数据分析项目报告

成员：贾志轩，张竞卓，向建宇，刘亚婷，张艺璇，

工作量：贾志轩：20%，张竞卓：20%，向建宇：20%，刘亚婷：20%，张艺璇：20%，

#### 摘要

本项目使用了来源于 MovieLens 中的ml-latest数据集进行了机器学习与深度学习的算法数据分析。其中机器学习中应用协同过滤算法进行电影的相似性计算与推荐，还同时包括了用TensorFlow来构建一套电影的推荐系统。然后使用深度学习算法，采用Keras框架，选取VGG16做模型，爬取相对应的海报做协同过滤算法推荐，得出推荐效果进行比较分析。

#### 项目背景：

在现在互联网的时代，你会发现有很多智能的推荐系统，比如说商品的推荐，歌曲的推荐，电影的推荐。在推荐系统中，协同过滤算法是诞生最早的，也是很常用的推荐算法。推荐算法的思想就是找到和你相似的人，也就是口味相同的人，把他喜欢的物品或者电影歌曲推荐给你；或者是将你买过的或者喜欢的物品，电影或者歌曲推荐给你；还有就是可能将你搜索过的物品、电影或者歌曲推荐给你；还有可能就是综合上述的算法做推荐。

举个例子，在电子商城购物，系统会根据用户的记录或者其他的信息来推荐相应的产品给客户，是一种智能的生活方式。之所以被称之为协同过滤，是因为在实现过滤推荐的时候是根据其他人的行为来做预测的，基于相似用户的喜好来实现用户的喜好预测。

具体的使用流程有以下几个部分：收集用户的喜好，查找相似的用户或者商品，计算推荐。其中，拿到了用户的数据之后要对数据进行处理，处理的依据主要可以分为：将不同的行为分组；将用户的行为进行加权操作。在做完以上的处理之后，还需要对数据进行一些预处理，主要的操作就是减噪和归一化。在获取的数据中，有许多数据是在非正常的情况下产生的，因此在得到的用户数据中，有许多的噪声，所以减噪的目的就是让数据更加纯净，让在后续的数据分析更加精确。还有一个处理就是归一化，因为获取的数据可能来源于不同的系统，所以数据的样式参差不齐，的加权操作就没办法进行，需要把数据统一标准，将数据归一化。协同过滤(collaborative filtering)算法主要有两个分支，基于用户的协同过滤(user-based collaborative filtering)和基于商品的协同过滤(item-based collaborative filtering)。基于用户的协同过滤推荐方法：基于用户的推荐大体意思就是来找到一些相似性很高的用户，那么别人喜欢的商品也很有可能是你喜欢的商品，以此作为依据推荐。还有一种就是基于商品的推荐，就是寻找商品之间的关系，商品之间的相似度，关联度，以此来对用户做推荐，所以经常在购物的时候发现一些推荐，比如购买了此商品的用户还购买了什么，这就是基于商品协同过滤的推荐。

#### 数据集的分析与选择：

选择MovieLens的ml-latest数据集（大于700M），它是一个关于电影评分的数据集，里面包含了从IMDB, The Movie DataBase上面得到的用户对电影的评分信息，具体内容解释为：

links.csv:

文件里面的内容是帮助你如何通过网站id在对应网站上找到对应的电影链接的。数据格式如下：

movieId, imdbId, tmdbId

movieId:表示这部电影在movielens上的id，可以通过链接https://movielens.org/movies/(movieId)来得到。

imdbId:表示这部电影在imdb上的id，可以通过链接http://www.imdb.com/title/(imdbId)/

来得到。

tmdbId:表示这部电影在themoviedb上的id，可以通过链接http://www.imdb.com/title/(tmdbId)/

来得到。

movies.csv:

movieId, title, genres

文件里包含了一部电影的id和标题，以及该电影的类别。数据格式如下：

movieId, title, genres

movieId:每部电影的id

title:电影的标题

genres:电影的类别（详细分类见readme.txt）

ratings.csv:

文件里面的内容包含了每一个用户对于每一部电影的评分。数据格式如下：

userId, movieId, rating, timestamp

userId: 每个用户的id

movieId: 每部电影的id

rating: 用户评分，是5星制，按半颗星的规模递增(0.5 stars - 5 stars)

timestamp: 自1970年1月1日零点后到用户提交评价的时间的秒数

数据排序的顺序按照userId，movieId排列的。

tags.csv:

文件里面的内容包含了每一个用户对于每一个电影的分类。数据格式如下：

userId, movieId, tag, timestamp

userId: 每个用户的id

movieId: 每部电影的id

tag: 用户对电影的标签化评价

timestamp: 自1970年1月1日零点后到用户提交评价的时间的秒数

数据排序的顺序按照userId，movieId排列的。

#### 第一部分：协同过滤推荐系统的实现（python）

计算两者之间的相似度：

通常会先把二维表格绘制在一个图中总，每个用户数据表示一个点。

度量相似度计算的方法：

a.曼哈顿距离计算（计算迅速，节省时间）

b.欧氏距离计算（计算两个点之间的直线距离）

对数据进行预处理，生成一个csv文件，采用python字典来表示每位用户评论的电影和评分，计算任何两位用户之间的相似度，由于每位用户评论的电影不完全一样，所以第一步要找到两位用户共同评论过的电影，然后计算两者之间的欧式距离，最后算出两者之间的相似度， 0表示两位的影评几乎一样，1表示没有共同的影评，然后根据相似度来推荐用户。会碰到因为两个用户之间数据由于数据膨胀，一方数据大，一方数据小，但是两者称明显的线性关系。引入Pearson相关系数来衡量两个变量之间的线性相关性。Pearson：-1~1   -1：完全负相关  1：完全正相关  0：不相关。

#### 第二部分：协同过滤推荐系统的实现（Tensorflow）

工作流程如下：读取电影评分的ratings.csv文件，读取电影库的.csv文件，增加一列名为movieRow的索引值，内容为行号；从电影列表中提取出”movieRow”、”movieId”和”title”三列数据，将这个列表作为处理好的数据储存起来方便后面用到，储存好了特征提取之后的数据之后，将评分列表和电影列表合并，并对其movieId这一列。合并两张表之后，再将合并好的表再来提取特征。特征提取完成之后，来获取评分的用户和电影的数量；创建一个movieNo行，userNo列，内容是0的矩阵；设置一个标签位flag，然后获取合并之后列表的列数，然后遍历rating矩阵每一个位置，将电影的评分填入矩阵中；获取一个电影是否被用户评分的列表，其中1代表该电影已经被用户评分，0是用户没有对该电影评分；得到结果矩阵与评分矩阵。构建模型之前定义一个标准化评分表的方法；将nan的地方转成数字0；计算电影平均的评分；用TensorFlow来构建模型；根据基于内容推荐算法的损失函数公式，公式中，r(x,y)是评分记录表，u是用户数量，θ(j)是j用户的喜好，y(i,j)是i用户对j电影的评分，xi表示电影的内容，n是特征数量，最后一部分是正则化项。后续的目的就是来最小化这个损失函数，接下来就是优化J(θ)，使其最小化，训练模型，递归5000次，直到收敛，评估一下训练的模型，输出想要的电影推荐。

#### 第三部分：深度学习电影推荐（Keras，vgg16）

采用Keras框架，选取VGG16做模型，不进行调参直接采用ImageNet训练好的参数，对电影对应的海报进行feature提取，再将生成的feature flaten成向量，基于海报进行特征提取，尽可能多的扒下来存到本地，计算不同feature的余弦相似度，得到最相似的电影。用urllib模块下载电影海报，并且为了方便多次执行，判断海报是否存在，要是存在的话就跳过。引入Keras VGG16模型，这里的VGG16舍掉了最后的3层FC层，因为出发点就是采用cnn来提取特征，而不需要分类。对每一个海报数据进行预处理，然后喂入模型计算提取（预测）后的特征。保存模型对每一个海报预测后的特征向量，保存到CSV中，判断CSV是否存在，存在的话以后执行就直接读进来不需要再计算一遍了。计算代表电影海报的特征向量间的余弦相似性，计算与电影最相似的n个海报，显示这些海报。

#### 代码及结果分析：

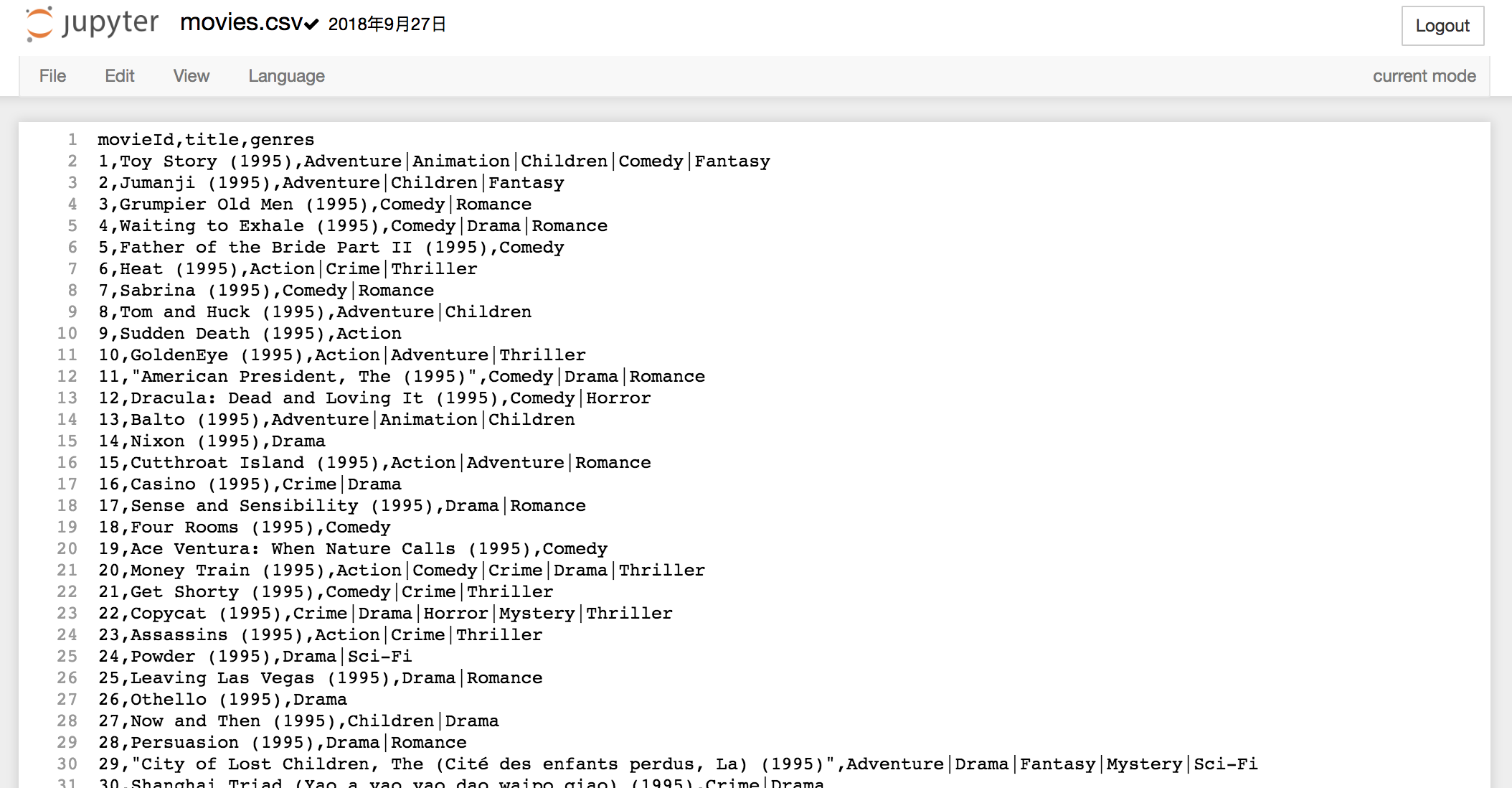
**1. 协同过滤推荐系统的实现（python）**

**1.1 数据预处理：**

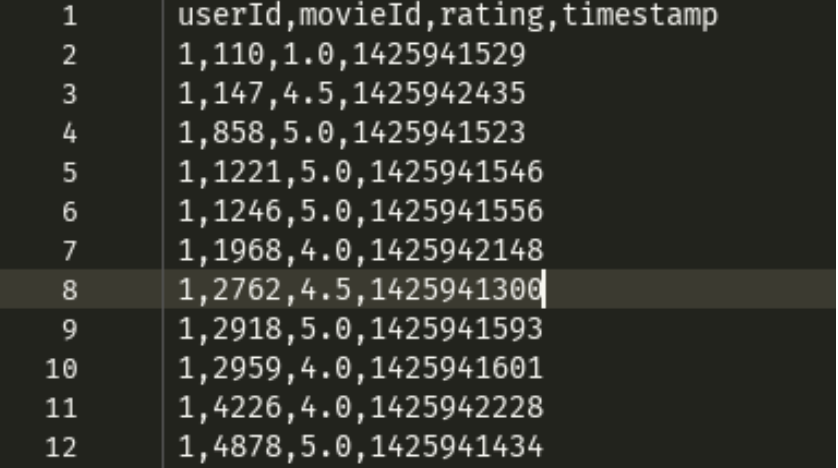
在网站：https://grouplens.org/datasets/movielens/ 下载movieLen数据集，解压读取movies.csv和ratings.csv文件

两个文件的数据格式如下：

**Movies.csv**



**Ratings.csv**



**1.2通过如下程序提取数据：**

"""

@Desc：读取用户的电影数据和评分数据

"""

import pandas as pd

movies = pd.read\_csv("./ml-latest/movies.csv")

ratings = pd.read\_csv("./ml-latest/ratings.csv")##这里注意如果路径的中文件名开头是r，要转义。

data = pd.merge(movies,ratings,on = 'movieId')#通过两数据框之间的movieId连接

data[['userId','rating','movieId','title']].sort\_values('userId').to\_csv('./ml-latest/data.csv',index=False)

结果在当前目录下，生成新合并数据：data.csv

**1.3采用python字典来表示每位用户评论的电影和评分**

"""

@Desc：采用python字典来表示每位用户评论的电影和评分

"""

file = open("./ml-latest/data.csv",'r', encoding='UTF-8')#记得读取文件时加‘r’， encoding='UTF-8'

##读取data.csv中每行中除了名字的数据

data = {}##存放每位用户评论的电影和评分

for line in file.readlines()[1:100]:

#注意这里不是readline()

line = line.strip().split(',')

#如果字典中没有某位用户，则使用用户ID来创建这位用户

if not line[0] in data.keys():

data[line[0]] = {line[3]:line[1]}

#否则直接添加以该用户ID为key字典中

else:

data[line[0]][line[3]] = line[1]

**1.4 计算任何两位用户之间的相似度。**

由于每位用户评论的电影不完全一样，所以兽先要找到两位用户共同评论过的电影，然后计算两者之间的欧式距离，最后算出两者之间的相似度。

"""计算任何两位用户之间的相似度，由于每位用户评论的电影不完全一样，所以兽先要找到两位用户共同评论过的电影

然后计算两者之间的欧式距离，最后算出两者之间的相似度

"""

from math import \*

def Euclidean(user1,user2):

#取出两位用户评论过的电影和评分

user1\_data=data[user1]

user2\_data=data[user2]

distance = 0

#找到两位用户都评论过的电影，并计算欧式距离

for key in user1\_data.keys():

if key in user2\_data.keys():

#注意，distance越大表示两者越相似

distance += pow(float(user1\_data[key])-float(user2\_data[key]),2)

return 1/(1+sqrt(distance))#这里返回值越小，相似度越大

#计算某个用户与其他用户的相似度

def top10\_simliar(userID):

res = []

for userid in data.keys():

#排除与自己计算相似度

if not userid == userID:

simliar = Euclidean(userID,userid)

res.append((userid,simliar))

res.sort(key=lambda val:val[1])

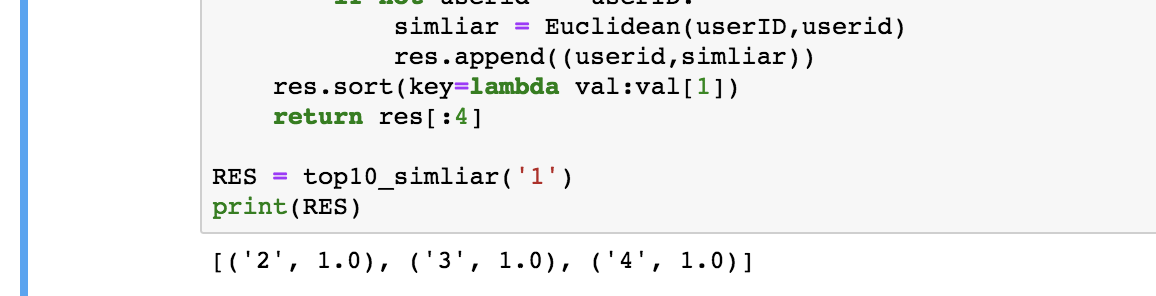
return res[:4]

RES = top10\_simliar('1')

print(RES)

实例：输入user\_id=1，输出与用户1相似的前三个用户，以及有无共同影评。

用户之间相似度结果：0表示两位的影评几乎一样，1表示没有共同的影评



**1.5 根据相似度来推荐用户**

'''

根据用户推荐电影给其他人

'''

def recommend(user):

#相似度最高的用户

top\_sim\_user = top10\_simliar(user)[0][0]

#相似度最高的用户的观影记录

items = data[top\_sim\_user]

recommendations = []

#筛选出该用户未观看的电影并添加到列表中

for item in items.keys():

if item not in data[user].keys():

recommendations.append((item,items[item]))

recommendations.sort(key=lambda val:val[1],reverse=True)#按照评分排序

#返回评分最高的10部电影

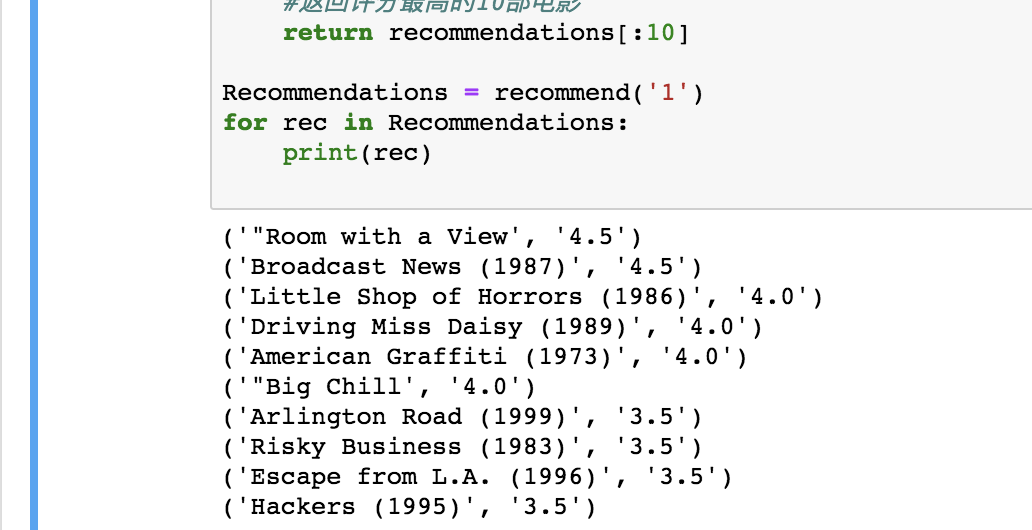
return recommendations[:10]

Recommendations = recommend('1')

for rec in Recommendations:

print(rec)

推荐结果：



如图所示，输入用户id = 1，返回根据协同过滤算法所推荐的电影，并返回电影所对应的推荐度。

**1.6 引入Pearson相关系数**

有时我们会碰到因为两个用户之间数据由于数据膨胀，一方数据大，一方数据小，但是两者称明显的线性关系

我们引入Pearson相关系数来衡量两个变量之间的线性相关性。

Pearson：-1~1   -1：完全负相关  1：完全正相关  0：不相关

相关系数 0.8-1.0 极强相关

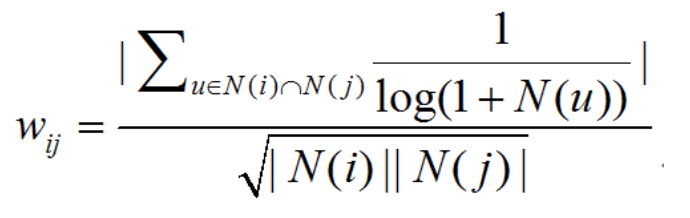
0.6-0.8 强相关

0.4-0.6 中等程度相关

0.2-0.4 弱相关

0.0-0.2 极弱相关或无相关

公式：



Pearson相关系数的Python实现

'''

计算两用户之间的Pearson相关系数

'''

def pearson\_sim(user1,user2):

# 取出两位用户评论过的电影和评分

user1\_data = data[user1]

user2\_data = data[user2]

distance = 0

common = {}

# 找到两位用户都评论过的电影

for key in user1\_data.keys():

if key in user2\_data.keys():

common[key] = 1

if len(common) == 0:

return 0#如果没有共同评论过的电影，则返回0

n = len(common)#共同电影数目

print(n,common)

##计算评分和

sum1 = sum([float(user1\_data[movie]) for movie in common])

sum2 = sum([float(user2\_data[movie]) for movie in common])

##计算评分平方和

sum1Sq = sum([pow(float(user1\_data[movie]),2) for movie in common])

sum2Sq = sum([pow(float(user2\_data[movie]),2) for movie in common])

##计算乘积和

PSum = sum([float(user1\_data[it])\*float(user2\_data[it]) for it in common])

##计算相关系数

num = PSum - (sum1\*sum2/n)

den = sqrt((sum1Sq-pow(sum1,2)/n)\*(sum2Sq-pow(sum2,2)/n))

if den == 0:

return 0

r = num/den

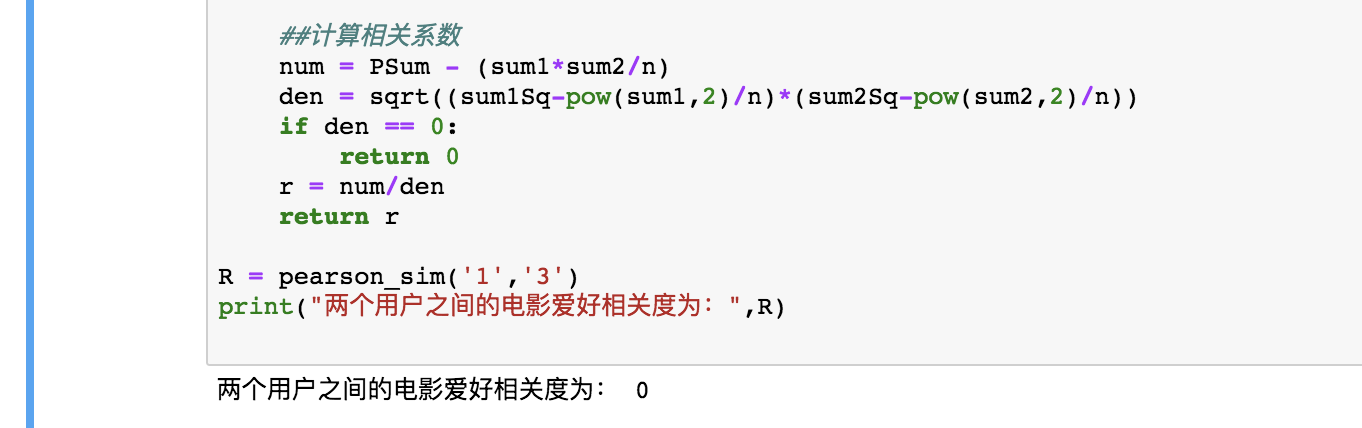
return r

R = pearson\_sim('1','3')

print(R)

输入两个用户Id，返回两个用户之间的相关度。

结果如下：



此外，在实验中，我们发现，通过Pearson系数得到用户的相似度和通过欧式距离得到结果可能不一样

**2. 协同过滤推荐系统的实现（Tensorflow）**

**2.1准备数据集**

首先我们要准备数据集，数据集可以从电影数据集链接 来下载解压得到.csv文件。

**2.2数据的清洗**

首先我们要导入Python的几个库：

import pandas as pd

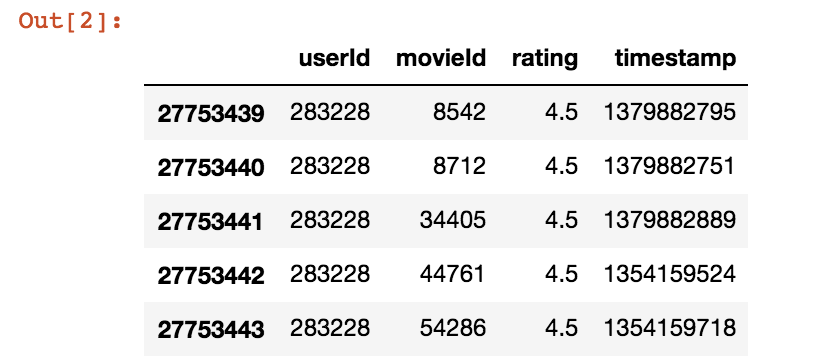
import numpy as np

import tensorflow as tf

然后是读取电影评分的ratings.csv文件，就可以查看这张电影评分表的内容了，我们来看一下这个表中最后几行的内容。

ratings\_df = pd.read\_csv("./ml-latest/ratings.csv")

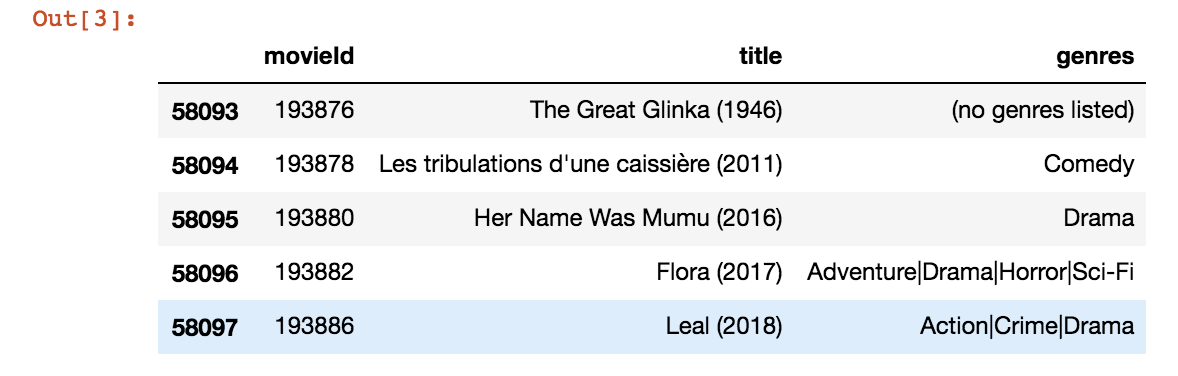
ratings\_df.tail()



同样，我们来读取电影库的.csv文件，我们来看看电影列表中的最后几行内容：

movies\_df = pd.read\_csv('Desktop/ml-latest-small/movies.csv')

movies\_df.tail()

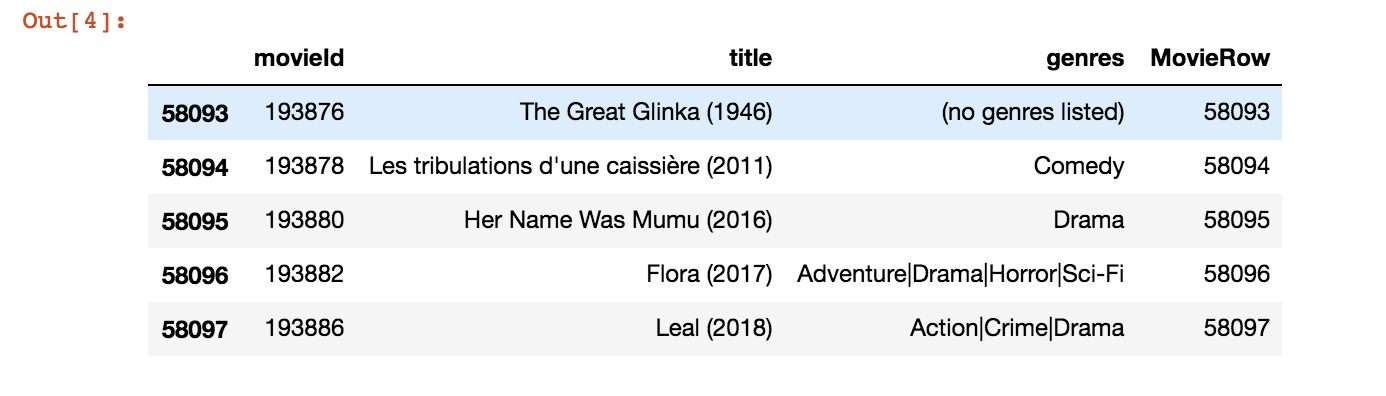


增加一列名为movieRow的索引值，内容为行号：

movies\_df['MovieRow'] = movies\_df.index

movies\_df.tail()

相较于上一张图，我们可以看到最后多了一列索引值，和行号相等。

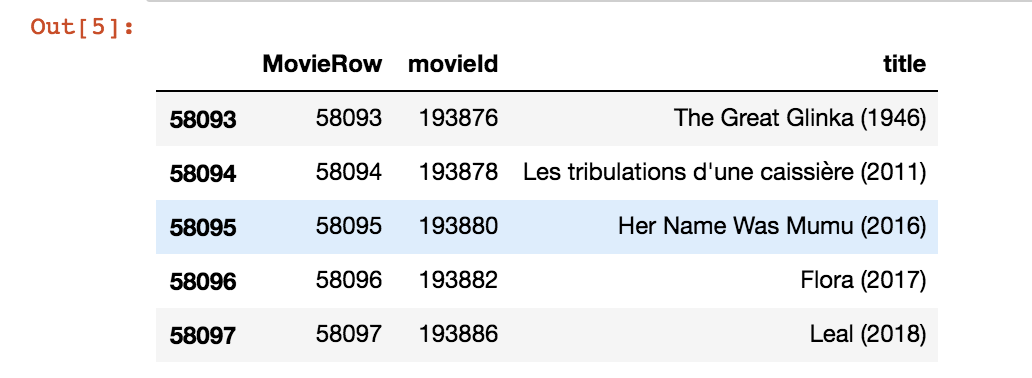


* 1. **特征提取**

我们清洗好了数据之后，就可以将数据进行特征提取，从而拿到我们需要的一些数据。

movies\_df = movies\_df[['MovieRow','movieId','title']]

movies\_df.tail()



我们从电影列表中提取出”movieRow”、”movieId”和”title”三列数据，可以得到：

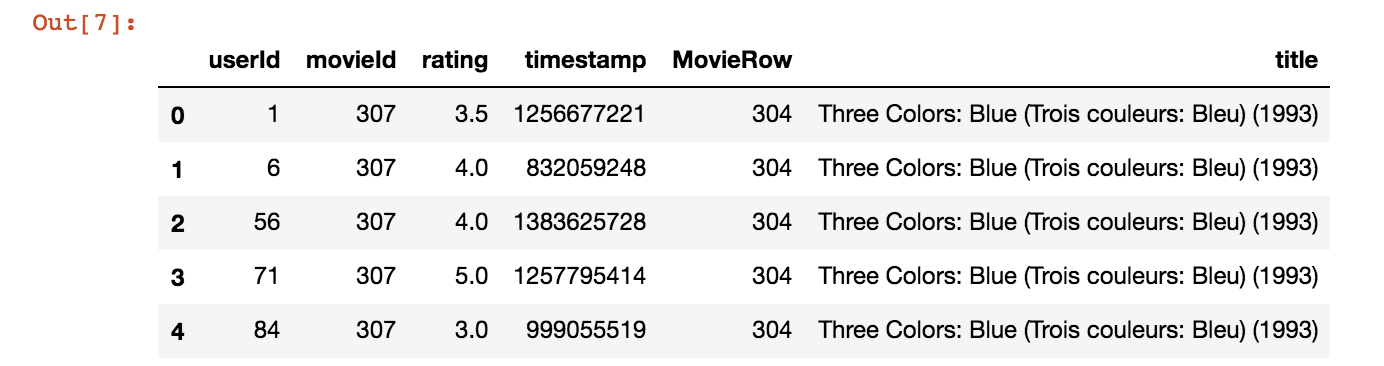
然后我们将这个列表作为处理好的数据储存起来方便后面用到：

movies\_df.to\_csv('Desktop/ml-latest-small/moviesProcessed.csv',index = False, header = True, encoding = 'utf-8')

储存好了特征提取之后的数据之后，我们将评分列表和电影列表合并，并对其movieId这一列。

ratings\_df = pd.merge(ratings\_df, movies\_df, on = 'movieId')

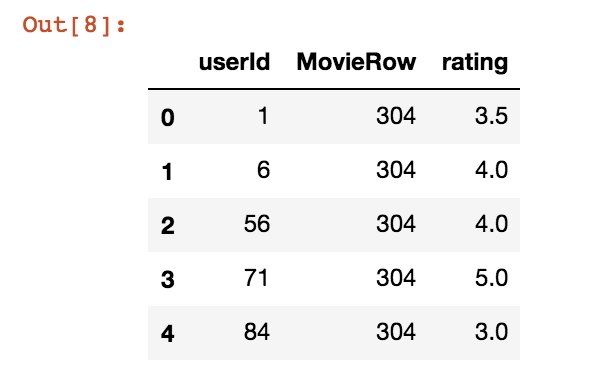
ratings\_df.head()



合并两张表之后，我们将合并好的表再来提取特征：

ratings\_df = ratings\_df[['userId','MovieRow','rating']]

ratings\_df.head()



**2.4创建电影评分矩阵rating和评分记录矩阵record**

特征提取完成之后，首先我们来获取评分的用户和电影的数量：

userNo = ratings\_df['userId'].max()+1

movieNo = ratings\_df['MovieRow'].max()+1

接下来创建一个movieNo行，userNo列，内容是0的矩阵：

rating = np.zeros((movieNo,userNo))

设置一个标签位flag，然后获取合并之后列表的列数，然后遍历rating矩阵每一个位置，将电影的评分填入矩阵中：

#标志位

flag = 0

#获取合并表中的列数

ratings\_df\_length = np.shape(ratings\_df)[0]

#遍历矩阵，将电影的评分填入表中

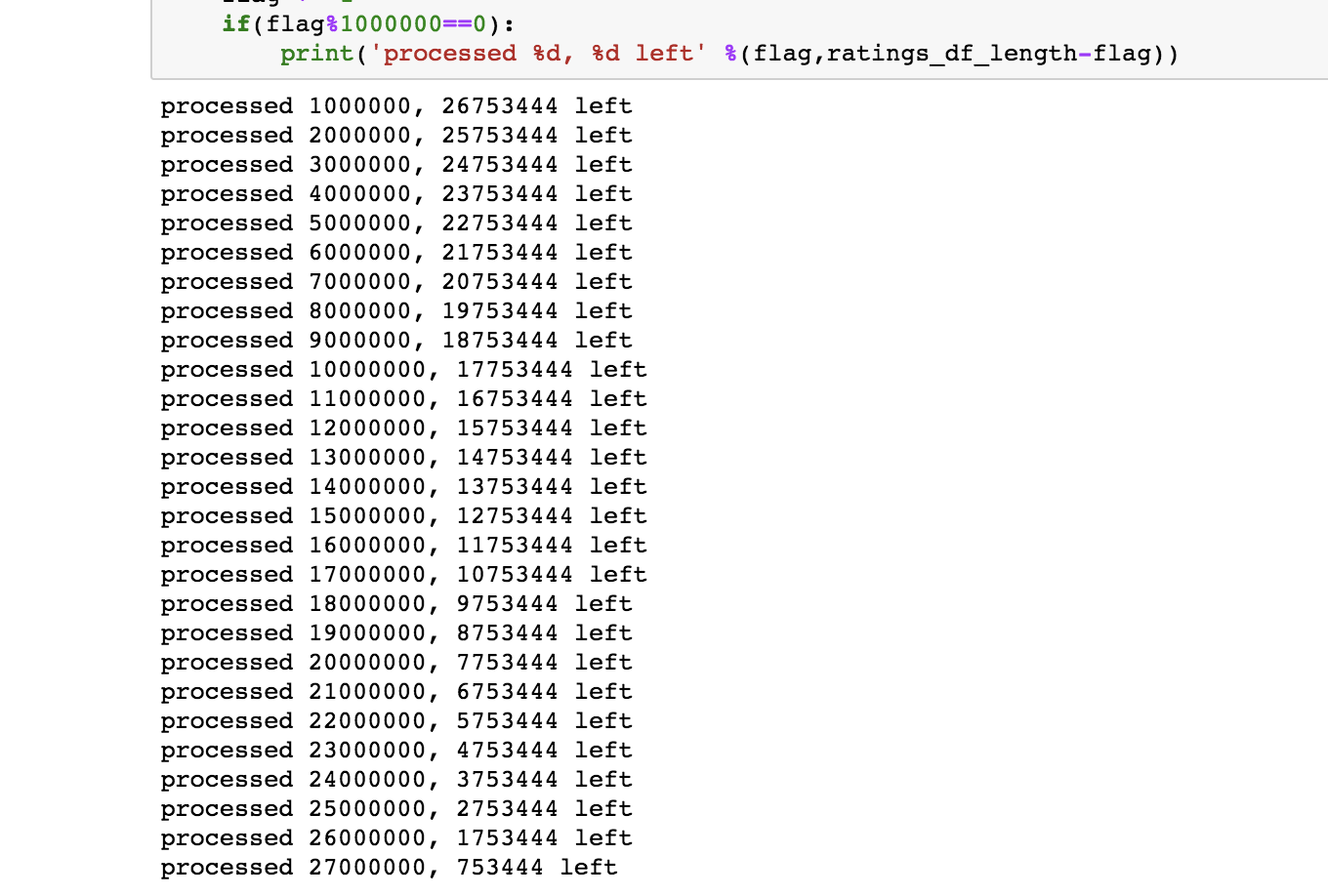
for index,row in ratings\_df.iterrows():

rating[int(row['MovieRow']), int(row['userId'])] = row['rating']

flag += 1

if(flag%1000000==0): # 每100000次，打印一次进度

print('processed %d, %d left' %(flag,ratings\_df\_length-flag))



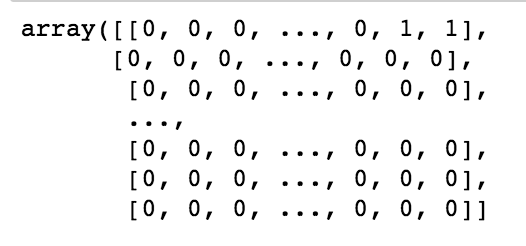
然后我们来获取一个电影是否被用户评分的列表，其中1代表该电影已经被用户评分，0是用户没有对该电影评分：

record = rating > 0

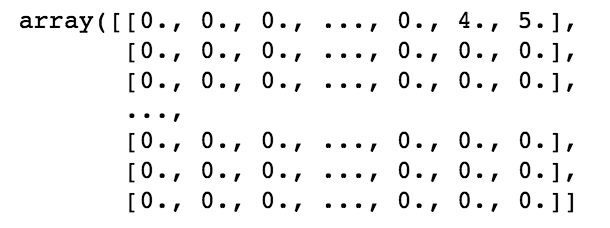
record = np.array(record, dtype = int)

record

可以得到结果矩阵：



同时我们也可以得到评分的矩阵：



**2.5构建模型**

构建模型之前，我们来定义一个标准化评分表的方法：

def normalizeRatings(rating, record):

#获取电影的数量m和用户的数量n

m,n = rating.shape

#rating\_mean-电影平均分 rating\_norm-标准化后的电影得分

rating\_mean = np.zeros((m,1))

rating\_norm = np.zeros((m,n))

rating\_norm = np.nan\_to\_num(rating\_norm)

for i in range(m):

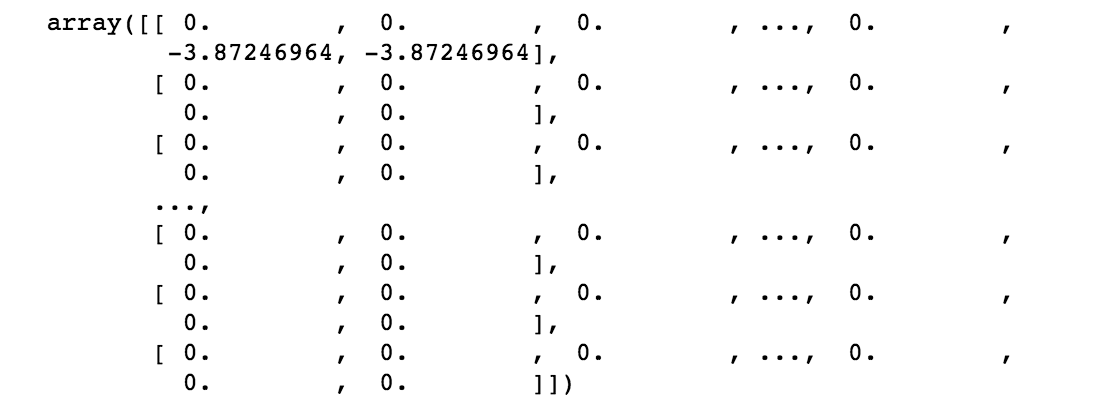
idx = record[i,:]!=0

rating\_mean[i] = np.mean(rating[i,idx])

rating\_norm[i,idx] -= rating\_mean[i]

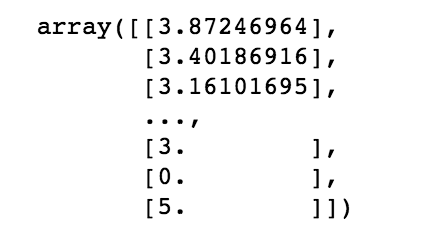
return rating\_norm, rating\_mean

可以得到标准化后的矩阵：



同理我们计算电影平均的评分：

rating\_mean = np.nan\_to\_num(rating\_mean)



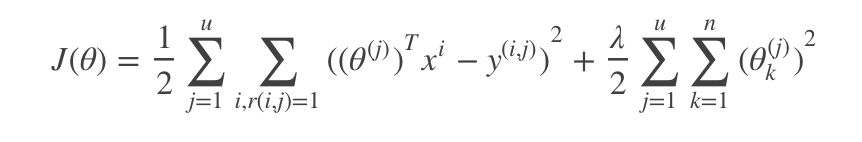
接下来就是用TensorFlow来构建模型：

num\_features = 10

X\_parameters = tf.Variable(tf.random\_normal([movieNo, num\_features],stddev = 0.35))

Theta\_parameters = tf.Variable(tf.random\_normal([userNo, num\_features],stddev = 0.35))

我们可以根据基于内容推荐算法的损失函数公式：



在这个公式中，r(x,y)是评分记录表，u是用户数量，θ(j)是j用户的喜好，y(i,j)是i用户对j电影的评分，xi表示电影的内容，n是特征数量，最后一部分是正则化项。我们后续的目的就是来最小化这个损失函数，接下来就是优化J(θ)，使其最小化。

optimizer = tf.train.AdamOptimizer(1e-4)

train = optimizer.minimize(loss)

**2.6训练模型**

tf.summary.scalar('loss', loss)

结果：



summaryMerged = tf.summary.merge\_all()

#merge\_all 可以将所有summary全部保存到磁盘，以便tensorboard显示。

filename = './movie\_tensorborad'

writer = tf.summary.FileWriter(filename)

#指定一个文件用来保存图。

sess = tf.Session()

#定义一个session

init = tf.global\_variables\_initializer()

sess.run(init)

#运行session

接下来就是递归5000次，直到收敛：

for i in range(5000):

\_, movie\_summary = sess.run([train, summaryMerged])

writer.add\_summary(movie\_summary, i)

模型训练完成之后，我们再来评估一下我们训练的模型：

Current\_X\_parameters, Current\_Theta\_parameters = sess.run([X\_parameters, Theta\_parameters])

# Current\_X\_parameters为用户内容矩阵，Current\_Theta\_parameters用户喜好矩阵

predicts = np.dot(Current\_X\_parameters,Current\_Theta\_parameters.T) + rating\_mean

errors = np.sqrt(np.sum((predicts - rating)\*\*2))

# sqrt(arr) ,计算各元素的平方根

errors



我得到的error为4045.411568958254。

**2.7最后我们可以尝试输出我们想要的电影推荐了**

userId = input('您要向哪位用户进行推荐呢？请输入用户编号：(smaller than 672)')

sortedResult = predicts[:, int(userId)].argsort()[::-1]

idx = 0

print('为该用户推荐的评分最高的20部电影是：'.center(80,'='))

for i in sortedResult:

print('score: %.3f, movie name: %s' % (predicts[i, int(userId)], movies\_df.iloc[i]['title']))

idx += 1

if idx == 20:break

我们可以输入用户的ID，然后就可以得到如下的电影推荐输出：



**3. 深度学习电影推荐（Keras，vgg16）**

**3.1 文件预处理**

先计算得到要下载电影的IMDB id并保存在list中：

def get\_poster(imdb,base\_url):

#query themovie.org API for movie poster path.

file\_path = ''

imdb\_id = 'tt0{0}'.format(imdb)

movie\_url = 'http://api.themoviedb.org/3/movie/{:}/images'.format(imdb\_id)

response = requests.get(movie\_url,params=payload,headers=headers)

try:

file\_path = json.loads(response.text)['posters'][0]['file\_path']

except:

print('Failed to get url for imdb: {0}'.format(imdb))

return base\_url+file\_path

df\_id = pd.read\_csv(link\_f,sep=',')

idx\_to\_mv = {}

for row in df\_id.itertuples():

idx\_to\_mv[row[1]-1] = row[2]

mvs = [0]\*len(idx\_to\_mv.keys())

for i in range(len(mvs)):

if i in idx\_to\_mv.keys() and len(str(idx\_to\_mv[i])) == 6:

mvs[i] = idx\_to\_mv[i]

mvs = list(filter(lambda imdb:imdb!=0,mvs))

mvs = mvs[:download\_posters]

total\_mvs = len(mvs)

再用urllib模块下载电影海报，并且为了方便多次执行，加入code判断海报是否存在，要是存在的话就跳过，不要浪费时间再下一次：

for m in mvs:

if(os.path.exists(poster\_pt+str(i)+'.jpg')):

print('Skip downloading exists jpg: {0}.jpg'.format(poster\_pt+str(i)))

i += 1

continue

URL[i] = get\_poster(m,base\_url)

if(URL[i] == base\_url):

print('Bad imdb id: {0}'.format(m))

mvs.remove(m)

continue

print('No.{0}: Downloading jpg(imdb {1}) {2}'.format(i,m,URL[i]))

urllib.request.urlretrieve(URL[i],poster\_pt+str(i)+'.jpg')

URL\_IMDB['url'].append(URL[i])

URL\_IMDB['imdb'].append(m)

i += 1

**3.2特征提取**

引入Keras VGG16模型，这里的VGG16舍掉了最后的3层FC层，采用cnn的卷积层和池化层来提取特征 ：

from keras.applications import VGG16

from keras.applications.vgg16 import preprocess\_input

from keras.preprocessing import image as kimage

image = [0]\*total\_mvs

x = [0]\*total\_mvs

prediction\_result = 'prediction\_result.csv'

def save\_predict\_res(matrix\_res):

pe = dict([(x,[]) for x in range(matrix\_res.shape[0])])

for i in range(matrix\_res.shape[0]):

pe[i].extend(matrix\_res[i])

df = pd.DataFrame(data=pe)

df.to\_csv(prediction\_result)

def load\_result\_matrix(file):

if not os.path.exists(file):

return None

m\_r = pd.read\_csv(file,sep=',')

f\_r = np.zeros((m\_r.shape[1]-1,m\_r.shape[0]))

for i in range(m\_r.values.shape[1]-1):

f\_r[i] = m\_r[str(i)].values.tolist()

return f\_r

matrix\_res = load\_result\_matrix(prediction\_result)

if matrix\_res is None:

for i in range(total\_mvs):

image[i] = kimage.load\_img(poster\_pt+str(i)+'.jpg',target\_size=(224,224))

x[i] = kimage.img\_to\_array(image[i])

x[i] = np.expand\_dims(x[i],axis=0)

x[i] = preprocess\_input(x[i])

model = VGG16(include\_top=False,weights='imagenet')

prediction = [0]\*total\_mvs

matrix\_res = np.zeros([total\_mvs,25088])

for i in range(total\_mvs):

prediction[i]=model.predict(x[i]).ravel()

matrix\_res[i,:] = prediction[i]

save\_predict\_res(matrix\_res)

**3.3计算相似度**

计算代表电影海报的特征向量间的余弦相似性：

similarity\_deep = matrix\_res.dot(matrix\_res.T)

norms = np.array([np.sqrt(np.diagonal(similarity\_deep))])

similarity\_deep = (similarity\_deep/(norms\*norms.T))

计算与电影base\_mv\_idx最相似的n\_display个海报：

n\_display = 5

base\_mv\_idx = 0

mv = [x for x in np.argsort(similarity\_deep[base\_mv\_idx])[:-n\_display-1:-1]]

显示这些海报：

from IPython.display import Image

from IPython.display import display

from IPython.display import HTML

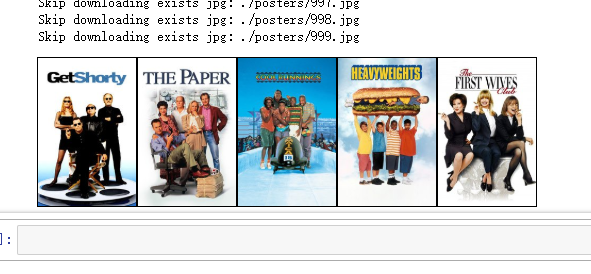
images = ''

for i in range(len(mv)):

images+="<img style='width: 100px; margin: 0px; float: left; border: 1px solid black;' src={0}.jpg />".format(poster\_pt+str(mv[i]))

display(HTML(images))

**3.4结果截图：**



第一张为输入的海报（即GetShorty）输出为后面4张推荐的海报，观察海报内容都是上面横着一行标题，下面居中站着一群人，图片的相似性时比较高的。

下面再输入一张与宠物相关的海报，得到的推荐全是跟宠物相关的，毕竟图像特征提取时，宠物和人还是有明显的区别



再用另一张海报测试，推荐的海报全是带有圈的，可以看到海报中间都有一个很明显的一个圈，这也说明cnn把它当成了一个主要的特征



更多测试结果：





总结：利用DL学习特征，来预测相似性的直观准确率是很高的。不过这里需要注意一点，就是海报库要尽可能的大，不然如果根本没有那么多相似的海报却非要找那么多相似的，强扭的瓜不甜的。

#### 参考文献：

1.《统计学习方法》 李航

2.《机器学习》 周志华

3. 哈林顿. 机器学习实战[M]. 北京: 人民邮电出版社, 2013.

4．《TensorFlow实战Google深度学习框架》

5. 王茜, 王均波. 一种改进的协同过滤推荐算法[J]. 计算机科学, 2010, 37(6):226-228243.

6. 周军锋, 汤显, 郭景峰. 一种优化的协同过滤推荐算法[J]. 计算机研究与发展, 2004, 41(10).

7. 邓爱林, 朱扬勇, 施伯乐. 基于项目评分预测的协同过滤推荐算法[J]. 软件学报, 2003, 14(9):1621-1628.

8. 丁少衡, 姬东鸿, 王路路. 基于用户属性和评分的协同过滤推荐算法[J]. 计算机工程与设计, 2015(2):487-491.

9. 刘芳先, 宋顺林. 改进的协同过滤推荐算法[J]. 计算机工程与应用, 2011, 47(8):72-75.

10. 杨阳, 向阳, 熊磊. 基于矩阵分解与用户近邻模型的协同过滤推荐算法[J]. 计算机应用, 2012, 32(02):395-398.