中南民族大学

毕业论文(设计)

学院：计算机科学学院

专业：智能科学与技术　年级：2013

题目：在推荐系统中引入 Serendipity 的算法研究

学生姓名：戴井之　学号：201321094002

指导教师姓名：康怡琳　职称：讲师

2017 年 5 月

中南民族大学本科毕业论文（设计）原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　年　　月　　日

目　　录

在推荐系统中引入 Serendipity 的算法研究

摘要：现今广泛使用的推荐算法所推荐的项目，往往出自于一个相对固定和狭窄的特征集合，因此忽略了用户兴趣的改变和对新鲜感的诉求。为了让用户有更好的用户体验，推荐系统应该考虑在准确率之外更加重要的衡量尺度，比如说新颖度、惊喜度、覆盖率、有效性等。这篇论文的核心是如何将推荐系统结合 serendipity 这个尺度。Serendipity 的定义是在一种轻松愉快的氛围中偶然发现让自己意想不到的物品，因此算法中融合了商品的属性特征 genre，评分越高的 genre 往往有着越低的 serendipity。算法代码是用带有科学计算包 numpy 的 pyhon 语言写成，并且在 100K MovieLens 数据集上进行测试，可以鉴别出在利用协同过滤算法推荐给用户的项目中，serendipity 较高的项目，进而提升推荐系统的用户体验。

关键词：推荐系统；协同过滤；Serendipity

Research on Serendipity Algorithm in Recommendation System

Abstract：Nowadays, the items recommended by widely used recommendation algorithms often come from a relatively fixed and narrow set of features, thus ignoring the changes of user's interests and the demands of freshness. Therefore, recommendation system should consider other more important metrics outside of accuracy such as novelty, serendipity, coverage, usefulness in order to provide a better user experience. The core of this thesis is how to incorporate serendipity into a recommendation. Serendipity is defined as finding something good by accident in a relaxed and delighted atmosphere, hence the algorithm considers items' genre feature. Usually higher ratings correspond to lower serendipity. The algorithm implemented using Python in conjunction with the scientific computing package Numpy and tested on the 100K MovieLens dataset is shown to be capable of identifying more serendipitous items in the item set recommended to users by algorithm based on collaborative filtering, thereby improving user experience of the recommendation system.

Key words：Recommendation System；Collaborative Filtering；Serendipity

1 引言

传统的推荐系统的目的在于准确预测在用户还没有看过的物品中可能评分较高的物品，可是预测的准确性并不能完全反映推荐系统的好坏。随着经济发展，各种各样种类的商品出现在公众眼前，然而用户频繁接触的可能只是其中的一小部分，很多用户不常接触到但是有用的商品可能会给用户带来更多的惊喜感。因此近年来，推荐系统的研究核心已经转变为如何为用户推荐新颖度高而且有用的物品，在这篇论文中，我们会介绍在传统推荐算法中如何提升serendipity，从而让用户更加满意。

1.1 研究背景

随着互联网的普及和信息技术的发展，我们已经到了一个信息膨胀的时代。在网络上呈现给我们的信息越来越多，找到有价值的信息反而变得越来越困难。在这样的现实背景下，推荐系统的出现给社会带来了积极正面的影响。一个好的推荐让用户在不刻意寻找的情况下发现有价值的信息，从而节省了用户大量的时间。因此推荐系统在各行各业都得到了广泛应用，如何提升推荐系统的性能是专家学者亟待研究的问题。

1.2 现有成果

信息过滤是一种移除用户不想要的信息进而为用户呈现有用且相关的内容的技术，推荐系统采用这种技术来为目标用户提供个性化的信息。随着互联网的普及，在各行各业里推荐系统都获得了广泛应用。推荐系统的目标是自动为用户找到最符合心意的项目，例如书籍、音乐、电影等等，而推荐算法可能是依据用户画像或者用户对项目的评分。一些受欢迎的推荐系统有的是利用基于历史内容的推荐，有的是利用协同过滤算法，有的是利用用户行为数据，还有的是利用社交网络数据，当然方法不限于此。

基于内容的推荐算法是利用信息过滤与信息检索技术发现用户的历史喜好进而把物品推荐给用户，而协同过滤算法是利用用户之间或者项目之间的相似度得到喜好相似的用户和评价相似的项目进而进行推荐。这两种类型的推荐算法都要求收集用户喜好，而用户的喜好可以根据显示方法和隐式方法进行收集。显示方法是在用户主动参与的情况下进行的，例如用户的评分、收藏夹、购物车等，隐式信息是在用户不知觉情况下收集的，例如用户的浏览历史、页面停留时间等。然而这两种推荐算法也会有局限性，比如“冷启动”问题，对于一个没有历史数据的新加入的商品和用户，总是难以进行推荐。虽然冷启动问题是大多数推荐系统都会遇到的，但是有些技术可以有效缓解其带来的影响，例如基于知识发现的推荐和多种算法相融合的技术，根据物品的属性和用户的关系或者其他网站上的现有数据进行推荐，因此无需用到商品的评分。

用以上几种推荐算法都可以得到比较高的准确率，因为主要依靠物品或用户的相似度作为依据，但是推荐的物品种类和范围却很局限，不利于用户数的增长。因此，推荐系统还应该考虑到商品覆盖率、用户惊喜度、商品新颖度这些因素。Serendipity 被定义为在不刻意找寻的时候偶然发现令自己满意的而且有用的东西，可以作为用户惊喜度的衡量标准。近年来，怎样提升推荐系统的 serendipity 的问题得到很多学者的关注。

在信息检索领域，Campos 和 Figueiredo 最先设计了一个软件代理，通过网络爬行来支持意外信息的发现[1]。Iaquinta 及其团队首先将 serendipity 的概念引入推荐系统并使推荐性能得到最优化[2]，他们认为与用户资料相似度较低的项目可能是更 serendipitous 的项目，并修改了基于内容的系统，以便将这些项目也包含在内。Onuma 及团队提出了一种基于图的算法来支持 serendipity[3]，并介绍了在用户项目二分图中计算项目节点的“桥接分数”的想法。最近，Oku 和 Hattori 提出了一个系统，可以通过融合用户资料中的两个内容特征来引发可能的 serendipitous 的建议[4]。

2 相关技术简介

2.1 协同过滤算法

协同过滤、基于内容过滤是大部分推荐系统都采用的两种基本方法。协同过滤算法是一种用于推荐系统的比较成功的算法。一般来说，它主要由两种技术组成：基于相似度的算法和基于模型的算法。

2.1.1 基于相似度的算法

协同过滤基本组成成分：

（1）评分矩阵 R。其行标签为用户 id, 列标签为物品 id，矩阵内容为用户对物品的评分 rating。

（2）用户之间的相似度 sim(u,v)。通常使用的向量相似度测量标准有向量余弦夹角和皮尔逊相关系数，其他的测量标准有斯皮尔曼等级相关系数，平方差，熵等。

（3）利用相似度和评分来产生预测的方法。

评分矩阵表示如下：

	1	2	……	i	……	N_{items}	Average rating
1	3	4		4			$\bar{r}_1$
2		5		2		4	$\bar{r}_2$
v		$r_{v,i}$				3	$\bar{r}_v$
……							
u		$r_{u,i}$		3		2	$\bar{r}_u$
……							
N_{users}		3		4		5	$\bar{r}_{N_{users}}$

表 2-1 评分矩阵

针对某个用户 u 而言，要用 user-user CF 预测 u 对物品 i 的评分，要先在其他已经为物品 i 做出评价的用户中根据 sim(u,v)计算得出一个相似度高的邻域 N, N **错误!未找到引用源。**。根据相似度的大小进行排序，N 可以选自 top-N users, 也可以选自某一个范围以内。例如，我们可以划定一个阈值为 0.8，N 为由 sim(u,v) >= 0.8 的用户组成。一旦有了可以用来参考的邻域 N，可以根据公式（2-1）来计算：

错误!未找到引用源。 **错误!未找到引用源。** **错误!未找到引用源。** （2-1）

错误!未找到引用源。表示用户 k 对物品 i 的评分，sim(u,k)表示用户 u 和 k 的相似度。

虽然 user-user CF 算法被广泛使用，但是这种算法只适用于用户较少的情况下。因为当用户的评价改变时，与其他人的相似度就会改变，然而这种情况每时每刻都在发生着，为了减少计算，item-item CF 算法被 Linden et al 提出[5]。这个算法是依据项目的相似度进行计算而非用户相似度。在用户远多于物品的推荐系统中，即使用户对物品的评分频繁发生改变，对于物品的相似度来说只会产生微小的改变，因此用户仍旧可以得到较好的推荐。皮尔逊相关系数和余弦夹角依旧可以用于 item-item CF 中来测量物品的相似度。

2.1.2 基于模型的算法

基于模型的推荐算法有用关联算法做协同过滤、用聚类算法做协同过滤、用分类算法做协

同过滤、用回归算法做协同过滤、用矩阵分解做协同过滤、用神经网络做协同过滤、用图模型做协同过滤、用隐语义模型做协同过滤等算法。

2.2 相似度计算

相似度的计算是基于协同过滤算法的推荐系统十分重要的一个环节，包括用户与用户之间的相似度、物品与物品之间的相似度或者用户与物品的相关性[6]。

1）共同邻居。直接计算两个用户的交集，见公式（2-2）：

$$Similarity_{u,v} = \Gamma(u) \cap \Gamma(v) \tag{2-2}$$

2）Jaccard 系数。基于共同邻居的相似度计算的一个显著问题是用户的物品集越大，越可能与其他用户相似，但 Jaccard 系数可以消除这一影响，见公式（2-3）：

$$Similarity_{u,v} = \frac{\Gamma(u) \cap \Gamma(v)}{\Gamma(u) \cup \Gamma(v)} \tag{2-3}$$

3）皮尔逊相关系数。皮尔逊相关系数法是协同过滤算法中最常使用的相似度计算方法。研究表明，在计算向量相似度方面，皮尔逊相关系数法和其他相似度计算方法相比准确率更高。具体定义如下,见公式（2-4）：

错误!未找到引用源。 (2-4)

5）向量余弦法。向量余弦法计算相似度的数学描述如下，见公式（2-5）：

错误!未找到引用源。 (2-5)错误!未找到引用源。

在公式（2-4）和公式（2-5）中，**错误!未找到引用源。**表示用户 u 的评分向量，**错误!未找到引用源。错误!未找到引用源。**表示用户 v 的评分向量，**错误!未找到引用源。**表示用户 u 对所有项目评分的平均分，**错误!未找到引用源。**表示用户 v 对所有项目评分的平均分。

3 需求分析

在推荐系统领域，很多研究学者都开始研究 serendipity。为了发现这个概念的特征，Iaquinta 团队[7]分析了这个词语的语源。他们发现在推荐系统相关领域，serendipity 与推荐系统的质量息息相关。它是一个相对主观的概念，因此是一个难以研究的课题。

3.1 SERENDIPITY 的定义

在推荐系统领域，对于 serendipity 有多种定义。例如，在文献[8]中，serendipity 被定义为衡量推荐项目对用户有吸引力和令人惊讶的程度。这意味着一个非常偶然的建议将有助于用户找到一个令人惊讶和有趣的项目。 根据这个定义，我们可以看到与 serendipity 有关的两个重要方面。 首先，这个项目应该是用户未曾发现的并且意想不到的；其次，这个项目应该是和用户的兴趣相关并且有用的。 在其他的研究中 serendipity 也有类似的定义。例如，在文献[9]中认为 Serendipity 是衡量这些推荐令人惊喜程度的衡量标准。在文献[10]中，serendipity 被定义为接收到意外和偶然推荐的用户体验。

3.2 SERENDIPITY 能解决的问题

理想情况下，当我们在推荐系统中成功融入 serendipity 时，我们能够避免协同过滤系统中的过于“明显”的建议，解决基于内容的系统中的过度规范问题，在用户非主动参与的情况下

帮助用户发现意外的兴趣。 然而，由于从主观意义上判断 serendipity，可能推荐物品会被用户认为是不满意的或者是无用的，因此带来难以预料的风险。

3.3 SERENDIPITY 的衡量标准

怎样衡量一个推荐系统的 serendipity 是一个开放的问题。但是从我们判断一个系统 serendipity 的高低时可以推荐出衡量 serendipity 至少有以下三个方面：是否意想不到、有效性和新颖性。

3.3.1 意想不到的度量

意想不到的度量标准可以用 UNEXPECT 来表示。为了衡量 UNEXPET，我们需要一个用户意料之中的推荐项目的集合。因此我们用传统方法找到参与人数最多的项目集合和评分最高的项目集合，并合并成同一个集合，我们称其为 PM 模型。则 PM 是由原始预测模型生成的一组建议，RS 表示由推荐系统生成的建议。当 RS 的元素不属于 PM 时，我们认为元素是一个意想不到的建议因此，我们计算出意想不到的建议如下： UNEXPECT= RS \ PM

3.3.2 有效性

有效性可以用 USEFUL 来表示。为了保证推荐项目的质量，可以设定一个阈值 k, 假设评分在 k 以上的项目被认为是有效的，评分低于 k 的项目被认为是无效的。

3.3.3 新颖性

项目的新颖性可以用 NOVELTY 来表示。增加推荐项目的新颖性可以用 genre-based 算法来实现。根据 genre-based 算法得到的评分较低的项目是用户兴趣较低的项目，从而增加推荐的新颖性。例如，用 User-user 协同过滤算法推荐电影，如果我们发现用户喜欢动作电影也可能喜欢喜剧电影，我们可以通过添加喜剧电影来多样化推荐列表。但是这不是偶然的推荐，因为他可能会自己发现这部电影，所以对这个推荐不会感到惊喜。因此需要 genre-based 算法对用户可能感兴趣的电影类型进行排除，进而增加推荐内容的新颖性，提升用户体验。

4 详细设计和实现

这一章节的内容介绍如何在推荐系统中引进 serendipity 的算法的详细设计和实现，以及算法实现所依托的语言和环境。Serendipity 的定义是在不刻意寻找的时候发现令自己满意并且有用的物品。在推荐系统中，它被定义为一种能够为用户找到意想不到的并且有用的物品的方法。因此在本章中介绍了一种利用协同过滤技术并结合物品内容产生 serendipity items 的方法，基本假设是当基于物品内容的算法 genre-based algorithm 计算得分越高，则该物品对于用户的 serendipity 越小[11]。

4.1 算法流程

产生 serendipity items 算法流程，见图 4-1：

第一步：利用协同过滤技术得出要推荐的物品单 RS，以及对物品预测的评分**错误!未找到引用源。**。

第二步：利用物品的评分分数排序评分数量排序和得出用户意想之中的物品单 PM。

第三步：在 RS 中排除 PM 中的物品，可以得到用户意料之外的物品单 UNEXPECT。

第四步：在 UNEXPECT 中排除评分低于某个阈值的物品，可以得到更有用的物品单 USEFUL。

第五步：结合物品的内容提升 USEFUL 的新颖度进而得到最能体现 serendipity 的物品单**错误!未找到引用源。**。

图 4-1 算法流程图

4.1.1 利用协同过滤算法得出 RS

首先我们用 user-user 协同过滤算法初步得出要推荐的项目集合, 用 RS 表示，具体算法如下：

User-based recommendation method

```
1  function User-basedRecommender(User u)
2      for each 没有被用户评分的物品 i do
3          for each 对物品 i 有评分的用户 v do
4              compute u 和 v 的相似度 sim(u,v)
5              compute v 的加权移动平均值
6          end for
7      compute u 对物品的评分 p(u,i)
8      end for
9      return 推荐单 RS 和预测评分
```

4.1.2 原始推荐算法模型 PM

原始推荐方法所推荐的是综合用户评分最高的物品以及物品评分数量最多的物品集。因此 PM 中包含两种物品：一种是平均分高的，另一种是人气最旺的，这两种无疑都是用户意料之中的物品，因此没有推荐价值。

关键代码：

PM=np.unique.vstack((topN_users_for_items[‘itemid’],topN_ratings_for_items[‘itemid’])).ravel()

4.1.3 得出用户意想不到的物品集 UNEXPECT

Serendipity items 的关键组成之一就是高度的 unexpected，PM 模型中的内容无疑是高度 expected, 在确保准确的情况下给用户更多的惊喜，可以在 RS 中排除 PM。得出的集合可以用 UNEXPET 表示。

$$UNEXPECT = \{x \in RS \; and \; x \notin PM\}$$

4.1.4 利用有效性得出 USEFUL

Serendipity items 的另一个关键组成就是提供有用的物品，我们可以确立衡量物品有效性的阈值，在实验中我们可以把这个阈值确立为 3，评分大于等于 3 的物品我们可以认定为 USEFUL，进而保留在推荐集中。

$$USEFUL = \{x \in UNEXPECT \; and \; p_{u,x} \geq 3\}$$

4.1.5 结合物品内容得出 SERENDIP(t)

这一步是根据物品的内容和属性进一步提升推荐系统带给用户的惊喜度，例如在 MoveiLens 数据集中，电影的属性是一个重要衡量指标。利用 genre-based 算法，USEFUL 中的电影将被进一步评估，得出基于 genre-based 算法的评分p_t。最终得出的推荐单为 SERENDIP(t)，t 代表的是用户的兴趣度。

中心假设：在 USEFUL 中的物品，如果利用 genre-based 算法求得的评分越低，反而越能

够满足 serendipity 这个条件。如果用第一步 user-user 算法得出的评分**错误!未找到引用源。**-**错误!未找到引用源。**<-0.5, 则认为这个物品不够 serendipitous，因此排除开[12]。

$$SERENDIP(t) = \{x | x \in USEFUL \ and \ x \notin less \ serendipitous\}$$

Genre-based 算法流程如下：

Genre-based recommendation method
(1)计算每一个用户对每一个 genre 的平均分。 (2)根据第一步，计算用户对物品的平均分。 (3)根据 RS 中的评分和第二步的评分得出 serendipity items。

在 MovieLens 中，每一个电影的属性都被表示为一个 19*1 的相量，19 代表电影总共被分成 19 种 gnr。对于每一部电影来说，Attrib(item)**错误!未找到引用源。**(**错误!未找到引用源。**)，如果电影属于 gnr1, 则**错误!未找到引用源。**=1，否则为 0，一般来说每部电影都属于多个 gnr。我们让**错误!未找到引用源。**代表用户 u 所评价的物品 item 所对应 gnr 的值, 则**错误!未找到引用源。**（0，1）。令**错误!未找到引用源。**代表用户 u 所评价的电影中属于种类 gnr 的电影的数量，则用户 u 对种类 gnr 的平均分可以用公式（4-1）表示为：

$$r_u^{gnr} = \frac{\sum_{item=0}^{Nitems} \cdot A_{item,gnr}^{u}}{M_{u,gnr}} \tag{4-1}$$

如果没有电影属于 gnr, 那么这个 gnr 不考虑。

令**错误!未找到引用源。**表示电影 item 所属的种类个数，即属性向量中 gnr 值为 1 的个数，用用户评分为：

错误!未找到引用源。 = **错误!未找到引用源。** (4-2)

同样的，genre-based 算法中的预测评分用 user-user CF 中与用户 u 相似度较高的用户 k 的 sim(u,k)进行计算，原因一是为了避免利用 genre-base 算法造成用户评分值得空缺，原因二是结合第一步的协同过滤算法能够提高推荐系统的准确率。因此用 genre-based 算法最终得出的预测评分为：

错误!未找到引用源。 = **错误!未找到引用源。**，其中 k **错误!未找到引用源。**u (4-3)

举例说明：

假设共 5 部电影，电影的种类 gnr 为 4 种，某用户为其中 3 部电影 item1、item3、item5 进行评分，那么其中黄色区域为有效计算区域，**错误!未找到引用源。**代表用户 u 所评价的电影中属于 gnr 的个数，由此可以计算出用户对种类 gnr 的平均分。**错误!未找到引用源。**代表每一部电影的 gnr 个数，即每一纵列中 gnr 值为 1 的个数，用每一部电影 item 的 Attrib(item)向量和用户 u 对每种 gnr 的平均分**错误!未找到引用源。**的向量相乘，再除以**错误!未找到引用源。**，即可预测该用户 u 用 genre-based 算法对电影 item 的评分。（在下图 4-2 中，紫色部分为用户 u 对其没有评价的电影的评分。）

genre \ rating	item1	item2	item3	item4	item5	$M_{u,gnr}$	$\bar{r}_u^{gnr}$
	4		3		5		
gnr1	1	1	0	0	1	2	4.5
gnr2	0	0	1	1	0	1	3
gnr3	1	0	0	1	0	1	4
gnr4	0	0	1	0	1	2	4
$N_{item,gnr}$	2	1	2	2	2		
$r_{u,tiem}$	4.25	4.5	3.5	3.5	4.25		

图 4-2 举例说明图例

4.2 PYTHON 和 NUMPY

这个算法使用 Annaconda(python2.7 的发行版)和 numpy 写的，以下介绍选用这种方法进行科学计算的原因。

4.2.1 工具的选择

Python 是一种在科学领域被广泛使用的通用编程语言[13]，在 python 中可以调用以前的 C、Fortran 或者 R 代码。同时 Python 是比 C 和 Fortran 更加高级的面向对象语言，可以写出易读、整洁并且缺陷最少的代码。Numpy 系统是 Python 的一种开源的数值计算扩展，它提供了许多高级的数值编程工具如矩阵数据类型、矢量处理，以及精密的运算库，使用 numpy 就可以很自然地使用数组和矩阵。同时 Numpy 中包含很多实用的数学函数，涵盖了线性代数运算、傅里叶变换和随机生成等功能，可以方便地进行数据科学计算。

4.2.2 代码实现

例一：向量余弦法计算相似度

```
def cosine_similarity(ratings_matrix,avg_ratings_of_users,i,j):
    mask = np.logical_and((ratings_matrix[i,:] > 0),(ratings_matrix[j,:] >0))
    if np.sum(mask) == 0:
        return 0
    r_i = ratings_matrix[i]
    r_j=ratings_matrix[j]
    numerator = np.dot(r_i[mask],r_j[mask])
    denom = np.linalg.norm(r_i[mask])*np.linalg.norm(r_j[mask])
    if denom < 1e-6:
        return 0
return numerator/denom
```

例二：在 RS 中排除 PM 中出现的项目

```
mask = np.in1d(RS['itemid'][:],PPM,invert=True)
UNEXPEC = RS[mask]
```

5 测试与结果

在这篇论文中介绍的算法采用来自 GroupLens 研究中心在 movielens.org 网站上发布的 100K MovieLens 数据集进行检测。检测结果结果将在这一章中进行展示。

5.1 输入文件

在该 100K Movielens 中记录了 943 个用户，1682 部电影和 100000 个评分，其中每个用户至少对 20 部电影进行了评分。

（1）文件 u.info 保存了该数据集的概要：

943 users

1682 items

100000 ratings

（2）文件 u.item 保存了 1682 部电影信息，电影 id 依次是 1、2、3、……、1682。每一行中的电影信息用如下格式来表示。

格式：Item id | 电影名称 | 上映日期 | URL | gnr1 | gnr2 | …… | gnr19

举例：

1 | Toy Story (1995) | 01-Jan-1995 | http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)

|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0

说明：最后 19 个字段保存的是电影的类型，一个字段对应一种类型 genre(值 0 代表不属于该类型，值 1 代表属于该类型)，类型信息保存在文件 u.genre 中。为了方便计算，在 u.item 文件中我们只取 id 和 gnr 这两种信息。

（3）文件 u.user 保存了 943 个用户信息，用户 id 依次是 1、2、……、943。每一行中的用户信息用如下格式来表示。

格式：User id | 年龄 | 性别 | 年龄 | 邮编

举例：1 | 24 | M | technician | 85711

（4）文件 u.data 保存了 100000 个评分记录，评分的分值为 1、2、3、4、5。每一行中记录的信息用如下格式来表示。

格式：User id | item id | 评分 | 时间戳

举例：196　242　3　881250949

（5）经过处理后最终输入的文件及格式如下：

```
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
```

图 5-1 u.data 文件格式

	A	B	C	D	E
1	1	24	M	technician	85711
2	2	53	F	other	94043
3	3	23	M	writer	32067
4	4	24	M	technician	43537
5	5	33	F	other	15213
6	6	42	M	executive	98101
7	7	57	M	administra	91344
8	8	36	M	administra	5201

图 5-2 u_.csv 格式

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
4	4	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
5	5	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
6	6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
8	8	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0

图 5-3 u_item.csv 格式

(6)构造评分矩阵 R

用 u.data 中的数据构造一个矩阵，矩阵的横轴标签为用户 id，纵轴标签为电影 id，矩阵内容为用户对电影的评分。用来构造此矩阵的 python 代码如下：

```
#导入 u.data，用户评分信息
data_ratings = np.genfromtxt('u.data',delimiter='\t',
                                        dtype=[('userID',int),
                                                ('itemID',int),
                                                ('rating',float),
                                                ('timestamp',int)])
#导入 u.user，用户信息
data_user = np.genfromtxt('u_user.csv',delimiter=',',
                                        dtype=[('userID',int),
                                                ('age',int),
                                                ('gender','S1'),
                                                ('occupation','S20'),
                                                ('zipcode',int)])
#导入 u.item，电影信息
data_items_genre= = np.genfromtxt('u_item.csv',delimiter=',',
                                                dtype=[('itemID',int),
                                                        ('genre','19float')])
np.sort(data_items_genre,order='itemID')
nitems = data_items_genre.shape[0]
nusers = data_user.shape[0]
#构造评分矩阵  Nusers x Nitems
ratings_matrix = np.zeros((nusers,nitems))
for i in xrange(data_ratings.shape[0]):
    userid = data_ratings[i]['userID']
    itemid = data_ratings[i]['itemID']
    rating = data_ratings[i]['rating']
ratings_matrix[userid-1,itemid-1] = rating
```

那么 ratings_matrix 即为评分矩阵。

5.2 代码测试

所有代码以 python 中的 numpy 包进行科学计算。我们以 id = 5 的用户 user 为例进行测试。

5.2.1 初步预测

用 user-user 算法得出用户集合 RS 及用户电影评分**错误!未找到引用源。错误!未找到引用源。**，用 genre-based 算法得出用户集合 RS_using_genre 及用户电影评分**错误!未找到引用源。**。因为 RS 和 RS_using_genre 中都为用皮尔逊相关系数算出的相似度大于 0.8 的用户，所以这两个集合完全相同，只是用两种算法预测得到的电影评分不同。

代码：

```
for user in users:
    RS, RS_using_genre = recommendations_genre_based(
        ratings_matrix,
```

items_and_their_genre,
avg_ratings_of_users,
pearson_similarity_population,
similarity_threshold,
user-1
)

所得 RS 和 RS_using_genre 的结果如图 5-4、图 5-5：

```
RS
```

```
array([(513, 5.000000000000001), (1251, 5.0), (483, 5.0), (361, 5.0),
       (187, 5.0), (971, 4.999999999999999), (523, 4.999999999999999),
       (251, 4.999999999999999), (171, 4.999999999999999),
       (855, 4.695646009390878), (190, 4.695646009390878),
       (919, 4.6777736917665145), (923, 4.673236253377411),
       (22, 4.666666666666667), (484, 4.658957231893198),
       (136, 4.64216663082775), (150, 4.63029088129892),
       (86, 4.629496848015121), (639, 4.547094891713963),
       (212, 4.547094891713963), (170, 4.547094891713963),
       (285, 4.545147345711594), (875, 4.535898384862246),
```

图 5-4 RS 截图

```
RS_using_genre
```

```
array([(166, 4.582572250214158), (923, 4.427605869326466),
       (361, 4.4147727272727275), (190, 4.406818711464293),
       (171, 4.399273390031427), (971, 4.331616405875686),
       (114, 4.330096383048501), (329, 4.281689719575601),
       (1028, 4.273165385642577), (86, 4.2636746626054896),
       (1176, 4.258676582761251), (883, 4.25388198757764),
       (855, 4.250291585454675), (136, 4.249453680502516),
       (603, 4.222603926318785), (483, 4.206239950313614),
       (756, 4.206105952271197), (752, 4.19193184009879),
       (180, 4.164550553780762), (750, 4.158123019901986),
```

图 5-5 RS_using_genre 截图

5.2.2 得出 UNEXPECT

在 RS 中排除 PM 中出现的电影，得到用户意想不到的电影。见图 5-6。

代码：

```
mask = np.in1d(RS['itemid'][:],PPM,invert=True)
UNEXPECT = RS[mask]
```

```
UNEXPECT
```

```
array([(513, 5.000000000000001), (1251, 5.0), (361, 5.0), (187, 5.0),
       (971, 4.999999999999999), (523, 4.999999999999999),
       (251, 4.999999999999999), (855, 4.695646009390878),
       (919, 4.6777736917665145), (923, 4.673236253377411),
       (484, 4.658957231893198), (136, 4.64216663082775),
       (86, 4.629496848015121), (639, 4.547094891713963),
       (212, 4.547094891713963), (170, 4.547094891713963),
       (875, 4.535898384862246), (603, 4.535898384862246),
```

图 5-6 UNEXPET 截图

5.2.3 得出 USEFUL

进一步提升 UNEXPECT 的质量，在 UNEXPECT 中选出评分不小于 3 的电影。

代码：

```
mask = (UNEXPECT['rating'] > =3)
USEFUL = UNEXPECT[mask]
```

USEFUL 即为有效电影的集合。

```
USEFUL
```

```
array([(513, 5.000000000000001), (1251, 5.0), (361, 5.0), (187, 5.0),
       (971, 4.999999999999999), (523, 4.999999999999999),
       (251, 4.999999999999999), (855, 4.695646009390878),
       (919, 4.6777736917665145), (923, 4.673236253377411),
       (484, 4.658957231893198), (136, 4.64216663082775),
       (86, 4.629496848015121), (639, 4.547094891713963),
       (212, 4.547094891713963), (170, 4.547094891713963),
```

图 5-7 USEFUL 截图

5.2.4 得出 SERENDIP(t)

在利用 genre-based 算法得到的 RS_using_genre 中过滤出包含在 USEFUL 集合并且不属于 less serendipitous 中的用户，并按照 genre-based 算法得出的评分**错误!未找到引用源。**从小到大进行排序。最终得到的结果为融入 serendipity 得到的电影推荐结果。

```
USEFUL_GENRE
```

```
array([(604, 3.0436370701897673), (338, 3.0666666666666673),
       (307, 3.144645588723544), (539, 3.2767857142857144),
       (294, 3.2803110125353565), (249, 3.3062696259448856),
       (32, 3.310337989838503), (628, 3.3125425514674083),
       (879, 3.33476256972998), (875, 3.3561926442096937),
       (324, 3.4020472982152814), (260, 3.421955063708122),
       (289, 3.452205353158749), (864, 3.452272842727106),
       (319, 3.46106672367823), (513, 3.4646168748025956),
       (298, 3.464777241681033), (748, 3.4891468309486156),
```

图 5-8 USEFUL-GENRE 截图

代码：

```
indices = [i for i,itemid in enumerate(RS_using_genre['itemid'])
           if itemid in USEFUL['itemid']]
```

```
USEFUL_GENRE = np.sort(RS_using_genre[indices],order = 'rating')
ratings_difference = np.sort(USEFUL,order='itemid')['rating'] –
                                np.sort(USEFUL_GENRE,order='itemid')['rating']
mask = (ratings_difference >= -0.5)
USEFUL_GENRE = np.sort(USEFUL_GENRE[mask],order = ‘rating’)
```

5.3 结果测试

输出数据收录在 plot_new_serendipity.dat 文件中进行保存。为了证明结果的有效性，我们对比了 RS 和**错误!未找到引用源。错误!未找到引用源。**这两个集合的推荐效果。

5.3.1 GENRE 比较

所选的数据为对用户 user=5 的推荐结果。对比 RS 中排名前 20 名的电影的种类分布和**错误!未找到引用源。**中排名前 20 的电影的种类分布和 genre 的数量分布，结果如下：

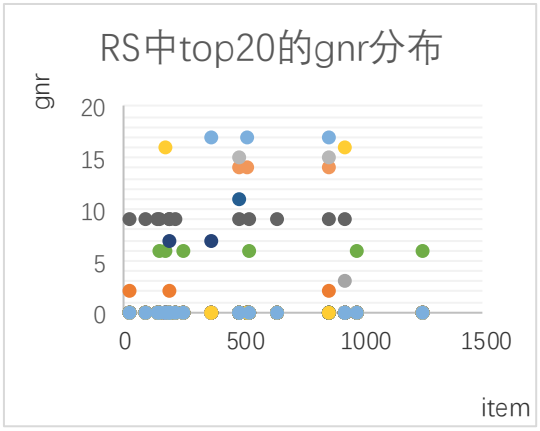

图 5-9 (a)RS 中 top20 的 gnr 分布

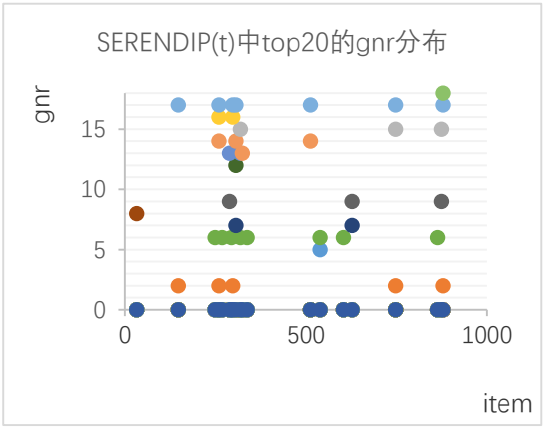

图 5-9 (b) SERENDIP(t)top20 的 genre 分布

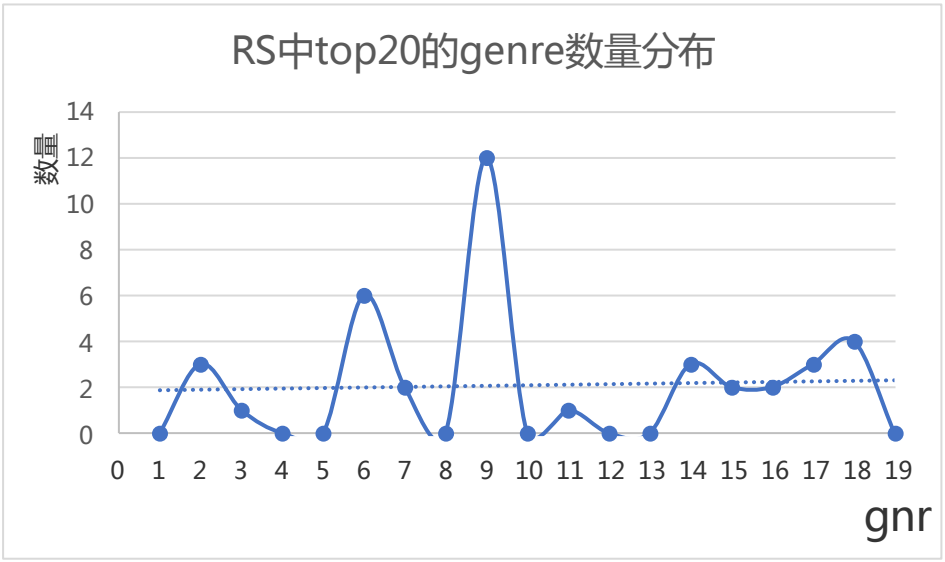

图 5-9 (c)RS 中 top20 的 gnr 数量分布

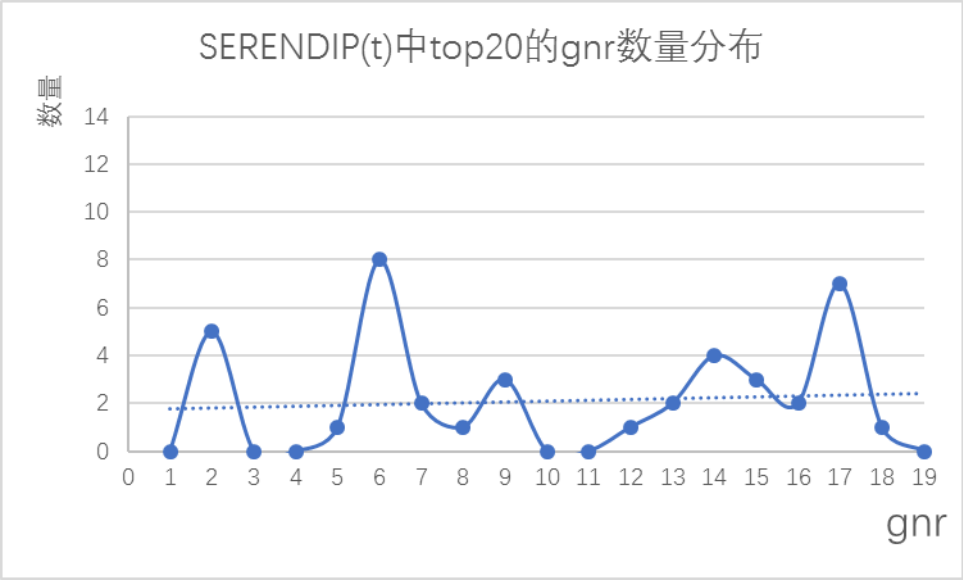

图 5-9 (d)SERENDIP(t)中 top20 的 gnr 数量分布

根据两种推荐方法的对比图，我们观察到：

（1）根据图 a 和图 b 中 item 和对应的 gnr 分布图，可以看出这两种方法推荐的电影 RS 和 SERENDIP(t)分布是比较均匀的。

（2）根据图 c 和图 d，观测到在 SERENDIP(t)中推荐的前 20 部电影中，gnr 的分布曲线与 RS 相比更加平缓，因此 SERENDIP(t)中类型 gnr 的分布更加均匀。

（3）在图 c 中 gnr=9 时出现峰值，由此推断出 user 偏向于观看 gnr 为 9 的影片，但是在图 d 中 gnr=9 时曲线较为平缓，可以推测出 SERENDIP(t)中减少了用户喜爱的电影类型的推荐。

5.3.2 用户测试

虽然用 genre-based 算法可以有效减少用户偏爱的类型的电影推荐，但是如果不询问用户的意见，这种惊喜度还是难以测量出来的[14]，为此在本次论证中我们找到 20 个志愿者进行评测。所得结果如表 5-1 所示。

具体步骤：

（1）志愿者被要求为在 u.item 中的不少于 20 部电影进行评分。

（2）把志愿者的评分结果加入评分矩阵 R。

（3）根据评分矩阵 R 分别得出用 user-user CF 算法和在此基础上融入 genre-based 算法之后得出的 RS 中 top20 的电影（记作 RS20）和 SERENDIP(t)中 top20(记作 SERENDIP(t)20)的电影。（4）让用户分别挑选出在 RS20 和 SERENDIP(t)20 中让自己满意的电影。

（5）计算用户满意的电影分别在 RS20 和 SERENDIP(t)20 中所占的比例。

结果：RS 中满意度的平均分为 50%，SERENDIP(t)中满意度的平均分为 52%。

可见 user-user CF 和 genre-based 相结合的算法可以提升用户的满意度。

表 5-1 用户满意度调查表

志愿者	RS	SERENDIP_t	志愿者	RS	SERENDIP_t
1	45%	45%	11	45%	45%
2	35%	35%	12	35%	45%
3	70%	75%	13	80%	70%
4	60%	55%	14	60%	55%
5	50%	65%	15	50%	50%
6	45%	70%	16	45%	50%
7	65%	60%	17	65%	60%
8	35%	40%	18	50%	40%
9	45%	40%	19	35%	45%
10	35%	40%	20	35%	60%

6 总结

6.1 整体评估

在本次研究中，我们提出了一种把 genre-based 算法融入 user-user 协同过滤算法中的方法，从而提高算法的 serendipity。这个算法使用 Annaconda 和 numpy 写的，Annaconda 是 python 的一个发行版本，算法的每一步运行的结果都可以非常清楚地展示在页面中。为了评估这个算法的有效性，我们用取自于 GroupLens 的 100K MovieLens 数据集进行测试。测试结果表明这种算法可以有效地减少用户偏爱的类型的电影推荐，从而提升用户的惊喜度和满意度，使用户体验更加轻松愉快。

6.2 存在的限制

因为在推荐系统中用到协同过滤算法，那么势必会有协同过滤算法遇到的一些限制[15]：

1、冷启动问题

（1）第一类是新用户问题，当协同过滤算法有新的用户出现时，由于没有新用户的任何历史行为数据，导致算法无法确定用户自身的信息需求，因而就无法完成推荐[16]。解决新用户问题有多种方法,例如根据用户的注册信息对用户分类并给用户推荐他所属分类中用户喜欢的物品，选择合适的物品启动用户的兴趣等。

（2）第二类是新项目问题，当协同过滤算法中有新的项目加入到系统中时，由于没有任何的评分信息，导致协同过滤算法在计算近邻集合的时候无法考虑到这些新项目，从而也导致了这些新的项目无法向用户进行推荐。但是新项目问题可以在基于内容的推荐算法得到很好的解决，因为基于内容推荐算法只需要考虑项目自身的内容信息即可完成推荐，因此我们可以将基于内容的推荐算法与协同过滤算法相结合[17]，不仅可以解决新项目问题，而且可以发挥协同过滤算法的挖掘用户潜在信息需求的能力。

2、评分的稀疏性问题

评分的稀疏性问题是指评分数据在整个系统中占有量非常少，冷启动问题就是评分稀疏性问题的极端例子。在推荐系统中，如果用户评价的项目相比于整个系统中的项目数量而言是非常少的，将会导致用户项目评分矩阵的稀疏。在数据极端稀疏的情况下，无论是基于用户的协同过滤算法还是基于项目的协同过滤算法所计算出的相似度的准确度都不高，因此无法作出准确可靠的推荐。一些学者对此进行了研究，文献[18]将奇异值分解技术应用到协同过滤中来，通过降低输入矩阵的维度降低数据的稀疏性。文献[19]提出了一种通过计算项目相似度填充用户评

分矩阵的方法，有效缓解了数据的稀疏性。

3、模糊用户问题

模糊用户问题又叫做“灰羊问题”[20]。在基于用户的协同过滤算法中，相似度计算是整个算法中非常重要的环节，然而，在系统中可能存在一些兴趣需求非常特殊的用户，很少甚至没有与其兴趣相似的用户，这将导致推荐变得十分困难。模糊用户问题是用户信息需求不明确造成的，这些问题可以通过结合基于内容的推荐算法到协同过滤算法那中来，从而增强需求信息的表达能力。

6.3 未来工作

关于 serendipity 推荐算法未来研究工作将集中在以下几个方面。一是当数据集更大时，由于编程语言与运行平台的限制，我们可能难以进行测试，因此需要云计算技术的支持。二是我们算法的测试是基于现有的数据后台测试，没有良好的交互界面，为了进一步提升交互体验，需要一个更加即时、便捷的交互平台。

致谢

我万分感谢我的导师康怡琳对我毕业设计热情、耐心而又周到的指导，为我论文最终的完成提供了非常多的帮助。当决定要做一份关于推荐系统的算法设计的研究时，有幸在康老师的指导下，在 couresa 上学习了一套关于推荐系统的课程，让我对推荐系统有了一个基本的了解，为后期的算法研究奠定下基础。在整个算法研究过程中，康老师为我提出很多有参考价值的建议，比如在推荐系统中引入 serendipity 这个指标，要选择一个恰当的方法检测推荐系统的好坏等。如果没有康老师的帮助，我在研究课题时不会这么顺利。

除了我的恩师，我也很感谢长久以来一直在我身边默默陪伴着我的同学，在我困难的时候给我出主意，和我一起经历着紧张、欢笑、泪水和幸福，我也很感谢长久以来默默关心着我的父母，因为他们的经济支持让我在学习上享受充分的自由。

参考文献

[1] J. Campos and A. D. de Figueiredo. Searching the unsearchable: Inducing serendipitous insights[C]. In *Procs. of the Workshop Program at the 4th International Conference on Case-Based Reasoning*, 2001:159–164.

[2] L. Iaquinta, M. de Gemmis, P. Lops, G. Semeraro, M. Filannino, and P. Molino. Introducing serendipity in a content-based recommender system[C]. In *Procs. Of the 8th International Conference on Hybrid Intelligent Systems*, 2008:168–173.

[3] K. Onuma, H. Tong, and C. Faloutsos. Tangent: a novel, ’surprise me’, recommendation algorithm[C]. In *Procs. of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pages 657–666.

[4] K. Oku and F. Hattori. Fusion-based recommender system for improving serendipity[C]. In *Procs. of the Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011)*, pages 19–25, 2011.

[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. 2013:4-6.

[6] 陶跃华. 基于向量的相似度计算方案[J]. 《云南师范大学学报》, 2001,21(5):17-19.

[7] Iaquinta L., Gemmis M., Lops P., Semeraro G. Introducing serendipity in a content-based recommender system[C]. Eighth International Conference on Hybrid Intelligent Systems, Barcelona, Spain. 2008:168-174.
[8] Herlocker J., Konstan J., Terveen L. and Riedl J. Evaluating collaborative filtering recommender systems[J]. ACM Transactions on Information Systems, 22(1), 2004:5–53.
[9] Shani G. and Gunawardana A. Evaluating Recommendation Systems[R]. Technical report, No. MSR-TR- 2009-159. 2009.
[10] Mcnee S., Riedl J and Konstan J. Accurate is not always good: How Accuracy Metrics have hurt Recommender Systems. 2006:1-2.
[11] 尤方圆. 电影推荐系统的设计与实现[D]. 华中科技大学. 2013:11-21.
[12] Suboojitha Sridharan. Introducing Serendipity In Recommender System Through Collaborative Methods[D]. 2014:14-15.
[13] Ivan Idris 著. Python 数据分析基础教程 Numpy 学习指南（第二版）[M]. 人民邮电出版社. 2014:1-6.
[14] 马咪咪. 基于用户测试的启发式评估研究[D]. 山东大学. 2013:13-15.
[15] 黄传飞. 基于项目的协同过滤算法的改进[D]. 江西师范大学. 2015:2.
[16] 徐登友. 协同过滤推荐系统中冷启动问题的研究[D]. 云南大学. 2016:42-46.
[17] 刘宇轩.混合协同过滤算法研究[D].北京邮电大学.2013:28-29.
[18] Sarwar BM, Karypis G, Konstan JA, Riedl J.Application of dimensionality reduction in recommendr system-A case study[C]. ACM WebKDD 2000 Workshop, 2000:82-90.
[19] 邓爱林，朱扬勇，施伯乐. 基于项目评分预测的协同过滤推荐算法[J]. 软件学报，2003，14（09）：1621-1628.
[20] 黎懋靓. 基于协同过滤和 QoS 预测的 Web 服务推荐方法研究[D]. 重庆大学. 2015:14-18.