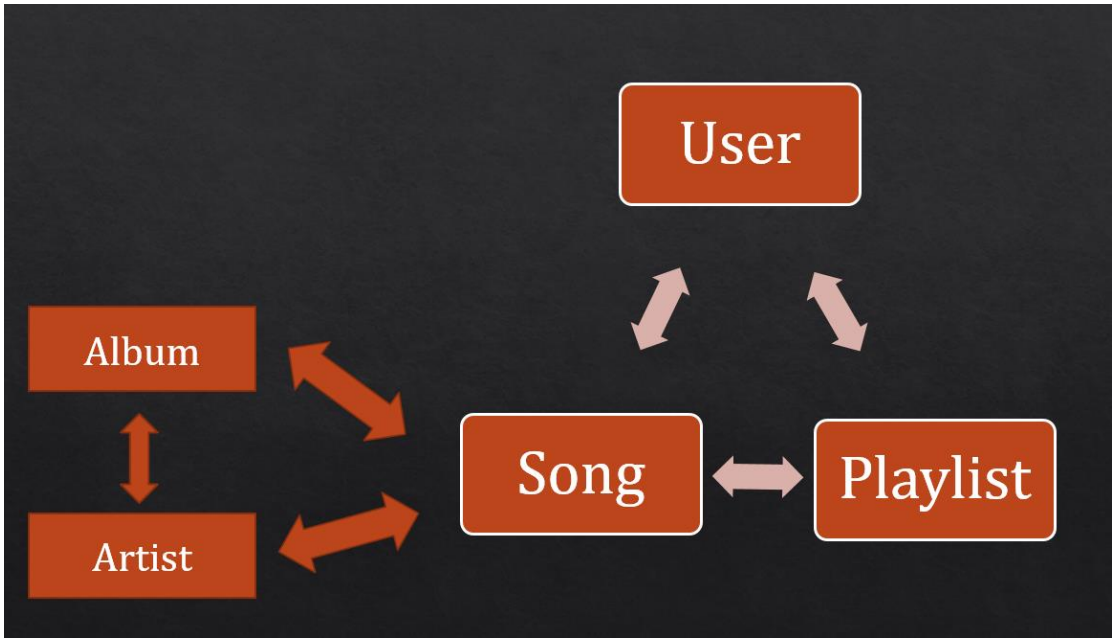
网易云音乐爬虫及推荐算法

第一部分 爬虫

因为使用的推荐算法是基于用户数据的，所以需要爬取大量用户数据进行计算。在实践中，改进的爬虫可以稳定地以每个 IP 每秒 200 个网页的速度爬取，并保证在较长的时间内不会被服务器阻止，最后爬取了 3,000,000 歌曲信息。

基本的网页关系如下图，主要爬取歌曲信息（包括基本信息、歌词、评论），用户创建的歌单（基本信息、评论），歌曲专辑（介绍），歌手（介绍）。爬虫程序的结构基本一致，均采取了负反馈、断点续爬等功能增强程序的鲁棒性。



负反馈调整爬取间隔时间

爬虫利用爬取结果的特征判断结果是否合理，并将此结果反馈到爬虫爬取频率中。定义最小等待时间 α，成功率 β，时间等待时间 t，其中：

$$t = \alpha e^{\min(\frac{1}{\varepsilon+\beta}, ln20)}$$

ε为较小的数可取 0.0001，防止 β=0 的特殊情况。等待时间 t 对 β 极为敏感，从而保证服务器开始拒绝时爬虫敏感地发现并伪装。成功率在每爬取 100 个网页时进行更新，并将变量成功爬取、失败爬取置为零。

在实际应用中，如爬取专辑或歌单的歌曲列表时，若服务器直接拒绝会返回 403error 或者返回空字符串，这是很容易用 try-except 和判断字符串是否为空解决的。但服务器一个隐蔽的欺诈是只返回一首歌的 ID，而非所有歌曲。考虑到我们爬取歌单的主要目的是推荐音乐，所以检索返回歌曲列表的长度，若小于等于一则认为访问失败。

网易云音乐会在同一 API 使用约 6 万次时开始拒绝，如果仍以相同频率爬取，将迅速停止对此 IP 的服务。上述方案很好地解决了这一问题。

管理 URL 并降低时间复杂度

随机访问部分 URL，防止内存溢出

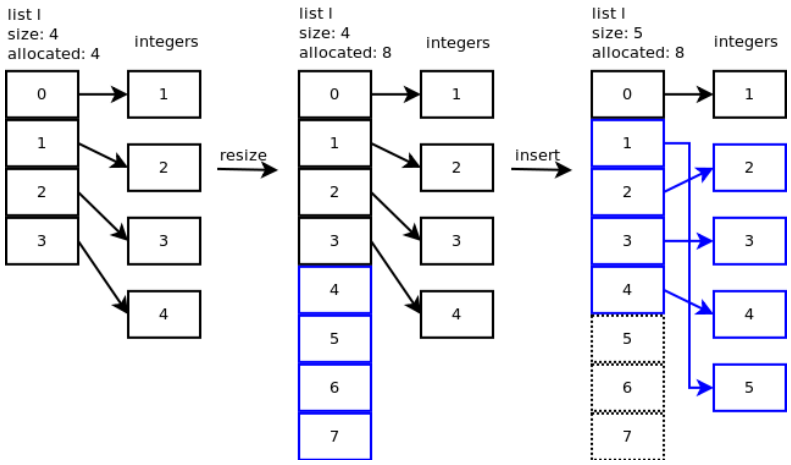
由于网易云音乐网页中，每个网页都会连接约 30 个其他不同的网页，所以在实际访问时，如果保留所有网页进行 BFS 内存会很容易溢出。

定义访问因子 alpha = 1.0 - (float(len(urlSet))/maxUrl)**2，如果 random.random()，则将该网页加入待访问的网页，这样可以使 urlSet 中网页稳定在 0.618* maxUrl，保证 URL 不会溢出。

修改函数 union_bfs，使 bfs 增广复杂度从 O(n²)变为 O(n)

由于 insert 调用会调用 python 内的 ins1 函数，它会在扩大 list 容量后，修改所有之前从插入点开始有元素位置的“指针”，让它指向原来的上一个元素（除了插入点），而后加入插入点的元素。这样耗费的时为 O(n²)。

```
INS1:
    RESIZE LIST TO SIZE N+1 = 5 -> 4 MORE SLOTS WILL BE ALLOCATED
    STARTING AT THE LAST ELEMENT UP TO THE OFFSET WHERE, RIGHT SHIFT EACH ELEMENT
    SET NEW ELEMENT AT OFFSET WHERE
    RETURN 0
```



而我采用新建一个 list，并将所有元素加入，再将原 list（a）进行 reverse，随后连接两个 lists，再将合并后的 list reverse。这样耗费的时间是 O(n)。

利用哈希表，使查重复杂度从 O(n)变为 O(1)

在网易云项目中多次用到元素的查重，包括数据处理、爬虫中 urlSet 管理、爬去结果处理等等，通常采用复杂度为 O(n)的 for-in 结构，这大大影响了爬虫的速度。于是采取了哈希表处理，python 中可以用字典结构快速实现哈希表查重：dict.get(element,False)。

利用 pickle 进行断点继爬

通常爬虫会选取一个网页开始进行爬取，但是希望重启程序的时候会有极多相同的网页，虽然上文管理 url 的随机访问算法中改善了这一问题，但是无法得到根本的解决。于是利用 pickle 在每爬取 5k 网页的时候将内存中变量储存在硬盘中，下次进行爬取时可以直接进行冷启动，保证工程的连续性。

快速 I/O

当进行大量 I/O 操作时，文件的读写操作成为了限制爬虫速度的主要因素。如果在同一个文件进行 I/O 操作，lock 操作会将线程阻塞。解决方案是将线程进行 hash（线程号->文件），创建较多文件平衡 I/O 操作时线程阻塞问题。

利用 requests json 库

利用 requests 库的 timeout 可以代替自己编写的多次访问尝试，同时避免了 python3 中 ssl 的问题；利用 json 库可以方便快捷地将 json 转为便于阅读和提取的结构。

原递归多次访问尝试算法：

```python
def get_page(page,valid = False,tryed = 0,trytime = 0):
    # import ssl
    # ssl._create_default_https_context = ssl._create_unverified_context
    content = ''
    Request = urllib2.Request(page)
    Request.add_header('User-Agent','Call Me Maybe '+str(random.random()))
    try:
        response = urllib2.urlopen(Request,timeout=1)
        content = response.read() # .decode("utf-8", "ignore")
        response.close()
    except:
        # traceback.print_exc()
        # print("Error in: " + page)
        if tryed >= trytime:
            return ''
        if valid :
            time.sleep(0.5)
            content = get_page(page,True,tryed+1)
        else:
            content = ''
    if content=='' and valid and tryed < trytime:
        time.sleep(0.5)
        content = get_page(page,True,tryed+1)
    return content
```

第二部分 推荐算法

歌曲-歌曲推荐算法主要基于两种思想：针对歌曲，推荐相似度较高的歌曲；针对用户，推荐用户满意的歌曲。实现结果为用户上传或选取一首歌，系统推荐歌曲。

文本-歌曲推荐算法基于歌曲评论、用户歌单的描述以及歌词，用关键词刻画一首歌的内容。实现结果为用户输入一段文字（如当时的心情），系统推荐相关度高的歌曲。

歌曲相似度计算

歌曲相似度计算分为两部分，一部分是基于用户歌单，另一部分是基于歌曲风格。

首先建立 N 维 list 记录歌曲相关信息（N*N 的矩阵过于稀疏）。

其次进行相似度的计算。

（1）若两首歌同时存在于一个歌单中，将为两首歌的相关度增加 δ

$$\delta = \frac{\rho}{\gamma}$$

式中ρ为歌单的热度，与歌单分享数、收藏数、评论数、评论点赞数有关，线性标准化到[0,1]区间。γ为该歌单歌曲总数。

（2）若两首歌属于同一风格，将为两首歌的相关进行修改

$$\delta' = \delta \cdot \sqrt{\tau}$$

式中τ为风格重合次数，如两首歌均属于“英文”、“乡村”，则τ=2。

用户风格计算

用户风格可以通过导入网易云音乐“我喜欢的”列表，也可以通过搜索历史计算。

用户风格为 N 维向量，每一个维度对应一种风格。简单累加后，将该向量标准化到[0,1]上。标准化算法为：

$$X_i = \frac{1}{e} \cdot e^{\frac{k_i}{\sigma}}, \text{if } k_i \text{ is not } 0$$

$$X_i = 0, \text{if } k_i \text{ is } 0$$

式中σ为标准化前向量 X 中不为零的维数，k_i是第 i 维的正序排序，若标准化前为零，取k_i=0。

结合用户风格的歌曲相似度为用户风格向量 X 与歌曲风格向量卷积值和上文计算的相似度的积。

文本相关度计算

该部分首先对歌曲进行预处理，利用歌词、热门评论和所属歌单风格及描述，用十个有代表性的关键词将歌曲特征化，并利用这部分关键词调整 moha 分词词典。用户输入一段文字后，利用 mohaema 分词系统将其关键词筛选出，与歌曲的关键词进行匹配。这部分利用二叉树进行快速检索。