天津工業大學
TIANJIN POLYTECHNIC UNIVERSITY

机器学习大作业

题　　目　基于 MovieLens 的电影推荐系统

指导教师　刘丁

小组成员　张先智 余元强 李俊杰 张家源 王晨晨

2019 年 1 月 10 日

目录

1. 前言

推荐系统属于资讯过滤的一种应用，它能够将可能受喜好的资讯或实物（例如：电影、电视节目、音乐、书籍、新闻、图片、网页）推荐给使用者。推荐系统首先收集用户的历史行为数据,然后通过预处理的方法得到用户-评价矩阵，再利用机器学习领域中相关推荐技术形成对用户的个性化推荐。有的推荐系统还搜集用户对推荐结果的反馈,并根据实际的反馈信息实时调整推荐策略，产生更符合用户需求的推荐结果。

在本次实验中，我们实现并使用了多种方式的推荐系统算法。

- 基于用户的协同过滤算法

通过计算用户对商品评分之间的相似性，搜索目标用户的最近邻居，然后根据最近邻居的评分向目标用户产生推荐。

- 基于物品的协同过滤算法：

基于物品协同过滤在于透过计算项目之间的相似性来代替使用者之间的相似性。所建立的一个基本的假设：“能够引起使用者兴趣的项目，必定与其之前评分高的项目相似”。

- 推荐系统之隐语义模型：LFM

隐语义模型是最近几年推荐系统领域最为热门的研究话题，它的核心思想是通过隐含特征 (latent factor)联系用户兴趣和物品。相当于一种聚类算法。

- 基于图的推荐算法：PersonalRank

基于图的模型（graph-based model）是推荐系统中的重要内容。其实，很多研究人员把基于 邻域的模型也称为基于图的模型，因为可以把基于邻域的模型看做基于图的模型的简单形式。把用户行为表示成二分图之后，我们的任务就是在二分图上给用户进行个性化推荐。

- 基于标签的推荐算法

推荐系统的目的是联系用户的兴趣和物品，这种联系需要不同的媒介，GroupLens 在一篇文章中表示目前流行的推荐系统基本上通过 3 种方式联系用户兴趣和物品。其中一种方法就是通过特征去联系用户和物品，标签就是一种重要的特征表示方式。

- 基于深度神经网络的推荐算法

卷积神经网络除了在图片识别方向发挥着重要的作用外，在自然语言处理方面也扮演重要角色，本节就是介绍如何利用 CNN 实现在 NLP 中的应用。

除了实现上述推荐系统方面的算法，我们为了有更好的可视化功能，还搭建了一个用来实现实时推荐的电影网站，以给不同的用户进行个性化推荐。

2. MovieLens 数据集介绍

MovieLens 数据集包含多个用户对多部电影的评级数据，也包括电影数据信息和用户属性信息。这个数据集经常用来做推荐系统，机器学习算法的测试数据集。尤其在推荐系统领域，很多著名论文都是基于这个数据集的。

这个数据集有许多个大小的版本，例如 1m、10m 以及 20m 等，出于速度和大小等综合因素的考虑，本文采用 1m 的版本。在该版本中，用户一共有 6040 个，电影一共有 3952 部。

将官网下载的 1m 的数据解压后，我们可以看到 3 个主要的 dat 文件，分别是 movies.dat、users.dat 、ratings.dat。

- users.dat 的格式为“用户 id | 用户年龄 | 用户性别 | 用户职业 | 用户邮政编码”。
- movies.dat 的格式为“影片 id | 影片名 | 影片类型”。

表 电影信息前 5 条记录

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

- ratings.dat 的格式为“用户 id | 影片 id | 评分值 | 时间戳”。

表 评分信息前 5 条记录

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

3. 评估方法

3.1 准确度

3.1.1 评分预测

如果知道了用户对物品的历史评分，就可以从中习得用户的兴趣模型，并预

测该用户在将来看到一个他没有评过分的物品时，会给这个物品评多少分。预测用户对物品评分的行为称为评分预测。

预测准确度度量一个推荐系统或者推荐算法预测用户行为的能力。这个指标是最重要的推荐系统离线评测指标（通过日志系统获得用户行为数据，并按照一定格式生成一个标准的数据集，将数据集按照一定的规则分成训练集和测试集，在训练集上训练用户兴趣模型，在测试集上进行预测），从推荐系统诞生的那一天起，几乎 99%与推荐相关的论文都在讨论这个指标。

这主要是因为该指标可以通过离线实验计算，方便了很多学术界的研究人员研究推荐算法。

通常预测准确度可以由两个指标计算，一个为均方根误差（RMSE），另一个为平均绝对误差（MAE），对于测试集中的一个用户 u 和物品 i，令r_{ij}是用户 u 对物品 i 的实际评分，而$\hat{r_{ij}}$是推荐算法给出的预测评分，那么 RMSE 的定义为

$$\mathrm{RMSE} = \frac{\sqrt{\sum_{u,i\in T}(r_{ij} - \hat{r_{ij}})^2}}{|T|}$$

MAE 采用绝对值计算预测误差，它的定义为：

$$\mathrm{MAE} = \frac{\sqrt{\sum_{u,i\in T}|r_{ij} - \hat{r_{ij}}|}}{|T|}$$

这两种方式各有各的优点，RMSE 主要可以加大对于预测不精准的惩罚。

3.1.2 TopN 推荐

电影推荐的目的是找到用户最有可能感兴趣的电影，而不是预测用户看了电影之后会有什么样的评分，因此在 TopN 推荐更符合实际应用。

网站在提供推荐服务时，一般是给用户一个个性化的推荐列表，这种推荐叫做 TopN 推荐。TopN 推荐的预测准确率一般通过准确率（precision）/召回率（recall）度量。

令 R(u)是根据用户在训练集上的行为给用户作出的推荐列表，而 T(u)是用户在测试集上的行为列表。

本文将召回率定义为：

$$\mathrm{Recall} = \frac{\sum_{u\in U}|R(u)\cap T(u)|}{|T(u)|}$$

准确率定义为：

$$\text{Precision} = \frac{\sum_{u \in U} |R(u) \cap T(u)|}{|R(u)|}$$

召回率描述有多少比例的用户—物品评分记录包含在最终的推荐列表中，而准确率描述最终的推荐列表中有多少比例是发生过的用户—物品评分记录。

3.2 覆盖率

覆盖率（coverage）描述一个推荐系统对物品长尾的发掘能力。覆盖率定义为推荐系统能够推荐出来的物品占总物品集合的比例。假设系统的用户集合为 U，推荐系统给每个用户推荐一个长度为 N 的物品列表 R(u)。那么推荐系统的覆盖率可以通过下面的公式计算：

$$\text{Coverage} = \frac{|\bigcup_{u \in U} R(u)|}{|I|}$$

当覆盖率为 100%时则推荐系统可以将所有物品至少推荐给一个用户，一般来说，热门排行榜的覆盖率时比较低的，而一个好的推荐系统不仅需要有比较高的用户满意度，也要有较高的覆盖率。

但是上面的定义过于粗略。覆盖率为 100%的系统可以有无数的物品流行度分布。为了更细 致地描述推荐系统发掘长尾的能力，需要统计推荐列表中不同物品出现次数的分布。如果所有的 物品都出现在推荐列表中，且出现的次数差不多，那么推荐系统发掘长尾的能力就很好。因此，可以通过研究物品在推荐列表中出现次数的分布描述推荐系统挖掘长尾的能力。如果这个分布比 较平，那么说明推荐系统的覆盖率较高，而如果这个分布较陡峭，说明推荐系统的覆盖率较低。

在信息论和经济学中有两个著名的指标可以定义覆盖率。

第一个是信息熵：

$$\text{H} = -\sum_{i=1}^{n} p(i) log\, p(i)$$

这里 p(i)是物品 i 的流行度除以所有物品流行度之和。

第二个是基尼指数：

$$\text{G} = \frac{1}{n-1} \sum_{j=1}^{n} (2j - n - 1) p(i_j)$$

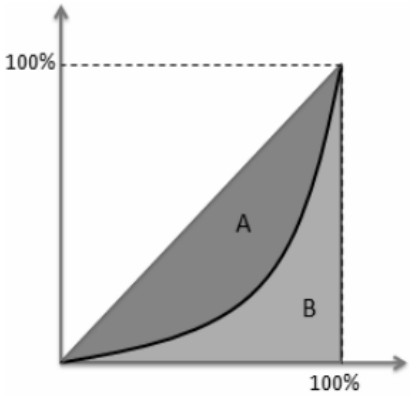
这里，i_j 是按照物品流行度 p()从小到大排序的物品列表中第 j 个物品

基尼系数的计算原理

首先，我们将物品按照热门程度从低到高排列，那么右图中的黑色曲线表示最不热门的 x%物品的总流行度占系统的比例 y%。这条曲线肯定是在 y=x 曲线之下的，而且和 y=x 曲线相交在 (0,0)和(1,1)。

令 SA 是 A 的面积，SB 是 B 的面积，那么基尼系数的形象定义 就是 SA / (SA + SB)，从定义可知，基尼系数属于区间[0,1]。

如果系统的流行度很平均，那么 SA 就会很小，从而基尼系 数很小。如果系统物品流行度分配很不均匀，那么 SA 就会很大， 从而基尼系数也会很大。



推荐系统的初衷是消灭马太效应，使得各种物品都有机会去展示对它们感兴趣的某一类人群。评测推荐系统是否具有马太效应的简单办法就是使用基尼系数。如 果 G1 是从初始用户行为中计算出的物品流行度的基尼系数，G2 是从推荐列表中计算出的物品流 行度的基尼系数，那么如果 G2 > G1，就说明推荐算法具有马太效应。

3.3 新颖性

新颖的推荐是指给用户推荐那些他们以前没有听说过的物品。在一个网站中实现新颖性的最简单办法是，把那些用户之前在网站中对其有过行为的物品从推荐列表中过滤掉。比如在电影网站中，新颖的推荐不应该给用户推荐那些他们已经看过、打过分或者浏览过的电影。但是，有些电影可能是用户在别的网站看过，或者是在电视上看过，因此仅仅过滤掉本网站中用户有过行为的物品还不能完全实现新颖性。

评测新颖度的最简单方法是利用推荐结果的平均流行度，因为越不热门的物品越可能让用户觉得新颖。因此，如果推荐结果中物品的平均热门程度较低，那么推荐结果就可能有比较高的新颖性。

4. 协同过滤算法

用户行为在个性化推荐系统中一般分两种——显性反馈行为和隐性反馈行为。显性反馈行为包括用户明确表示对物品喜好的行为，很多网站都使用了 5 分的评分系统来让用户直接表达对物品的喜好，但也有些网站使用简单的“喜欢”或者“不。隐性反馈行为指的是那些不能明确反应用户喜好的行为，最具代表性的隐性反馈行为就是页面浏览行为，用户浏览一个物品的页面并不代表用户一定喜欢这个页面展示的物品，比如可能因为这个页面链接显示在首页，用户更容易点击它而已。我们可以通过用户停留在页面上的时间来评估其喜好程度。

本文采用的为隐性反馈行为，每一条行为记录仅仅包含用户 ID 和物品 ID。

4.1 基于用户的协同过滤算法

基于用户的协同过滤算法是推荐系统中最古老的算法，通常包含两个步骤：

- 找到和目标用户兴趣相似的用户集合。
- 找到这个集合中的用户喜欢的，且目标用户没有听说过的物品推荐给目标用户。

4.1.1 原理

对于任务一，最重要的是构建用户之间的兴趣相似度，直观上认为，当两个用户之间观看的电影的交集较高时，则他们之间相似度较高。对于给定的用户 u 和用户 v，令 N(u)表示用户 u 曾经有过正反馈的物品集合，令 N(v)为用户 v 曾经有过正反馈的物品集合。那么，我们可以通过余弦相似度公式简单地计算 u 和 v 的兴趣相似度：

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

但用余弦相似度计算两两用户之间的相似度当用户数量庞大时时间复杂度较高，很多计算是不必要的，因为很多用户之间并没有对相同的电影进行评分。为此，本文建立了物品到用户的倒排表，对于每个物品都保存对该物品产生过行为的用户列表，流程图如下所示。

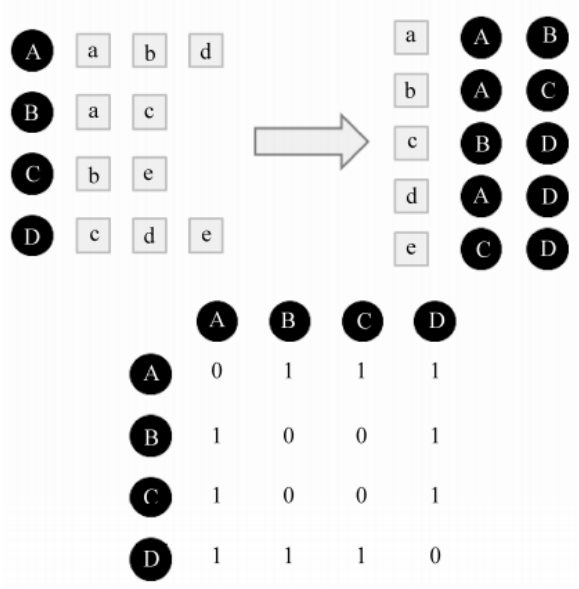


图 物品-用户倒排表

在构建完用户相似度矩阵后，接下来完成任务二，即将物品推荐给目标用户。

利用倒排表找出与某个用户相似度最高的 K 的用户，利用 K 个用户喜欢的电影对该用户进行推荐，具体公式如下，该公式度量了基于用户的协同过滤算法

中用户 u 对物品 i 的感兴趣程度。

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi}$$

其中，S(u, K)包含和用户 u 兴趣最接近的 K 个用户，N(i)是对物品 i 有过行为的用户集合，w_{uv} 是用户 u 和用户 v 的兴趣相似度，r_{vi} 代表用户 v 对物品 i 的兴趣，因为使用的是单一行为的隐反馈数据，所以所有的 r_{vi} 为 1；若使用的为显反馈数据，则可以令 r_{vi} 为对应的评分。

4.1.2 实验结果

对于基于用户的协同过滤算法来说，最重要的超参数为 K，即选取最接近的用户的个数，不同的取值对于模型的性能会有不同的影响。本文在 K 值为 10、20、40 的情形下分别做了实验，结果汇总如下：

表 UserCF 算法在不同 K 参数下的性能

K	准确率	召回率	覆盖率	流行度
10	33.73%	6.80%	40.94%	6.7833
20	37.67%	7.59%	31.75%	6.9197
40	40.41%	8.14%	23.95%	7.0309

从结果可以看出，随着 K 值的增大，准确率、召回率、流行度均有所提高，但覆盖率有所降低，所以实际上 K 的选取需要结合综合性因素加以考虑。

4.2 基于物品的协同过滤算法

基于物品的协同过滤算法是目前业界应用最多的算法。使用这个算法的原因主要有两点。首先是随着网站的用户数目越来越大，计算用户兴趣相似度矩阵将越来越困难；其次基于用户的协同过滤很难对推荐结果作出解释，而使用基于物品的协同过滤则可以较好的做出解释。

基于物品的协同过滤算法和基于用户的协同过滤算法类似，通常也分为两步：

- 计算物品之间的相似度。
- 根据物品的相似度和用户的历史行为给用户生成推荐列表。

4.2.1 原理

首先我们需要计算物品之间的相似度，从直观上说两个物品的相似度可定义如下：

$$W_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|}$$

其中 N(i)表示喜欢物品 i 的用户，N(j)表示喜欢物品 j 的用户，则$N(i) \cap N(j)$表

示同时喜欢物品 i 和物品 j 的用户。使用这个公式在绝大多数情况下计算物品间的相似度是合适的，但本文考虑到特殊情况，即当物品 j 为热门物品时，$N(i) \cap N(j)$趋向于 N(i)，则 W_{ij} 趋向于 1，则在实际运行中不会达到较好的效果。故我们在公式中加上对喜欢物品 j 的用户惩罚项，改进后的公式为:

$$W_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

通过使用这个公式可以减轻热门物品会和很多物品相似的可能性。一般来说，同系列的电影、同主角的电影、同风格的电影、同国家和地区的电影会有比较大的相似度。

在构建完物品相似度矩阵后，任务二需要将待推荐的物品推荐给目标用户。

本文使用的方法与基于用户的协同过滤算法思想类似，即通过如下公式计算用户 u 对物品 j 的兴趣：

$$\mathrm{p(u,j)} = \sum_{i \in N(u) \cap S(j,k)} w_{ji} r_{ui}$$

其中 N(u)是用户喜欢的物品的集合，S(j,K)是和物品 j 最相似的 K 个物品的集合，w_{ji} 是物品 j 和 i 的相似度，r_{ui} 是用户 u 对物品 i 的兴趣，对于隐反馈数据集，如果用户 u 对物品 i 有过行为，即可令 r_{ui} 为 1；对于显反馈数据，可以令 r_{ui} 为用户对某部电影的评分。

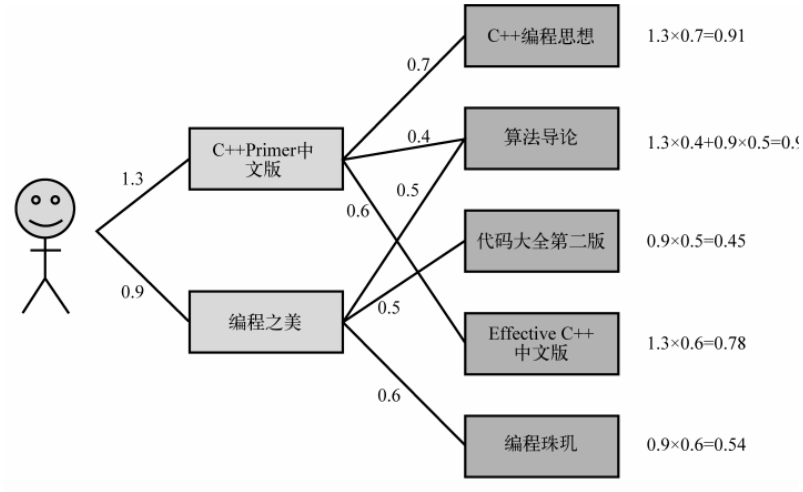


图 计算过程可视化框图

4.2.2 实验结果

本文在 K 值为 10、20、40 的情形下分别做了实验，结果汇总如下：

表 ItemCF 算法在不同 K 参数下的性能

K	准确率	召回率	覆盖率	流行度
10	38.29%	7.72%	19.86%	7.0914
20	37.92%	7.64%	17.29%	7.1730
40	37.23%	7.50%	15.25%	7.2173

从结果可以看出，随着 K 值的增大，准确率、召回率、覆盖率均有所降低，但流行度有所升高，所以实际上 K 的选取需要结合综合性因素加以考虑。

5. 基于隐语义模型的推荐算法：LFM

5.1 概述

隐语义模型是最近几年推荐系统领域最为热门的研究话题，它的核心思想是通过隐含特征 (latent factor)联系用户兴趣和物品。

该算法可以拆分为三个步骤：

- 对电影进行分类。
- 确定用户对哪些类感兴趣以及感兴趣的程度。
- 对于一个给定的类，选择哪些属于这个类的电影推荐给用户，以及如何确定这些电影在一个类中的权重。

对于第一个任务，比较简便的方法为专门雇人对电影进行分类，但这样做是有一些弊端的，例如带有主观因素、难以控制分类的粒度等。

但是隐语义分析技术可以基于用户行为统计而自动聚类，较为客观。

LFM 通过如下公式计算用户 u 对物品 i 的兴趣：

$$\text{Preference}(u,i) = \sum_{f=1}^{F} p_{uk} q_{ik}$$

这个公式中 p_{uk} 和 q_{ik} 是模型的参数，其中 p_{uk} 度量了用户 u 的兴趣和第 k 个隐类的关系，而 q_{ik} 度量了第 k 个隐类和物品 i 之间的关系。

故 LFM 的主要任务即为找到用户-潜在因子矩阵 Q 与潜在因子-电影矩阵 P。

5.2 Q 和 P 矩阵的计算方法

Q 矩阵表示不同的用户对于电影不用元素的偏好程度，如下表所示：

表 用户-潜在因子矩阵 Q 示例

	动作	爱情	喜剧
小杰	0.9	0.9	0.5
小强	0.8	0.7	0.9

而 P 矩阵表示每种电影含有各种元素的成分，如下表所示：

表 潜在因子-电影矩阵 P 示例

	动作	爱情	喜剧
西游记	0.7	0	0.1
敢问路在何方	0.8	0	0.2

根据 Q 和 P 矩阵，我们便可以得到用户对电影的偏爱程度，如下表所示：

表 用户对电影的偏爱程度示例

	西游记	敢问路在何方
小杰	0.68	0.82
小强	0.65	0.82

但通常情况下用户和电影的数量很多，而且用户对电影的偏爱程度比较稀疏，因为每个用户可能仅仅对其中一些电影进行评分，而更多的电影则没有进行评分。我们需要对偏爱矩阵 R 进行矩阵分解，将其分解为 Q 和 P 矩阵，即

$$\mathrm{R} = \mathrm{Q}P^T$$

但在分解时希望往最优化的目标上进行，即

$$\min \sum (r_{ui} - \widehat{r_{ui}})^2 + \lambda||P||^2 + \lambda||Q||^2$$

其中$\lambda||P||^2 + \lambda||Q||^2$是用来防止过拟合的正则项，该算法为最优化理论中最基础的优化算法，可以通过求参数的偏导数找到最快速的下降方法，即本文采用梯度下降法计算矩阵 Q 与 P，分别对其求偏导，得

$$\frac{\partial C}{\partial p_{uk}} = -2q_{ik} + 2\lambda p_{uk}$$

$$\frac{\partial C}{\partial q_{ik}} = -2q_{uk} + 2\lambda p_{ik}$$

由梯度下降法，可以将上式写为

$$p_{uk} = p_{uk} + \alpha(q_{ik} - \lambda p_{uk})$$

$$q_{ik} = p_{uk} + \alpha(p_{uk} - \lambda q_{ik})$$

5.3 实验结果

本文在隐类个数 F 值为 50、100 的情形下分别做了实验，结果汇总如下：

表 LFM 算法在不同 F 参数下的性能

F	准确率	召回率	覆盖率	流行度
50	35.45%	10.14%	45.34%	6.4328
100	38.92%	12.23%	48.12%	7.1534

从结果可以看出，随着 F 值的增大，所有指标均有上升，但 F 越大时计算速度约慢，所以实际应用时应综合考虑。

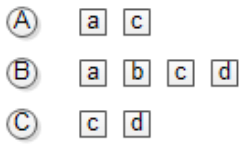
6. 基于图的推荐算法：PersonalRank

6.1 基本概念

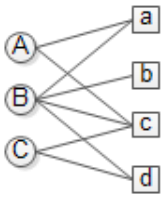
将用户行为数据表示为一系列的二元组，每一个二元组(u,i)代表用户 u 对物品 i 产生过行为，这样便可以将这个数据集表示为一个二分图。

6.2 算法分析

假设我们有以下的数据集，只考虑用户喜不喜欢该物品而不考虑用户对物品的喜欢程度：



其中用户 user=[A,B,C],物品 item=[a,b,c]，用户和物品有以下的关系：



假设要给用户 u 进行个性化推荐，可以从用户 u 对应的节点 v_u 开始在用户物品二分图上进行随机游走。游走到任何一个节点时，首先按照概率 α 决定是继续游走，还是停止这次游走并从 v_u 节点开始重新游走。如果决定继续游走，那么就从当前节点指向的节点中按照均匀分布随机选择一个节点作为游走下次经过的节点。这样，经过很多次随机游走后，每个物品节点被访问到的概率会收敛到一个数。最终的推荐列表中物品的权重就是物品节点的访问概率。

其中算法中人与物品之间的连接是无向的连接。把上边文字描述用公式来表达就是：

$$\mathrm{PR(v)} = \begin{cases} \alpha \sum\limits_{v' \in in(v)} \dfrac{PR(v')}{|out(v')|} (v \neq v_u) \\ (1-\alpha) + \alpha \sum\limits_{v' \in in(v)} \dfrac{PR(v')}{|out(v')|} (v = v_u) \end{cases}$$

PR 代表该节点被访问的概率，初始时只有根节点的访问概率为 1，其他节点的访问概率均为零。经过多次迭代之后各个节点的访问概率会达到一个稳定的

值。

表 模型评估

α	准确率	召回率	覆盖率	流行度
0.8	16.45%	7.95%	3.42%	7.6928

*数据来源:《推荐系统实践》

6.3 算法改进：矩阵实现

由于 PersonalRank 算法在计算的过程中要多次迭代,所需时间复杂度非常高不适合在线甚至离线生成推荐列表，对与大规模数据集更是无能为力。因此为了减少计算时间,基本上有两种方法:第一就是减少迭代次数,在收敛前就停止（为了有较高准确率，仍然需要多次迭代）第二就是从矩阵的角度出发。

把上述公式更改为矩阵形式如下：

$$\mathrm{Rank} = (1-\alpha)M^T Rank + \alpha r_0$$

其中 r0 作为初始的各个节点的概率向量。

M（n*n）的定义如下：

$$\mathrm{M}(\mathrm{v},\mathrm{v}') = \frac{1}{|out(v)|}$$

只要根据公式求解 Rank 的值就可以得到最终结果，求解时也有以下几种方法：

- 直接线性解方程。
- 采用 gmres 方法求解。
- 求逆矩阵法。
- 一次性计算出从任意节点开始游走的 PersonalRank 结果。

6.4 实验结果

根据试验结果，最后一种方法求解速度时最快的，因此此次算法的最终实现也采用了最后一种方法。

表 PersonalRank 算法在不同 Alpha 参数下的性能

Alpha	推荐数量	测试集比例	准确率	召回率	覆盖率	流行度
0.7	15	30%	14.42%	6.80%	37.83%	5.21
0.8	15	30%	15.33%	7.23%	39.10%	5.21
0.9	15	30%	15.37%	7.25%	39.13%	5.24
1	15	30%	22.75%	10.72%	3.32%	5.67

从表中可以看出，随着 Alpha 取值的增加，准确率、召回率、流行度均有所

增加，但覆盖率有所降低。

7. 基于标签的推荐算法

7.1 基本概念

一个用户标签行为的数据集一般由一个三元组的集合表示，其中记录(u, i, b)表示用户 u 给物品 i 打上了标签 b。

打标签作为一种重要的用户行为，蕴含了很多用户兴趣信息，因此深入研究和利用用户打标签的行为可以很好地指导我们改进个性化推荐系统的推荐质量。同时，标签的表示形式非常简单，便于很多算法处理。

打标签需要考虑三个问题：

- 用户为什么要打标签？
- 用户怎么打标签？
- 用户打什么样的标签？

7.1.1 用户为什么要打标签？

在设计基于标签的个性化推荐系统之前，我们需要深入了解用户的标注行为(即打标签的行为)，知道用户为什么要标注，用户怎么标注，只有深入了解用户的行为，我们才能基于这个行为设计出令他们满意的个性化推荐系统。

7.1.2 用户怎么打标签？

在互联网中每个人的行为都是随机的，但其实这些表面的行为隐藏着很多规律，所以我们引入了标签流行度来对用户打的标签进行统计。我们定义的一个标签被一个用户使用在一个物品上，它的流行度就加 1.

7.1.3 用户打什么样的标签？

在用户打标签时，我们希望他打的标签能够准确描述物品内容属性的关键词，因此我们希望用户打如下几类标签。

- 表明物品是什么
- 表明物品的种类
- 表明物品的所有者
- 表达用户的观点
- 用户相关的标签
- 用户的任务

7.2 标签算法及优化

7.2.1 算法流程

1、统计每个用户的常用标签

2、对于每个标签，统计打过这个标签次数较多的物品

3、对于每一个用户，找到他的常用标签，然后找到具有这些标签的最热门物品

7.2.2 用户 u 对物品 i 的兴趣公式

$$\mathrm{P}(\mathrm{u},\mathrm{i}) = \sum_{b} n_{u,b} n_{b,i}$$

$N_{u,b}$ 是用户 u 打过的标签 b 的次数 ，$N_{b,i}$ 是物品 i 被打过标签 b 的次数。

7.2.3 缺点

上述算法有缺点，会倾向于给热门标签对应的热门物品有很大的权重，会造成推荐热门的物品给用户，会降低推荐结果的新颖性。因此我们对上述算法进行优化，给热门程度加一个惩罚项。

$$\mathrm{p}(\mathrm{u},\mathrm{i}) = \sum_{b} \frac{n_{u,b}}{\log(1+n_b^{(u)})} \times \frac{n_{b,i}}{\log(1+n_i^{(u)})}$$

对于标签很少的情况，可以通过计算相似标签来扩展标签，计算算法可以为余弦相似度。标签扩展可以提升准确率和召回率。

7.2.4 标签清理

标签系统中经常出现词形不同而词意相同的标签，我们应该去除这些因词根不同造成的同义词，去除因分隔符而造成的同义词。这些标签也不能包含没有意义的停止词和表示情绪的词。

7.3 给用户推荐标签

给用户推荐标签有两个好处：一是方便用户输入标签，二是提高标签质量。

用户 u 给物品 i 打标签，我们有 4 种简单的方法：

- 给用户 u 推荐整个系统里最热门的标签。
- 给用户 u 推荐物品 i 上最热门的标签。

- 给用户 u 推荐他自己经常使用的标签。
- 前面两种的融合，既给用户推荐物品 i 上最热门的标签，也推荐他自己经常使用的标签。

8. 基于深度神经网络的推荐算法

除了常规的方法外，还可以采用基于深度神经网络的推荐算法，依然可以达到较好的推荐效果，但可解释性没有以上算法好，由于在实际应用中不常使用该算法，仅仅在理论上加以研究，故此部分仅作简单介绍。

8.1 数据预处理

users.dat 的格式为“用户 id | 用户年龄 | 用户性别 | 用户职业 | 用户邮政编码”，本文中将“用户邮政码”字段去除，并将用户性别中 F 改为 1，M 改为 0，并将年龄字段转换为连续的数字 0-6。

movies.dat 的格式为“影片 id | 影片名 | 影片类型”，由于影片名中包含上映日期，故在预处理时使用正则表达式将其去除；影片类型通常包含多个类别，故我们可以将其转换为词向量。

8.2 嵌入层（Embedding）

引入嵌入层的原因主要有两点：

- 独热编码（One-hot encoding）向量是高维且稀疏的。
- 每个嵌入向量会在训练神经网络时更新，我们可以可视化词语之间的关系。

例如句子“Liu Ding is a very good teacher and we all like him.”中有 12 个单词，则不使用嵌入层而只使用 one-hot 编码时每个词向量时 12 维的，而且该向量非常稀疏，会占用大量不必要的资源，而且不能看出每个单词之间的关系。

而通过嵌入层可以将每个单词的潜在因素找出来，通常个数远远小于单词的总个数，从而可以使词向量变为稠密的，提高效率。

8.3 文本卷积网络

卷积神经网络也可以用在自然语言处理上，在处理影片名称时需要使用文本卷积网络，因为影片名称中往往包含了多个单词。

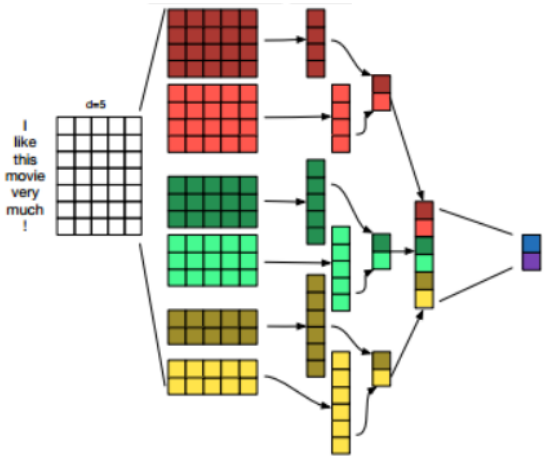


图 文本卷积网络示意图

在文本卷积网络中，卷积核的宽度为每个单词的维度，高度分别为 4、3、2，每个卷积核分别有 2 个。在滑动时是按照单词进行，卷积后进行最大池化，再经过 softmax 层即可将每个句子转化为 2 个像素进行后续的处理。

8.4 流程框架

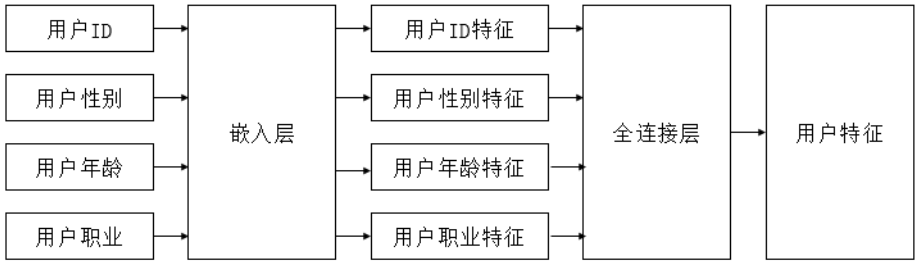


图 获取用户特征流程图

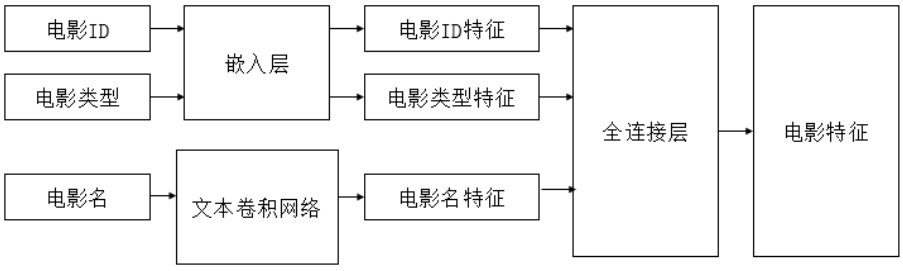


图 获取电影特征流程图

获取用户特征以及电影特征后，即可对评分进行预测。

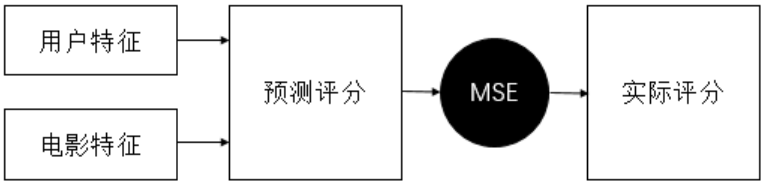


图 评分预测以及优化流程图

9. 算法简要对比

9.1 基于用户与基于物品的协同过滤算法

	UserCF	ItemCF
性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数量明显小于用户数量的场合，如果物品很多，计算物品的相似度矩阵代价会很大
领域	时效性较强，用户个性化兴趣不太明显。例如：新闻	长尾物品丰富，用户个性化需求较强
实时性	用户有新行为，不一定造成推荐结果立即变化	用户有新行为一定会导致推荐结果变化
冷启动	新用户对很少物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他产品
	新物品上线后，一旦有用户对它产生行为，就可以把新物品推荐给和该用户行为相似的用户	没有办法在不更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的理由。例如：“购买过该物品的用户也购买xxx”	利用用户的历史行为做推荐解释，有很高的说服力。例如：“因为您之前买过 xxx，我们猜您也喜欢 xxx”

9.2 隐语义模型和协同过滤算法

	LFM	协同过滤算法
理论基础	LFM 具有比较好的理论基础，它是一种学习方法，通过优化一个设定的指标 建立最优的模型	只是一种统计方法，没有训练过程
离线计算空间复杂度	LFM 在建模的过程中，如果是 F 个隐类，需要的存储空间是 O(F*(M+N))，这在 M 和 N 很大的	在离线计算的过程中，需要维护一张离线的相关表。如果用户/物品数量很多，将会占据很大的内

	时候可以节省离线计算空间	存，如果有 M 个用户，N 个物品，UserCF 需要 O(M*M) 空间，ItemCF 需要 O(N*N)空间
离线计算时间复杂度	假设有 M 个用户、N 个物品、K 条用户对物品的行为记录。LFM，如果用 F 个隐类，迭代 S 次，那么它的计算复杂度 是 O(K * F * S) 。在一般情况下，LFM 的时间复杂度要稍微高于 UserCF 和 ItemCF，这主要是因为该算法需要多次迭代	UserCF 计算用户相关表的时间复杂度是 O(N * (K/N)^2)，而 ItemCF 计算物品相关表的时间 复杂度是 O(M*(K/M)^2)。 但总体上，这两种算法的时间复杂度没有质的差别
在线实时推荐	LFM 在给用户生成推荐列表时，需要计算用户对所有物品的兴趣 权重，然后排名，返回权重最大的 N 个物品。那么，在物品数很多时，这一过程的时间 复杂度非常高，可达 O(M*N*F)。LFM 不太适合用于物品数非常庞大的系统，另一方面，LFM 在生成一个用户推荐列表时速度太慢，因此不能在线实时计算	UserCF 和 ItemCF 在线服务算法需要将相关表缓存在内存中，然后可以在 线进行实时的预测。以 ItemCF 算法为例，一旦用户喜欢了新的物品，就可以通过查询内存中的相关表将和该物品相似的其他物品推荐给用户。因此，一旦用户有了新的行为， 而且该行为被实时地记录到后台的数据库系统中，他的推荐列表就会发生变化
推荐理由	LFM 无法提供良好的解释，它计算出的隐类虽然在语义上确实代表了一类兴趣和物品，却很难用自然语言描述并生成解释展现给用户。	ItemCF 算法支持很好的推荐解释，它可以利用用户的历史行为解释推荐结果。

9.3 基于图的模型和协同过滤算法

	图模型（矩阵形式）	协同过滤
离线计算空间复杂度	假设有 M 个用户 N 个物品，需要的存储空间是 O((M+N)*(M+N))，所以图模型需要的储出空间要大得多	UserCF 需要 O(M*M) 空间，ItemCF 需要 O(N*N)空间
离线计算时间复杂度区别	需要对占用空间是 O((M+N)*(M+N))的矩阵求逆矩阵，整体上实验中计算速度要比协同过率要快	需要对用户和用户（或者物品和物品）求相似度矩阵
在线实时推荐	用户的行为发生变化，就需要重新计算矩阵的数据，在物品和用户数量很多的时候，实时推荐比较困难	ItemCF 可以直接从缓存表中查找数据，UserCF 则需要重新计算
兴趣度体现	图模型只能体现用户对哪些物品有过兴趣，而不能体现对不同物品的兴趣度高低（或者叫做暂时无法实	可以体现不同用户对不同物品有着不同的兴趣度，然后把兴趣度较高的物品排名靠前

	现该功能）	
推荐理由	无法提供良好的解释，图模型存储着用户与用户，物品与用户，物品与物品之间的关系，计算过程无法直观用自然语言去描述	良好的可解释性

10. 推荐网站展示

开发环境：Python3 + Windows

第三方模块：Tornado

集成开发环境：PyCharm

10.1 前端部分

所用技术和框架：Bootstrap、jQurey，

具体实现:前端部分首先建立电影推荐网站的模板，整体模板如下:

导航栏在上部:

左侧网站内容

右侧登录注册功能

左侧电影分类推荐:

中间用户电影推荐

右侧评分最高电影推荐

前端代码结构如下:

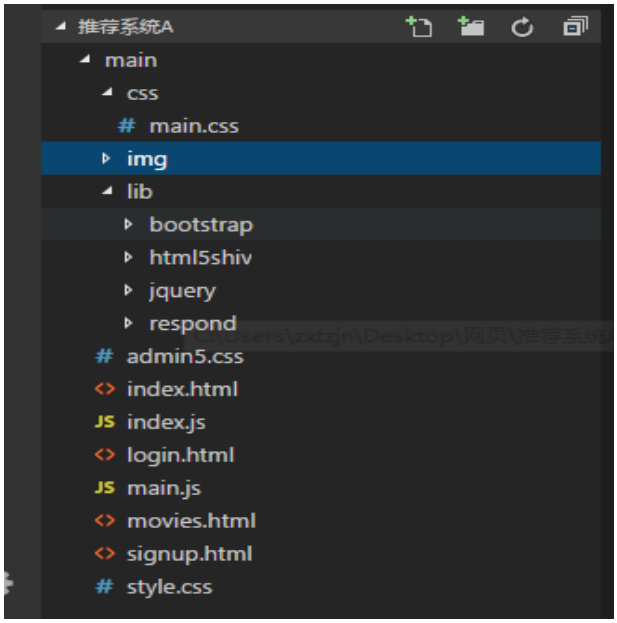


图 前端代码结构

Bootstrap 的目标是在最新的桌面和移动浏览器上有最佳的表现，也就是说，在较老旧的浏览器上可能会导致某些组件表现出的样式有些不同，但是功能是完整的。

被支持的浏览器：

手机端：

	Chrome	Firefox	Safari
Android	✔ Supported	✔ Supported	N/A
iOS	✔ Supported	✔ Supported	✔ Supported

图 可支持手机端

PC 端：

	Chrome	Firefox	Internet Explorer	Opera	Safari
Mac	✔ Supported	✔ Supported	N/A	✔ Supported	✔ Supported
Windows	✔ Supported	✔ Supported	✔ Supported	✔ Supported	✖ Not supported

图 可支持 PC 端

在网站中我们使用了 Bootstrap 栅格系统进行页面布局：

Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多 12 列。它包含了易于使用的预定义类，还有强大的 mixin 用于生成更具语义的布局。

.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1
.col-md-8								.col-md-4			
.col-md-4				.col-md-4				.col-md-4			
.col-md-6						.col-md-6					

图 前端栅格系统

我们使用 Bootstrap 的表单系统实现用户的登录功能:

后端分析：可供 web 后端开发的 Python web 框架有 Django、Tornado、和 Flask。先分析一下三者在三方面上的性能。（参考网站：https://www.jianshu.com/p/9960a9667a5c）

比较方面如下：

- JSON：序列化一个对象，并返回一个 json。
- 远程性能：从远程服务器上返回 http response 的时间
- 数据库性能：使用 ORM（对象关系映射）从数据库获取数据，并渲染到模板上的时间

A. 最基本的 JSON 测试：Django 与 Flask 占优

单纯在本地测试 JSON 的序列化， Django 完成一次 JSON 序列化的平均时间 42.52 毫秒，每秒请求量 4762 次。Flask 在此项测试中，与 Django 的比较不相上下，Flask 平均时间 43.33 毫秒，每秒请求量 4630 次。Tornado 完成 JSON 序列化的平均时间高达 77.51 毫秒，是所有框架中耗时最长的，每秒请求数是 2578 次，也是低于 Django 与 Flask 的水准。这仅仅说明框架在本地处理 JSON 的速度。框架还涉及 http request/response 以及数据库的读写，后面还需要综合来分析框架的性能。

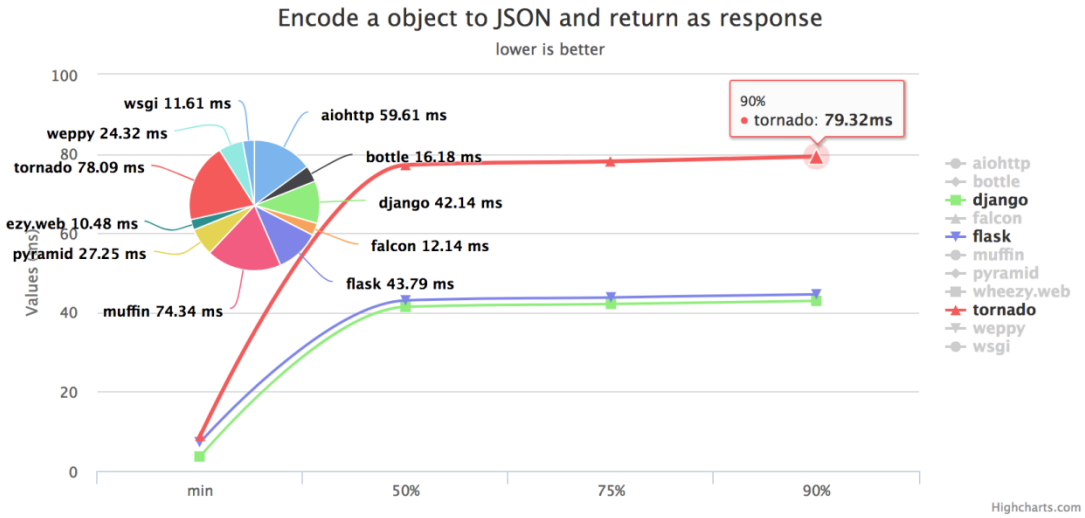


图 JSON 测试分析饼状图

Name	50% (ms)	75% (ms)	Avg (ms)	Req/s	Timeouts
WSGI	9.47	11.61	9.17	21821	
Wheezy.Web	10.23	10.48	10.24	19542	
Falcon	11.7	12.14	11.72	17062	
Bottle	15.89	16.18	15.61	12838	
Weppy	23.66	24.32	24.05	8345	
Pyramid	26.84	27.25	27.13	7383	
Django	41.41	42.14	42.52	4762	
Flask	43.03	43.79	43.33	4630	
Aiohttp	58.96	59.61	64.49	3368	
Muffin	73.5	74.34	81.7	2703	
Tornado	77.17	78.09	77.51	2578	

图 JSON 测试分析表格数据

B. 处理远程 http 请求的能力：Tornado 占绝对优势

从这项测试开始，Tornado 的强悍开始显现。Tornado 完成 http 请求的平均时间是 1.04 秒，而 Flask 是 3.34 秒，Django 是 3.48 秒，http 响应速度 Tornado 比 Flask 以及 Django 快三倍。

值得注意是，如果综合考虑 http 相应速度以及 JSON 处理速度，如果把两项指标的平均时间相加：Tornado 耗时 1114.48 毫秒，Flask 是 3387.60 毫秒，Django 是 3519.88 毫秒。

Tornado 的好成绩得益于其自带的异步特性，而 Django 与 Flask 是同步框架，在处理请求时性能受限。但是实际使用中，一般是 Django/Flask + Celery + Redis/Memchaned/RabbitMQ 的模式，由此带上了异步处理的能力。

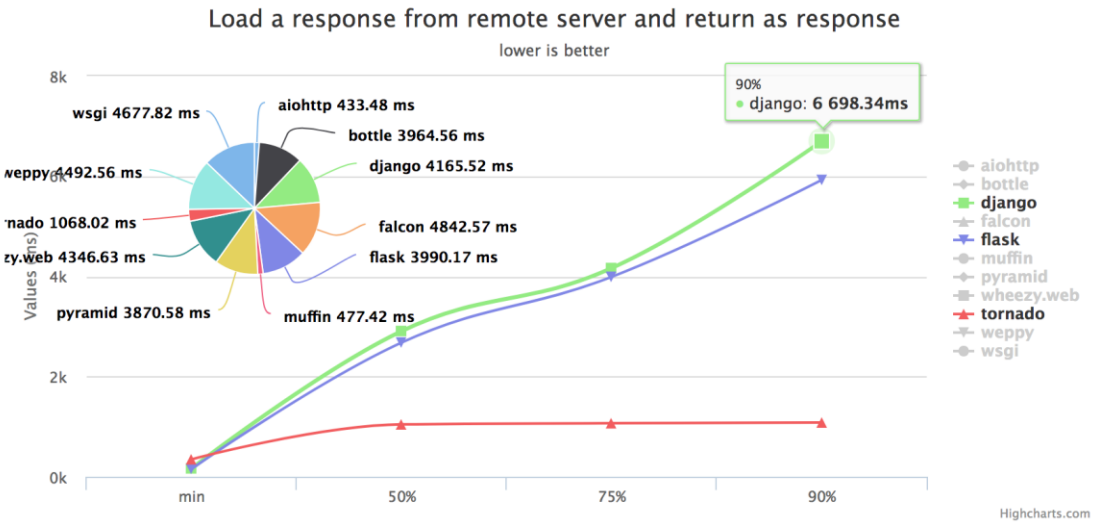


图　http 请求能力分析饼状图

Name	50% (ms)	75% (ms)	Avg (ms)	Req/s	Timeouts
Aiohttp	412.08	433.48	406.53	483	615.20
Muffin	451.77	477.42	449.69	439.1	617.90
Tornado	1044.8	1068.02	1036.97	187.8	103.65
WSGI	2607.26	4677.82	3578.6	18.4	16.15
Falcon	2944.88	4842.57	3629.31	18.35	14.20
Wheezy.Web	3125.03	4346.63	3573.43	18.35	13.70
Bottle	2885.36	3964.56	3207.14	18.3	15.85
Weppy	2593.71	4492.56	3468.05	18.25	16.15
Flask	2679.56	3990.17	3344.27	18.15	13.25
Django	2908.53	4165.52	3477.36	18.1	14.30
Pyramid	2980.71	3870.58	3305.18	18	14.10

图 http 请求能力表格数据

C.数据库与模板处理性能：Tornado 与 Flask 旗鼓相当

Django 饱受诟病的地方就是 Django ORM 确实很慢，加上模板处理时间，Django 的平均时间 2904.04 毫秒，每秒处理请求量 42.9 次。然而 Django 的大部分功能是建立在其 Django ORM 基础上，比如 models, admin, forms 甚至第三方框架 django-rest-framework。Django 的开发效率与维护非常地棒，然而 Django ORM 深度绑定了该框架，如果你需要把 Django ORM 换成其它轮子，

那么也意味着 Django 的诸多优秀特性将从此告别。

Flask 事实上的 ORM 是 SQLAlchemy，根据董伟明的估计，SQLAlchemy 比 MySQLdb 的耗时多 5% 左右，所以是性能相当不错的数据库 ORM。得益于 SQLAlchemy 的优异性能，Flask 的每秒处理请求数为 123 次，平均处理时间 1440.24 秒，与 Tornado 性能相当。

Tornado 的每秒处理请求数为 143 次，平均处理时间 1344.69 秒。对于数据库与模板的处理，Tornado 与 Flask 不相上下。

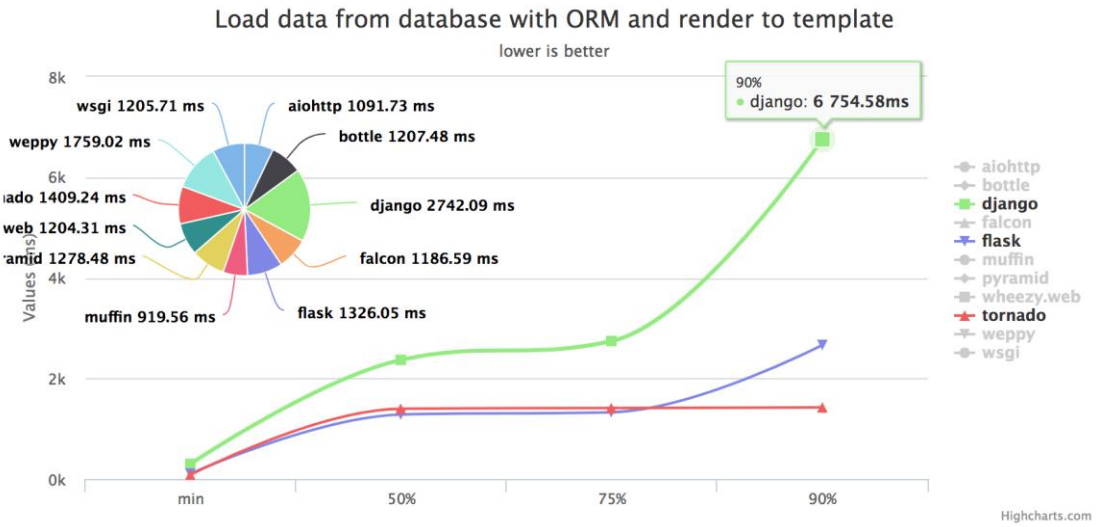


图 数据库与模板处理性能分析饼状图

Name	50% (ms)	75% (ms)	Avg (ms)	Req/s	Timeouts
Aiohttp	10.51	1091.73	925.21	204.8	
Muffin	737.57	919.56	990.87	158.3	39
Falcon	1142.31	1186.59	1269.32	146.85	
WSGI	1165.66	1205.71	1331.75	144.35	
Tornado	1395.43	1409.24	1344.69	143	
Wheezy.Web	1129.41	1204.31	1377.88	138.05	
Bottle	1167.93	1207.48	1398.16	128.2	
Flask	1281.38	1326.05	1440.24	123	
Pyramid	1247.13	1278.48	1497.29	120.65	
Weppy	1578.43	1759.02	1899.08	88.5	
Django	2367.04	2742.09	2904.04	42.9	16

图 数据库与模板处理性能表格数据

10.2 后端部分

前端发送 http 请求

后端 socket 服务发送数据，处理 http 请求

路由系统处理 http 请求的分发

响应网页并返还给浏览器

最终经过解析渲染展现给用户

本项目在前端的基础上利用 Tornado 框架实现 Python web 开发。

大体流程是导入 Tornado 框架、实现各部门类（比如登陆界面部门类、主页面部门类）、实现路由选择系统、导入写好的算法最终实现。

11. 冷启动问题

11.1 产生原因

协同过滤推荐算法在缺少数据或者数据稀疏的情况下推荐效果急剧下降，这是由于算法无法利用足够的信息进行用户行为分析，因此无法产生准确的推荐效果。表 1 为用户－项目评分事例(评分范围为 1～5 分) ，“－”表示没有评分。

表 用户对项目的评分事例

	Item1	Item2	Item3	Item4	Item5
User1	3	5	-	5	4
User2	-	-	-	-	-
User3	2	3	-	-	-
User4	-	-	-	4	3
User5	4	5	-	5	5
User6	3	4	-	5	-

该表中描述了 6 个用户$(User_1, User_2, User_3, User_4, User_5, User_6)$对 4 个物品$(Item_1, Item_2, Item_3, Item_4, Item_5)$ 的评分信息。系统根据用户已评分信息，可以分析出 $User_1$，$User_5$与 $User_6$具有相似的喜好，因为他们对 $Item_1$，$Item_2$和 $Item_4$有着极为相似的评分，那么系统会 推荐 $Item_5$给 $User_6$，因为与 $User_6$偏好相似的 $User_1$与 $User_5$均对 $Item_5$的评分较高。同时，$User_2$作为新用户进入系统而不具有历史评分信息,因此就无法根据协 同过滤推荐算法的原理分析出 $User_2$的行为偏好; 而从物品的角度看，$Item_3$作为新项目存在于系统中，因缺少评分信息导致系统无法感知它的存在，所以 $Item_3$也就无法被推荐出去。这就形成了协同过滤推荐系统中的冷启动问题。

11.2 分类

(1) 新用户冷启动: 假设新用户 u 进入该模型，由于 u 不完善的个人信息与行为信息，因此系统不能通 过模型分析 u 的偏好，推荐系统也就无法利用推荐算法向该用户推荐其可能喜欢的物品。这种现象被称为 新用户的冷启动问题。

(2) 新物品冷启动: 假设新物品 i 进入该模型，由于 i 没有被用户评价的信息，导致评分数据的稀少，系 统不能建立更详细的特征信息表，也就不能将其与用 户的喜好进行很好的匹配，因此新物品 i 被推荐的概 率就大大减少。这种现象称为新物品的冷启动问题。

(3) 新系统冷启动: 可以将其理解为(1) 、(2) 的一种特殊情况，即所有的用户、项目对于系统来说都是新的，一个全新的系统无法产生相应的推荐即为新系统 的冷启动问题。

11.3 解决方案

（1）针对用户冷启动，通常参考如下解决方案：

A. 人口统计学信息

利用用户注册时提供的年龄、性别等数据做粗粒度的个性化。

* 通常情况下，包括用户的年龄、性别、职业、名族、学历和居住地等，这些信息有的可以通过注册获得一部分。

* 对于缺失的部分可以考虑建立分类模型来进行预测。例如性别等标签可以单独建立分类模型。

B. 用户兴趣描述-兴趣 tag 收集

有一些网站会让用户用文字描述他们的兴趣。具体的产品形式，

* 比如在注册初始阶段，引导用户选择他们感兴趣的标签，比如豆瓣类的阅读网站。

* 比如在登录时对一些物品进行反馈，收集用户对这些物品的兴趣信息，然后给用户推荐那些和这些物品相似的物品。

C. 非个性化推荐-热门推荐

非个性化推荐的最简单的例子就是热门排行榜，等到用户数据收集到一定的时候,再切换为个性化推荐。

D. 互动询问方式

解决用户冷启动问题的方法还有就是在新用户第一次访问推荐系统时，不立即给用户展示推荐结果，而是给用户提供一些物品。

让用户反馈他们对这些物品的兴趣，然后根据用户反馈给提供个性化推荐。

* 比较热门

* 具有代表性和区分性

* 启动物品集合需要有多样性

E. 第三方数据

有时候，为了获得更丰富的用户行为数据，可以考虑从其他网站导入的用户站外行为数据。比如用户通过豆瓣、新浪微博等账户登录，在用户同意的情况下可以获取一些行为数据。 也有通过某些主键去进行关联，比如用户手机号等，去第三方获取关联的相关标签。

利用用户的社交网络账户登录(需要授权)，导入用户在社交网站上的好友信息，然后给用户推荐其好友喜欢的物品。

（2）针对用户冷启动，通常参考如下解决方案：

A. 利用物品的内容信息

用来解决物品的冷启动问题，即如何将新加入的物品推荐给对它感兴趣的用户。物品冷启动问题在新闻网站等时效性很强的网站中非常重要，因为这些网站时时刻刻都有新物品加入，而且每个物品必须能够再第一时间展现给用户，否则经过一段时间后，物品的价值就大大降低了。

针对协同过滤的两种推荐算法——UserCF 算法、ItemCF 算法来分别了解一下物品冷启动的问题。

UserCF 算法：

针对推荐列表并不是给用户展示内容的唯一列表（大多网站都是这样的）的网站。

当新物品加入时，总会有用户通过某些途径看到，那么当一个用户对其产生反馈后，和他历史兴趣相似的用户的推荐列表中就有可能出现该物品，从而更多的人对该物品做出反馈，导致更多的人的推荐列表中出现该物品。因此，该物品就能不断扩散开来，从而逐步展示到对它感兴趣用户的推荐列表中。

针对推荐列表是用户获取信息的主要途径（例如豆瓣网络电台）的网站。

UserCF 算法就需要解决第一推动力的问题，即第一个用户从哪儿发现新物品。最简单的方法是将新的物品随机战士给用户，但是太不个性化。因此可以考虑利用物品的内容信息，将新物品先投放给曾经喜欢过和它内容相似的其他物品的用户。

ItemCF 算法：

对 ItemCF 算法来说，物品冷启动就是很严重的问题了。因为该算法的基础是通过用户对物品产生的行为来计算物品之间的相似度，当新物品还未展示给用户时，用户就无法产生行为。所以，只能利用物品的内容信息计算物品的相关程度。基本思路就是将物品转换成关键词向量，通过计算向量之间的相似度（例如计算余弦相似度），得到物品的相关程度,将它们推荐给喜欢过和它们相似的物品的用户。

表 常见物品的内容信息

图书	标题、作者、出版社、出版年代、丛书名、目录、正文
论文	标题、作者、作者单位、关键字、分类、摘要、正文
电影	标题、导演、演员、编剧、类别、剧情简介、发行公司
新闻	标题、正文、来源、作者
微博	作者、内容、评论

B. 采用专家标注

很多系统在建立的时候，既没有用户的行为数据，也没有充足的物品内容信息来计算物品相似度。这种情况下，很多系统都利用专家进行标注。

代表系统：个性化网络电台 Pandora、电影推荐网站 Jinni。

以 Pandora 电台为例，Pandora 雇用了一批音乐人对几万名歌手的歌曲进行各个维度的标注，最终选定了 400 多个特征。每首歌都可以标识为一个 400 维的向量，然后通过常见的向量相似度算法计算出歌曲的相似度。

12. 总结

经过一个学期在机器学习方向的研究，因此想借着这次机会做出一个比较完整的、具有实用功能的系统。因此我们选择了推荐系统作为这次作业的课题。

这次作业之前，我们从未接触过这方面的内容。面对一个新的方向，开始的时候真的无从下手，之后在推荐下看了本《推荐系统实践》，开始对推荐系统有了一个宏观的认识。经过简单的讨论开始分配工作各自完成相应的任务。

这次实践作业，我们完成了基于用户的协同过滤、基于物品的协同过滤、隐语义模型的推荐、基于图的推荐及其变形、基于标签的、基于深度神经网络的推荐（未完全完成）算法的设计和各个算法在 MovieLens 数据集上的测试评估，总结各个算法在不同领域的优缺点。除此之外掌握了评分预测和 TopN 推荐之间的区别以及不同评估模型在算法上的评估侧重点。区分在不同的数据集（有代表性的数据集分为以下四种：无上下文信息的隐性反馈数据集、无上下文信息的显性反馈数据集、有上下文信息的隐性反馈数据集、有上下文信息的显性反馈数据集）上采用不同的算法（或修改）达到最好的推荐效果。

在前端展示部分，通过采集用户在网站上的行为，来判定用户的兴趣爱好，同时实时在热门电影和用户喜欢电影部分实现实时刷新，向用户推荐可能喜欢的电影。单一的算法并不能达到理想的效果，之后会采用多种模型的融和方式完成推荐过程。

很多地方做的还是显得很不成熟，之后的时间中也会继续修改。