFromString(DATA)

# DATA是二进制数据
# DATA = resp.content
# 或者
# with open('./test.so','rb') as f:
#     DATA = f.read()

for j in my_seg.elems:
    parse_data = text_format.MessageToString(j, as_utf8=True)

# 此时的parse_data可以直接print
# text_format.MessageToString只能处理一个数据，而my_seg.elems返回的是数据列表
```

打印 `parse_data` 的结果：



```
PS D:\pythonCode\Big_Data> d: cd 'd:\pythonCode\Big_Data'; & 'd:\Python310\python.exe' 'c:\Users\JOHN\.vscode\extensions\ms-python.python-2022.18.2\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '52224' '--' 'd:\pythonCode\Big_Data\test3.py'
id: 1189356572003038976
progress: 32897
mode: 1
fontSize: 25
color: 16777215
midlash: "6b31b9ba"
content: "为啥没歪啊啊啊"
ctime: 1668873551
idStr: "1189356572003038976"
PS D:\pythonCode\Big_Data> []
```

查看其中各个参数的含义：

https://github.com/SocialSisteryi/bilibili-API-collect/blob/bb437d2012e6291b38c78d42755db9d836d4975f/danmaku/danmaku_proto.md

(3) 生成数据集

A 正则匹配

使用正则匹配得到想要的部分

这里截取弹幕在视频中出现的时间和弹幕内容

```
tim = re.findall('progress: (.*)\n', parse_data)    #tim和content是列表，取tim[0]
content = re.findall('content: "(.*)"\n', parse_data)
```

B 排序

根据弹幕在视频中出现的时间对弹幕进行排序

```
if not m.__contains__(tim[0]):    #未存在
    m[tim[0]] = [content[0]]
else:                            #已存在
    m[tim[0]].append(content[0])
...
m = sorted(m.items(), key=lambda x:x[0])    #按照键从小到大排序
f = open(r"D:\大数据导论\danmu-1.txt", mode='w', encoding='utf-8')
for content in m:
    for i in content[1]:
        f.write(i+'\n')
f.close()
```


C 保存数据

解析的所有数据存放在 danmu.txt 中

D 数据集格式与说明

弹幕信息数据集为 TXT 格式

对弹幕进行按行存储

danmu > src >  danmu.txt

```
1  许愿纳西妲，出必还愿
2  每天一遍，扬了教令院
3  别怀疑，就是不对劲，米哈游又开始了！他不演了！
4  别怀疑，就是不对劲，米哈游又开始了！他不演了！
5  草神不歪，玩到关服！
6  不对劲，后面是不是有刀
7  战前总动员
8  纳西妲不歪，玩到关服！！！！！！！！！！！！！！！！
9  枪在手，跟我走。救草神，杀贤狗！！
10 ！！！危！！！
11 掀了教令院！！！
12 快走！！！已经结束啦！！！
13 纳西妲PV真好看，后面的就不用看了，我满足了
14 纳西妲不歪，玩到关服！
15 危
16 开头见！
17 契约已成，食言者当受食言之罚
18 许愿纳斯妲，出必还愿
19 纳西妲不歪，玩到关服
```



2 词频统计

A 结巴分词

```
def jiebaCut(answers_filePath):
    """
    结巴分词
    :param answers_filePath: danmu.txt路径
    :return:
    """
    # 读取danmu.txt
    # answersRdd每一个元素对应danmu.txt每一行
    answersRdd = sc.textFile(answers_filePath)

    # 利用SpardRDD reduce()函数,合并所有弹幕
    str = answersRdd.reduce(lambda x, y: x+y)

    # jieba分词
    # 手动添加一些游戏内常用语
    jieba.add_word("纳西妲")
    jieba.add_word("玩到关服")
    jieba.add_word("大贤者")
    jieba.add_word("出必还愿")
    jieba.add_word("教令院")
    jieba.add_word("小草神")
    jieba.add_word("非洲人")
    jieba.add_word("米哈游")
    jieba.add_word("草神")
    jieba.add_word("刀在手")
    jieba.add_word("跟我走")
    jieba.add_word("天动万象")
    jieba.add_word("大慈树王")
```

```
words_list = jieba.lcut(str)
return words_list
```

首先，利用结巴分词，对弹幕内容进行分词操作；

此外，鉴于本弹幕内容中与原神相关的游戏常用语较多，故使用 jieba.add_word() 向字典中手动添加一些游戏用词，以增加结巴分词的准确性。

B 词频统计

```
def wordcount(isvisualize=False):
    """
    对所有弹幕进行
    :param visualize: 是否进行可视化
    :return: 将序排序结果RDD
    """
    # 读取停用词表
    stopwords = getStopWords(SRCPATH + 'stop_words.txt')

    # 调用结巴分词
    words_list = jiebaCut("file://" + SRCPATH + "danmu.txt")

    # 词频统计
    wordsRdd = sc.parallelize(words_list)

    # wordcount: 去除停用词等同时对最后结果按词频进行排序

    resRdd = wordsRdd.filter(lambda word: word not in stopwords) \
        .filter(lambda word: len(word) > 1) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda a, b: a+b) \
        .sortBy(ascending=False, numPartitions=None, keyfunc=lambda
x: x[1]) \

    # 调用可视化展示
    if isvisualize:
        v = visualize()
        # 饼状图及柱状图可视化
        pieDic = v.rdd2dic(resRdd, 10)
        v.drawPie(pieDic)
        v.drawBar(pieDic)
        # 词云可视化
        wwDic = v.rdd2dic(resRdd, 1500)
        v.drawWorccCloud(wwDic)
    return resRdd
```

根据分词结果进行词频统计；

此外，根据游戏内容和分词结果，手动添加一些停用词以进一步减少错分词的影响；

例如，添加了诸如“我要”、“玩到”、“关服”（实际上是“玩到关服”）、“纳西”（实际上是“纳西妲”）等。

3 可视化处理

A 数据预处理

首先对分词结果进行数据预处理，将RDD转换为Dic，并截取指定长度topK

```
def rdd2dic(self, resRdd, topK):  
    """  
    将RDD转换为Dic，并截取指定长度topK  
    :param resRdd: 词频统计降序排序结果RDD  
    :param topK: 截取的指定长度  
    :return:  
    """  
  
    resDic = resRdd.collectAsMap()  
    # resDic =  
    # 截取字典前K个  
    K = 0  
    wordDic = {}  
    for key, value in resDic.items():  
        # 完成循环截取字典  
        if K > topK:  
            break  
        else:  
            wordDic[key] = value  
            K += 1  
    return wordDic
```

B 词云可视化

```
def drawWorcloud(self, wordDic):  
    """  
    根据词频字典，进行词云可视化  
    :param wordDic: 词频统计字典  
    :return:  
    """  
    # 生成词云  
    # 设置词云背景  
    mask = np.array(Image.open("/home/hadoop/danmu/nahida.png"))  
    wc = wordCloud(font_path='/usr/share/fonts/wqy-microhei/wqy-microhei.ttf',  
                   background_color='white',  
                   mask=mask,  
                   max_words=2000,  
                   width=1920, height=1080,  
                   margin=5)  
    wc.generate_from_frequencies(wordDic)  
    # 保存结果  
    if not os.path.exists(SAVAPATH):  
        os.makedirs(SAVAPATH)  
    wc.to_file(os.path.join(SAVAPATH, '词云可视化.png'))
```

根据统计结果进行词云可视化;

此外, 考虑到视频内容, 我们为词云设置了主角造型的背景层 "nahida.png":



最终得到 “词云可视化.png” 如下:



C 饼图可视化

```
def drawPie(self, wordDic):
    """
    饼图可视化
    :param wordDic: 词频统计字典
    :return:
    """

    key_list = wordDic.keys()      # wordDic所有key组成list
    value_list = wordDic.values()  # wordDic所有value组成list

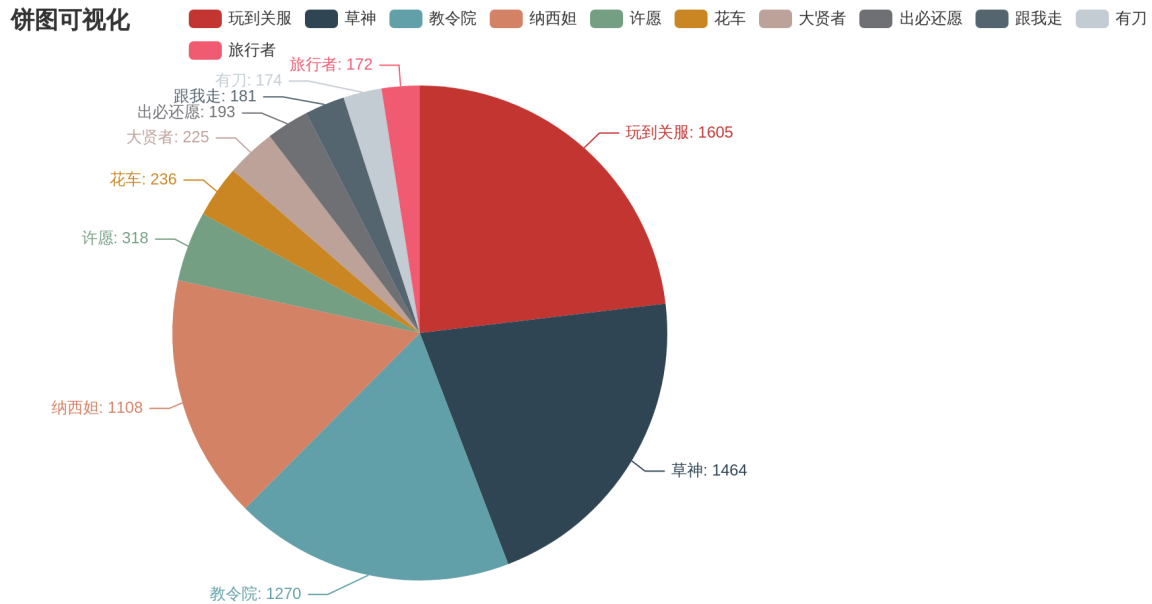
    def pie_position() -> Pie:
        c = (
            Pie()
            .add
            (
                "",
                [list(z)
                 for z in zip(key_list, value_list)], # dic -> list
                center=["35%", "50%"],
            )
            .set_global_opts
            (
                title_opts=opts.TitleOpts(title='饼图可视化'), # 设置标题
                legend_opts=opts.LegendOpts(pos_left="15%"),
```

```

    )
    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}:{c}"))
    )
    return c
# 保存结果
make_snapshot(snapshot, pie_position().render(),
               SAVAPATH + '饼图可视化.png')

```

根据统计结果，绘制饼图，得到“饼图可视化.png”如下：



D 柱状图可视化

```

def drawBar(self, wordDic):
    """
    柱状图可视化
    :param wordDic: 词频统计字典
    :return:
    """
    key_list = list(wordDic.keys()) # wordDic所有key组成list
    value_list = list(wordDic.values()) # wordDic所有value组成list

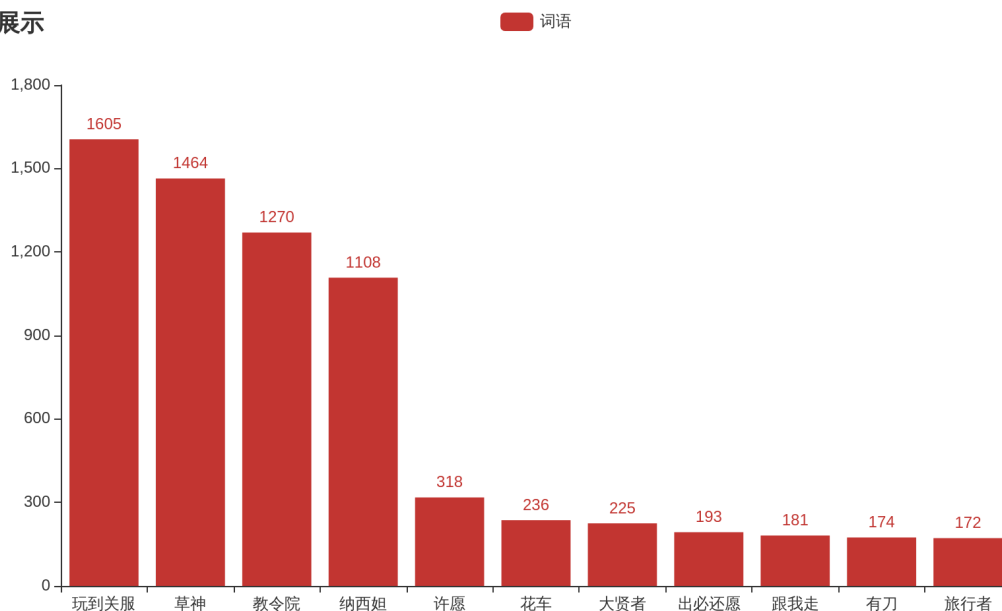
    def bar_position() -> Bar:
        bar = (
            Bar()
            .add_xaxis(key_list)
            .add_yaxis("词语", value_list)
            .set_global_opts(title_opts=opts.TitleOpts(title="词频展示"))
        )
        return bar

    # 保存结果
    make_snapshot(snapshot, bar_position().render(),
                  SAVAPATH + '柱状图可视化.png')

```

根据统计结果，绘制柱状图，得到“柱状图可视化.png”如下：

词频展示



4 弹幕情感分析

get_sentiments() 函数代码如下:

```
def get_sentiments():
    print("弹幕情感分析中...")
    text=open("/home/hadoop/danmu/src/danmu.txt")
    sentences=text.readlines()
    size=len(sentences)
    # 划分数据以平滑化情感变化，降低噪音的影响
    div=200
    idx = np.array([i for i in range(int(size/div))])
    scores=np.array([])
    score=0
    cnt=div
    for i in sentences:
        s = SnowNLP(i)
        sentiment=s.sentiments
        # 情感分级处理
        if sentiment <0.3:      #strongly negative
            score+=1
        elif sentiment <0.5:   #weakly negative
            score+=2
        elif sentiment <0.7:   #weakly positive
            score+=3
        else:                  #strongly positive
            score+=4
        cnt-=1
        if cnt==0: # 当前分组结束时提交得分并重置
            scores=np.append(scores,score)
            score=0
            cnt=div

    # 曲线平滑化（插值法）
    idx_new = np.linspace(min(idx),max(idx),10000)
```

```

scores_smooth = make_interp_spline(idx,scores)(idx_new)
plt.plot(idx_new,scores_smooth)

# plt.plot(idx,scores)
plt.xlabel("process of danmu")
plt.xticks([])
plt.ylabel("score of sentiments")
plt.yticks([])
plt.title("The Change Of Sentiments")
plt.savefig("/home/hadoop/danmu/results/情感分析.png")
print("弹幕情感分析结束...")

```

A SnowNLP情感分析

调用 snownlp 库下的 SnowNLP() 函数，对弹幕进行逐句地情感分析，得分为0（消极）-1（积极）内的值：

```

def get_sentiments():
    ...
    for i in sentences:
        s = SnowNLP(i)
        sentiment=s.sentiments
    ...

```

鉴于 SnowNLP() 给出的评分并非在0-1上的均匀分布，评分的统计结果往往呈现两端少，中间多的迹象，不利于体现情感变化，故而对情感得分进行分级处理,为两端分配更大空间：

```

def get_sentiments():
    ...
    # 情感分级处理
    if sentiment <0.3:      #strongly negative
        score+=1
    elif sentiment <0.5:    #weakly negative
        score+=2
    elif sentiment <0.7:    #weakly positive
        score+=3
    else:                   #strongly positive
        score+=4
    ...

```

划分数据以平滑化弹幕情感的变化，降低噪音的影响：

例如此处我们对相邻的200条弹幕的得分进行求和处理，以此减小同一时间段内个别弹幕与其相邻主体的得分差异较大而带来的噪音影响，使得评分更具可读性，充分反映整体的情绪变化：

```

def get_sentiments():
    ...
    # 划分数据以平滑化情感变化，降低噪音的影响
    div=200
    idx = np.array([i for i in range(int(size/div))])
    scores=np.array([])
    score=0
    cnt=div
    for i in sentences:

```

```

s = SnowNLP(i)
sentiment=s.sentiments
# 情感分级处理
if sentiment <0.3:      #strongly negative
    score+=1
elif sentiment <0.5:    #weakly negative
    score+=2
elif sentiment <0.7:    #weakly positive
    score+=3
else:                  #strongly positive
    score+=4
cnt-=1
if cnt==0: # 当前分组结束时提交得分并重置
    scores=np.append(scores,score)
    score=0
    cnt=div
...

```

B 情感变化曲线绘制

在上一步我们已然对情感分析的得分数据进行了些许平滑化处理与噪音消除处理，但是这时直接对该数据进行绘制仍然称不上美观（得到的结果如同股票涨势图）

故而，我们借助 `np.linspace()` 以及 `make_interp_spline()` 函数，利用插值法对数据做平滑化处理，从而绘制出平滑的情绪变化曲线：

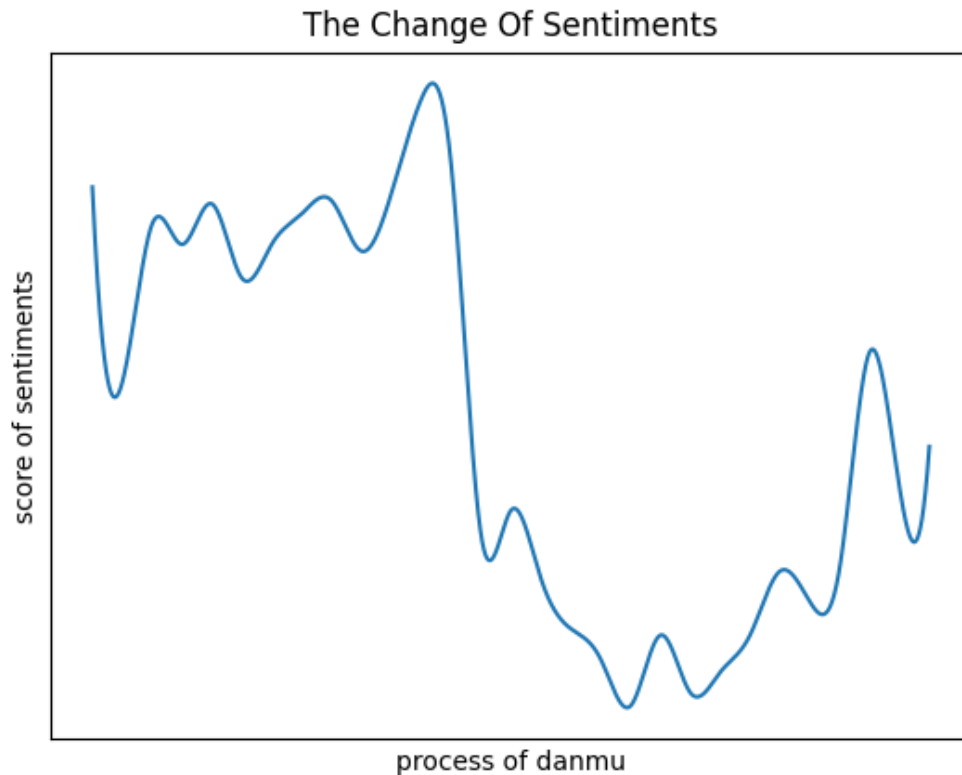
```

def get_sentiments():
    ...
    # 曲线平滑化（插值法）
    idx_new = np.linspace(min(idx),max(idx),10000)
    scores_smooth = make_interp_spline(idx,scores)(idx_new)
    plt.plot(idx_new,scores_smooth)

    # plt.plot(idx,scores)
    plt.xlabel("process of danmu")
    plt.xticks([])
    plt.ylabel("score of sentiments")
    plt.yticks([])
    plt.title("The Change Of Sentiments")
    plt.savefig("/home/hadoop/danmu/results/情感分析.png")
    print("弹幕情感分析结束...")

```

最终绘制情绪变化曲线图“情感分析.png”如下：



可以发现，此曲线对于原视频中观众的情绪变化描述贴切，达到了预期目标。

5. 项目部署 (Hadoop+Spark)

A 启动Hadoop集群

```
cd /usr/local/hadoop
sbin/start-all.sh
```

- [hadoop@master ~]\$ cd /usr/local/hadoop
- [hadoop@master hadoop]\$ sbin/start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-namenode-master.out
slave01: datanode running as process 3568. Stop it first.
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hadoop-resourcemanager-master.out
slave01: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hadoop-nodemanager-slave01.out

B 启动Spark集群

```
cd /usr/local/spark
sbin/start-master.sh
sbin/start-workers.sh
```

- [hadoop@master hadoop]\$ cd /usr/local/spark
- [hadoop@master spark]\$ sbin/start-master.sh
t-slaves.shstarting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-hadoop-org.apache.spark.deploy.maste
r.Master-1-master.out
- [hadoop@master spark]\$ sbin/start-workers.sh
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hadoop-org.apache.spark.deploy.worke
r.Worker-1-master.out
slave01: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hadoop-org.apache.spark.deploy.worker.
Worker-1-slave01.out

云服务器_云主机_弹性云服务器

弹性云服务器 - 控制台

Spark Master at spark://master:7077

+

←

↺

🏠

⚠️ 不安全 | 110.41.156.172:8080

🔍

🔖

🔄

🔒

👤

⋮

spark

3.1.3

Spark Master at spark://master:7077

URL: spark://master:7077

Alive Workers: 2

Cores in use: 8 Total, 0 Used

Memory in use: 12.8 GiB Total, 0.0 B Used

Resources in use:

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20221122162848-192.168.0.131-41972	192.168.0.131:41972	ALIVE	4 (0 Used)	6.4 GiB (0.0 B Used)	
worker-20221122162848-192.168.0.240-42707	192.168.0.240:42707	ALIVE	4 (0 Used)	6.4 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

C 上传集群并运行

```
cd /usr/local/spark
bin/spark-submit --master spark://master:7077 --executor-memory 5G
/home/hadoop/danmu/wordCount.py
```

```
[hadoop@master spark]$ bin/spark-submit --master spark://master:7077 --executor-memory 5G /home/hadoop/danmu/WordCount.py
22/11/22 16:50:41 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
正在爬取B站弹幕...
标题: 《原神》纳西妲角色PV—「生日快乐」
总弹幕数: 63123
视频数量: 4
cid: [872755686, 872756900, 872755555, 872754998]
子标题: ['《原神》纳西妲角色PV—「生日快乐」', '日-《原神》纳西妲角色PV—「生日快乐」', '英-《原神》纳西妲角色PV—「生日快乐」', '韩-《原神》纳西妲角色PV—「生日快乐」']
弹幕开始时间(年-月-日): 2022-10-27
弹幕结束时间(年-月-日): 2022-11-1
为了方便展示, 本次仅爬取中配版视频弹幕...
100%|██████████| 6/6 [00:19<00:00, 3.30s/it]
cid: 872755686 弹幕数: 5948 子标题: 《原神》纳西妲角色PV—「生日快乐」
共获取弹幕5948条!
正在进行词频统计与可视化处理...
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.604 seconds.
Prefix dict has been built successfully.
弹幕情感分析中...
弹幕情感分析结束...
```

云服务器_云主机_弹性云服务器

弹性云服务器 - 控制台

Spark Master at spark://master:7077

ex2 - Spark Jobs

+

←

↺

🏠

⚠️ 不安全 | 110.41.156.172:4040/jobs/

🔍

🔖

🔄

🔒

👤

⋮

spark

3.1.3

Jobs

Stages

Storage

Environment

Executors

ex2 application UI

Spark Jobs (?)

User: hadoop

Total Uptime: 1.4 min

Scheduling Mode: FIFO

Completed Jobs: 2

Event Timeline

Completed Jobs (2)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

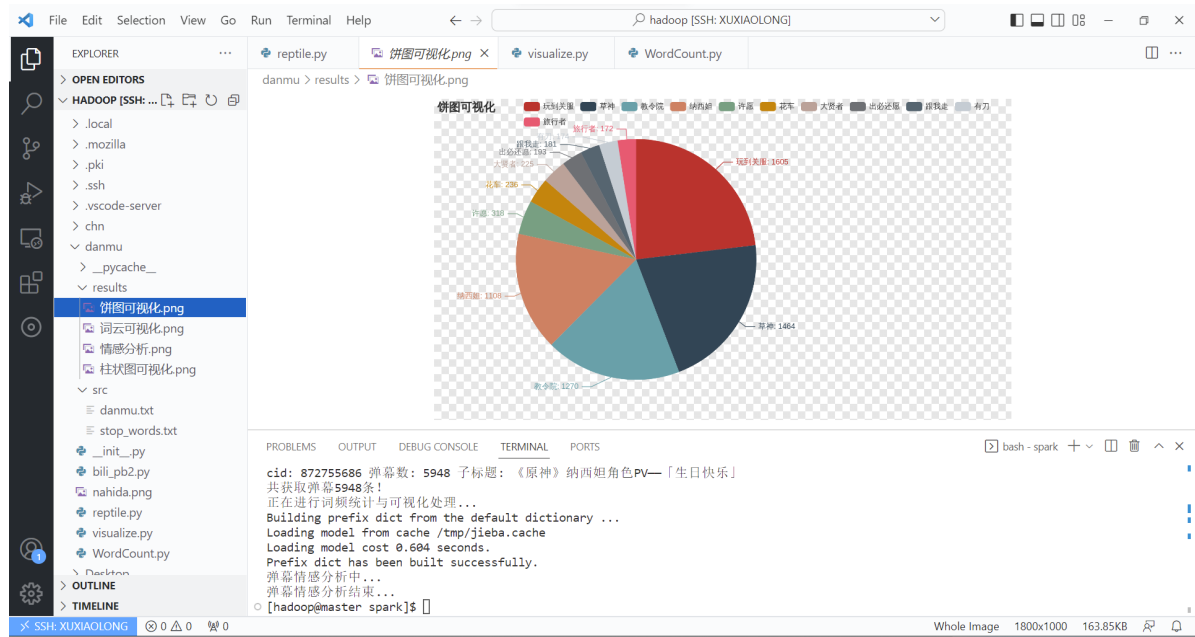
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	collectAsMap at /home/hadoop/danmu/visualize.py:34 collectAsMap at /home/hadoop/danmu/visualize.py:34	2022/11/22 16:37:02	1 s	2/2	2/2
0	reduce at /home/hadoop/danmu/WordCount.py:36 reduce at /home/hadoop/danmu/WordCount.py:36	2022/11/22 16:37:00	0.9 s	1/1	1/1

Page: 1

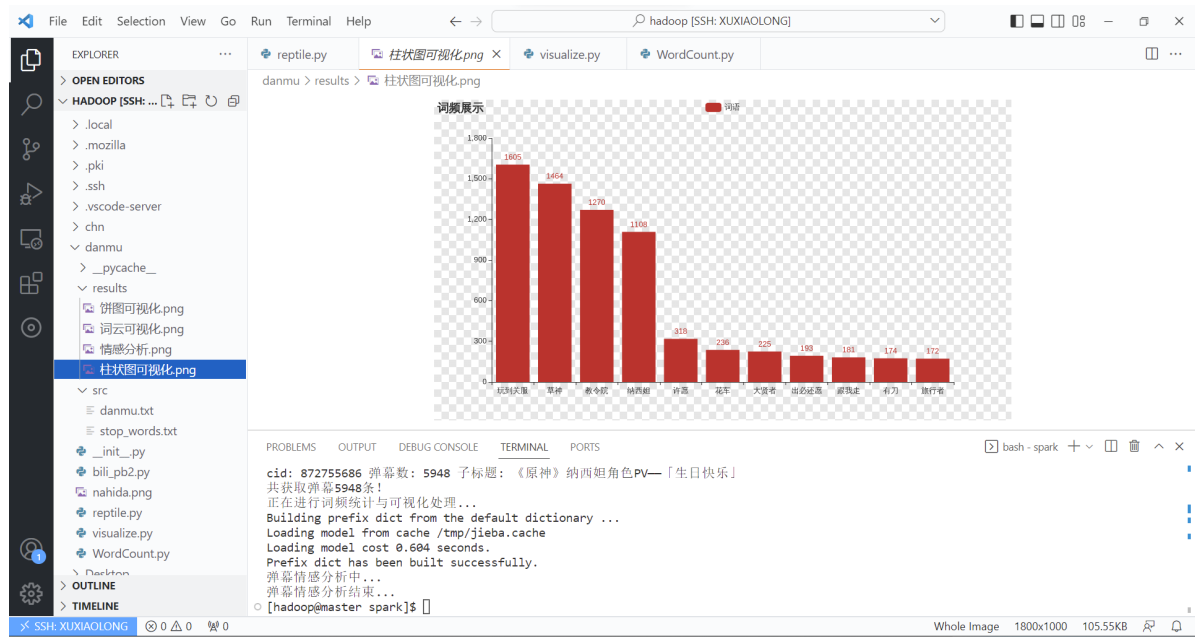
1 Pages. Jump to 1 . Show 100 items in a page. Go

D 运行结果展示

a 饼图



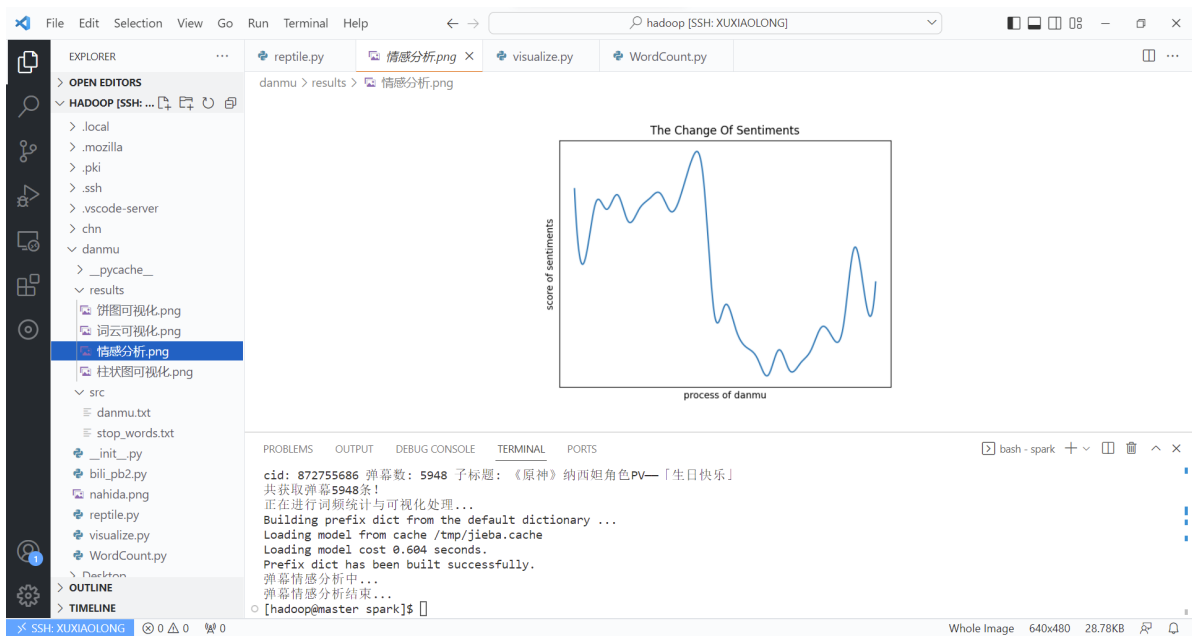
b 柱状图



c 云图



d 情感变化曲线



6 课程总结

通过本次课程的学习，加深了对大数据技术的理解。大数据是一种资源，也是一种工具，提供了一种新的思维方式去理解当今这个信息化世界。通过四个大数据实验的实践学习，掌握了搭建 hadoop + spark 平台并部署运行的方法，学会将大数据技术与其他技术（例如：机器学习）相结合，解决大数据问题。通过本次课程项目，我们开始尝试使用大数据工具来发现寻常可见事物的规律，并以可视化的方式直观地呈现结论。我们共同讨论出项目主题，分工后整合，不断地调试与修改，最终完成了本次项目，可以说收获颇丰。这次学习经历为今后解决问题提供了新的思路，具有启发意义。