计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201000107

数据驱动的开源贡献度量化评估与持续优化方法

范家宽[1]　王皓月[1]　赵生宇[2]　周添一[1]　王　伟[1]
1 华东师范大学数据科学与工程学院　上海 200062
2 同济大学电子与信息工程学院　上海 201804
(jkfan@stu.ecnu.edu.cn)

摘　要　在当今数字化时代,开源技术、开源软件和开源社区日益重要,而通过量化分析方法研究开源领域的问题也已经成为一个重要的趋势。开发者是开源项目中的核心,其贡献度的量化以及量化后的贡献度提升策略,是开源项目能够健康发展的关键。文中提出了一种数据驱动的开源贡献度量化评估与持续优化方法,并通过一个实际的工具框架 Rosstor(Robotic Open Source Software Mentor)进行了实现。该框架包含两个主要部分:1)贡献度评估模型,采取了熵权法,可以动态客观地评估开发者的贡献度;2)贡献度持续优化模型,采取了深度强化学习方法,最大化了开发者的贡献度。文中选取了 GitHub 上若干著名的开源项目的贡献者数据,通过大量且充分的实验验证了 Rosstor 不仅能够使所有项目上开发者的贡献度得到大幅度提升,而且还具有一定的抗干扰性,充分证明了所提方法和框架的有效性。Rosstor 框架为当下广泛开展的开源项目和开源社区的可持续健康发展提供了方法和工具方面的支持。

关键词:开源软件;贡献度测量;贡献增强;深度强化学习;模仿学习
中图法分类号　TP391

Data-driven Methods for Quantitative Assessment and Enhancement of Open Source Contributions

FAN Jia-kuan[1],WANG Hao-yue[1],ZHAO Sheng-yu[2],ZHOU Tian-yi[1] and WANG Wei[1]
1 School of Data Science and Engineering,East China Normal University,Shanghai 200062,China
2 College of Electronical and Information Engineering,Tongji University,Shanghai 201804,China

Abstract　In recent years,open source technologies,open source software and open source communities have become increasingly significant in digital era,and it has become an important trend to study the open source field through quantitative analysis methods. Developers are the core of open source projects,and the quantification of their contributions and the strategies to improve their contributions after quantification are the key to the healthy development of open source projects. We propose a data-driven method for quantitative assessment and continuous optimization of open source contributions. Then,we implement it through a practical framework,Rosstor (Robotic Open Source Software Mentor). The framework consists of two main parts. One is a contribution evaluation model,it adopts an entropy-weight approach and can dynamically and objectively evaluate developers' contributions. Another is a model to enhance contributions,it adopts a deep reinforcement learning approach and can maximize developers' contributions. Contributors' data from a number of famous open source projects on GitHub are selected,and through massive and sufficient experiments,it verifies that Rosstor not only makes the developers' contributions on all projects to be greatly improved,but also has a certain degree of immunity,which fully proves the effectiveness of the framework. The Rosstor framework provides methodological and instrumental support for the sustainable health of open source projects and the open source community.

Keywords　Open source software,Contribution measurement,Contribution enhancement,Deep reinforcement learning,Imitation learning

1　引言

开源技术、开源软件和开源社区在当今数字化时代日益重要,GitHub 作为全球最大的代码托管平台,目前所托管的项目数量已经超过 1 亿,并且用户数量也达到约 4000 万。开源项目作为开源软件非常重要的载体之一,正在快速地重构着世界。无论是公司、学校、开发者组织还是非盈利团体都在积极地参与开源项目,通过开源项目提高其在社区的影响力,其中踊跃出了一批对于开发者来说家喻户晓的开源项目,包括 Microsoft 的 vscode 项目、Facebook 的 react-native 项目和 Google 的 Tensorflow 项目等。

近年来,以 Apache 社区为代表的开源软件取得了巨大

到稿日期:2020-10-19　返修日期:2021-03-10
通信作者:王伟(wwang@dase.ecnu.edu.cn)

的成功[1]。开源社区也吸引了越来越多的开发者加入，在不同地区拥有不同开发经验的开发者会因为自身的兴趣、声誉和就业需求而自发地聚集在一起。这些开发者通常通过经验交流[2]、调试代码[3]和提交功能补丁[4]来为项目做出贡献。在这之中，许多研究又一直将项目贡献作为评估开发者状态的重要指标[5]。一般而言，开发者对于项目的贡献度主要指开发者通过新建 issue、参与讨论或者提交代码等各种形式参与到项目的发展中，为项目的发展和演化做出的贡献(包括代码和非代码类型)的总和。

在许多实际开源项目中，项目经理需要因业务或者团队优化等各种因素来比较和量化不同开发者的贡献度。虽然基于传统价值的软件工程[6]专注于创造经济价值，并以此作为对资源分配和调度进行优先排序的一种方式，但是，在某些开源项目中，关于价值的其他标准的衡量可能更为重要。比如，教师需要一种工具来评估单个学生对小组项目的代码贡献(除非代码贡献之外)，而衡量代码贡献的方法与经济回报是无关的；项目经理可能需要定量评估团队成员的绩效。此外，对于开源软件项目，开发者的个人贡献也会极大地影响项目的协作和发展[7-8]。

因此，对项目的主要贡献者的所有数据进行建模，并且定量分析所有动作之间的相关性，以期从贡献者的数据中学到普遍规律来指导开发者如何提升他们的贡献度，这是本文的研究目的。对于开发者的贡献度的量化和提升，将极大地帮助软件工程项目经理根据开发者的个人资料建立项目团队，通过优化团队人员的开发配置，根据各成员的贡献度分配其最合适的开发工作，从而提高团队开发的生产力。但据我们所知，目前对于如何提高开发者在开源项目中贡献度的研究是非常少的，而且因为开发者可以通过多种方式做出贡献，所以很难获得一般的通用指南，这也导致相关研究很少涉及到如何参与开源项目的指导。

本文首先定义了贡献度量化模型，该模型反映了随着时间的推移开发者动作之间的相互重要关系，并从项目整体的角度考虑了所有可能的动作(包括编码的和非编码的)。进一步，本文提出了 Rosstor(Robotic Open Source Software Mentor)框架，通过将实际问题转化为强化学习问题来帮助开发者最大程度地提升自身贡献度。然后，本文通过优化训练过程中参数的利用率显著提高了算法的性能。本文的主要贡献包括：

(1)提出了一种数据驱动的贡献度评估模型，该模型可以根据数据的变化动态评估开发者的贡献度。

(2)首次探索了将深度强化学习技术引入开源软件治理中，以解决提高开发者贡献度的问题。

(3)通过大量的实验，证明了本文的框架的有效性。

Rosstor 的功能类似于开源导师，它会通过数据驱动的方式，不断给开发者提供优化贡献度的策略，带领该开发者为开源项目持续做出最优的贡献。

本文第 2 节重点介绍了 Rosstor 框架；第 3 节通过实验验证了本文模型的性能，并分析了实验结果；第 4 节给出了相关工作；最后总结全文。

2 Rosstor 框架

本节首先量化了关于贡献度的评估框架，并给出了框架的总体架构说明，将开源项目中的实际问题量化为强化学习算法的基本要素，然后介绍了模型的整体算法流程，说明了模型的训练过程。

2.1 Rosstor 整体架构

(1)贡献度量化。衡量开发者贡献度的工作目前基本上停留在可视化层面，关于量化的方法一般也都是直接通过计算 Github 事件的问题、问题评论、提交请求、提交请求评论的数量等得出动作对应的权重，然后通过对动作权重和数量的乘积累计求和量化贡献度[9]。但是这种方法的问题在于没有对项目整体数据进行分析以获取符合客观规律的结果，并且该方法无法反映各个项目的特征和随时间的变化情况。本文首先引入了信息熵的概念来衡量开发者可执行的动作对开发者的辨识度，然后分析整个项目所有的数据，以便从客观角度确定动作之间的权重关系。信息熵的计算式如下：

$$H(x)=-\sum_{i=1}^{n}P(x_i)\log P(x_i) \tag{1}$$

其中，$P(x_i)$代表事件x_i发生的概率。在信息论中，熵代表信息的离散程度，离散程度越高，熵越大，代表的信息量就越大。因此，当离散程度呈现平均分布时熵最大。本文选取了开源项目中开发者实际执行的动作事件，通过计算动作事件的熵值来衡量动作的离散程度。若 $H(x)$在第 i 个动作维度上的熵值越大，则信息量越大，说明第 i 个动作对于开发者的辨识度越低，也就意味着所有人更倾向于执行这个动作。进一步地，利用熵权法求出在具体开源项目中，各动作相对于其他动作的重要性程度。

信息熵的计算存在一个前提条件，即假设动作之间具有独立性。在实际开源项目中，由于动作之间的信息交互问题(例如问题和问题评论之间存在很强的相关性)，原始的权重量化方式并不能满足熵加权的计算规则。为了解决互信息问题，本文引入了条件熵的计算方法，计算式如下：

$$H_i(Y|X)=-\sum_{x\in X,y\in Y}^{n}P_i(x,y)\log\frac{P_i(x,y)}{P_i(x)} \tag{2}$$

通过引入条件概率来解决开源项目中的动作之间不独立的问题。通过上述方法，可以为项目的所有动作维度计算权重向量 $\boldsymbol{W}_i$，进而得到贡献度的计算方法：

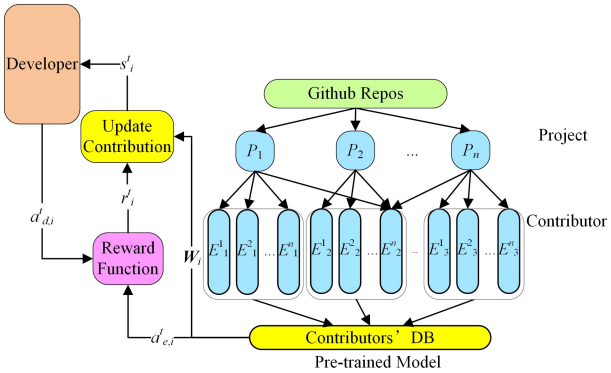
$$C_{(i,k)}=\sum_{t=1}^{T}\boldsymbol{W}_{(i,k)}*A_{(i,k)}^{t} \tag{3}$$

其中，$A_{(i,k)}^{t}$表示在第 i 个项目的第 k 轮迭代中第 t 步动作被执的次数；$\boldsymbol{W}_{(i,k)}$是根据条件熵模型计算得出的归一化后的权重向量，也就是说$\sum_{t=1}^{T}\boldsymbol{W}_t=1$。

(2)Rosstor 架构。这是首次将深度强化学习技术应用到开源项目治理领域中的探索。该模型的目标是在开发者进行序列化的多次执行动作之后最大化开发者的累积贡献度。该模型的详细流程如图 1 所示。首先，环境部分是经过预训练的模型，该模型使用贡献度量化方法(见式(3))对权重向量 $\boldsymbol{W}_i$进行预训练，并且得到包含贡献者的动作数据集$E_i=\{e_i^1,e_i^2,e_i^3,\cdots,e_i^n\}$，开发者作为智能体会根据自己的策略在状态$s_i^t$中选择动作$a_{d,i}^t$。然后，Rosstor 根据开发者此时的贡献度的

状态去匹配相同贡献度下贡献者的动作$a^t_{e,i}$，根据$a^t_{d,i}$和$a^t_{e,i}$的相似性关系得到当前时刻执行$a^t_{d,i}$的回报r^t_i。最后，通过预训练模型的$\boldsymbol{W}_i$更新开发者当前时刻的贡献度C^t_i，状态会转移至下一个状态，继续以上操作，从而不断地更新策略。

上述过程将如何提高开发者对开源软件贡献度的问题转变为了一个强化学习问题，从而最大程度地提高了对项目的贡献度。此外，本文提出的框架 Rosstor 还可以针对开发者能力水平的不同状态推荐自适应难度级别，即将开发者当前状态、动作模式和累积贡献度相结合，以推荐匹配开发者能力的下一个最合适的动作模式。



注：环境部分是使用贡献度量化方法（见式(3)）预训练的模型

图 1 Rosstor 的整体框架

Fig. 1 Framework of Rosstor

2.2 强化学习基本元素定义

(1)动作。定义动作$A=[a_1,a_2,a_3,\cdots,a_n]$，它表示开发者可以在开源项目中执行的动作类型的集合，集合中的任何元素都具有特定的形式：$a^t_i=[N^t_{a_{i1}},N^t_{a_{i2}},\cdots,N^t_{a_{in}}]$表示第$i$个项目在$t$个月内在每个动作维度上执行操作的次数。向量的每个维度的索引代表开发者可以执行的操作类型，例如创建问题、问题评论、关闭问题、提交请求、提交请求评论等。与向量的维度索引对应的值代表在当前月份已执行操作的次数。

(2)状态。定义状态$s^t_i=[e^t_{a_{i1}},e^t_{a_{i2}},\cdots,e^t_{a_{in}},c^t_{a_i}]$，其中状态由反映立即状态的短期特征$[e^t_{a_{i1}},e^t_{a_{i2}},\cdots,e^t_{a_{in}}]$和反映累积贡献度的长期特征$c^t_{a_i}$组成。状态主要分为两个部分：第一部分包括与开发者执行的相同贡献下的所有贡献者数据的期望值$\frac{1}{N}\sum e^t_{a_i}$，这部分反映了短期特征的最佳近似值；第二部分反映了开发者累计贡献度的长期特征，用于描述开发者当前的熟练程度。

(3)奖励函数。奖励函数的设计直接体现了要优化的目标，所获得的每个奖励都表明了执行当前动作的好坏，因此设计一个好的奖励函数对于成功实现强化学习中的目标非常重要。本文的目标是最大程度地提高开发者的贡献度（式(3)给出了贡献度的计算），一个明显存在的问题是，开发者执行的次数越多，他得到的贡献就越多。但是，事实并非如此，如果开发者执行的每个动作与贡献者的动作都不相似，则该动作的执行应该是不好的。相反，如果相似度很高，则说明这是好动作。因此，至关重要的是，奖励函数的设计应具有在给定贡献度场景下衡量开发者行为与贡献者行为之间相似性的能力。考虑到这一点，本文定义了奖励函数：

$$r_t=\boldsymbol{W}*A_t*\exp(-\lambda\|(a^t_d-a^t_e)*\boldsymbol{W}_t\|^2_2) \quad (4)$$

其中，$\exp(-\lambda\|(a^t_d-a^t_e)*\boldsymbol{W}_t\|^2_2)$代表$t$时刻开发者动作$a^t_d$和贡献者动作$a^t_e$的相似性程度，$\lambda=\frac{1}{2\sigma^2}$，$\sigma$是控制$a^t_d$和$a^t_e$相似性的域宽，$\|(a^t_d-a^t_e)*\boldsymbol{W}_t\|^2_2$代表开发者和贡献者动作间的欧氏距离，由于不同的动作代表的权重不同，因此权重向量可以增大相似动作带来的贡献度，同时减小非相似动作衡量的相似度，从而更加客观地评估开发者的状态。

(4)环境。本文使用预训练的模型作为与智能体进行交互的环境。全局项目的数据信息存储在 GitHub 数据库中，其中包含 GitHub API 定义的所有事件信息。对于要测试的项目，本文从中抽取了排名较高的贡献者，并将他们的历史行为数据用作贡献者库，期望从贡献者库中学到一般的开发规律，从而向开发者不断提出适合他的动作建议。对于开发者采取的每项操作，环境都会根据项目的贡献者数据库中的数据来匹配开发者当前状态下每个贡献者的行为数据。此外，它会计算所有行为者数据的平均行为模式，并将其作为环境的输出，最后计算开发者和贡献者的行为相似度以获得奖励。

(5)目标函数。本文的目标是最大化开发者的贡献度。结合式(4)，目标函数定义为：

$$J^{\theta^k}_t=\sum_{(s_t,a_t)}\min(\pi_t(\theta)A^{\theta^k}_t,clip(\pi_t(\theta),1-\varepsilon,1+\varepsilon)A^{\theta^k}_t) \quad (5)$$

其中，$\pi_t(\theta)=\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}$表示两个分布$p_\theta(a_t|s_t)$和$p_{\theta^k}(a_t|s_t)$之间的重要性采样比例；$A^{\theta^k}_t$是参数$\theta^k$的优势函数；$\varepsilon$是一个超参数，其值域为$[0,1]$。

2.3 Rosstor 算法流程

在开源项目治理中存在两个难题，第一个是如何量化开发者的贡献度。本文创新性地引入了信息熵的概念来度量动作的识别度，然后解决了通过熵权法计算动作之间的重要程度的问题。本文又针对不满足熵定律的独立性假设问题，通过计算条件熵来解决条件独立性假设问题，最后系统地提出了良定义的开发者贡献度评估方式。

在解决了如何量化开发者的贡献之后，再讨论第二个难题，即如何最大化贡献度。在此之前，相关工作主要集中在如何可视化和量化贡献上，而很少有研究涉及如何改进它。但本文认为最重要的是如何帮助开发者做出决策，以提高他们参与开源项目的能力。本文将如何提高开发者在开源项目治理中贡献的实际需求抽象为一个强化学习问题，并尝试以这种方式最大程度地提高开发者的贡献度。算法 1 给出了具体的算法流程。

算法 1 Rosstor

1. 通过式(2)预训练开源软件项目；
2. 输入贡献者的数据$e_i=e^1_i,e^2_i,\cdots,e^l_i$以及从预训练模型中获取到的权重$\mathbf{W}_i$；
3. 用权重Θ^Q，Θ^μ初始化评价网络$Q(s,a|\Theta^Q)$和动作网络$\mu(s|\Theta^\mu)$；
4. for each i∈[1,M] do
5. Receive initial observation state s_0；
6. for each t∈[1,M] do
7. select developer's action a^t_d according to the policy and exploration noise；
8. execute a^t_d, observe new state s_{t+1}, observe contributors'

action a_e^t;
compute similarity of (a_d^t, a_e^t), and contribution c_t using Eq. 4;
if t % batchsize == 0 then
update actor network parameter Θ^μ by Eq. 5;
update critic network Θ^Q;
end if
end for
end for

此外，Rosstor 通过在每个训练回合中实现批处理大小来优化参数，而区别于 PPO 算法在训练回合之后对其进行优化，这样可以更好地控制参数优化的方向，实验表明这种方式可以很大程度地提升算法的性能。

3 实验结果与分析

为了验证本文提出的 Rosstor 框架的有效性，本节将通过大量实验并结合具体的实证研究，回答如下几个关键问题：

(1)与已有贡献度评估模型相比，本文提出的贡献度评估模型的科学性与合理性是什么？

(2)与已有基准方法相比，Rosstor 在提高开发者贡献度方面的表现如何？

(3)本文的贡献度模型有什么性质？

3.1 实验对比算法

近年来，深度强化学习取得了令人印象深刻的效果，在较多领域都取得了一定的成果，如游戏[10]、音乐[11]等领域，此外，利用专家经验进行引导式学习的方法也逐渐成为强化学习研究的热点[12]。本文对比了本文方法和以往优秀的算法 DDGP，GAIL 和 PPO。DDPG[13]是同时学习 Q 函数和策略的算法，它使用离线数据和 Bellman 方程来学习 Q 函数，并使用 Q 函数来学习策略，适用于动作连续和高维空间情况。GAIL[14]采用和传统强化学习相反的思路，先从专家的数据中学得回报函数，然后进行正向的训练，适用于无法确定回报函数或者回报稀疏的情况。PPO[15]通过不断衡量两个策略分布之间的距离来保证每次步长更新都朝着更好的方向来提升策略。本文提出的 Rosstor 框架的核心算法是基于 PPO 算法的进一步优化。

3.2 实验数据介绍

本文从 GitHub 中挑选了全球非常活跃的 6 个项目，这些项目在类型上尽可能地趋于多样化。在以下 6 份数据集中分别验证本文提出的框架的实验效果。

本文通过 GitHub API 收集了这些 GitHub 项目创建开始到 2019 年 12 月的所有用户在项目的贡献数据，包括 open-issue，issue-comment，open-pr，pr-comment 4 种事件字段。数据中记录了每一个贡献者在该段时间内所做的事件类型的次数。根据贡献者的提交次数，本文提取了每个项目中前 120 名贡献者的行为轨迹。将数据引入到框架中主要有 3 个步骤：首先，对于抽取的 120 名贡献者的行为轨迹，按照以月为单位进行切分，划分出该贡献者当月执行各种动作事件的次数；然后，针对每一个项目的贡献者数据，用式(2)计算该项目动作事件的权值；最后，将整理好的数据存储到专家库中，每一次动作执行后都会将该动作与专家库中贡献者的最优动作轨迹进行匹配。

(1)Ansible/ansible[1]：Ansible 是一个非常简单的 IT 自动化系统。

(2)MicrosoftDocs/azure-docs[2]：azure-docs 是 Microsoft Azure 的开源文档。

(3)DefinitelyTyped/DefinitelyTyped[3]：DefinitelyTyped (D.T.)用于高质量的 TypeScript 类型定义。

(4)Flutter/flutter[4]：Flutter 是 Google 的 SDK，它可以通过一个代码库为 Web 和桌面等快速构建美观的应用程序。

(5)Elastic/kibana[5]：Kibana 是进入弹性堆栈的窗口。具体来说，它是 Elasticsearch 的基于浏览器的分析和搜索仪表板。

(6)Kubernetes/kubernetes[6]：Kubernetes 是一个开放源代码系统，用于跨多个主机管理容器化的应用程序。

3.3 训练细节

(1)K 度松弛。本文将数据集整理成以月份为单位的开发者轨迹数据的集合，每一条记录代表着开发者在当前项目中某一个月在所有动作维度上执行的次数。本文采用模仿学习的思路，将提取出来的贡献度较高的开发者的轨迹数据作为经验库，期望从所有的经验库中学到普遍适用的规律来帮助普通开发者不断提升贡献度。因此，在模仿学习过程中会不可避免地存在问题，即贡献者的状态分布很可能不涉及所有的开发者的状态分布，这样就会导致在某些极端状态下开发者无从学习。为了解决该问题，本文提出了 K 度松弛法来匹配开发者的数据。具体而言，当开发者执行某一具体动作时，会将该动作代表的贡献度去匹配经验库中相同贡献度下开发者执行的动作。但是，匹配时采取的并非是精确匹配而是模糊匹配，即每次匹配贡献度的上下 K 度空间。当在 K 度空间内依然无法匹配到贡献者的数据时，会触发递归机制，K 度空间会被依次向外扩增直到匹配到相关数据为止。这样就解决了极端情况下开发者分布和贡献者分布不匹配的问题。

(2)回报函数。对于回报函数，本文通过衡量开发者动作和专家动作的相似度，并添加约束条件来定义回报。本文起初设计了余弦相似度方法来衡量。但是，余弦相似度的结果优化的只是向量间的方向，而向量的长度即动作执行的数量经常会收敛到极端状态，从而导致策略失效，因此本文改造了高斯核径向基函数来衡量两者的相似度(见式(4))。

(3)算法。本文构造了动作网络和评价网络两个网络，动作网络有一层中间层，通过两次网络计算分别输出动作的均值和方差，动作来自于通过均值和方差构建的正态分布的随机采样。对于训练过程中的动作、状态，本文都做了归一化处理，以便模型能快速地训练和更好地收敛。两个网络的学习率都被设置成了 0.01，并且在每 10 步之后会同时更新网络

1) https://github.com/ansible/ansible
2) https://github.com/MicrosoftDocs/azure-docs
3) https://github.com/DefinitelyTyped/DefinitelyTyped
4) https://github.com/flutter/flutter
5) https://github.com/elastic/kibana
6) https://github.com/kubernetes/kubernetes

的参数。在计算新旧策略的比例时，为了防止旧策略的值为0从而导致无法正常求解梯度的问题，本文对旧策略的概率进行了截断处理，最小值被设置为10^{-5}。本文对每一个项目总共训练了1 000轮，每一轮共计训练200步。

对于动作网络的建立，本文使用ReLU构建了一个由64个神经元组成的网络。对于网络输出的均值和方差，本文分别使用了激活函数tanh，$softplus$。

3.4 实验结果

本文提出的贡献度的量化方法具有重要意义：首先，贡献度量化权重的确定完全基于数据，这与文献[13]通过专家经验人为确定贡献度量化权重相比具有非常客观的优势，不会包含由于人为原因的误判导致评价准确性降低的问题；其次，权重是动态变化的，即项目数据随时间的变化以及项目状态的变化而变化，都会导致权重的更新。因此，本文定义的贡献度量化权重不仅反映了随着时间的推移动作之间的相互重要性关系，还反映了对于不同项目之间的动作重要性关系，这对评价某垂直细分领域的开源项目行业报告具有重要的业务价值。

(1)Rosstor与Baseline的对比。为了对比Rosstor和基准算法之间的性能，本文进行了此实验。如表1所列，对于6个项目数据集，本文训练了GAIL算法、DDPG算法和Rosstor算法，以找到使贡献度最大化的最佳策略。为了减小一次实验引起的误差，对于每个项目数据集，本文随机选择10个开发者的轨迹并进行了3次实验，用3个结果的平均值作为最终结果。从表1中的数据可以很容易地看出，本文的Rosstor在绝大多数项目中都显示出了巨大的优势。此外，本文还比较了开发者的实际轨迹的贡献度，并将结果显示在表的REAL列中。本文认为产生此结果的原因是Rosstor能够更好地解决学习步长选择问题，以便每次都能在更好的方向上对策略进行优化。显然，Rosstor不仅可以提供比基线更好的结果，而且在现实项目中具有巨大的优势。

表1 不同算法的平均单步贡献

Table 1 Average single-stepcontribution of different algorithms

Projects	contributions			
	DDPG	GAIL	REAL	Rosstor
flutter	3.53	13.18	47.90	**68.31**
ansible	70.38	14.80	33.43	**77.81**
azure-docs	85.68	19.00	14.18	**85.90**
D.T.	49.84	13.89	10.13	**72.38**
kubernetes	8.46	20.46	107.05	**197.57**
kibana	89.05	15.30	16.01	49.65

(2)Rosstor与PPO的对比。本文设计了以下实验来验证Rosstor是否具有比原始模型PPO更好的性能。图2给出了Rosstor和PPO两种算法在多次实验后的平均结果的比较情况。在算法训练之后，本文对两种算法的10个项目进行了实际测试。从选取的截图中可以看出，本文的Rosstor在每个项目中的表现最好。我们分析可能的原因是，PPO算法仅在每轮训练之后才进行参数更新，降低了训练期间参数的利用率；而本文的Rosstor将在每个训练回合中以每批次大小的长度更新参数，保留了有关参数的更多信息，从而使算法更优。因此，本文提出的Rosstor具有比原始算法更好的性能。

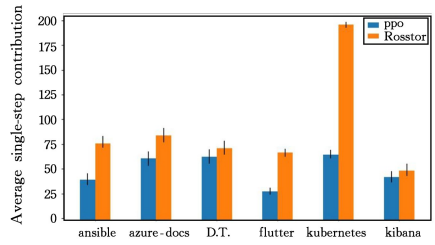


图2 PPO和Rosstor的对比结果

Fig. 2 Comparison of PPO and Rosstor

(3)案例研究。为了更详细地展示Rosstor的效果，本文随机选择了18个月内参与者的真实轨迹，遵循他们的初始状态，本文使用Rosstor为贡献者推荐策略。在学到的推荐策略中，存在贡献度的变化情况呈波浪式上升的现象。这是更符合真实情况的，当开发者熟悉项目之后，贡献度会越来越大，但当某个月进行了高强度贡献之后，之后的一段时间内其更倾向于短暂休息，因此贡献度会稍微降低，而之后会继续升高，且后续的贡献度峰值会大于前一个峰值。通过分析实验结果发现，真实轨迹和推荐轨迹都有一个共同的特征：波浪式起伏上升。如图3所示，从第12－18个月可以发现，推荐策略的贡献度和真实贡献度的分布是相匹配的，即推荐的策略在开发者的能力范围内。但是，在执行操作的第4个月之后的6个月内(即5－11月)，开发者未参与任何贡献，这显然不是最佳途径。Rosstor在第4个月后的6个月内采用了更好的策略，考虑了贡献者的休息时间和间隔时间，以增加其贡献度。图3中的虚线显示了Rosstor推荐的方法，可以看出其可以更好地提升贡献度。

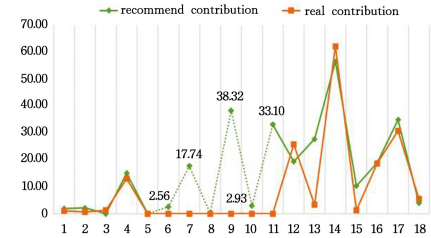


图3 贡献度真实轨迹和推荐轨迹的对比

Fig. 3 Comparison of real contribution and recommended contribution

4 相关工作

在传统的软件开发模型中，开发者贡献的最初衡量标准是字数[16]。最经典的度量是开发者编写的代码行数(Lines of Code，LOC)，这也是最简单的方法。LOC是每个开发者所有提交的修改行数的累积。两种主要的代码测量方法为物理代码行(物理SLOC，LOC)和逻辑代码行(逻辑SLOC，LLOC)测量。

Gousions等[17]总结了来自数据源(例如代码仓库、邮件列表和论坛、缺陷数据库、Wiki等)的近30个度量标准。这些度量标准包括代码内容和非代码内容。他们根据影响将这些指标分为正面和负面两个方面，并提出了一个更复杂的衡量体系。但是，该研究方法没有在特定的研究过程中完全使

用，也没有研究每个指标的重要性以及是否可以用来衡量开发者的贡献。

Zhou 等[18]提议从动机、能力和环境 3 个方面研究开发者的贡献，其中动机涉及各种动机，包括协作、社会、补偿、用户需求等。他们将指标应用于 Mozilla 和 Gnome 项目，然后通过机器学习方法研究了长期开发者的特征。Lima 提出了一套指标，旨在评估开发者的贡献，其涵盖了 4 个方面：代码贡献、方法的平均复杂度、错误介绍和错误修复。

结束语 本文首次提出了基于深度强化学习的开发者贡献度提升方法，并且探索了一种可以提高开发者贡献度的框架。本文设置了多组实验对比，并将该框架的实验结果和实际开发者轨迹进行了对比，实验结果验证了本文方法的有效性。

目前，本文只是初步地考虑了开发者执行的次数。除此之外，我们相信还有很多能反映开发者工作量的指标可以被纳入进来。同样，对于动作次数的考虑也需要涉及到动作的细粒度内容，如评价该动作会带来多大的影响。

在未来的工作中，一方面会继续探索全面客观地衡量开发者贡献度的方法以满足实际生产的需求。另一方面，我们也将探索对于一个项目来说，多个开发者之间如何协作来提高整体的贡献度。

参 考 文 献

[1] XUAN Q, GHAREHYAZIE M, DEVANBU P T, et al. Measuring the effect of social communications on individual working rhythms: A case study of open source software[C]// 2012 International Conference on Social Informatics. 2012: 78-85.

[2] SOWE S K, STAMELOS I, ANGELIS L. Understanding knowledge sharing activities in free/open source software projects: An empirical study[J]. Journal of Systems and Software, 2008, 3(81): 431-446.

[3] BACHMAN N, BERNSTEIN A. When process data quality affects the number of bugs: Correlations in software engineering datasets[C]// 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). 2010: 62-71.

[4] GOURLEY B A, DEVANBU P. Detecting patch submission and acceptance in oss projects[C]// Fourth International Workshop on Mining Software Repositories (MSR'07). ICSE, 2007: 26.

[5] HO J, ERMON S. Generative adversarial imitation learning [C]// Advances in Neural Information Processing Systems. 2016: 4565-4573.

[6] BOEH M, LI G H. Value-based software engineering: a case study[J]. Computer, 2003, 36(3): 33-41.

[7] MARLOW J, DABBISH L, HERBSLEB J. Impression formation in online peer production: Activity traces and personal profiles in github[C]// Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW '13). New York, NY, USA, 2013: 117-128.

[8] TSAY J, DABBISH L, HERBSLEB J. Influence of social and technical factors for evaluating contribution in github[C]// Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). New York, NY, USA, 2014: 356-366.

[9] X-lab. Github's digital annual report for 2019[OL]. https://github.com/X-lab2017/github-analysis-report-2019, 2020.

[10] YE D, LIU Z, SUN M, et al. Mastering Complex Control in MOBA Games with Deep Reinforcement Learning[C]// AAAI. 2020: 6672-6679.

[11] JIANG N, JIN S, DUAN Z, et al. RL-Duet: Online Music Accompaniment Generation Using Deep Reinforcement Learning [C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2020: 710-718.

[12] JING M, MA X, HUANG W, et al. Reinforcement Learning from Imperfect Demonstrations under Soft Expert Guidance [C]// AAAI. 2020: 5109-5116.

[13] LILLICRAP T P, HUNT J J, PRITZEL A, et al. Continuous control with deep reinforcement learning [J]. arXiv: 1509.02971, 2015.

[14] JENSE N, SCACCHI W. Role migration and advancement processes in ossd projects: A comparative case study[C]// 29th International Conference on Software Engineering (ICSE'07). 2007: 364-374.

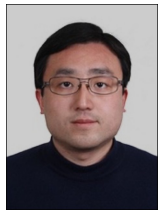
[15] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[J]. arXiv: 1707.06347, 2017.

[16] THUNG T F, BISSYANDE, LO D, et al. Network structure of social coding in github[C]// 17th European Conference on Software Maintenance and Reengineering. 2013: 323-326.

[17] GOUSIOS G, KALLIAMVAKOU E, SPINELLIS D. Measuring developer contribution from software repository data[C]// Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR '08). New York, NY, USA, 2008: 129-132.

[18] ZHOU M, MOCKUS A. Who will stay in the floss community? modeling participant's initial behavior[J]. IEEE Transactions on Software Engineering, 2015, 41(1): 82-99.

FAN Jia-kuan, born in 1995, master. His main research interests include reinforcement learning and multi agent reinforcement learning.



WANG Wei, born in 1979, professor, Ph.D, is a member of China Computer Federation. His main research interests include computational education and open source digital platform.