网络出版时间：2017-02-20 15:06:09
网络出版地址：http://www.cnki.net/kcms/detail/11.2560.TP.20170220.1506.027.html

e 软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.005230]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

RepoLike: 基于多维特征的开源项目个性化推荐方法*

杨 程, 范 强, 王 涛, 尹 刚, 王怀民

(并行与分布处理国家重点实验室，国防科技大学 计算机学院,湖南 长沙 410073)

通讯作者: 杨程, E-mail: delpiero710@126.com

摘 要: 随着软件协同开发技术与社交网络的深度融合，社交化开发范式已成为当前软件创作与生产的重要方式。这一软件开发模型的灵活性与开放性，吸引了大规模的外围贡献者加入到开源社区中，形成了巨大的软件生产力。在开源社区中，这些分布广泛、规模巨大的外围贡献者主要以一种无组织的松散方式进行协同。他们需要花费大量的时间和精力，在海量的开源项目中寻找到自己真正感兴趣的项目并进行长期贡献。为了提高大规模群体协同的效率，本文提出一种基于多维特征的开源项目个性化推荐方法（即 RepoLike）。该方法从开源项目自身流行度、关联项目技术相关度以及大众贡献者之间的社交关联度等三个维度度量开发者和开源项目之间的关联关系，并利用线性组合和 Learning To Rank 方法构建推荐模型，从而为开发者提供个性化的项目推荐服务。通过大规模的实证实验表明，RepoLike 在推荐 20 个候选项目时的推荐命中率超过 25%，能够有效地为开发人员提供有价值的推荐服务。

关键词: 社交化编程；开源项目；个性化推荐；Learning to Rank；GitHub
中图法分类号: TP311

中文引用格式: 杨程, 范强, 王涛, 尹刚, 王怀民. RepoLike:基于多维特征的开源项目个性化推荐方法. 软件学报,2017, 28(6). http://www.jos.org.cn/1000-9825/5230.htm

英文引用格式: Cheng Y, Qiang F, Tao W, Gang Y, Huaimin Wang. RepoLike: A multi-feature based personal recommendation approach for open source project. Ruan Jian Xue Bao/Journal of Software, 2017,28(6) (in Chinese). http://www.jos.org.cn/1000-9825/5230.htm

RepoLike: A multi-feature based personal recommendation approach for open source project

YANG Cheng[1], FAN Qiang[1], WANG Tao[1], YIN Gang[1], WANG Huai-Min[1]

[1](Key Lab. of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China)

Abstract: With the deeply integration of software collaborative development and social networking, social coding represents a new style of software production and creation paradigm. Because of the flexibility and openness, a large number of external contributors are attracted to the open source communities. They are playing a significant role of open source development. However, the open source development online is a globalized and distributed cooperative work. If left unsupervised, the contribution process may result in inefficiency. It takes contributors a lot of time to find suitable projects or tasks to work on from thousands of open source projects in the communities. In this paper, we propose a new approach, called RepoLike, for recommending repositories to developers based on linear combination and learning to rank. It utilizes the project popularity, technical dependencies among projects and social connections among developers to measure the correlations between a developer and the given projects. The experiment results show that our approach can achieve over 25% of hit ratio when recommending 20 candidates, which means it can recommend closely correlated repositories to social

* 基金项目: 国家自然科学基金(61432020, 61472430, 61502512); 国家重点研发计划(2016YFB1000805)

Foundation item: National Natural Science Foundation of China (61432020, 61472430, 61502512); National key research and development program (2016YFB1000805)

收稿时间: 2016-10-17; 修改时间: 2016-10-26; 采用时间: 2016-12-22; jos 在线出版时间: 2017-02-20

developers.

Key words: Social Coding, Open Source Software, Personal Recommendation, GitHub.

近年来，社交媒体[1]与软件开发工具的深度融合[2, 3]，快速改变着互联网时代的软件开发形态。软件开发活动本质上开发者群体之间沟通交流、灵感激发的创作过程。社交媒体提供的诸如收藏（star）、关注（watch）、 @等机制，为大规模协同开发提供了极其便利且低成本的沟通与交流方式。因此，以 GitHub 为代表的互联网软件开发平台将关注点从软件项目转移到开发者身上，并提出了 fork 等新型协同开发机制，开创了全新的社交化编程理念（即 social coding）[4, 5]。此类社区中的贡献者可以通过社交工具随时收藏或关注自己感兴趣的开源项目，按需追踪代码高手的开发活动，自由评论他人贡献的代码。大量的软件用户或开源爱好者被吸引进来，以一种社交化、透明化的形式进行代码协作，形成了巨大的软件生产力。截至 2016 年 4 月，全球最大的开源社区 GitHub 中已经托管了超过 3,500 万个软件代码库，吸引了超过 1,400 万个大众贡献者参与到开源活动中。这些贡献者社区中产生了超过 1,200 万个代码合并请求[6]，大大推动了开源运动的持续快速发展。

然而，这种大众化的群体创作行为[7]通常以一种全开放的方式展开。大众贡献者分布在全球各地，以兴趣驱动的方式承担着各类开发任务，例如缺陷修复、代码测试和文档编写等。这些开发者拥有各种各样的性格特征、不同的教育背景和参差不齐的知识水平，其参与的开源项目往往与他们的兴趣和特长并不匹配。因此，如果不能恰当的引导这种大众化的社交化开发活动，实现大众智力资源和开源开发任务的优化匹配，整个开源开发生态的良性发展必定受到影响。在如今的开源社区中，一方面大众贡献者往往需要花费大量的时间和精力去寻找自己感兴趣的开源项目；另一方面开源项目中存在频繁的人员流失现象，大大增加了项目的人员培训和任务管理成本。

面对上述挑战，本文提出一种基于多维特征的开源项目个性化推荐方法（即 RepoLike）。该方法从开源项目自身流行度、项目间技术相关度以及开发者之间的社交关联度等三个方面量化潜在开发者和开源项目之间的关联关系，为大众贡献者提供个性化的项目推荐服务，从而提升大规模群体协同的效率。本文主要基于 GitHub 中的大规模开源资源和开发者数据开展研究，而在 GitHub 中同一个开源项目可能由于被其他开发者 fork 等而存在多个版本库，本文推荐的项目是指开源项目的根版本库。具体来说，本文的主要贡献包括：

1）从项目流行度、技术相关度和社交关联度三个维度，分析了大众贡献者在挑选开源项目时的关键因素，并设计了详细的量化度量方法；

2）基于上述度量提出了一种个性化推荐方法，基于线性组合和 LTR 构建个性化推荐模型；

3）构建了一个包含 16,249 个大众贡献者和 801,228 个项目库的大规模实验数据集，以定性和定量的方式验证了 RepoLike 的推荐效果。

本文的组织结构如下：第一章介绍了个性化推荐服务的研究动机；第二章详细阐述了本文提出的推荐方法框架及推荐算法；第三章为本文的实验设计与结果分析；相关工作和总结展望分别在第四章和第五章给出。

1 研究动机

在开源社区中，大众贡献者需要花费大量的时间和精力，在海量的开源项目列表中搜寻感兴趣的且合适的开源项目进行贡献。个性化的推荐系统可以大大缩减上述过程的时间成本。然而，为实现这一目标，我们需要解决以下两个问题。第一，数量庞大的开源项目质量参差不齐，如何高效地从中甄别出高质量的项目推荐给用户？第二，对于个性化推荐而言，在如何有效地度量大众贡献者个性化的兴趣点？在这一部分中，我们通过 GitHub 社区中的实际案例，来论证我们的研究动机。

在 GitHub 社区中，海量的开源项目之间、项目与开发者之间通过各式各样的方式相互关联，例如 watch 和 star（开发者和项目库之间）的关联关系，fork（项目库之间）的关联关系。这些潜在的关联数据为我们甄别高质量的项目版本库以及提供个性化推荐服务提供了重要的线索。例如，以 rails 为基础架构的软件项目中，

diaspora 和 paperclip 项目的代码质量和被关注的程度更高，如图 1 所示。因此，推荐系统如果从 rails 项目出发，更可能将上述两个项目推荐给开源贡献者。

图 1 Rails 家族项目的 fork 网络
Figure 1 The fork-network of Rails family

除了项目之间的关联，开发者之间通过社区提供的社交媒体展开广泛交流，其中蕴含的社交关联关系可以从另一个方面辅助我们实现个性化的推荐。如图 2 所示，开发者 Lars Kanis 与 Terence Lee 在社区中是好友关系。因此，他们之间的技术关注点或开发兴趣可能更为类似。如此一来，我们利用这种社交关联关系，分别为 Lars Kanish 和 Terence Lee 推荐软件项目，例如可以为 Lars Kanish 推荐 Terence Lee 目前正在参与的软件项目 jruby 等。

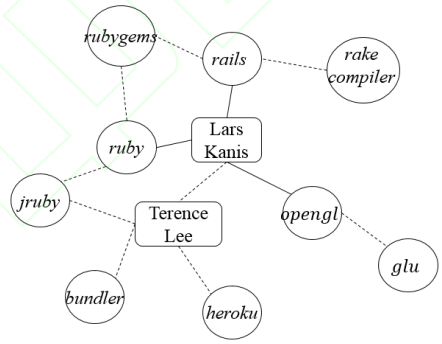

图 2 GitHub 中的开发者参与项目示意图
Figure 2 The context of GitHub developers

2 基于多维特征的个性化推荐方法

本节首先给出 RepoLike 的个性化推荐方法框架；然后介绍在个性化推荐中开发者与开源项目之间相关性度量的三个维度；在此基础上本节最后给出基于多维特征的个性化推荐算法。

2.1 推荐方法框架

与在线商业网站的推荐方法相比，由于大众贡献者参与项目的开发历史数据和开源项目本身的数据存在类型多样且高度分散的特点，开源项目个性化推荐面临的最大挑战在于如何准确度量大众贡献者的个人兴趣以及与开源项目之间的关系，从而对其未来可能参与的开源项目进行个性化的推荐。

本文方法的主要思想是充分利用大众贡献者在 GitHub 中的活动数据，从项目自身维度、项目之间的技术

依赖、开发者之间的社交关联等三个维度构建开发者与开源项目间的关联度量模型，从而进行个性化推荐。图 3 给出了 RepoLike 方法的总体框架，主要包括以下 5 个阶段：

1）阶段 1：开发者开发历史数据获取。在该阶段，我们首先整体获取 GitHub 的数据集，并将分散的开发历史数据以开发者为中心进行聚合。该阶段采集的开发历史数据，针对项目的包括 watch，issue，issue comment，pull request， pull request comment，fork，针对开发者的包括 follow 信息等。

2）阶段 2：关联网络的构建。我们从以项目为中心和以开发者为中心两个维度构建关联网络：以项目为中心，我们基于项目之间的技术依赖关系构建项目间技术关联网络；以开发者为中心，我们基于这些贡献者在 GitHub 社区中的社交关系构建开发者社交关联网络。项目间技术关联网络反映了项目之间的技术关联，一名开发者在某一个时间段内的技术关注点是有限的，通常会关注联系比较紧密的相关项目或者技术。利用项目技术关联网络，发现与开发者当前参与项目技术关联密切的项目，作为推荐的一个重要因素。同时，在 GitHub 中，开发者之间由于技术爱好或者关注点相似也会出现“人以群分”，技术兴趣偏好相似的开发者之间更可能发生密切的交互关联，因此这些密切关联的其他开发者所关注和参与的项目很有可能也是该开发者感兴趣的项目。基于兴趣相似性，利用开发者社交关联网络，将相关开发者作为项目个性化推荐的另一个重要因素。本文我们可以基于这两个网络分析筛选出用户可能感兴趣的项目。

3）阶段 3：推荐模型训练。推荐模型训练主要包括两个步骤：（a）训练数据排名标注；（b）训练数据特征提取。其中，排名标注是对每个开发者参与所有项目的根据参与深度和活跃度进行排名。因为开发者在不同项目中所投入的时间和精力是不一样的，这种不同差异反映了开发者技术偏好的不同，排名标注的项目将作为推荐模型训练的输入。特征提取主要包括训练数据中项目自身的项目流行度特征数据、基于项目关联网络的技术相关度特征数据以及基于开发者关联网络的社交关联度特征数据。具体的特征提取和度量方法见 2.2 节。在获得训练数据项目排序和多维特征之后，我们提出了一种个性化推荐方法，通过标记排名的数据集来训练特征集与标记值之间的关系，分别采用线性组合和 LTR 的方法，优化特征之间的参数并建立推荐模型。

4）阶段 4：推荐测试。在完成个性化推荐模型后，我们基于测试数据集对给定的开发者推荐可能感兴趣的开源项目，并将排名 Top-N 的项目返回给用户。

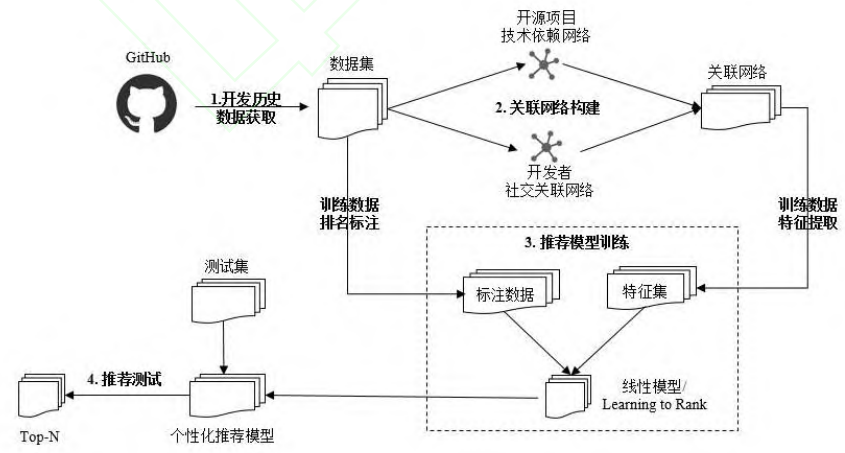

Figure 3 Overview of the personal recommendation framework

图 3　个性化推荐框架

2.2 特征提取与量化度量

本文我们从三个不同的维度对大众贡献者的技术偏好以及项目关系进行度量，这三个维度包括项目流行

度、技术相关度和社交关联度。这三个维度分别从项目自身、关联项目以及开发者之间的关联等方面全面地反映开发者和与被推荐项目之间的关系。具体各个维度的特征提取与量化度量如下。

2.2.1 项目流行度

有学者研究表明，一个大众贡献者在选择是否加入某个开源项目或是否会成为长期贡献者的一个关键因素是项目本身是否活跃[8]。开发者更可能加入处于活跃状态的、在开源社区中受到广泛关注的开源项目，因为这些项目可能更具有发展潜力和发展前景，而不是那些已经不活跃或者只有较少人感兴趣的项目。因此，项目流行度这个维度主要考虑项目自身的活跃情况以及大众贡献者对项目的关注情况。

GitHub 社区提供了 watch 和 fork 机制，一个大众贡献者如果对某个开源项目感兴趣就可以利用 watch 和 fork 机制来订阅这个开源项目，从而实现对该项目的持续跟踪和本地开发。因此，watch 数和 fork 数是反映一个给定的项目是否受到开发者关注的因素。具体的，我们基于一个项目的 watch 数和 fork 数来对项目的流行度进行度量。

Watch 数：关注一个开源项目最简单的办法就是订阅感兴趣项目的信息。大众贡献者可以通过点击项目的“Watch”按钮，就可以从 GitHub 的信息推送系统中获取项目的最新实时信息。因此，项目的 Watch 数量可以反映大众贡献者对项目的关注程度。

Fork 数：Fork 是 Git 提供的一个功能。如果一个用户感兴趣一个项目，并且希望加入它的开发，那么他可以将这个项目 fork 到本地分支，获得源代码并进一步研究或就地进行贡献。因此，一个项目获得的 fork 数越多则表明该项目越受欢迎。

为量化度量一个开源项目的流行度，我们基于 watch 数和 fork 数这两个指标。我们设计了如下公式来计算一个开源项目的流行度：

$$pp(j) = nor(watch(j)) + nor(fork(j))$$

其中，$pp(j)$ 表示项目 j 的流行度，$watch(j)$ 和 $fork(j)$ 分别表示开源项目 j 对应的 watch 数量和 fork 数量。因为 watch 和 fork 是两个不同层面的关注，fork 相对于 watch 而言大众贡献者参与更为深入。因此，为了更准确地度量和比较，我们对 watch 和 fork 数据进行归一化处理后进行加和操作。

2.2.2 技术相关度

软件复用是软件开发活动中常见的开发行为，大多数软件都或多或少地依赖于一些其他的项目或构件，这在无形之中引入了一种项目之间的依赖关系。这种技术依赖关系形成了一个软件生态系统，反映了项目之间的技术相关度。

通常情况下，如果一个开发者正在参与项目 A，另外一些项目与 A 之间存在紧密的技术相关性，那么他将更有可能对这些技术相关的项目感兴趣。基于这种直觉，我们的思路是通过构建项目之间的技术依赖网络，并在此基础上提出技术相关度度量方法，从而实现对技术相关度的量化度量。

Blincoe 等研究学者[9]对 GitHub 中项目间的技术依赖进行了深入研究。他们通过分析项目 issue、pull-request、评论、提交信息等文本数据，通过分析其中存在的指向其他开源项目的链接，将其作为项目间技术依赖的证据。本文基于 Blincoe 等提供的研究数据构建了一个项目依赖网络。然后，我们提出了一种开发人员参与项目和推荐候选项目之间技术相关度的度量方法。在此基础上，我们就可以量化度量开发人员和候选项目之间的相关关系。具体方法如下：

首先，我们根据开源项目间技术依赖关系构建了一个开源项目依赖网络。在该网络中，一个节点是一个开源项目，每一条边表示连接的两个项目之间存在技术依赖，边的权重表示两个项目间的技术依赖的程度，其取值为两个项目间存在的链接数。

然后，在项目依赖网络的基础上，我们利用开发者参与项目情况，构建一个面向该开发者的项目依赖子网络。在该子网络中，以该开发者为中心，与该开发者直接连接的是其参与的所有开源项目；与其参与开源

项目连接的是所有与这些项目存在技术依赖的其他项目。图 4 给出了一个示意图，其中开发者直接参与了 repo1, repo2, repo3 和 repo4 四个项目。根据项目依赖网络，我们发现另外一个开源项目 $repo_a$ 与 repo1 和 repo2 之间都存在技术依赖，且两两之间的链接数分别为 2 和 5，其他技术依赖项目类似。

最后，基于该面向用户的技术依赖子网络，我们设计了一个开发者与推荐候选项目之间的技术相关度度量公式。我们将开发者与某候选项目之间的技术相关度定义为开发者已参与的所有项目与该候选项目之间的技术相关度的加和，其中两个项目之间的技术相关度为项目依赖网络中边的权重。基于该公式，利用给定开发者参与的项目就可以计算出该开发者与候选项目之间的技术相关度。具体的计算公式如下：

$$tr(a,j)=\sum_{i} n_{ref}(i,j)$$

其中，$tr(a,j)$ 表示给定开发者与项目 j 之间的技术相关度，$n_{ref}(i,j)$ 表示项目 i 与项目 j 之间的技术相关度，其中项目 i 为开发者 a 参与的所有项目。

具体的计算，以图 4 中开发者与 $repo_a$ 之间的技术相关度计算为例，因为该开发者参与了 1 和 2 两个项目，这两个项目与项目 a 之间的技术相关度分别为 2 和 5，因此该开发者与项目 a 之间的技术相关度就为 7。

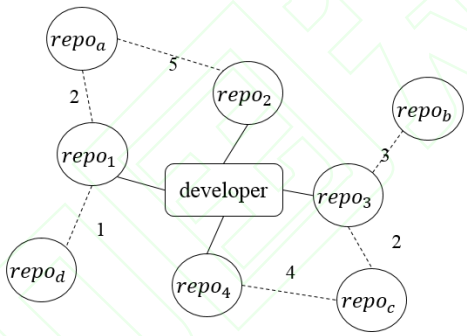

图 4 面向开发者的项目依赖子网络示意图

Figure 4 An example of developer-oriented dependence-sub-network

2.2.3 社交关联度

作为社交化编程网站，GitHub 提供了包括@、watch、评论等社交机制来加快开发者之间的交互，提升开发者群体间的透明度。利用这些服务，开发人员可以轻松高效地与其他开发人员建立社交关系，形成相互影响。一个开发者在 GitHub 中的社交关系反映出了该开发者的个人技术兴趣。在 GitHub 上，最常见的有深度的社交活动，是开发者对他人发布的 issue 和 pull-request 的评论。因此，在本文中我们专注于基于评论的社交关系分析。

Issue 评论：Issue Tracker System 是在 GitHub 当中的一个管理系统，大众贡献者使用该系统来来讨论项目存在的缺陷或者提出新的功能需求，包括对 issue 细节理解的讨论，或者提供解决方法或建议。

Pull-request 评论：pull-request 是一个 GitHub 进行贡献汇聚的有效方法，它为非核心开发人员参与项目开发提供了分支版本库管理和贡献汇聚的机制。但是大众贡献者群体提交的 pull-request 的质量差异往往很大。因此，在一个 Pull-Request 被合并前往往会被大量相关大众贡献者进行深度的讨论和测试。

对于大众贡献者来说，他们提交或者发表评论的那些 issue 或 pull-request 通常是他们感兴趣或者比较熟悉的领域，不同开发者参与同一个 issue 或者 pull-request 表明他们之间有着共同的兴趣或者技术偏好。因此，评论活动可以反映用户之间技术兴趣的相似性。开发者越相似，则他们就越可能对同一个项目感兴趣。本文我们基于大众贡献者参与 issue 和 pull-request 的情况建立开发者之间的社交网络，然后基于这个社交网络度量开发者之间的社交关联度。

该社交网络反映了开发者之间的基于评论形成的社交关系。其中，每个节点是一个开发者，节点之间的边是开发者之间在同一个 issue 或者 pull-request 评论中同现的次数。同现次数越多，表明这两个开发者之间的兴趣偏好越相似。

在该社交网络的基础上，我们基于开发者之间的社交相关度来度量一个开发者与特定项目之间的社交关联度。我们设计了一个方法来对这个关联度进行计算，具体格式如下：

$$sr(a,j)=\sum_{b} n_{\text{int}inacy}(a,b)*n_{participation}(b,j)$$

其中，$n_{\text{int}inacy}(a,b)$ 表示开发者 a 与 b 之间的社交关联度，$n_{participation}(b,j)$ 表示开发者 b 参与项目 j 的深度，$sr(a,j)$ 表示开发者 a 与项目 j 之间的社交关联度，b 是与开发者 a 具有社交关联的所有其他开发者集合。

具体计算方法我们通过一个示意图 5 来进行说明。在该图中，圆角矩形代表一个开发者，圆圈代表一个开源项目。开发者之间的边表示两者之间存在关系，边的权重就是在 issue 或 pull-request 参与中同现的次数。开发者和项目之间的边表示该开发者参与了该项目，边的权重表明该开发者参与该项目的深度，该值是该开发者参与该项目的所有活动的计数，具体包括提交 issue 和 pull-request 的数量，以及参与 issue 和 pull-request 评论的数量。

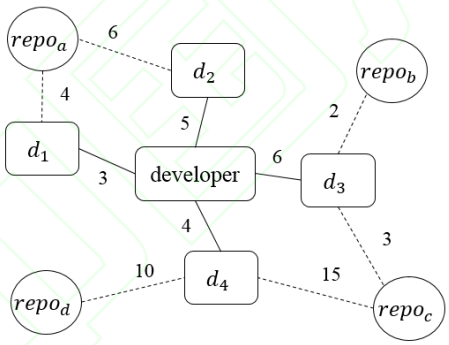

图 5 开发者与项目间的社交关联

Figure 5 Social connections between developers and repositories

在该图中，该开发者与另外 4 个开发者 d1，d2，d3 和 d4 都在某些项目的 issue 和 pull-request 中同现了。同时，d1 和 d2 都参与了 repo_a 这个项目。那么，该 developer 与 repo_a 之间的社交关联度为 42。该值从社交层面上开发者与推荐候选项目之间的距离。

2.3 基于多维特征的推荐算法

项目流行度、技术相关度和社交关联度从不同的维度反映了一个开发者与推荐候选项目之间的相关性，如何将这三个维度有机结合起来，从而全面准确反映开发者与项目之间的相关性是个性化推荐能否准确的关键。本文提出了两种不同的方法。

2.3.1 基于线性组合的推荐方法

我们将三个维度的因素进行线性综合，提出了一个相关度计算方法，具体公式如下：

$$score(a,j)=\alpha*nor(pp(j))+\beta*nor(tr(a,j))+\gamma*nor(sr(a,j))$$

其中，

$$nor(x) = x / \max$$

归一化的过程就是将每个特征值与该特征最大值的比值，通过归一化使得各个特征在相同的数量级下进行操作，便于参数的调节。

在该公式中我们综合了三个不同的维度。其中，$pp(j)$表示项目流行度，$tr(a,j)$表示技术相关度，$sr(a,j)$表示社交关联度。我们首先对三个维度的值进行归一化处理，然后进行加权求和，我们通过手动调节参数，然后通过结果反馈来调整参数，使参数达到最优，最终我们可以得到一个候选项目 j 与开发者 a 之间相关度的得分。在此基础上，我们对所有的候选推荐项目进行排名，根据分数将 Top-N 的候选项目推荐给开发者。

2.3.2 基于 Learning to Rank 的推荐方法

LTR 是利用机器学习的方法，基于给定的特征集合，利用训练数据集构建模型，学习最终的排序函数。在软件工程领域，已有相关工作基于 LTR 方法来对软件数据进行分析和排序[10,11]。本文通过对用户的历史行为数据进行分析，抽取多维特征，并构建用户参与项目的目标排序，基于该项目集，进行带监督学习，建立最终的排序模型。

筛选推荐项目集：通过为指定用户建立项目依赖网络，能够得到与用户参与的项目有依赖的项目；通过建立社交网络，能够得到与用户有交流的用户所参与的项目。在此基础上，我们可以选出用户还没有参加的项目，这些项目将作为推荐的候选项目，而用户实际参与的项目则作为 LTR 的训练数据集。

标记排名：排名是 LTR 方法学习的目标值，反映了数据记录的重要程度，越重要的记录排名越靠前。在用户的历史数据中，用户越重视一个项目，他在项目中所进行的活动应该越多，所以本文选取用户实际参与项目的频率作(即单位时间内参与项目的次数)为目标值，作为数据记录的标记值，用于最终的 LTR 模型训练和测试。

模型训练：通过对项目集进行特征提取和排名标记，我们能够获得用于 LTR 训练的数据集。在数据集中，每条记录反映一个用户与一个目标待推荐项目之间的关系，特征值即为用户与项目之间关于三个维度的特征，目标值即为标记排名过程中所标记的用户参与项目的频率。模型训练过程中，利用用户历史行为数据提取到的特征来训练 LTR 模型，让模型去拟合数据的标记值，即用户实际参与项目的频率；通过调整模型的结构和参数，寻找使得损失函数（如 NDCG，交叉熵等）最小的模型，用于预测用户的参与项目的频率；通过对频率进行排序，将高频率的项目排在前面，最终得到为用户推荐项目的排序结果。最终，基于该数据，我们可以训练出一个基于 LTR 的推荐排序模型。具体的，我们在建立 LTR 模型时，使用 RankLib 包来实现。

3 实验结果与分析

3.1 实验数据集

本文使用的数据集与前期工作相同[12]，主要使用了 GHTorrent (2016/6/3)提供的公开数据集。在实验过程中，我们希望筛选出比较有代表性的开发者作为实验样本，利用这些开发者的历史数据进行分析，并通过他们的实际行为记录来评估我们的推荐方法。

在进行开发者筛选时，我们主要遵循以下规则：1）在 GitHub 上活跃的一定的时间，并且现在还活跃在 GitHub 社区中；2）曾经对超过 1 个项目提供过贡献；3）与 GitHub 社区中的其他开发者进行过交流。根据以上原则，我们最终筛选出 16,249 个开发者，这些被选中的开发者是项目的候选推荐人。这些开发者至少参

与过 5 个开源项目，至多参加过 20 个开源项目，并且 follow 过 5 到 50 个其他开发者。通过对 16,249 个开发者行为信息进行提取，主要包括 issue，issue 评论，pull request，pull request 评论，以及项目之间的 fork 关系，用户与项目之间的 watch 关系，用户之间的 follow 关系等。数据集中的行为信息涉及 144,543 个开发者，这些开发者是与筛选出的候选推荐人有交集的开发者，并包含了 170,5841 个相关的开源项目。在最终实验使用的数据集中，我们对项目进行了简单的筛选，选出至少获得一个 fork 或者 watch 的项目，最终筛选出 80,1228 个开源项目。在构建项目之间的依赖网络时，我们使用的文献[9]中的项目间技术依赖数据，他们通过对 issue 和 Pull request 信息中的链接进行分析来获取项目之间的引用关系的。

在此基础上，我们将完整的数据集划分为两个部分，其中一个部分是作为我们的训练集来训练推荐模型，而另一个是用来测试以评价本文所提方法的推荐结果。

3.2 评价指标

为了验证本文方法的有效性，我们设计了一套科学的评价指标来。在本文中，我们根据活动的创建时间对数据集进行分组：将开发者最近一年参与的项目作为预测目标，希望这些项目尽可能多的出现在我们的推荐列表中；从一年之前的历史行为信息中提取用户行为特征，用来构建我们的推荐模型。评估过程中通过对特征信息和预测数据集的时间跨度进行控制，来获取使用不同信息时对推荐模型的影响。在特征选取上，我们分别对单个特征以及不同特征之间的组合进行评估，来验证特征的有效性以及选出效果最佳的特征集合。在进行 LTR 模型训练时，我们采用交叉验证的方法，即对数据集划分为 N 等份，每次选取一组数据作为测试集，其他组作为训练集，共训练 N 次，最终取 N 次测试的平均值作为模型的评价。

考虑到实际情况的特殊性，开发者实际参与一个项目的成本很高，需要花费较多的时间和精力，这也导致在一段时间内，开发者实际参与的项目数量不会很多。这种情况使得评估实验结果时，使用传统评估推荐模型的方法（如准确率，召回率，F 值等）难以获得较为理想的结果，同时无法反应推荐方法能够适用的开发者数量。以我们的数据集分析结果为例，开发者每年加入的版本库的平均值为 19.5，但我们需要从 801，228 个开源项目中选出用户实际关心的开源项目，通过社交网络和项目依赖网络的限制，我们最终为每个用户筛选出的项目平均有 337.8 个，远远大于用户的实际参加项目数量。

为了更好的反应推荐结果能够适用的开发者数量，在评估我们的实验结果时，我们主要使用命中率作为评价标准，通过限定推荐项目的数量（即 top-5, top-10, top- 15 和 top-20）来获取推荐不同数量的项目能够达到的推荐效果。其中命中率表示在所有开发者中，实际参与了我们所推荐的项目的开发者所占的比例，命中率的定义如下面公式所示：

$$ratio_{hit} = n_{hit} / n_{total}$$

其中，n_{total} 表示参与测试的开发者总数，n_{hit} 表示在所有参与测试的开发者中，有多少个开发者实际参与了至少一个推荐候选中的项目，这其中我们认为参与过项目 issue 或者 pull-request，关注了项目或成为项目的核心便是参与项目。

同时，作为推荐，推荐结果的排序质量是反映推荐结果好坏的重要指标。为了进一步评价推荐结果的排序质量，评估是否将更相关的项目排在前面，我们在进行 LTR 方法和线性方法的对比时也使用了 MRR（mean reciprocal rank）作为我们的评价指标。MRR 是一个通用的排序算法评价指标，能够反应目标结果所在的排序位置。其定义如下：

$$MRR = \frac{1}{|Q|}\sum_{i}^{|Q|}\frac{1}{rank_i}$$

其中$|Q|$为所有命中的结果，$rank_i$为命中结果i在推荐结果中的排名。

3.3 实验结果分析

在评价我们推荐方法结果的过程中，我们首先使用一个时间维度比较，以性能随着时间的推移和讨论评估我们的方法。然后，我们准备单维度和多维度的一系列比较，从三个维度选取，即流行维度为 PL，依赖维度为 TL 和依赖维度为 SL，探讨如何提高版本库推荐的性能。

本节分别介绍了基于线性模型的推荐和基于 LTR 模型的推荐结果。我们通过线性模型的结果来说明利用开发者的历史信息为开发者推荐软件项目的可行性，并通过对使用特征维度和时间的控制进行了对比实验。

3.3.1 基于线性模型的推荐

（1） 不同维度对推荐结果的影响分析

在进行不同维度的影响分析实验中，我们主要从三个维度对推荐模型的影响进行分析，即项目流行度信息（PL），项目技术相关度信息（TL）和开发者社交关联度信息（SL），进而探讨如何提高模型的性能。

首先，我们对比分析了只采用其中单一维度的信息时的推荐结果。

为了验证我们选取的特征的有效性，我们针对不同特征单独使用来建立模型。在构建模型过程中，因为只有社交网络和项目依赖网络能够限定项目选取范围，所以我们只针对 TL 和 SL 信息进行模型评估。结果如下表所示，其中 TL 是项目依赖信息，SL 是社交网络信息。

表 1 单一维度推荐结果比较

Table 1 Comparison of recommendation performance based on single dimension

	Top-5	Top-10	Top-15	Top-20
TL	6.04%	8.52%	10.45%	11.77%
SL	6.24%	8.47%	10.40%	11.60%

如表所示，单一维度的实验中 TL 和 SL 的表现相当，在 Top-20 推荐取得了最优的结果，分别达到了 11.77%和 11.6%。而 Top-5 的推荐结果将会降到 6.04%和 6.24%。结果表明单独使用 TL 和 SL 建立推荐模型都能够起到一定的效果，但是效果一般，还有很大的提升空间。

然后，我们对比分析了多维度信息组合利用时的推荐结果，这里我们采用多维信息线性组合的方法。

为了讨论不同特征组合对模型结果的影响，我们将三个维度（PL，TL，SL）中任意两个组合，获得推荐模型的结果与单一维度模型的结果比较如图 6 所示：

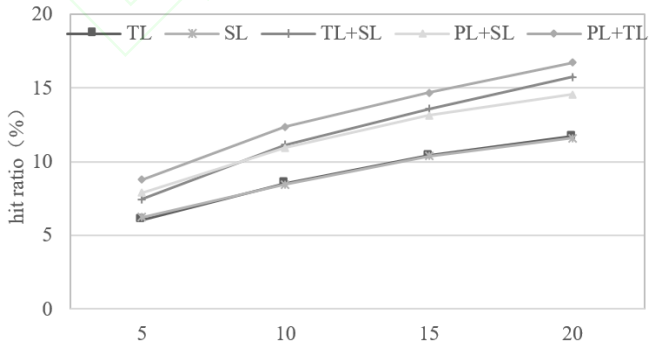

图 6 多维信息组合下的推荐结果比较

Figure 6 Comparison of recommendation performance for combination of two-dimension information

如图 6 所示，所有双维度组合推荐模型的结果都好于单一维度模型，其中 PL+TL 组合效果好于 TL+SL 和 PL+SL 组合效果，在 Top-20 推荐中命中率达到 16.7%，相比于单维度模型的最好结果提升了 4.93%。

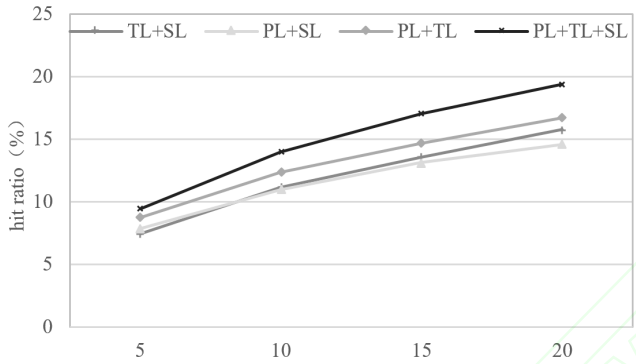

图 7 多维信息综合下的推荐结果比较

Figure 7 Comparison of recommendation performance for combination of multi-dimension information

图 7 中展示的是三个维度共同构建的推荐模型与两个维度组合的结果比较。从图中我们发现三个维度组合的结果好于两个维度的结果，在推荐 Top-20 中命中率达到 19.38%，接近单维度结果的两倍，而 Top-5 的推荐命中率也达到了 9.46%，接近单维度的 Top-20 推荐命中率。对比结果说明在建立推荐模型时，当我们能够取得的特征越多，对于开发者的个性化推荐结果越好。

（2） 测试时间长度对推荐结果的影响分析

对于开发者而言，参与一个开源项目通常意味着大量的时间和精力投入。因此，一方面，开发者在选择加入某个开源项目时会比较慎重；另一方面，在特定时间内，开发者参与贡献的开源项目是有限的。所以，在给定的长度时间间隔内，由于开发者精力有限，实际所参与的开源项目的数量是有限的。但是，这并不是说开发者对其他开源项目不感兴趣，也并不表明推荐结果不准确，而可能是由于没有推荐所以开发者没有注意到相应的项目，或者是由于精力限制暂时没有参与，如果将时间长度放大，则该开发者仍然可能参与这项项目。这一小节我们通过设定不同的测试时间间隔，分析时间长度间隔对推荐结果的影响。

本节，我们将开发者在 GitHub 中的活动数据截取三个不同的时间窗作为测试数据集，包括 3 个月、6 个月和 12 个月的活动信息，并将这些时间窗中该开发者实际参与的项目作为项目推荐是否准确的判定标准。然后采用基于线性模型的多维因素综合方法，分别进行测试比较。图 8 给出了具体的实验结果。

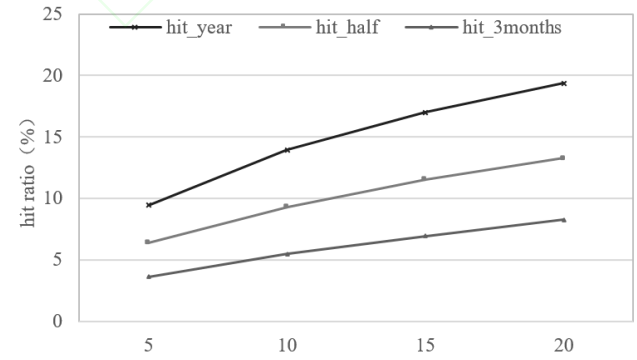

图 8 不同时间窗对推荐准确度的影响

Figure 8 The influence of different snapshots on the hit ratio

如图 6 所示，从推荐数量的维度看，在分别推荐 Top-5，Top-10，Top-15 和 Top-20 个项目时，不论是 3 个月、6 个月还是 12 个月，推荐准确度都是呈现上升趋势的。这是由于推荐候选越多，能命中开发者实际参

与的某个项目的可能性越大。

从测试时间窗的维度看，在推荐候选数量相同的情况下，我们对比分析不同测试时间窗下的推荐准确度。我们发现，不论是推荐多少个候选项目，测试时间窗越长，推荐命中率越高。从图中可以看出，从开发者实际参与的标准来看，很多推荐项目虽然开发者在较短时间内（如 3 个月内）没有参与，但当从更长的时间内看该开发者实际上参与了更多的推荐项目的。说明只要开发者有足够的时间和精力，他们可能会实际参与更多本文推荐的项目。因此，本文以实际参与项目作为推荐准确度的评判标准过于严格，实际的推荐效果严格比本文实验结果更好。另一方面，也说明开发者在搜索查找自己感兴趣的开源项目的过程是一个非常耗时的过程，个性化推荐将加速这一过程，推动开源软件的发展。

3.3.2 基于 LTR 模型的推荐结果

本节对基于 LTR 模型的推荐结果与基于线性组合模型的推荐结果进行比较，其中命中率和 MRR 的比较结果如下表 2 所示。

从表 2 可以看出，从 Top-5 到 Top-20，在 hit ratio 指标下，使用 LTR 的结果都要好于手动调优的线性模型。在提升幅度上，在 Top-5 和 Top-10 推荐上获得了较大的提升，但是在 Top-20 上提升较小，说明使用 LTR 能够更好的挖掘特征之间的关系，但是受限于特征的选取，最终的提升幅度有限。在 MRR 的指标下，LTR 的结果同样优于线性模型，进一步说明 LTR 能够更好的将可能的结果排在前面。

表 2 线性组合模型与 LTR 模型下推荐结果比较

Table 2 Comparison of recommendation performance for Linear-Combination Model and LTR model

评价指标		Top-5	Top-10	Top-15	Top-20
Hit ratio	线性模型（%）	9.46	14.00	17.02	19.38
	LTR（%）	11.44	15.28	17.44	19.60
	提升率（%）	**20.93**	**9.14**	**2.47**	**1.14**
MRR	线性模型	0.0538	0.0599	0.0622	0.0635
	LTR	0.0657	0.0708	0.0725	0.0734
	提升率（%）	**22.12**	**18.20**	**16.56**	**15.59**

在此基础上，我们将用户之间的 follow 信息与交流频率信息加入到社交关联度度量中，得到的推荐准确率的提升如下表 3 所示:

表 3 加入更多社交关联信息后的线性组合模型与 LTR 模型下推荐结果比较

Table 3 Comparison of recommendation performance for Linear-Combination Model and LTR model after adding more social connection information

评价指标		Top-5	Top-10	Top-15	Top-20
Hit ratio	线性模型（%）	10.57	15.07	18.44	20.99
	LTR（%）	16.04	20.20	23.16	25.04
	提升率（%）	**51.75**	**34.04**	**25.60**	**19.29**
MRR	线性模型	0.0609	0.0669	0.0695	0.0709
	LTR	0.1172	0.1227	0.1251	0.1262
	提升率（%）	**92.45**	**83.41**	**80.00**	**78.00**

当特征的数量增加时，手动调优的线性模型的调优能力受到了限制，而 LTR 在增加特征上获得的提升远大于手动调优的结果，最终，命中率和 MRR 两个评价指标中 LTR 都得到了极大的提升。

3.3.3 案例分析

为了更全面的了解本文中提出的推荐方法的有效性，我们随机挑选了 2 个用户（分别为 Jan Hovancik 和 Yankun），并人工分析推荐项目与其历史开发活动的关系。

在进行案例分析时，我们主要专注于以下几个问题：1）推荐给用户的项目是什么?2）这些项目是否与用户的历史活动有关。因为使用随机抽样，并对结果进行人工评估，我们能够对推荐方法的优劣有一个更直观的认识，更加全面的了解我们推荐方法的细节内容。

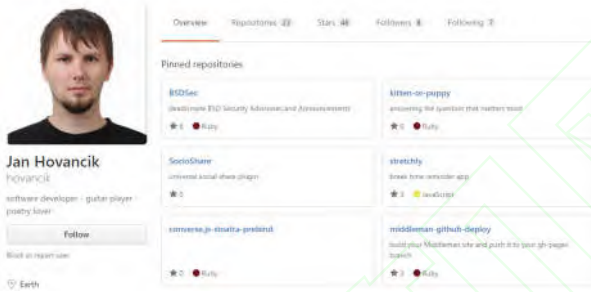

图 9 开发者 Jan Hovancik 的用户主页

Figure 9 Homepage of Jan Hovancik in GitHub

图 9 是开发者 Jan Hovancik 在 GitHub 的用户主页，从其主页中我们能够发现 Hovancik 是一个 Ruby 开发者，主要从事 Ruby 和 JavaScript 的开发。表 4 是我们的推荐方法为其推荐的项目，在推荐项目列表里，有 7 款为 Ruby 项目，2 款涉及前端内容，这些项目在 GitHub 中都具有一定的影响力，又与 Hovancik 的兴趣领域相关，我们认为这是一个比较理想的推荐列表。

表 4 开发者 Jan Hovancik 的推荐项目列表

Table 4 The recommendation repo list for developer Jan Hovancik

项目名称	项目语言	项目描述
Homebrew	Ruby	Beer, The missing package manager for OS X.
habitrpg-shared	CSS	Shared resources useful for multiple HabitRPG repositories. Assets (sprites, imgs, etc), CSS, algorithms, and more.
Paperclip	Ruby	Easy file attachment management for ActiveRecord
factory_girl	Ruby	A library for setting up Ruby objects as test data.
habitrpg-mobile	JavaScript	HabitRPG mobile application under development. Angular + PhoneGap
WebService-HabitRPG	Perl	Access the HabitRPG API from Perl
Guides	Ruby	A guide for programming in style.
shoulda-matchers	Ruby	Collection of testing matchers extracted from Shoulda
Bourbon	Ruby	A lightweight Sass tool set.
Clearance	Ruby	Rails authentication with email & password.

开发者 Yankun 是一个前端工程师，他参与的项目主要为 CSS 和 JavaScript 开发的项目，在我们推荐给他的项目中 9 款为 JavaScript 项目，如 edp-core，less.js 等，一款 CSS 项目（bootstrap），这些项目都是在 GitHub 中比较成熟的项目，而且与开发者 Yankun 的技术特点相符，对于 Yankun 来说具有快速参与这些项目的基础，而且这些项目都是其参与较多的领域内的项目，我们认为这也是一个比较理想的推荐结果。

这 2 个实例表明，我们的方法可以推荐有相同的技术背景或编程语言的开源项目给用户。对于用户而言，如果用户具有一定的知识储备，并且具有一定的开发经验，那么用户会更容易的参与到推荐的项目中来，并

可能提供贡献。

4 相关工作

开源资源的急剧增长带来了丰富的可复用软件，但同时也带来了信息过载的问题。为了解决这个问题，许多研究人员开始研究软件推荐和检索技术。根据推荐目标的类型不同，相关工作主要包括面向源代码、开发者和文档方面的推荐和搜索引擎技术。

4.1 源代码的推荐

在软件工程领域，软件源代码是最常被复用的组件，已经有很多相关工作专门研究可复用源代码的检索和推荐。

很多工作将软件源代码看做一种文本，然后利用文本挖掘技术，通过分析文本相似度来定位源代码。CodeBroker[13]提出了一种从软件源代码中提取方法名称、类名以及注释的方法，将这些提取出来的信息作为代表软件源代码的文本，然后利用语义相似度来推荐相关软件代码。Crechanik 等人[14]则利用信息检索和软件分析技术，通过分析软件的主题来定位所需的源代码。McMillan 等[15]则将源代码中的 API 调用信息作为度量软件的特征，并利用这一特征信息来对比和发现相似的软件。

除了文本信息外，软件源代码本身也是一种结构化的数据，它包含了有价值的结构化信息，可以作为推荐的特征信息进行利用，如文献[16-18]等都研究如何源代码的结构信息来检索和定位结构类似的软件。Xie 等人[18]利用源代码的结构信息提出了一章基于频繁项集分析的方法来为给定的 API 推荐相应的使用示例代码。他们首先通过搜索引擎获取相关的源代码文件，然后利用机器学习的方法挖掘检索到的代码中 API 的使用模式，然后推荐一个该 API 的使用示例代码片段。

此外，对于源代码推荐，社交媒体是一个新的信息来源。知识问答社区 StackOverflow 上涵盖有大量的软件技术相关讨论帖和代码片段。 Zagalsky 等人[19]利用群体讨论和投票信息来度量代码片段的质量，从而对推荐的代码片段进行有效排序，将高质量代码片段推荐给开发者。

推荐系统通常必须依赖于上下文信息，而搜索引擎则可以利用用户输入的查询关键词来检索所需的资源。当前，有很多研究人员提出和设计了相应的软件源代码搜索引擎。Sourceer[20]充分利用代码中存在的引用和依赖等结构化信息，设计和实现了一个面向大规模开源代码检索与分析系统。OCEAN[21]则为开发人员提供了一个单一检索界面的搜索系统，它将来自多个 Web 搜索引擎的检索结果进行综合分析和汇聚，并将这些结果通过单一的入口呈现给用户。OSSEAN[22]则是一个面向全球开源软件的分析检索平台，它爬取了来自开源软件协同开发社区和知识分享社区的海量开源软件资源，大规模参与者讨论和反馈中蕴含的群体智慧对开源软件进行排名，进而推荐高质量的开源软件。不同于此，Brandt 等人[23]将 Web 搜索界面的整合到集成开发环境中，可以让开发者在进行编程开发的同时很方便地在 IDE 上找到示例代码。

目前的相关研究大多数是面向程序开发的，分析的粒度主要集中在源代码层面。不同于此，我们的工作是以开发人员为中心，在项目的粒度上将开发者可能感兴趣的开源项目推荐给开发人员。

4.2 技术专家推荐

在全球化、分布式软件开发中，如何快速准确的找到一个合适的人来解决一个给定的问题或者与之合作完成项目开发是一个非常挑战但具有价值的问题，该问题也吸引了研究人员的广泛关注。

任务指派是软件开发过程中最常见的一类开发管理活动，对提高软件开发效率和质量具有非常重要的意义。针对 issue 指派问题，Anvic 等人[24]首先利用机器学习算法，基于文本信息找到与给定 issue 类似的历史 issue，然后将参与这些历史 issue 修订的开发者进行排序并指派给指定 issue。Jeong 和 Bhattacharya 等人[25,26]则基于给定 issue 的演化历史构造一个图，然后基于此图利用分类方法来寻找合适的解决人员。Yu 等研究了 pull-requests 的过载问题，提出了基于开发者社交信息和技术兴趣的评阅人推荐技术。

为查找合适的合作者，Surian 等人[27]构造了一个名为“开发项目语言/软件类”的网络，基于这个网络

定义了一个开发者社交关系距离的度量方法，并基于该距离来推荐潜在的合作者。Canfora 等人[28]则通过分析邮件列表和版本控制来了解开发者的社交和技术活跃度，从而为为新加入的参与者推荐更好的“引路人”。

这些技术专家推荐的相关研究主要探讨了如何从开发者的历史开发活动和社交行为来做推荐，这些研究为本文如何度量开发者的技术能力和技术兴趣提供了启发。

4.3 知识文档推荐

不同于商业软件，开源软件开发过程通常缺乏规范的文档，这给开发人员对这些软件的复用带来了困难，但是互联网上的丰富文档资料成为这些软件非常重要的知识来源。例如，在 Stackoverflow 中，关于 Android API 的讨论文档覆盖了 Android 所有类文件的 87%，而且这些讨论被浏览了超过 7 千万次[29]。当前，很多研究人员对面向开发者的知识文档推荐问题展开研究。

Favre 等人[30]提出了一种基于模式匹配的方法来定位源代码对应的文档，Chen 等人[31]则将正则表达式、关键字和聚类方法等结合起来，利用 SVM 模型提高文档推荐的准确度。此外，Rigby 等人[32]通过提取 posts 中的代码判断来帮助查找相关的讨论。

此外，Alberto 等人[33]设计了一个 Eclipse 插件，该插件能够根据开发者在编辑器中输入的代码来自动链接到相应的 StackOverflow 讨论帖。Tao 等人[34]聚焦于将项目 issue 与 Stackoverflow 相关讨论的自动联接，提出了一种将语义相似度和时间局部性相结合的方法，发现和定位与 issue 紧密相关的讨论贴，从而保证项目参与者获得 issue 更全面的信息以更好的解决该 issue。

5 总结

近年来，GitHub 社交化开源开发社区中，发布的开源项目和参与开源的开发者保持持续高速的增长。但是其中大量的开源项目仅仅是昙花一现，还有一些项目则仅仅只是吸引了很少量的开发者参与其中。如何实现开发者个人兴趣与开源项目技术需求之间的最佳匹配，是推动开源项目持续快速发展的关键。一方面，从项目的角度，吸引到最合适的开发者积极踊跃的参与和持续贡献是开源项目获得成功的关键；另一方面，从开发者的角度，每个开发者的精力是有限的，从海量开源项目中找到自己最感兴趣的项目并合理的分配时间持续参与是开发者参与开源不断成长的重要因素。但是，开源资源库的大规模和高速增长，以及开发者参与活动数据的高度分散，为实现这种开发者与开源项目的匹配带来巨大挑战。

在本文中，我们提出了一个名为 RepoLike 的方法，充分利用 GitHub 中开源项目与开发者行为活动数据，综合多个维度的特征，基于 LTR 实现对开发者进行个性化开源项目推荐。具体的，我们综合考虑开源项目自身流行度、开源项目之间的技术相关度以及开发者之间的社交关联度三个不同的维度的特征，通过对开发者已经参与项目的深度对开源项目进行排序并构建训练数据集，利用 LTR 的方法构建基于三个维度信息的开发者个性化开源项目推荐模型。最后，本文对所提方法进行了全面深入的实验验证，对比分析了基于线性综合的推荐方法与基于 LTR 的推荐模型在考虑不同维度信息以及不同时间间隔对实验结果的影响。实验结果表明，本文所提方法综合考虑多维信息，能够以较高的准确度实现面向开发者的个性化开源项目推荐。

在未来，我们可以从不同的方面做进一步的工作。首先，现在开源项目之间的技术依赖关系我们只考虑了评论里的技术参数，这在很大程度上可能会丢失那些没有出现在评论里的真正的技术依赖关系。我们将溯源源代码，挖掘出更深层次的依赖关系。其次，在 GitHub 中，开发者之间除了在 issue 和 pull-request 中的直接交互外，还有如共同修改过相同的代码文件等间接交互信息，下一步我们将进一步挖掘开发者之间的社交关联。最后，为了更精确地评估本文方法的效果，我们会将推荐结果发送给部分 GitHub 中相应的开发人员获取他们对推荐结果的反馈，从而获得更准确的关于推荐效果的评价。

6 致谢

本文工作得到国家资源科学基金项目（No.61432020, No. 61472430 and No. 61502512）和国家重点研发计划课题（No.2016YFB1000805）的资助。本文实验和论文撰写过程中得到国防科学技术大学余跃博士的指导

和建议。在此表示感谢。

References:

[1] Ellison N B, et al. Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication. 2007, 13 (1): 210–230.

[2] Storey M-A, Treude C, van Deursen A, et al. The impact of social media on software engineering practices and tools. In FSE/SDP Workshop on Future of Software Engineering Research. 2010: 359–364.

[3] Begel A, DeLine R, Zimmermann T. Social media for software engineering. In FSE/SDP Workshop on Future of Software Engineering Research. 2010: 33–38.

[4] Begel A, Bosch J, Storey M-A. Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. IEEE Software. 2013, 30 (1): 52–66.

[5] Dabbish L, Stuart C, Tsay J, et al. Social coding in GitHub: Transparency and collaboration in an open software repository. In CSCW. 2012: 1277–1286.

[6] Yue Yu, Huaimin Wang, Gang Yin, Tao Wang. Reviewer Recommendation for Pull-Requests in GitHub: What Can We Learn from Code Review and Bug Assignment?. Information and Software Technology Journal, Elsevier, 2016, volume 74, pp. 204-218.

[7] Wang H M, Yin G, Xie B. Research on network-based large-scale collaborative development and evolution of trustworthy software[J]. Sci Sin Inform, 2014, 44: 1-19.

[8] Zhou M, Mockus A. Does the initial environment impact the future of developers?. Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 271-280.

[9] Blincoe K, Harrison F, Damian D.: Ecosystems in GitHub and a Method for Ecosystem Idendification using Reference Coupling. Mining Software Repositories (MSR), 2015

[10] Zhu J, Shen B, Hu F. A Learning to Rank Framework for Developer Recommendation in Software Crowdsourcing. 2015 Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2015: 285-292.

[11] Chen X, Qin Z, Zhang Y, et al. Learning to rank features for recommendation over multiple categories. Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. ACM, 2016: 305-314.

[12] Yang C, Fan Q, Wang T, et al. RepoLike: Personal Repositories Recommendation in Social Coding Communities. Proceedings of the 8th Asia-Pacific Symposium on Internetware on Internetware. ACM, 2016.

[13] Y. Ye and G. Fischer, Information Delivery in Support of Learning Reusable Software Components on Demand, Proc. 2002 Int'l Conf. Intelligent User Interfaces (IUI '02), pp. 159-166, 2002.

[14] GrechanikM, Fu C, Xie Q, et al. A search engine for finding highly relevant applications. In Software Engineering, 2010 ACM/IEEE 32nd international Conference on. 2010: 475–484.

[15] C. McMillan, D. Poshyvanyk, and M. Grechanik. Recommending source code examples via api call usages and documentation. In Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, RSSE '10.

[16] Lozano A, Kellens A, Mens K. Mendel: Source code recommendation based on a genetic metaphor. Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011: 384-387.

[17] R. Holmes and G. C. Murphy, Using structural context to recommend source code examples, in ICSE 2005, 2005, pp.117–125

[18] Xie T, Pei J. MAPO: Mining API usages from open source repositories. In Proceedings of the 2006 international workshop on Mining software repositories. 2006: 54–57.

[19] Zagalsky A, Barzilay O, Yehudai A. Example overflow: Using social media for code recommendation. Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering. IEEE Press, 2012: 38-42.

[20] Bajracharya S, Ossher J, Lopes C. Sourcerer: An internet-scale software repository. In Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation. 2009: 1–4.

[21] Kokkoras F, Ntonas K, Kritikos A, et al. Federated Search for Open Source Software Reuse. In Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on. 2012: 200–203.

[22] Yin G, Wang T, Wang H, et al. OSSEAN: Mining Crowd Wisdom in Open Source Communities//Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 2015: 367-371.

[23] Brandt J, Dontcheva M, Weskamp M, et al. Example-centric programming: integrating web search into the development environment. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2010: 513–522.

[24] Anvik J, Hiew L, Murphy G C. Who Should Fix This Bug?. In ICSE. 2006: 361–370.

[25] Bhattacharya P, Neamtiu I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In ICSM. Sept 2010: 1–10.

[26] Jeong G, Kim S, Zimmermann T. Improving Bug Triage with Bug Tossing Graphs. In FSE. 2009: 111–120.

[27] Didi Surian, Nian Liu, David Lo, Hanghang Tong, Ee-Peng Lim, Christos Faloutsos: Recommending People in Developers' Collaboration Network. WCRE 2011: 379-388

[28] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, Sebastiano Panichella: Who is Going to Mentor Newcomers in Open Source Projects? FSE 2012: 44.

[29] Miltiadis Allamanis, Charles A. Sutton: Why, when, and what: analyzing stack overflow questions by topic, type, and code. MSR 2013: 53-56.

[30] Favre J-M, Lammel R, Leinberger M, et al. Linking documentation and source code in a software chrestomathy. In Reverse Engineering (WCRE), 2012 19th Working Conference on. 2012: 335–344.

[31] Chen X, Grundy J. Improving automated documentation to code traceability by combining retrieval techniques. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. 2011: 223–232.

[32] Rigby P C, Robillard M P. Discovering essential code elements in informal documentation. In Proceedings of the 2013 International Conference on Software Engineering. 2013: 832–841.

[33] Bacchelli A, Ponzanelli L, Lanza M. Harnessing stack overflow for the ide. In Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on. 2012: 26–30.

[34] Wang T, Yin G, Wang H, et al. Linking stack overflow to issue tracker for issue resolution. Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware. ACM, 2014: 11-14.

附中文参考文献:

[7] 王怀民, 尹刚, 谢冰等. 基于网络的可信软件大规模协同开发与演化. 中国科学信息科学. 2014, 44 (1): 1 - 19.