**专业综合设计（大数据方向）课程报告**

2022-2023学年第1学期

|  |  |  |  |
| --- | --- | --- | --- |
| 项目评分表 | | | |
| 考察项目 | 评分项目 | 分值 | 得分 |
| 项目代码  （30分） | 满足项目功能需求，能以较快速度进行计算 | 15 |  |
|
| 代码结构合理，规范，易维护 | 15 |  |
|
| 项目答辩  （20分） | 解决方案的合理性 | 10 |  |
|
| 收获与体会 | 10 |  |
|
| 项目报告  （30分） | 问题分析与抽象，解决方案 | 5 |  |
| 设计系统结构，确定软硬件组成，确定系统边界 | 5 |  |
| 项目社会、健康、安全、法律、文化以及环境等可行性分析 | 4 |  |
| 技术规范标准适用分析，确定系统的主要技术指标 | 4 |  |
| 项目实施对法律、安全、健康、伦理与文化所产生的潜在影响分析及解决方案 | 4 |  |
| 团队成员职责分工与协调组织 | 4 |  |
| 软硬件选型的技术与经济性分析 | 4 |  |
| 项目总结  （10分） | 问题的抽象与分析 | 2 |  |
| 项目创新点说明与分析 | 2 |  |
| 技术指标说明，技术合理性分析 | 2 |  |
| 团队分工协作说明，成员完成任务情况说明 | 2 |  |
| 项目中工程管理方法与经济决策方法的使用说明 | 2 |  |
| 项目得分 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学生成绩表 | | | | |
| 学号 | 姓名 | 团队作用 | 组内评分  （10分） | 合计成绩[[1]](#footnote-0) |
| 20194285 | 兰庆秋 | 后端开发人员 | 10 |  |
| 20194284 | 秦霄潇 | 前端开发人员 | 10 |  |
| 20194231 | 章俊文 | 核心算法开发人员 | 10 |  |
| 20194281 | 李思颀 | 核心算法开发人员 | 10 |  |
| 20194202 | 周飞燕 | 前端开发人员 | 10 |  |
| 评阅教师： | | | | |
| 评阅时间：2023年1月6日 | | | | |

|  |  |
| --- | --- |
| 专业综合设计任务书 | |
| 名称 | 电影推荐系统的设计与实现 |
| 类型 | □验证性 □设计性 综合性 |
| 内容 | （1）模仿MovieLens（https://movielens.org）构建一个电影评分网站；  （2）利用MovieLens Latest数据集（265MB，下载地址：http://files.grouplens.org/datasets/movielens/ml-latest.zip）作为网站的初始数据；  （3）采用推荐系统算法，计算用户偏好，根据用户的偏好向用户推荐电影；  （4）网站界面美观，电影推荐实时性较好，能以较少的内存运行。 |
| 要求 | （1）利用Docker或者云计算平台搭建分布式的数据处理集群，建议使用Spark作为大数据的处理引擎；  （2）推荐算法必须自己实现，并能在分布式数据处理平台上运行，**不允许**调用现有的软件库，**禁止使用**Spark中的推荐系统算法；  （3）数据存储使用NoSQL数据库，例如MongoDB；  （4）在处理大数据时，具有良好的性能，服务响应时间较短，并在报告中对算法的理论时间复杂度、空间复杂度和实际运行时间进行分析；  （5）推荐算法应具备处理MovieLens Latest数据集（265MB）的能力。 |
| 小组成员 | 20194285 计算机科学与技术专业 兰庆秋  20194284 计算机科学与技术专业 秦霄潇  20194231 计算机科学与技术专业 章俊文  20194281 计算机科学与技术专业 李思颀  20194202 计算机科学与技术专业 周飞燕 |
| 任务时间 | 2023年 1月 6日 |

目录

[专业综合设计（大数据方向）课程报告 1](#_Toc159)

[一、项目背景介绍 6](#_Toc11792)

[1.1 项目简介 6](#_Toc17494)

[1.2 国内外研究现状 6](#_Toc4837)

[二、 需求分析 7](#_Toc27519)

[2.1 数据处理需求 7](#_Toc27535)

[2.2 数据存储需求 8](#_Toc2484)

[2.2.1 静态数据 8](#_Toc24693)

[2.2.2 历史数据 8](#_Toc25617)

[2.2.3 更新数据 8](#_Toc12110)

[2.2.4 衍生数据 8](#_Toc16980)

[2.3 系统功能需求 8](#_Toc1596)

[2.3.1 登陆和注册 8](#_Toc29195)

[2.3.2 电影标记 8](#_Toc15549)

[2.3.3 电影推荐 8](#_Toc21559)

[2.3.4 电影搜索 8](#_Toc326)

[三、 系统设计 9](#_Toc20752)

[3.1 系统功能设计 9](#_Toc2173)

[3.1.1 登陆注册 9](#_Toc10951)

[3.1.2 用户主页 9](#_Toc29544)

[3.1.3 电影标记 9](#_Toc12153)

[3.1.4 电影搜索 9](#_Toc3095)

[3.2.5 电影推荐 9](#_Toc3002)

[3.2 数据存储设计 9](#_Toc13452)

[3.3 数据处理方案 12](#_Toc19029)

[3.4 推荐算法设计 13](#_Toc493)

[3.4.1 非个性化推荐 13](#_Toc13113)

[3.4.2 个性化推荐 13](#_Toc3657)

[1、原理 14](#_Toc2734)

[2、具体实现 14](#_Toc7537)

[1、离线计算阶段 15](#_Toc31554)

[2、 在线推荐阶段 16](#_Toc21943)

[四、 系统展示 19](#_Toc5012)

[五、 总结 22](#_Toc4683)

[参考文献 24](#_Toc22401)

## 一、项目背景介绍

### 项目简介

本项目建立了一个电影评分网站，模仿了MovieLens（https://movielens.org）并使用Movie Lens数据集作为网站的初始数据进行电影展示和推荐。网站采用多种推荐算法，包括非个性化推荐和个性化推荐，为用户提供多种维度的电影推荐。网站拓展了个人主页模块，提供包括个人影评与电影评分在内的展示页面。

本项目前端基于主要基于 vue 和 element-ui实现。同时也使用了最新的前端技术栈，技术栈基于 ES2015+、vue、vuex、vue-router 、vue-cli 、axios 和 element-u。

本项目的后端主要基于python和fastapi，推荐算法共有5种：高分电影推荐、热门电影推荐、基于标签推荐、基于流派推荐、协同过滤推荐。用到的主要python库有，pyspark、pandas、numpy、gensim、pymongo等。

本项目的数据存储采用mongodb数据库。

项目环境基于docker搭建了可以连接mongodb的spark集群，后端未部署服务器，运行在组内电脑上通过花生壳暴露给另一台电脑上的前端访问。

本项目的github仓库地址 https://github.com/Junwen-Zhang/movie\_recommendation

### 国内外研究现状

随着互联网的普及和发展，网络上人们可以获取到极大量的信息。但由于信息量过大，用户反而难以从海量信息中准确获取想要的或感兴趣的信息，使得对信息的使用效率降低了，这就是信息超载（information overload）问题。

推荐系统是解决信息超载问题的一个有效途径。推荐系统的形式化定义如下[1-2]：设C是所有用户的集合，S 是所有可以推荐给用户的对象的集合。设效用函数u可以计算对象s 对用户c 的推荐度，即u: C × S→ R，R是一定范围内的全序的非负实数，推荐要研究的问题就是找到推荐度R最大的那些对象S\* ，即：



根据个性化程度的不同，推荐系统可以分为三类[3]: 非个性化推荐系统、半个性化推荐系统和完全个性化推荐系统。**非个性化推荐**对所有用户的推荐是无差别的，一般基于数据统计与分析；**半个性化推荐**基于用户的浏览行为对用户产生推荐，因为各个用户的浏览行为不同，推荐效果也不同；**完全个性化推荐系统**会保存用户的各种历史信息和行为，并从中分析和了解用户，进而向用户推荐用户可能喜好的内容。

**协同过滤算法**属于半个性化推荐，其主要特点有：不需要对推荐内容本身做太多了解便可以完成推荐工作；能够挖掘用户的潜在兴趣；需要依赖大量用户对项目的评分矩阵，存在新用户与新物品的冷启动问题。

**基于内容的算法**属于完全个性化推荐：用户之间保持独立性，推荐结果与他人的行为无关； 新项目可以立即得到推荐，新的项目加入到系统中时，只要拥有标签，便可以和用户的兴趣模型进行匹配完成推荐；但是新用户冷启动问题，由于新的用户没有历史行为记录，会存在冷启动问题，这也是为什么很多产品第一次登陆时都会让用户选择自己感兴趣的内容分类，就是为了快速对用户建立兴趣标签。

而将将基于内容的推荐算法与协同过滤算法相结合，便可以又推荐自己感兴趣的内容，又拓宽兴趣看到一些新奇的内容。（本项目分别完成了这两种算法）

在实际应用和大数据背景下，由于条目太多，常采用**召回加排序**的策略。召回阶段的主要职责是，从千万量级的候选物品里，采取简单的条件将推荐物品候选集合快速筛减到千级别甚至百级别，这样将候选集合数量降下来，之后在排序阶段往往用一些复杂模型（神经网络预测打分等），细致地对候选集进行个性化排序，如WIDE&DEEP，LR+GBDT等。

## 需求分析

### 2.1 数据处理需求

课程要求的ml-latest数据集比较大，它包含了280,000名用户对58,000部电影的27,000,000个评分和110万个标签数据，还包括标签基因组数据，包括1100个标签与的14,000,000个相关性得分。这个数据集带给我们的挑战有：

1. 用户、电影、标签数据条目多，需要我们采用比较快的数据处理工具，比如spark。
2. 评分和标签涉及多个个体的交叉，数据量非常大，直接使用效率很低，需要我们处理数据,

比如降维或提取关键信息从而提高推荐的效率。

1. 数据类型多，比如流派、年代、标签、评分等，我们需要分析各个数据的作用，尽可能全面

的用到这些数据，从而达到又符合用户口味、又拓宽兴趣范围的推荐结果。

### 2.2 数据存储需求

根据数据产生的时间、数据的用途以及数据的来源的不同，我们把数据分为静态数据、历史数据、更新数据和衍生数据。

2.2.1 静态数据

静态数据是数据库的基本数据，是系统数据的基础。在该推荐算法中，就是存储电影基本信息的movies数据表和记录电影和标签相关程度的genome-scores数据表，前者包含“movieId，title，genres”三个字段，用户在该系统中进行浏览的电影数据就来源于这张数据表；后者包含“movieId，tagId，relevance”三个字段，记录了电影和标签之间的相关程度（用小数表示）。

2.2.2 历史数据

历史数据，是构建推荐模型的数据来源，包括ratings（记录用户的评分记录），tags（记录用户给电影标记的标签）。

2.2.3 更新数据

更新数据的形式和历史数据一致，将新用户的产生的数据追加到ratings和tags两张表格中，用作推荐模型的输入。除此之外新增一个保留用户基本信息的user表，用于用户身份的验证。

2.2.4 衍生数据

衍生数据是以上的数据进行处理、形成的为满足某种便利的数据。包括movie\_rate（统计用户的评分记录，得出电影的均分，方便页面显示）、user\_movielist（记录了用户评分过的电影列表，加快表的读取速度）、trend\_rank（仅存储评分人数最多的前250部电影。设计该表格的原因是加快热门电影的推荐速度）。

### 2.3 系统功能需求

2.3.1 登陆和注册

第一次进入推荐系统，需要进行注册，注册成功后直接进入系统，开始使用系统内的功能。已经拥有账号的，直接登陆进入系统，开始使用系统功能。

2.3.2 电影标记

用户对观看过的电影可以进行评分、点评，用户主页会保留评分和点评记录，方便用户查看，用户的浏览记录是推荐系统核心的数据支撑。

2.3.3 电影推荐

推荐系统的核心就是根据用户的浏览记录（包括用户的评分等）进行个性化的推荐

2.3.4 电影搜索

支持电影的模糊搜索，方便用户进行电影的查找和挑选。

## 系统设计

### 3.1 系统功能设计

3.1.1 登陆注册

登陆注册实现用户身份的唯一性，以此存储和标识用户对于电影的观看和评分数据，以便对用户进行个性化推荐。

3.1.2 用户主页

用户可以查看个人的电影点评和评分记录.

3.1.3 电影标记

用户可以对观看过的电影进行评分、并留下点评。

3.1.4 电影搜索

按照电影的名字进行搜索，返回和该名字相关性较高的电影列表，以便用户选择和观看。

3.2.5 电影推荐

根据用户对电影的浏览和评分记录，给用户个性化地推荐用户可能感兴趣的电影，从五个维度进行推荐。“高分电影推荐”和“热门电影推荐”统计全部电影的数据，计算电影的平均得分和总评分人数，并据此进行推荐；“基于流派推荐”根据单个用户的观影打分记录预测用户喜好的电影流派，并据此推荐同一流派的高分电影；“猜你喜欢” 基于矩阵SVD算法，根据用户的观影打分记录训练模型，并根据模型预测用户可能喜欢的电影。

### 3.2 数据存储设计

存储信息的数据表如下：

| 表名 | **user** | | |
| --- | --- | --- | --- |
| 字段名 | \_id | username | password |
| 类型 | ObjectId | string | string |
| 说明 | mongdb自带的objectId类型的id | 用户名 | 密码 |

| 表名 | **movies** | | | |
| --- | --- | --- | --- | --- |
| 字段名 | \_id | movieId | title | genres |
| 类型 | ObjectId | int | string | string |
| 说明 | mongdb自带的objectId类型的id | 电影编号（从1开始） | 电影标题 | 电影类型（戏剧、动作......） |

|  |  |  |  |
| --- | --- | --- | --- |
| 表名 | **genome-scores** | | |
| 字段名 | movieId | tagId | relevance |
| 类型 | int | int | float |
| 说明 | 电影编号 | 标签编号 | 电影和标签的相关程度 |

|  |  |  |
| --- | --- | --- |
| 表名 | **genome-tags** | |
| 字段名 | tagId | tag |
| 类型 | int | string |
| 说明 | 标签编号 | 标签 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | **ratings** | | | | |
| 字段名 | \_id | userId | movieId | ratings | timestamp | |
| 类型 | ObjectId | string | string | int | string | |
| 说明 | mongdb自带的objectId类型的id |  |  | 评分（0~5） | 评分时间与1970年1月1日间隔的秒数 | |

| 表名 | **user\_movielist** | | |
| --- | --- | --- | --- |
| 字段名 | \_id | userId | rated\_movie\_list |
| 类型 | ObjectId | int | Array |
| 说明 | mongdb自带的objectId类型的id | 用户序号 | 用户已评分过的电影ID列表 |

设计user\_movielist的主要目的是加快对用户已标记的电影的查询，如果沿用原始数据表`ratings.csv`，查询某个用户标记过的电影需要19s；而设计该表优化后只需要1s以内即可完成。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | **comments** | | | | |
| 字段名 | \_id | userId | movieId | comment | timestamp | |
| 类型 | ObjectId | string | string | float | string | |
| 说明 | mongdb自带的objectId类型的id |  |  | 点评内容 | 点评时间与1970年1月1日间隔的秒数 | |

| 表名 | **top250** | | | |
| --- | --- | --- | --- | --- |
| 字段名 | \_id | movieId | avg\_rating | rating\_cnt |
| 类型 | ObjectId | int | float | int |
| 说明 | mongdb自带的objectId类型的id | 电影ID | 电影的平均分 | 电影的评分人数 |

| 表名 | **movie\_rate** | | | |
| --- | --- | --- | --- | --- |
| 字段名 | \_id | movieId | cnt\_rating | avg\_rating |
| 类型 | ObjectId | int | int | double |
| 说明 | mongdb自带的objectId类型的id | 电影编号 | 电影的评分次数 | 电影的平均评分 |

| 表名 | **trend\_rank** | | |
| --- | --- | --- | --- |
| 字段名 | \_id | movieID | count |
| 类型 | ObjectId | int | int |
| 说明 | mongdb自带的objectId类型的id | 电影编号 | 电影的评分次数 |

trend\_rank表格仅存储评分人数最多的前250部电影。设计该表格的原因是加快热门电影的推荐速度，将热门电影推荐从每次对整个评分表格的查询和统计简化为定期维护和更新这一表格，需要推荐时从这一表格中抽取热门电影，减少了每次推荐的时间。

| 表名 | **movie\_genre\_rank** | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 字段名 | \_id | movieID | title | genre | cnt\_rating | avg\_rating |
| 类型 | ObjectId | int | string | string | int | double |
| 说明 | mongdb自带的objectId类型的id | 电影ID | 电影名 | 电影类别 | 电影的评分次数 | 电影的平均分 |

movie\_genre\_rank表格存储每个分类中平均分最高的50部电影的信息。设计该表格的原因是加快按类型推荐时的推荐速度，避免遍历整个电影类别表格，以此减少每次推荐的用时。

### 3.3 数据处理方案

课程要求的ml-latest数据集比较大，它包含了280,000名用户对58,000部电影的27,000,000个评分和110万个标签数据，还包括标签基因组数据，包括1100个标签与的14,000,000个相关性得分。为了使用户推荐时间在满意的范围内，我们主要采用了以下方法：

1. 离线训练模型并保存结果。训练往往基于全部的数据，因此我们只需在一段时间后数据变化

较多时更新模型即可。在线推荐时，我们只需要加载本地模型，并在前处理数据上进行比较快速的推荐。

1. 数据前处理并保存在数据库中，使得查询所需数据更方便快捷。
2. 在线推荐和离线训练，我们大量用到了基于内存处理的pyspark和pandas等用于大数据处

理的python库。

1. 利用自然语言处理gensim库中的Doc2Vec模型，可以将文本转成语义向量，相当于对电影

的标签特征进行了降维。

我们共有5个推荐算法的接口，其中：

“**高分电影推荐**”“**热门电影推荐**”只用到了方法（2）；

“**基于流派推荐**”用到了方法（2）和方法（3）spark；

“**基于标签推荐**”用到了方法（1）（2）和（4），gensim库生成了电影相似度最高100个标签的20维语义向量，起到降维效果；

“**猜你喜欢**”的后端使用的是基于矩阵的协同过滤算法，主要用到了方法（1）（2）和方法（3）spark基于内存的数据处理。

### 3.4 推荐算法设计

**3.4.1 非个性化推荐**

（1）高分推荐

推荐平均评分最高的前x部电影。

具体实现方式：使用`ratings.csv`中的数据，统计每部电影的平均得分，然后基于平均分降序排序，这时发现有很多5.0分的电影，我们推测这是由于这部分电影的评分人数很少，因此我们加入一个筛选条件“评分人数大于50”，然后再取最高的250部电影存入数据库集合`top250`中。每次请求高分推荐的电影时，我们随机返回10部电影。

因此该算法的计算复杂度为O(n\*logn)，而推荐复杂度为O(1)；空间复杂度为O(1)；实际运行时间非常短。

（2）热门推荐

推荐被评分次数最高的前x部电影。

具体实现方式：使用rating.csv中的数据，统计每个电影所获得的评分次数，并按评分次数进行降序排序，取前y名（选择y=250）保存在数据库trend\_rank表格中。每次进行推荐时，随机抽取x（选择x=10）部电影作为推荐列表。

进行排序时，对于含有n部电影的评分表，进行排序以寻找评分次数最高的电影的时间复杂度为O(n log n)。进行推荐时，复杂度为O(1)。

**3.4.2 个性化推荐**

（1）基于矩阵的协同推荐算法

1、原理

我们使用了基于矩阵奇异值分解（SVD）的方法来进行协同推荐。原始数据ratings.csv中，数据格式为userID，MovieID，rating，可以生成一个*M* × *N*的评分矩阵A，每行代表一个user （即UserID），每列代表一个item（即MovieID），其中的元素表示user对item的打分Rating。

评分矩阵A是一个规模很大的稀疏矩阵，其中缺失的评分居多。对于这样规模的大矩阵，传统的矩阵计算方式很难处理。因此，可以根据奇异值分解的原理对评分矩阵A进行降维，将*M* × *N*的大稀疏矩阵转化为*M × K* 和*K × N*的两个稠密矩阵的乘积。

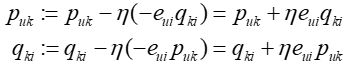
基于评分矩阵A，SVD的计算公式如下所示：



进而可以近似表示为 ，其中*M*表示user数量，*N*表示item数量，*K*表示隐式语义维度。

从向量的角度，可以得到Rui = Pu Qi，即id为u的用户对id为i的物品的预测评分为矩阵P的第u行与矩阵Q的第i行的点乘。

进行训练时，需要让总误差平方和减小，从而让预测值尽可能接近真实值。按照梯度下降法，沿负梯度方向更新用户向量和物品向量，更新公式为：



采用这种方法，复杂度可以从O(mn) 简化为O(k\*(m+n))

2、具体实现

由于原始数据集的大小，不能直接将rating.csv完整读取并转化为矩阵。在训练时，我们采用每次从数据库中读取一位用户的评分信息然后遍历电影列表的方法进行训练。

算法的具体实现为：对于userID为u的用户，从数据库user\_movielist表中读取该用户的评分信息，记录其对电影的评分列表；从数据库user\_matrix中尝试读取 Pu，若数据表中无记录，则初始化为一个1×K的向量，每一位都在(0,1)范围内。遍历物品列表，依次读取物品向量（若无记录则使用随机数进行初始化），计算用户向量和物品向量的点乘得到预测的用户u对物品i的评分并保存。之后，对用户的评分信息表进行遍历，将用户的实际评分和预测评分相减计算出误差，并根据更新公式进行梯度下降，更新用户向量和物品向量。

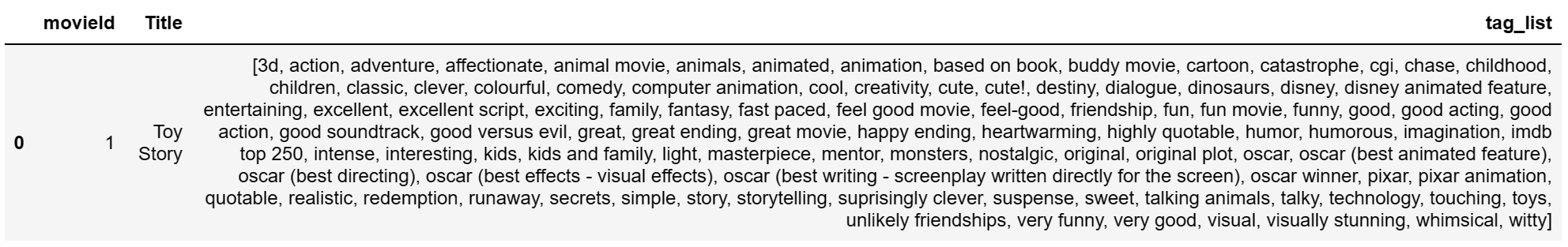
进行推荐时，使用用户的向量和每一部电影的物品向量做点乘，得到预测的用户对所有电影的评分；随后对评分进行排序，返回预测的评分最高的10部电影的id。

（2）基于电影标签（tag）推荐

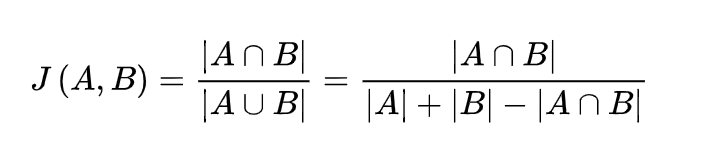
1、离线计算阶段

基于ratings.csv生成每个用户评分过的电影列表，该数据表user\_movie\_list包含两个字段“userId”“rate\_movie\_list”。（这个数据表需要保存，用于在线推荐阶段）

然后开始我们模型训练的数据准备，首先基于genome-scores.csv生成每部电影相关性最强的100个标签，该数据表movie\_tags\_list包含三个字段“movieId”“Title”“tag\_list”。

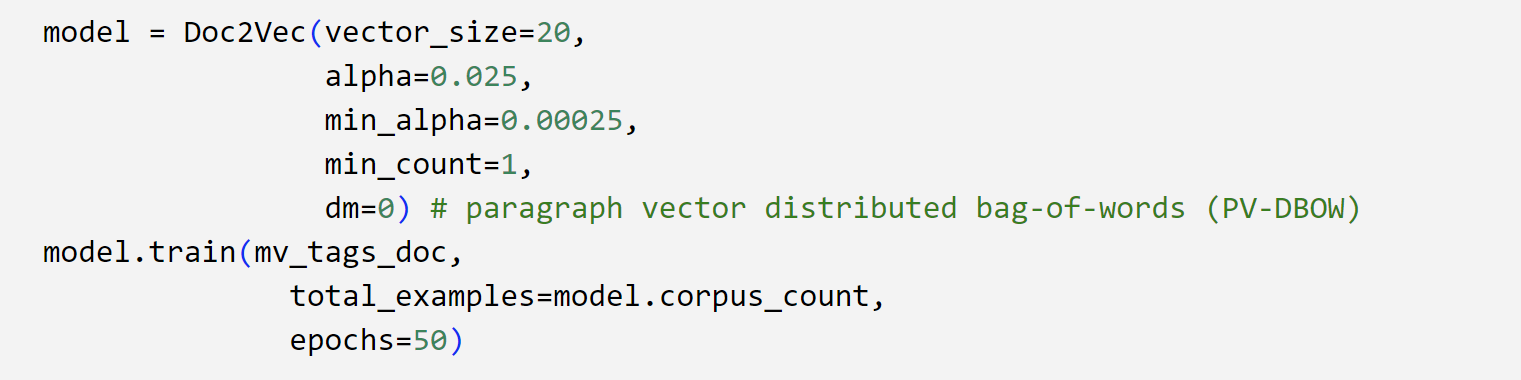


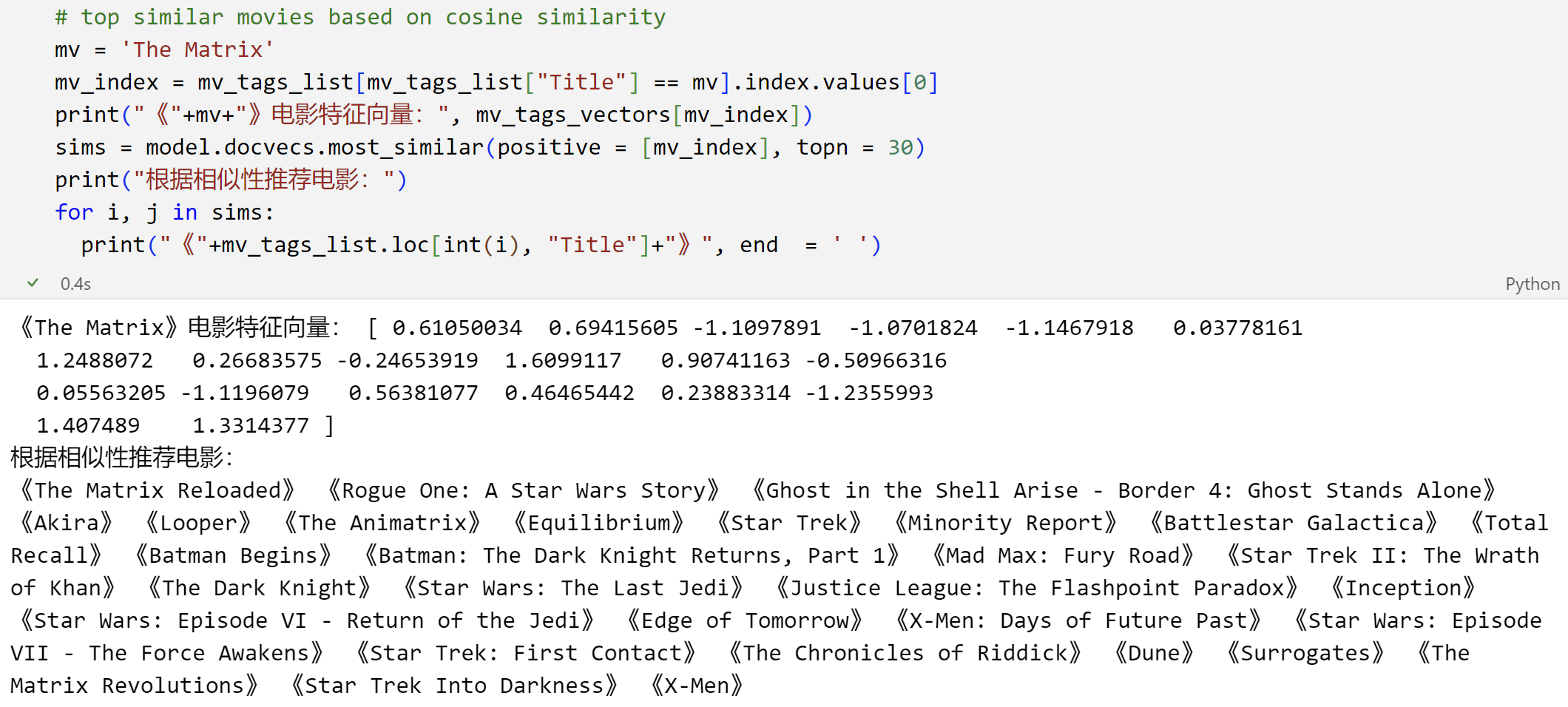
最初我们是根据“tag\_list”字段生成各个movie之间的jaccard相似度矩阵，形成新的数据表movie\_tags\_list\_sim。其中Jaccard相似系数用于比较有限样本集之间的相似性与差异性。Jaccard系数值越大，样本相似度越高。其公式为



但可以看到这是一个简单的标签推荐模型，这种相似度计算过于粗糙，适用于两两之间的相似性度量，且**只适用于**寻找**单部电影**的相似电影推荐，在实际模型中我们采用了其他更实用的相似性评价方式——语义向量，可以评价任意电影之间的相似性。

我们使用了**gensim库中的Doc2Vec模型**（这是一个google公司推出的**自然语言处理无监督模型**，第一轮操作会统计词频来构建内部的词典数结构，第二轮操作会进行神经网络训练，不同于Word2Vec模型，在快速训练的基础上，该模型输出的向量包含语义信息），即将之间处理好的前100个相似度最高的标签文本转换成一个20维向量，用于衡量电影的特征。我们设置的训练参数：向量维数=20、epoch=50，总共的训练时间为**13m37s**。



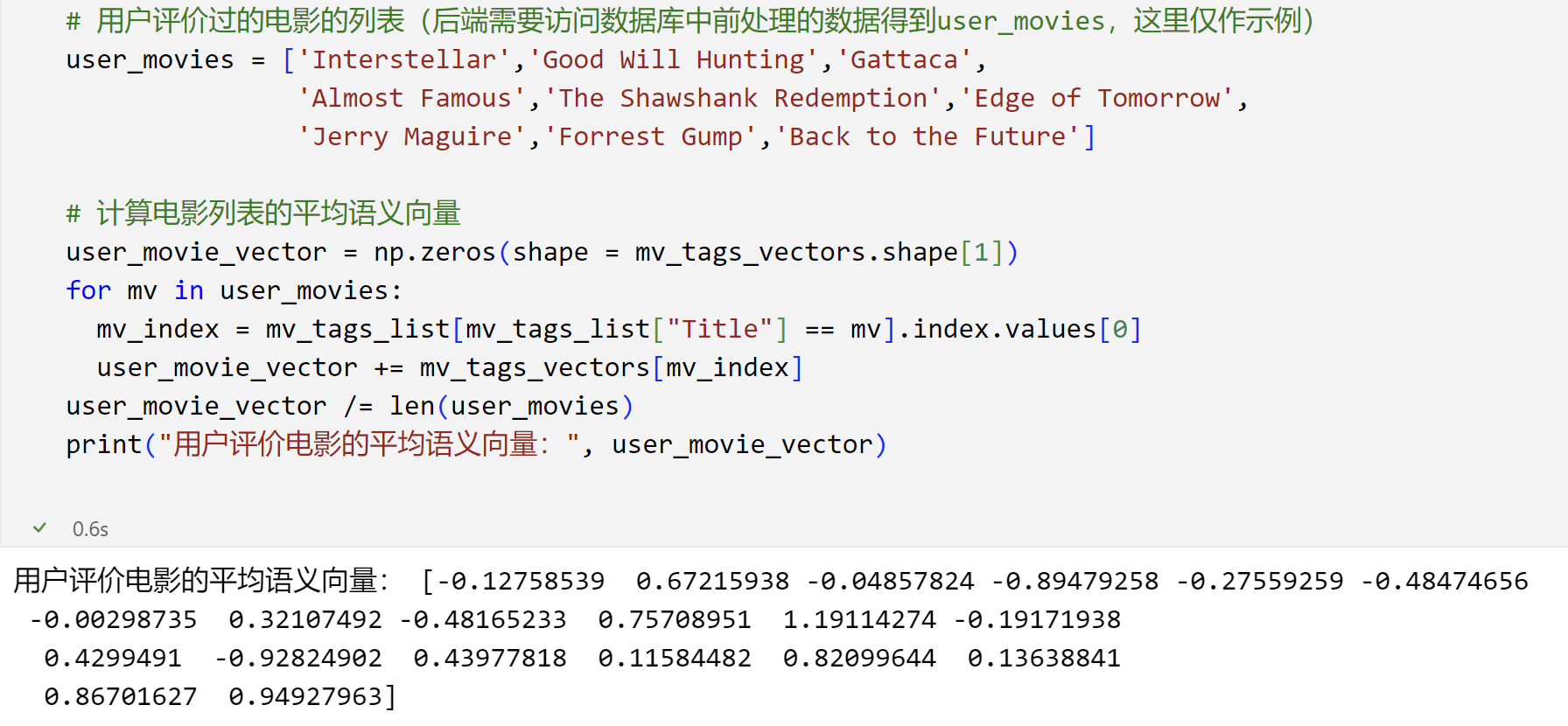
训练好后在**本地保存模型**，大小为**3.93M**。我们使用非常有名的一部电影《黑客帝国》（The Matrix）进行测试，发现推荐效果非常好，这些电影都和《黑客帝国》很相似。说明这个训练出来的电影语义向量可以比较好的表达电影特征，可以在具体推荐算法中使用。

1. 在线推荐阶段

假设电影库中有n部电影，用户标记电影共m部，需要向用户推荐k部电影。（n>>m, n>>k）

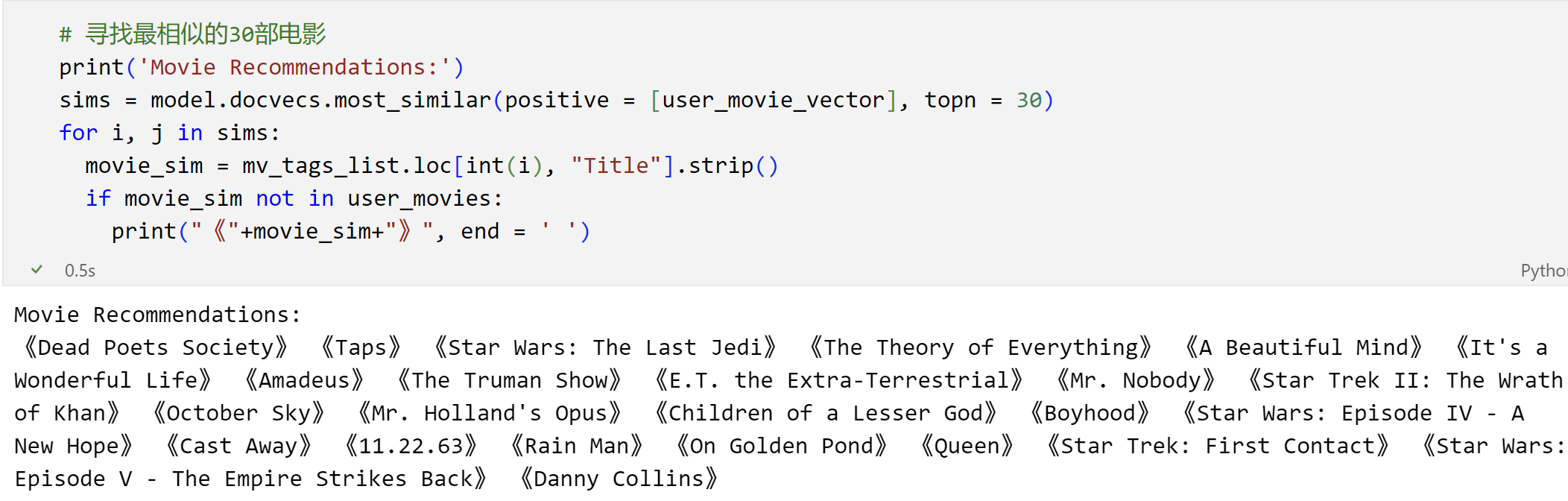
**A、**计算平均语义向量：**复杂度O(m)≈O(1)**

用户访问api获取电影推荐时，我们需要用到离线阶段保存在本地的模型和数据。首先根据用户的userID基于离线阶段保存的数据表user\_movie\_list非常快速的获取所评价过的movie\_list。然后我们只需要累加这些电影的特征向量取平均值，就可以衡量该用户的电影兴趣。



**B、**找出相似性最大的k部电影：**复杂度O(kn)≈O(n)**

我们doc2vec模型中most\_similiar函数，即可快速的获得最相似的电影推荐。



实际在后端运行时，还需要加上本地模型加载的时间，不过模型很小，总共耗时0.6s。算法效果很好，前端访问不流畅主要是docker卡顿还有网络的问题。

（3）基于电影流派（genre）推荐

根据用户的历史评分记录计算用户喜欢的流派，按照流派推荐用户可能喜欢的电影。

**def** cal\_genre():

*# 加载数据*

spark = SparkSession.builder.appName('getTrend').getOrCreate()

complete\_ratings\_file = os.path.join('datasets', 'ml-latest', 'ratings.csv')

rating = spark.read.csv(complete\_ratings\_file, sep=',', inferSchema=**True**, header=**True**)

complete\_movies\_file = os.path.join('datasets', 'ml-latest', 'movies.csv')

movies = spark.read.csv(complete\_movies\_file, sep=',', inferSchema=**True**, header=**True**)

*# 数据处理*

movies = movies.where(movies.genres != "(no genres listed)") *# 去掉空值*

movies.summary("count").show()

*# 按分类统计*

mlist = []

rows = movies.collect()

**for** row **in** rows:

mid = row[0]

t = row[1]

gs = row[2].split("|")

**for** g **in** gs:

l = []

l.append(mid)

l.append(t)

l.append(g)

mlist.append(l)

df = pd.DataFrame (mlist, columns = ["movieID", "title", "genre"])

movie\_genre = spark.createDataFrame(df)

*# 计算评分*

rating = rating.groupby("movieId").agg(count("\*").alias("cnt\_rating"),avg("rating").alias("avg\_rating"))

*# 连接类型和评分*

df\_join = movie\_genre.join(rating, movie\_genre.movieID==rating.movieId, "inner")

df\_join = df\_join.drop("movieId")

*# 去掉评分人数过少的电影*

df\_join = df\_join.where(df\_join.cnt\_rating>=200)

df\_join.show(10)

*# 按类别统计*

movie\_list = []

genre\_list = ["Action","Adventure","Animation","Children's","Comedy",\

"Crime","Documentary","Drama","Fantasy","Film-Noir","Horror","Musical",\

"Mystery","Romance","Sci-Fi","Thriller","War","Western"]

**for** g **in** genre\_list:

l = df\_join.where(df\_join.genre == g).sort(['avg\_rating'], ascending=**False**).limit(50).collect()

movie\_list.extend(l)

movie\_genre = spark.createDataFrame(movie\_list)

movies.csv统计了电影的流派分类信息。数据一共有十八种类别，每部电影可能属于零或多个类别。依照数据，统计每个类别下最高分的n部电影，保存在数据库中。由于部分电影的评分数据很少，导致其平均分为5.0，因此排序时去除掉评分数少于250 的电影。保存每个分类下排名前n（n取50）的电影存入数据库movie\_genre\_rank表中。

每次进行推荐时，依据该用户的观影评分记录计算用户最喜欢的类别。遍历用户的观影评分列表，对于每部电影，对它所属于的类别加上用户对这部电影的打分，最后得到累计得分最高的类别即视为用户最喜爱的类别。从这一类别中，随机抽取x（选择x=10）部电影作为基于分类的推荐列表。

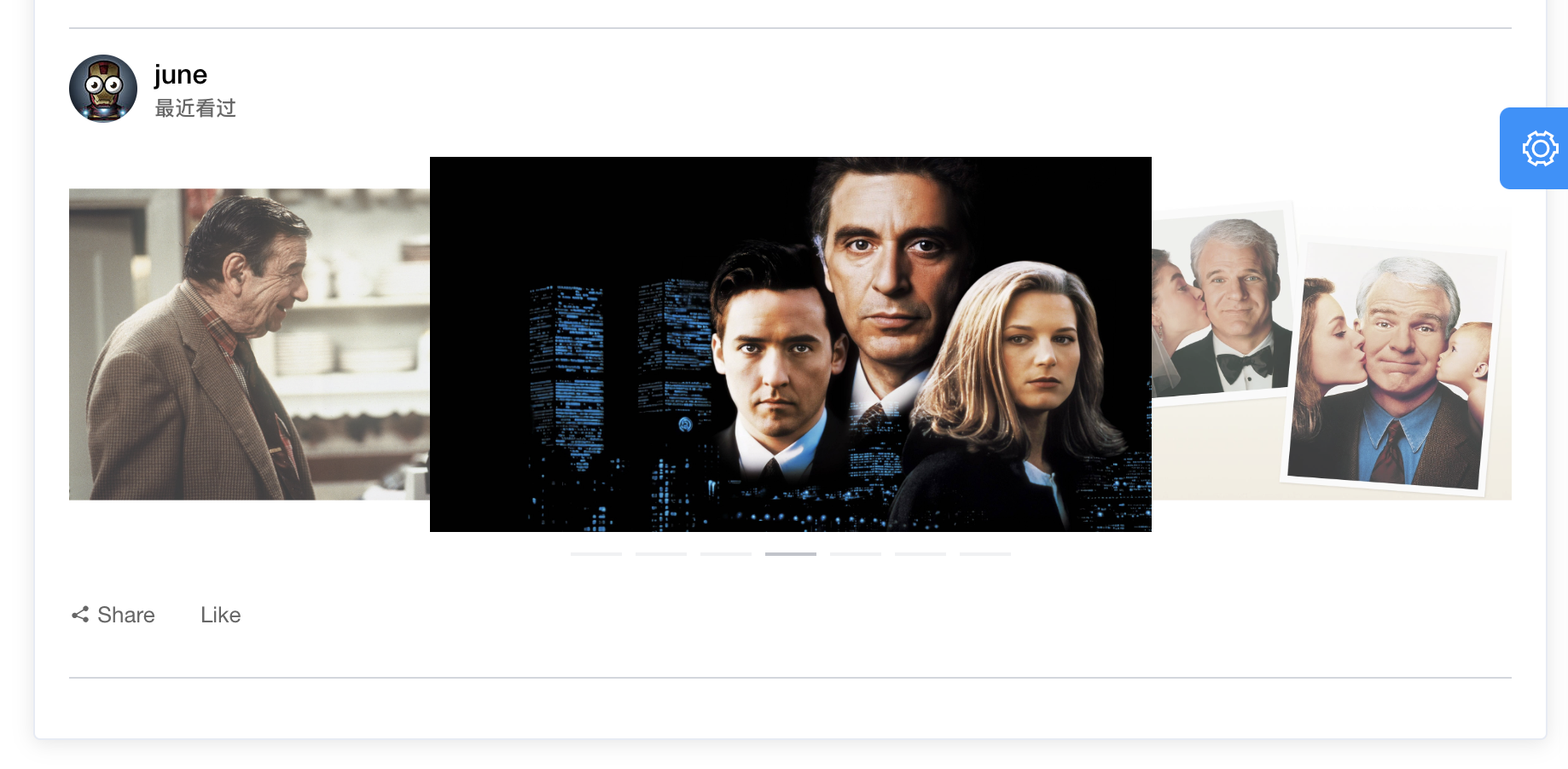
进行电影分类排序时，对于含有n部电影的电影类别表，统计每个分类下的评分最高的电影的时间复杂度为O(n log n)。进行推荐时，对于含有n部电影评分的用户评分表，推荐的时间复杂度为O(n)。

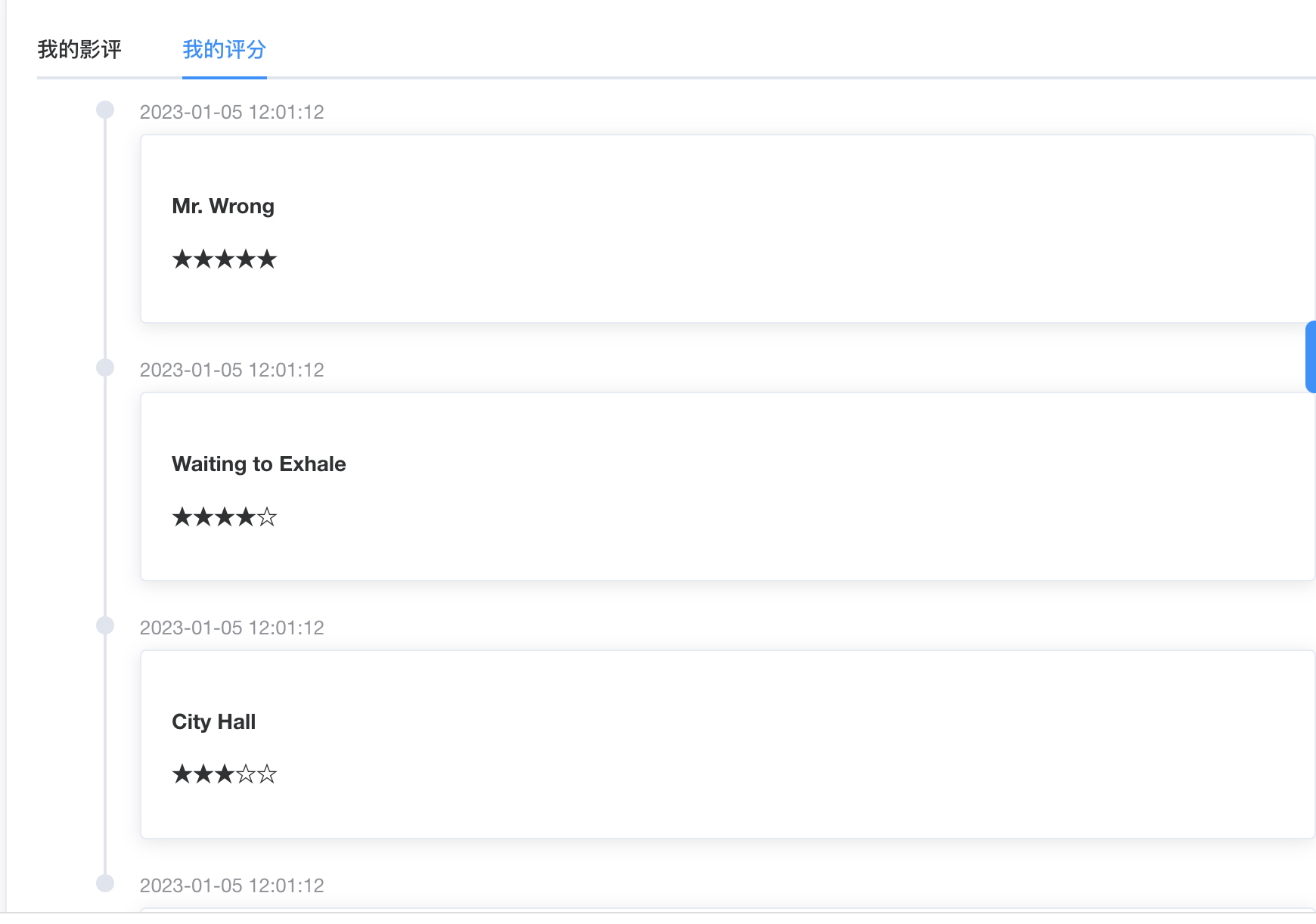
## 系统展示

**登录页面**：登录页面需要输入匹配的用户名和密码，匹配错误时会弹窗提醒。也可以新注册账号，但是注册账号的用户名不能与数据库中存在的用户名重复。

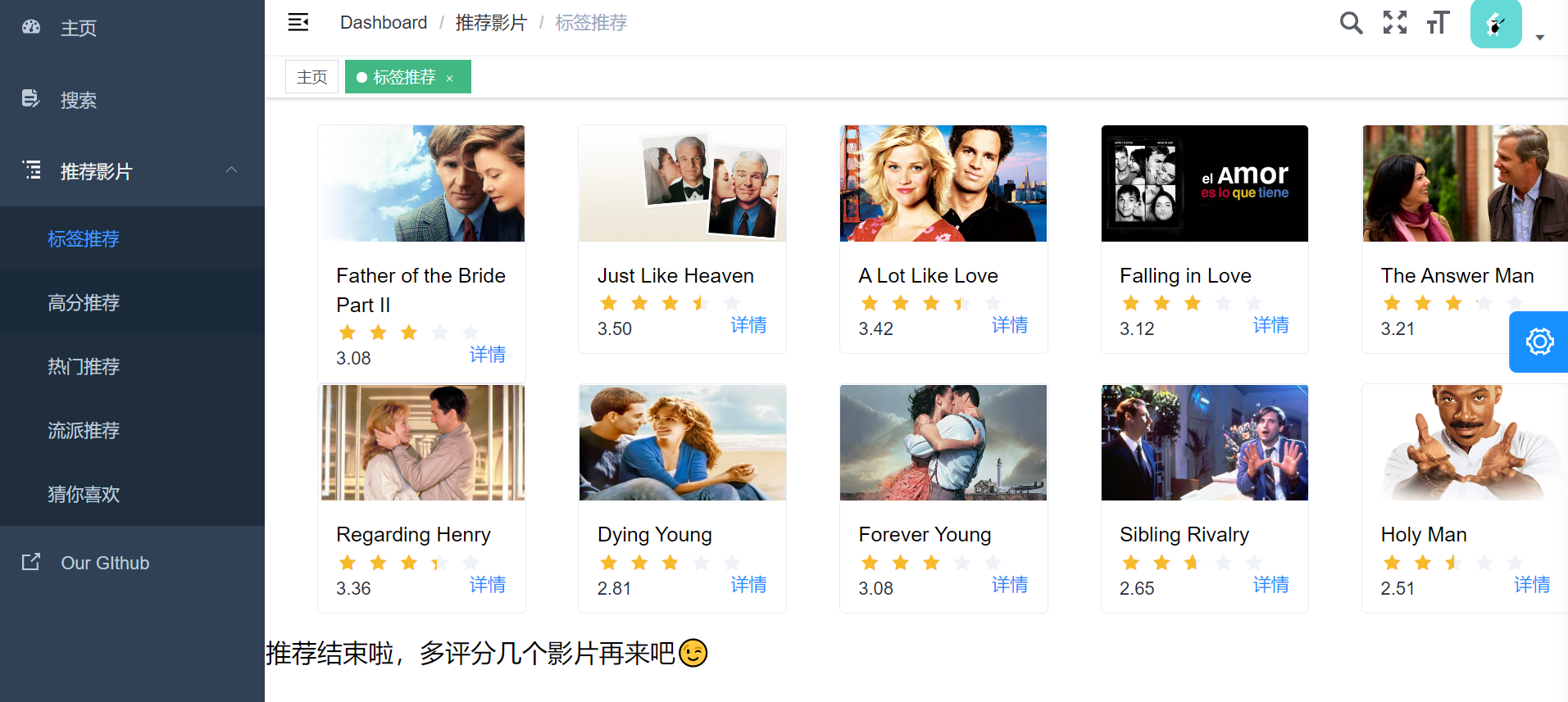
**主页面**：主页面由导航栏，侧边栏和标签页组成，当前页面路径会显示在导航栏中，可以通过点击进行页面的跳转。侧边栏包括搜索页面，五种电影推荐页面和跳转到github的链接。

**导航栏**：导航栏右侧是用户头像，可以通过下拉框进入用户主页、回到主页或者跳转到本项目的github网址。

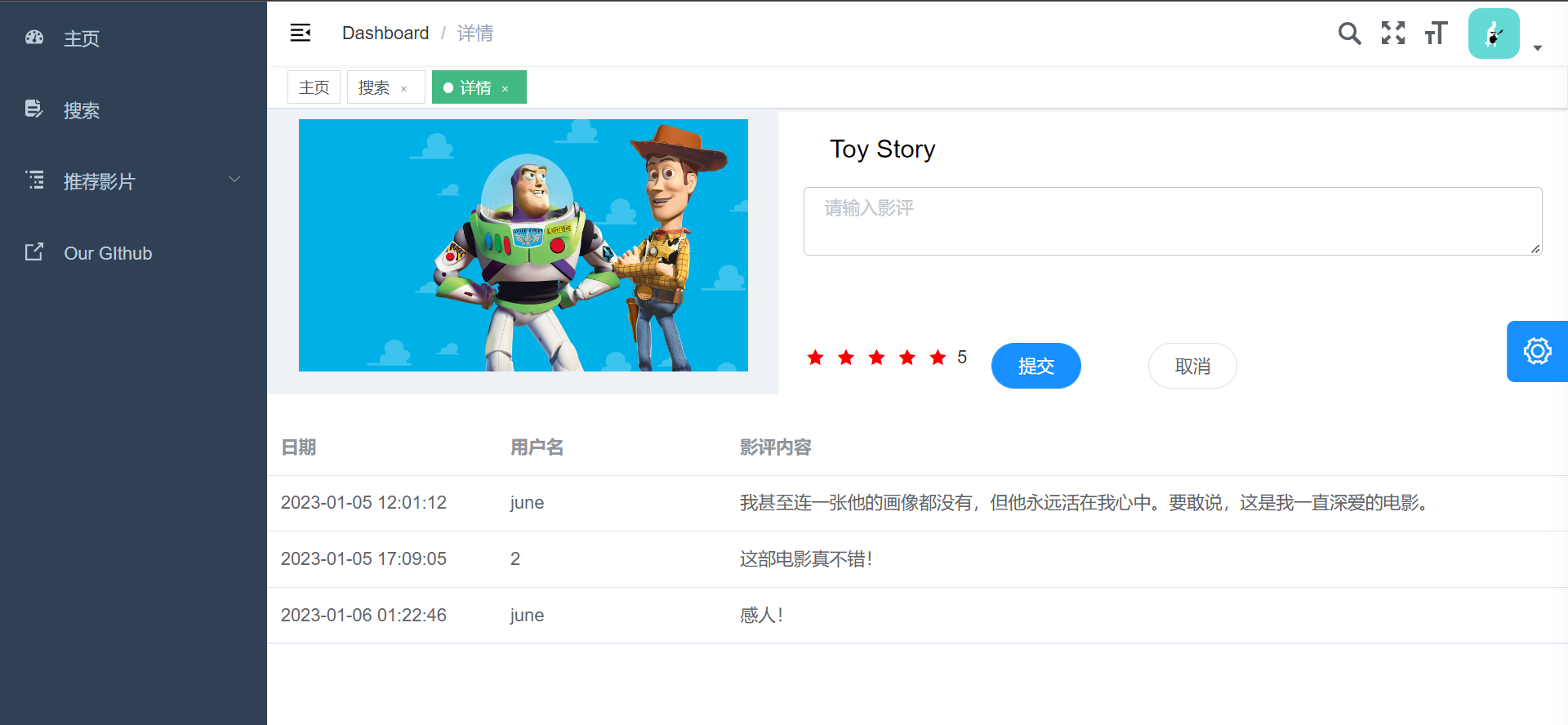
**用户主页**：用户主页左侧是用户的基本信息，右侧会展示当前用户的影评记录，在底部用走马灯显示用户看过的电影图片，另一个标签页用时间线展示的当前用户的评分记录。



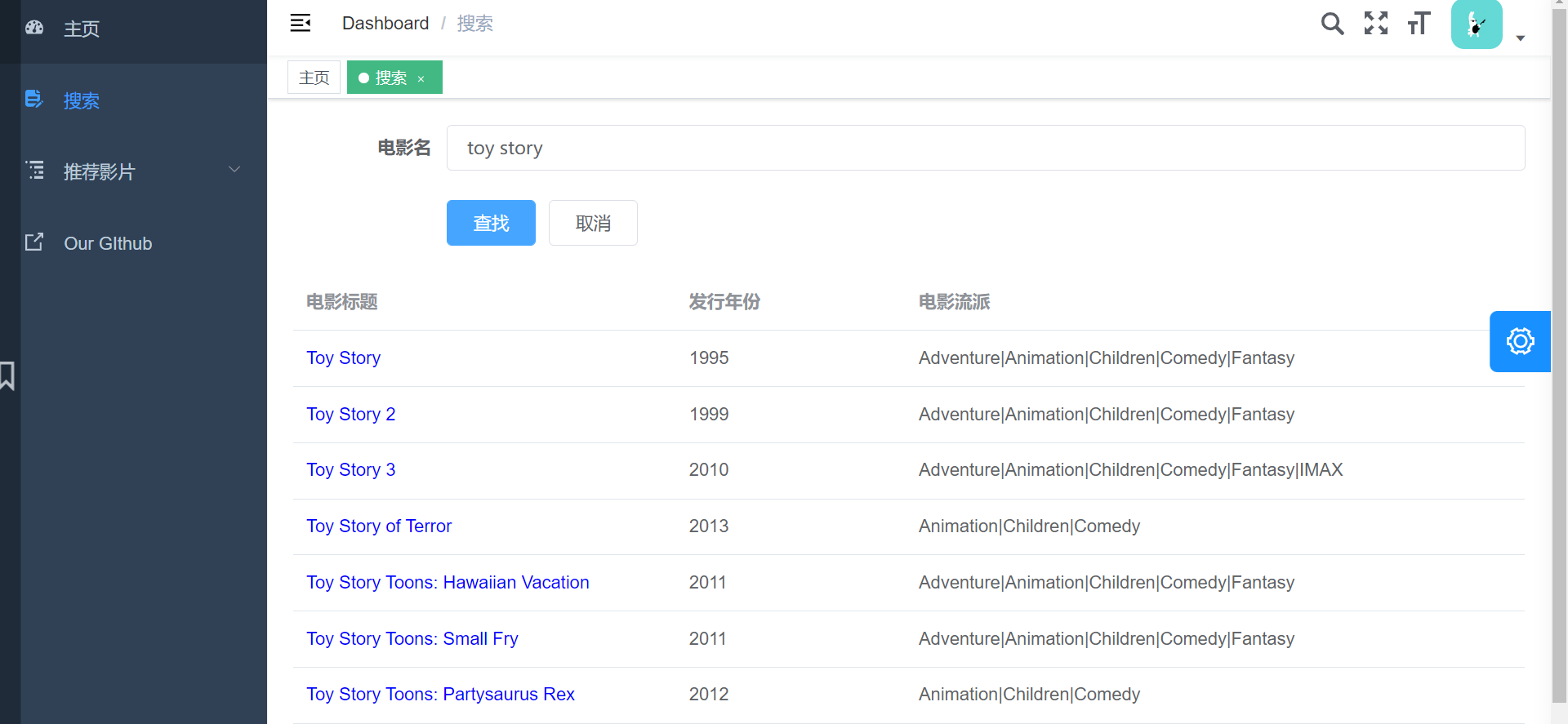
**推荐页面**：推荐页面使用无限滚动列表，拖动到列表底部时会自动加载十部推荐电影，卡片上显示当前电影的标题和数据库中的平均评分。



**详情页面**：可以通过点击卡片中的详情跳转到当前电影的详情页面，用户可以在详情页面给当前电影评分，添加影评。当前的电影的所有影评会显示在详情页底部，添加的影评可以实时更新。



**搜索页面**：搜索页面包括搜索框和搜索结果列表，可以通过电影名对电影进行模糊搜索，相关性最高的搜索结果会显示在下方列表中，可以通过点击标为蓝色的电影名一栏跳转到搜索结果的详情页面。



## 总结

|  |  |  |
| --- | --- | --- |
| 兰庆秋 | 贡献 | 后端除推荐算法之外的接口 |
| 总结 | 好的工具可以使项目开发事半功倍。前后端分离的情况下，使用fastapi可以高效地进行后端的调试、便于前后端的数据传输的规范；navicat工具对数据库的查看方便后端对数据库的链接和测试。 |

|  |  |  |
| --- | --- | --- |
| 秦霄潇 | 贡献 | 前端个人主页开发，电影海报爬虫 |
| 总结 | vue作为一套构建用户界面的渐进式框架，它采用自底向上增量开发的设计。它的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。同时，前后端分离的情况，利用fastapi可以与后端高效地协同沟通与测试。 |

|  |  |  |
| --- | --- | --- |
| 李思颀 | 贡献 | 后端推荐算法（热门推荐，流派推荐，协同推荐） |
| 总结 | 对大数据进行处理时，Spark将数据集合抽象成一个RDD对象，然后对RDD进行各种计算处理。RDD之间通过依赖关系形成类似流水线的结构，因此有高容错性和速度快的优点。  矩阵分解的方法可以将一个大的稀疏矩阵近似地用两个低维的稠密矩阵表示，同时对稀疏矩阵中的空缺值进行预测。这样处理后，可以减少推荐的时间和空间需求，并且有较好的推荐效果。 |

|  |  |  |
| --- | --- | --- |
| 章俊文 | 贡献 | 后端推荐算法（高分推荐，标签推荐），后端与数据库整合 |
| 总结 | 在负责基于标签数据推荐的过程中，发现除协同过滤外，论文中用到比较多的还有关于知识图谱、词嵌入之类的在NLP和ML领域的新技术，就想尝试一下。而我们的标签数据又有1128个，很多标签语义相近、重复但属于不同的标签，必须处理一下，联系《自然语言处理》课程，感觉可以用到这方面的知识。这时我发现了Doc2Vec这个模型，恰好满足我们的需求（处理无时序的文本语义），原来电影与标签的相似性矩阵有58098\*1128那么大，且无法反映电影间的相似性。经过提取每部相似性前100的标签训练生成20维的语义向量后，搜索空间降到了58098的大小，电影间的相似性可以直接通过余弦距离来衡量，用户对电影的口味也可以通过向量各维度平均数得到。 |

|  |  |  |
| --- | --- | --- |
| 周飞燕 | 贡献 | 前端推荐页面，搜索页面和详情页面开发 |
| 总结 | 使用elementUI能够方便地构建美观的页面，例如使用breadcrumb组件可以清晰地展示页面的路径。但是在构建页面的时候要注意接口的性能，在展示推荐电影页面时由于爬取电影图片接口耗时较长，导致加载过多推荐电影会导致后端负载过重。因此需要将原本设计的无限滚动列表修改为只显示十个电影。 |

## 参考文献

[1] Adomavicius G，Tuzhilin A.Toward the next generation of recommender systems：A survey of the state-of-the-art and possible extensions[J].IEEE Trans on Knowledge and Data Engingeering，2005，17（6）：734-749.

[2] Sung H H.Helping online customers decide through Web personalization[J].IEEE Intelligent Systems，2002，9：34-43.

[3]杜定宇. 基于特征向量的个性化推荐算法研究[D].重庆大学,2011.

[4]王国霞,刘贺平.个性化推荐系统综述[J].计算机工程与应用,2012,48(07):66-76.  
[5]刘青文. 基于协同过滤的推荐算法研究[D].中国科学技术大学,2013.  
[6]杨志伟. 基于Spark平台推荐系统研究[D].中国科学技术大学,2015.  
[7]李改,李磊.基于矩阵分解的协同过滤算法[J].计算机工程与应用,2011,47(30):4-7.  
[8]黄创光,印鉴,汪静,刘玉葆,王甲海.不确定近邻的协同过滤推荐算法[J].计算机学报,2010,33(08):1369-1377.  
[9]孙海峰,甘明鑫,刘鑫,吴越.国外电影推荐系统网站研究与评述[J].计算机应用,2013,33(S2):119-124.  
[10]肖旋. 基于多模态情感融合的个性化推荐算法研究[D].上海师范大学,2021.DOI:10.27312/d.cnki.gshsu.2021.002432.  
[11]李光明,房靖力.Spark平台下电影推荐系统的设计[J].计算机应用与软件,2020,37(11):28-34.

[12]Ali, Syed M., Gopal K. Nayak, Rakesh K. Lenka和Rabindra K. Barik. 《Movie Recommendation System Using Genome Tags and Content-Based Filtering》. Advances in Data and Information Science

[13]Dutta, Sambo, Soumita Das, Joydeep Das, Subhashis Majumder. 《Tag-Cloud Based Recommendation for Movies》.

[14]Lee, Yen-Hsien, Tsai-Hsin Chu. 《An Incremental Clustering Approach to Personalized Tag Recommendations》

[15]Li, Jung-Bin, Szu-Yin Lin, Yu-Hsiang Hsu, Ying-Chu Huang. 《Implementation of an Alternating Least Square Model Based Collaborative Filtering Movie Recommendation System on Hadoop and Spark Platforms》

[16]Luo, Ling, Haoran Xie, Yanghui Rao, Fu Lee Wang. 《Personalized Recommendation by Matrix Co-Factorization with Tags and Time Information》

[17]Putri, Debby Cintia Ganesha, Jenq-Shiou Leu. 《Towards the Implementation of Movie Recommender System by Using Unsupervised Machine Learning Schemes》.

[18]Shin, Seungkyu, Minsang Namgoong, Juyong Park. 《Analysis of Evolution of Movies Using Massive Movie-Tag Meme Network Data》.

[19]Wang, Beilun, Haoqing Xu, Chunshu Li, Yuchen Li, Meng Wang. 《TKGAT: Graph Attention Network for Knowledge-Enhanced Tag-Aware Recommendation System》.

[20]Wei, Shouxian, Xiaolin Zheng, Deren Chen, Chaochao Chen. 《A Hybrid Approach for Movie Recommendation via Tags and Ratings》.

1. 合计成绩=项目得分+组内评分 [↑](#footnote-ref-0)