软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(4):1666−1694 [doi: 10.13328/j.cnki.jos.006721]

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

基于动态偏好和竞争力的群智化任务推荐*

王红兵[1], 严 嘉[2], 张丹丹[2], 陆荣荣[2]

[1](东南大学 计算机科学与工程学院, 江苏 南京 210096)

[2](智能服务系统与应用实验室(东南大学), 江苏 南京 210096)

通信作者: 王红兵, E-mail: hbw@seu.edu.cn

摘 要: 近年来, 群智化软件开发已被学术和工业广泛关注. 与传统方法相比, 群智化软件开发最大化利用全球开发者资源完成复杂的发展任务, 有效降低开发成本并提高发展效率. 考虑将推荐技术引入开发者和任务的匹配问题中, 即向软件开发者推荐合适的软件开发任务. 考虑从两方面解决该问题: 一方面, 开发人员的任务选择与兴趣偏好的变化有关, 因此需要准确捕获. 另一方面, 软件开发任务与传统商品或其他内容相比具有专业特性, 只有相应技能的人才能完成和竞争性质的平台更多, 所以开发者也会考虑是否其在众多竞争对手中有较高的评分. 因此, 研究并完成以下工作: 对开发者建模时考虑其动态偏好和竞争力并定义相关的参数指标. 提出一个两阶段的群智化软件任务推荐模型: 第 1 阶段使用基于注意力机制的长短期记忆神经网络预测开发者当前的动态偏好, 并利用相似度从大量候选任务中筛选出符合偏好的Top-*N*任务; 第2阶段利用开发者的竞争力, 使用基于差分进化算法的极端梯度提升方法预测开发者在第 1 阶段任务上的评分, 并按照评分高低向开发者推荐 Top-*K* 任务. 为了验证其有效性, 进行了一系列的实验与已有方法作对比. 实验结果表明, 所提出的模型在群智化软件任务推荐上有显著优势.

关键词: 群智化; 任务推荐; 长短期记忆神经网络; 注意力机制; 极端梯度提升; 差分进化算法

中图法分类号: TP311

中文引用格式: 王红兵, 严嘉, 张丹丹, 陆荣荣. 基于动态偏好和竞争力的群智化任务推荐. 软件学报, 2023, 34(4): 1666–1694. http://www.jos.org.cn/1000-9825/6721.htm

英文引用格式: Wang HB, Yan J, Zhang DD, Lu RR. Group-intelligent Task Recommendation Based on Dynamic Preferences and Competitiveness. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1666–1694 (in Chinese). http://www.jos.org.cn/1000-9825/6721.htm

Group-intelligent Task Recommendation Based on Dynamic Preferences and Competitiveness

WANG Hong-Bing[1], YAN Jia[2], ZHANG Dan-Dan[2], LU Rong-Rong[2]

[1](School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

[2](Intelligent Service System and Application Laboratory (Southeast University), Nanjing 210096, China)

Abstract: As a novel schema of software development, software crowdsourcing has been widely studied by academia and industry. Compared with traditional software development, software crowdsourcing makes the most use of developers all over the world to complete complex development tasks which can effectively reduce costs and improve efficiency. Nevertheless, because there are a large number of complex tasks in the current crowdsourcing platform and inaccurate task matching will affect the progress and quality of task solutions, it is very important to study the matching problem between developers and tasks. Therefore, this study utilizes the dynamic preferences and competitiveness features of developers and proposes a task recommendation model to recommend appropriate software development tasks for developers. First, the attention mechanism based-long short-term memory network is adopted to predict the current preference of a developer to screen out the top-*N* tasks that conform to the preference from the candidate tasks. On this basis, according to

* 基金项目: 国家重点研发计划(2018YFB1003800); 国家自然科学基金(62072097); 江苏省重点研发项目(BE2021001-2); 江苏省软件新技术与产业化协同创新中心、无线通信技术协同创新中心

收稿时间: 2021-04-12; 修改时间: 2021-12-09, 2022-01-30; 采用时间: 2022-06-07

the developer's competitiveness, differential evolution algorithm based-extreme gradient boosting is used to predict the developer's scores of top-*N* tasks, thus further filtering out the top-*K* tasks with the highest scores to recommend to the developer. Finally, in order to verify the validity of the proposed model, a series of experiments is carried out to compare the existing methods. The experiment results illustrate that the proposed model has significant advantages in task recommendation in software crowdsourcing.

Key words: crowdsourcing; task recommenddation; long short-term memory network; attention mechanism; extreme gradient boosting; differential evolution algorithm

随着时代的发展, 软件的规模和复杂性呈现不断增长的趋势, 软件的开发组织方式也在不断演化. 由于互联网环境的出现和发展以及群体智能思想的涌现, 一种基于互联网群体智能的软件开发方式(简称为群智化软件开发)应运而生在该种开发模式下, 一个公司或者个人通过利用群体的智慧, 把过去由员工执行的工作任务以自由自愿的形式发布给非特定的(而且通常是大型的)大众网络来完成[1]. 针对于不同的任务, 大众可以采取不同的方式来完成该任务, 可以是纯粹竞争性质的方式, 对于一项需求方发布的任务, 多个个体同时单独完成该任务, 最后质量最好的结果会被采纳且相应个体能够获得一定的报酬. 或者, 也可以是纯粹协作的方式, 多个人一起协作完成某一项任务, 个体之间会不断地相互交流, 以充分利用群体的智慧. 可以是既有竞争又有协作的方式, 当单个人的成果无法满足需求方的需求时, 可以综合多个人的结果来达到需求的目标. 因此, 相比于传统的软件开发, 群智化软件开发可以最大限度地利用分布在世界各地的开发人员资源, 采用群体竞争或协作的方式来完成复杂的开发任务, 能够有效降低开发的成本, 提高开发的效率.

目前, 互联网上已经有了许多支持群智化的平台, 供需求方发布任务以及大众来获取并完成任务通用的群智化平台, 如 Amazon Mechanical Turk (AMT)和 CrowdFlower, 这两个平台的任务通常是一些数美分起价的普通任务, 比如数据收集、数据验证、图片识别等等. 另外, 也有一些专门针对软件开发的群智化平台, 如 TopCoder、AppStori 和 uTest. TopCoder 是最早的群智化软件开发平台, 它拥有专门的软件开发模型, 对于软件开发流程中的每一阶段, 它都设计了标准化的文档, 从而保证每一阶段的开发人员可以在不与前后阶段开发人员沟通的情况下工作, 使得各阶段的开发人员具有独立性. AppStori 是一个用于移动应用程序开发的群智化平台, 它使用众筹模式为开发提供资金, 并吸引开发人员和消费者密切合作. 开发人员可以发布他们的项目, 从群众中筹集资金, 或者招募其他开发人员用于应用程序的实现. 消费者可以为新的应用程序开发提出想法, 贡献资金, 充当测试人员, 并为现有项目提供反馈. 整个开发过程, 从概念到发布, 都是通过开发人员和消费者之间的协作来实现的[2]. uTest 是一个专门针对软件测试的群智化平台, 声称是世界上最大的支持软件测试的开放社区. 它支持多种测试方式, 如功能测试、可用性测试、本地化测试和负载测试等等.

虽然群智化软件开发模式已经取得了一定的成功, 但是依然面临着一些问题.

(1) 信息过载, 挑选任务困难. 目前群智化平台上存在着大量需求方发布的任务, 且任务数量呈现日益上涨趋势. 开发者往往需要花费大量的时间在选择任务上, 这就会影响开发者的积极性, 同时增加了任务从发布到完成的时间. 另外, 开发者更容易选择刚刚发布出的任务, 某些任务可能由于排序靠后而长时间无人问津, 使任务难以完成.

(2) 任务复杂, 质量难以保证. 由于软件开发任务流程复杂, 工作量大, 因此需要开发人员具备专业领域知识和技能. 开发人员凭借自己的主观意愿选择的任务可能并不是最适合自己的, 不准确的任务匹配最后会影响任务完成的进度和质量, 继而满足不了需求方的需求.

推荐技术作为一种有效地解决信息过载的方法, 已经在学术界研究了多年, 并且在许多领域得到了应用, 如商品推荐、微博推荐、App 推荐等. 因此, 本文考虑将传统领域的推荐技术引入开发者和任务的匹配问题解决中, 即为群智化软件开发者推荐其合适的软件开发任务, 从而减少开发者挑选任务的时间, 提高开发者完成任务的质量, 提升开发者参与任务的积极性, 最终使得整个群智化软件开发平台能够高效有序地运转.

对于传统的网络平台中的用户, 他们选择一项商品或内容的出发点往往是其自身的兴趣偏好, 与其他因素没有太大的关联. 但是, 对于一名群智化软件开发平台中的开发人员而言, 他们对于任务的选择不仅基于自身的兴趣偏好, 还会考虑自己能否顺利地完成该任务并获得一定的收益. 尤其是在更为常见的竞争性质的

软件开发任务中, 开发人员需要拥有特定的领域知识和技能, 需求方会从众多开发者或其提交的任务结果中选择一个或几个满意的结果, 这个过程反映的是开发者之间竞争力的差异, 而这种竞争力可能与开发者的开发能力以及他所拥有的空闲时间或者精力有关, 最后需求方的选择决定了双方能否达成交易, 即开发者能否获得报酬. 因此, 开发者的兴趣偏好和竞争力的度量在群智化软件开发任务选择中显得非常重要.

综上所述, 根据群智化软件开发领域的特点, 如何综合考虑开发者的兴趣偏好和竞争力, 提出一种能够有效帮助开发者解决任务选择困难的推荐方法是值得深入钻研的问题. 本文针对群智化软件开发任务的选择问题进行了探索研究, 构建了任务推荐方法, 有着重要的理论及实践意义. 在理论方面, 探索影响开发者选择软件任务的影响因子, 基于开发者的兴趣偏好和竞争力构建开发者模型, 有利于加深群智化平台中开发者行为的科学认识并提出了一套适用于群智化软件开发领域的的任务推荐方法, 扩大了传统推荐方法的应用范围. 在实践方面, 对于开发者而言, 帮助他们找到满足自己兴趣偏好并且具有竞争力的任务, 减少挑选任务的时间, 提高开发效率, 使开发者能够更快更好地获得报酬; 对于需求方而言, 任务能够更快地推送给合适的开发者, 有助于提高任务结果的质量; 对于群智化软件开发平台而言, 提高用户体验和平台效率, 提升开发者参与任务的积极性, 使整个平台能够高效有序地运转, 充分显示出群智化软件开发模式的优越性.

为了应对群智化软件任务推荐所面临的问题, 本文在对开发者建模时考虑了开发者的动态偏好以及竞争力, 定义了反映这两个特征的参数指标, 并提出了一个两阶段的群智化软件任务推荐模型: 基于注意力机制的长短期记忆神经网络(attention mechanism-based long short-term memory, A-LSTM)与基于差分进化算法改进的极端梯度提升(differential evolution algorithm-based extreme gradient boosting, DE-XGBoost)相结合的群智化软件任务推荐, 后面简称为 A-LSTM+DE-XGBoost. 本文的方法可以用于群智化软件开发平台中, 在众多的软件开发任务中给开发者筛选并推荐合适的开发任务.

本文的具体内容如下:

(1) 本文根据开发者的动态偏好和竞争力来对开发者建模, 并定义了相应的参数指标: 根据开发者报名的历史任务信息来获得开发者的任务偏好, 并按照时间顺序划分来得到任务偏好序列; 以开发者报名、提交、获胜的历史任务表现情况、当前任务参与情况以及潜在竞争对手信息来构建开发者的竞争力; 以上述参数特征作为相应模型的输入, 进而实现训练.

(2) 本文提出了一个两阶段的群智化软件任务推荐模型: 第 1 阶段使用基于注意力机制的长短期记忆神经网络(A-LSTM)预测开发者当前的动态偏好, 并利用相似度从大量候选任务中筛选出符合偏好的 Top-N 任务; 第 2 阶段利用开发者的竞争力, 使用基于差分进化算法改进的极端梯度提升方法(DE-XGBoost)预测开发者在前一阶段筛选出的任务上的评分, 并按照评分从高到低向开发者人推荐 Top-K 任务, 从而进一步提高推荐的准确性.

(3) 本文收集了真实的群智化软件开发平台的数据, 针对不同的参数取值设置了多组实验, 并将我们提出的模型与几种用于推荐的经典方法进行了对比, 实验结果表明了我们的模型具有更好的鲁棒性和更高的推荐准确度.

本文第 1 节对群智化软件开发的相关工作进行介绍, 主要包括群智化软件开发的相关概念以及国内外有关群智化软件任务推荐相关工作的研究现状. 第 2 节主要介绍本文所提出的群智化软件任务推荐模型涉及到的相关技术的理论基础, 其中包括长短期记忆神经网络(long short-term memory, LSTM)、注意力机制、极端梯度提升(extreme gradient boosting, XGBoost)和差分进化算法, 为后文模型的建立提供了理论支持. 第 3 节细致讨论如何建立基于开发者动态偏好和竞争力的群智化软件任务推荐模型. 首先给出本文所解决问题的描述; 接着对开发者建模中的两类特征即动态偏好和竞争力进行了定义, 给出了相关的参数指标; 最后介绍了如何依据上述特征建立模型从而给软件开发者推荐合适的任务. 第 4 节通过实验的方式对本文提出的任务推荐模型的性能进行验证, 首先介绍实验所使用的数据集; 接着说明实验的验证环境和模型参数等; 然后列出用于性能评估的评价指标; 最后分析实验的结果, 探讨不同模型之间的性能差异. 第 5 节对本文的研究工作及贡献进行总结, 并指出本文工作中存在的一些问题和不足, 在此基础上对未来的研究方向进行了展望, 为后续

工作的展开提供了建议.

1 相关工作

本节针对群智化平台中软件任务推荐问题进行相关工作的阐述. 首先介绍群智化软件开发的相关概念, 包括互联网群体智能的概念、基于互联网群体智能的软件开发的相关角色、激励方式、组织形式和以 TopCoder 为代表的群智化软件开发的一般流程. 然后对群智化平台中任务推荐的国内外研究现状进行了分析, 涉及通用的普通任务推荐以及专门的软件开发任务推荐.

1.1 群智化软件开发概述

互联网的不断发展给软件系统及其开发方式带来了机遇与挑战. 一方面, 互联网形成了一个开放且具有动态性的软件运行环境, 使得软件系统的规模和复杂性急剧增加, 给软件开发带来了挑战. 另一方面, 互联网打破了人与人之间的物理时空限制, 使得任何分布在世界各地的人类个体可以通过互联网形成一个紧密的群体, 从而促成了“基于互联网的群体智能”, 为解决软件开发面临的问题提供了一种新的解决方案, 即基于互联网群体智能的软件开发, 通过利用互联网加快个体之间的信息传播和共享, 汇聚不同个体的开发能力, 提高软件开发的效率和质量.

1.1.1 基于互联网群体智能

群体智能的思想是指多个个体通过协作和竞争所涌现出的集体智慧, 其强调了多个个体在组成群体之后, 整体上能够显现出远超其中任一个体智能水平的智能能力, 从而在复杂问题求解上具有明显的优势[3−5].

群体智能的提出最早来自于人们对于低等生物或动物群体的观察, 他们发现自然界中部分生物虽然单个个体缺乏智能或者只有有限的智能, 但是个体之间通过分工合作后可以完成较为复杂的任务, 表现出较高的智能. 这是一种由量变产生质变的现象, 是一种涌现现象, 是群体中的个体通过不断交互而产生的一种自组织行为[6]. 针对这些生物的群体行为规律, 早期学者提出了一系列进化优化算法, 其中包括蚁群优化算法[7]、蚂蚁聚类算法[8]、粒子群优化算法、蜂群算法[9]等, 并应用于 TSP 问题、数据分析、医疗系统、无人机编队等领域.

群体智能并不仅仅会在低等生物或动物群体中出现, 在人类个体和社会中, 都会产生这样的现象. 从单个人来看, 人脑的神经元结构中包含了大约 1 000 亿的神经元[10], 一个人的智能本身就是一种群体智能, 单个人拥有复杂的认知和心理就是在大规模神经元的错综复杂的交互中产生的. 从人类社会来看, 社会的不断发展和演化、人类的许多重要文明成果都是群体智能的体现, 都是人类个体在群体化、社会化生活中不断演化的结果. 无论是低等生物或动物群体的群体智能, 还是面向人类群体的群体智能, 这两种群体智能的共同之处在于群体对于单个个体智能的增强放大作用, 也即基本个体无论具有怎样的智能, 在群体层次上涌现出的集体智能会超过任一个体的智能水平.

但是, 对于大规模人类群体智能的利用则较为有限, 这主要是因为受制于特定的物理空间和时间段, 进行大规模的人类群体协同往往需要付出巨大的代价, 可行性很低. 互联网的出现及其快速发展有效地解决了这一难题. 互联网拉近了两个分布在世界各地的人类个体间的距离, 使得任何一群在地理上分布的人类个体都可以利用互联网形成一个大规模的紧密群体, 同时, 互联网提高了人类信息的总量以及不同个体之间信息的传播和共享速度, 从而诞生了“基于互联网的群体智能”. Li 等人[11]给出了在当前互联网环境下他们所理解的群体智能的定义: “群体智能来自于集合大量自主个体的智能的结果, 这些自主的人类个体被激励在一定的基于互联网的组织结构下执行具有挑战性的计算任务从而超越人类个体智能的限制”. 基于此, 大量基于互联网群体智能的应用也在各种领域内陆续出现和不断发展, 这其中包括多人协作进行知识收集的百科全书 Wikipedia; 利用大规模人类群体辅助药物开发的 InnoCentive; 以及采用开发者竞争或协作方式的分布式软件开发[12−14]等等. 虽然面向的领域和所解决的问题种类不同, 但这些基于互联网群体智能的应用都体现出 3 个重要的特征: (1) 一个开放的大规模人类群体; (2) 群体中的个体之间进行直接或间接的交互; (3) 大量个体行为的融合或汇集.

1.1.2 基于互联网群体智能的软件开发

随着信息技术和互联网的不断发展, 近年来, 软件的规模和复杂程度也在呈现着巨大的增长趋势, 现代软件呈现出需求持续变化、规模持续增长、系统持续演化的新形态. 当前的软件开发方式面临着巨大的挑战, 传统的软件开发方式如自动化软件开发方法和工程化软件开发方法等已无法解决当前软件开发各阶段所遇到的各类复杂问题. 为了提高软件开发的效率和质量, 软件工程领域的学者们不断地探索软件开发的新方式.

基于互联网群体智能的软件开发则是一个重要的探索方向, 后简称为群智化软件开发. 李未院士指出, 相比于传统的软件开发方式, 群智化软件开发具有以下的特点: 软件的开发过程由原本的封闭式走向开放式; 开发的方法从机器工程走向社会工程; 开发的组织形式由传统企业走向社区模式; 开发的人员由专业的 IT 公司精英开发者走向普通的大众开发爱好者. 王青研究员在自己关于“软件工程中的群体智慧”的报告中指出, 软件是知识密集型的产物, 随着软件渗透到社会中的各个层面, 大规模的全球分布式协同以及开放合作的软件开发形式是必然的趋势, 群体合作和知识共享将成为现代软件开发的重要形式.

群智化软件开发中通常涉及到 3 类角色, 如图 1 所示, 分别是需求提出方、基于互联网的群智化平台和开发者[2].

(1) 需求提出方: 拥有软件开发需求的公司或者个人, 公司利用群智化开发完成公司的业务需求可以缩短开发周期, 提高开发效率, 降低开发成本, 个人也可以利用该模式帮助自己开发想要的软件.

(2) 基于互联网的群智化平台: 是群智化软件开发的核心, 负责管理整个软件开发流程的业务逻辑, 理想情况下应具备项目管理工具、软件开发工具、质量保障和改进工具以及知识分享和协同工具等等.

(3) 开发者: 既可以是个人也可以是一个开发团队, 依据自己的兴趣和能力选择合适的软件开发任务.

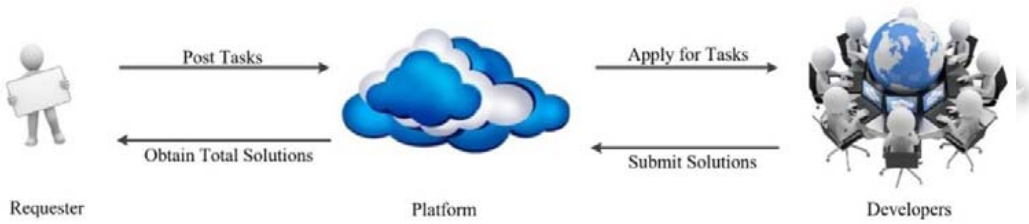


图 1 群智化软件开发中的 3 类角色

为了吸引大规模的个体参与到群智化软件开发中来, 激励机制是不可或缺的, 通常会有以下几种激励方式: 第 1 种即是金钱, 即支付给相应的参与者一定数量的金钱; 第 2 种是声誉, 在群智化平台中完成的软件开发任务数量越多且质量越高, 则越能得到平台和其他开发者的认可, 随着时间的流逝, 这样的开发者会变成平台中的核心开发者, 通常遵循二八定律, 即 20%的开发者会成为核心开发者; 第 3 种则是学习经历, 一部分开发者, 通常是学生或科研人员, 他们参与的目的不是为了金钱或声誉, 而是学习他人提交的结果从而增强自己相应的开发能力.

依据组织形式的不同, 群智化软件开发方式主要可以分为协作式的和竞争式的. 协作式的一种典型形式是开源软件, 在这种开发方式下, 需要解决的软件开发任务一般需要开发者共同合作才能完成, 且完成任务后往往没有物质奖励的报酬; 而在竞争模式下, 需要解决的开发任务通常由个人或小团体单独去完成, 然后以专家评审的方式选择出任务的最佳解决方案并且相应的任务结果提交者能够得到一定的报酬, 本文所研究的即是这种竞争模式下的群智化软件开发, 其中 TopCoder(http://www.topcoder.com)是目前世界上软件开发领域规模最大的竞争式群智化平台, 下面对其进行详细介绍.

TopCoder 建立于 2001 年, 到 2021 年 12 月为止, 该平台上注册的软件开发者已累计达到 150 万人, 开发者们总共获得的累计奖励报酬值已超过 9 900 万美元. 从顶层架构来看, TopCoder 的软件开发过程类似于瀑布模型, 在每一个阶段都采用竞争的形式, 每一个开发任务都将以一个竞赛的形式来完成. 任务通常会依据其类型发布到平台上, 发布的信息包括任务描述、报名截止时间、提交截止时间、报酬金额等等. 每个开发者可以根据自己的兴趣和能力参加竞赛并提交任务解决方案. 在提交截止时间之后, 由人群中选出的评审委员

会依据预先设置的评分规则对提交结果进行评估, 通常平均得分排在前 2 位的开发者会得到相应的奖励, 而获胜的解决方案会作为下一个开发阶段的输入.

整个开发过程如图 2 所示[15], 它开始于需求收集和项目定义阶段, 在这个阶段, 来自于群众或者 TopCoder 平台的项目经理会与需求方进行沟通, 以确定项目目标、任务计划、时间安排和预算. 随后的体系结构阶段依据上一阶段产生的需求规格说明书将软件项目分解为一系列组件(TopCoder 采用基于组件的软件开发方式), 并生成一整套设计文档, 例如 UML 图和组件说明文档. 然后, 组件设计将在随后的开发阶段中实现. 开发阶段可能会使用预先构建的可重用组件, 同时该阶段产生的组件也可能成为可重用软件目录的一部分, 以供将来进行重用. 完成后的组件将在接下来的组装阶段根据体系结构设计进行组合, 并进行系统级的测试. 在最后的部署阶段, 组装的软件将被部署到需求方的质量保障环境中. 经过一段时间的用户验收测试, 所有阶段的成果都会移交给需求方. 据 TopCoder 声称, 与传统的软件开发方式相比, TopCoder 的群智化软件开发模式能够降低 30%–80%的开发成本, 同时缺陷率能够下降 5–8 倍[16].

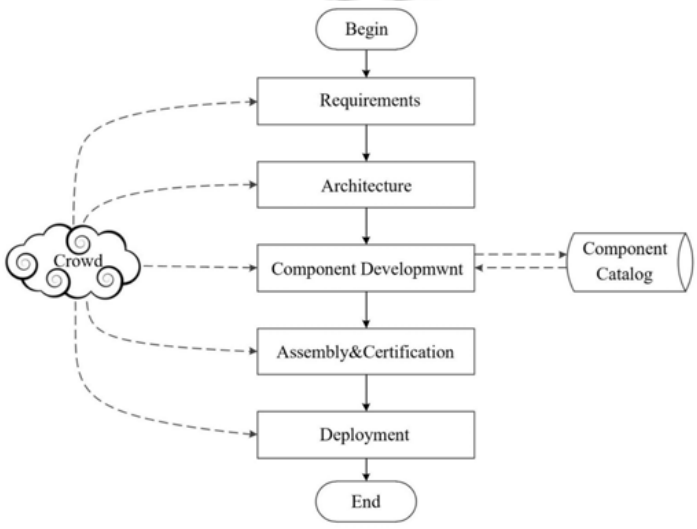


图 2 TopCoder 软件开发过程

1.2 国内外研究现状

近年来, 随着互联网群体智能思想在多个领域的应用, 对于群智化平台中任务推荐的相关理论和关键技术的研究引起了国内外学术界的广泛关注. 其中, 主要的研究方向是针对通用群智化平台的普通任务推荐, 而由于群智化软件开发的发展较慢, 因此相关的软件任务推荐研究较为有限. 以下将分别从通用的普通任务推荐与专门的软件开发任务推荐两个方面对相关国内外研究现状进行阐述.

(1) 普通任务推荐

Ambati 等人[17]依据人员的历史任务信息对其偏好进行建模, 分别采用词袋方法和最大熵方法对任务列表进行重排序, 将合适的任务推荐给他们, 并在小规模数据集上进行了验证. Lin 等人[18]提出了一种采用正负隐式反馈信息来反映员工的兴趣偏好的方法, 从而提高任务推荐质量. Difallah 等[19]利用员工在社交网络中的信息从而识别其偏好类型, 然后综合任务的描述信息, 从而进行任务的推荐. Yuen 等人对该问题研究了多年, 成果较多. 他们通过人员在不同类别的历史任务选择和完成情况来给每一个类别的待推荐任务进行打分[20], 从而挑选出各个类别中分数排名靠前的任务推荐给人员. 之后, 他们引入了人员的历史任务搜索情况[21−23], 并根据人员的历史行为将人员对任务的偏好分为多个等级, 并采用概率矩阵分解方法求解人员选择不同任务的偏好等级, 从而进行任务推荐. 他们还将概率矩阵分解与主动学习方法相结合[24,25]利用主动学习方法在线更新人员、任务和任务类别的特征空间, 从而提高推荐的准确度. Ayswarya 等[26]增加人员获得任务奖励的概

率和参与任务的信息作为预测的特征, 并采用概率隐因子模型和贝叶斯个性化排序矩阵进行任务的推荐, 使得推荐结果更加准确. Kang 等人[27]则将任务推荐问题转换成多臂赌博机问题, 选用多种策略多次尝试去学习人员的偏好和可靠性, 通过累计最大的奖励值来选择最佳的任务.

张亭亭等人[28]运用敏感性分析方法研究了用户对各任务属性特征的敏感性程度, 并结合二部图原理构建用户潜在偏好挖掘模型, 挖掘出用户行为规律中所包含的隐性偏好信息, 使得任务的分配更具有针对性. 还有文献[29]采用了近年来突出的深度学习技术, 基于 Word2vec 建立了语义标签相似性矩阵, 通过该矩阵来计算人员和任务的相似性来向人员推荐合适的任务, 不同的语义矩阵适用于不同的领域, 具有一定的可扩展性.

由于普通任务通常比较简单, 对人员能力要求不高, 不需要专业的知识就可以完成, 而且完成时间较短, 任务和人员模型的描述相对简单, 但是软件开发任务通常比较复杂, 需要具有专业领域知识的人才能完成且周期较长, 无法利用零碎的时间进行完成, 两者建模时需要考虑的因素并不一样, 因此这些推荐方法并不适用于群智化软件任务推荐.

(2) 软件开发任务推荐

与普通任务推荐相比, 由于软件任务的复杂性且群智化软件开发才刚刚受到关注, 因此专门针对软件任务推荐的研究相对较少. 文献[30]提出了一种基于分析的决策支持方法, 利用随机森林技术推荐最有可能获胜的任务以指导开发者进行选择. Tunio 等人[31,32]提出将开发者的性格因素作为一项表示开发者偏好的重要因子, 来对开发者和任务进行合适的匹配, 但是只是一个设想, 没有实验来验证其有效性. Li 等人[33]依据开发者的行为如搜索和投标任务来构建开发者社会网络, 通过协同过滤的方法, 基于开发者朋友的偏好来给其推荐任务. Yu 等人[34]基于动态效用提出了一个任务分配算法. 首先. 利用员工的注册信息估计员工开发能力的初始值. 其次, 员工的开发能力根据他(她)完成的历史任务来进一步计算. 然后, 根据技能权重, 计算出任务需求与员工技能匹配的程度. 最后, 以员工开发能力和技能匹配度的乘积作为分配效用, 以总效用最大化为优化目标.

虽然目前已经有了一些软件开发任务推荐方法, 但是这其中存在着一些问题. 为了给开发者推荐合适的任务, 通常需要考虑两方面因素. 一方面, 开发者需要对该任务感兴趣, 但是以往的这些推荐方法往往考虑的是开发者的静态兴趣偏好, 但是开发者的兴趣偏好是在不断变化的, 如何充分把握开发者当前的兴趣偏好对于推荐结果是有重要影响的. 另一方面, 开发者需要有能力去完成任务并能够获得较高的评分且赢得相应的报酬, 然而上述的相关工作往往只度量了开发者相关技能的熟练度, 没有考虑到其他的因素. 由于软件开发任务周期较长, 因此还需要将开发者当前的时间精力因素考虑进去, 同时对于更为常见的竞争性质的软件开发任务, 每一个任务能够获胜的只有一个或几个开发者, 且获胜者会依据评分及名次的高低得到相应的报酬, 所以在推荐任务时还需要关注其竞争对手的信息, 以便提高开发者在任务上获得的评分, 从而增加获胜的概率以及报酬的金额. 在本文中, 我们通过开发者在一项任务提交的结果上获得的评分来度量一个开发者在该任务上的竞争力. 综上所述, 本文提出了一种基于开发者动态偏好和竞争力的群智化软件任务推荐方法, 该方法考虑了开发者的偏好变化, 能够更加精准地捕获开发者的兴趣, 同时考虑开发者的综合竞争力, 基于其竞争力, 帮助开发者找到最适合其完成并能够获得最高评分的任务.

2 理论基础

本节对本文所涉及到的相关技术基础进行了细致的介绍, 主要包括长短期记忆神经网络、注意力机制、极端梯度提升、差分进化算法等. 其中, 长短期记忆神经网络是一种有效的处理时序数据的神经网络模型, 在其之上引入注意力机制后可以进一步地挖掘出历史时序数据对当前数据的影响. 而极端梯度提升模型是一种对梯度提升决策树(gradient boosting decision tree, GBDT)算法的有效改进, 引入了正则化项, 性能得到很大提升, 很适合用于处理回归预测问题. 差分进化算法是一种基于群体差异的随机搜索优化算法, 具有良好的全局搜索能力, 通过该算法对极端梯度提升模型的大量参数进行寻优能够进一步提高模型的性能.

2.1 长短期记忆神经网络(LSTM)

LSTM 模型是循环神经网络模型(recurrent nerual network, RNN)的改进, 是一种适用于处理时序数据的神经网络模型, 它能够充分利用数据的顺序信息. Hochreiter 等人[35]提出了 LSTM 模型来解决 RNN 的这一问题. LSTM 模型在 RNN 的基础上在隐藏层引入了记忆单元来保存历史信息和长期状态, 使用门控来控制信息的流动. 如图 3 所示为 LSTM 的结构图. 在其隐藏层的记忆单元中存在 3 种门控: 遗忘门、输入门和输出门, 在门控的作用下, LSTM 的前向传播过程如下.

(1) 计算遗忘门的值: 遗忘门用于控制上一时刻神经元的历史信息, 决定保留或者舍弃上一时刻神经元的哪些历史信息.

$$f_t=\sigma\left(W_{xf}{}^{x_t}+W_{hf}{}^{h_{t-1}}+b_f\right) \tag{1}$$

(2) 计算输入门的值: 输入门用于控制当前时刻神经元的输入信息, 决定将当前时刻神经元的哪些输入信息更新到当前时刻神经元的记忆单元中.

$$i_t=\sigma\left(W_{xi}{}^{x_t}+W_{hi}{}^{h_{t-1}}+b_i\right) \tag{2}$$

(3) 计算输出门的值: 输出门用于控制当前时刻神经元的输出信息, 决定将当前时刻神经元的记忆单元中的哪些信息输出给下一时刻神经元.

$$o_t=\sigma\left(W_{xo}{}^{x_t}+W_{ho}{}^{h_{t-1}}+b_o\right) \tag{3}$$

(4) 更新当前时刻神经元的记忆单元值: 由遗忘门和输入门共同决定其值.

$$g_t=\sigma\left(W_{xg}{}^{x_t}+W_{hg}{}^{h_{t-1}}+b_g\right) \tag{4}$$

$$c_t=f_tc_{t-1}+i_tg_t \tag{5}$$

(5) 计算当前时刻神经元的输出值: 由输出门决定其值.

$$h_t=o_t\tanh(c_t) \tag{6}$$

其中, x_t 为当前时刻神经元的输入值, h_{t-1} 为上一时刻神经元的输出值: f_t、i_t、o_t、g_t、c_t、h_t 分别代表遗忘门、输入门、输出门、候选记忆单元值、当前时刻记忆单元值、当前时刻输出值; W 为权重矩阵, b 为偏置; σ 代表 sigmoid 函数, 取值在 0 到 1 之间; tanh 代表双曲正切函数, 取值在–1 到 1 之间.

LSTM 权值和偏置的更新可以采用基于时间的后向传播算法 BPTT (back-propagation through time).

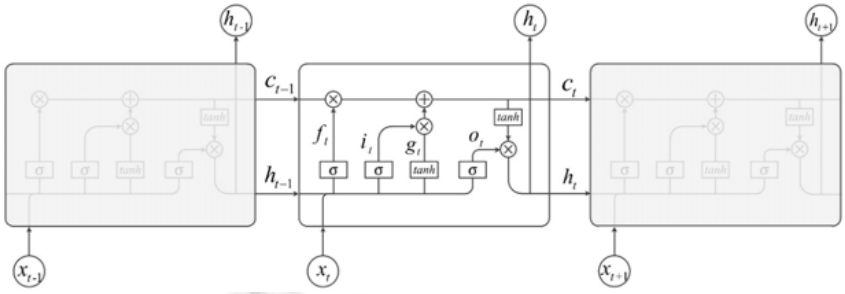


图 3 长短期记忆神经网络结构图

2.2 注意力机制

注意力机制[36]模拟人脑注意力的特点, 核心思想是: 对重要的内容分配较多的注意力, 对其他部分分配较少的注意力. 注意力机制已在很多领域得到应用, 包括图像标注[36,37]、文本分类[38,39]、机器翻译[40]等.

注意力机制实际上就是对输入的每一个状态进行自动加权求和, 其主要用于处理这样的问题: 对于 T 个 d 维的特征向量 $h_t(t=1,2,\ldots,T)$, 需要整合这 T 个特征向量包含的信息, 并从中提取一个 d 维的新特征向量 h^*. 对于这个问题, 主要有两种处理方法.

(1) 简单地对这 T 个 d 维向量按照对应元素平均值, 从而得到一个相同维数的向量 h^*;

(2) 对这 T 个向量分配各自的权重 $a_t(t=1,2,\ldots,T)$, 然后对其进行加权求和.

$$h^* = \sum_{t=1}^{T} a_t \cdot h_t \tag{7}$$

显然, 第 1 种方法过于简单, 其计算结果不够合理, 相对而言, 第 2 种方法提取的新向量 h^*更加合理, 而注意力机制实现的就是对这 T 个输入向量分配合理的权重 a_t, 并对所有 h_t 加权求和来提取 h^*向量.

因此, 注意力机制主要完成的就是对其每个输入向量计算出一个合理的权重 a_t. a_t 的计算一般分为两个步骤.

(1) 打分函数 f 的设计. 设计一个打分函数, 根据每一个输入 h_t 与注意力机制所关注的对象的相关程度, 计算其权重分数 s_t, s_t 的值越大, 则说明 h_t 与所关注对象的相关性越高, 对其影响越大.

(2) 对权重归一化. 由于打分函数 有多种设计方法, 其计算出来权重分数量级各不相同, 所以注意力机制对 f 计算出来的每一个 s_t, 使用一个 softmax 函数进行归一化, 得到最终的权重 a_t.

$$a_t = \frac{e^{s_t}}{\sum_{j=1}^{T} e^{s_j}} \tag{8}$$

针对打分函数的不同, 注意力机制可以主要分为以下 3 类[41].

(1) 注意力机制. 该机制关注的对象是 h_t 本身, 直接根据状态 h_t 来计算权重.

$$c_t = f_t C_{t-1} + i_t g_t \tag{9}$$

其中, $W_a^T \in R^d$, $b_a \in R$, 这两者随机初始化, 并通过训练不断学习.

(2) General 注意力机制. 该机制不仅关注 h_t 本身, 还考虑了 h_t 与指定对象 h_e 的相关性, 通过一个矩阵 $W_a \in R^{d\times d}$ 来捕获两者的关联.

$$s_t = W_a^T \cdot h_t + b_a \tag{10}$$

W_a 也是随机初始化, 通过训练来不断学习的.

(3) Concatenation-based 注意力机制. 在该机制中, 通过对h_t和 h_e 进行拼接, 然后使用一个多层感知机来计算对应的打分, 一般使用 tanh 函数作为激励函数.

$$s_t = v_a^T \cdot \tanh\left(W_a \cdot \left[h_t, h_e\right]\right) \tag{11}$$

其中, $W_a \in R^{d\times 2d}$ 和 $v_a^T \in R^q$ 是多层感知机中待学习的模型参数, q 为多层感知机的输出维度.

本文认为开发者当前的任务兴趣偏好, 主要与近期参与任务中的一个或几个任务有关, 而与其他任务无关. 因此在 LSTM 上引入注意力机制, 并且主要关注的是开发者近期的任务兴趣偏好本身, 没有其他关注的对象, 因此采用 Location-based 注意力机制. 通过对 LSTM 的每一个隐藏层状态施加权重, 评估其对当前任务兴趣偏好的影响, 并通过对所有隐藏层状态进行加权求和, 从而使 LSTM 更专注于找到近期行为中与当前任务偏好显著相关的有用信息, 降低开发者偶然参与任务的干扰, 来生成更准确的动态兴趣偏好特征.

2.3 梯度提升决策树(GBDT)

提升(boosting)方法是一种常用的统计学习方法, 如图 4 所示. 它的一般工作机制是先从初始训练集中训练出一个基学习器, 再根据基学习器的表现对训练样本分布进行调整, 提高那些在前一轮被基学习器分错样本的权值, 降低前一轮分对样本的权值. 接着基于调整后的样本分布来训练下一个基学习器, 如此重复进行, 直至基学习器的数目达到事先指定的值, 最终将这些基学习器进行加权结合.

GBDT[42]是一种非常流行的提升方法. 它以分类与回归树(classification and regression trees, CART)作为基学习器, 并以梯度提升(gradient boosting)作为训练算法. 梯度提升是提升算法中的一大类算法, 其基本思想是根据当前模型损失函数的负梯度信息来训练新加入的基学习器, 然后将训练好的基学习器以累加的形式结合到现有模型中. 假设有一个样本容量为 N 的数据集 $D=[(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)]$ 其中, N 为样本容量, $x_i \in X \subseteq R^d$, d 为特征个数, $y_i \in Y \subseteq R$ 则由分类与回归树和梯度提升结合得到的梯度提升决策树的模型可以表示如下:

$$\hat{f}\left(x_i\right) = \varphi\left(x_i\right) \overline{=} F \sum_{m=1}^{M} f_m\left(x_i\right),\ f_m \in F \tag{12}$$

其中，$f=\left\{f(x)=\omega_{q(x)}\right\}\left(q:R^d\to J,\omega\in R^J\right)$ 是分类与回归树空间，q 代表树的结构，能够将一个样本点映射到相应的叶子节点的索引. J 代表树中叶子节点的个数. 在每一个叶子节点上有一个权重ω_j，对于一个给定的样本点，通过 q 的映射规则将其映射到对应的叶子节点，并通过累计所有叶子节点的权重来预测最后的输出值. 该模型的算法流程如算法 1 所示.

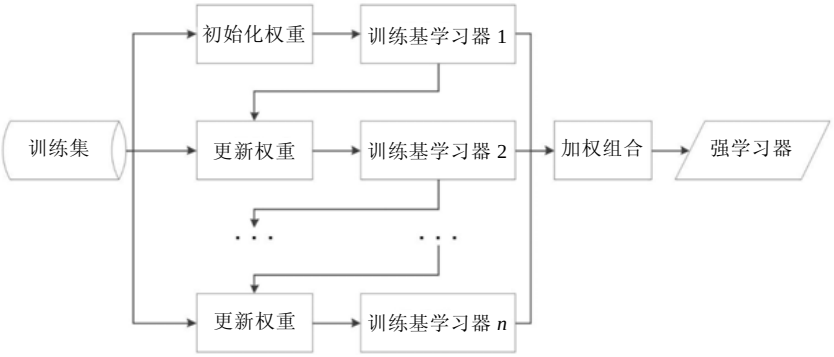


图 4　提升方法示意图

算法 1. GBDT 算法.

输入: 训练数据集 $D=\left[(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\right]$，　$x_i\in X\subseteq R^d$, $y_i\in Y\subseteq R$ 损失函数 $L\left(y,f(x)\right)$;

输出: 梯度提升树 $\hat{f}(x)$.

1. begin
2. 　初始化 $f_0(x)=\arg\min_c\sum_{i=1}^{N}L(y_i,c)$，其中, C 为使损失函数最小的常数
3. 　for m=1,…,M do
4. 　　for i=1,…,N do，计算损失函数的负梯度
5. 　　$$r_{mi}=-\left[\frac{\partial L\left(y_i,f(x_i)\right)}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$
6. 　　endfor
7. 　　对 r_{mi} 拟合一个回归树，得到第 m 棵树的叶子节点区域 R_{mj}, j=1,2,…,J
8. 　　for j=1,2,…,J do，计算每一个叶子节点区域的最优权重
9. 　　$$\omega_{mj}=\arg\min_c\sum_{x_i\in R_{mj}}L\left(y_i,f_{m-1}(x_i)+c\right)$$
10. 　　endfor
11. 　　更新第 m 棵树，其中 $I()$为指示函数，当括号内的式子成立时返回 1，否则返回 0
12. 　　$$f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}\omega_{mj}I\left(x\in R_{mj}\right)$$
13. 　endfor
14. 　得到梯度提升树 $\hat{f}_m(x)=\sum_{m=1}^{M}f_m(x)=\sum_{m=1}^{M}\sum_{j=1}^{J}\omega_{mj}I\left(x\in R_{mj}\right)$
15. end

2.4 极端梯度提升(XGBoost)

XGBoost 由 Chen 等人[43,44]在 2015 年提出，是对 GBDT 的一种改进，在其原来的损失函数上加上了正则化项，并对损失函数进行了二阶泰勒展开，有效提高了算法的性能，避免了过拟合.

在 XGBoost 中，正则化项定义为

$$\Omega(f)=\gamma J+\frac{1}{2}\lambda\sum_{j=1}^{J}\omega_j^2 \tag{13}$$

其中, J 是叶子节点总数, ω_j 为第 j 个叶子节点的权重, γ和λ为惩罚系数, 用于调整模型的复杂度.

在引入正则化项后, 用于模型优化的目标函数变为

$$obj=\sum_{i=1}^{N}L\left(y_i,\hat{f}\left(x_i\right)\right)+\Omega\left(f_m\right) \tag{14}$$

其中, $\hat{f}\left(x_i\right)=\sum_{m=1}^{M}f_m\left(x_i\right)$ 为预测的输出值, $\Omega(f_m)$为第 m 棵树的正则化项.

当算法迭代到第 m 轮时, 即求第 m 棵树时, 采用贪心的思想, 我们当前所需优化的目标函数表示如下:

$$obj=\sum_{i=1}^{N}L\left(y_i,\hat{f}_{m-1}\left(x_i\right)+f_m\left(x_i\right)\right)+\Omega\left(f_m\right) \tag{15}$$

其中, $\hat{f}_{m-1}\left(x_i\right)$ 表示现有的 m–1 棵树的最优解.

采用二阶泰勒公式

$$f\left(x+\Delta x\right)\approx f\left(x\right)+f'\left(x\right)\Delta x+\frac{1}{2}f''\left(x\right)\Delta x^2 \tag{16}$$

对该目标函数进行展开, 并定义每个样本在损失函数上的一阶导数和二阶导数

$$g_i=\frac{\partial L\left(y_i,\hat{f}_{m-1}\left(x_i\right)\right)}{\partial\hat{f}_{m-1}\left(x_i\right)} \tag{17}$$

$$h_i=\frac{\partial^2 L\left(y_i,\hat{f}_{m-1}\left(x_i\right)\right)}{\partial\hat{f}_{m-1}\left(x_i\right)} \tag{18}$$

则目标函数可以改写为

$$obj\approx\sum_{i=1}^{N}\left[L\left(y_i,\hat{f}_{m-1}\left(x_i\right)+g_i f_m\left(x_i\right)+\frac{1}{2}h_i f_m^2\left(x_i\right)\right)\right]+\Omega\left(f_m\right) \tag{19}$$

对于 XGBoost 来说, 每一个样本 x_i 都会落到一个叶子节点上, 而落在同一个叶子节点上的样本的输出都是一样的, 假设 I_j 表示所有属于叶子节点 j 的样本的索引的集合, 并将$\Omega(f_m)$展开同时省去常数项, 则目标函数可以进一步改写为

$$obj=\sum_{j=1}^{J}\left[\left(\sum_{i\in I_j}g_i\right)\omega_j+\frac{1}{2}\left(\sum_{i\in I_j}h_i\right)\omega_j^2\right]+\gamma J \tag{20}$$

对ω_j求偏导可得第 j 个叶子节点最优的权重为

$$\omega_j^*=-\frac{\sum_{i\in I_j}g_i}{\sum_{i\in I_j}h_i+\lambda} \tag{21}$$

带回目标函数可得

$$obj^m=-\frac{1}{2}\sum_{j=1}^{J}\frac{\left(\sum_{i\in I_j}g_i\right)^2}{\sum_{i\in I_j}h_i+\lambda}+\gamma J \tag{22}$$

该目标函数可用于判断第 m 棵树的结构质量, 该值越小, 则树的结构越好. 然而从所有的树结构中寻找最优的树结构是一个 NP-hard 问题, 因此在实际中往往采用贪心法来构建出一个次优的树结构, 基本思想是从根节点开始, 每次对一个叶子节点进行分裂, 针对每一种可能的分裂, 根据特定的准则选取最优的分裂. 假设 I_L 和 I_R 分别是在节点分裂后得到的左节点的样本集和右节点的样本集, $I=I_L\cup I_R$, XGBoost 的分裂准则采用分裂之后的目标函数之和与不分裂的目标函数值之间的差值:

$$obj_{spilt}=\frac{1}{2}\left[\frac{\left(\sum_{i\in I_L}g_i\right)^2}{\sum_{i\in I_L}h_i+\lambda}+\frac{\left(\sum_{i\in I_R}g_i\right)^2}{\sum_{i\in I_R}h_i+\lambda}+\frac{\left(\sum_{i\in I}g_i\right)^2}{\sum_{i\in I}h_i+\lambda}\right]-\gamma \tag{23}$$

通过最大化这个差值来进行决策树的构建, 寻找使得目标函数前后相差最大时对应的分裂方式.

XGBoost 的优点很多, 例如在损失函数里加入了正则化项, 用于控制模型的复杂度. 从偏差-方差权衡角

度来说, 正则化降低了模型的方差, 防止过拟合; 使用二阶泰勒展开式对目标函数做近似展开, 使得求最优解时变得简单; 采了与随机森林相似的策略, 支持列抽样, 能够降低过拟合, 并减少计算量; 可以处理稀疏、缺失数据, 可以自动学习出缺失值的处理策略以及可以多线程进行.

虽然 XGBoost 的优点很多, 但是它拥有的参数很多, 参数的选取对于该模型的精度和性能有较大影响, 因此本文考虑利用差分进化算法来找到最优的模型参数.

2.5 差分进化算法

差分进化算法是一种基于群体差异的智能进化算法, 由 Storn 等人首次提出[45], 其主要思想是: 首先初始化种群, 接着对种群进行交叉和变异操作来得到新的个体, 然后利用贪心策略在原个体和新个体之间进行选择, 将更优秀的个体保留到下一代. 它的算法参数较少、收敛快且简单易懂, 相比于遗传算法、粒子群算法等其他进化算法具有更优越的性能. 其主要过程如下所示.

(1) 初始化种群: 进化算法通常首先需要初始化种群, 初始化方式如下:

$$\left\{X_i(0)\middle|x_{i,j}^L \leqslant x_{i,j}(0) \leqslant x_{i,j}^U; i=1,2,\ldots,N; j=1,2,\ldots,D\right\} \tag{24}$$

$$x_{i,j}(0)=x_{i,j}^L+rand(0,1)\left(x_{i,j}^U-x_{i,j}^L\right) \tag{25}$$

其中, N 为种群中个体的数量也即种群的规模; $x_i(0)$表示第 0 代的第 i 个体; j 表示第 i 个体的第 j 维(基因); D 表示个体进行实数编码后向量的维数; $x_{i,j}^L$ 和 $x_{i,j}^U$ 分别表示第 j 维的下界和上界; $rand(0,1)$表示[0,1]上的随机数.

(2) 变异操作: 是差分进化算法最基本的操作, 其步骤是从种群中随机选择若干个体进行减法操作, 接着乘以一个缩放因子, 其取值在 0–1 之间, 然后与种群中另一个体相加, 最终得到变异的个体. 变异操作的方式很多, 最常用的一种是 *DE*/*rand*/1, 具体公式如下所示:

$$V_i(g)=X_{r_1}(g)+F\times\left(X_{r_2}(g)-X_{r_3}(g)\right) \tag{26}$$

其中, $V_i(g)$代表第 g 代的第 i 个变异个体, $r_1,r_2,r_3\in\{1,2,...,N\}$ 是互不相等且不等于 i 的随机数, F 是缩放因子, 用于对差分向量进行放大或缩小.

(3) 交叉操作: 对种群中的目标个体 $X_i(g)$和相应的变异个体 $V_i(g)$按一定的交叉概率进行交叉, 通常有两种方式, 分别为指数交叉和二项式交叉, 其中二项式交叉较为简单, 具体操作如下:

$$U_{i,j}=\begin{cases}V_{i,j}(g), & rand(0,1)\leqslant CR or j=j_{rand}\\ X_{i,j}(g), & \text{others}\end{cases} \tag{27}$$

其中, j_{rand} 为 1–D 范围内随机生成的整数, CR 为交叉概率, 取值在 0–1 之间.

(4) 选择操作: 目的是为了选择出优秀的下一代个体, 通过采用贪心策略, 比较实验个体 $U_i(g)$和目标个体 $X_i(g)$对应的适应度值的大小, 选取适应度值更优的个体作为下一代种群中的成员, 具体操作如下:

$$X_i(g+1)=\begin{cases}U_i(g), & f(U_i(g))\leqslant f(X_i(g))\\ X_i(g), & \text{others}\end{cases} \tag{28}$$

$X_i(g+1)$表示经过选择操作后得到的下一代种群个体, f 为适应度函数.

由于 XGBoost 模型存在大量的参数, 如基学习器个数、学习率、最大树深、最小叶子权重等等, 寻找最优的模型参数对任务推荐效果具有重要作用, 差分进化算法具有良好的全局搜索能力, 能够找出全局最优的 XGBoost 参数. 因此, 基于差分进化算法改进的 XGBoost 模型能够得到更加准确的评分预测, 从而获取更加优秀的任务推荐结果.

3 群智化软件任务推荐

本节首先描述了本文所解决的问题, 然后对相关概念给出了定义和参数指标, 包括待推荐的任务、开发者的任务兴趣偏好以及开发者的竞争力. 最后, 对利用本文模型向开发者推荐群智化软件任务的过程进行了详细的阐述.

3.1 问题描述

本文所要解决的问题是群智化软件任务推荐问题, 即给群智化软件开发平台中的开发者推荐一组适合其完成的任务. 如图 5 所示, 问题具体描述如下: 假设在一个群智化软件开发平台中, D 代表所有开发者组成的集合, T 代表所有报名时间未截止的任务的集合也即待推荐的任务集合, 则对于其中的一个开发者 $D_i \in D$, 根据其参与的历史任务信息, 推荐一组适合其完成的任务 $\{T_1, T_2, ..., T_K\} \in T$, 其中, K 代表推荐的任务的数量.

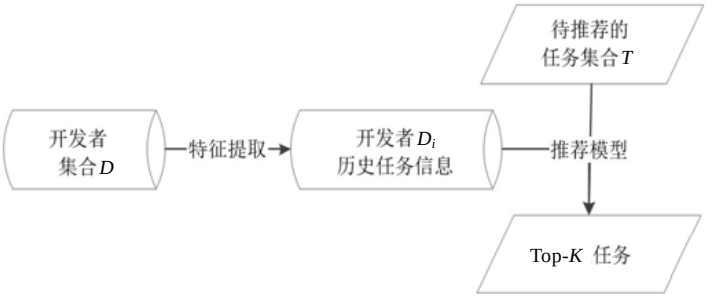


图 5 推荐问题示意图

3.2 相关定义

为了给开发者推荐合适的任务, 我们首先需要对任务和开发者进行建模. 科学合理的模型是进行有效推荐的重要保障.

目前, 对于任务的建模已经有了一些相关研究[15,46], 这些模型大同小异, 反映的是任务的基本属性. 对于每一个开发者而言, 群智化平台中的任务包括他已经参与过的历史任务以及他尚未参与的可供推荐的任务. 结合上述参考文献的做法, 我们首先对待推荐的任务进行定义.

定义 1(待推荐的任务). 假设用 T 代表平台中所有未截止的任务, 则一个待推荐的任务 $T_i \in T$ 采用一个五元组来描述, $T_i = \{T_x, T_y, T_s, T_C, T_r\}$, 其中,

- T_x 代表任务的需求描述文本;
- T_y 代表任务的类型;
- T_s 代表完成任务所需的技能集合, $T_s = \{T_{s_1}, T_{s_2}, ..., T_{s_m}\}$, m 是所需技能的个数;
- T_c 代表完成任务所拥有的开发周期;
- T_r 代表完成任务可以获得的报酬.

在对开发者进行建模时, 我们主要依据开发者的历史任务信息来进行, 对于一名群智化软件开发平台中的开发者来说, 他选择一项任务的初衷一方面是源于自身的兴趣爱好, 更多的还会考虑自己能否在有限的时间内顺利地完成该项任务并收获一定的报酬. 在更为常见的竞争性质的软件开发任务中, 能够赢得任务的人员很少, 开发人员在报名任务时通常会查看竞争对手人数及人员的相关信息, 因此他们在选择任务时会综合考虑多方面的因素.

基于此, 本文在对开发者建模时结合了两方面因素, 一方面是开发者的兴趣偏好, 这主要通过开发者历史报名过的任务的基本信息来度量, 定义如下.

定义 2(开发者的任务兴趣偏好). 开发者的任务兴趣偏好由其所报名过的历史任务信息构成, 假设历史任务集合为 P, 对于每一个历史任务 $P_i \in P$, 采用一个五元组来描述, $P_i = \{P_x, P_y, P_s, P_C, P_r\}$, 其中,

- P_x 代表任务的需求描述文本;
- P_y 代表任务的类型;
- P_s 代表完成任务所需的技能集合, $P_s = \{P_{s_1}, P_{s_2}, ..., P_{s_m}\}$, m 是所需技能的个数;
- P_c 代表完成任务所拥有的开发周期;

• P_r 代表完成任务可以获得的报酬.

另一方面, 开发者在一项任务上的竞争力分为 3 个角度来考虑, 第 1 个角度是开发者是否具备完成相关任务的经验和技能, 这主要通过开发者的历史相似任务的表现情况来度量, 第 2 个角度是开发者是否具备完成该项任务的时间, 即是否有精力来完成该任务, 这主要与开发者当前的任务参与情况有关, 第3个角度则是与开发者竞争的相关人员的情况. 综上所述, 我们对开发者的任务竞争力定义如下.

定义 3(开发者的任务竞争力). 开发者的任务竞争力由其报名、提交、获胜过的历史相似任务表现情况、当前任务参与情况以及任务的竞争对手信息构成, 对于每一个待推荐的任务 T_i, 开发者对于该项任务的竞争力 C_i 用一个多元组描述, $C_i=\{\{Ab,As,Aw,Ac,Ap\},\{En,Ec,Et,Er\},\{On,Ow,Oa,Oe\}\}$, 其中,

• {*Ab*,*As*,*Aw*,*Ac*,*Ap*}表示开发者对于该项任务的开发能力, 而

※ *Ab* 代表相似任务的报名次数;

※ *As* 代表相似任务的提交次数;

※ *Aw* 代表相似任务的获胜次数;

※ *Ac* 代表相似任务的平均所需开发周期;

※ *Ap* 代表相似任务的平均得分.

• {*En*,*Ec*,*Et*,*Er*}代表表示开发者的开发精力, 而

※ *En* 代表开发者当前报名且未提交的任务个数;

※ *Ec* 代表开发者当前报名且未提交的任务平均开发周期;

※ *Et* 代表开发者当前报名且未提交的任务平均剩余提交期限;

※ *Er* 代表开发者当前报名且未提交的任务平均报酬.

• {*On*,*Ow*,*Oa*,*Oe*}表示同时竞争该任务的竞争对手信息, 而

※ *On* 代表该任务的报名总人数;

※ *OW* 代表报名者中相似任务获胜率排名前 3 的平均获胜率;

※ *Oa* 代表报名者中相似任务获胜率排名前 3 的上述开发能力的各项平均值;

※ *Oe* 代表报名者中相似任务获胜率排名前 3 的上述开发精力的各项平均值.

3.3 模 型

本文综合考虑开发者的动态兴趣偏好和竞争力特征, 提出了一种基于注意力机制的长短期记忆神经网络(A-LSTM)与基于差分进化算法改进的极端梯度提升(DE-XGBoost)相结合的群智化软件任务推荐模型. 其中, A-LSTM 用于预测开发者的兴趣偏好来筛选出符合偏好的前 N 个任务, DE-XGBoost 基于开发者的竞争力来预测开发者完成任务后的评分, 从而进一步筛选出前 N 个任务中评分最高的 K 的任务来推荐给开发者, 推荐流程如图 6 所示, 详细步骤介绍如下.

3.3.1 数据预处理

由于在本文中涉及到的参数特征较多, 参数类型各异, 包含了文本类型和数字类型的特征且不同参数特征拥有不同的量纲, 为了避免量纲不同所导致的模型效果不佳同时简化计算, 提高模型的训练效率, 我们对所有的参数进行向量化、归一化处理, 将其映射到统一的范围之内, 具体步骤如下.

对于连续的文本类型的特征, 包括任务的需求描述文本 T_x 和 P_x, 通过删除其中连接词、停顿词, 仅保留有意义的描述性单词, 将其转换成词向量. 假设总共有 m 个词汇, 则对于每一个任务 i 需求描述文本 d_i 而言, 词向量表示为 $v_i=\{\omega_{i,1},\omega_{i,2},\ldots,\omega_{i,m}\}$, 每个 $\omega_{i,j}$ 代表第 j 个词汇 t_j 的权重, 权重通过 TF-IDF 方法[47]计算得到. TF-IDF (term frequency-inverse document frequency, 词频-逆向文件频率)是用于信息检索与文本挖掘的常用加权技术. TF 词频代表词汇(关键字)在文本中出现的频率, IDF 词频逆向文件频率则表示某一词汇出现在某些文本中频率越高, 在其他文本中较少, 则越利于文本的划分, 相关计算公式如下.

$$TF_{i,j}=\frac{n_{i,j}}{\sum_k n_{i,k}} \tag{29}$$

$$IDF_j=\log\frac{|T|}{\left|i:t_j\in d_i\right|+1} \tag{30}$$

$$\omega_{i,j}=TF_{i,j}*IDF_j \tag{31}$$

其中, $n_{i,j}$ 表示词汇 t_j 在文本 d_i 中出现的次数, $\sum_k n_{i,k}$ 表示文本 d_i 中所有词汇出现的次数 之和, $|T|$表示任务数或者说是所有需求描述文本的个数, $|i:t_j\in d_i|$表示含有词汇 t_j 的需求描述文本的个数, 为了防止分母为 0, 进行加 1 处理.

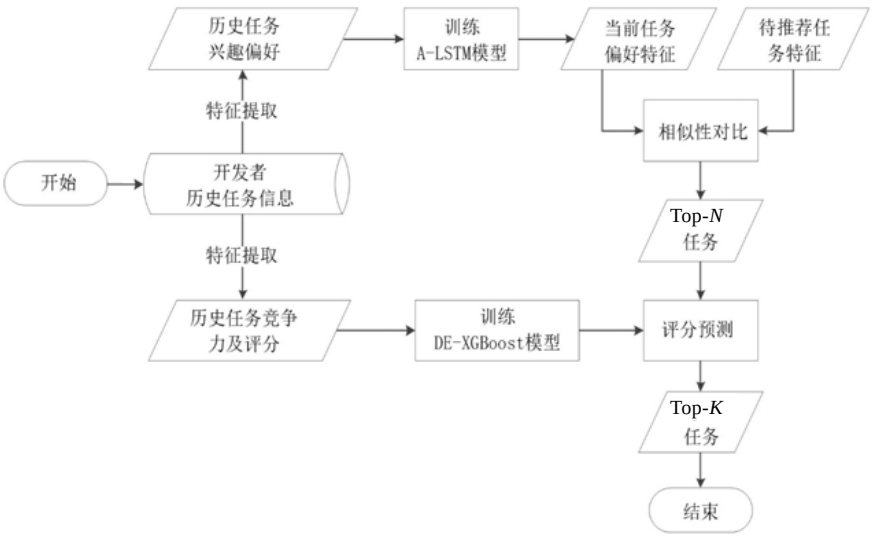


图 6 任务推荐流程

对于离散的文本类型的特征, 包括任务的类型 T_y、P_y, 采用 One-Hot 编码, 即采用 n 位状态寄存器来对 n 个状态进行编码, 每个状态都有它独立的寄存器位, 并且在任意时候只有一位有效即取值为 1, 也即假设有 n 种任务类型, 则用一个 n 维向量表示某种任务类型, 只有其中代表该任务类型的 1 位值为 1, 其余位都为 0. 相似地, 对于具有多个值的离散文本类型的特征, 包括任务所需的技能集合 T_s、P_s, 则采用 Multi-Hot 编码即有多个位的值为 1, 相应位的 1 代表了任务需要该技能.

对于数字类型的特征, 包括任务的开发周期 T_c、T_r, 任务的报酬 P_c、P_r 以及开发者任务竞争力的所有特征, 则采用最大最小值归一化方法, 将所有特征都映射到[0,1]之间. 所利用的公式如下:

$$D^*=\frac{D-D^{\min}}{D^{\max}-D^{\min}} \tag{32}$$

其中, D 代表相应数字特征的原始值, D^*代表相应数字特征归一化后的值, $D^{\max}$ 代表相应数字特征的最大值, $D^{\min}$ 代表相应数字特征的最小值.

3.3.2 开发者动态偏好预测与 Top-N 任务筛选

开发者的兴趣偏好通常不会是一成不变的, 他们的兴趣爱好可能会依据平台的变化以及自己参与过的任务的情况来改变, 为了捕捉到开发者兴趣偏好变化的规律, 我们依据开发者短期内参与的任务序列预测出他当前的兴趣偏好特征. 基于此, 我们首先给出开发者任务兴趣偏好序列的定义.

定义 4(开发者的任务兴趣偏好序列). 假设一个开发者所报名过的所有任务的数量为 N, 我们将其按照时间先后顺序以一定的时间步分为若干个大小相等的任务集合, 每一个任务集合中的任务信息按时间先后顺序所组成的序列即为开发者任务兴趣偏好序列, 记为

$$\vec{P}=\{P_1,P_2,...,P_s\} \tag{33}$$

其中, P_i 为开发者所报名的一项任务的基本信息, 如定义 2 所示, s 为任务兴趣偏好序列中任务的数量, 也即为时间步的大小.

通过开发者的历史任务兴趣偏好序列, 我们来预测下一个时间步开发者可能报名的任务兴趣偏好特征. 这是一个典型的时序数据预测问题, 在这里, 我们采用基于注意力机制的长短期记忆神经网络(A-LSTM)进行兴趣偏好的预测. LSTM 拥有独特的门控结构, 可以有效地控制历史信息的去留, 在解决时序数据方面有很大的优势. 同时, 开发者当前的任务兴趣往往只与近期参与的某些任务有关, 而与其他任务无关, 因此我们在 LSTM 上引入了注意力机制, 通过在 LSTM 的每一个隐藏层状态加上一个权重, 评估其对当前任务偏好的影响, 并通过对所有隐藏层状态进行加权组合, 从而生成更加准确的动态偏好特征, 降低开发者偶然兴趣偏好的干扰. 引入注意力机制的长短期记忆神经网络结构图如图 7 所示.

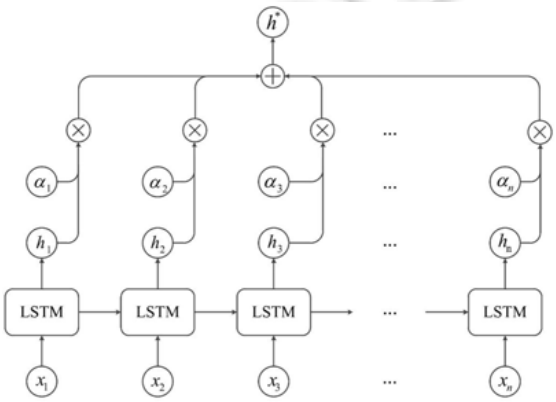


图 7 基于注意力机制的长短期记忆神经网络结构图

利用 A-LSTM 进行开发者当前兴趣偏好预测的具体步骤描述如下.

(1) 将数据集按照某个时间点 p 划分为开发者历史参与的任务即训练数据集与待推荐给开发者的任务即测试数据集两个集合;

(2) 从开发者历史参与的任务中根据定义 2 提取出开发者的任务兴趣偏好特征, 并将其与待推荐的任务特征即定义 1 都依据第 3.3.1 节所示的方式进行向量化、归一化的预处理, 避免量纲不同对模型训练造成的影响;

(3) 将开发者的任务兴趣偏好特征向量按照时间顺序划分为多个样本即多个开发者任务兴趣偏好序列(依据定义 4)作为 A-LSTM 的输入数据, 并将当前实际的下一个时间步的开发者的任务兴趣偏好特征向量, 作为 A-LSTM 的标签数据;

(4) 对 A-LSTM 进行训练, 在模型的最后一层选用 sigmoid 函数, 以获得最后的输出.

$$\hat{y}=sigmoid\left(W_y h^*+b_y\right)=\frac{1}{1+\exp\left(W_y h^*+b_y\right)} \tag{34}$$

其中, h^*为加权的隐藏层输出; W_y 为隐藏层到输出层的权重; b_y 为隐藏层到输出层的偏置. 模型的损失函数选用平方误差损失函数, 定义为

$$L=\frac{1}{n}\frac{1}{d}\sum_{i=1}^{n}\sum_{j=1}^{d}\left(y_{i,j}-\hat{y}_{i,j}\right)^2 \tag{35}$$

其中, n 为样本的数量; d 代表标签数据的维度; $y_{i,j}$ 代表第 i 个样本的第 j 维标签值, $\hat{y}_{i,j}$ 代表第 i 个样本的第 j 维预测值. 详细的训练算法见算法 2.

算法 2. A-LSTM 模型训练算法.

输入: 训练数据集: 经过数据预处理后的开发者任务兴趣偏好序列及其相应标签数据;

输出: 梯度提升树 $f(x)$.

1: begin

2: 初始化 A-LSTM 模型的所有权重和偏置在(0,1)之间

3: 迭代次数 epoch=1

4: repeat

5: for i=1,2,…,n do, n 为样本个数

6: 利用公式(1)–(6)进行 LSTM 网络的前向传播

7: 利用公式(7)–(9)通过注意力机制计算出加权的隐藏层输出 h^* 并利用公式(34)获得最终的输出向量, 即预测出的开发者当前的任务兴趣偏好特征向量

8: 利用公式(35)计算模型的损失函数 L

9: 利用基于时间的反向传播算法 BPTT 更新模型的所有权重和偏置

10: endfor

11: epoch=epoch+1

12: until 达到收敛条件, 即迭代次数 epoch 达到预设值或者损失函数值 L<预先设定阈值

13: end

(5) 通过训练好的 A-LSTM 神经网络, 以时间点 p 之后的数量为时间步长度 s 大小的每个开发者任务偏好兴趣偏好序列作为该网络的输入值, 预测出该开发者当前的任务兴趣偏好特征向量.

通过 A-LSTM 获取到某个开发者当前的兴趣偏好特征向量之后, 我们对所有待推荐的任务进行第 1 阶段的筛选, 筛选的标准采用相似性度量的方式, 度量公式基于欧几里得距离公式, 公式如下:

$$d\left(P^*,T_i\right)=\sqrt{\sum_{j=1}^{D}\left(P_j^*-T_{i,j}\right)^2} \tag{36}$$

$$Sim\left(P^*,T_i\right)=\frac{1}{1+d\left(P^*,T_i\right)} \tag{37}$$

其中, P_j^* 为预测出的某个开发者当前的兴趣偏好特征向量的第 j 维, $T_{i,j}$ 为第 i 个待推荐任务 T_i 的特征的第 j 维, Sim 值越大代表相似度越高, 依据相似度从高到低我们从所有待推荐的任务中筛选出相似度排名前 N 的任务, 作为第 1 阶段推荐的结果.

3.3.3 开发者任务评分预测与 Top-K 任务推荐

为了进一步提高推荐的准确性, 能够把开发者更容易获胜且完成结果质量可能较高的任务推荐给开发者, 我们基于定义 3 的开发者任务竞争力, 来预测其在待推荐任务上提交结果并经过专家评审后可以得到的评分(评分越高, 开发者获胜的概率就越高, 且完成的质量也更好, 以实现开发者和需求者的双赢), 从而进一步筛选出相似度排名前 N 的任务中预测评分最高的 K 的任务来推荐给开发者.

XGBoost 模型近年来被广泛运用在 Kaggle 竞赛和许多其他机器学习竞赛中并取得了很好的成绩, 而且很适合解决评分预测的回归问题, 但是它的缺点在于拥有的参数很多, 关键参数的选取会对模型的性能有较大影响, 人工调整参数一般取决于主观判断, 工作量大且准确度较低, 因此本文通过简单且性能优异的智能进化算法——差分进化算法来找到 XGBoost 模型的最佳参数, 从而优化模型的性能, 提高推荐的效果.

XGBoost 模型中主要的参数见表 1, 我们主要针对几个对模型性能影响较大的参数进行调优, 分别为基学习器个数(n_estimators)、学习率(learning_rate)、最大树深(max_depth)、最小叶子权重(min_child_weight)、每棵树随机采样比例(subsample)和列随机采样比例(colsample_bytree), 其余参数设置为默认值.

表 1 XGBoost 模型的主要参数

参数名称	默认值	取值范围	说明
n_estimators	–	–	基学习器个数即树的个数, 该值需要事先指定
learning_rate	0.3	[0,1]	学习率, 用于控制每一轮树的权重
max_depth	6	[0,+∞]	最大树深, 该值过大时容易陷入局部最优, 导致过拟合
min_child_weight	1	[0,+∞]	最小叶子权重, 该值过小时会过拟合, 值过高时则会欠拟合
gamma	0	[0,+∞]	用于控制节点的分裂, 规定了节点分裂所需的最小损失函数下降值, 该值越大算法越不容易过拟合
subsample	1	(0,1]	控制每棵树随机采样的比例, 减小该值可以避免过拟合
colsample_bytree	1	(0,1]	控制每棵树随机采样的列数的比例, 每一列即代表一个特征, 值太小会欠拟合
colsample_bylevel	1	(0,1]	控制每棵树每次分裂节点时随机采样的列数的比例
lamda	1	–	$L2$ 正则化项的惩罚系数, 越大模型越不容易过拟合
alpha	0	–	$L1$ 正则化项的惩罚系数, 越大模型越不容易过拟合
…	…	…	…

我们将这些参数的上下界分别设置为 n_estimators: [50,500]、learning_rate: (0,1]、max_depth: [3,15]、min_child_weight: [0,20]、subsample: [0.5,1]、colsample_bytree: [0.5,1], 并利用差分进化算法进行调优, 其中差分进化算法的参数根据经验分别设置为种群规模 100、缩放因子 F 为 0. 5, 交叉概率 CR 为 0. 5, 最大进化代数为 500, 差分进化算法所需优化的目标函数为 XGBoost 模型的评价指标, 由于是回归问题, 因此采用均方根误差(root mean square error, RMSE)这一指标, 计算公式如下, 预定精度为 0.04, 调优流程图如图 8 所示.

$$RMSE=\sqrt{\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2} \tag{38}$$

其中, n 代表验证集中样本的个数, 也即预测的次数, y_i 代表第 i 次预测的真实评分值, $\hat{y}_i$ 代表第 i 次预测的预测评分值.

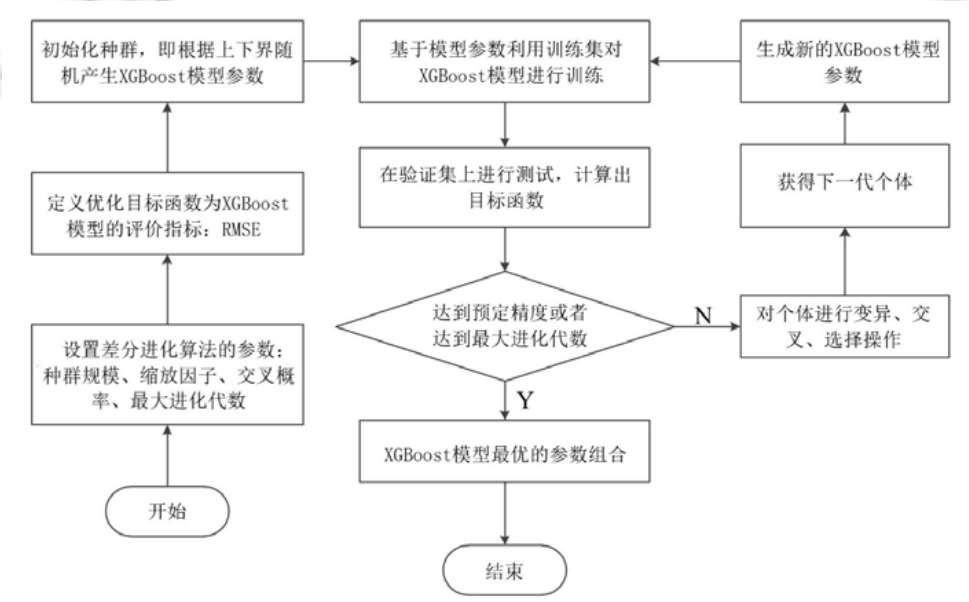


图 8 基于差分进化算法改进的 XGboost 模型参数寻优流程图

算法 3. XGBoost 模型训练算法.

输入: 训练数据集 $D=\left\{\left(x_1,y_1\right),\left(x_2,y_2\right),...,\left(x_N,y_N\right),x_i\in X\subseteq R^d\right\}$, 为开发者的历史任务竞争力特征, $y_i\in Y\subseteq R^d$ 为开发者在任务上的评分; 损失函数为 $L(y,f(x))$;

输出: XGBoost 模型 $\hat{f}\left(x\right)$.

1: begin

2: 初始化 $f_0(x)=\arg\min_c\left(\sum_{i=1}^{N}L(y_i,c)+\Omega(f_1)\right)$, 其中 c 为常数

3: for m=1,…,M do, M 为树的个数

4: for i=1,…,N do

5: 利用公式(17)和(18)计算损失函数的一阶导数和二阶导数

6: endfor

7: 利用分裂准则公式(23)拟合一个回归树, 得到第 m 棵树的叶子节点区域 R_{mj}, $j=1,2,\ldots,J$

8: for j=1,2,…,J do

9: 利用公式(35)计算每一个叶子节点区域的最优权重 ω_{mj}

10: endfor

11: 更新第 m 棵树, 其中 $I()$为指示函数, 当括号内的式子成立时返回 1, 否则返回 0

12: $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}\omega_{mj}I\left(x\in R_{mj}\right)$

13: endfor

14: 得到 XGBoost 模型 $\hat{f}_m(x)=\sum_{m=1}^{M}f_m(x)=\sum_{m=1}^{M}\sum_{j=1}^{J}\omega_{mj}I\left(x\in R_{mj}\right)$

15: end

利用基于差分进化算法改进的 XGBoost 模型(DE-XGBoost)进行开发者任务评分预测的具体步骤如下.

(1) 针对上一阶段划分过的数据集, 我们从训练数据集中每一个开发者历史参与的任务中依据定义 3 提取出开发者对于该项任务的竞争力, 其中相似任务的计算方式同公式(37), 并进行向量化、归一化的预处理, 其中为了进行模型参数调优, 我们将训练数据集均等地分为 k 份, 采用 k 折交叉验证的方式来得到最佳的模型参数;

(2) 将开发者历史参与的任务竞争力特征作为 XGBoost 模型的输入数据, 而该项任务开发者实际获得的提交结果的评分作为标签数据;

(3) 按照算法 3 对 XGBoost 模型进行训练并利用图 8 所示流程进行参数调优, 其中 XGBoost 模型中损失函数采用平方误差损失函数, 如下所示:

$$L=\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2 \tag{39}$$

其中, n 为样本的个数, y_i 代表第 i 次预测的真实评分值, $\hat{y}_i$ 代表第 i 次预测的预测评分值.

(4) 利用最终训练好的 DE-XGBoost 模型, 对上一阶段所筛选出的 N 个任务, 对于每一个开发者, 提取开发者的任务竞争力作为输入, 预测出开发者在 N 个任务上的评分, 其中排名前 K 的任务作为最终的推荐结果.

4 实验与分析

为了对本文提出的群智化软件任务推荐模型的性能进行评价, 本文爬取了 TopCoder 平台上真实的开发者和任务的数据集进行实验, 观察该模型在实际场景下的性能表现并探索不同参数对推荐性能的影响. 同时, 为了说明所提出模型的显著性, 本文将该模型与其他几种常用于推荐的模型进行比较. 实验结果证明, 本文模型在群智化软件任务推荐方面比其他方法更具优势.

4.1 数据集

关于进行实验所采用的数据集, 我们爬取了典型的群智化软件开发平台 TopCoder 的数据进行实验. 数据集包括了从 2018 年 1 月 1 日–2020 年 12 月 31 日的开发者和任务信息. 在使用数据集之前, 我们对其进行了数据清洗, 清洗的方式如下.

(1) 将其中任务状态为“Cancelled-No Submissions”的任务进行删除, 这类任务因为没有任何人提交结果被取消, 任务信息不完整, 因此将其从数据集中删除;

(2) 将任务信息中存在异常数据的任务删除, 异常情况包括提交者信息不在报名者之列、任务发布日期超越当前实际日期、任务奖金数额异常等等;

(3) 为防止数据稀疏, 我们将参与任务次数小于等于 20 或者从未有过提交结果的开发者从数据中删除.

最终我们得到了总计 1 134 个开发者的信息和总计 5 410 个已截止且正常完成的任务信息, 其中开发者-任务参与(报名、提交)关系 91 234 条, 对于每一个开发者, 他所拥有的任务参与记录信息样例见表 2, 而每一个任务具体包含的信息样例见表 3.

我们将数据集按照时间先后顺序划分为训练数据集和测试数据集, 其中后 4 个月的开发者和任务信息作为测试数据集, 剩余的数据作为训练数据集. 我们从这些信息中提取出本文所定义的待推荐任务特征、开发者的任务兴趣偏好特征和任务竞争力特征, 其中任务特征中所拥有的开发周期通过任务提交截止日期与任务发布日期的差值来计算, 竞争力特征中相似任务的判断通过公式(37)来计算, 任务平均剩余提交期限由任务提交截止日期减去当前日期, 开发者是否获胜通过其最终获得名次是否小于等于任务提供的奖励个数来判断, 提取出特征后我们按照第 3.3.1 节所示进行预处理, 通过上述数据我们完成以下的实验来验证本文所提出模型的有效性.

表 2 TopCoder 开发者信息样例

Handle (开发者 Id)	Task_Id (任务 Id)	RegistrationDate (报名日期)	SubmissionDate (提交日期)	Placement (名次)	Score (评分)
creeya	30108353	2019-11-29	2019-11-30	4	70.0
	30106385	2019-11-19	2019-11-20	3	80.0
	30106924	2019-11-17	2019-11-18	2	90.0
	…	…	…	…	…

表 3 TopCoder 任务信息样例

参数名称	说明	样例
TaskId	任务 Id	30108600
Name	任务名称	[96hr] Benefits CRM prototype part 1
DetailedRequirements	任务需求描述	Build the prototype in angular 8 from given .sketch design file$\cdots$
Type	任务类型	DEVELOP
Technologies	任务所需技术列表	[Angular 2+,SCSS,HTML,JavaScript]
Status	任务当前状态	Completed
PostingDate	任务发布日期	2019-11-30
RegistrationEndDate	任务报名截止日期	2019-12-04
SubmissionEndDate	任务提交截止日期	2019-12-04
Prizes	任务能够获得报酬	[1400.0,700.0]
NumRegistrants	报名者数量	49
Registrants	报名者报名信息, 包括 报名者 Id 报名日期	[{ Handle: torghul, RegistrationDate: 2019-11-30}, {…}]
NumSubmitters	提交者数量	2
Submitters	提交者提交信息, 包括 提交者 Id 提交日期 获得名次 获得评分	[{ Handle: ronakkaria SubmissionDate: 2019-12-04, Placement: 1, Score: 96.33333}]

4.2 实验设置

本文中实验运行环境的主要参数为: CPU 型号为 Intel(R) Core(TM) i7-6700 CPU @ 3.41 GHz, CPU 拥有的内存为 8 GB, 操作系统版本为 Windows 10 64 位, 实验代码采用 Java 语言, Jdk 1.8 实现.

本文中使用的模型包含了基于注意力机制的长短期记忆神经网络(A-LSTM)与基于差分进化算法改进的极端梯度提升(DE-XGBoost), 这两种模型中包含了许多超参数, 超参数的设置会对模型的性能有一定的影响, 其中, DE-XGBoost 的参数调优使用的即是本文所利用的差分进化算法, 而 A-LSTM 中主要需要调整的超参数是隐藏层节点数和学习率, 我们通过网格搜索(grid search)的方式来找到最佳的参数. 最终得到的最优参数见

表 4. 其他一些和评价指标及特征有关的参数, 在没有其他特别说明的情况下, 分别取推荐列表长度 K=5, 开发者任务兴趣偏好序列时间步大小 s=6, 竞争力特征中相似任务判断的相似度阈值采用 0.8.

表 4　A-LSTM 以及 DE-XGBoost 模型的参数设置

参数	数值
A-LSTM 输入层节点数	265
A-LSTM 隐藏层节点数	128
A-LSTM 输出层节点数	265
A-LSTM 学习率	0.01
A-LSTM Dropout	0.5
A-LSTM 迭代次数	500
Top-N 中的 N	20
DE-XGBoost 基学习器个数	100
DE-XGBoost 学习率	0.8
DE-XGBoost 最大树深	10
DE-XGBoost 最小叶子权重	0.6
DE-XGBoost 每棵树随机采样比例	0.8
DE-XGBoost 每棵树列随机采样比例	0.8

4.3 评估指标

本文是一个典型的 Top-K 推荐问题, 准确率(precision)、召回率(recall)与 $F1$ 值($F1$-score)是 Top-K 推荐系统中主要的评价指标, 因此, 我们首先利用这 3 个指标来判断开发者对于我们给他推荐的任务其参与度是否足够高, 计算方式如下:

$$Precision(k)=\frac{1}{|U|}\sum_{u\in U}\frac{|R_u\cap L_u|}{|L_u|} \tag{40}$$

$$Recall(k)=\frac{1}{|U|}\sum_{u\in U}\frac{|R_u\cap L_u|}{|L_u|} \tag{41}$$

$$F1(K)=2\cdot\frac{Precision(K)\cdot Recall(K)}{Precision(K)+Recall(K)} \tag{42}$$

其中, U 代表测试集中需要进行推荐的开发者集合, R_u 代表测试集中开发者 u 报名的任务列表, L_u 代表根据训练集为开发者 u 推荐的任务列表.

另外, 准确率、召回率和 $F1$ 值是一种与推荐顺序无关的 Top-K 推荐评价指标, 在本文中我们的模型还希望推荐列表越靠前的任务开发者越能够有效完成, 获得的评分能够越高, 所以我们还选用了一种广泛应用的与顺序相关的 Top-K 评价指标, 即归一化折损累计增益(normalized discounted cumulative gain, NDCG)[48,49], 其计算方式如下:

$$DCG_u(K)=\sum_{p=1}^{K}\frac{2^{S(u,p)}-1}{\log(p+1)} \tag{43}$$

$$NDCG(K)=\frac{1}{|U|}\sum_{u\in U}\frac{DCG_u(k)}{IDCG_u(k)} \tag{44}$$

其中, $S(u,p)$是开发者 u 在推荐列表中第 p 个任务上获得的评分, 若开发者未参与或完成该任务, 则令该值为 0, $IDCG_u(K)$用于进行归一化, 是 $DCG_u(K)$理想情况下的最大值.

4.4 实验结果分析

本文中使用的推荐模型是 A-LSTM 和 DE-XGBoost, 我们将其简称为 A-LSTM+DE-XGBoost. 首先, 为了验证推荐模型中各个模块的有效性, 我们将其和 LSTM、A-LSTM、LSTM+XGBoost 和 A-LSTM+XGBoost 进行比较, 其中, LSTM 和 A-LSTM 代表直接使用第 1 阶段筛选后的任务中排名前K的结果作为最后的推荐结果, 同时对推荐列表长度、数据集规模、偏好序列特征中时间步长度以及竞争力特征中相似任务阈值对推荐结果的影响进行了实验. 之后, 我们将本文的模型和其他一些常见的推荐方法进行比较, 以说明模型在推荐

效果上的显著性.

4.4.1 对比实验

为了客观地分析本文所提出推荐模型的有效性, 我们将以往文献中用于推荐的几种经典方法与本文所提出的模型进行对比. 它们分别为: 基于内容的推荐方法、基于用户的协同过滤推荐方法和基于物品的协同过滤推荐方法. 具体介绍如下.

(1) 基于内容的推荐方法(content-based recommendation): 后面简称为 CB, 通过开发者所有参与过的历史任务特征向量化取平均值得到开发者的特征, 并与所有待推荐任务特征进行相似性对比, 将相似性最高的任务推荐给开发者, 其中任务特征采用 TF-IDF 处理后的任务需求描述, 相似性计算公式采用余弦相似度, 如公式(45)所示.

$$Sim\left(\vec{D},\vec{T}\right)=\frac{\vec{D}\cdot\vec{T}}{\left\|\vec{D}\right\|\left\|\vec{T}\right\|} \tag{45}$$

其中, $\vec{D}$ 代表开发者的特征向量, $\vec{T}$ 代表任务的特征向量.

(2) 基于用户的协同过滤推荐方法(user-based collaborative filtering recommendation): 后面简称为 User-CF, 本文中用户指代开发者, 通过找到和目标开发者任务兴趣最相似的开发者集合, 找到这个集合中开发者感兴趣的且目标开发者尚未参加的任务推荐给目标开发者, 开发者之间的相似度计算方式如公式(46)所示.

$$Sim(u,v)=\frac{\left|N(u)\cap N(v)\right|}{\left|N(u)\cup N(v)\right|} \tag{46}$$

其中, $N(u)$代表开发者 u 参加过的任务集合, $N(v)$代表开发者 v 参加过的任务集合.

(3) 基于物品的协同过滤推荐方法(item-based collaborative filtering recommendation): 后面简称为 Item-CF, 本文中物品指代任务, 通过根据所有开发者的历史任务参与数据计算任务之间的相似性, 然后把与目标开发者参与过的任务相类似的任务推荐给目标开发者, 任务间的相似性计算方式如公式(47)所示.

$$Sim(i,j)=\frac{\left|N(i)\cap N(j)\right|}{\sqrt{\left|N(i)\cup N(j)\right|}} \tag{47}$$

其中, $N(i)$代表参加过任务 i 的开发者集合, $N(j)$代表参加过任务 j 的开发者集合.

对比实验的结果如图 9 所示.

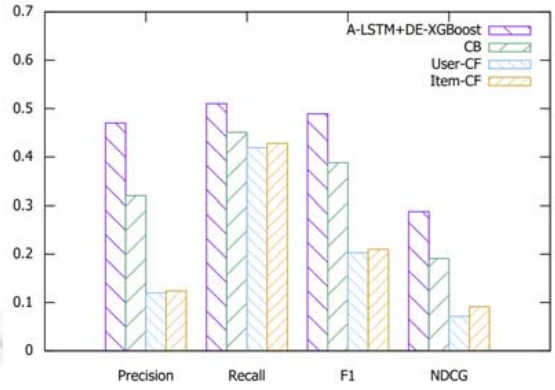


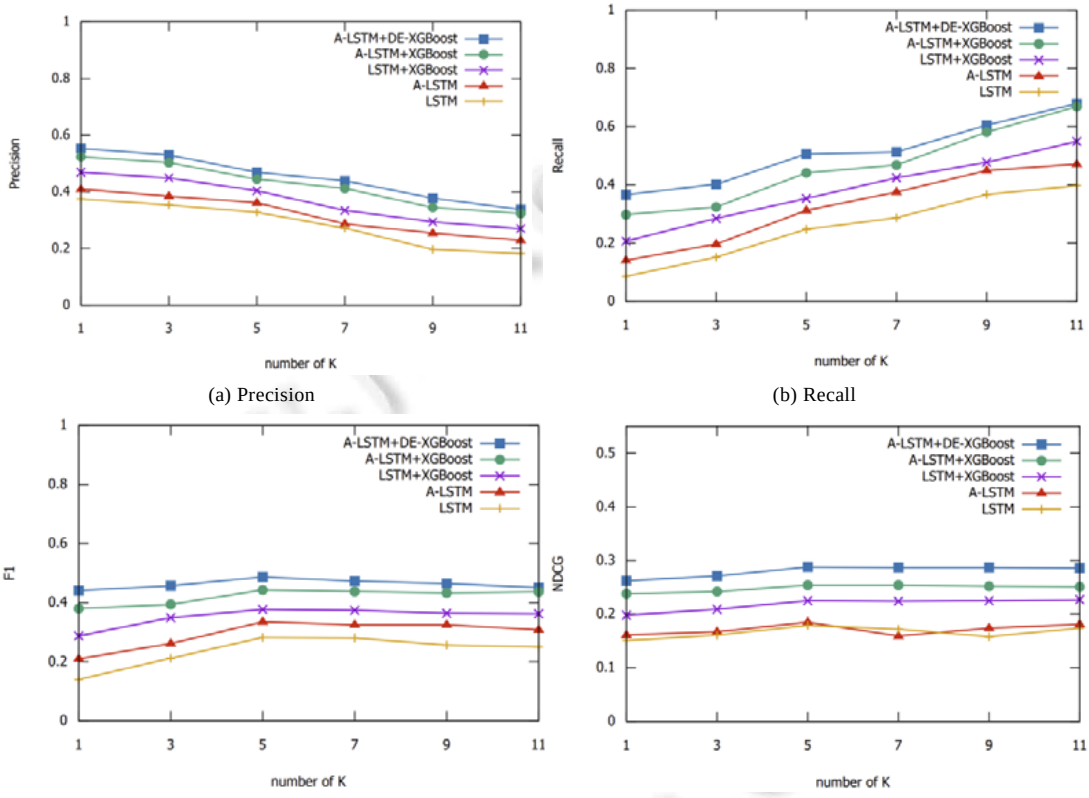
图 9 对比实验

从图中的结果可以看出, 两种协同过滤方法即 Item-CF 和 User-CF 的推荐表现不是很好, 因为这两个方法没有利用任何的任务特征信息, 是纯粹的协同过滤方法, 而由于群智化平台中的任务的生命周期较短, 所以形成的开发者协同和任务协同信息较少, 从而导致推荐效果较差. 相比之下, 基于内容的推荐算法采用了任

务特征，一定程度上提升了推荐性能，但是由于采用的特征是开发者长期参与的平均任务特征，没有捕捉到开发者的近期兴趣偏好，且没有考虑到开发者对于一项任务的竞争力，因此效果低于本文所提出的模型.

4.4.2 推荐列表长度对推荐结果的影响

Top-*K* 问题首先需要考虑的就是推荐列表长度 *K* 对模型推荐效果的影响，在本节实验中，我们将 *K* 分别取 1、3、5、7、9、11，并将本文提出的模型和 LSTM、A-LSTM、LSTM+XGBoost、A-LSTM+XGBoost 进行对比，查看他们在 Precision、Recall、*F*1 和 NDCG 指标上推荐效果的表现. 实验结果如图 10 所示.



(a) Precision (b) Recall

(c) *F*1-score (d) NDCG

图 10 推荐列表长度对推荐结果影响

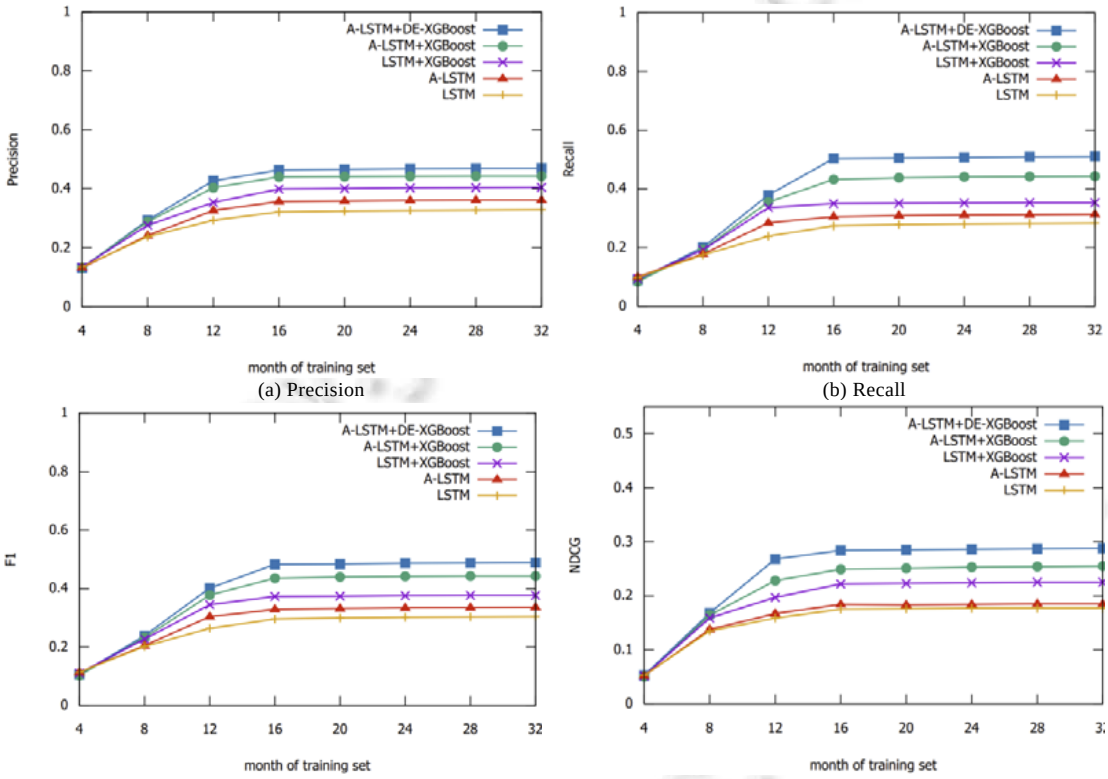
从图 10 可以看出，推荐列表长度 K 对不同的评价指标具有不同的影响. 各方法中 Precision 随着推荐列表长度的增加而降低(如图 10(a)所示)，Recall 随着推荐列表长度的增加而提高(如图 10(b)所示). 这主要是因为 Precision 反映的是推荐的精度，而 Recall 反映的是推荐的是否齐全. 随着推荐列表长度的增加，开发者实际未参加的任务的比例会上升，从而 Precision 会下降，但是推荐的范围更广，因而找全所有开发者参加的任务概率就会上升，所以 Recall 会增加. 而从综合 Precision 和 Recall 指标的 *F*1 值的结果来看，在图 10(c)中，*F*1 值先上升后下降，在推荐列表长度为 5 左右达到最大值，这说明了此时推荐效果最好，在不断下降的 Precision 和不断上升的 Recall 之间达到了最佳的平衡点. 而从与排序顺序有关的 NDCG 指标(如图 10(d)所示)来看，除 LSTM 和 A-LSTM 以外，其他的算法都比较稳定，说明了这些算法无论在推荐列表长度为多少时都能将评分靠前的任务排到列表的前面，而 LSTM 和 A-LSTM 因为未考虑到评分因素，所以导致任务的排序相对混乱，因而导致 NDCG 有所波动.

此外，从不同方法的横向比较来看，包含有注意力机制模块的相关模型相比于没有注意力机制的模型相关指标要更高，同时，包含有 XGBoost 模块的模型比起没有的指标也更高，而拥有 DE 算法的模型指标则更加优秀，这说明了无论是注意力机制对于开发者近期兴趣中与当前有关兴趣的重点关注，还是 XGBoost 模型对

于开发者在任务上评分的预测即对于开发者任务竞争力的度量以及差分进化算法对于 XGBoost 模型参数的优化，都对推荐的性能起到了一定的帮助作用，有利于提高推荐的效果.

4.4.3 数据集规模对推荐结果的影响

为了研究数据集规模对于模型推荐效果的影响，本文将之前所介绍的训练数据集按照距离测试数据集时间的远近分成离测试集数据 4 个月、8 个月、12 个月、16 个月、20 个月、24 个月、28 个月和 32 个月数据的子集，并将本文的模型和 LSTM、A-LSTM、LSTM+XGBoost、A-LSTM+XGBoost 进行比较，以观察不同模型在不同规模数据集下的表现. 实验结果如图 11 所示.



(a) Precision (b) Recall

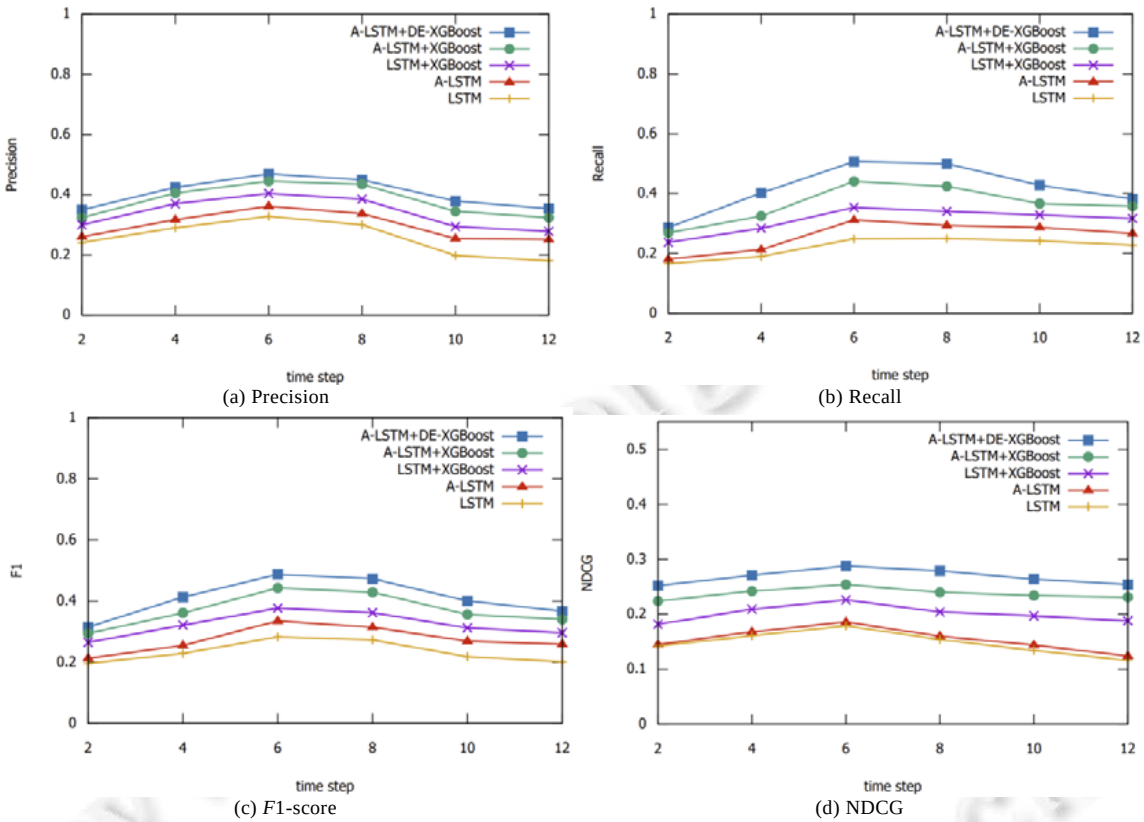
(c) *F*1-score (d) NDCG

图 11 数据集规模对推荐结果的影响

从图中的结果可以看出，当数据集规模较小时，不同算法的各个指标结果都比较低，意味着算法的推荐结果都不是很理想，这主要是因为模型中包含了深度神经网络，模型的训练依赖于大量的数据，而随着数据集规模的增加，所有的算法的各项指标都有了明显的增加. 当数据集规模达到 16 个月时，各算法都达到了收敛的状态，相应的评价指标都基本不再变化，这意味着样本的数量已经足够巨大，模型已经有了充分的数据进行训练. 同时，从收敛后的结果可以看出，本文的模型相比于其他几个算法的性能要更加优秀. 本节实验表明，训练集规模会对模型的推荐效果造成影响，我们可以通过提高样本的数量来提高模型的准确性. 但是，考虑到样本数量的增加会增大模型训练的时间，因此我们在保证模型效果的同时要选择合适的训练集规模，以达到训练效果和训练时间之间的平衡.

4.4.4 偏好序列特征中时间步长度对推荐结果的影响

在对开发者的兴趣偏好特征即定义 4 进行预测时，本文采用了长短期记忆神经网络，其中通过开发者的近期参与任务数据来预测其当前偏好特征，在这之中包含了时间步参数即定义 4 中的 s 的选取. 为了分析该参数对推荐结果的影响，本节分别选取时间步长度为 2、4、6、8、10、12 进行实验，实验结果如图 12 所示.



(a) Precision (b) Recall

(c) *F*1-score (d) NDCG

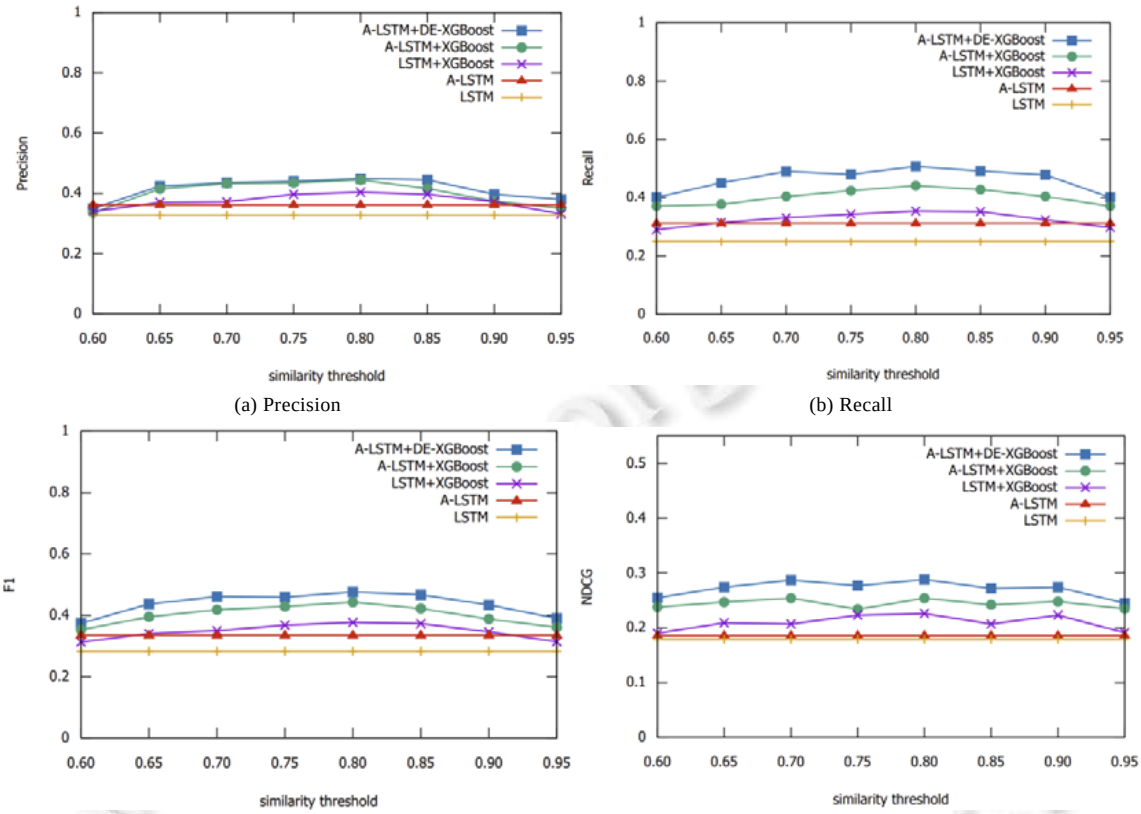
图 12 时间步长度对推荐结果的影响

从图中结果可以看出，时间步长度会对推荐结果造成一定的影响. 一开始，随着时间步长度的增加，所有算法的各指标都呈现上升趋势，这代表推荐的效果得到了提高，这主要是因为该值越大，模型对开发者近期的任务偏好特征考虑越全面，但是当时间步到达 6 以后，各指标却有所下降，经过对数据集进行研究分析发现，随着时间步长度的增加，将数据集分割后产生的偏好序列数量在下降，这就导致了训练样本数的减少，从而导致模型产生了偏差继而引起推荐精度的下降. 另外，从横向对比来看，本文的模型在不同的时间步长度下依然要优于其他几个算法.

4.4.5 竞争力特征中相似任务阈值对推荐结果的影响

在构建开发者的竞争力即定义 3 时，为了度量其开发能力，我们考虑其报名、提交、获胜过的历史相似任务表现情况，其中涉及到当前任务与开发者历史任务的相似度判断，计算方式为公式(37)，其中相似性阈值的选取可能会对推荐结果造成影响. 为了研究该参数所带来的影响，本节分别选取相似度阈值为 0.60、0.65、0.70、0.75、0.80、0.85、0.90、0.95 进行实验，实验结果如图 13 所示.

首先，由于 LSTM 和 A-LSTM 两种方法中并没有涉及到竞争力特征，也就没有相似度阈值参数，因此该参数的变化并不会对这两种方法的各指标造成影响. 而对于另外 3 种方法即 LSTM+XGBoost、A-LSTM+XGBoost 和 A-LSTM+DE-XGBoost 来说，从图 13(a)–图 13(c)来看，相似度阈值对于 Precision、Recall、*F*1 这 3 个指标的影响在于当阈值过高或过低时，推荐效果都会变得较差，甚至会低于 LSTM 和 A-LSTM 方法，这可能是因为阈值过高或过低会影响相似任务的判断，从而过高或者过低地估计了开发者对于一项任务的开发能力，进而影响了对于开发者任务竞争力的度量，导致推荐精度的下降. 另外，从图 13(d)来看，NDCG 指标随着相似度阈值的变化呈现出不断波动的趋势，这可能由于开发能力特征即相似任务的不同估计，导致最后预测的评分不断变化，任务评分的排序就会相对混乱，所以相应的指标就会时好时坏，最后显示上下波动的态势.



(a) Precision (b) Recall

(c) *F*1-score (d) NDCG

图 13 相似度阈值对推荐结果的影响

5 局限性讨论

本文提出的方法有一定的局限性. 首先, 本文不考虑在推荐系统中经常遇到的新开发人员的数据稀疏性和冷启动问题, 并影响实验结果. 冷启动问题是如何在没有任何交互数据的情况下向新开发人员推荐任务. 对于新开发人员, 有必要定义初始特征和获取规则. 因此, 采用获取最近邻作为新用户的初始值来完成推荐的方法[50], 提高了推荐的可扩展性, 但如何提高对新用户特征的准确描述一直是研究的热点; 使用用户偏好问卷和相似度计算可以得到更能反映个人主观偏好的推荐结果[51]. 但是, 问卷需要很多专家的努力, 而且过多的操作会降低用户体验. 今后我们将进一步研究冷启动问题. 其次, 在对开发人员进行建模时, 本文主要考虑了开发人员的兴趣偏好和竞争力, 但还有其他因素, 如开发人员的个性, 可能会影响开发人员对任务的选择, 在今后的工作中可以考虑将其他特征或参数引入进来, 从而改善模型的性能. 最后, 本文的推荐主要是针对竞争性质的群智化平台的, 在这之中开发者之间存在着竞争性, 但是还有另外一种协作式的群智化开发方式, 如何对于协作式的软件开发方式进行合理的推荐同样是值得去研究的.

6 总 结

随着软件系统的日益复杂以及规模的增长, 基于互联网群体智能的软件开发方式, 即群智化软件开发受到了学术界和工业界的广泛关注, 并成为了当前软件工程领域研究的热点. 该种开发方式通过汇集互联网上大量分布在世界各地的开发者资源, 充分利用集体的智慧, 有效地解决复杂的软件开发任务, 在提高软件开发效率的同时降低了软件开发的成本. 然而, 随着这种开发方式的盛行, 这其中也产生了一些问题. 群智化平台上大量的需求方任务使得开发者需要花费大量的时间在挑选任务上, 降低了开发效率, 同时软件开发任务

的复杂性和专业性使得开发者凭主观意愿去选择的任务并不是最适合自己的, 从而满足不了需求方的需求. 针对目前存在的这些问题, 通过引入传统领域中用于处理信息过载的推荐技术来解决.

为了给开发者推荐合适的任务, 需要考虑多个方面的因素. 一方面, 开发者会首先基于兴趣爱好选择任务, 然而开发者的兴趣偏好是不断变化的, 如何精准捕获开发者当前的偏好会对任务推荐的结果造成很大的影响. 另一方面, 与传统推荐领域中的商品、电影等内容不同, 仅考虑开发者的偏好进行推荐是不够的, 软件开发任务需要具有特定知识和技能的人才能完成, 同时, 竞争性质的开发任务较多, 能够最终获胜并赢得奖励的人数有限, 因此开发者在选择任务时还会考虑其是否有能力在众多竞争对手中获得较高的评分从而获得胜利. 基于此, 本文提出了一种基于开发者动态偏好和竞争力的群智化软件任务推荐方法, 从而综合考虑多方面因素, 给开发者推荐最合适的任务. 具体来说, 本文的主要研究成果如下.

本文综合考虑开发者的动态偏好和竞争力特征对开发者建模, 并定义了相应的参数指标, 其中开发者的任务偏好由开发者的报名历史任务信息来描述, 开发者的任务竞争力以开发者历史相似任务表现情况、当前任务参与情况以及潜在的竞争对手信息来构建.

本文提出了一个两阶段的任务推荐模型: 在第 1 个阶段, 使用 A-LSTM 预测出开发者当前的动态偏好, 并利用相似度从大量候选任务中筛选出符合偏好的 Top-*N* 任务; 在第 2 个阶段, 基于开发者的竞争力, 使用 DE-XGBoost 预测开发者在前一阶段筛选出的任务上的评分, 并按照评分从高到低向开发者人推荐 Top-*K* 任务, 从而进一步提高推荐的准确性.

本文通过一系列在真实的群智化软件开发平台数据的实验, 验证了本文模型的有效性, 同时与其他几种推荐的经典方法的对比表明本文模型具有更高的推荐准确性.

References:

[1] Howe J. The rise of crowdsourcing. Wired Magazine, 2006, 14(6): 1−4.

[2] Mao K, Capra L, Harman M, *et al*. A survey of the use of crowdsourcing in software engineering. Journal of Systems and Software, 2017, 126: 57−84.

[3] He X, Zhu Y, Wang M. Knowledge emergence and complex adaptability in swarm intelligence. Information and Control-Shenyang, 2005, 34(5): 560.

[4] Krause J, Ruxton GD, Krause S. Swarm intelligence in animals and humans. Trends in Ecology & Evolution, 2010, 25(1): 28−34.

[5] Xu L, Wu H. Collective intelligence based software engineering. Journal of Computer Research and Development, 2020, 57(3): 487 (in Chinese with English abstract).

[6] Isaeva VV. Self-organization in biological systems. Izvestiia Akademii Nauk Seriia Biologicheskaia, 2012, 39(2): 110−118.

[7] Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies. In: Proc. of the 1st European Conf. on Artificial Life. 1991, 142: 134−142.

[8] Chu SC, Roddick JF, Su CJ, *et al*. Constrained ant colony optimization for data clustering. In: In: Proc. of the Pacific RIM Int'l Conf. on Artificial Intelligence. Berlin, Heidelberg: Springer, 2004. 534−543.

[9] Xu Y, Fan P, Yuan L. A simple and efficient artificial bee colony algorithm. Mathematical Problems in Engineering, 2013, 2013(1): 526315.1−526315.9.

[10] Herculano-Houzel S. The human brain in numbers: A linearly scaled-up primate brain. Frontiers in Human Neuroscience, 2009, 3: 31.

[11] Li W, Wu W, Wang H, *et al*. Crowd intelligence in AI 2.0 era. Frontiers of Information Technology & Electronic Engineering, 2017, 18(1): 15−43.

[12] Herbsleb JD. Global software engineering: The future of socio-technical coordination. In: Future of Software Engineering (FOSE 2007). IEEE, 2007. 188−198.

[13] Kogut B, Metiu A. Open-source software development and distributed innovation. Oxford Review of Economic Policy, 2001, 17(2): 248−264.

[14] Xu X, Wu W, Wang Y, *et al*. Software crowdsourcing for developing software-as-a-service. Frontiers of Computer Science, 2015, 9(4): 554−565.

[15] Mao K, Yang Y, Wang Q, *et al*. Developer recommendation for crowdsourced software development tasks. In: Proc. of the 2015 IEEE Symp. on Service-oriented System Engineering. IEEE, 2015. 347−356.

[16] Khanfor A, Yang Y, Vesonder G, *et al*. Failure prediction in crowdsourced software development. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2017. 495−504.

[17] Ambati V, Vogel S, Carbonell J. Towards task recommendation in micro-task markets. In: Proc.of the Workshops at the 25th AAAI Conf. on Artificial Intelligence. 2011.

[18] Lin CH, Kamar E, Horvitz E. Signals in the silence: Models of implicit feedback in a recommendation system for crowdsourcing. In: Proc. of the 28th AAAI Conf. on Artificial Intelligence. 2014.

[19] Difallah DE, Demartini G, Cudré-Mauroux P. Pick-a-crowd: Tell me what you like, and i'll tell you what to do. In: Proc. of the 22nd Int'l Conf. on World Wide Web. 2013. 367−374.

[20] Yuen MC, King I, Leung KS. Task matching in crowdsourcing. In: Proc. of the 2011 Int'l Conf. on Internet of Things and the 4th Int'l Conf. on Cyber, Physical and Social Computing. 2011. 409−412.

[21] Yuen MC, King I, Leung KS. Task recommendation in crowdsourcing systems. In: Proc. of the 1st Int'l Workshop on Crowdsourcing and Data Mining. 2012. 22−26.

[22] Yuen MC, King I, Leung KS. Taskrec: Probabilistic matrix factorization in task recommendation in crowdsourcing systems. In: Proc. of the Int'l Conf. on Neural Information Processing. 2012. 516−525.

[23] Yuen MC, King I, Leung KS. Taskrec: A task recommendation framework in crowdsourcing systems. Neural Processing Letters, 2015, 41(2): 223−238.

[24] Yuen MC, King I, Leung KS. An Online-updating approach on task recommendation in crowdsourcing systems. In: Proc. of the Int'l Conf. on Neural Information Processing. 2016. 91−101.

[25] Yuen MC, King I, Leung KS. An online-updating algorithm on probabilistic matrix factorization with active learning for task recommendation in crowdsourcing systems. Big Data Analytics, 2016, 1(1): 14.

[26] Kurup AR, Sajeev GP. Task recommendation in reward-based crowdsourcing systems. In: Proc. of the 2017 Int'l Conf. on Advances in Computing, Communications and Informatics (ICACCI). 2017. 1511−1518.

[27] Kang Q, Tay WP. Task recommendation in crowdsourcing based on learning preferences and reliabilities. IEEE Trans. on Services Computing, 2020.

[28] Zhang TT, Zhao YX, Zhu QH. Mining user preferences in crowdsourcing community with sensitivity analysis. Data Analysis and Knowledge Discovery, 2018, 2(5): 23−31 (in Chinese with English abstract).

[29] Pan Q, Dong H, Wang Y, *et al*. Recommendation of crowdsourcing tasks based on Word2vec semantic tags. Wireless Communications and Mobile Computing, 2019.

[30] Yang Y, Karim MR, Saremi R, *et al*. Who should take this task? Dynamic decision support for crowd workers. In: Proc. of the 10th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. 2016. 1−10.

[31] Tunio MZ, Luo H, Wang C, *et al*. Crowdsourcing software development: task assignment using PDDL artificial intelligence planning. Journal of Information Processing Systems, 2018, 14(1): 129−139.

[32] Tunio MZ, Luo H, Wang C, *et al*. Task assignment model for crowdsourcing software development: TAM. Journal of Information Processing Systems, 2018, 14(3): 621−630.

[33] Li N, Mo W, Shen B. Task recommendation with developer social network in software crowdsourcing. In: Proc. of the 23rd Asia-Pacific Software Engineering Conf. (APSEC). 2016. 9−16.

[34] Yu D, Wang Y, Zhou Z. Software crowdsourcing task allocation algorithm based on dynamic utility. IEEE Access, 2019, 7: 33094−33106.

[35] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735−1780.

[36] Mnih V, Heess N, Graves A. Recurrent models of visual attention. In: Advances in Neural Information Processing Systems. 2014. 2204−2212.

[37] Ramachandran P, Parmar N, Vaswani A, *et al*. Stand-alone self-attention in vision models. In: Advances in Neural Information Processing Systems, 2019. 32.

[38] Vaswani A, Shazeer N, Parmar N, *et al*. Attention is all you need. In: Advances in Neural Information Processing Systems, 2017. 30.

[39] Jaegle A, Gimeno F, Brock A, *et al*. Perceiver: General perception with iterative attention. In: Proc. of the Int'l Conf. on Machine Learning. PMLR, 2021. 4651−4664.

[40] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473, 2014.

[41] Ma F, Chitta R, Zhou J, *et al*. Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2017. 1903−1911.

[42] Ke G, Meng Q, Finley T, *et al*. LightGBM: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 2017, 30: 3146−3154.

[43] Chen T, He T, Benesty M, *et al*. XGboost: Extreme gradient boosting. R package version 0.4-2, 2015, 1(4): 1–4.

[44] Chen T, Guestrin C. Xgboost: A scalable tree boosting system. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2016. 785–794.

[45] Storn R, Price K. Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997, 11(4): 341–359.

[46] Xiao J, Wang Q, Li M, *et al*. A constraint-driven human resource scheduling method in software development and maintenance process. In: Proc. of the 2008 IEEE Int'l Conf. on Software Maintenance. 2008. 17–26.

[47] Zhang Z, Lei Y, Xu J, *et al*. TFIDF-FL: Localizing faults using term frequency-inverse document frequency and deep learning. IEICE Trans. on Information and Systems, 2019, 102(9): 1860–1864.

[48] Järvelin K, Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans. on Information Systems, 2002, 20(4): 422–446.

[49] Moffat A, Zobel J. Rank-biased precision for measurement of retrieval effectiveness. ACM Trans. on Information Systems, 2008, 27(1): 1–27.

[50] Vlachos M, Dünner C, Heckel RG, *et al*. Addressing interpretability and cold-start in matrix factorization for recommender systems. IEEE Trans. on Knowledge and Data Engineering, 2018, 31(7): 1253–1266.

[51] Aharon M, Anava O, Avigdor-Elgrabli N, *et al*. Excuseme: Asking users to help in item cold-start recommendations. In: Proc. of the 9th ACM Conf. on Recommender Systems. 2015. 83–90.

附中文参考文献:

[28] 张亭亭, 赵宇翔, 朱庆华. 众包社区中基于敏感性分析的用户偏好挖掘模型及实验. 数据分析与知识发现, 2018, 2(5): 23–31.

王红兵(1966－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为服务计算, 机器学习.

张丹丹(1993－), 女, 博士生, 主要研究领域为机器学习, 服务计算.

严嘉(1995－), 男, 硕士生, 主要研究领域为机器学习, 推荐系统.

陆荣荣(1996－), 女, 硕士生, CCF 学生会员, 主要研究领域为机器学习, 数据挖掘.