目录

[0 源码地址 1](#_Toc53940652)

[1 处理思路 1](#_Toc53940653)

[1.1 算法选择 1](#_Toc53940654)

[2 算法实现 2](#_Toc53940655)

[2.1 数据预处理 2](#_Toc53940656)

[2.2 协同过滤 4](#_Toc53940657)

[用户-电影评分矩阵 4](#_Toc53940658)

[用户相似度矩阵 5](#_Toc53940659)

[KNN 7](#_Toc53940660)

[预测评分 7](#_Toc53940661)

[获得推荐 8](#_Toc53940662)

[2.3 基于内容 8](#_Toc53940663)

[每类电影排名 8](#_Toc53940664)

[用户-电影类型偏好矩阵 10](#_Toc53940665)

[推荐喜欢类型的高分电影 11](#_Toc53940666)

[2.4 保存结果 11](#_Toc53940667)

[3 模型评估 12](#_Toc53940668)

[3.1 数据划分 12](#_Toc53940669)

[3.2 测试方法 12](#_Toc53940670)

[RMSE 13](#_Toc53940671)

[Coverage 13](#_Toc53940672)

[3.3 测试结果 13](#_Toc53940673)

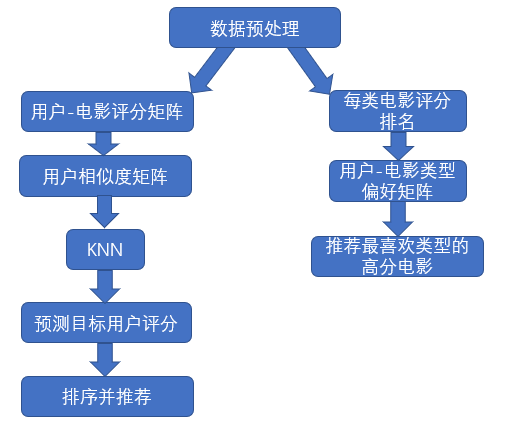
[4 推荐结果 14](#_Toc53940674)

# 0 源码地址

https://github.com/Cpaulyz/BigDataAnalysis/tree/master/Assignment6

# 1 处理思路

总体处理思路如图所示



## 1.1 算法选择

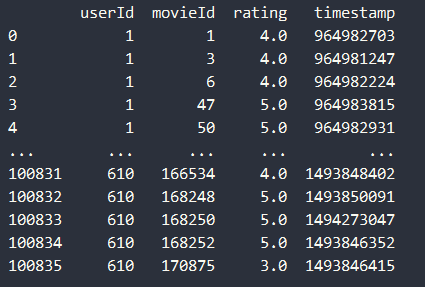
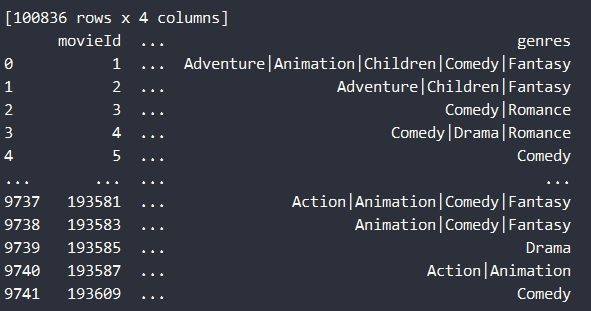
* **协同过滤算法**
  + 通过分析，我们发现一共有610位用户和9742篇电影，为了缩小相似度矩阵的大小，选择了**基于用户的的协同过滤算法**。
* **基于内容的推荐算法**
  + 通过电影数据，可以得到每类电影下的评分排名；根据用户历史评分数据，可以得出用户对各类电影的偏爱程度。
  + 因此，可以向用户推荐偏爱类型电影下的高分电影。

# 2 算法实现

## 2.1 数据预处理

目标是对每个用户进行电影推荐，数据集采用MovieLens数据集

主要使用了以下两个数据集的内容

* ratings.csv
  + 
  + 使用到了**userId，movieId和rating**字段，通过userId和movieId建立矩阵ratings\_matrix，值为user对movie的评分rating
* movies.csv
  + 
  + 使用到了**movieId和genres**字段，通过分词建立每个电影的类型数据
  + 经过分析，发现一共有以下几种类型
  + genres\_list = movies['genres'].values.tolist()  
    type\_list = []  
    for item in genres\_list:  
     movie\_types = item.split('|')  
     for movie\_type in movie\_types:  
     if movie\_type not in type\_list:  
     type\_list.append(movie\_type)  
    print(type\_list)  
      
    # ['Adventure', 'Animation', 'Children', 'Comedy', 'Fantasy', 'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror', 'Mystery', 'Sci-Fi', 'War', 'Musical', 'Documentary', 'IMAX', 'Western', 'Film-Noir', '(no genres listed)']

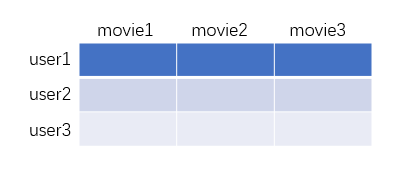
在数据预处理阶段，我们导入表格并进行一些基本的处理

def get\_list\_index\_map(list):  
 """  
 将列表转为map  
 :param list:输入列表  
 :return:  
 1. map item:index  
 2. map\_reverse index:item  
 """  
 map = {}  
 map\_reverse = {}  
 for i in range(len(list)):  
 map[list[i]] = i  
 map\_reverse[i] = list[i]  
 return map, map\_reverse  
  
def get\_type\_list():  
 """  
 获得所有类型的列表  
 :return: 所有电影类型的list  
 """  
 type\_list = []  
 for item in genres\_list:  
 movie\_types = item.split('|')  
 for movie\_type in movie\_types:  
 if movie\_type not in type\_list and movie\_type != '(no genres listed)':  
 type\_list.append(movie\_type)  
 return type\_list  
  
# 读表  
ratings = pd.read\_csv('./dataset/ratings.csv', index\_col=None)  
movies = pd.read\_csv('./dataset/movies.csv', index\_col=None)  
# 转为list  
user\_list = ratings['userId'].drop\_duplicates().values.tolist()  
movie\_list = movies['movieId'].drop\_duplicates().values.tolist()  
genres\_list = movies['genres'].values.tolist()  
type\_list = get\_type\_list()  
# 获得原index-数组index的map，便于后续处理  
type\_map, type\_map\_reverse = get\_list\_index\_map(type\_list)  
user\_map, user\_map\_reverse = get\_list\_index\_map(user\_list)  
movie\_map, movie\_map\_reverse = get\_list\_index\_map(movie\_list)

## 2.2 协同过滤

### 用户-电影评分矩阵

根据ratings.csv下的内容，构建一个如下所示的二维数组



def get\_rating\_matrix():  
 """  
 构造评分矩阵  
 :return: 二维数组，[i,j]表示user\_i对movie\_j的评分，缺省值为0  
 """  
 matrix = np.zeros((len(user\_map.keys()), len(movie\_map.keys())))  
 for row in ratings.itertuples(index=True, name='Pandas'):  
 user = user\_map[getattr(row, "userId")]  
 movie = movie\_map[getattr(row, "movieId")]  
 rate = getattr(row, "rating")  
 matrix[user, movie] = rate  
 print(matrix)  
 return matrix  
  
ratings\_matrix = get\_rating\_matrix()

### 用户相似度矩阵

使用余弦相似度来计算用户之间的相似度关系

def get\_user\_sim\_matrix(input\_matrix):  
 """  
 构造用户相似度矩阵  
 :param input\_matrix: 输入矩阵，每i行代表用户i的特征向量  
 :return: 对称矩阵，[i,j]=[j,i]=sim(user\_i,user\_j)  
 """  
 size = len(input\_matrix)  
 matrix = np.zeros((size, size))  
 for i in range(size):  
 for j in range(i + 1, size):  
 sim = cosine\_similarity(input\_matrix[i], input\_matrix[j])  
 # sim = 1  
 matrix[i, j] = sim  
 matrix[j, i] = sim # 对称矩阵，对角线为0  
 return matrix  
  
  
def cosine\_similarity(list1, list2):  
 """  
 计算余弦相似度  
 :param list1: 用户1的特征向量  
 :param list2: 用户2的特征向量  
 :return: 两个特征向量之间的余弦相似度  
 """  
 res = 0  
 d1 = 0  
 d2 = 0  
 for index in range(len(list1)):  
 val1 = list1[index]  
 val2 = list2[index]  
 # for (val1, val2) in zip(list1, list2):  
 res += val1 \* val2  
 d1 += val1 \*\* 2  
 d2 += val2 \*\* 2  
 return res / (math.sqrt(d1 \* d2))  
  
user\_sim\_matrix = get\_user\_sim\_matrix(ratings\_matrix)  
np.savetxt('user\_sim\_matrix.csv', user\_sim\_matrix, delimiter = ',')



这里因为要计算很久，为了避免重复计算，我们将结果存到了user\_sim\_matrix.csv中，后续使用时候只需要

user\_sim\_matrix\_by\_rating = np.loadtxt(open("./user\_sim\_matrix.csv", "rb"), delimiter=",")

即可

### KNN

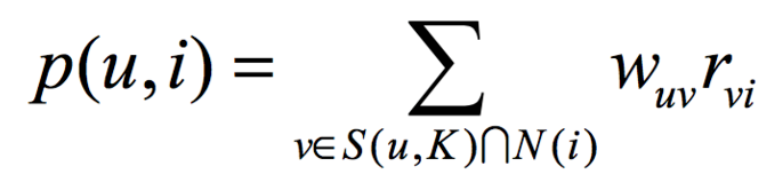
对于目标用户第index位用户，根据相似度矩阵，选择k个与其相似度最高的用户

def k\_neighbor(matrix, index, k):  
 """  
 输入相似矩阵，读取k邻居的index  
 :param matrix: 相似矩阵  
 :param index: 目标index  
 :param k:  
 :return: list([k-index,相似度],....)  
 """  
 line = matrix[index]  
 tmp = []  
 for i in range(len(line)):  
 tmp.append([i, line[i]])  
 tmp.sort(key=lambda val: val[1], reverse=True)  
 return tmp[:k]

### 预测评分

根据knn的结果，计算目标用户第index位用户，预测其对所有电影的评分。这里默认选择k=10

计算公式为



def get\_predict(matrix, index, k):  
 """  
 获取预测评分  
 :param matrix: 相似矩阵  
 :param index: 目标index  
 :param k:k邻居的k  
 :return: 根据KNN，获得对第index位用户评分的预测  
 """  
 neighbors = k\_neighbor(matrix, index, k)  
 all\_sim = 0  
 rate = [0 for i in range(len(ratings\_matrix[0]))]  
 for pair in neighbors:  
 neighbor\_index = pair[0]  
 neighbor\_sim = pair[1]  
 all\_sim += neighbor\_sim  
 rate += ratings\_matrix[neighbor\_index] \* neighbor\_sim  
 rate /= all\_sim  
 return rate

### 获得推荐

根据前一步得到的预测评分，对预测评分进行排序，向用户推荐预测评分最靠前的电影

需要注意的是，如果用户已经对某个电影评过分了，将不再重复推荐，因此需要在这里对将其预测评分修改为0

def get\_CFRecommend(matrix, index, k, n):  
 """  
 获取推荐  
 :param matrix: 相似矩阵  
 :param index: 目标index  
 :param k: k邻居的k  
 :param n: 获取推荐的topN  
 :return: list([movie\_index,预测评分],...)  
 """  
 rate = get\_predict(matrix, index, k) # 获取预测评分  
 for i in range(len(rate)): # 如果用户已经评分过了，把预测评分设为0，也就是不会再推荐看过的电影  
 if ratings\_matrix[index][i] != 0:  
 rate[i] = 0  
 res = []  
 for i in range(len(rate)):  
 res.append([i, rate[i]])  
 res.sort(key=lambda val: val[1], reverse=True)  
 return res[:n]

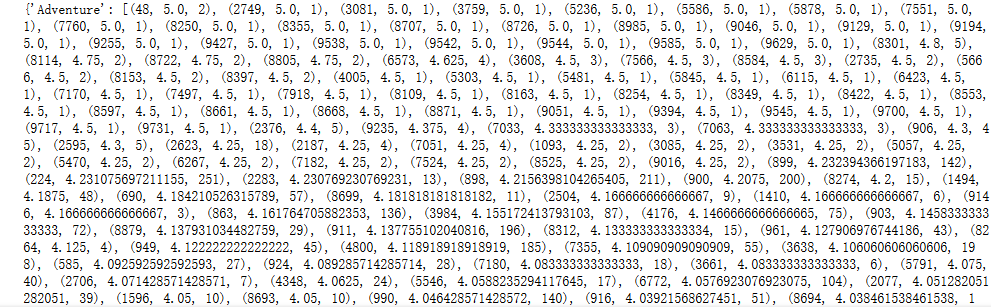
## 2.3 基于内容

### 每类电影排名

先根据评分高的优先，评分相同则评分人数多的优先

def type\_rank\_map():  
 """  
 计算每类电影排名  
 :return: map{'type':[(movie\_id,平均评分,评分人数),...],...}  
 """  
 map = {}  
 for t in type\_list:  
 map[t] = []  
 for movie in range(len(genres\_list)):  
 print('正在处理电影', movie)  
 # 计算该电影的用户均分  
 rates = np.array(ratings\_matrix)[:, movie]  
 count = 0  
 rate = 0  
 for r in rates:  
 if r != 0:  
 rate += r  
 count += 1  
 if count != 0: # 避免除0  
 rate = rate / count  
 # 将(电影,评分,评分人数)加到对应的map中  
 types = genres\_list[movie].split('|')  
 for t in types:  
 map[t].append((movie, rate, count))  
 # 排序，先根据评分高的优先，评分相同则评分人数多的优先  
 for t in type\_list:  
 temp = map[t]  
 temp.sort(key=lambda val: (val[1], val[2]), reverse=True)  
 map[t] = temp  
 return map  
  
type\_rank\_map = type\_rank\_map()  
import json  
with open('type\_rank\_map.json','w') as file\_obj:  
 json.dump(type\_rank\_map,file\_obj)

结果如下



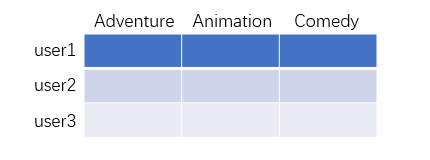
因为这一步需要计算很久，所以也将其存为文件type\_rank\_map.json

后续使用时只需要

type\_rank\_map = {}  
with open('type\_rank\_map.json') as file\_obj:  
 type\_rank\_map = json.load(file\_obj)

### 用户-电影类型偏好矩阵

为了获得用户对电影类型的偏好，需要构造如下的用户-电影类型偏好矩阵



构造算法为：

1. 假设用户对电影moviei进行评分为scorei，那么用户就会对moviei所属的电影类型增加scorei的喜爱度
2. 对每行进行归一化，也就是每位用户对所有电影类型的喜爱度之和为1

def get\_user\_favor\_matrix():  
 """  
 构造用户偏好矩阵  
 :return: [i,j]表示用户i对第j类型电影的喜爱程度  
 """  
 matrix = np.zeros((len(user\_list), len(type\_list)))  
 for user in range((len(user\_list))):  
 weight = 0  
 rating = ratings\_matrix[user]  
 for movie in range(len(rating)):  
 if rating[movie] != 0:  
 # update favor  
 types = genres\_list[movie].split('|')  
 for t in types:  
 if t in type\_map.keys():  
 matrix[user][type\_map[t]] += rating[movie]  
 weight += rating[movie]  
 matrix[user] /= weight  
 return matrix

### 推荐喜欢类型的高分电影

1. 根据用户-电影类型偏好矩阵，可以推算出用户最喜欢的电影类型
2. 根据每类电影排名，可以选择出每个类型下的高分电影，对用户进行推荐

需要注意的是：

* **为了避免过拟合，设置threshold，需要评分人数>threshold才可能被推荐**
* **为了避免重复推荐，不推荐用户评分过的电影**

def get\_CBRecommend(user\_index, user\_favor, type\_rank, threshold=10):  
 """  
 获得基于内容的推荐，就推荐一个  
 :param user\_index: 目标用户  
 :param user\_favor: 用户偏好矩阵  
 :param type\_rank: 每类电影排名map  
 :param threshold: 至少有threshold个人评分才算有效  
 :return: list([movie\_index,平均评分,评分人数],...)  
 """  
 favors = user\_favor[user\_index]  
 max\_val = 0  
 index = [] # 考虑如果有多个类型都一样喜欢，那么就挑可选出的评分最高的  
 for i in range(len(favors)):  
 if max\_val != 0 and favors[i] == max\_val:  
 index.append(i)  
 elif favors[i] > max\_val:  
 max\_val = favors[i]  
 index = [i]  
 candidate = []  
 for i in index:  
 tmp = type\_rank[type\_map\_reverse[i]] # 获取到该类排名list  
 for movie in tmp:  
 if movie[2] > threshold and ratings\_matrix[user\_index][movie[0]] != 0:  
 # 必须满足评分人数>threshold且用户没有看过  
 candidate.append(movie)  
 break  
 # 排序，选择最优的  
 candidate.sort(key=lambda val: (val[1], val[2]))  
 return candidate[0]

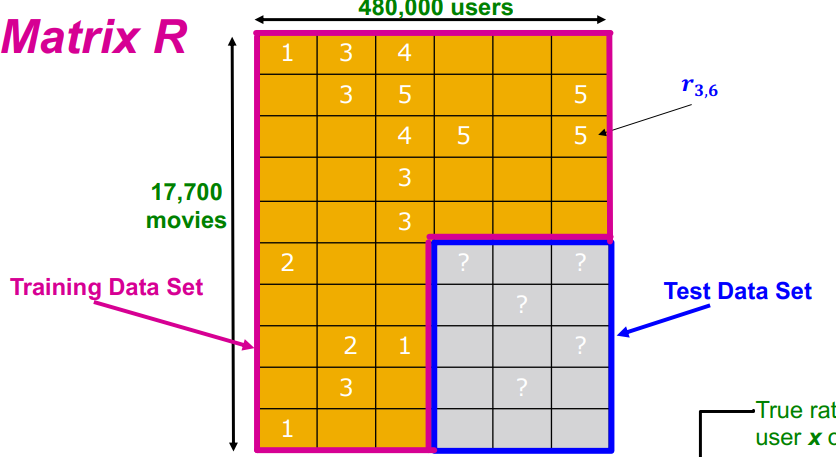
## 2.4 保存结果

if \_\_name\_\_ == '\_\_main\_\_':  
 output = [['userId', 'movieId']]  
 user\_sim\_matrix\_by\_rating = np.loadtxt(open("./user\_sim\_matrix.csv", "rb"), delimiter=",")  
 for user in user\_list:  
 res1 = get\_CFRecommend(user\_sim\_matrix\_by\_rating, user\_map[user], 10, 1)[0]  
 res2 = get\_CBRecommend(user\_map[user], user\_favor\_matrix, type\_rank\_map)  
 if res1[0] != res2[0]:  
 output.append([user, movie\_map\_reverse[res1[0]]])  
 output.append([user, movie\_map\_reverse[res2[0]]])  
 else:  
 output.append([user, movie\_map\_reverse[res1[0]]])  
 np.savetxt('movie.csv', output, delimiter=',', fmt="%s")

# 3 模型评估

这里对协同过滤算法的质量进行评估

## 3.1 数据划分



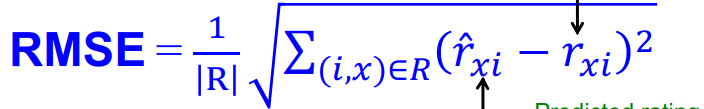
使用如下方法对评分矩阵进行划分，比例为0.2

即后20%的用户对后20%的电影评分为测试集

## 3.2 测试方法

使用RMSE、Coverage作为评价指标进行模型评估

### RMSE



### Coverage

测试推荐的覆盖率，即为每位用户推荐预测评分top10的电影的并集与电影总集合的比例

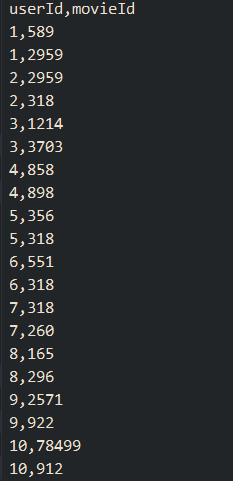
def evaluation(user\_sim\_matrix, split=0.2):   
 """   
 评估推荐模型准确度   
 :param user\_sim\_matrix: 用户相似度矩阵   
 :param split: 测试集比例   
 :return: RMSE计算结果   
 """   
 n = 0   
 res = 0   
 user\_start = int(len(user\_list) \* (1 - split))   
 movie\_start = int(len(movie\_list) \* (1 - split))   
 cover = {} # 计算覆盖率，标记推荐的电影列表   
 for user\_index in range(user\_start, len(user\_list)):   
 predict = get\_predict(user\_sim\_matrix, user\_index, 10)   
 for movie\_index in range(movie\_start, len(movie\_list)):   
 if ratings\_matrix[user\_index][movie\_index] != 0:   
 res += (predict[movie\_index] - ratings\_matrix[user\_index][movie\_index]) \*\* 2   
 n += 1   
 for user\_index in range(len(user\_list)):   
 recommend = get\_CFRecommend(user\_sim\_matrix, user\_index, 10, 10)   
 for movie in recommend:   
 cover[movie[0]] = 1   
 print(cover.keys())   
 cover\_rate = len(cover.keys()) / len(movie\_list)   
 print('RMSE={},Coverage={}'.format(math.sqrt(res / n), cover\_rate))

## 3.3 测试结果



# 4 推荐结果

结果数据结构如图所示



保存在movie.csv中，对每个用户推荐两个电影，分别由协同过滤和基于内容的推荐算法生成。