计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.191000087

基于特征提取的开源社区 Fork 摘要自动生成方法

张 超[1] 毛新军[1,2] 卢 遥[1]
1 国防科技大学计算机学院 长沙 410000
2 复杂系统软件工程重点实验室 长沙 410000
(rainstar817@sina.com)

摘 要 当前,基于 P/R 的分布式协同开发已经成为开源社区中的主导软件开发方式。开发者通过 Fork 复制软件项目的版本库,创建自身分支,并在新建分支中进行独立开发。由于 P/R 协同开发模型具有开放性、透明性和并行化等特征,开发人员在 Fork 项目时难以掌握项目的 Fork 概况,不知道其他开发人员是否已通过 Fork 开展相同或类似的开发工作,从而容易产生重复性的贡献和冗余性开发。针对这个问题,提出一种 Fork 摘要的自动生成方法以帮助项目管理者加强项目管控,避免冗余贡献,增强合作交流。该方法首先爬取开源社区中具有 Feature 和 Bug 标签信息的 Issue 数据,采用随机森林方法训练一个分类器模型,以对 Fork 特征进行分类;随后收集 Fork 分支的软件开发活动数据,采用 TextRank 算法生成 Fork 详细信息以解释 Fork 的主要目的;最后设计了一组组合规则及相应的算法来整合 Fork 的类别、特征和其他信息,以形成完整的 Fork 摘要。为了检验所提方法在指导分布式协同开发方面的有效性,在 Github 上进行了 30 组人工测试和 60 组实际案例测试。结果表明,所提方法生成的 Fork 摘要的准确率达到 67.2%,实验中 76%的项目管理者认为 Fork 摘要有助于更好地管理项目,加强沟通与合作。

关键词:开源软件;开源社区;Fork 摘要;分布式开发
中图法分类号 TP311

Approach of Automatic Fork Summary Generation in Open Source Community Based on Feature Extraction

ZHANG Chao[1], MAO Xin-jun[1,2] and LU Yao[1]
1 College of Computer Science and Technology, National University of Defense Technology, Changsha 410000, China
2 Key Laboratory of Complex System Software Engineering, Changsha 410000, China

Abstract At present, distributed collaborative development based on P/R has become the dominant software development method in open source community. Because of the openness, transparency and parallelism of the software development in P/R model, it is difficult for developers to obtain the complete Fork profile of the whole project, and know whether other developers have accomplished the same or similar development tasks, which are prone to duplicate contributions and redundant development. To solve this problem, this paper proposed an automatic generation method of Fork summary to help project managers strengthen project management, avoid redundant contributions, and enhance cooperation and communication among developers. The proposed method firstly crawls Issue data with feature and Bug label information in open source community, and trains a classifier model with random forest method to classify Fork features. Then, it collects the data of Fork branch's software development activities and uses TextRank algorithm to generate detailed Fork information to explain the main purpose of Fork activity. Finally, a set of combination rules and corresponding algorithm are designed to integrate Fork's categories, features and other information to form a complete Fork summary. In order to validate the effectiveness of the proposed method, 30 groups of manual tests and 60 groups of actual live study were conducted on Github. The results show that the accuracy of Fork summary generated by this method is 67.2%. In the experiment, 76% of project managers believe that Fork summary can help to better manage projects, and strengthen communication and cooperation.

Keywords Opens source, Open source community, Fork summary, Distributed cooperative development

到稿日期:2019-10-15 返修日期:2020-01-05 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划项目 (2018YFB1004202);NSFC (61532004)
This work was supported by the National Key R&D Program of China (2018YFB1004202) and Research on Mechanism and Method of Massive Online Collaborative Learning (61532004).
通信作者:毛新军(xjmao@nudt.edu.cn)

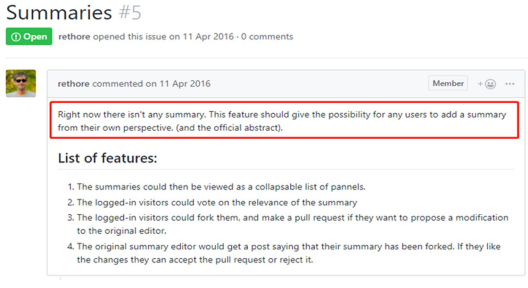
1 引言

在开源软件社区中，基于 P/R 的分布式协同开发已经成为开源社区的重要方式。Fork 是通过复制另一个存储库来创建新的软件存储库，从而将开源代码作为自己的代码进行更改的活动。开发人员可以自由地 Fork 公共存储库，并在自己的分支中进行更改[1-2]。Fork 是启动新贡献的一种方式。

然而，开源社区的快速发展给基于 Fork 的开发带来了一些挑战。一方面，开发者的快速增长带来了大量的 Fork 和贡献，丰富了开源社区生态的多样性。另一方面，随着 Fork 数目的不断增加，现有的 Fork 可视化工具已经无法对单个 Fork 中发生的事情以及项目的整体状况进行直观和系统的展示，因此开发人员不得不依赖手动方法逐个检索 Fork。此外，由于开发人员的经验和习惯差异较大，存在大量笔记不完整、功能不明确、信息不透明的 Fork，这些 Fork 可能会在一定程度上消耗开发人员的时间和精力，使他们无法直接理解其他开发人员所做贡献的目标和特性[3]。由于 Fork 信息不透明且缺乏合适的工具，用人工方法识别出大量 Fork 是非常困难且耗时的。因此，外围开发者很难找到所需的信息，核心开发人员很难做出正确的决策。

对于开发人员来说，其他人的 Fork 贡献了什么？自己所需特定目标的 Fork 在哪里？如何找到最佳 Fork 点？如何确定自己的贡献是否冗余？诸如此类的问题在 Fork 信息不透明的情况下很难得到快速解决，因此可能导致冗余开发、低效贡献[4-5]等问题。如图 1(a)所示，这个开发人员正在寻找一个好的 Fork 点，但是他无法快速识别每一个 Fork 点来避免重复开发。Gousios 等[6]总结了 Github 上 290 个项目中拒绝 Pull-Request 的 9 个原因，其中 23%是由于冗余开发(并行开发或替代其他 Pull-Request)。

(a)没有合适的 Fork 信息，开发者很难找到最优的 Fork 点



(b)一个维护人员请求为 Fork 制作摘要工具

图 1 很多开发者面临 Fork 信息获取困难的问题

Fig. 1 Many developers facing difficulties in obtaining Fork information

对于管理维护人员来说，利用传统的 Fork 工具管理整个项目很困难[7]。项目中 Fork 的贡献有哪些？如何通过 Fork 了解整个项目？如何确保所有 Fork 都能有效地工作，或者它们是否背离项目作者的初衷？Fork 的功能是否与其他的功能重复？管理维护人员由于不知道项目中发生了什么，很难高效地管理项目、开发设计、促进合作[8-9]。如图1(b)所示，维护人员希望有一个摘要能够概括开发贡献，以便于审查，并避免丢失贡献。Fork 信息的不透明性，限制了基于 Fork 的开发效率。事实上，效率是软件工程的首要关注点[10-11]。

针对上述问题，本文开展 Fork 摘要的调查研究，提出了一种自动生成 Fork 摘要的方法，以帮助开发人员获得 Fork 的准确信息描述，避免重复贡献，或找到目标 Fork 点；同时，帮助维护人员获得良好的 Fork 信息描述，以促进项目管理和协同开发。

本文第 1 节介绍开源社区的背景和基于 Fork 开发的现状，阐述当前 Fork 存在的问题；第 2 节介绍当前国内外的研究前沿、存在的优势和有待继续研究的问题；第 3 节对 Fork 摘要的定义、特性、功能进行具体分析；第 4 节重点阐述生成 Fork 摘要的方法；第 5 节验证 Fork 摘要的准确性和生成摘要方法的有效性；最后对全文进行总结。

2 相关工作

2.1 基于 Fork 的分布式开发

在社会编码兴起之前，Fork 传统上是指分割一个新的独立开发分支，以与原始项目竞争或取代原始项目。Gousios 等探索了 Ggithub 的 Pull-Request 模型，其中 Fork 是必不可少的组件[12-13]。他们的工作证实了 Fork 为社区参与者提供了更多的机会，但也强调了由于冗余开发和缺少协调等原因，大多数的 Pull-Request 被拒绝，只有很少的贡献被整合。此外，一些学者对开源和工业产品线开发中的 Fork 实践进行了研究[14-16]。但是，这些研究只揭示了所讨论的问题，没有提供任何解决办法。

2.2 Fork 开发的透明度

冗余开发是由于在基于 Fork 的开发中缺乏足够的透明度而造成的。在当前的现代社会编码平台中，透明度已经被证明是决策所必须考虑的因素[17]。例如，开发人员的活动或项目的受欢迎程度这些可视化的因素，会影响生态系统中的决策和声誉。但是，随着 Fork 的增加，已有工具很难提供一个清晰、直观的 Fork 视图。针对这些问题，Zhou 等[8]开发了一个名为“forks insight”的工具，它通过关键字分类提供 Github 中 Fork 的特征视图；该团队还提出了一种方法和相应的工具，使用源代码分析、社区检测和信息检索技术来自动识别和总结项目的 Fork 特征[4]。

2.3 Fork 摘要生成

在 Github 中，Utkarsh 开发了一个名为“Lovely Forks”的浏览器插件，可以帮助他注意到 Github 项目中值得注意的 Fork。Github 的另一位开发人员 Daniel 开发了一个用于显示按创建日期排序的 Fork 并查找活动 Fork 的工具[1)]。然

1) https://github.com/dblock/faf

而，现有的研究成果只包括 Fork 分类、Fork 特征提取等，对于 Fork 摘要，尤其是用流畅的自然语言简洁、直观地描述 Fork 信息，尚未涉及[18]。开发人员需要通过人工查询操作来获取 Fork 信息，不能直接获取 Fork 摘要。因此，本文致力于思考 Fork 摘要的定义和功能，以及如何使用 Fork 摘要让开发人员直观、全面地了解 Fork 活动。

3　Fork 摘要分析

学术界对 Fork 摘要的定义尚不明确。维基百科中对摘要的定义是，“摘要：在不改变词义的情况下，用不同的词语和句子缩短一段文字或一篇文章”，或者“摘要：是指对重要内容的简明、准确的描述，正确的摘要，没有主观的解释和评论，以便读者在最短的时间内掌握内容”。同样，我们需要生成一个简洁的 Fork 自然语言描述，展现 Fork 的重要特性，提高读者获取 Fork 信息的效率。因此，本文将要生成的 Fork 摘要定义为：Fork 的一种自然语言描述，来自任意时间段的两个提交(Commit)，并包含大多数功能和特点，甚至列出 Commit 或 Pull-Request 的具体细节并分析贡献变化(文件更改、特性更改等)[19]。

当开发人员浏览 Fork 时，他们通常会关注 Fork 中包含的 Pull-Request，因为成功合并的 Pull-Request 包含已验证的特征信息，即贡献的表述。因此，很多研究人员一直致力于基于 Pull-Request 的冗余贡献检测[3,20]，甚至有人质疑：为什么要做 Fork 摘要，而不是 Pull-Request 摘要？

然而，许多研究者只从开发人员的角度而不是从维护人员的角度对特定的点做出贡献[13]。研究 Fork 摘要的原因有：1)Fork 的功能更加全面，不仅体现在 Pull-Request 上，还体现在 Issue，Commit，Comment，Wiki，Code 等方面[21-22]；2)Fork 中的 Pull-Request 不是完全不重叠的，它们之间存在依赖关系、关联关系、主从关系，仅列出 Pull-Request 特点是不可能准确推断 Fork 特点的[23]；3)Pull-Request 是低效且隐蔽的，冗余和无效的 Pull-Request 常常被拒绝合并，一些已完成的合并 Pull-Request 可能包含对项目的重要贡献，但无法引起开发人员的注意；4)有些开发人员在 Fork 之后继续提交代码，但从不提交 Fork，这并不意味着他们没有贡献。因此，本文更加关注 Fork 摘要，旨在更全面、准确地表达 Fork 的特性，同时也参考 Pull-Request 信息。

将 Commit 作为生成 Fork 摘要的最小粒度，其原因有如下几个方面。1)我们认为 Pull-Request 是一种特殊类型的 Issue，Issue 是基于 Commit 的，Commit 是 Pull-Request 的最小操作单元。2)如上所述，有些开发人员只在 Fork 之后不断使用 Commit 迭代贡献，而从不进行 Pull-Request 操作，从不将贡献合并到 Fork 的上游分支中，这些人只是为自己的 Fork 工作，从不主动向其他人展示所取得的贡献；或者其他人也做了很多开发并提交 Pull-Request 请求，但是直到最后一个 Pull-Request 请求的那一刻，他们所做的所有工作都不会被直接发现和分享。每次开发人员执行 Commit 操作时，都有可能提供新的和有用的功能和特性，这些功能和特性不应被忽略。3)一些管理者更关注 Fork 中某个特定时间段的贡献，或者某个特定时间段内的 Commit 贡献，这就需要将贡献精确定位到特定的时间点。因此，选择 Commit 作为最小粒度是合适的。4)以 Commit 作为最小粒度，可以清楚地看到，与原始 Commit 相比，新 Commit 存在哪些更改。

生成 Fork 摘要在实际的 P/R 开发中具有以下意义和价值。1)Fork 摘要允许开发人员直观、高效地获取 Fork 信息，从而增强协作和管理，避免冗余贡献，帮助开发者和管理者提高开发效率[24]。2)为了进一步的研究，我们基于任意两个 Commit 生成摘要，这意味着可以在 Fork 中快速获得任意一个区段的 Commit 摘要。因此，我们可以探讨多个 Commit 之间的关系，例如检测具有相同功能和相同类别的 Commit。基于这样的特性，可以检测开发者是否会做出相似的贡献，从而避免重复贡献，加强合作沟通。

Fork 和 Commit 的关系如图 2 所示，其中节点是 Commit，边是 Commit 与父类的关系，合并的 Commit 有多个父类关系，虚线代表未来计划。

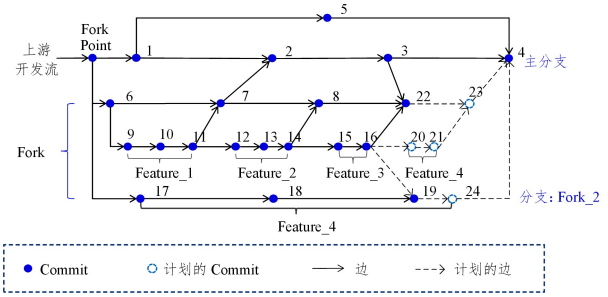


图 2　Fork 中 Commit 和 upstream 的历史记录

Fig. 2　History of Commit and upstream in Fork

以图 2 为例进行说明：有一个项目的维护者计划从当前的 Fork 点开启一段包含 4 个功能的开发分支，并用 Issue 的方式来实现。开发者 A 在 Fork_1_branch_1 中进行贡献，并在 Fork_1_branch_2 中执行详细的代码工作；在完成 Feature_1，Feature_2 和 Feature_3 等功能时，请求 Fork_1_branch_1 合并。如果我们是项目的维护者，并且想了解开发人员是如何工作的，那么最常见但效率较低的方法是逐个浏览 Commit 来获取 Feature_1，Feature_2 和 Feature_3。然而，通过 Fork 摘要可以直接得到两个 Commit 的任意部分的数据，以获取 Feature_1，Feature_2 和 Feature_3，并推断下一个 Commit 20 和 Commit 21 是关于 Feature_4 的。此外还可以看到，另一个开发人员 B 在 Fork_2 中贡献了 Feature_4，尽管他到目前为止还没有提交合并请求。作为维护者，我们可以建议开发人员 A 从 Commit 16 合并 Fork 分支到 Fork_2 的 Commit 19。这样一方面可以避免重复开发 Commit 20 和 Commit 21，因为 Commit 17，Commit 18 和 Commit 19 都是关于 Feature_4 的贡献)；另一方面能提升开发人员和维护人员的协调效率。更重要的是，这样能使得希望贡献 Feature_4 的新开发人员很容易地从 Commit 1 到 Commit 4 和 Commit 17 到 Commit 19 获得摘要，从而找到 Commit 19 的最佳 Fork 点，而不像大多数开发人员从 Fork 点开始一步一步重新贡献，为开发人员节省了多余的贡献。

4 Fork 摘要自动生成方法

Fork 摘要自动生成方法的框架如图 3 所示。首先，从 Github 的项目历史数据中获取带有功能标签的 Issue 数据，用于训练分类器模型。然后，当有新的 Commit 输入时，提取 Commit 数据并将其分为两部分使用。将一部分数据放入分类器模型进行预测，得到 Commit 特征；用另一部分数据生成 Commit 内容描述。之后，结合 Commit 特性和内容，通过特定的规则生成 Commit 摘要。最后，使用已定义的模板将所有的 Commit 摘要合并为一个 Fork 摘要。

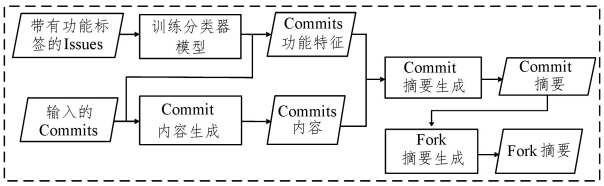


图 3 Fork 摘要自动生成方法的框架

Fig. 3 Framework of automatically generating Fork summary

4.1 训练分类器模型

从图 3 中可以看到，所提方法使用 Github 项目中带有特性标签的数据和输入 Commit 数据之间的关系来对 Commit 分类。因此，本文将 Issue 数据作为机器学习算法的输入并对其进行预处理，以训练分类器模型。最后，用户输入新的 Commit 到训练模型，以预测其特征类别。

1)数据预处理。对文本数据的预处理包括数据清理、数据标记和停用字删除。从 Github 项目中爬取的原数据包含大量不适合计算的“脏”数据：

①含有链接的数据，如 Commit 消息“This change is src=https://review.io/ btn.svg”。

②重复数据，即两个或更多具有相同内容的 Commit 正文/消息。

③不完整数据，如带有空消息的 Commit。

④数据是格式错误的数据，如非标准化格式的数据及乱码等。

数据是机器学习算法的输入，本文将数据标记为“feature”“bug”和“contribution”3 类(在以后的工作中会更加细化类别)。含有“feature”和“bug”标签的数据或者没有这两种标签但是文本中含有“feature”和“bug”的相关关键字的数据，被标记为对应的“feature”和“bug”标签；其他数据则被标记为“contribution”。最后，移除常见的停用词(如“the”和“a”)，因为这些字出现频率太高且对分类几乎没有影响。

2)数据转换。本文将预处理后的数据转换为向量空间模型(VSM)中的多维向量。利用 python-sklearn 开发类中的 CountVector 模型将文本单词转换为词频矩阵，例如包含元素 $text[i][j]$ 的矩阵，表示 i 类型文本下 j 单词的单词频率。然后，根据词频-逆向文件频率(TF-IDF)模型，利用下式将 CountVector 处理后的计数矩阵转换为标准化的 TF-IDF 矩阵。

$$wi;k=tfi;k\times idfk \tag{1}$$

其中，$tfi;k$ 是指给定术语 k 出现在文本 i 中的频率；$idfk$ 是基于包含它的文本数量的一般重要性度量，被称为逆术语频率；$wi;k$ 的值是 $text[i][j]$ 中 k 项的权重。

3)训练模型。分类器模型的目标是将 Commit 分类为“feature”“bug”和“contribution”3 类，将一个新的 Commit 输入模型后，输出其类别。有多种机器学习算法可以用于分类，如支持向量机(SVM)、AdaBoost、逻辑回归、神经网络、决策树、随机森林和 k 近邻等[25]。在这项研究中，大部分数据都是文本形式且长度很短，根据此特点，本文选择支持向量机、逻辑回归、随机森林 3 种方法分别进行测试，择优而用。结果显示，随机森林的分类效果最好，我们采用管道(Pipeline)技术和循环调参的方式多次迭代优化随机森林分类模型。

4.2 生成 Commit 摘要

当获取新的输入 Commit 时，我们需要得到 Commit 的信息描述，即 Commit 摘要。在文本自动摘要算法中，TF-IDF 是最常用且最容易实现的算法，但其效果不如 TextRank。TextRank 是受 Google 的 PageRank 算法启发而设计的一种权重算法，用于自动生成文本摘要。它使用投票的原则，让每个词都为它的邻居(任期窗口)投赞成票，投票的权重取决于它自己的票数。该算法把每个句子看作图中的一个节点。如果两个句子之间有相似性，则认为对应的两个节点之间有一条无向边，权重为相似性[26]。用 TextRank 算法计算出的最重要的句子可被看作摘要。

TextRank 算法生成提交内容的过程如下：

1)预处理。将输入文本集 T 的内容拆分成句子，即 $T=[S_1,S_2,\cdots,S_m]$，S_m 表示第 m 个句子。构建图 $G=(V,E)$，其中 V 是句子集，E 是边。分割句子，删除停止词，得到 $S_i=[t_{i,1},t_{i,2},\cdots,t_{i,m}]$，$t_{i,j}\in S_j$ 是保留的候选关键字。

2)计算句子相似度。在图 G 中建立边缘集 E。根据句子之间的内容覆盖率，给定两个句子 S_i 和 S_j，按式(2)计算它们之间的相似度：

$$Similarity(S_i,S_j)=\frac{|\{w_k|w_k\in S_i\&w_k\in S_j\}|}{\log(|S_i|)+\log(|S_j|)} \tag{2}$$

如果两个句子之间的相似度大于给定的阈值，则认为这两个句子在语义上是相关的，在图中是相连的，即边缘权重 $w_{j,i}=Similarity(S_i,S_j)$。

3)计算句子得分。根据式(2)，通过迭代传播权重计算每个句子的得分。

4)提取候选摘要。对前一步得到的句子分数进行倒序排序，抽取重要性最高的语句作为候选摘要语句。

5)形成摘要。根据单词或句子的要求，从候选摘要语句中提取句子，形成摘要。

根据获得的 Commit 分类和 Commit 内容，本文制定一组规则将它们集成到完整的 Commit 摘要中。我们认为，完整的 Commit 摘要应该包括作者、功能、内容、状态和变化元素，例如：

Commit 1：Alfonso Alonso Lorenzo commits a bug about ['Set dirty_read to FindOne']

Status：merged

Change：Fix OpenSSL tvOS build

Status 让开发人员知道当前的提交状态，而 Change 则突出了 Commit 与其父级之间的区别。因此，我们得出 Commit 摘要的模板如下：

@Commit-i：@author commits a @feature{ feature，bug，contribution } about @content.

Status：@Status

Change：@Change

其中，@Commit-i 表示 Fork 中的第 i 次 Commit，@author 表示提交者，@feature 和@content 是 Commit 特性，@feature 包含 feature，bug，contribution 这 3 个元素；@Status 和@Change 从 Commit 中提取相关性数据。

4.3　生成 Fork 摘要

Fork 由 Commit 的多个区段组成，Fork 摘要也由多个 Commit 摘要组成。因此，我们分析得到的是 Commit 摘要集，将其集成到最终的 Fork 摘要中[27]。生成 Fork 摘要的框架如图 4 所示。首先，收集 Fork 中的所有 Commit 摘要进行统计分析，得到 feature，bug 和 contribution 3 类统计数据；然后，将这些零碎的数据放入合成器中，生成最终的 Fork 摘要。

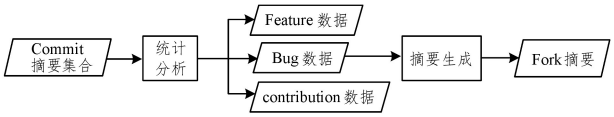


图 4　合成 Fork 摘要的框架

Fig. 4　Framework for generateing Fork summary

1）Commit 数据统计。将多个 Commit 摘要拆解，统计每个特征类别的贡献量和内容。相关算法如算法 1 所示。

算法 1　Commit 数据统计

输入：lists：the list of commit summaries

输出：the list data of feature，bug，other contribution

```
1. let number_i be a list
2. let content_i be a list
3. for summary in lists:
4.    for label in (feature,bug,contribution):
5.       if getFeature(summary).equal(label):
6.          let k be a int
7.          k ←intof(label)
8.          number_k++
9.          append getContent(summary) into content_k
10. return number_k,content_k
```

2）合成摘要。到目前为止，我们已经获得了每个特性类别的 Commit 统计信息。下面将构建一组合成规则，将这些数据放在模板的相应位置，获取最终的 Fork 摘要。Fork 摘要模板的定义如下：

模板 1：

@fork_summary= commit：@b_commit to @e_commit of fork：@fork_name contain @fork_content.

模板 2：

@fork_content=$\sum$@feature _content

@feature _content=@numk@feature about @contentk

($k \in$(feature，bug，contribution))

其中，@fork_summary 是我们想要的最终结果。在模板中，@b_commit 和@e_commit 表示用户选择的起始 Commit 和结束 Commit。方便起见，我们使用 commit sha 的最后 4 位数字来表示 Commit 编号。@fork_name 是从输入数据中获取的 Fork 的名称，@fork_content 是生成的 Fork 的内容描述。模板 1 显示了 Fork 摘要的结构和元素。模板 2 显示了 Fork 的内容是如何形成的。k 是 feature，bug 和 contribution 3 种类别的任意一种。因此，变量@numk 和@contentk 对应于每个 k 条件的数量和内容，这是在前面的统计过程中获得的数据。@feature 是 Commit 的功能类别。因此，@feature _content 是每个功能的内容，@fork_content 是所有功能的总和。总的来说，模板 2 显示了 Fork 对特定特性所做的详细工作。

当然，不是每个 Fork 摘要都含有这 3 类信息。在大多数情况下，我们只能得到一小部分数据。那么，如何处理丢失的部分呢？本文构造了以下规则来匹配不同的数据默认值，以确保特性类别缺失、特性内容缺失、Commit 特性冲突、Fork 内容缺失情况下 Fork 摘要的语言流畅性。

Rule1：if (@numk==0)：@feature _content=null

Rule2：if (@contentk ==null)：@feature _content=null

Rule3：if (b_commit [-4：-1]==e_commit [-4：-1])：b_commit=e_commit=e_commit [0：4]

Rule4：if ($\sum$@numk =0)：@fork_summary ="Sorry，nothing contribution"

5　实验与评估

本节通过实验及评估来分析本文所提出的 Fork 摘要自动生成方法的可行性和有效性。

5.1　实验数据集

为了评估方法，我们构建了一个基于 Github 开源项目的数据集，其中包含 20 000 对 Issue/Pull-Request 数据。我们的目的是将 Commit 数据分为"feature"和"bug"两类，因此带有这两个标签的数据是我们首先要考虑的。更具体地说，上文提到了使用带有"feature"和"bug"特定标签的 Issue 数据来训练分类器模型，因此我们筛选出了必须有"feature"或"bug"标签的项目。我们发现，无论搜索哪个项目，"feature"或"bug"标签总是所有标签中的一小部分，约占总数的 10%～15%。为了获得足够的数据，我们在一个项目中选择的 Issue 和 Pull-Request 的总和必须在 500 条以上。根据 Github 上"feature"和"bug"标签的搜索结果，我们选择了前 30 个项目并爬取数据。表 1 列出了项目的统计数据。

表 1　数据集中的开源项目

Table 1　Open source projects in data set

项目	Issue	Pull-Request	带有"feature"标签的 Issue/Pull-Request	带有"bug"标签的 Issue/Pull-Request
safe-ios	326	605	52	42
FieryVoid	454	353	11	20
opencast	0	998	111	195
intellij-rust	1829	2287	184	10
PrestaShop	3406	11100	147	1993
metasploit-framework	3189	8850	1089	1409
ezplatform-admin-ui	0	1027	0	302
ansible	23473	34609	11365	28561
Baystation12	1800	3021	781	0
Yogstation-TG	711	6101	920	288
SemanticMediaWiki	1148	3162	148	74
open-event-android	1084	1291	126	0

另外，我们注意到在获取数据时有些地方存在争议。例如，项目“ezplatform-admin-ui”只有“bug”标签，而项目“open-event-android”没有任何“bug”标签，只有 126 个“feature”标签。更具体地说，在一些项目中，管理者会将这些数据标记成不同的标签。同样地，在一些项目中，一个关于“feature”的开发活动也可能在其他项目中被定义为“bug”，甚至有特性显著、本该被定义的数据却完全没有被定义。打标签是一种人工操作，很大程度上取决于操作人员的主观判断，因此，同名的标签在不同项目中可能代表不同的意义。

因此，当项目之间的标签定义存在冲突或重叠时，可能会影响分类模型的准确性。为了排除开发人员主观因素对实验结果的影响，我们将项目分成两部分，并使用不同类型的标签。在表 2 中，将只有“feature”标签的数据放入 B 组训练 feature-分类器，将只含有“bug”标签的项目数据放入 C 组训练 bug-分类器。对于 A 组，由于项目维护者的分类比较清晰，我们使用同时带有“feature”和“bug”标签的项目数据来训练 feature-bug-分类器。

表 2 数据集的划分

Table 2 Partitioning of data set

组	标签类别	项目数量	Issue 和 PR 数量
A	“feature”和“bug”	18	12187
B	“feature”	5	3190
C	“bug”	7	4623

5.2 实验问题及结果分析

实验试图回答以下两方面的问题：1)方法生成的 Fork 摘要在多大程度上准确地描述了 Fork？2)本文方法如何帮助维护人员获得良好的 Fork 描述来管理开发或增强协调？

5.2.1 Fork 摘要的准确性

Fork 摘要是一种自然的语言描述，可以直接向开发人员提供信息概览。在 Github 中，开发人员最关心 Fork 中的贡献是什么以及有无贡献丢失，这就是完整性和有效性。其次，他们也会关注描述的语句是否流畅等。我们认为，一个好的 Fork 摘要必须包含以下特征：

1)Fork 摘要的描述要准确；

2)Fork 摘要的描述要完整；

3)Fork 分类要准确；

4)Fork 摘要的文本表达要流畅。

开发者主要从内容准确性、内容完整性、语句流畅性、特征提取正确性等方面判断摘要是否准确，并且具有一定的主观性。另外，Fork 摘要实际上是 Commit 内容的集中和细化。如果 Commit 的数量太大，则很难确本文生成的 Fork 摘要能够涵盖所有 Commit 特性。我们认为 Fork 摘要的准确性还与它包含的 Commit 数有关。因此，本文定义 commits/max_commits 变量来消除数量的影响。通过考虑影响 Fork 摘要准确度的因素，提出了如下的准确度评价公式：

$$Acc\_rate=\frac{k_1\times Con_{Acc}+k_2\times Con_{Int}+k_3\times Sta_{Flu}+k_4\times Fea_{Pre}}{k_1+k_2+k_3+k_4}\times\frac{commits}{Max\_commits} \tag{3}$$

其中，Con_{Acc} 表示内容准确性，它描述生成的摘要与真实 Fork 内容的匹配程度；Con_{Int} 表示摘要覆盖选定 Commit 内容的程度；Sta_{Flu} 用于表示摘要陈述是否流畅易读；Fea_{Pre} 是指特征分类是否准确。此外，*commits* 表示 Commit 数量；*Max_commits* 表示 Fork 中含有 Commit 的最大数量。参数 k_1，k_2，k_3，k_4 是每个因子的权重，初始值设为 1。

为了评估 Fork 摘要的准确性，首先评估特征分类器的准确性。我们使用人工验证的方式，在 Github 项目中随机选择 30 组 Fork，让学校实验小组和部分社区开发者进行测试。

特征分类器的实验结果如表 3 所列。含有“feature”和“bug”标签的数据被分到对应的类别，其他数据类型被定义为“contribution”。可以看到，A 组中每个标签的预测值都在 0.5 以上，召回率很高，最大值(0.66)和最小值(0.59)之间的差异(0.07)在 0.1 以内，非常小。在 BC 组中，“feature”和“bug”的准确度都在 0.7 以上，但“contribution”的准确度太低。这是因为 B 组、C 组的原始数据中对“feature”和“bug”标签的定义有重叠、冲突或外延，从而缩小了“contribution”数据的范围。经计算，A 组的平均预测率 Fea_{Pre} 为 0.63，BC 组的 Fea_{Pre} 为 0.59。

表 3 分类器的训练结果

Table 3 Training results of classifier

组	标签	精准度	召回率	F1-分数	支持用例
A	“contribution”	0.59	0.79	0.67	896
	“feature”	0.66	0.78	0.58	686
	“bug”	0.64	0.67	0.72	400
BC	“contribution”	0.35	0.37	0.36	724
	“feature”	0.73	0.79	0.63	441
	“bug”	0.71	0.69	0.75	352

通过收集实际测试数据 Con_{Acc}，Con_{Int} 和 Sta_{Flu}，可以计算出 Fea_{Pre} 和 *Max_num_commit*。然后，对于每组数据，通过 30 组实验得到最终的 *Acc_rate*，如图 5 所示。

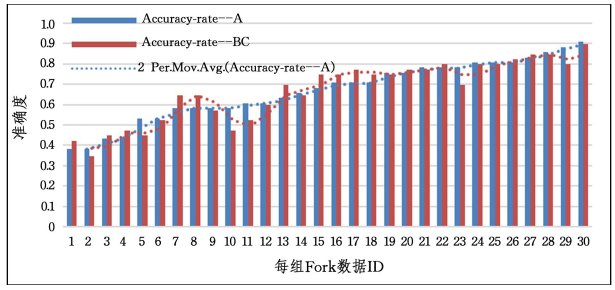


图 5 Fork 摘要准确度测试

Fig. 5 Fork summary accuracy test

可以看出，A 组 Fork 摘要的准确率最低为 0.383，最高为 0.908，平均准确率为 0.672；BC 组 Fork 摘要的准确率为 0.348～0.897，平均准确率为 0.668。最终，准确率在一个可接受的范围内，因此本文算法是相对准确的。

然而，通过与移动平均线 A，BC 以及 ID 2，7，8，10，11 和 23 等差距较大的数据比较，可发现 ConAcc 是关键点，因为分类器根据 A 组和 BC 组的原始数据给出了不同的分类，使得人们对 Fork 摘要的内容准确性有不同的看法；尤其是上述 ID 中有争议的数字，其“contribution”类别的 Commit 比其他

类别多，表 3 中 BC 组的“contribution”精度最低。

经过分析发现，在精度最低的前 5 组数据中，每个 Fork 包含的 Commit 数相对较多，接近表 4 所列 24 个测试的最高值。此外还可以发现，在这些测试的 Fork 中，“contribution”类别的 Commit 数量相对较多。在初始算法设计中，“contribution”代表了除“feature”和“bug”之外的所有贡献，是一个范围大且无法准确判断的范畴。因此，这一类的数目越大，Fork 摘要在一定程度上就越不准确。

表 4　准确度最低/最高的 5 组测试数据

Table 4　Top five groups of lowest/highest accuracy data

	ID	Accuracy-rate	“feature”	“bug”	“contribution”
5 组精度最低的数据	19	0.383	7	5	8
	15	0.383	5	1	16
	5	0.433	2	5	4
	13	0.433	1	6	7
	2	0.533	2	5	10
5 组精度最高的数据	30	0.808	4	0	0
	12	0.833	0	6	0
	8	0.858	0	2	0
	18	0.883	11	0	0
	28	0.908	6	0	0

我们反复检查了低精度数据，发现有一些未定义和未注释的 Commit，如图 6 所示。这些 Commit 中没有任何注释，即使是唯一的描述性信息也很模糊，无法通过人工审查来判断。因此，这些数据确实影响了本文算法的准确性。相反，前 5 组精度最高的数据并没有任何模棱两可的贡献，所有的“feature”和“bug”都可以准确判断，所以开发人员会认为生成的摘要更准确。

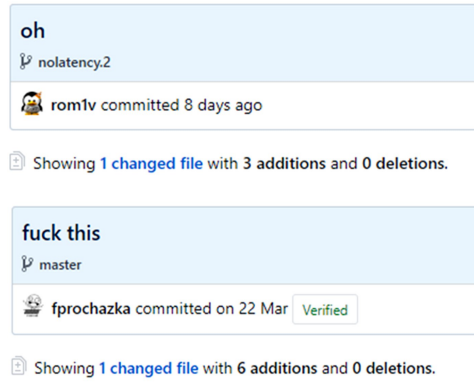


图 6　表述模糊的 Commit

Fig. 6　Ambiguous commits

实验表明，本文方法生成的 Fork 摘要在一般情况下可以达到 0.672 的准确率。

5.2.2　Fork 摘要生成方法的有效性

本文选择了 17 段持续贡献且与 Fork 摘要主题相关的 Commit 进行实际案例测试。特别地，将开发人员和维护人员分开，以验证本文方法是否同时适用于这两者。我们将项目 Fork 到自己的私有代码仓库，生成 Fork 摘要，并向源 Fork 发送合并请求，最后根据源 Fork 维护者的实际评估来研究本文方法对开发人员是否有帮助。实例研究的结果如表 5 所列。

表 5　实际案例的测试结果

Table 5　Results of live studies

项目	作者类别	Commits	等待审查	拒绝	接受	合并
devilutionX	maintainer	#bf4a8c-274abd	0	0	1	0
Xamarin.Forms	maintainer	#e095e2-3c12f9	0	0	0	1
EntityFrameworkCore	developer	#a6fe1c-525a15	0	1	0	0
lazydocker	developer	#f0650a-dba014	1	0	0	0
scrcpy	maintainer	#10069c-6901e9	0	0	0	1
scrcpy	developer	#772a49-6dc0fd	0	0	1	0
DeepFaceLab	developer	#bf78d3-29daad	0	0	0	1
DeepFaceLab	developer	#338e13-73f78b	1	0	0	0
semaphore	maintainer	#a65eb2-8f8fad	0	0	0	1
semaphore	developer	#ecde61-b5ef1e	0	1	0	0
semaphore	developer	#769d57-7b15ce	0	1	0	0
SyliusWishlistPlugin	maintainer	#9085ea-8a5d4f	1	0	0	0
SyliusWishlistPlugin	developer	#f90ff0-9284d2	0	1	0	0
a10-ansible	developer	#ecd486-be997b	0	0	0	1
sherlock	maintainer	#c4c4a5-054a24	0	0	1	0
ansibullbot	maintainer	#5e0add-6ebff7	1	0	0	0
silverstripe-installer	maintainer	#16009c-ef4982	0	1	0	0
总计			4	5	3	5

从表 5 可以看出，在 17 个 Commit 区段中，4 个 Pull-Request 被拒绝，5 个仍在等待审查，3 个被作者接受，5 个被合并。通过与开发人员沟通，我们认为已接受和合并的 Pull-Request 有助于开发。结果表明，本文方法可以在一定程度上帮助开发人员和维护人员。

通过研究 4 个 Pull-Request 被拒绝的原因可以发现，在某些情况下，摘要中存在一些错误，本文算法还有很大的改进空间。

表 6 列出了被拒绝的 Pull-Request，可以看到 Commit #80ad34 和 #44e1f37 中的“fix”和“more”消息没有构成完整的陈述语句，因此基于这些 Commit 生成的摘要的表达并不能构成流畅的陈述语句。在第二个摘要中，Commit #9284d22 消息是“Yaml is stupid”，因此生成的摘要没有意义。在第三个摘要中，Commit #16009c 和 #74f717 具有相同的消息“Fix main.php path”，且与 #8a507e 重复，因此摘要包含重复的内容，开发人员显然会拒绝这个摘要。

表 6 被拒绝的 Fork 摘要

Table 6 Rejected Fork summaries

Commit 编号	Commit 内容	Fork 摘要
#769d57	fix dumb error	commit #769d57c to #7b15ce of Fork semaphore-installer/Tom Whiston contain 3 bugs of [fix dumb error, fix, fix casting issue] and 2 contributions of [use middleware creator for all project api endpoints, more]
#80ad34	fix	
#1aa834	use middleware creator for all project api endpoints	
#44e1f37	more	
#7b15ce	fix casting issue	
#f90ff0	Fixed coding standards	commit #f90ff0 to #9284d2 of Fork SyliusWishlistPlugin/mamazu contain 1 feature of [Added coding stadnards], 3 bugs of [Fixed coding standards, Fixed coding standards path, Fixed coding standards path] and 1 contribution of [Yaml is stupid]
#0e3a07	Added coding stadnards	
#662d3b	Fixed coding standards path	
#6856ea	Fixed coding standards path	
#9284d2	Yaml is stupid	
#16009c	Fix main. php path	commit #16009c to #ef4982 of Fork silverstripe-installer/Ingo Schommer contain 2 features of [Removed stylesheet from frameworkmissing file, Adjust phpunit path to framework] and 3 bugs of [Fix main. php path, Fix main. php path, Fix main. php path in install. php]
#74f717	Fix main. php path	
#f73996	Removed stylesheet from frameworkmissing file	
#8a507e	Fix main. php path in install. php	
#ef4982	Adjust phpunit path to framework	

那么，本文方法如何帮助维护人员获得良好的 Fork 描述并以此来管理开发或增强协调？

对于此问题，我们做了更深入的研究。邀请上述实验中的开发人员进行同样的实验，并与他们逐一讨论了其赞成或反对的理由。表 7 列出了 30 组测试的结果和本文方法能帮助项目维护者的原因。可以看出，维护人员比外围开发人员更加关心如何知道其他人的开发情况。实验结果中，6 个项目(20.0%)的维护人员认为本文方法可以帮助他们提高协调合作，避免重复开发；7 个项目(23%)的维护人员认为用本文方法可以较好地管理项目开发进度，通过详细的讨论我们知道他们更喜欢对整个项目进行管理，尤其是对分支开发人员的贡献进行审查；4 个项目(5.8%)的人员认为本文生成的 Fork 摘要能够提供一个好的信息表述。我们认为好的描述是最重要的原因，因为它是提供人们所需信息的基础。

表 7 实例研究结果

Table 7 Case study results

结果	原因	数量
Approved	Good overview	4
Approved	Good for coordination	6
Approved	Avoidding dulplicated development	6
Approved	Good for progress management	7
Rejected	Not good overview	2
Rejected	Usefulless	3
Rejected	Unreliable summary	2

总的来说，实验数据中“接受”的比率(76.7%)在一定程度上证明了本文方法能够帮助维护人员获得良好的 Fork 描述，以促进项目管理开发或加强协调。

在实际开源软件开发中，本文方法的性能可能更弱，毕竟不是每个人都对 Fork 摘要有需求。但作为一种方法和工具，本文研究是有意义的。

结束语 本文分析了由于 Fork 开发模式的不透明性导致开发人员在 Fork 项目时难以掌握项目分支和贡献的概况，从而容易产生重复性的贡献和冗余性开发的问题，并且提出了一种生成 Fork 摘要的方法。我们首先训练了一个 Commit 特征分类的机器学习模型，然后通过自然语言处理算法生成 Commit 摘要，并将 Commit 摘要整合成为 Fork 摘要。在 Github 中进行了 30 组人工测试，并对 60 个 Fork 的实际案例进行了测试，以评估 Fork 摘要的准确性和实用性。结果表明，本文方法生成的 Fork 摘要达到了 67.2%的准确率，能帮助实验中 76%的项目管理者更好地管理项目，加强开发者间的沟通合作。

当然，本文研究也存在一些不足和有待改进的地方。在方法层面，对 Fork 特征的分类还不够准确，Fork 数据没有被充分利用；在实验层面，很难精准定位到对 Fork 摘要存在需求的开发者群体，对方法的有效性检验不够深入，缺乏深入的量化分析。下一步，我们将在现有工作的基础上展开相应的研究工作，优化现有算法，细化特征分类和摘要生成规则；并且将特征深入到代码层面，使摘要能够反映代码层面的特征变化。

参 考 文 献

[1] JIANG J, LO D, HE J, et al. Why and how developers fork what from whom in GitHub[J]. Empirical Software Engineering, 2016, 22(1): 1-32.

[2] BITZER J, SCHRODER P. The Impact of Entry and Competition by Open Source Software on Innovation Activity[J]. Industrial Organization, 2005.

[3] REN L, ZHOU S, KASTNER C, et al. Identifying Redundancies in Fork-based Development[C]// 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2019.

[4] YU Y, LI Z, YIN G, et al. A dataset of duplicate pull-requests in github[C]// Mining Software Repositories. 2018: 22-25.

[5] GOUSIOS G, PINZGER M, VAN DEURSEN A, et al. An exploratory study of the pull-based software development model [C]// International Conference on Software Engineering. 2014: 345-355.

[6] REN L, ZHOU S, KÄSTNER C. Forks insight: providing an

overview of GitHub forks[C]// Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. ACM, 2018.

[7] NYMAN L, MIKKONEN T. To Fork or Not to Fork: Fork Motivations in SourceForge Projects[C]// Open Source Systems: Grounding Research - 7th IFIP WG 2.13 International Conference(OSS 2011). DBLP, 2011.

[8] ZHOU S, STANCIULESCU S, LEBENICH O, et al. Identifying features in forks[C]// International Conference on Software Engineering. 2018: 105-116.

[9] YIN G, WANG T, LIU B X, et al. Survey of software data mining for open source ecosystem[J]. Journal of Software, 2018, 29(8): 2258-2271.

[10] SADOWSKI C, AFTANDILIAN E, EAGLE A, et al. Lessons from building static analysis tools at Google[J]. Communications of the ACM, 2018, 61(4): 58-66.

[11] SALTON G, BUCKLEY C. Term-weighting approaches in automatic text retrieval[J]. Information Processing and Management, 1988, 24(5): 323-328.

[12] JAMES G, WITTEN D, HASTIE T, et al. An Introduction to Statistical Learning[M]. Springer New York, 2013.

[13] GOUSIOS G, ZAIDMAN A, STOREY M, et al. Work practices and challenges in pull-based development: the contributor's perspective[C]// International Conference on Software Engineering. 2015: 285-296.

[14] VASILESCU B, BLINCOE K, XUAN Q, et al. The sky is not the limit: multitasking across GitHub projects[C]// International Conference on Software Engineering. 2016: 994-1005.

[15] ROBLES, GREGORIO, GONZÁLEZBARAHONA J. A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes[C]// Open Source System. 2012: 1-4.

[16] LI L S, REN Z L, LI X C, et al. How are Issue Units Linked? Empirical Study on the PSECLinking Behavior in GitHub[C]// Asia-Pacific Software Engineering Conference(APSEC). 2018.

[17] DABBISH L, STUART C, TSAY J, et al. Social coding in GitHub: transparency and collaboration in an open software repository[C]// Conference on Computer Supported Cooperative Work. 2012: 1277-1286.

[18] DABBISH L, STUART C, TSAY J, et al. Leveraging Transparency[J]. IEEE Software, 2013, 30(1): 37-43.

[19] Gail Cecile Murphy. Lightweight structural summarization as an aid to software evolution[OL]. https://core.ac.uk/display/20786603.

[20] ZHU J, ZHOU M, MOCKUS A. Effectiveness of code contribution: from patch-based to pull-request-based tools[C]// Acm Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2016.

[21] POSHYVANYK D, MARCUS A. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code[C]// International Conference on Program Comprehension. IEEE Computer Society, 2007: 37-48.

[22] STOREY M A D, CHENG L T, BULL R I, et al. Shared waypoints and social tagging to support collaboration in software development[C]// Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work(CSCW 2006). Banff, Alberta, Canada, ACM, 2006.

[23] KUHN A, DUCASSE S, GIRBA T, et al. Semantic clustering: Identifying topics in source code[J]. Information & Software Technology, 2007, 49(3): 230-243.

[24] STANCIULESCU S, SCHULZE S, WASOWSKI A. Forked and integrated variants in an open-source firmware project[C]// 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2015.

[25] KHATAVKAR V, KULKARNI P. Comparison of Support Vector Machines With and Without Latent Semantic Analysis for Document Classification [C]// Data Management, Analytics and Innovation. 2018: 263--274.

[26] LANDAUER T K. Latent Semantic Analysis[M]// Encyclopedia of Cognitive Science. Berlin: Springer, 2006.

[27] BERGER T, NAIR D, RUBLACK R, et al. Three Cases of Feature-Based Variability Modeling in Industry[C]// Model Driven Engineering Languages and Systems. 2014: 302-319.

ZHANG Chao, born in 1991, postgraduate, is member of China Computer Federation. His main research interests include software engineering and open source community.

MAO Xin-jun, born in 1970, Ph.D, professor, is member of China Computer Federation. His main research interests include software engineering and open source community.