实验5 推荐系统

1. 实验目的

通过本次实验，学习和掌握基于用户的协同过滤算法和基于物品的协同过滤算法，实现一个简单的推荐系统。

1. 实验步骤
2. 对照基于用户的协同过滤算法的伪代码，完善代码注释。
3. 在样例代码的数据集上，实现基本的基于物品的协同过滤算法（IBCF）。
4. 实验结果展示

3.1 基于用户的协同过滤算法说明

算法流程：

A) 针对「目标用户」没有看过的电影，寻找看过电影的用户，称为「候选用户」；

B) 在这些「候选用户」中选出前 Nn 个用户，称为「邻居用户」；

C) 获得「邻居用户」与「目标用户」的相似度；

D) 计算每个 「邻居用户」 对自己看过的电影的评分的平均值；

E) 利用用户相似度、「邻居用户」的评分平均值、「目标用户」的评分平均值，计算获得电影的评分；

F) 对每个「目标用户」未看过的电影的评分进行排序，推荐 topN 的电影。

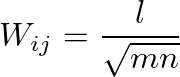
3.2 基于物品的协同过滤算法说明

算法流程：

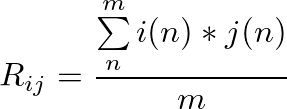
1. 计算电影之间的相似度

对任意的电影对：电影 i 和电影 j

1. 获得喜欢电影 i 的用户数量 m
2. 获得喜欢电影 j 的用户数量 n
3. 获得同时喜欢电影 i 和 j 的用户数量 l
4. 相似度



1. 计算用户对没有行为的电影的评分
   1. 获得「目标用户」 u 有过行为的电影集合 i
   2. 获得集合 i 中各部电影与「目标电影」 j 之间的相似度集合 k
   3. 电影评分

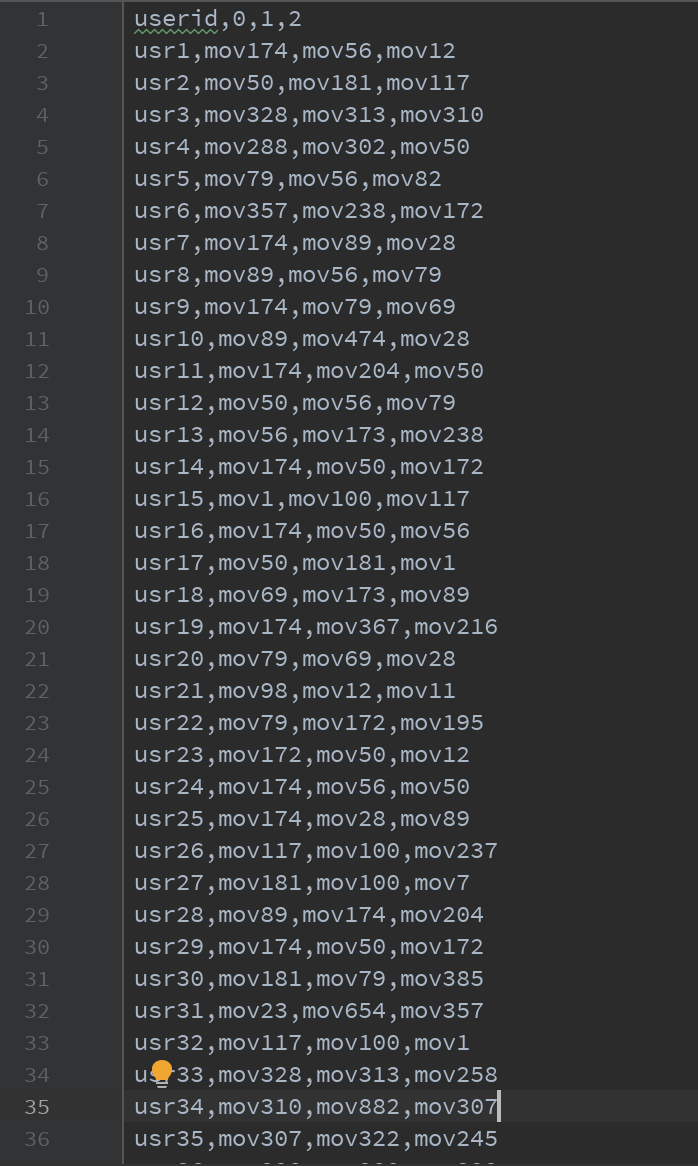


1. 生成推荐列表

3.3 基于物品的协同过滤算法实现

import pandas as pd  
  
  
def read\_data(train\_path, test\_path, pre\_process=True):  
 """ 读取数据  
  
 :param train\_path:  
 :param test\_path:  
 :param pre\_process:  
 :return:  
 """  
 train\_data = pd.read\_csv(train\_path, sep='\t', header=None, index\_col=None)  
 test\_data = pd.read\_csv(test\_path, sep='\t', header=None, index\_col=None)  
 if pre\_process:  
 train\_data.drop(3, axis=1, inplace=True)  
 test\_data.drop(3, axis=1, inplace=True)  
 train\_data.rename(columns={0: 'userid', 1: 'movid', 2: 'rat'}, inplace=True)  
 test\_data.rename(columns={0: 'userid', 1: 'movid', 2: 'rat'}, inplace=True)  
 return train\_data, test\_data  
  
  
def data\_reformat(data):  
 """ 数据格式转换  
  
 :param data:  
 :return:  
 """  
 re\_df = data.pivot(index='userid', columns='movid', values='rat')  
 re\_df.rename(index={i: 'usr%d' % i for i in re\_df.index}, inplace=True)  
 re\_df.rename(columns={i: 'mov%d' % i for i in re\_df.columns}, inplace=True)  
 return re\_df  
  
  
def get\_movie\_counter(MU\_rat):  
 """ 计算喜欢目标电影的用户数量  
  
 分数大于 2.5 视作喜欢  
  
 :param MU\_rat: 电影-用户 评分信息表  
 :return: M\_counter 每部电影 的喜欢用户数量  
 """  
 M\_counter = pd.DataFrame(index=MU\_rat.index, columns=["num"])  
 for mov in MU\_rat.index:  
 median = MU\_rat.loc[mov][:].dropna()  
 M\_counter.loc[mov][0] = len(median.loc[median > 2.5])  
 return M\_counter  
  
  
def get\_movie\_movie\_counter(UM\_rat):  
 """ 对任意两部电影 i 和 j ，计算有多少用户同时喜欢  
  
 :param UM\_rat: 用户-电影 评分信息表  
 :return: MM\_counter： 同时喜欢电影 i 和电影 j 的用户数量表  
 """  
 MM\_counter = pd.DataFrame(index=UM\_rat.columns, columns=UM\_rat.columns)  
 MM\_counter.loc[:][:] = 0  
 for user in UM\_rat.index:  
 ratted\_movies = UM\_rat.loc[user].dropna()  
 ratted\_movies = ratted\_movies.loc[ratted\_movies > 2.5]  
 for fi in range(len(ratted\_movies)):  
 fi\_movie = ratted\_movies.index[fi]  
 for se in range(fi + 1, len(ratted\_movies)):  
 se\_movie = ratted\_movies.index[se]  
 MM\_counter.loc[fi\_movie][se\_movie] += 1  
 MM\_counter.loc[se\_movie][fi\_movie] += 1  
 print("user index (MM counter, 462 total) : ", user)  
 return MM\_counter  
  
  
def calculate\_similarity(M\_counter, MM\_counter):  
 """ 计算电影之间的相似度  
  
 :param M\_counter: 每部电影 的喜欢用户数量  
 :param MM\_counter: 同时喜欢电影 i 和电影 j 的用户数量表  
 :return: MM\_similarity: 电影之间的相似度表  
 """  
 MM\_similarity = pd.DataFrame(index=M\_counter.index, columns=M\_counter.index, dtype=float)  
 MM\_similarity.loc[:] = 1.0  
 for fi in range(len(M\_counter)):  
 fi\_movie = MM\_counter.index[fi]  
 for se in range(fi+1, len(M\_counter)):  
 se\_movie = MM\_counter.index[se]  
 similarity = float(MM\_counter.loc[fi\_movie][se\_movie]) / \  
 ((M\_counter.loc[fi\_movie][0] \* M\_counter.loc[se\_movie][0] + 1) \*\* 0.5)  
 MM\_similarity.loc[fi\_movie, se\_movie] = similarity  
 MM\_similarity.loc[se\_movie, fi\_movie] = similarity  
 print("complete (similarity) : ", fi / len(M\_counter) \* 100)  
 return MM\_similarity  
  
  
def predict\_rat(UM\_rat, MM\_similarity):  
 """ 预测目标用户对其没有行为的电影的评分  
  
 1. 获得目标用户 u 有过行为的电影集合 i  
 2. 获得集合 i 中各部电影与目标电影 j 之间的相似度集合 k  
 3. 取集合 i 和集合 k 的元素，两两对应相乘，并对乘积求和，最后计算平均和 l  
 将 l 视作目标电影 j 的评分:  
 ∑ i(n)\*j(n)  
 l = ------------- n ∈ (1, m)  
 m  
  
 如此求出的评分，并不满足 5 分制，仅仅为推荐排序做依据参考  
  
 返回的数据结构：  
 user\_new\_rats [user, movies, new\_rats]  
 用户索引，用户没有行为的电影列表，电影对应的评分列表  
  
 :param UM\_rat:  
 :param MM\_similarity:  
 :return: user\_new\_rats: 预测的评分信息  
 """  
 user\_new\_rats = []  
 for user in UM\_rat.index:  
 M\_rat = UM\_rat.loc[user]  
 M\_rat\_boolean = M\_rat.isnull()  
 ratted\_movies = M\_rat[M\_rat\_boolean.loc[:] == False]  
 unratted\_movies = M\_rat[M\_rat\_boolean.loc[:] == True]  
 # 循环没有看过的电影  
 movies = []  
 new\_rats = []  
 for un\_mov in unratted\_movies.index:  
 similarity = MM\_similarity.loc[un\_mov][ratted\_movies.index]  
 res = similarity \* ratted\_movies  
 new\_rat = sum(res) / len(similarity)  
 new\_rats.append(new\_rat)  
 movies.append(un\_mov)  
 user\_new\_rats.append([user, movies, new\_rats])  
 print("user index (predict\_rat): ", user)  
 return user\_new\_rats  
  
  
def recommend\_movies(UM\_rat, user\_new\_rats, num):  
 """ 决定推荐的电影  
  
 :param UM\_rat: 用户-电影 评分表  
 :param user\_new\_rats: 预测的评分信息  
 :param num: 推荐的电影数量  
 :return: U\_recommend: 推荐的电影表  
 """  
 U\_recommend = pd.DataFrame(index=UM\_rat.index,  
 columns=[i for i in range(num)], dtype='object')  
 for user\_info in user\_new\_rats:  
 user = user\_info[0]  
 movies = user\_info[1]  
 new\_rats = user\_info[2]  
 for i in range(num):  
 max\_index = new\_rats.index(max(new\_rats))  
 U\_recommend.loc[user][i] = movies[max\_index]  
 del movies[max\_index]  
 del new\_rats[max\_index]  
 print("user index (recommend\_movies): ", user)  
 return U\_recommend  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 train\_path = '../data/u1.base'  
 test\_path = '../data/u1.test'  
 train, test = read\_data(train\_path, test\_path)  
 UM\_rat\_train = data\_reformat(train)  
 UM\_rat\_test = data\_reformat(test)  
 UM\_rat = UM\_rat\_train.loc[UM\_rat\_test.index]  
 MU\_rat = UM\_rat.T  
 print("preprocess complete")  
  
 print("M\_counter")  
 # M\_counter\_path M\_counter 保存路径  
 M\_counter\_path = '../data/M\_counter.csv'  
 M\_counter = get\_movie\_counter(MU\_rat)  
 M\_counter.to\_csv(M\_counter\_path)  
 M\_counter = pd.read\_csv(M\_counter\_path, sep=',', index\_col='movid')  
  
 print("MM\_counter")  
 # MM\_counter\_path MM\_counter 保存路径  
 MM\_counter\_path = '../data/MM\_counter.csv'  
 MM\_counter = get\_movie\_movie\_counter(UM\_rat)  
 MM\_counter.to\_csv(MM\_counter\_path)  
 MM\_counter = pd.read\_csv(MM\_counter\_path, sep=',', index\_col='movid')  
  
 print("MM\_similarity")  
 # MM\_similarity\_path MM\_similarity 保存路径  
 MM\_similarity\_path = '../data/MM\_similarity.csv'  
 MM\_similarity = calculate\_similarity(M\_counter, MM\_counter)  
 MM\_similarity.to\_csv(MM\_similarity\_path)  
 MM\_similarity = pd.read\_csv(MM\_similarity\_path, sep=',', index\_col='movid')  
  
 print("user\_new\_rats")  
 user\_new\_rats = predict\_rat(UM\_rat, MM\_similarity)  
  
 print("U\_recommend")  
 # U\_recommend\_path U\_recommend 保存路径  
 U\_recommend\_path = '../data/U\_recommend.csv'  
 U\_recommend = recommend\_movies(UM\_rat, user\_new\_rats, 3)  
 U\_recommend.to\_csv(U\_recommend\_path)  
  
 print("all complete")

3.4 基于物品的协同过滤算法实现结果



1. 实验总结与收获

通过本次实验，我学习和掌握了两种基础的基于用户的协同过滤算法和基于物品的协同过滤算法。