青岛科技大学

《数据可视化》

课程设计



**2021年05月10日**

**网易云民谣歌词、评论可视化**

**2021年05月10日**

# 网易云民谣歌词、评论可视化

## 可视化目标与任务

“民谣有三：爱情、理想、远方，听者有三：孤独、平庸、落魄，听后有三：费烟、费酒、费心。”

这是网络上广为流传的一句话，是对于大部分民谣的一个描述，网易云音乐又因评论区众多网友带有浓重消极情绪的评论而被调侃为“网抑云”，那么，这群流浪在祖国大地的现代游吟诗人们都在唱些什么，网友们又为何频频“抑郁”，本文通过对网易云900余首民谣音乐的歌词和1000000+评论的分析并进行可视化，直观地反映出歌手们所唱的内容和网友们抑郁的原因。

本文使用Python对数据进行采集、分析和处理，使用MongoDB存储采集到的数据，使用matplotlib对数据进行可视化并使用echarts.js制作可视化展示大屏。

## 数据采集与存储

过对浏览器抓包，获取网易云的部分API，使用Python编写爬虫，爬取网易云音乐民谣歌单中的音乐，以及音乐评论区的评论、用户信息，将爬取到的数据保存到MongoDB中，供后续分析可视化使用，总计爬取歌词数据900余条，评论数据1000000余条。数据的采集通过python爬虫来实现，我们使用的方案是用多协程爬虫和多进程爬虫来完成数据的采集。

我们采用多进程、多协程来加速爬虫，爬虫采用requests库开发，核心代码如下：

1. r = requests.get(url, params=param, headers=self.headers,
2. cookies=self.cookies)
3. result = r.json()

1.多协程爬虫

线程是指进程内的一个执行单元,也是进程内的可调度实体。线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位。线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈)，但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。线程间通信主要通过共享内存，上下文切换很快，资源开销较少，但相比进程不够稳定容易丢失数据。

协程是一种用户态的轻量级线程，协程的调度完全由用户控制。从技术的角度来说，“协程就是你可以暂停执行的函数”。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快

协程与线程的区别：

* 调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位；并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行；
* 拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源。进程所维护的是程序所包含的资源（静态资源）， 如：地址空间，打开的文件句柄集，文件系统状态，信号处理handler等；线程所维护的运行相关的资源（动态资源），如：运行栈，调度相关的控制信息，待处理的信号集等；
* 系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。但是进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个进程死掉就等于所有的线程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。

协程的优势：

* 极高的执行效率：因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显；
* 不需要多线程的锁机制：因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

2.多进程爬虫

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。每个进程都有自己的独立内存空间，不同进程通过进程间通信来通信。由于进程比较重量，占据独立的内存，所以上下文进程间的切换开销（栈、寄存器、虚拟内存、文件句柄等）比较大，但相对比较稳定安全。

多进程的优点是稳定性好，一个子进程崩溃了，不会影响主进程以及其余进程。基于这个特性，常常会用多进程来实现守护服务器的功能。 然而多线程不足之处在于，任何一个线程崩溃都可能造成整个进程的崩溃，因为它们共享了进程的内存资源池。由于python设计之初，没预料到多核cpu能够得到现在的发展，只考虑到了单核cpu。为了更好的实现多线程之间数据完整性与状态同步，于是设计出了一个全局解析器锁（GIL, global interpreter lock)。GIL确保Python进程一次运行一个线程(其它线程处于等待I/O或者睡眠状态)，无论当前cpu有多少核心。这就意味着Python虽然可以实现多线程，但是在任意时间点仅有一个核心在执行Python指令（即线程无法并行运算)，无法发挥现代多核cpu的性能。而多进程可以利用多核cpu。

3.我们的方案：多协程+多进程

由于GIL的原因，python解释器中总是只有一个线程存在，因此很难利用多线程来达到并行的目的。但多进程并不受此限制，因此我们可以利用python中的多进程来达到并行的目的。同时，我们可以在每个进程中使用协程来实现异步的处理。综上，我们可以利用多进程+协程来极大的提升我们爬虫的效率。

借助gevent实现5个协程，加快爬虫爬取的速度，核心代码如下：

1. n = len(playlist)
2. jobs = [
3. gevent.spawn(self.start, playlist[: n // 5]),
4. gevent.spawn(self.start, playlist[n // 5: (n // 5) \* 2]),
5. gevent.spawn(self.start, playlist[(n // 5) \* 2: (n // 5) \* 3]),
6. gevent.spawn(self.start, playlist[(n // 5) \* 3: (n // 5) \* 4]),
7. gevent.spawn(self.start, playlist[(n // 5) \* 4:])
8. ]
9. gevent.joinall(jobs)

借助Process库实现多进程，因为进程在启动时需要传入启动函数，故将爬虫的启动写在函数run中，代码如下：

1. def run(pl, i):
2. print('start...', i)
3. spider = NeteaseSpider()
4. spider.run(pl)

为了防止重复爬取，我们将歌单等分成5份，每个进程爬取一份，核心代码如下：

1. begin = 0
2. end = 20
3. step = 20
4. for \_ in range(5):
5. p = Process(target=run, args=(playlist[begin:end], \_))
6. p.start()
7. begin = end
8. end += step

4.数据存储方案

我们采用MongoDB存储数据，MongoDB是文档类型的非关系型数据库，其中存储的是json格式的数据，这与Python的字典数据结构基本一致，极大方便开发，提高了开发的效率。

mongodb与关系型数据库相比的优缺点：

1、与关系型数据库相比，MongoDB的优点：

1. 弱一致性（最终一致），更能保证用户的访问速度：

举例来说，在传统的关系型数据库中，一个COUNT类型的操作会锁定数据集，这样可以保证得到“当前”情况下的较精确值。这在某些情况下，例 如通过ATM查看账户信息的时候很重要，但对于Wordnik来说，数据是不断更新和增长的，这种“较精确”的保证几乎没有任何意义，反而会产生很大的延 迟。他们需要的是一个“大约”的数字以及更快的处理速度。

但某些情况下MongoDB会锁住数据库。如果此时正有数百个请求，则它们会堆积起来，造成许多问题。我们使用了下面的优化方式来避免锁定：

每次更新前，我们会先查询记录。查询操作会将对象放入内存，于是更新则会尽可能的迅速。在主/从部署方案中，从节点可以使用“-pretouch”参数运行，这也可以得到相同的效果。

使用多个mongod进程。我们根据访问模式将数据库拆分成多个进程。

1. 文档结构的存储方式，能够更便捷的获取数据。

对于一个层级式的数据结构来说，如果要将这样的数据使用扁平式的，表状的结构来保存数据，这无论是在查询还是获取数据时都十分困难。

1. 内置GridFS，支持大容量的存储。

GridFS是一个出色的分布式文件系统，可以支持海量的数据存储。

  内置了GridFS了MongoDB，能够满足对大数据集的快速范围查询。

1. 内置Sharding。

提供基于Range的Auto Sharding机制：一个collection可按照记录的范围，分成若干个段，切分到不同的Shard上。

Shards可以和复制结合，配合Replica sets能够实现Sharding+fail-over，不同的Shard之间可以负载均衡。查询是对 客户端是透明的。客户端执行查询，统计，MapReduce等操作，这些会被MongoDB自动路由到后端的数据节点。这让我们关注于自己的业务，适当的 时候可以无痛的升级。MongoDB的Sharding设计能力较大可支持约20 petabytes，足以支撑一般应用。

这可以保证MongoDB运行在便宜的PC服务器集群上。PC集群扩充起来非常方便并且成本很低，避免了“sharding”操作的复杂性和成本。

1. 第三方支持丰富。(这是与其他的NoSQL相比，MongoDB也具有的优势)

现在网络上的很多NoSQL开源数据库完全属于社区型的，没有官方支持，给使用者带来了很大的风险。而开源文档数据库MongoDB背后有商业公司10gen为其提供供商业培训和支持。而且MongoDB社区非常活跃，很多开发框架都迅速提供了对MongDB的支持。不少知名大公司和网站也在生产环境中使用MongoDB，越来越多的创新型企业转而使用MongoDB作为和Django，RoR来搭配的技术方案。

1. 性能优越：

在使用场合下，千万级别的文档对象，近10G的数据，对有索引的ID的查询不会比mysql慢，而对非索引字段的查询，则是全面胜出。 mysql实际无法胜任大数据量下任意字段的查询，而mongodb的查询性能实在让我惊讶。写入性能同样很令人满意，同样写入百万级别的数 据，mongodb比我以前试用过的couchdb要快得多，基本10分钟以下可以解决。补上一句，观察过程中mongodb都远算不上是CPU杀手。

与关系型数据库相比，MongoDB的缺点：

1. mongodb不支持事务操作。

所以事务要求严格的系统（如果银行系统）肯定不能用它。(这点和优点是对应的)

1. mongodb占用空间过大。

  关于其原因，在官方的FAQ中，提到有如下几个方面：

* 1. 空间的预分配：为避免形成过多的硬盘碎片，mongodb每次空间不足时都会申请生成一大块的硬盘空间，而且申请的量从64M、128M、256M那 样的指数递增，直到2G为单个文件的较大体积。随着数据量的增加，你可以在其数据目录里看到这些整块生成容量不断递增的文件。
  2. 字段名所占用的空间：为了保持每个记录内的结构信息用于查询，mongodb需要把每个字段的key-value都以BSON的形式存储，如果 value域相对于key域并不大，比如存放数值型的数据，则数据的overhead是较大的。一种减少空间占用的方法是把字段名尽量取短一些，这样占用 空间就小了，但这就要求在易读性与空间占用上作为权衡了。我曾建议作者把字段名作个index，每个字段名用一个字节表示，这样就不用担心字段名取多长 了。但作者的担忧也不无道理，这种索引方式需要每次查询得到结果后把索引值跟原值作一个替换，再发送到客户端，这个替换也是挺耗费时间的。现在的实现算是 拿空间来换取时间吧。
  3. 删除记录不释放空间：这很容易理解，为避免记录删除后的数据的大规模挪动，原记录空间不删除，只标记“已删除”即可，以后还可以重复利用。
  4. 可以定期运行db.repairDatabase()来整理记录，但这个过程会比较缓慢

③MongoDB没有如MySQL那样成熟的维护工具，这对于开发和IT运营都是个值得注意的地方。MongoDB适合存储一些关系简单、数据量又很大的数据，比如我们的平台上虚拟机的监控信息，包括内存、IO、CPU、网络等数据，每隔几秒就采集一次数据，每周、每月，量很大，而且旧的监控数据也不会保留太长时间，就使用的mongodb来存储这些数据；

另外mongodb的集群部署相对比较简单，易于扩展；比如主从复制，在mongo.conf配置几个参数就OK了；分片集群的配置也比较简单。还支持使用命令行来进行动态地添加和删除节点；

2、Mongodb的优点与不足

（1）Mongodb的不足之处

* 1. 在集群分片中的数据分布不均匀
  2. 单机可靠性比较差
  3. 大数据量持续插入，写入性能有较大波动
  4. 磁盘空间占用比较大

（2）Mongodb的过人之处

* 1. 无模式
  2. 查询与索引方式灵活，是最像SQL的Nosql
  3. 支持复制集、主备、互为主备、自动分片等特性

3、Mongodb与redis相比较：

1. mongoDB 源码语言是C++,redis也是C或C++,
2. mongodb 文件存储是BSON格式类似JSON，或自定义的二进制格式。
3. mongodb与redis性能都很依赖内存的大小，mongodb 有丰富的数据表达、索引；最类似于关系数据库，支持丰富的查询语言，redis数据丰富，较少的IO ，这方面mongodb优势明显。
4. mongodb不支持事物，靠客户端自身保证，redis支持事物，比较弱，仅能保证事物中的操作按顺序执行，这方面 redis优于mongodb。mongodb对海量数据的访问效率提升，redis 较小数据量的性能及运算,这方面 mongodb性能优于redis .monbgodb 有mapredurce功能，提供数据分析，redis 没有 ，这方面 mongodb优于redis 。

关于MongoDB的连接、插入等操作，主要用到如下代码：

连接：

1. def \_\_init\_\_(self):
2. ... ...
3. self.client = pymongo.MongoClient(config.mongodb\_url)
4. self.db = self.client[config.db\_name]
5. self.mongodb = self.db[config.col]
6. ... ...

插入：

1. self.mongodb.insert\_one(info)

## 数据处理方案

本文使用Python对数据进行分析处理，对于歌词和评论内容，使用jieba分词器进行分词预处理，分词同时删除停用词，并使用正则表达式过滤特殊符号等无关内容，以便于后续分析。

本文主要借助jiagu、Snownlp等NLP库对歌词进行了情感倾向分析，同时统计了歌词中一些代表性词语的出现频率，根据分析结果，回答了民谣歌手们喜欢吟诵哪里、喜欢吟诵哪个季节、喜欢未来还是过去等几个问题。

对于评论数据，主要通过统计一些代表性词汇的词频，分析得知网友们为什么“抑郁”，并根据评论时间得知什么时间“抑郁”。

爬取到的数据中，还包含了大量用户信息数据，本文简要处理了用户信息数据，得到了“抑郁”人群的年龄分布、地区分布、性别等数据。

1. 分词

分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。我们知道，在英文的行文中，单词之间是以空格作为自然分界符的，而中文只是字、句和段能通过明显的分界符来简单划界，唯独词没有一个形式上的分界符，虽然英文也同样存在短语的划分问题，不过在词这一层上，中文比之英文要复杂得多、困难得多。

得益于深度学习和NLP技术的飞速发展，Python拥有众多第三方NLP库，本文采用jieba库，对数据进行分词处理。

核心代码如下：

1. sentence\_depart = jieba.cut(sentence.strip())
2. *# 输出结果为outstr*
3. outstr = ''
4. *# 去停用词*
5. for word in sentence\_depart:
6. if word not in stop\_words:
7. if word != '\t':
8. outstr += word
9. outstr += " "
10. 情感分析

文本情感分析又称意见挖掘、倾向性分析等。简单而言,是对带有情感色彩的主观性文本进行分析、处理、归纳和推理的过程。歌词一定程度上可以反应作曲者在创作时的情绪，本文通过jiagu对歌词进行情感分析，并将分析结果在可视化大屏中展示。

Jiagu以BiLSTM等模型为基础，使用大规模语料训练而成。将提供中文分词、词性标注、命名实体识别、情感分析、知识图谱关系抽取、关键词抽取、文本摘要、新词发现、情感分析、文本聚类等常用自然语言处理功能。

核心代码如下：

1. def lyric\_emotion(song\_id):
2. res = dao.find\_by\_id(song\_id)
3. lyric = dao.remove\_stop\_words(res['lyric'])
4. emotions = []
5. result = []
6. i = 0
7. for line in lyric.replace(' ', '').split('\n'):
8. i += 1
9. emotions.append([i, jiagu.sentiment(line)])
10. for emotion in emotions:
11. if emotion[1][0] == 'positive':
12. result.append([emotion[0], round(emotion[1][1], 2)])
13. else:
14. result.append([emotion[0], -1 \* round(emotion[1][1], 2)])
15. return {'name': '《' + res['name'] + '》', 'data': result}
16. 代表词统计

通过对歌词、评论中一些关键词的统计，对得到的结果进行可视化，可以更加直观地反映出歌词具体写的是什么，用户评论地内容和什么相关等潜在信息，进而可以进一步分析歌手、用户的情绪等。

所统计的关键词及类别如下表：

|  |  |
| --- | --- |
| 类别 | 关键词 |
| 地点（歌手喜欢哪里） | 南方、北方、远方、家乡、故乡 |
| 季节（歌手喜欢哪个季节） | 春天、夏天、秋天、冬天 |
| 哪一天（歌手更喜欢哪一天） | 前天、昨天、今天、明天、后天 |
| 亲情（为什么网抑云） | 奶奶、外婆、外公、爷爷、爸、  妈、家人、姐姐、哥哥 |
| 梦想（为什么网抑云） | 未来、成功、上岸、考研、高考、中考 |
| 爱情（为什么网抑云） | 前任、女友、结婚、男友、分手、  恋爱、感情、两个人、一个人 |
| 绝症（为什么网抑云） | 绝症、癌症、晚期、病魔 |
| 事业（为什么网抑云） | 创业、老板、工作、上班、同事、公司、辞职 |
| 过往（为什么网抑云） | 小时候、上初中、年轻的时候、  高中的时候、初中的时候、童年、记忆、时光 |

核心代码如下（以歌词词频统计为例）：

1. area = {'南方': 0, '北方': 0, '远方': 0, '家乡': 0, '故乡': 0}
2. season = {'春天': 0, '夏天': 0, '秋天': 0, '冬天': 0}
3. lyric\_time = {'前天': 0, '昨天': 0, '今天': 0, '明天': 0, '后天': 0}
4. lyric = []
5. pattern = re.compile(r'[\u4e00-\u9fa5]\*')
6. for k in tqdm(lyrics.keys()):
7. t = re.sub('[a-zA-Z0-9’!"#$%&\'()\*+,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^\_`{|}~\s]+', "",
8. lyrics[k].replace('\n', ' '))
9. lyric.append(pattern.search(t).string)
10. for line in lyrics[k].split('\n'):
11. for word in area.keys():
12. if word in line:
13. area[word] += 1
14. for word in lyric\_time.keys():
15. if word in line:
16. lyric\_time[word] += 1
17. for word in season.keys():
18. if word.replace('天', '') in line:
19. season[word] += 1

## 可视化呈现方案及结果

本文使用matplotlib进行数据分析过程中的可视化，并使用wordcloud制作歌词和评论的图云，直观表现出出现次数最多的几个词语，最后，选取代表性的图表，使用echarts.js制作可视化大屏。

在数据分析的过程，使用matplotlib绘制了歌手更喜欢哪个季节、用户性别占比、用户VIP占比等饼图，评论发布时间曲线以及用户年龄分布柱状图等图表。

最终，选取歌词情感分析、用户VIP占比、用户分布、评论emoji频次、用户年龄统计、评论时间分布等六个图表，汇总制作可视化大屏。采用rem作为CSS的布局单位，尽可能去兼容不同分辨率、不同大小的屏幕。

1. echart.js绘制图表

ECharts是一个使用JavaScript实现的开源可视化库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖矢量图形库ZRender，提供直观，交互丰富，可高度个性化定制的数据可视化图表。

主要有以下个特性：

* 丰富的可视化类型
* 多种数据格式无需转换直接使用
* 千万数据的前端展现
* 移动端优化
* 多渲染方案，跨平台使用
* 深度的交互式数据探索
* 多维数据的支持以及丰富的视觉编码手段
* 动态数据
* 绚丽的特效
* 无障碍访问（4.0+）

我们采用echart.js进行展示大屏图表的渲染，为方便与歌词联动，采用函数的形式，通过ajax请求数据，核心代码如下：

1. function func\_name(song\_id) {
2. $.ajax({
3. url: domain + '/<api>/' + <params>,
4. success: function (res) {
5. res = JSON.parse(res);
6. var data = res['data'];
7. var myChart = echarts.init(document.getElementById('<id>'));
8. var option;
9. option = {
10. // echarts配置信息
11. };
12. myChart.setOption(option);
13. }
14. })
15. }
16. JQuery介绍

jQuery库可以通过一行简单的标记被添加到网页中，是一个JavaScript函数库。

jQuery库包含以下特性：

* HTML元素选取
* HTML元素操作
* SS操作
* HTML事件函数
* JavaScript特效和动画
* HTML DOM遍历和修改
* AJAX

歌词的滚动和颜色变化主要通过JQuery实现和CSS3的animate动画实现，核心代码如下：

1. $("#lyric").text("歌词");
2. *//把上一条歌词颜色进行恢复*
3. $("#lyricDiv ul li").css("color", "#cbc7c7");
4. *//把现在这条歌词高亮显示*
5. $("#lyricDiv ul li").get(lyricIndex + j).style.color = "#ff6666";
6. *//初始歌词的高度是80,所以将80-叠加的高度则得出歌词需要滑动的高度*
7. $("#lyricDiv ul").animate({
8. "top": 80 - lyricHeight[lyricIndex - 1] + "px"
9. }, 1000);

数据的请求，我们采用JQuery提供的Ajax实现：

1. $.ajax({
2. url: <url>,
3. method: 'GET',
4. async: false,
5. success: function(res) {
6. // 处理返回数据
7. }
8. })

热门评论的滚动采用了定时执行函数的方式实现：

1. var show = document.getElementsByClassName("show1")[0];
2. var timeId = setInterval(add, 2000);
3. function add(){
4. try{
5. i = 0
6. $.ajax({
7. url: 'http://127.0.0.1:5000/comment',
8. method: 'GET',
9. success: function(res) {
10. res = JSON.parse(res);
11. if((res['code']+1) % 2 === 1) {
12. show.innerHTML = show.innerHTML + '<li><p style="float: left;"><b>'+res['song']+'</b><br>' +
13. res['content'] + '<br> </p> <p class="fr pt17">' + res['nickname'] +'</p> </li>';
14. i = 1;
15. } else {
16. show.innerHTML = show.innerHTML + '<li class="bg"><p style="float: left;"><b>'+res['song']+'</b><br>' +
17. res['content'] + '<br> </p> <p class="fr pt17">' + res['nickname'] +'</p> </li>';
18. i = 0;
19. }
20. },
21. error: function(res){  }
22. })
23. } catch(err){ }
24. }
25. add();
26. main.onmouseover = function(){
27. clearInterval(timeId)
28. };
29. main.onmouseout = function(){
30. timeId = setInterval(add, 2000);
31. };
32. 页面布局

采用CSS、CSS3进行页面布局设计，页面整体布局如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 情感分析 | 音乐播放 | | 热门评论 |
| 用户VIP占比 |
| 用户分布 | 评论Emoji使用比例 | 用户年龄统计 | 评论时间分布 |

1. rem自适应布局

rem（font size of the root element）是指相对于根元素的字体大小的单位。简单的说它就是一个相对单位。看到rem大家一定会想起em单位，em（font size of the element）是指相对于父元素的字体大小的单位。它们之间其实很相似，只不过一个计算的规则是依赖根元素一个是依赖父元素计算。屏幕适配有多种方法，综合考虑在我们使用rem自适应布局。

1. matplotlib、wordcloud可视化结果

* 歌词数据

民谣歌词大多数有歌手自己创作，本文通过对歌词数据的分析，来展示歌手的内心世界。

图表, 饼图

描述已自动生成

根据春夏秋冬四季相关词语的词频统计结果绘制饼图，由上图可知，这群游荡的诗人们更喜欢春天，其他三个季节占比相似，差别不大。

图表, 条形图

描述已自动生成

根据明天、后天等几个关键词的词频统计结果绘制柱状图，可见歌手们最喜欢明天，最少吟诵后天，结果可想而和，“明天”写在歌词中也跟该押韵，而“后天”可能略显拗口。

图表, 饼图

描述已自动生成

根据南北方、家乡远方等词语的词频分析结果绘制饼图，结果如上图，可见歌手们最喜欢远方，其次受乡愁情感的影响，也经常吟诵家乡，南方和北方占比相差不多。



使用wordcloud对分词后的歌词绘制词云，一个大大的“爱”字映入眼帘，可见这群诗人们大多在吟诵爱情。

以上分析结果用一句话来概括，无疑就是“民谣有三：爱情、理想、远方”。

* 用户数据

图表, 条形图

描述已自动生成

对用户的年龄进行统计，绘制柱形图，如上图，可见网易云音乐的用户群体大多为20岁左右的年轻人，也正是这批人日常在网络上“抑郁”。

图表, 饼图

描述已自动生成

统计用户的性别绘制饼图，可见网易云的用户群体中男女比例基本相平，爱好音乐不分男女。

图表, 饼图

描述已自动生成

对用户听歌量进行统计，绘制饼图，如上所示，可见活跃在评论区的大部分为听歌量大于1000的用户，即老用户。

图表, 饼图

描述已自动生成

对会员数据进行统计，绘制饼图，如上所示，可见大部分用户并没有开通会员，一方面网易云现阶段音乐版权少，导致用户不愿付费，另一方面，无需会员权限的音乐满足了大多数用户的需求。而开通会员的用户中，大部分为黑胶会员，开通音乐包的较少，音乐包的性价比确实也不如黑胶会员。

* 评论数据

图表, 饼图

描述已自动生成

根据评论关键词词频统计结果，绘制饼图，如上所示，可见导致大部分网友“抑郁”的原因是爱情，其次是亲情，最少的是绝症。民谣歌手们吟诵最多的也是爱情，看来这群诗人们唱到了网友们的心里去了。

图表, 折线图

描述已自动生成

对评论的时间进行统计、可视化，可见评论大多在23点发布，因为网易云用户群体以20岁左右年轻人为主，23点的年轻人大多还没有睡，一般这个时间大家都躺在床上玩手机听着音乐，可能因为某句歌词产生共鸣，突然就“抑郁”了，于是在评论区留下评论，而到了晚上12点以后，大部分人熬不住了，睡着了，所以评论数量也直线式减少。

文本, 白板

描述已自动生成

随机选取部分评论数据绘制词云，如上图所示，较为醒目的词语有“喜欢”、“爱”、“想”、“加油”等，可见“抑郁”网友们经常在评论区讲述自己的爱情、亲情、理想等内容，这一结果与“为什么网抑云”一图所反映的结果相符。

1. 大屏后台

数据大屏要和数据库中的数据进行联动，必然不可缺少后台，为方便开发，提高开发效率，本文使用轻量化WEB框架Flask进行开发，并借鉴MVC的设计思想，分层开发，解耦程序。由于后台程序规模较小，为加快开发速度，故仅实现了dao层、service层、controller层，直接使用Python的字典封装数据，没有实现实体类。

因测试时直接打开html文件，因此需要配置Flask，允许跨域访问，代码如下：

1. app = Flask(\_\_name\_\_, root\_path='./', template\_folder='./template')
2. CORS(app, supports\_credentials=True)
3. 大屏展示截图



## 总结

如果把民谣拟人化，那么他是一个喜欢北方的小伙子，觉得世界很很不公平，但抱怨归抱怨，到底是对生活有希望的，憧憬着明天，向往诗和远方，在春天感到快乐，在冬天感到孤独，为了理想远离家乡，偶尔受到挫折会想家。评论区的网友们听了民谣的故事，感觉他的经历有些地方和自己的相似，不禁有感而发，倾诉自己，关于“抑郁”，则是大部分人无聊而已。

本次课程设计分析了网易云音乐部分民谣的歌词、评论，直观了解到了民谣歌手内心的世界、了解到了评论区网友们“抑郁”的原因，最重要的是巩固了课程所学知识。同时，分组协作提高了我们的沟通能力，小组成员之间互帮互助、共同思考，相互查漏补缺，互相给予信心，保证高速、高效率的完成任务，充分体现了团队精神。

## 附录

* Jupyter Notebook文件需要使用PyCharm打开，PyCharm开启Jupyter Server后使用浏览器打开，其他方式开启Jupyter Notebook，因根目录设置不同，需要将config.py复制一份到同级目录，否则会报错。
* 关于大屏展示加载速度，因为刚打开时需要同步请求部分API获取在线数据，供展示使用，所以打开时加载速度稍慢。
* 网易云api：

|  |  |
| --- | --- |
| 地址 | 说明 |
| http://music.163.com/api/v1/user/detail/<user\_id> | 获取用户信息  user\_id：用户id |
| http://music.163.com/api/playlist/detail?id=<list\_id> | 获取歌单详情信息  list\_id：歌单id |
| http://music.163.com/api  /song/lyric?id=<song\_id>&lv=1&kv=1&tv=-1 | 获取歌词  song\_id：歌曲id  tv=-1(不翻译)  tv=1(翻译) |
| http://music.163.com/api/search/suggest?  s=<keywords>&type=<type\_id> | 搜索  keywords：搜索关键字  type\_id：搜索类型  1(默认，单曲搜索)  10(专辑搜索)  100(歌手搜索)  1000(歌单搜索)  1002(用户搜索)  1004(mv搜索) |
| http://music.163.com/api/v1/resource/comments/  R\_SO\_4\_<song\_id>?limit=10&offset=100 | 获取歌曲评论  song\_id：歌曲id  limit：每页评论数  offset：评论数偏移量 |