

✓ BiliSearchCST

Homework 2 for 2020-2021 Summer Course, CST.



- For information about the developer

写于 author.txt 内，已添加入 .gitignore 文件中。

用于网络学堂身份验证。

- For information about this repo

Would be made public after the final submission date of the homework and grading process.

If you are a student taking this course years later, please **DO NOT** copy any of these files directly into your homework. The developer will not be responsible for any possible loss of your work.

✓ 数据获取

数据获取方式的选择

我们决定摒弃掉简洁而美妙的 API 接口这种方法，而使用 selenium 爬取 + beautifulsoup4 解析网页 HTML 数据。我们在 Crawler 文件夹下整合了 `videoInfo.py`, `upInfo.py` 两个文件，其中均含有一些可以在控制台运行的命令。

- `run <id>` 对 ID 的视频/UP主进行一次爬取
- `generate` 生成爬取数据的 Bash 文件
- `verify` 校验数据爬取是否完成，或是否符合要求

我们在下述的数据范围内，首先对所有的视频进行一次尝试爬取。然后，对于所有有返回结果的视频，我们再通过解析拿出其作者的 ID，然后对其作者再进行一次爬取。总计而言，成功的数据爬取量（含视频与作者）有 10k 条左右，耗时大概为 15h.

数据范围与规模

视频数据

B 站在 2020/03，也就是其视频总上传量即将到达 100M 的时候，推出使用了 BV 号以代替原有的自增主键 AV 号的形式。这里我们为了数据爬取的简便性、易控制性与可操作性，决定对 AV 号小于 100M 的视频进行分段式爬取。

我们采用了【分段式】的爬取方式，即对特定的分段进行一定数据量的爬取。准确的说：

数据范围 VID (M)	代表月份	尝试爬取的数据量
1	2014/03	1100
3	2015/10	1499
5	2016/06	1499
8	2017/01	1499
10	2017/04	1499
20	2018/02	1499
30	2018/08	1499
50	2019/04	1499
70	2019/10	1499

数据范围 VID (M)	代表月份	尝试爬取的数据量
90	2020/02	1499

事实上我们之后会看到，每个【分段】最终成功爬取到的数据量是不同的，而每个【分段】时间的通过率在大致上却是相同的。（见【数据分析】【结论1】）

最终，我们成功获取的数据量为：

分类	条目数量
视频	5670
UP 主	4804
视频标签	39513
视频评论	14971

数据存储格式

我们将获取后的数据按照 Json 格式来进行存储，而标准的数据的 Json 格式定义如下：

```
{
  "id": 1005605,
  "title": "【真赤】四十七【无力的翻唱了】",
  "abstract": "本家sm22626781重投。压制感谢奏奏。大家好久不见，也是很久没投稿了。这首曲子是两个月前就录好了，到现在才想起来投稿。声音太懒就别吐槽啦哈哈哈！还是多亏了女朋友的鼓励才录完了，我没晒，我真没晒。感谢听完的大家。[稍微高了一点的音质mp3请戳http://fc.5sing.com/11935486.html]",
  "authorId": 181529,
  "uploadTime": 1394547769.0,
  "playCount": 567,
  "commentCount": 4,
  "bulletCommentCount": 52,
  "imageUrl": "http://i1.hdslb.com/bfs/archive/6654eb449cd1dfd6ce54b2aa2dd71dcfa0c08658.jpg",
  "feedback": {
    "like": "0",
    "coin": "2",
    "star": "2",
    "share": "0"
  },
  "comments": [
    "这次音质没有以前的好了哎...不过还是很萌ww不过还是要举起火把",
    "\\\真酱\\\真酱\\\真酱/",
    "声音很好听，感觉好像调音和录制的不太好吧",
    "（▽▽）挺好看的"
  ],
  "tags": [
    "音乐",
    "翻唱",
    "四十七",
    "真酱"
  ]
}
```

```
"MIKITOP"
```

```
]  
}
```

```
{  
    "id": 1578,  
    "username": "雜草",  
    "signature": "你站不讓波了，有事上推，私信",  
    "level": 5,  
    "avatarUrl":  
        "https://i0.hdslb.com/bfs/face/e0bfc0c7b5a8ea10ed51567fdb6e50ff64665893.png",  
    "followerCount": "5433",  
    "followingCount": "28"  
}
```

✓ 数据呈现

我们使用 Django 来进行前后端管理与呈现。

前端渲染

效果呈现图

视频列表页面



The screenshot shows a list of video posts from various users. Each post includes a thumbnail image, the title, a brief description, the upload date, view count, comment count, like count, and share count.

- [某炮S] 论BGM (Santorini)
自制论BGM，战道浪失败
2014-03-11 22:37:25 ▶ 1313 ⚡ 1 ● 40 ★ 8
- 【mhp3】弓箭道具流 禁基♂的巨大VS沙漠暴君
战道浪失败。。。Orz
2014-03-11 22:37:54 ▶ 898 ⚡ 0 ● 10 ★ 5
- [MMD] 1080P 电光闪闪的 Sweet Devil
自制 新人自制，渣技术初作。为战道浪不得已13倍速，请稍等缓冲。渲染：AutoLuminous4, Cook-Torrance(024改2), o...
2014-03-11 22:43:28 ▶ 4582 ⚡ 34 ● 89 ★ 265

Navigation and footer information:

- Pagination: « 1 2 3 ... 566 567 »
- Jump to page: 跳转至 _____ 页
- Copyright: © c7w 2021. All rights reserved.
- Powered by Django & Bootstrap

UP主列表页面

The screenshot shows a detailed list of videos uploaded by a specific user. Each video entry includes a thumbnail, the title, a brief description, the upload date, view count, comment count, like count, and share count.

- saber酱
个人微博: http://weibo.com/sabерchanchan/
等级 Lv. 6 ⚡ 粉丝数 17.6万 ⚡ 关注数 0
- 上甘玲Official
现接歌词翻译(漫画汉化插画绘制, 详情私聊/头像是花园腰/音质问题欢迎私信/夜でも
こにちは! 杂食系游戏&绘画主播, 上甘玲です!
等级 Lv. 6 ⚡ 粉丝数 4216 ⚡ 关注数 119
- 大佐
你知道吗? 我几乎几天就会有不认识的人@我, 只因为我的ID够简单// 微博 @地球防卫大佐mk4
等级 Lv. 6 ⚡ 粉丝数 2230 ⚡ 关注数 59
- 雜草
- shiiinamafuyu
等级 Lv. 6 ⚡ 粉丝数 317 ⚡ 关注数 58
- fumo
过气up主 联机群307837974
等级 Lv. 6 ⚡ 粉丝数 2.7万 ⚡ 关注数 98
- Mr.Quin
新浪微博: @Mr_Quin http://weibo.com/mrquin33
等级 Lv. 6 ⚡ 粉丝数 50.8万 ⚡ 关注数 22

Navigation and footer information:

- Pagination: « 1 2 3 ... 480 481 »
- Jump to page: 跳转至 _____ 页
- Copyright: © c7w 2021. All rights reserved.
- Powered by Django & Bootstrap

UP主详情页面



视频详情页面

精选评论...

- 超喜欢这首!
- 此时此刻Hanser的三个流水y都在直播
- 建议你2p放一个开始谆谆教诲~~~快的 学的超像
- 曲(原词: YUNOMI译词: ST唱) Hanser湛蓝的海水漾出的梦好像那暖洋洋的天空游荡在孤岛上空的云朵嬉笑无助的你我左摇右晃的气垫船也无法搁浅怎么知道今生只能困在这里静感受这轻柔的海风独自一人听着浪声感受阳光洒满沙滩向我走来我脸颊羞得通红写下大大的SOS顾自许下一起离开的Promise在这只有二人的孤岛上度过没有终止的Holiday想要与你想要与你共舞一曲在南国的岛屿在盛夏夜晚里夜以继日以继夜不停地私语伴着你声音呼吸你的吐息Dancefloor用单恋结下的果实或许会是苦涩的甘甜的乘船在月下魔法的梦想留下浅尝的痕迹快要搁出的心跳声振动了明月夜的星辰被玻璃夜空照耀着的我们 背靠着背无声偶尔划过的那一颗颗流星点缀了梦境期待着这份爱恋某天能传达给你明月洒下了皎洁的Moonlight照亮沙滩留下一片Whitewhite忽然间想起曾经小时候惧怕黑暗的Lonelynight哪怕最终没有结果也好哪怕什么都没有说也好可是我却无论如何都想要独占你温暖怀抱想要与你想要与你共舞一曲这南国的风息消失在盛夏夜里夜以继日以继夜不停地私语那单恋的秘密却深藏在心底Dancefloor用单恋结下的果实或许会是苦涩的甘甜的乘船在月下魔法的梦想留下轻咬的痕迹或许最后我的恋爱会是那苦涩的一场空一场梦但我仍想在最后的梦里尝尽仅剩的甜美
- 喜欢喜欢! (=・ω・=)

搜索界面

Bili Search CST

视频 Search...

搜索完成, 共计 1383 条搜索结果, 用时 0.527316 秒.

< 返回 Bilibili 前往 UP 主列表 >

【真赤】四十七【无力的翻唱了】

本家sm22626781 重制, 压制感谢奏麦. 大家好久不见, 也是很久没投稿了. 这首曲子是两个月前就录好了, 到现在才想起来投稿. 声音太懒就别吐槽啦哈哈哈...

2014-03-11 22:22:49 ▶ 567 ● 4 ● 52 ★ 2

【韦诺之战】-试录教学关卡

自制 试录, 看大家有没有兴趣吧, 有兴趣的话我会试着做低难度的战役

2014-03-11 22:53:25 ▶ 585 ● 12 ● 22 ★ 3

东方Sky Arena 幻想乡空战姬

自制 求大神教用Fraps啊, 为毛我一撮就是朱军画质, 怎么压制啊QAQ！！！

2014-03-11 22:53:58 ▶ 934 ● 1 ● 6 ★ 1

【茜茜】恋爱循环 (Full ver)

Bili Search CST

视频 Search...

搜索完成, 共计 66 条搜索结果, 用时 0.063997 秒.

< 前往视频列表 返回 Bilibili >

舍长驾到

这家伙很懒惰, 但还是勉强留下了他的新浪微博: 舍长驾到. 商务合作钉钉号: darcy96

等级 Lv. 6 ● 粉丝数 69.7万 ● 关注数 46

加班第一帅

天凤&雀魂玩家, 立直麻将解说, 时不时玩其他游戏. 加班群: 790240695

等级 Lv. 6 ● 粉丝数 5837 ● 关注数 85

珠峰喵

珠峰很瘦很冷的, 乃们不要销他www 经常用 (假社) 的微博: @珠峰喵, QQ群: 377327358

等级 Lv. 6 ● 粉丝数 5.3万 ● 关注数 395

Bili Search CST

视频 Search...

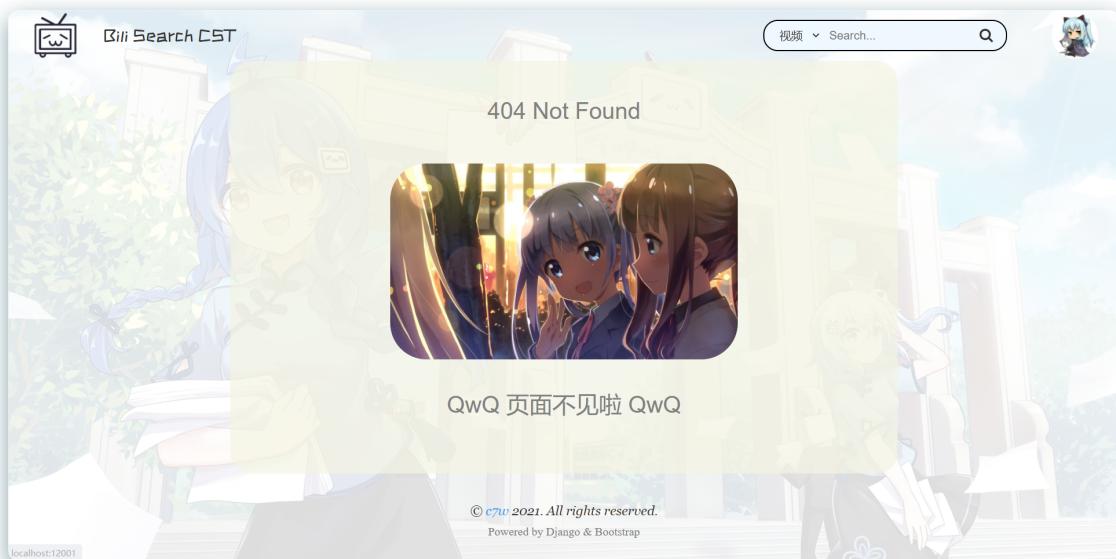
搜索完成, 无搜索结果, 用时 0.004996 秒.

< 前往视频列表 返回 Bilibili >

无可用搜索结果

© c7w 2021. All rights reserved.
Powered by Django & Bootstrap

404 Not Found



基于模板的 Django 渲染

页面的 `header` 元素和 `footer` 元素在每次渲染网页时都是相同的，这是因为使用了基于模板的 Django 渲染功能。

具体而言，`Base.html` 中通过 `<main>{% block main %} {% endblock main %}</main>` 语句实现了其他文件的 HTML 可以直接插入 `main` 块中。

前端美化

分页器的美化使用了 Bootstrap 提供的 CSS 文件。

`global.js` 中主要实现了对于输入框回车事件的处理，对于搜索框搜索事件，以及分页器切换页面事件的处理。

各种 HTML 和 CSS 文件中广泛使用了盒模型的概念，各种以 `-container` 结尾的 `<div>` 块均使用 `display: flex;` 作为样式。

后端数据管理

数据处理模型（Models）

`DataType` 其中的 * 表示建立为索引。

Up

Field	DataType
<code>id</code>	<code>Integer*</code>

Field	DataType
username	Text*
signature	Text*
level	Integer
avatarUrl	Text
followerCount	Text
followingCount	Text

Video

Field	DataType
id	Integer
title	Text
abstract	Text
authorId	Integer
author	Foreign Key
uploadTime	DateTime
playCount	Integer
commentCount	Integer
bulletCommentCount	Integer
imageUrl	Text
like	Text
coin	Text
star	Text
share	Text

Comment

Field	DataType
id	BigInteger Auto *
video	Foreign Key

Field	DataType
content	Text*

Tag

Field	DataType
id	BigInteger Auto *
video	Foreign Key
tagName	Text*

Json 数据导入 sqlite

在访问 `/merge` 链接时，便会自动执行 json 数据导入 sqlite。具体而言，是把 `/root/to/your/website/ResultPool/` 中 `/UpResult` 的内容先导入到 `Up` 表中，再把 `/VideoResult` 的内容拆分后导入到 `Video`，`Tag`，`Comment` 表中。

搜索功能

我们使用了 `django.db.models.query_utils` 提供的 `Q` 对象，以实现对 `title` 和 `abstract`，以及 `username` 和 `signature` 的取并集操作。

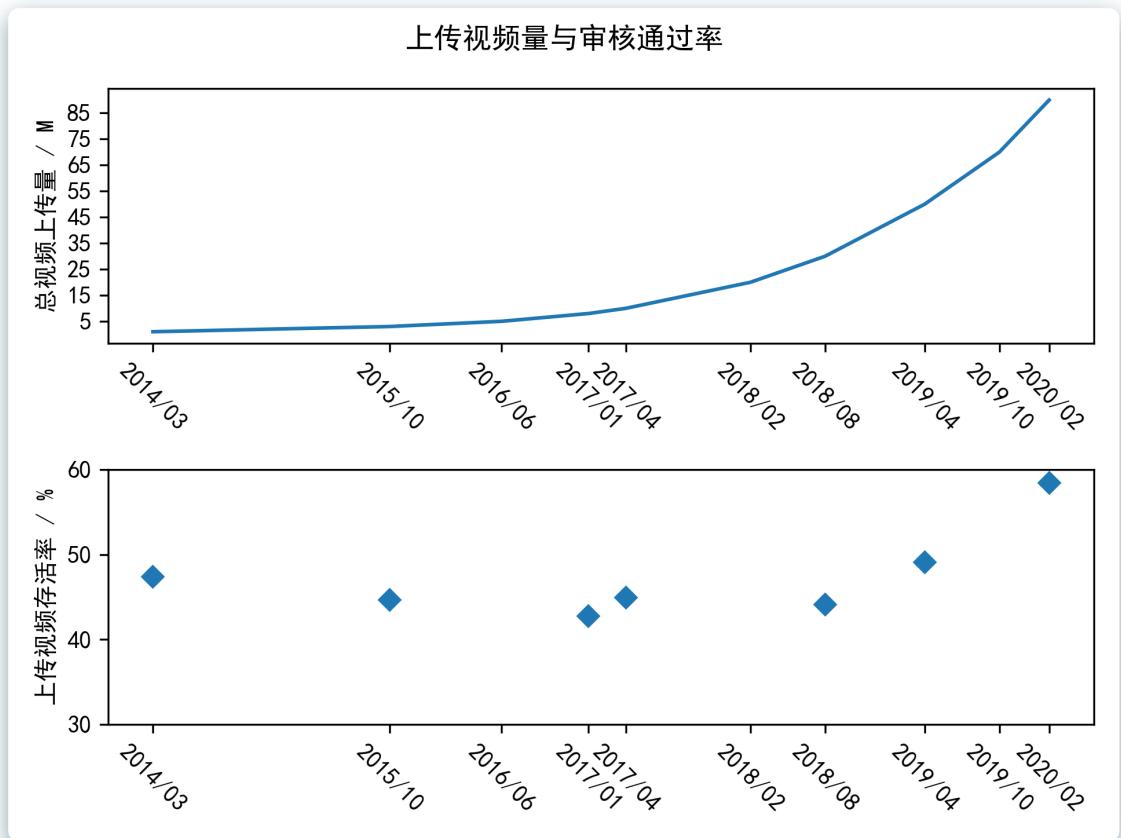
整体的搜索功能流程如下：首先使用 `Q` 对象进行 `filter`，然后使用自行编写的分页器类 `Pagination` 对搜索结果进行分页处理。

✓ 数据分析

综合数据的选取，数据的有效性，数据内容等等因素，在 Python 相关数据分析工具的帮助下，我们可以得出以下结论。

结论 1

近年来 B 站可播放视频总量增长趋势逐年加快



图表说明

可以看到，近年来总视频上传量成指数级增长趋势；而视频上传后，其通过率在一定范围内维持相对稳定；因此根据【可播放视频量 = 上传量 * 通过率】，我们得出结论，近年来 B 站的可播放视频总量的增长趋势逐年加快。

结论来源分析

在【数据获取】节的【数据范围与规模】中我们已经提及到，我们采用了分段式的数据获取模式，即在不同的 AV 号区段内尝试获取一定规模的数据。根据我们获取到的数据总数与尝试获取规模数的比值，我们可以在一定程度上推断当时的视频审核通过率（图2）。对于每一个视频【分段】，由于 AV 号是连续的，上传时间也大致相同，我们可以对每一个分段取平均 ID 作为其 ID，平均上传时间作为其上传时间。然后我们描点绘制出折线图（图1），便可以得到视频上传总量随时间变化的大致趋势。

代码实现

```

def Conclusion1():
    idList = np.array(videos['id'])
    timeList = list(videos['uploadTime'])
    timeList = np.array(list(map(lambda x: x.timestamp(), timeList)))
    idRepresent = idList // 1000000

    # Paint Graphs Preparation
    x = []
    x_label = []
    y = []
    y2 = []

```

```

# Calc average time for each uniqueId
idUnique = np.unique(idRepresent)
for tid in idUnique:
    mask = (idRepresent == tid)
    averageTime = np.dot(mask, timeList) // mask.sum()
    x.append(datetime.datetime.fromtimestamp(averageTime) -
              datetime.datetime.fromtimestamp(1388505600).days // 30) # 2014-1-1
00:00:00
    x_label.append(datetime.datetime.fromtimestamp(
        averageTime).strftime('%Y/%m'))
    y2.append(mask.sum())

y = idUnique
y2 = np.array(y2) / np.array([1100, 1499, 1499,
                             1499, 1499, 1499, 1499, 1499, 1499]) * 100

# Calc x-id and x-label
# Start 2014.01 <-> 1, with month as unit
# print(x)
# print(y)
# print(x_label)
plt.rcParams['savefig.dpi'] = 300
plt.suptitle('上传视频量与审核通过率')

# Draw graph 1
plt.subplot(2, 1, 1)
plt.plot(x, y)

# Chinese tags support and minus symbol, reference
https://blog.csdn.net/qq\_40563761/article/details/102989770
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.ylabel('总视频上传量 / M')

plt.xticks(x, x_label, rotation=315)
plt.yticks(np.arange(5, 95, 10))

# plt.savefig('1-1.png')
# plt.close()

# Draw graph 2
plt.subplot(2, 1, 2)
plt.plot(x, y2, 'D')

plt.ylabel('上传视频存活率 / %')

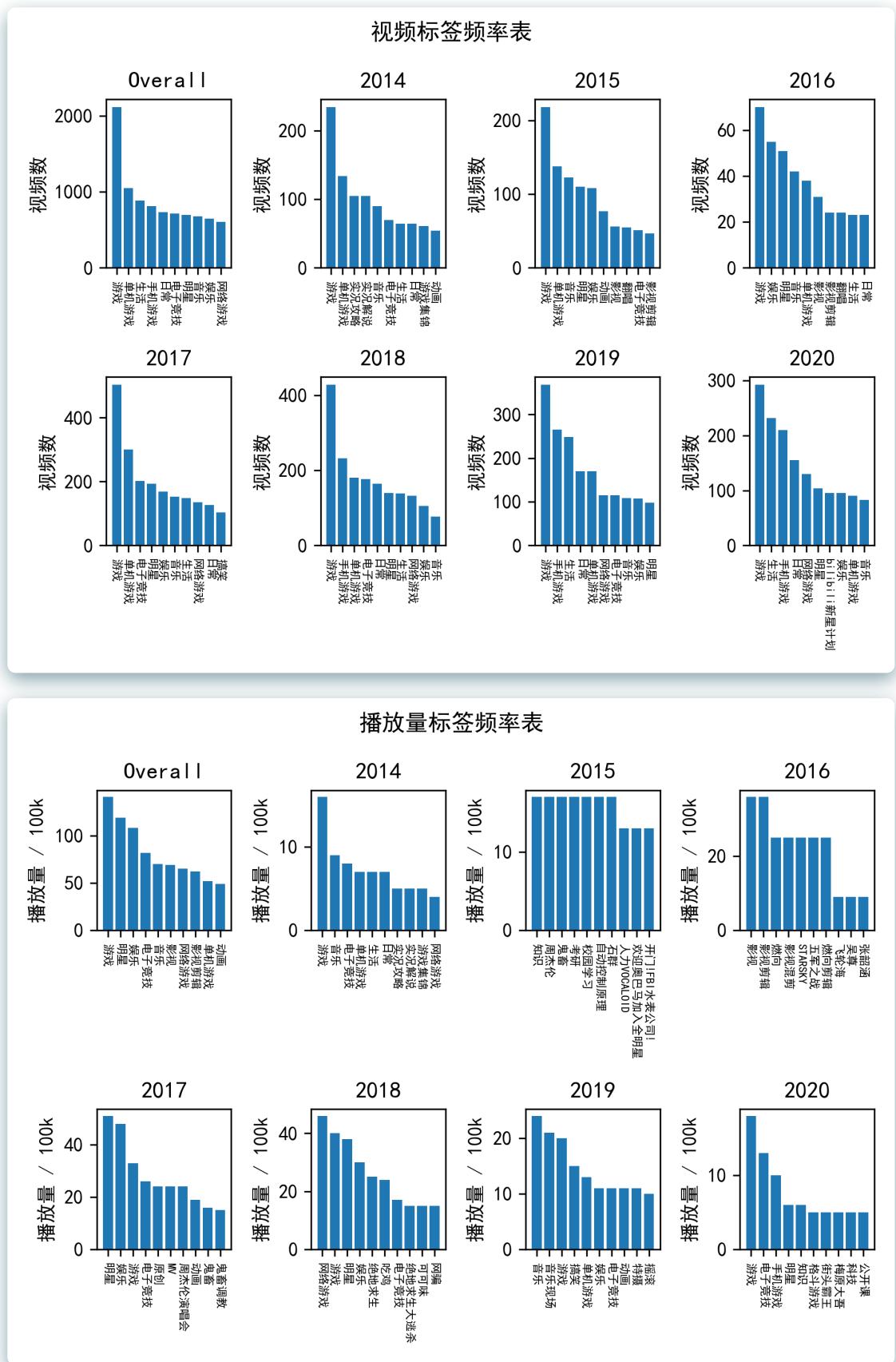
plt.xticks(x, x_label, rotation=315)
plt.ylim(30, 60)

# Save Figure
plt.tight_layout()
plt.savefig('1.png')
plt.close()

```

结论 2

游戏持续是 B 站视频总量与播放量的杰出贡献者，而 B 站在多元化的道路上不断迈进



图表说明

我们给出了所有视频标签的频率表（图1）与所有播放量的标签的频率表（图2）。可以看到，游戏始终在各个年份的频率表中处于高位，为B站贡献了大量的视频与播放量。而从每播放量的标签频率分布来看，知识、影视、明星等等播放量也不时深居榜首，这说明B站在走向多元化的道路上不断前进。

结论来源分析

在爬取视频时，我们已经收集了每个视频的标签。我们将不同视频首先按照年份归类，然后按照每视频（每视频对其下属所有标签的贡献权重为1）、每播放量（每视频对其下属所有标签的贡献权重为其播放量）两种方式，分别统计不同标签的权重大后，进行一次降序排序。然后按照排序后的结果切片出前十名最受欢迎的标签后，形成图像。

代码实现

```
def Conclusion2():
    frequencies = {} # 2014~2020, all
    frequencies['all'] = {}
    frequenciesWithHits = {} # 2014~2020, all
    frequenciesWithHits['all'] = {}
    # print(frequencies)

    for index, row in videos.iterrows():
        id = (row['id'])
        uploadYear = str(row['uploadTime'].year)
        hits = row['playCount']

        # Find tags related to id
        relatedTags = tags[tags['video_id'] == id]
        for index, tagRow in relatedTags.iterrows():
            tagName = tagRow['tagName']

            # Push Tagname to frequencies list
            if uploadYear not in frequencies:
                frequencies[uploadYear] = {}
                frequenciesWithHits[uploadYear] = {}
            if tagName not in frequencies[uploadYear]:
                frequencies[uploadYear][tagName] = 0
                frequenciesWithHits[uploadYear][tagName] = 0
            if tagName not in frequenciesWithHits['all']:
                frequenciesWithHits['all'][tagName] = 0
            if tagName not in frequencies['all']:
                frequencies['all'][tagName] = 0

            frequencies[uploadYear][tagName] += 1
            frequencies['all'][tagName] += 1
            frequenciesWithHits[uploadYear][tagName] += hits
            frequenciesWithHits['all'][tagName] += hits

    # Carry out sort in frequencies dict and truncate top 10 tags
    for key in frequencies.keys():
        frequencies[key] = sorted(
            frequencies[key].items(), key=lambda x: x[1], reverse=True)[0:10]
    for key in frequenciesWithHits.keys():
        frequenciesWithHits[key] = sorted(
            frequenciesWithHits[key].items(), key=lambda x: x[1], reverse=True)[0:10]
```

```

# print(frequencies)
# print(frequenciesWithHits)

# Draw Graph 1
plt.rcParams['savefig.dpi'] = 300
plt.suptitle('视频标签频率表')
k = 0
for key in frequencies.keys():
    k += 1
    plt.subplot(2, 4, k)
    if key == "all":
        plt.title('Overall')
    else:
        plt.title(key)
    plt.bar(np.arange(1, 11, 1), list(
        map(lambda x: x[1], frequencies[key])))
    plt.xticks(np.arange(1, 11, 1), list(
        map(lambda x: x[0], frequencies[key])), rotation=270, fontsize=6)
    plt.ylabel('视频数')

# Save Figure
plt.tight_layout()
plt.savefig('2-1.png')
plt.close()

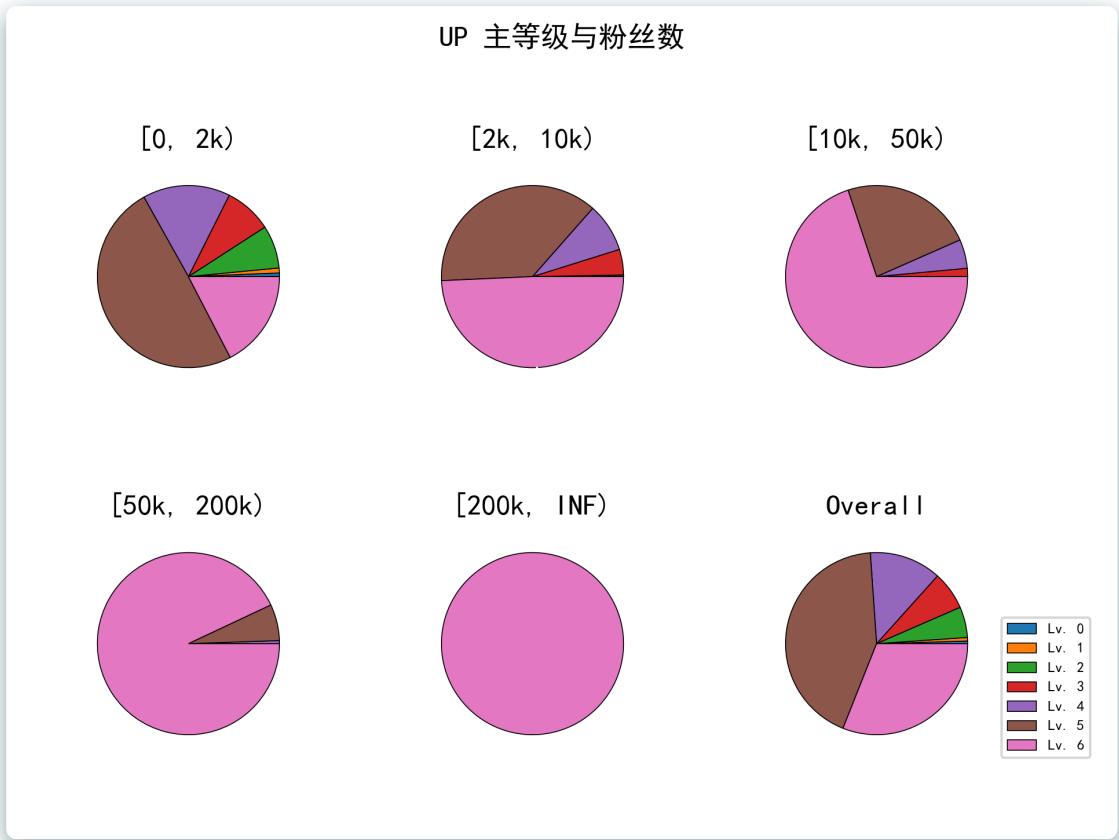
# Draw Graph 2
plt.rcParams['savefig.dpi'] = 300
plt.suptitle('播放量标签频率表')
k = 0
for key in frequenciesWithHits.keys():
    k += 1
    plt.subplot(2, 4, k)
    if key == "all":
        plt.title('Overall')
    else:
        plt.title(key)
    plt.bar(np.arange(1, 11, 1), list(
        map(lambda x: x[1]//100000, frequenciesWithHits[key])))
    plt.xticks(np.arange(1, 11, 1), list(
        map(lambda x: x[0], frequenciesWithHits[key])), rotation=270, fontsize=6)
    plt.ylabel('播放量 / 100k')

# Save Figure
plt.tight_layout()
plt.savefig('2-2.png')
plt.close()

```

结论 3

粉丝数越多的 UP 主，等级一般就越高



图表说明

(图1) ~ (图5) 的标题代表 UP 主的粉丝数所处的范围，而对应的饼图则代表在这个区段的 UP 主的等级频数分布。可以看到，随着 UP 主粉丝数的逐渐升高，处于低等级的用户的占比逐渐下降，而处于高等级的用户的占比逐渐增加。也就是说，一个 UP 主的粉丝数越多，ta 的等级一般也就越高。

结论来源分析

在爬取 UP 主信息时，我们已经获取了每个 UP 主的等级。于是我们可以在 UP 主的数据表中进行信息提取。首先，我们根据 UP 主的粉丝数所在范围，对 UP 主进行分段，最终分成了图中所呈现的 5 个区间。然后，我们再统计每个区间中的 UP 主的等级频数，并分别绘制成饼图。

代码实现

```

def Conclusion3():

    # Get author follower rank
    def getRank(n):
        checkpoint = [0, 2000, 10000, 50000, 200000, 1145141919810]
        for i in range(1, 6):
            if (checkpoint[i-1] <= n < checkpoint[i]):
                return i
        return -1

    LevelFrequencies = {}
    LevelFrequencies['all'] = [0, 0, 0, 0, 0, 0]

```

```

for index, row in ups.iterrows():
    lvl = row['level']
    rawFollowerCount = row['followerCount']
    if rawFollowerCount[-1] != '万':
        follower = int(rawFollowerCount)
    else:
        follower = int(float(rawFollowerCount[:-1]) * 10000)
    rank = str(getRank(follower))

    # Push to dict
    if not rank in LevelFrequencies:
        LevelFrequencies[rank] = [0, 0, 0, 0, 0, 0, 0]
    LevelFrequencies[rank][lvl] += 1
    LevelFrequencies['all'][lvl] += 1

# print(LevelFrequencies)

# Paint Figure
plt.rcParams['savefig.dpi'] = 300
plt.suptitle('UP 主等级与粉丝数')

i = 0
for key in sorted(LevelFrequencies.keys()):
    i += 1
    plt.subplot(2, 3, i)

    # Set subfigure title
    if key == 'all':
        plt.title('Overall')
    elif key == '1':
        plt.title('[0, 2k]')
    elif key == '2':
        plt.title('[2k, 10k]')
    elif key == '3':
        plt.title('[10k, 50k]')
    elif key == '4':
        plt.title('[50k, 200k]')
    elif key == '5':
        plt.title('[200k, INF]')

    labels = ['Lv. ' + str(i) for i in range(0, 7)]
    plt.pie(LevelFrequencies[key], wedgeprops={
        'edgecolor': 'black',
        'linewidth': '0.4'
    }, textprops={'color': 'white'}, labels=labels)
    plt.legend(prop={'size': 6}, bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)

# Save Figure
plt.tight_layout()
plt.savefig('3.png')
plt.close()

```

✓ 遇到的问题与心得

Django 数据库批量创建记录

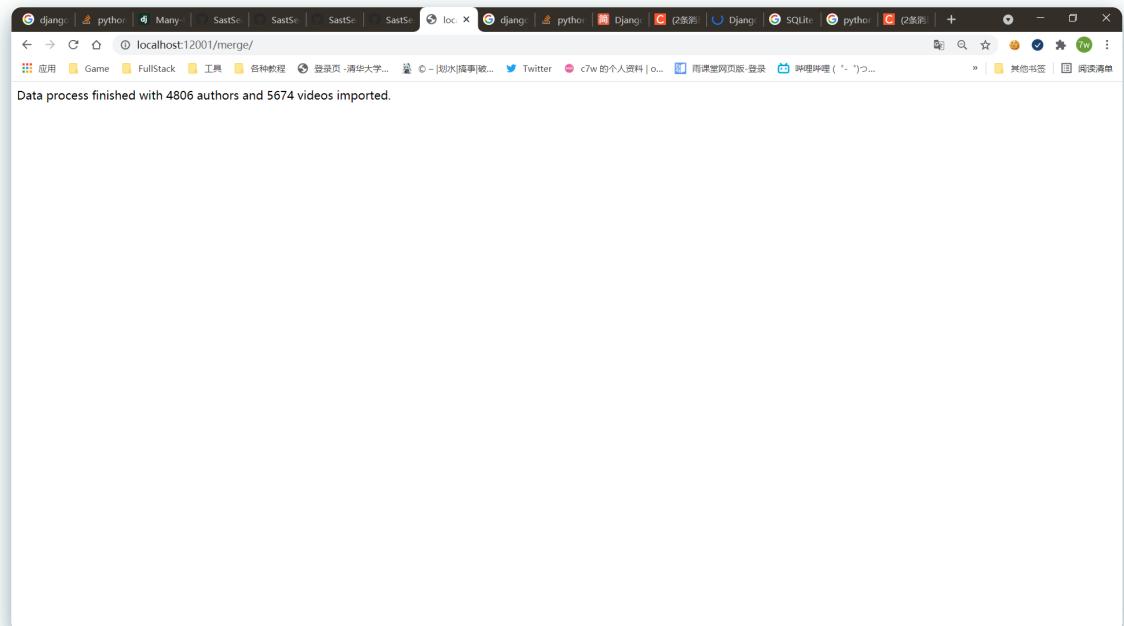
比起 for 循环创建 object 然后使用 `object.save()`，我们在这里面对规模足够大的数据量，应该先创建一个 `object_list`，每创建一个 object 便将其 append 到 `object_list` 中，最后使用 `bulk_create()` 函数来完成记录的批量创建，这样可以显著提高数据库 I/O 的效率。

```

C:\Windows\system32\cmd.exe - python manage.py runserver 12001
[...]
# up.css          # 404.css      # videolist.css
processor.py > MergeData
process_authors
    #1 = os.listdir('./ResultPool/upResult')
    for fileName in upl:
        ti = []
        ti.append(time.time())
        f = open("./ResultPool/upResult/" + fileName, 'r', encoding='utf-8')
        j = json.load(f)
        f.close()
        ti.append(time.time())
        author = upl[j['id']]
        author.username=j["username"]
        author.signature=j["signature"]
        author.level=j["level"]
        author.avatarUrl=j["avatarUrl"]
        author.followCount=j["followerCount"]
        author.followingCount=j["followingCount"]
        ti.append(time.time())
        author.save()
        ti.append(time.time())
        authorCount += 1
        os.remove("./ResultPool/upResult/" + fileName)
        ti.append(time.time())
        print(ti[1]-ti[0], ti[2]-ti[1], ti[3]-ti[2], ti[4]-ti[3])
    [...]
    # process videos
    vfl = os.listdir('./ResultPool/crawlResult')
    for fileName in vfl:
        f = open("./ResultPool/CrawlResult/" + fileName, 'r', encoding='utf-8')
        j = json.load(f)
        f.close()
        ti.append(time.time())
        video = crawlResult(j['id'])
        video.title=j["title"]
        video.abstract=j["abstract"]
        video.uploadDate=j["uploadDate"]
        video.playCount=j["playCount"]
        video.commentCount=j["commentCount"]
        video.bulletCount=j["bulletCount"]
        video.imageUrl=j["imageUrl"]
        video.feedback=j["feedback"]
        [...]
        video.comments=j["comments"]
        video.tags=j["tags"]
        video.like=j["like"]
        video.coin=j["coin"]
        video.star=j["star"]
        video.share=j["share"]
        [...]
        video.save()
        ti.append(time.time())
        videoCount += 1
        os.remove("./ResultPool/CrawlResult/" + fileName)
        ti.append(time.time())
        print(ti[1]-ti[0], ti[2]-ti[1], ti[3]-ti[2], ti[4]-ti[3])
    [...]
    print("Data process finished with %d authors and %d videos imported." % (authorCount, videoCount))
    [...]

```

(图1：在使用 `bulk_create` 之前，单条数据的插入十分耗时，需要 0.5 sec per record on average)



(图2：使用了 `bulk_create` 之后，数据在数秒之内，很快就处理完成了。)

Reference: <https://stackoverflow.com/questions/41051050/slow-saving-to-django-database>

为什么前端还是这么丑？



First edited by c7w, 2021-9-3 00:10:32.