# 网易云音乐用户数据的分析和可视化技术研究与实现

作者: 孟政元

指导教师: 王康平

日期: 2018年

学院: 计算机科学与技术学院

专业: 计算机科学与技术专业

## 摘要

网易云音乐中的用户大致可以分为两类, 一类是普通听众, 另一类是歌手, 音乐创作人这类发布音乐的人.在整个用户群体中, 存在着许多的关联关系. 这些用户可以随意的互相关注, 而每个用户又有着自己独特的特征, 比如性别, 地域, 年龄等等.用户之间相互关注的情况可以用一个很大的有向图来表示, 用户是有向图中的点, 每一个用户对另外一个用户的关注就是有向图的一条边. 这篇论文对部分用户信息(大概N用户)进行了采集,分析了用户的个体特征, 用户之间相互关注, 用户之间间接性的相互关联(基于floyd算法)等等在地域上的分布情况,并且基于分析结果进行回归预测, 从而估计整个用户群体的情况, 最后进行可视化展现, 使得结果直观明了.

## 数据的爬取

### 原理

跨站请求伪造 (CSRF), 伪造请求头 , 调用官方 API. 爬虫通过提供与正常请求相同的参数, 通过post方法向网易云的后台发起请求, 从而获取相应的数据.

### 调用官方后台API的具体方法

#### 访问方式

网易云音乐的后台api的url为`http://music.163.com/weapi/${api\_name}?csrf\_token=${token}, 通过发起http请求即可访问网易云音乐的后台api，但是后台的api需要加上正确的参数并对参数进行相应的处理，否则会被网易云音乐的后台判定为非法请求，直接屏蔽掉

#### 参数的处理

上面已经提到，在对网易云音乐的后台发起请求的时候，必须带上相应的参数并进行正确的处理，否则请求会被当做非法的而自己屏蔽掉。参数的具体处理方法如下(以获取歌曲的评论为例)：

1. csrf\_token

csrf(cross-site request forgery)即跨站请求伪造，这个参数主要是解决一些基于web的攻击，在请求的时候可以设为空字符串（假设客户端之前没有访问过网易云音乐的网站），但不能没有这个参数。

1. 其余的各个参数（offset，rid，limie）

网易云音乐的后台请求一律采取的是post方法，请求数据放在body部分。

将这些参数组成一个对象，如下所示，作为未加密处理的body部分。

const body =  
 {  
 offset: req.query.offset || 0,  
 rid: rid,  
 limit: req.query.limit || 20,  
 csrf\_token: ""  
 };

接下来通过相应的方法对请求进行加密处理：用一个16位长度的随机字符串作为加密的秘钥，然后按照下面代码所示的方法进行加密，最后返回一个对象，该对象具有两个属性：params是加密后的数据，encSecKey代表加密用的秘钥，用于服务器端对加密后的数据进行解密。

function Encrypt(obj)

{

const text = JSON.stringify(obj)

const secKey = createSecretKey(16)

const encText = aesEncrypt(aesEncrypt(text, nonce), secKey)

const encSecKey = rsaEncrypt(secKey, pubKey, modulus)

return {

params: encText,

encSecKey: encSecKey

}

}

然后通过post方法发起请求：url为该API对应的url，请求方法为post方法，body的内容就是上面返回的对象经过querystring序列化之后的返回结果。如下所示(注意：为了直观明了，代码有所删减，不能直接运行)：

function createWebAPIRequest

(

data,

callback,

errorcallback

)

{

const cryptoreq = Encrypt(data);

const options =

{

body: querystring.stringify

(

{

params: cryptoreq.params,

encSecKey: cryptoreq.encSecKey

}

),

proxy: proxy

};

request(options, function(error, res, body)

{

if (error)

{

console.error(error);

errorcallback(error);

}

else

{

callback(body, cookie);

}

});

}

createWebAPIRequest

(

"music.163.com",

`/weapi/v1/resource/comments/R\_SO\_4\_${rid}/?csrf\_token=`,

"POST",

data,

cookie,

music\_req =>

{

res.send(music\_req);

},

err => res.status(502).send(err.message)

)

注意：代码中用到了ES6标准中的模板字符串和箭头函数。

1. 模板字符串使用反引号 (` `) 来代替普通字符串中的用双引号和单引号。模板字符串可以包含特定语法(${expression})的占位符。占位符中的表达式和周围的文本会一起传递给一个默认函数，该函数负责将所有的部分连接起来，如果一个模板字符串由表达式开头，则该字符串被称为带标签的模板字符串，该表达式通常是一个函数，它会在模板字符串处理后被调用，在输出最终结果前，你都可以通过该函数来对模板字符串进行操作处理。在模版字符串内使用反引号（`）时，需要在它前面加转义符（\）。[1]
2. 箭头函数表达式的语法比函数表达式更短，并且不绑定自己的this，arguments，super或 new.target。这些函数表达式最适合用于非方法函数，并且它们不能用作构造函数。引入箭头函数有两个方面的作用：更简短的函数并且不绑定this。在箭头函数出现之前，每个新定义的函数都有它自己的 this值（在构造函数的情况下是一个新对象，在严格模式的函数调用中为 undefined，如果该函数被称为“对象方法”则为基础对象等）。This被证明是令人厌烦的面向对象风格的编程。箭头函数不会创建自己的this；它使用封闭执行上下文的this值。[2]
   * 1. *结果*

通过上面所述的请求方法，可以比较方便的通过调用网易云音乐后台API的方式进行数据的采集，免去了通过爬取网页来提取内容的麻烦，可以让爬虫以更快的速度进行数据的采集。但是，由于网易云音乐的后台设有严格的反爬虫机制，所以还要通过下面所说的方式来进行规避，防止爬虫被屏蔽。

* 1. 反爬虫策略

由于网易云的后台设置有反爬虫的机制, 所以不能爬取的太快, 否则IP会被封掉. 但是, 由于做分析需要的数据量较大,爬取的速度太慢无法再短时间内怕取到足够的数据. 因此必须采取一定的策略加快数据爬取的速度. 根据网易云的反爬虫机制, 单个IP地址一分钟最多发送20个请求. 因此, 这里采用多个代理并行爬取的方式. 同时, 在http请求头中, 随机化user-agent, 更好的把爬虫伪装成正常用户.

* 1. 代理程序

在搭建代理的时候, 由于我们的代理是为了隐藏爬虫, 所以必须使用http正向代理中的高匿代理, 否则会被网易云的后台发现异常. 这里使用的是自己使用Node.js编写的一个简单的匿名代理.

代理的原理是客户端先将请求发送到代理服务器, 由代理服务器向目标服务器发起请求, 并将请求结果发回客户端.

代码如下:

const PROXY\_PORT = 65535

const http = require('http')

const net = require('net')

const url = require('url')

function request(cReq, cRes)

{

const u = url.parse(cReq.url)

console.log(`[http.request] ${cReq.method} ${u.hostname}:${u.port || 80} ${u.path}`)

const options =

{

hostname : u.hostname,

port : u.port || 80,

path : u.path,

method : cReq.method,

headers : cReq.headers,

}

const pReq = http.request(options, function(pRes)

{

cRes.writeHead(pRes.statusCode, pRes.headers)

pRes.pipe(cRes);

}).on('error', function(e)

{

console.error(e)

cRes.end()

});

cReq.pipe(pReq)

}

function connect(cReq, cSock)

{

console.log(`[http.connect] ${cReq.url}`)

const u = url.parse('http://' + cReq.url);

const pSock = net.connect(u.port, u.hostname, function()

{

cSock.write('HTTP/1.1 200 Connection Established\r\n\r\n');

pSock.pipe(cSock);

}).on('error', function(e)

{

console.error(e)

cSock.end();

});

cSock.pipe(pSock);

}

http.createServer()

.on('request', request)

.on('connect', connect)

.on('error', function(err)

{

console.error(err)

})

.listen(PROXY\_PORT, function()

{

console.log(`proxy server online: http://localhost:${PROXY\_PORT}`)

});

* 1. Node.js基于事件循环的异步非阻塞并发机制

由于需要爬取的数据量较大, 传统的单线程同步程序耗时非常长, 所以必须采取并发机制. 这里采用的是Node.js基于事件循环的异步非阻塞并发机制. 在使用并发机制的时候, 需要使用互斥锁来确保程序有序运行.

## 数据的存储

* 1. 数据库

数据存储在自己搭建的MongoDB数据库中, 共五个collection, 总计N万条用户数据. MongoDB数据库是一种结构自由, 无schema的文档数据库, 便于存储这种对象化的用户数据.

## 数据的结构

* 1. 用户数据结构

1. user

{

"\_id":"5a51ba300604d374f28af6fb",

"profile":

{

"userId":363516402,

"gender":0,

"birthday":-2209017600000,

"city":220100

},

"\_\_processing":

{

"follow":false,

"followed":false,

"playlist":false,

"detail":false

},

"\_\_processed":

{

"follow":true,

"followed":true,

"playlist":false,

"detail":true

}

}

各个字段的含义如下:

1. \_id(数据库主键)

2. profile.userId(用户id)

3. profile.gender(用户性别, 0为男性, 1为女性)

4. profile.birthday(用户的出生日期)

5. profile.city(用户所在的城市)

6. \_\_processing(互斥锁, 用于并发)

7. \_\_processed(表示这个用户是否已经处理过了)

1. follow

用户的个人profile存储在follow这个collection中, 主要的结构如下:

{

"\_id":"5a4b488c954454ebecda66cb",

"from":"test",

"to":"test1",

"updatedAt":1515168662545

}

各个字段的含义如下:

1. \_id(数据库主键)

2. from(关注别人的那个人的userId)

3. to(被关注的那个人的userId)

4. updatedAt(这条记录的插入时间)

* 1. 待定

## 数据分析

* 1. 用户的总体分布情况

用饼状图来表示比例

4.1.1 性别分布情况

在表示用户个人信息的数据结构中, profile.gender表示用户的性别(0代表未知, 1代表男性, 2代表女性)

样本所包含的用户总数: 232433

查询性别未知的用户的数量: {'profile.gender': 0}, 结果为 39466

查询男性用户的的数量: {'profile.gender': 1}, 结果为 103419

查询女性用户的的数量: {'profile.gender': 2}, 结果为 89548

|  |  |  |
| --- | --- | --- |
| 性别不明 | 男性用户数量 | 女性用户数量 |
| 39466 | 102419 | 89548 |

4.2.2 年龄分布情况

查询70后的用户数量: {'profile.birthday': {$gte: 0, $lt: 315532800000}}

查询80后的用户数量: {'profile.birthday': {$gte: 315532800000, $lt: 631152000000}}

查询90后的用户数量: {'profile.birthday': {$gte: 631152000000, $lt: 946684800000}}

查询00后的用户数量: {'profile.birthday': {$gte: 946684800000, $lt: 1262304000000}}

查询10后的用户数量: {'profile.birthday': {$gte: 1262304000000}}

样本总人数: 88260

|  |  |
| --- | --- |
| 年龄段 | 人数 |
| 70后 | 849 |
| 80后 | 10873 |
| 90后 | 53504 |
| 00后 | 20019 |
| 10后 | 3015 |

* 1. 用户在不同省份的分布

用中国地图来表示不同省份的分布情况

4.2.1 数量分布情况

查询代码如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const db = await $conn

const user = db.collection('user')

const result = {}

for(const gb in regions)

{

const num = await user.find

(

{

'profile.province': Number(gb)

}

).count()

console.log(`| ${regions[gb]} | ${num} |`)

result[regions[gb]] = num

}

// console.log(result)

fs.writeFileSync(path.resolve(\_\_dirname, 'province.json'), JSON.stringify(result))

})()

结果如下表所示:

|  |  |
| --- | --- |
| 省份 | 用户数量 |
| 北京 | 11766 |
| 天津 | 2781 |
| 河北 | 5923 |
| 山西 | 3889 |
| 内蒙古 | 3526 |
| 辽宁 | 5265 |
| 吉林 | 2901 |
| 黑龙江 | 3730 |
| 上海 | 7317 |
| 江苏 | 14033 |
| 浙江 | 11514 |
| 安徽 | 7044 |
| 福建 | 5835 |
| 江西 | 5772 |
| 山东 | 11301 |
| 河南 | 10174 |
| 湖北 | 8216 |
| 湖南 | 7733 |
| 广东 | 23985 |
| 广西 | 4246 |
| 海南 | 1146 |
| 重庆 | 4387 |
| 四川 | 10242 |
| 贵州 | 2656 |
| 云南 | 5013 |
| 西藏 | 516 |
| 陕西 | 6274 |
| 甘肃 | 2798 |
| 青海 | 917 |
| 宁夏 | 969 |
| 新疆 | 5598 |
| 台湾 | 635 |
| 香港 | 832 |
| 澳门 | 196 |

使用D3.js可视化之后结果如下图所示:

4.2.2 性别分布情况

在基于所有的样本进行分析之后, 得到的不同性别的用户在全国各个省份的分布情况.

代码如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const db = await $conn

const user = db.collection('user')

const result = {}

for(const gb in regions)

{

const female = await user.find

(

{

'profile.province': Number(gb),

'profile.gender': 2

}

).count()

const male = await user.find

(

{

'profile.province': Number(gb),

'profile.gender': 1

}

).count()

console.log(`| ${regions[gb]} | ${male} | ${female} |`)

result[regions[gb]] = [male, female]

}

fs.writeFileSync(path.resolve(\_\_dirname, 'sex.json'), JSON.stringify(result))

})()

结果如下:

|  |  |  |
| --- | --- | --- |
| 省份 | 男性用户数量 | 女性用户数量 |
| 北京 | 5513 | 4058 |
| 天津 | 1249 | 1005 |
| 河北 | 2696 | 2197 |
| 山西 | 1688 | 1522 |
| 内蒙古 | 1581 | 1357 |
| 辽宁 | 2462 | 1951 |
| 吉林 | 1324 | 1092 |
| 黑龙江 | 1654 | 1467 |
| 上海 | 3223 | 2641 |
| 江苏 | 6319 | 5136 |
| 浙江 | 4848 | 4184 |
| 安徽 | 3233 | 2671 |
| 福建 | 2582 | 2065 |
| 江西 | 2547 | 2188 |
| 山东 | 5046 | 4204 |
| 河南 | 4523 | 3837 |
| 湖北 | 3729 | 3027 |
| 湖南 | 3314 | 3250 |
| 广东 | 11343 | 8195 |
| 广西 | 1956 | 1546 |
| 海南 | 574 | 383 |
| 重庆 | 1904 | 1725 |
| 四川 | 4558 | 3971 |
| 贵州 | 1195 | 1046 |
| 云南 | 2288 | 2005 |
| 西藏 | 232 | 200 |
| 陕西 | 2908 | 2433 |
| 甘肃 | 1274 | 1104 |
| 青海 | 438 | 357 |
| 宁夏 | 389 | 403 |
| 新疆 | 2696 | 2058 |
| 台湾 | 274 | 298 |
| 香港 | 382 | 358 |
| 澳门 | 96 | 76 |

使用d3.js进行可视化展示之后的结果如下:

1. 男性用户在全国的密度分布

2. 女性用户在全国的密度分布

3. 女性用户在所在的省份的用户中的比率在全国的分布

4.2.3 年龄分布情况

查询代码如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const db = await $conn

const user = db.collection('user')

const result = {}

for(const gb in regions)

{

const age70 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 0, $lt: 315532800000}

}

).count()

const age80 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 315532800000, $lt: 631152000000}

}

).count()

const age90 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 631152000000, $lt: 946684800000}

}

).count()

const age00 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 946684800000, $lt: 1262304000000}

}

).count()

const age10 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 1262304000000}

}

).count()

console.log(`| ${regions[gb]} | ${age70} | ${age80} | ${age90} | ${age90} | ${age10} |`)

result[regions[gb]] = [age70, age80, age90, age00, age10]

}

fs.writeFileSync(path.resolve(\_\_dirname, 'age.json'), JSON.stringify(result))

})()

结果如下:

|  |  |
| --- | --- |
| 省份 | 用户数量 |
| 北京 |  |
| 天津 |  |
| 河北 |  |
| 山西 |  |
| 内蒙古 |  |
| 辽宁 |  |
| 吉林 |  |
| 黑龙江 |  |
| 上海 |  |
| 江苏 |  |
| 浙江 |  |
| 安徽 |  |
| 福建 |  |
| 江西 |  |
| 山东 |  |
| 河南 |  |
| 湖北 |  |
| 湖南 |  |
| 广东 |  |
| 广西 |  |
| 海南 |  |
| 重庆 |  |
| 四川 |  |
| 贵州 |  |
| 云南 |  |
| 西藏 |  |
| 陕西 |  |
| 甘肃 |  |
| 青海 |  |
| 宁夏 |  |
| 新疆 |  |
| 台湾 |  |
| 香港 |  |
| 澳门 |  |

使用d3.js进行可视化展示之后的结果如下:

1. 70后在全国的分布

1. 80后在全国的分布

1. 90后在全国的分布

1. 00后在全国的分布

1. 10后在全国的分布

* 1. 用户之间相互关注的情况的对比分析

使用弦图来进行展示

男女之间互相关注的数量的对比

* 1. 采用floyd算法进行计算用户之间的距离

## 数据的可视化

## 结论

##### 参考文献

[1] https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/template\_strings [2] https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow\_functions

[7] 涂聪. 大数据时代背景下的数据可视化应用研究[J]. 电子制作, 2013, 47(5x):118-118. [8] 赵聪. 可视化库D3.js的应用研究[J]. 信息技术与信息化, 2015(2):107-109.