一、GetID 获取productID

利用python程序处理原始的txt文件，并从中截取productId的信息，共获取到253059个productId。

```
import re
import csv


def extract_productId(input_file):
    file = open(input_file, 'rb')
    new_file = open('product.txt', 'wb')
    for line in file:
        str = "product/productId:"
        str = str.encode()
        if str in line:
            new_file.write(line)
    file.close()
    new_file.close()
    print('productId is extracted')


def extract_id():
    file = open('product.txt', 'r')
    with open('id.csv', 'w') as csv_file:
        writer = csv.writer(csv_file, lineterminator='\n')
        writer.writerow(["Id"])
        for line in file:
            Id = re.findall(r"product/productId:(.+?)\n",line)
            writer.writerow(Id)
    print('Id is extracted')
    file.close()

def duplicate_id():
    file = 'id.csv'
    data = []
    try:
        with open(file) as f:
            reader = csv.reader(f,dialect=csv.excel_tab);
            data = [row for row in reader]
    except csv.Error as e:
        print(reader.line_num,e);
    data.drop_duplicates(inplace=True)

if __name__ == '__main__':
    data_file = './movies.txt'
    print('**********Start***********')
    extract_productId(data_file)
    extract_id()
    # duplicate_id()
    print('**********Done**********')
```

二、WebCrawler 爬取网页信息

异常页面检测

空页面检测：

某些URL对应的页面已经不存在，在爬取时会返回空页面，此时返回的Response不为200，可以通过Response的值进行检测。

验证码页面检测：

当亚马逊服务器检测到某一ip获取了过多的数据后，会返回验证码页面进行验证。在爬取过程中，通过某一URL获得了页面信息后，解析该页面的title信息，若title为“Amazon.com”，不包含任何关于产品的信息，则可以判定该页面为验证码页面。

```
soup = BeautifulSoup(r.text, "lxml")
if ((str(r) != "<Response [200]>") | (str(soup.title) == '<title
dir="ltr">Amazon.com</title>')):
    error = error + 1
```

应对反爬机制

动态cookie：

在程序中用Selenium与ChromeDriver先获取一系列cookie，在爬取网页时随机选取其中的cookie进行访问。

更换ip：

在爬取信息过程中，若检测到异常页面数量超过一个阈值，则更换ip，并继续爬取。

```
def get_cookies():
    from selenium import webdriver
    cookies_list = []
    for _ in range(10):
        url = 'https://www.amazon.com/dp/00006HAXW/'
        driver = webdriver.Chrome("chromedriver.exe")
        driver.get(url=url)
        driver.refresh()
        c = driver.get_cookies()
        cookies = {}
        # 获取cookie中的name和value,转化成requests可以使用的形式
        for cookie in c:
            cookies[cookie['name']] = cookie['value']
        driver.quit()
        cookies_list.append(cookies)
        # 使用该cookie完成请求
    return cookies_list
def getHtml():
    import requests
    import random
    from bs4 import BeautifulSoup
    # 获取cookies
    cookies_list = get_cookies()
    for id in open("./productIDResult.txt"):
        url = "https://www.amazon.com/dp/" + id
```

```
        random_cookie = random.randint(0, len(cookies_list) - 1)  # 随机获取一个
cookie
        r = requests.get(url=url, headers=headers,
cookies=cookies_list[random_cookie])
        # print(r)
        soup = BeautifulSoup(r.text, "lxml")
        with open(file, "w", encoding='utf-8') as f:
            f.write(r.text)
```

三、GetData 爬取电影有效信息

getData部分主要任务是从所有已经爬取下来的完整html文件中提取出所需要的信息，主要使用了BeautifulSoup这个python库。

通过观察以及相关的代码调试，我们发现所有的html文件可以分成三类：空页面（即相关产品已经下架）、普通页面、prime页面。代码实现也是根据这三类进行划分处理的。

我们总共从html文件中提取了id、title、release date、director、actor等十余种信息。

具体流程如下：

1. 从文件中读取id为productTitle标签。经过观察，若该标签存在，就是普通页面，也就实现了对普通页面和空页面、prime页面的区分。对空页面和prime页面，采用了BeautifulSoup的find函数查找data-automation-id为title的h1标签。如果这个标签存在，那么它的内容就是prime页面的title，否则就是空页面。
2. 在对普通页面的进一步处理中，主要找了id为detailBullets_feature_div 的div块下的ul里面的所有li标签。这每一个li的内容就是actor、director等内容。

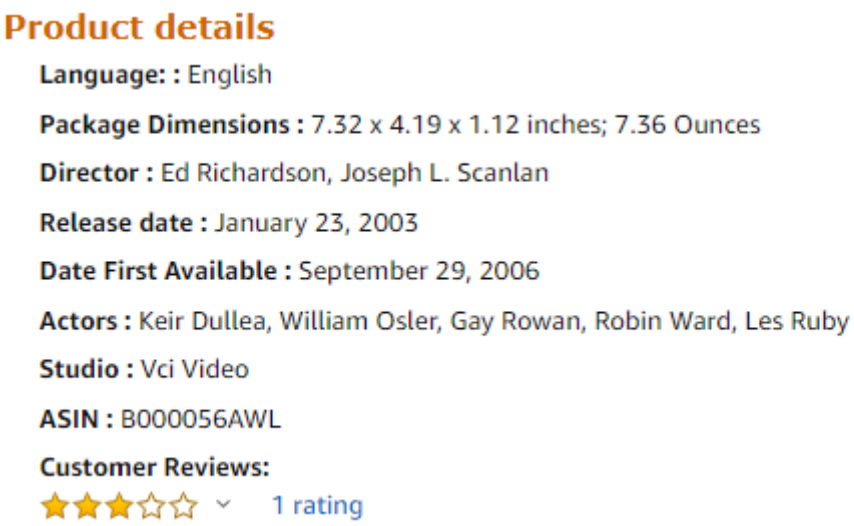

图1-普通页面提取信息的主要部分

3. 在对prime页面的进一步处理中，主要处理了三个部分，如下图：

图2-prime页面提取信息部分一

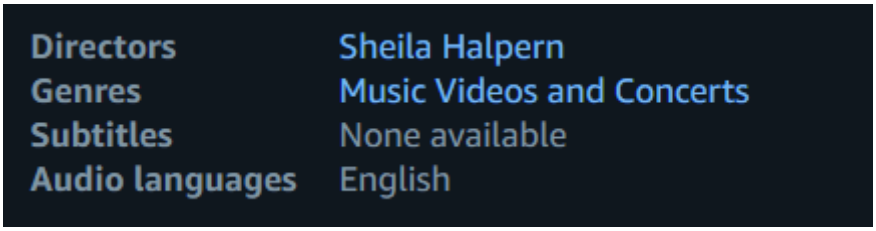

图3-prime页面提取信息部分二

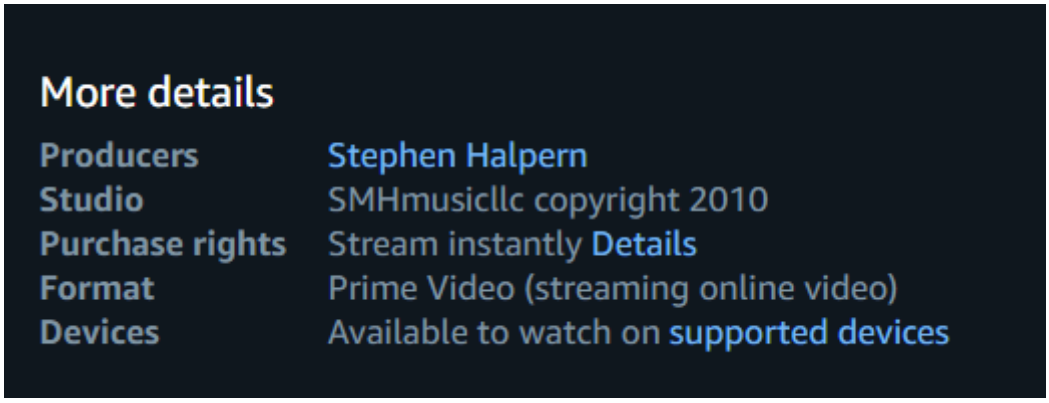

图4-prime页面提取信息部分三

这些信息都是通过BeautifulSoup的find函数按对应的data-automation-id查找出的，不过其中的标签类型不同，id不同，不再赘述。

四、Process Data 处理数据

判断是否为电影

本部分相对宽松

若没有导演，则不为电影（因为会和后续部分筛选有很大重合，所以不用过于担心）。

若运行时间不超过1分钟的，不为电影（判断标准，时间是否包括秒"sec"）。

将相同电影合并

1. 将导出的数据按照title（电影名）的顺序排序
2. 只需要一个循环就可以遍历所有（因为电影名称不同于其他的字符串比对，对于同一个系列这个特征尤为突出：两个电影名称若代表的是相同的电影，则绝大多数电影名称从开头开始保持一致，这使得可以一个循环内保证了准确性）
3. 在循环中，首先设置第0个字符串为temp 1，从1开始取temp 2，若满足判断为相似，则进入下个循环，将下一个设为temp 2；若判断为不相似，则将temp 1导入新的csv文件，将temp 1转换成当前temp 2。循环结束后，最终将剩下的temp 1导入csv文件。（所有的temp都为python的字典类型dictionary）
4. 判断是否为相似的方法为：基本判断为判断电影名称的相似度是否大于某一个阈值（相似度为字符串1转换到字符串2的变换距离除以最大的字符串的长度），接下来针对电影名的独特属性（电影名称从开头开始保持一致）进行判断，若满足电影名称较短，则若两个电影名称的前大半的子字符串相似度相对较高，则判断为两个电影名一致。同时，需要保证电影的发行日期"release date"一样才能保证为相同的电影。
5. 若判断为相同电影（由于我们通过csv储存信息，非常方便，所以可以在后面随意添加字符串），则在后面的新字符串添加了相同电影的编号"link id"和电影名称"'link title'"，方便后续进行操作，同时方便我们自己查看审核。

6. 若取消发行日期"release date"这一要求，则可将同系列电影IP进行合并。

共导出不同电影120985个，不同IP共27262个。

```
for i in range(len(final)):
                temp2 = final[i]
                temp2['link id'] = ''
                temp2['link title'] = ''

                min_lenth = min(len(temp1['title']), len(temp2['title']))
                min_lenth = min(min_lenth, 12)
                min_lenth2 = min(min_lenth, 6)

                if temp1['title'] == temp2['title'] and len(temp1['title']) <= 4
and len(temp2['title']) <= 4 and JudgeTime(temp1['release date'],temp2['release
date']):
                    temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['actor'] = Max(temp1['actor'], temp2['actor'])
                    temp1['supporting actors'] = Max(temp1['supporting actors'],
temp2['supporting actors'])
                    temp1['genres'] = Max(temp1['genres'], temp2['genres'])
                    temp1['producers'] = Max(temp1['producers'],
temp2['producers'])
                    temp1['run time'] = Max(temp1['run time'], temp2['run
time'])
                    temp1['Date First Available'] = Max(temp1['Date First
Available'], temp2['Date First Available'])
                    # temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['link id'] = temp1['link id'] + "\n" + temp2['id']
                    temp1['link title'] = temp1['link title'] + "\n" +
temp2['title']
                    continue
                elif string_similar(temp1['title'], temp2['title']) > 0.6 and
JudgeTime(temp1['release date'],temp2['release date']):
                    temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['actor'] = Max(temp1['actor'], temp2['actor'])
                    temp1['supporting actors'] = Max(temp1['supporting actors'],
temp2['supporting actors'])
                    temp1['genres'] = Max(temp1['genres'], temp2['genres'])
                    temp1['producers'] = Max(temp1['producers'],
temp2['producers'])
                    temp1['run time'] = Max(temp1['run time'], temp2['run
time'])
                    temp1['Date First Available'] = Max(temp1['Date First
Available'], temp2['Date First Available'])
                    # temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['link id'] = temp1['link id'] + "\n" + temp2['id']
                    temp1['link title'] = temp1['link title'] + "\n" +
temp2['title']
                    continue
```

```
                elif string_similar(temp1['title'][:min_lenth], temp2['title']
[:min_lenth]) > 0.9 and len(temp1['title']) >= 8 and len(temp2['title']) >= 8
and len(temp1['title']) <= 18 and len(temp2['title']) <= 18 and
JudgeTime(temp1['release date'],temp2['release date']):
                    temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['actor'] = Max(temp1['actor'], temp2['actor'])
                    temp1['supporting actors'] = Max(temp1['supporting actors'],
temp2['supporting actors'])
                    temp1['genres'] = Max(temp1['genres'], temp2['genres'])
                    temp1['producers'] = Max(temp1['producers'],
temp2['producers'])
                    temp1['run time'] = Max(temp1['run time'], temp2['run
time'])
                    temp1['Date First Available'] = Max(temp1['Date First
Available'], temp2['Date First Available'])
                    # temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['link id'] = temp1['link id'] + "\n" + temp2['id']
                    temp1['link title'] = temp1['link title'] + "\n" +
temp2['title']
                    continue
                elif string_similar(temp1['title'][:min_lenth2], temp2['title']
[:min_lenth2]) > 0.9 and len(temp1['title']) >= 4 and len(temp2['title']) >= 4
and JudgeTime(temp1['release date'],temp2['release date']):
                    temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['actor'] = Max(temp1['actor'], temp2['actor'])
                    temp1['supporting actors'] = Max(temp1['supporting actors'],
temp2['supporting actors'])
                    temp1['genres'] = Max(temp1['genres'], temp2['genres'])
                    temp1['producers'] = Max(temp1['producers'],
temp2['producers'])
                    temp1['run time'] = Max(temp1['run time'], temp2['run
time'])
                    temp1['Date First Available'] = Max(temp1['Date First
Available'], temp2['Date First Available'])
                    # temp1['director'] = Max(temp1['director'],
temp2['director'])
                    temp1['link id'] = temp1['link id'] + "\n" + temp2['id']
                    temp1['link title'] = temp1['link title'] + "\n" +
temp2['title']
                    continue
                writer.writerow(temp1)
                # out_final.append(dict(temp1))
                temp1 = temp2
writer.writerow(temp1)
```