27 组项目大作业实验报告

肖嘉源　李欣泽　欧阳妍妍

一、前期准备

（一）　项目目标

•完成一群用户和许多图书的储存管理，实现相关特征（评论数、得分、特征、用户画像的构建），并实现信息的更改与增减

•实现图书特征的抽象，用户画像的构建，在此基础上，实现兴趣图书、兴趣用户的推荐。

（二）　项目目标

功能类型	功能描述	内容
基本功能	图书 / 电影信息的储存、管理	1.　用 Google LevelDB 实现底层数据集的存储。 2.　系统中的图书有评论和评级（假设使用 1-10 级的评分标准），有总的评论数和平均得分。 3.　可以添加新的图书，也可以删除或修改已有的图书，实现更改。 4.　书抽象出相关特征，以供推荐算法的识别。
	用户信息的储存管理	1.　用户可以对系统中的图书进行评论和评级 2.　可以注册新的用户，删除或修改已有的用户的信息，实现更改。 3.　根据用户行为抽象出相关特征，以供推荐算法的识别。
推荐功能	登陆系统推荐	1.　对新用户做简单的测试，刻画用户画像。 2.　实现用户画像和书籍特征的匹配。 3.　输出用户最有可能阅读但还未阅读过的 10 本图书。
	可能好友推荐	1.　通过用户行为（评分）进一步刻画用户画像。 2.　实现用户画像之间的匹配 3.　输出用户兴趣相投的好友

二、系统设计

（一） 推荐算法设计

1）数据介绍

1. 数据来源：MovieLens：http://grouplens.org/datasets/movielens/

2. 数据描述：数据集（ml-latest-small）于 2016 年 10 月 17 日生成，描述了电影推荐服务 MovieLens 的 5 星评级和自由文本标记活动。它在 9125 部电影中包含 100004 个评分和 1296 个标记。这些数据来自于 1995 年 1 月 9 日至 2016 年 10 月 16 日期间随机选择的 671 个用户，所有选定的用户对至少 20 部电影进行了评分。

3. 数据内容： links.csv，movies.csv，ratings.csv 和 tags.csv

links.csv：<movieId><imdbId><tmdbId>，电影对应电影网站的编号

movies.csv：<movieId><title><genres>，电影相关信息

ratings.csv：<userId><movieId><rating><timestamp>，评分记录

tags.csv：<userId><movieId><tag><timestamp>，标签记录

2）推荐算法

1.基于用户的协同过滤

计算用户 u,v 相似度,采用 Pearson 相关系数：

$$PC(\mathrm{u},\mathrm{v}) = \frac{\sum_{i\in I_{uv}}(r_{ui}-\overline{r_u})(r_{vi}-\overline{r_v})}{\sqrt{\sum_{i\in I_{uv}}(r_{ui}-\overline{r_u})^2\sum_{i\in I_{uv}}(r_{vi}-\overline{r_v})^2}}$$

由此，选取最相关的 N 个评价过电影 i 的用户构成$N_i(u)$,用户评分预测为：

$$\widehat{r_{ui}} = \overline{r_u} + \frac{\sum_{j\in N_i(u)}\omega_{uv}(r_{vi}-\overline{r_v})}{\sum_{j\in N_i(u)}|\omega_{uv}|}$$

其中$\omega_{uv} = PC(\mathrm{u},\mathrm{v})$

由此，得到用户对未浏览电影的预测评分，输出前十部电影。由用户间的相似度，输出推荐用户。

4.隐语义模型与 LFM 算法

用户先前对物品的评分行为可以利用用户－物品评分矩阵 R 来描述，R[u][i]表示目标用户 u 对特定物品 i 的打分值。先用均值插补为 R`,后做因子分解：

$$R` = USV^T$$

但由于储存空间需求大，计算复杂，无法实现。故使用 LFM 技术实现。即：

$$R \approx PQ^T = \hat{R}$$

其中$P \in R^{m\times k}, Q \in R^{n\times k}$,k 是分解的特征的数量，做预测：

$$\widehat{r_{ui}} = p_u {q_i}^T = \sum_{t=1}^{k} p_{ut} q_{it}$$

其中，$p_{ut}q_{it}$是模型的参数，优化函数最小如下：

$$\mathrm{C(p,q)} = \sum_{(u,i)\epsilon Train} (r_{ui} - \widehat{r_{ui}})^2 = \sum_{(u,i)\epsilon Train} (r_{ui} - \sum_{t=1}^{k} p_{ut} q_{it})^2$$

正则化后如下:

$$\mathrm{C(p,q)} = \sum_{(u,i)\epsilon Train} (r_{ui} - \widehat{r_{ui}})^2 = \sum_{(u,i)\epsilon Train} (r_{ui} - \sum_{t=1}^{k} p_{ut} q_{it})^2 + \lambda|p_u| + \lambda|q_i|$$

优化方法采用梯度下降法：

$$p_{ut} = p_{ut} + \alpha((\left(r_{ui} - \sum_{t=1}^{k} p_{ut} q_{it}\right) \quad q_{it} - \lambda p_{ut}$$

$$q_{it} = q_{it} + \alpha((\left(r_{ui} - \sum_{t=1}^{k} p_{ut} q_{it}\right) \quad p_{ut} - \lambda q_{it}$$

初始化为$\frac{3.75rand(0,1)}{sqrt(k)}, \alpha_1 = k_1\alpha_1, \alpha_2 = k_2\alpha_2$

其中，$k_1 = \sqrt[nusers]{c_1}, k_2 = \sqrt[nmovies]{c_2}$

（二） 数据库设计

1）数据库介绍

LevelDB 是 Google 开源的持久化 KV 单机数据库，具有很高的随机写，顺序读/写性能。对于本项目而言，其最大的优势在于它是用 C++语言编写的一款数据库，相比于其他数据库而言其使用方法更加简便。下面根据 LevelDB 官方文档归结出了其主要特性。

其主要特点有：

1、key 和 value 都是任意长度的字节数组；

2、每一条 K-V 记录默认是按照 key 的字典顺序存储的，我们可以重载这个排序函数；

3、提供的基本操作接口；

4、支持批量操作以原子操作进行；

5、可以创建数据全景的 snapshot(快照)，并允许在快照中查找数据；

6、可以通过前向（或后向）迭代器遍历数据

7、可移植性；

其主要限制有：

1、非关系型数据模型（NoSQL），不支持 sql 语句，也不支持索引；

2、一次只允许一个进程访问一个特定的数据库；

3、没有内置的C/S架构，但开发者可以使用LevelDB库自己封装一个server；

2）数据库存储实现思路

根据项目需求，数据库需要完成三个基本的功能：

1.将原始的 csv 文件经整合处理后导入 leveldb 数据库

2.接收并存储推荐算法端和可视化界面端提供的新信息

3.提供算法端与可视化界面端所需的各项信息

因此本项目中创建了一个名为 LeveldbAPI 的大类，将数据库对象的所有相关属性及方法封装起来，并根据需求提供多种接口接收或输出信息，从而将上述三种功能整合在一起。

其中，在导入数据部分，由于 LevelDB 是一款 Nosql 键值数据库，无法通过一张表或设置主键来访问各项对应值，这就要求我们设计一系列合理的键名来存储和获取各项数据。同时，为了避免每次运行程序时都重复进行冗杂的数据导入步骤，我们认为应设计一个开关检测数据是否已经存在。而对于接收和输出数据部分，主要是在已储存数据的基础上，在类内定义一系列成员函数，对指定数据库的键和值进行操作。

（三） 可视化界面设计

1）选用 node.js 搭建网站后端

考虑到系统使用的便捷性，选用网站的形式来展示成果，node.js 是 javacript 在后端的运行平台，其主要引擎 V8 是由 C++编写，node.js 本身支持用 C++编写一些拓展模块，编译为.node 文件后在 javascript 中调用，但有一些限制，如要使用 V8 中的类库的等，另外 node.js 支持通过 levelup 与 leveldown 中间件来使用 leveldb。

项目使用了 node.js 中的 Express 框架，使用 ejs 模板来渲染 html，无静态页面，网站提供的主要功能有：

1.用户注册，登陆；

2.对电影按关键字检索，按标签检索

3.查看电影详情，对电影进行评分

4 根据用户的评分记录对用户推荐 10 部电影

（四） UML 图设计

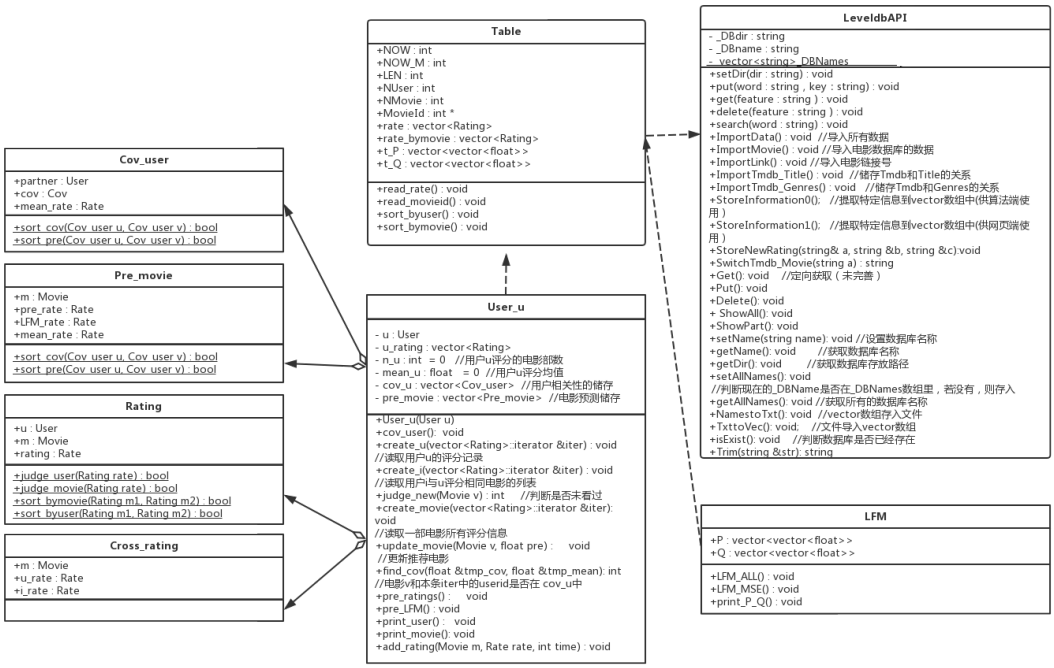

Table 类储存从数据库读入的数据，评分数据 rate、按电影号排序的评分数据 rate_bymovie、评分个数 LEN、用户数 NUser、电影数 NMovie、电影号 MovieId、此时的 P、Q 矩阵。

三、系统实现

（一） 推荐算法实现

1）实现过程中的参数估计问题

实验需要外部参数α_1，c_1，α_2，c_2，需要先确定参数的取值范围，满足：

(1)参数不能过大，否则越过最优解，矩阵 P,Q 的元素达到无穷大。

(2)参数不能过小，否则不能达到最优解。

选择最优参数时在一次训练中寻找，不考虑多次训练。下做实验设计完成参数选取。

通过预实验给出参数的取值范围如下：

X_1: Q 矩阵迭代初值α_1：0.08—0.22；

X_2: P 矩阵迭代初值α_2：0.06—0.13；

X_3: Q 矩阵迭代更新参数c_1：0.005—0.02；

X_4: P 矩阵迭代更新参数c_2：0.00001—0.01；

其中 P 矩阵迭代初值α_1取 8 个水平，分别为 0.08、0.1、0.12、0.14、0.16、0.18、0.2、0.22；Q 矩阵迭代初值α_2取 8 个水平，分别为 0.06、0.07、0.08、0.09、0.1、0.11、0.12、0.13；P 矩阵迭代更新参数c_1取 4 个水平，分别为 0.005、0.01、0.015、0.02；Q 矩阵迭代更新参数c_2取 4 个水平，分别为 0.00001、0.0001、0.001、0.01；对后两个因素采用拟水平法，每个水平重复使用，形式上也为 8 个水平。

因素水平表如下：

表 1　因素水平表

因素＼水平	1	2	3	4	5	6	7	8
X_1	0.08	0.1	0.12	0.14	0.16	0.18	0.2	0.06
X_2	0.06	0.07	0.08	0.09	0.1	0.11	0.12	0.05
X_3	0.005	0.005	0.01	0.01	0.015	0.015	0.02	0.02
X_4	0.00001	0.00001	0.0001	0.0001	0.001	0.001	0.01	0.01

2）实验设计与结果

初始化后模型 MSE 为 200491.6。

使用$U_5^*(8^5)$后的实验设计及结果如下：

表 2　实验设计与结果

实验号	因素				MSE
	X_1	X_2	X_3	X_4	
1	0.08	0.07	0.01	0.01	85393.04
2	0.1	0.09	0.02	0.001	81869.60
3	0.12	0.11	0.01	0.0001	82941.89
4	0.14	0.05	0.02	0.00001	82839.08
5	0.16	0.06	0.005	0.01	81648.04
6	0.18	0.08	0.015	0.001	78645.72
7	0.2	0.1	0.005	0.0001	81385.89
8	0.06	0.12	0.015	0.00001	89602.25

1. 深入分析

对结果进行回归分析，显然 MSE 与因素间不是线性关系，故采用二次多项式回归。本实验中 s = 4，不满足估计回归参数的必要条件，需要采用逐步回归。

结果为：

```
Call:
lm(formula = y ~ x1 + V2 + x4 + V4, data = tdata)

Residuals:
      1       2       3       4       5       6       7       8
-296.68  195.87  110.30 -239.19  300.94 -256.20   80.47  104.49

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   120352       3912  30.762 7.55e-05 ***
x1            197927      33378   5.930  0.00958 **
V2           -172220      23332  -7.381  0.00514 **
x4           1593443     167810   9.496  0.00248 **
V4           -177194      18382  -9.639  0.00237 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 350.9 on 3 degrees of freedom
Multiple R-squared:  0.995,     Adjusted R-squared:  0.9884
F-statistic: 149.5 on 4 and 3 DF,  p-value: 0.0008789
```

其中，$\mathrm{V2} = x_1^{1/2}, \mathrm{V4} = x_4^{1/2}$

求解得到，$X_1 = 0.189$ $X_4 = 0.003$。

试拟合关于X_2的回归方程，其中，$\mathrm{V2} = x_2^{1/2}$，此时$X_2 = 0.072$，此时模型效果不好，但估计值与实验结果基本相符。同理，估计$X_3 = 0.015$。

综上，$X_1 = 0.189$ $X_4 = 0.003$ $X_2 = 0.073$ $X_3 = 0.015$.

验证结果为 MSE = 78094.21，小于实验中任一 MSE，认为实验效果不错。

3）模型后续结果

多次迭代结果如下：

表 3　多次迭代结果

迭代次数	MSE
1	78094.21
2	72943.81
3	69872.38
4	68392.34

以此 P,Q 估计结果进行用户对每一部电影的评分预测。

表 4　部分结果展示

movieId	pre.x	pre.y	userId	rating
47	4.025305	3.585574	2	4
50	4.112032	3.981167	2	4
52	3.401611	3.403205	2	3
62	3.778367	3.293400	2	3
110	4.055734	3.426260	2	4
144	3.106274	2.976700	2	3
150	4.375832	4.214704	2	5
153	3.376312	2.567627	2	4
161	3.565550	3.462637	2	3
165	3.145097	3.314666	2	3
168	2.838175	3.034126	2	3

上图是对用户 2 评分的预测，pre.x 为基于用户的协同过滤预测结果，pre.y 为 LFM 技术的预测结果，rating 为真实评分结果，直观上看，预测结果效果不错。

（二）　数据库操作实现

1）底层数据的导入

本项目中我们设计 LeveldbAPI 类来管理底层数据。该类有三个私有属性，分别为数据库存放路径，数据库名称和一个静态的存储所有数据库名称的 vector 数组。在整个项目中我们需要两个 LeveldbAPI 类的对象 Movie("Movie")和 User("User")，分别管理以电影为主体的和以用户为主体的两大块数据。

下面详细介绍一下这一功能实现中涉及到的几个关键点。

①键名的设计

之前提到，对于 NoSQL 键值数据库而言，我们无法像操作主流的 SQL 数据库那样通过表来访问数据。这就要求我们必须通过设置合理的键名来获取各项信息。举个例子，在存储用户评分的 Excel 表格中共有三列数据：用户 ID、电影 ID 以及评分。显然，直接两两对应储存是行不通的，因此我们给出的解决方案是：设置键的名称使它包含前两种信息（用户 ID+电影 ID），不同属性之间用冒号分隔，之后读取时再写一个函数将键中的信息连同值一并取出来。下面是我们使用

的一些较为典型的键名结构：

格式	示例
"Movie"+MovieID+":"+"Title"	Movie99:Title
"Movie"+MovieID+":"+"Genres"	Movie88:Genres
"User"+UserID+":"+"Movie"+MovieID+":"+"Rating"	User666:Movie33:Rating

②LevelDB 的使用

设计好键名之后，我们就着手将 csv 文件的数据导入数据库。然而在链接 LevelDB 数据库时我们遇到了一些困难。由于之前我们并没有在 Qt 上链接第三方库的经验，加之关于 LevelDB 的书籍和资料相对较少，我们经过无数次的调试，最终才确定了可以正常使用的方法：将 leveldb 文件夹放进项目文件中，在 pro 文件中新增 INCLUDEPATH 和 LIBS 的内容将 Cmake 生成的.a 文件链接进来，然后数据库就会默认在构建好的工程里的 debug 文件夹中出现。

③开关的设置

为了避免重复导入数据，刚开始我们设想将存储数据这一部分单独作为一个项目运行，后来考虑到项目的完整性，我们决定改为在程序内设置一个简易的开关来解决这一问题。我们设置了一个名为 out.txt 的文本文件，初始时无内容。程序开始时通过 LeveldbAPI::TxttoVec() 函数将文件内容存到_DBNames 数组里。在创建 LeveldbAPI 对象之后，通过 isExist() 函数判断该对象的_DBname 是否在数组里，只有当不存在时，才执行 ImportData()导入数据，并将_DBname 压入数组中。最后，再通过 LeveldbAPI::NamestoTxt()函数将现在_DBNames 内的所有内容传入 out.txt 中。这样再次执行程序时就能避免重复导入。

2）底层数据的更新与输出

将所有所需的数据导入数据库中之后，接下来的工作就是配合网页端和算法端实现数据的输出和更新。以算法端为例，需要我们提供的是 UserID，MovieID 和 Rating。因此我们需要将之前存储的键和值通过一系列操作（如按冒号分割、string 转 int 等）转化为符合要求的数据传给他们。这些操作主要是

结合 LevelDB 自身的迭代器以及其他的 C++库函数封装成新的函数来完成，起到主要作用的有 StoreInformation0()，StoreInformation1(), StoreNewRating(string& a, string &b, string &c)等等。

（三） 网站前后端的实现

1）网站后端

用 java 爬取了数据集链接中的 45000 部电影的海报，剧照和内容简介，用 ejs 模板渲染以下部分：

1.mainpage.ejs：主页

2.vip.ejs： 用户登录后的主页

3.register.ejs： 登陆界面.

4.login.ejs: 登陆界面

5.recommend: 给用户推荐电影的界面

6.menu.ejs ：渲染用户登陆前后的选项

7.result.ejs: 渲染搜索电影结果的界面

8.good.ejs :渲染电影详情页的界面

后端使用的数据库有 leveldb 和 redis

Leveldb 存储与电影相关的数据，如电影名，类别标签，id 号，redis 存储和用户有关的信息：以 hash 表来存储：举例如下：

```
127.0.0.1:6379> hgetall user:500
 1) "name"
 2) "500"
 3) "pass"
 4) "$2b$12$HFEaMnIULVqa.1ppxkBFUejsuBYPwB/IkTKqD8J6e5q8sQQsf3dvC"
 5) "email"
 6) "rdlxz@ruc.edu.cn"
 7) "sex"
 8) "man"
 9) "recommend"
10) "null"
11) "id"
12) "500"
13) "salt"
14) "$2b$12$HFEaMnIULVqa.1ppxkBFUe"
```

实现网站后端与数据库和算法的结合方法是：

用 Qt 的网络类进一步封装了算法和数据库相关操作，采用 node.js 提供的 网络模块与算法模块进行通信，举例如下：

1.当用户登陆时，后台会自动发送用户的 ID 号给实现网络通信的推荐算法类，算法进行计算，将推荐的用户和电影 ID 号返回网站后端，后端受到消息后断开连接，后端将数据存储在 redis 该用户哈希表中 recommend 字段下，当用户寺点击推荐选项后，后端通过 levelbd 查找出电影 ID 号对应的电影名，标签，以及海报，用 recommend.ejs 渲染出来，当用户点击图片时后端查找相应的剧照的 URL，内容简介，类别标签，用 good.ejs 渲染出来。

2.当进行查找功能时，后端将用户输入关键字通过 TCP 协议传给 QT 工程 B（由 Qt 网络通信类，leveldb 相关操作，C++vector）组成，工程 B 将 ID 和电影名读入二维 Vector 数组，利用 find 方法实现关键字查找，返回查找结果给后端，用 result.ejs 进行渲染

3.评分功能的实现，用户再点击评分时，会提交评分分数，与当前电影的 ID 号，后台再提取当前会话用户的 ID，组成 UserID+MovieID+rating 的字符串发送给 Qt 工程，存储到 leveldb 中。