# 大作业 推荐系统任务书

## **5.1实验目的**

1. 了解推荐系统的多种推荐算法并理解其原理。
2. 实现**User-User的协同过滤算法**并对用户进行推荐。
3. 实现**基于内容的推荐算法**并对用户进行推荐。
4. 对两个算法进行电影预测评分对比
5. 在学有余力的情况下，**加入minihash算法**对效用矩阵**进行降维处理**

## **5.2 实验内容**

给定MovieLens数据集，包含电影评分，电影标签等文件，其中电影评分文件分为训练集train\_set和测试集test\_set两部分

基础版必做一：**基于用户的协同过滤推荐算法**

对训练集中的评分数据构造用户-电影效用矩阵，使用**pearson相似度计算方法计算用户之间的相似度**，也即相似度矩阵。**对单个用户进行推荐时，找到与其最相似的k个用户，用这k个用户的评分情况对当前用户的所有未评分电影进行评分预测，选取评分最高的n个电影进行推荐。预测评分按照以下方式计算：**

在测试集中包含100条用户-电影评分记录，用于计算推荐算法中预测评分的准确性，对测试集中的每个用户-电影需要**计算其预测评分**，**再和真实评分进行对比，误差计算使用SSE误差平方和**。

选做部分提示：此算法的进阶版采用minihash算法对效用矩阵进行降维处理，从而得到相似度矩阵，注意minihash采用jarcard方法计算相似度，需要对效用矩阵进行01处理，也即**将0.5-2.5的评分置为0，3.0-5.0的评分置为1**。

基础版必做二：**基于内容的推荐算法**

将数据集movies.csv中的电影类别作为特征值，**计算这些特征值的tf-idf值**，得到关于电影与特征值的n（电影个数）\*m（特征值个数）的**tf-idf特征矩阵**。根据得到的tf-idf特征矩阵，**用余弦相似度的计算方法，得到电影之间的相似度矩阵**。

对某个用户-电影进行预测评分时，获取当前用户的已经完成的所有电影的打分，通过电影相似度矩阵获得已打分电影与当前预测电影的相似度，按照下列方式进行打分计算：

**选取相似度大于零的值进行计算，如果已打分电影与当前预测用户-电影相似度大于零，加入计算集合，否则丢弃**。（相似度为负数的，强制设置为0，表示无相关）假设计算集合中一共有n个电影，score为我们预测的计算结果，score’(i)为计算集合中第i个电影的分数，sim(i)为第i个电影与当前用户-电影的相似度。如果n为零，则score为该用户所有已打分电影的平均值。

**要求能够对指定的userID用户进行电影推荐，推荐电影为预测评分排名前k的电影。userID与k值可以根据需求做更改。**

推荐算法准确值的判断：**对给出的测试集中对应的用户-电影进行预测评分，输出每一条预测评分，并与真实评分进行对比，误差计算使用SSE误差平方和**。

选做部分提示：进阶版采用minihash算法对特征矩阵进行降维处理，从而得到相似度矩阵，注意minihash采用jarcard方法计算相似度，特征矩阵应为01矩阵。因此进阶版的特征矩阵选取采用方式为，**如果该电影存在某特征值，则特征值为1，不存在则为0，从而得到01特征矩阵**。

**进阶部分：**

本次大作业的进阶部分是在基础版本完成的基础上大家可以尝试做的部分。进阶部分的主要内容是使用**迷你哈希（MiniHash）算法对协同过滤算法和基于内容推荐算法的相似度计算进行降维**。同学可以把迷你哈希的模块作为一种近似度的计算方式。

协同过滤算法和基于内容推荐算法都会涉及到相似度的计算，迷你哈希算法在牺牲一定准确度的情况下对相似度进行计算，其能够有效的降低维数，尤其是对大规模稀疏01矩阵。同学们可以**使用哈希函数或者随机数映射来计算哈希签名**。哈希签名可以计算物品之间的相似度。

最终降维后的维数等于我们定义映射函数的数量，我们设置的映射函数越少，整体计算量就越少，但是准确率就越低。**大家可以分析不同映射函数数量下，最终结果的准确率有什么差别**。

**对基于用户的协同过滤推荐算法和基于内容的推荐算法进行推荐效果对比和分析，选做的完成后再进行一次对比分析。**

## **5.3 实验过程**

### 5.3.1 编程思路

**基于用户的协同过滤推荐算法编程思路：**

本次实验我们需要运用到User-User的协同过滤算法，这是一种通过用户评分来计算相似性然后来进行推荐的算法。算法中相关关键系数和概念介绍如下：其一是相似性度量，用于衡量用户之间相似性的方法，比如余弦相似度、杰卡德相关系数、皮尔逊相关系数、聚类算法中提及的欧氏距离等等……这些系数可以用来衡量用户对同一个电影的评分的相似性。在本次实验中将采用皮尔逊相关系数来判断用户之间的相似度，皮尔逊相关系数的公式如下所示：

公式5.1 皮尔逊相关系数

其二是邻居选择，需要输入邻居的数量k，以来从这k个和被推荐用户相似的用户评分情况对当前用户没有评分的电影进行预测。其三是预测方法，本次实验中我们需要预测评分，预测方式如下公式所示：

公式5.2 预测评分公式

首先讲一下我理解的User-User协同过滤算法大概思路：首先要进行数据准备，读取对电影的评分数据，构造对应的矩阵；然后计算用户之间的相似性，通过皮尔逊相关系数来计算相似性的矩阵；再根据输入的k，来选择与被推荐用户最像的k个邻居用户；最后我们根据上面的公式去预测目标用户评分，再据此去为被推荐用户去推荐物品的列表。实验代码的函数以及大致流程如下图所示：

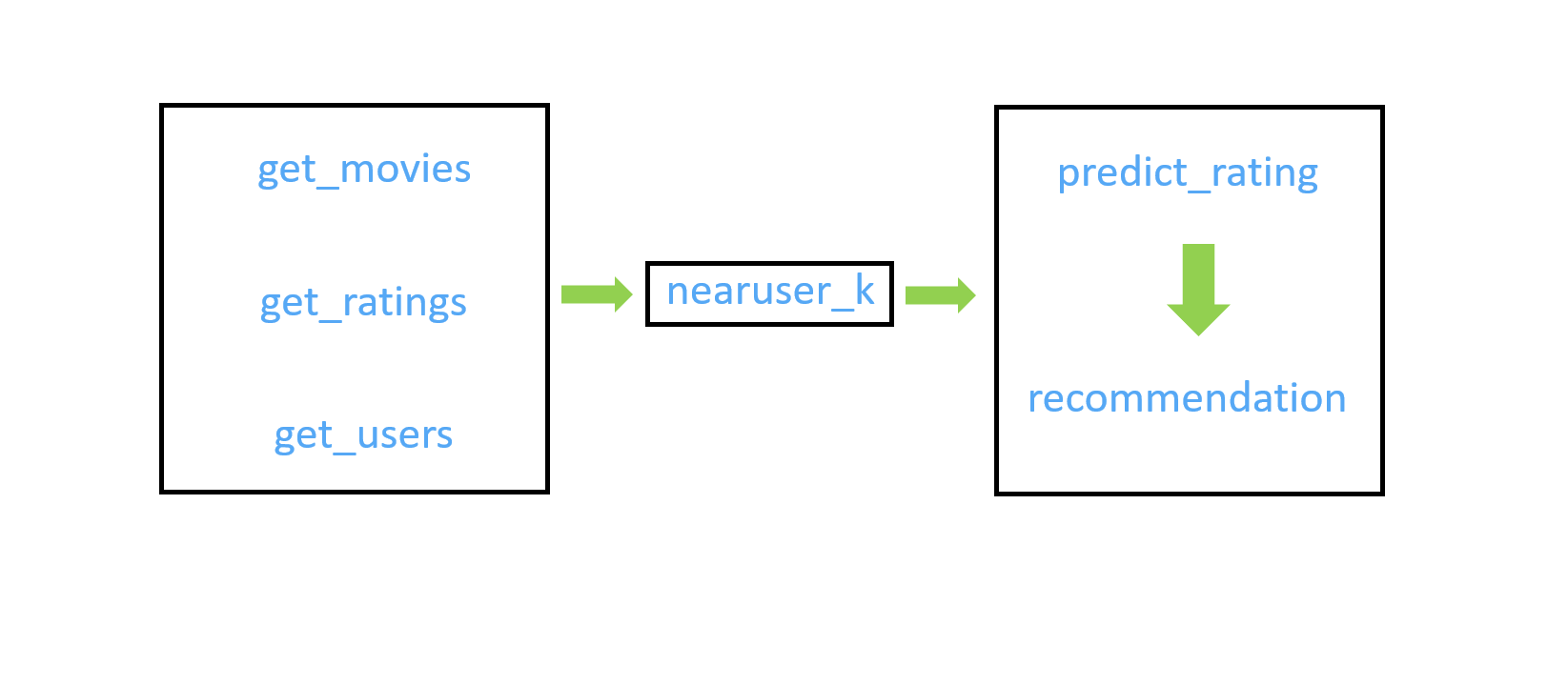
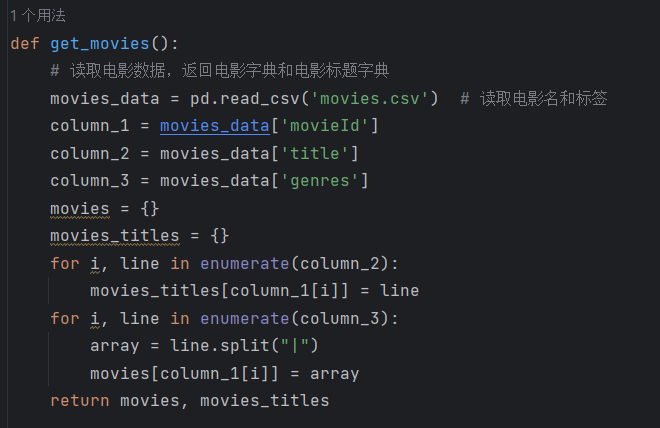


图5.1 实验函数大致流程

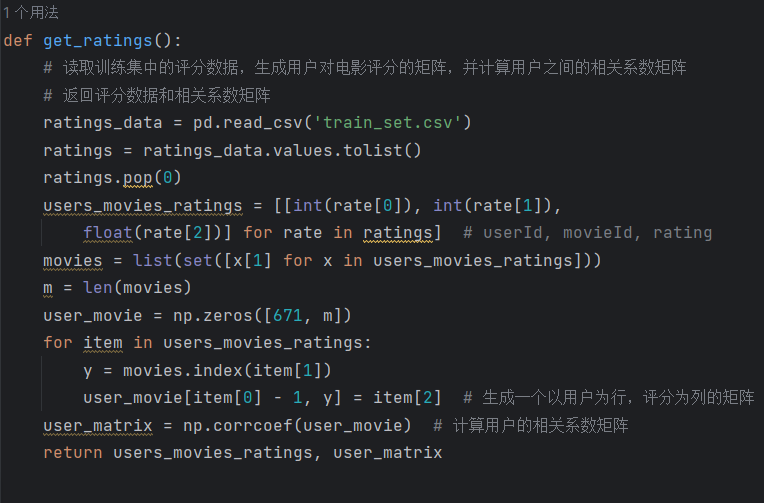
第一步定义**get\_movies()**函数来读取movies.csv文件，通过movieId来分别映射titles和genres，最后返回这两个映射。movies.csv文件中第一列是movieId，第二、三列分别是titles、genres。相关函数代码如下：



**通过movieId映射**

图5.2 get\_movies()函数

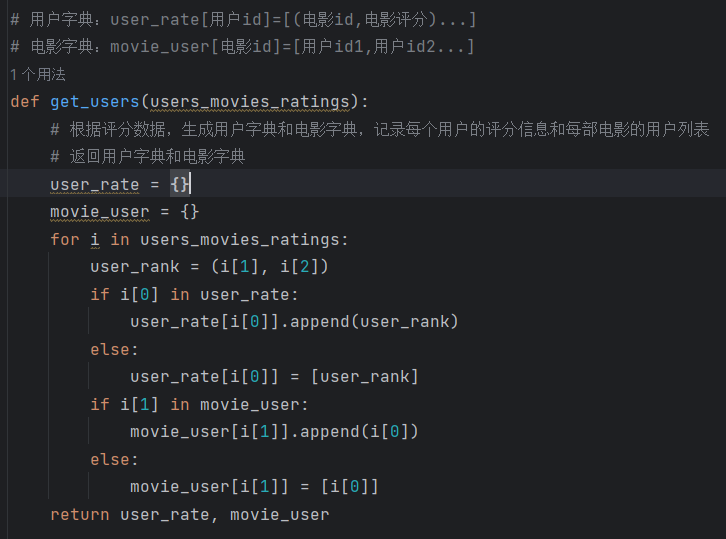
第二步定义**get\_ratings()**函数来生成用户-评分的矩阵、用户-用户之间的相关矩阵，返回评分数据和相关系数矩阵。对于用户-评分矩阵，我们可以通过for循环，来完成这个矩阵；对于用户-用户矩阵，我们可以直接使用numpy库中的corroef函数来计算用户与用户之间的皮尔逊相关系数的矩阵。相关代码如下图所示：



**构造两个矩阵**

图5.3 get\_ratings()函数具体实现

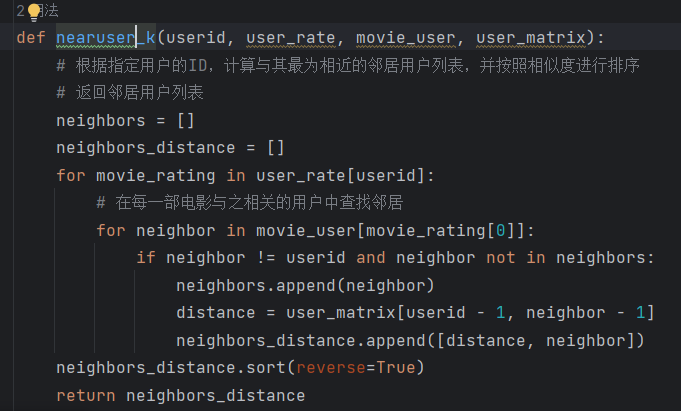
第三步定义**get\_users(users\_movies\_rating)**函数，通过上一个函数生成的用户-电影评分数据，在这一函数中去生成用户和电影的字典记录每个用户的评分信息和每部电影的用户列表，并返回这两个列表。相关代码如下图所示：



**构造两个字典**

图5.4 get\_users(users\_movies\_rating)函数具体实现

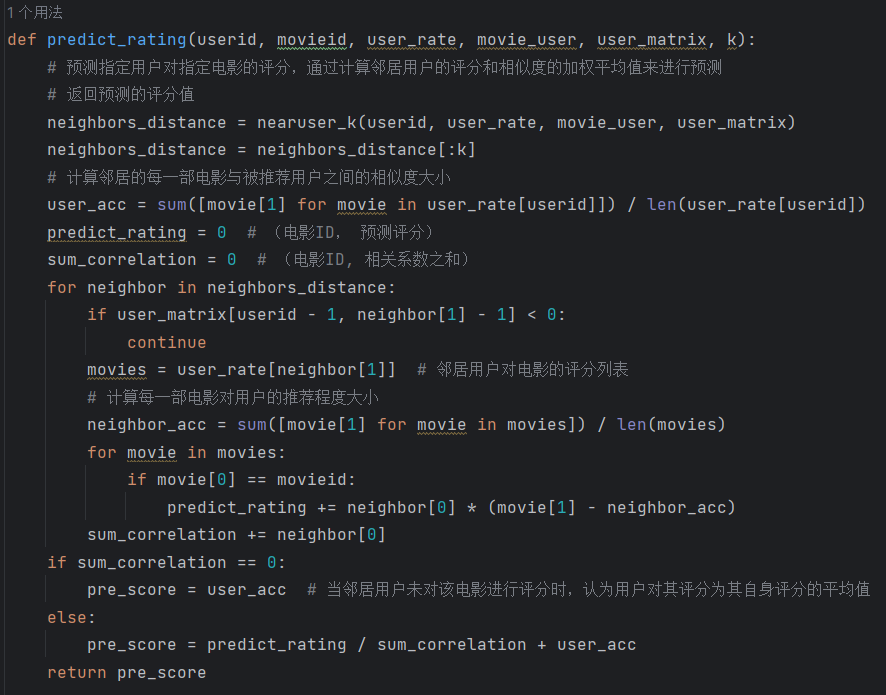
第四步定义**nearuser\_k(userid, user\_rate, movie\_user, user\_matrix)**函数。userid是被推荐者的ID，计算他的邻居用户列表；user\_rate是用户字典，可以查询每个用户的评分记录；movie\_user是电影字典，可以查询对此电影评分了的用户；user\_matrix是用户的皮尔逊相关系数矩阵。通过遍历在同一部电影下的所有的用户，去计算被推荐者对这个用户的距离（按照相似度来计算）。将所有和被推荐者有关的全部计算出来，然后按照相似度的由大到小顺序排序。相关代码如下图所示：



**生成邻近用户列表**

图5.5 nearuser\_k(userid, user\_rate, movie\_user, user\_matrix)函数具体实现

第五步定义**predict\_rating(userid, movieid, user\_rate, movie\_user, user\_matrix, k)**函数。相关参数有：userid被推荐者的id，计算预测他的指定电影评分；movieid是指定的电影id；user\_rate是用户字典，可以查询每个用户的评分记录；movie\_user是电影字典，可以查询对此电影评分了的用户；user\_matrix是用户的皮尔逊相关系数矩阵；k是预测时查询被推荐者k个邻近用户。通过公式来计算每部电影和被推荐用户之间的相似度大小。通过for循环遍历每个邻居，判断如果相似度（user\_matrix）小于零则跳过，否则获取这个用户的电影评分列表（movies）。接着计算每个邻居对电影的推荐程度，计算neighbor对所有电影评分的平均值，然后计算这个平均分和指定电影的评分差值，这个差值（predict\_rating）和邻居对被推荐者的相似度（neighbor[0]）进行加权计算，将最后的结果累加在邻居的相似度系数之和（sum\_correlation）。用于后续给被推荐者的推荐一些电影。相关代码如下图所示：



**计算邻居用户的相似度之和**

图5.6 predict\_rating(userid, movieid, user\_rate,

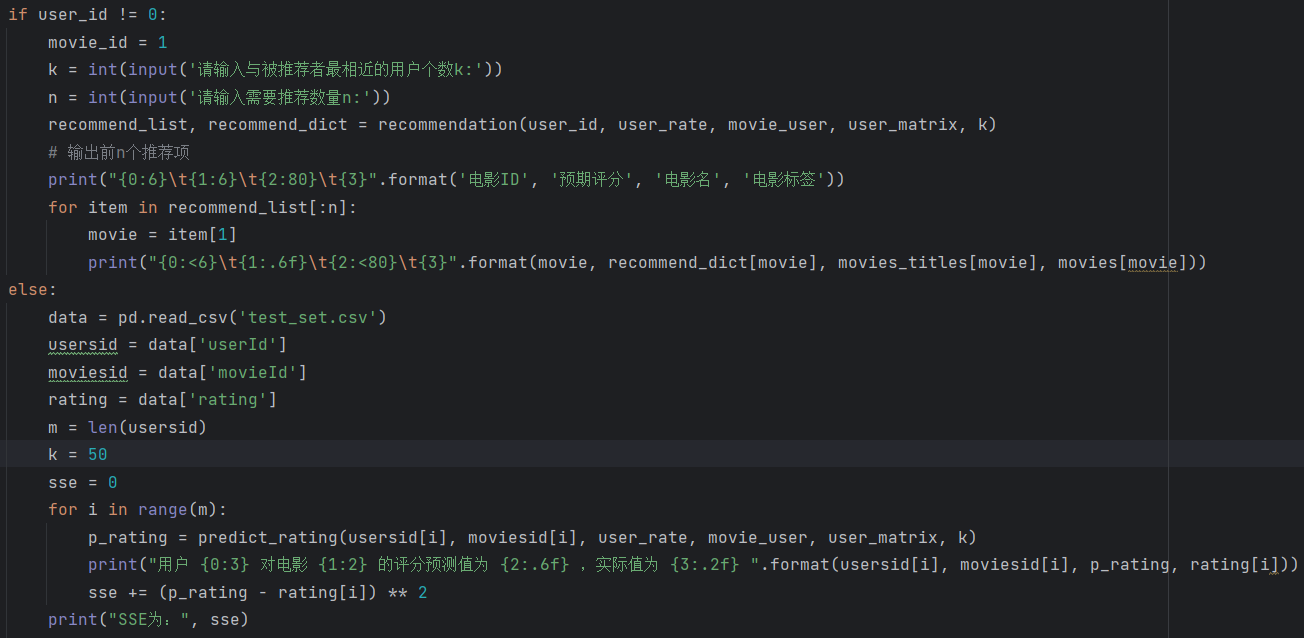
movie\_user, user\_matrix, k)函数具体实现

第六步定义**recommendation(userid, user\_rate, movie\_user, user\_matrix, k)**函数，这个函数的参数和上一个函数意义大致相同。这个函数基于用户之间的相似度，为指定用户生成推荐电影列表，并且计算推荐程度。为了计算每个邻居对电影的推荐程度大小，首先计算该邻居对所有电影的评分之和除以电影数量，得到邻居的电影评分的平均值。然后遍历邻居的评分列表，我们只用去看当前的指定电影，计算这个邻居对这个电影的推荐程度大小，还要将这些相似度累加。在计算推荐列表的时候，我们对每个电影去计算它的推荐程度，将用户-电影的相关系数之和去除以套公式计算的相关系数之和，在加上用户的平均评分得出一个电影的推荐程度，然后再将这个推荐表按照由大到小排序方便后续推荐。相关代码如下图所示：



图5.7 recommendation(userid, user\_rate, movie\_user, user\_matrix, k)函数具体实现

第七步编写main函数，分别实现测试集测试(test\_set.csv)和SSE，以及输入被推荐者ID，用于计算的邻居个数k，需要推荐的电影个数n，就可以输出n个推荐的电影、电影ID、电影预期评分、电影标签。我们根据用户第一个输入的数字使用if循环来判断，若0为测试集，不为0就是去给推荐者推荐n个电影。对于测试集而言，只需要先获得文件中的数据，遍历所有的用户去计算每个预测的评分即可。相关代码如下图所示：



**计算测试集中的预测评分**

图5.8 User-User算法主体运行部分

**基于内容的推荐算法编程思路：**

在这个算法中我们用来衡量的相似度为余弦相似度，其中余弦相似度的计算公式为如下所示：

公式5.3 余弦相似度计算公式

在这个算法中我们将把数据集movies.csv中的电影类别作为特征值，计算这些特征值的TF-IDF值，得到关于电影与特征值的TF-IDF特征矩阵。其中TF\_IDF计算公式如下图所示：

公式5.4 TF-IDF计算公式

首先，对于基于内容的推荐算法，我理解的流程如下：首先要将csv文件中的数据读取、进行数据准备，并且还要将每个电影类别作为特征值，求出一个特征向量；接着要进行TF-IDF计算，其中TF用于表示某个特征（类别）在电影中出现的频率，IDF表示这个特征在数据集中的重要性，而TF-IDF为这两者的乘积；然后通过构造特征矩阵，形成每个类别与每个电影在某个特征上的TF-IDF值；通过余弦相似度来计算每个电影之间的相似度得到一个两个电影之间相似度的矩阵，通过这个矩阵来进行推荐以及预测评分。实验代码的函数以及大致流程如下图所示：

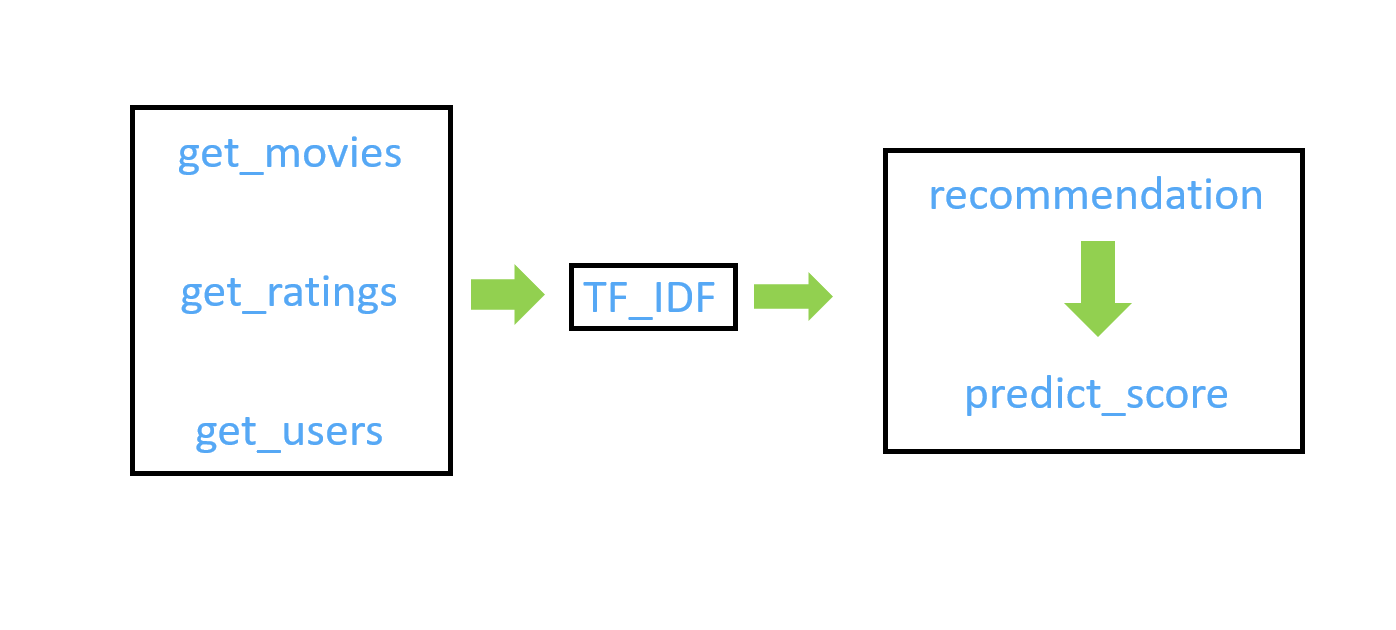


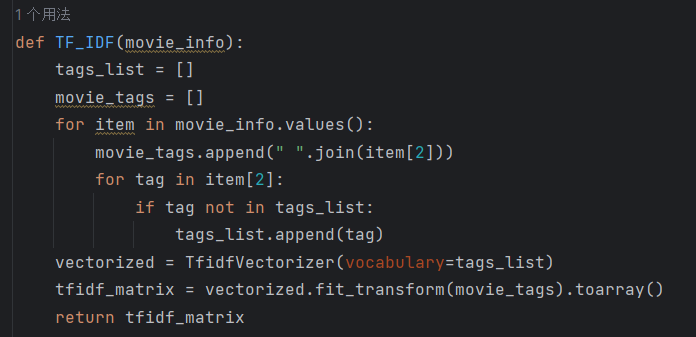
图5.9 基于内容的推荐算法函数大致流程

第一步定义**get\_ratings()**函数，负责从“train\_set.csv”文件读取用户的评分信息，并且读取以用户为行、电影评分为列的矩阵user\_movie。返回值r中存储userId，movieId，rating这三列数据。

第二步定义**get\_user(r)**函数，通过第一步函数返回的r用户-电影-评分列表，生成两个字典数据：user\_rate和user\_movie。具体的内容为： user\_rate[用户id]=[(电影id,电影评分)...]； movie\_user[电影id]=[用户id1,用户id2...]。

第三步定义**get\_movie\_info()**函数，这个函数和之前的算法类似，从movies.csv文件中读取电影信息（movieId，title，genre），将相关信息保存在movie\_info和movie\_ID之中，最后也返回这两个对象。

第四步定义**TF\_IDF(movie\_info)**函数，在这一步我们可以简化计算步骤，使用scikit-learn库中TfidfVectorizer函数。这个函数可以将文本集合转换成我们所需要的TF-IDF特征矩阵，相对于我们自己编写的多重for循环（所需时间比较久），时间复杂度减少了许多（改进后缩短至几秒以内）。相关函数如下图所示：



**构造TF-IDF特征矩阵**

图5.10 TF\_IDF(movie\_info)函数具体实现

第五步定义recommendation(user\_rate, user\_id, movie\_ID, cos\_sim, user\_movie)函数，这里面的参数前面函数都已经提及过这里就不再赘述。这个函数可以根据这些参数，为这个用户生成推荐电影列表。并且根据用户的评分和电影类别的相似度来计算预期评分并进行排序，这样方便主函数中给用户推荐电影。函数返回recommend\_list和recommend\_dict。

第六步定义predict\_score(user\_rate, user\_id, movie\_ID, cos\_sim, movieid)函数。这个函数根据传入的参数，以及用户的评分和电影类别的相似度来预测用户对某个电影的评分，最后返回pre\_score。得到用户的预测评分。

### 5.3.2 遇到的问题及解决方式

**问题一：基于内容的推荐算法运行速度相对于User-User的协同过滤算法较慢原因有哪些？**

比较明显的问题在于TF-IDF特征矩阵计算时，多个for循环极大的延缓算法的计算速度。我们可以通过使用TfidfVectorizer类，直接从movie-genre表中获得tf-idf特征矩阵。这个类所在的库经过了优化调优，并且利用了并行计算的能力，加快处理数据的速度。

### 5.3.3 实验测试与结果分析

**User-User的协同过滤算法测试集：**



图5.11 User-User的协同过滤算法测试集结果

**基于内容的推荐算法测试集：**

图5.12 基于内容的推荐算法测试集结果

对于相同的测试集，两种算法有截然不同的两种结果：不仅对电影的评分预测不同，算法结果的SSE值也不相同。从测试结果我们得知User-User的协同过滤算法测试集结果比较好（因为评分预测值差值并不大），可能原因有以下几条：其一是算法中可能忽略了电影类别的关系，基于内容的推荐算法更加关注于电影本身的内容特征；其二是可能是还需要更多用户对电影的类别选择的数据来改进最后的结果；其三是可能用户的观影选择还有更多的影响因素，也就是电影的类别并不能很好的概括用户的偏好导致推荐不能很好的满足用户的需求，最终导致基于内容的推荐算法的SSE值比较大。

通过输入被推荐者用户ID为5，需要推荐5个电影，在User-User的协同过滤算法只观察最好的5个邻居用户，我们得出以下的测试结果：

**User-User的协同过滤算法输入（5，5，5）：**

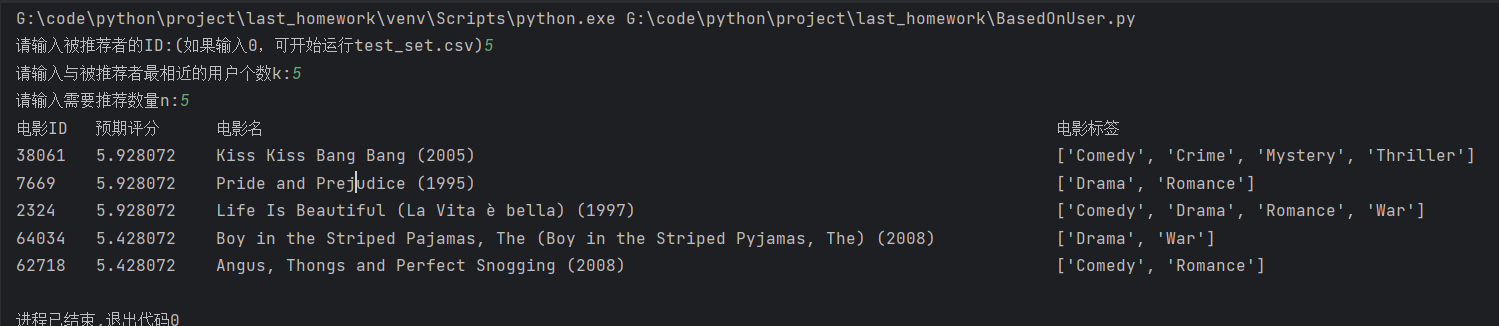


图5.13 User-User的协同过滤算法（用户ID 5，相邻5个邻居、推荐5个电影）

**基于内容的推荐算法输入（5，5，5）：**

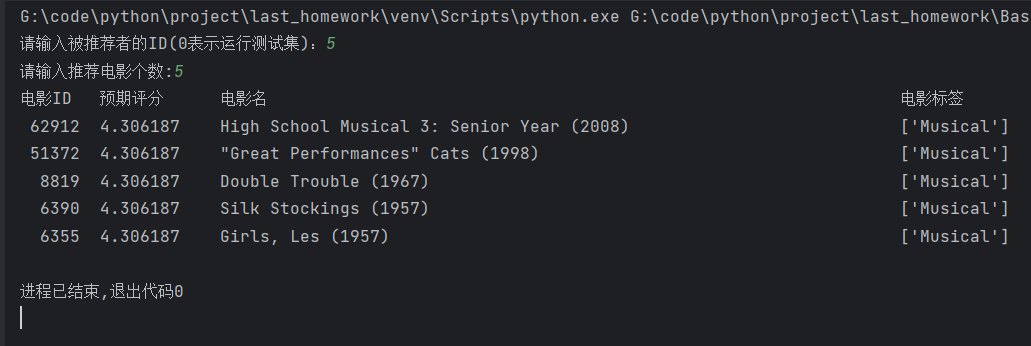


图5.14 基于内容的推荐算法（用户ID 5，推荐5个电影）

我们可以明显观察到基于内容的推荐算法结果在电影标签上是统一的‘Musical’。基于内容的推荐算法更加关注用户的个性化、偏好，考虑了用户对电影的喜好以及电影本身的类别，减少对用户评分的关注；User-User的协同过滤算法更加关注于用户评分以及邻居评分和相似系数等数据。

## **5.4 实验总结**

本次实验我们主要使用User-User的协同过滤算法和基于内容的推荐算法，并且通过测试结果来比较衡量两者的相似之处与不同之处。从名字上我们很明显知道User-User的协同过滤算法是**从用户的角度（user）**，以用户评分以及用户的相似度来推荐以及预测电影；而基于内容的推荐算法是**从电影特征的角度（genre）**，来推荐以及预测特定用户的电影。由于角度的不同，二者更好解决的问题也不相同，前者可以来应对类似**user-movie的推荐问题**，比如网购平台中的“猜你想买”板块；后者可以用来应对类似**movie-movie的推荐问题**，比如网购平台在商品界面中的“类似商品推荐”板块。在计算推荐优先度的时候，我们需要用到一个比较重要的系数——相似度系数。前者需要使用皮尔逊相关系数来计算用户之间的相似度，后者需要使用TF-IDF以及余弦相似度来计算电影（genre）之间的相似度。从本次实验中，我了解到一些关于通过大数据来进行用户推荐的两种方法，并且不同方法之间有各自的优势区间。根据不同的大数据类型、使用场景以及希望的功能，我们需要灵活去选择不同的推荐算法。这也体现出对大量数据进行初步处理的重要性。