

一、引言

近年来，开源运动和社会编码对软件开发有着巨大的影响。随着越来越多的应用程序转移到了云上，Github 已经成为了管理软件开发以及发现已有代码的首选方法。在 GitHub 中，开发人员可以相互 follow 以形成社交网络，通过 star 和 watch 功能跟踪项目的更新，通过 commit 和 pull request 功能贡献代码，或者通过 issue 功能参与新功能设计或 bug 修复的讨论。在这样一个复杂的系统中，识别有影响力的开发者和项目对于开发者能力的提高和开源社区的繁荣具有重要意义，同时在服务推荐方面也有重要的应用。

二、问题描述

本项目实现了对仓库和开发者的影响力分析。如今，Github 不仅是管理软件开发以及开源代码的首选方法，也同时由于其如 follow, star, watch 等功能的存在而兼具了社交网络的特征。如果能利用其社交网络的特征，分析得到仓库和开发者的影响力，将能够更有利于平台用户获悉当前更有价值的仓库，找到所关心领域的权威开发者。然而，Github 本身并没有推出一个多年来所有的仓库与开发者的综合排名。因此，本项目选取了从 2016 年 1 月起至 2021 年 12 月止的 6 年时间中 Github 日志中的 pull request 事件相关数据，使用 BiRank, PageRank 以及 BurstBiRank 三种算法分别对仓库和开发者的影响力进行了分析。BiRank 和 PageRank 算法的输入为仓库 ID，开发者 ID；BurstBiRank 的输入为仓库 ID，开发者 ID，以及事件创建时间。算法的输出为按照排名排列的仓库 ID 列表与开发者 ID 列表。在评价中，我们将各个算法得到的结果与单纯使用 watch 次数（用以评估仓库）和 follower 数量（用以评估开发者）统计信息得到的结果进行相关性分析，并且通过交叉验证，对三种算法得到的 TopK 的仓库集合和开发者集合进行集合运算分别得到其分别准确率、召回率和 F1 值，从而对三种算法得到的结果的相关性进行了评估。

三、方法

在数据预处理中，如何处理庞大的 github 日志数据是一个难题。由于本项目的二分图网络是基于 Pull Request 事件建立，并且由于 BurstBiRank 需要使用突发性计算权值，而突发性是基于时间间隔度量，所以需要选取仓库 ID，开发者 ID，创建时间。此外，由于总体时间间隔较长，用户名以及仓库名会有变动，导致同一 ID 有多个名称，故同时也选取出项目名称和开发者名称，以便后续能够实现 ID 与名称的唯一对应。我们将抽取出的 96939748 条记录存放在中间表中，并在后续操作中不断从该中间表中使用 ODPS SQL 读取数据。

在 BiRank 算法与 PageRank 算法所需处理中，我们随后通过将 ID 按照从大到小的方式排列，并将这些 ID 在表中的行号赋予其作为新 ID，实现原项目 ID 与新项目 ID、原开发者 ID 与新开发者 ID 的一一对应，得到 NEW_ID 表。这一操作的主要目的是减少算法构建矩阵的大小。最终我们基于新 ID 构建 Weight 表，将每对新项目 ID 和新开发者 ID 的边上权值置为 1。为使得算法结果能够同时展示 ID 与名称，我们也将新 ID 与时间上最靠后的名称进行对应，去除掉被用户抛弃的名称，以保证每个 ID 唯一对应一个最新的名称，最终得到 NEW_ID_NAME 表。算法中将读入 Weight 表用以构建图网络，并将在结果展示时对算法结果表、NEW_ID 表与 NEW_ID_NAME 表做合并。

在 BurstBiRank 算法所需的处理中，我们认为记录数过少的开发者具有的影响力水平较低，对最终结果影响小，但这部分开发者人数较多，所以，需要首先去除总的记录数 ≤ 10 的开发者，以减少数据量。与 BiRank 和 PageRank 所需处理不同的是，我们需要对日期数据进行较多处理来得到权值。通过项目 ID 与开发者 ID 将全部记录分区后，将各个分区下一条记录的创建时间与当前记录做差得到日期差，并在各个分区分别计算出均值 m 与标准差 σ ，从而计算得 $burst = \frac{\sigma - m}{\sigma + m}$ ，并记权值为 $-burst$ 。ID 重新编号的逻辑和 ID 与名称的一一对应的逻辑与上述一致。此外，还需再次将新 ID 与 BiRank 和 PageRank 所需预处理的新 ID 一一对应，以便后续将三个算法的结果进行交叉验证。

由于算法结果需要与 watch 和 follower 指标结果做比较，所以还需用到 Watch 事件的日志数据以及 ods_github_users 表中的 follow 相关数据。我们通过对每一个仓库被 watch 的次数求和得到其 watch 总数，并按照 watch 总数进行排名，将结果记为仓库 ID 与 watch 排名的对应。在对 follower 指标的处理中，需要选取表中的开发者名称，跟随者列表和更新时间三个字段，并将跟随者列表处理为跟随者数目。由于 ods_github_users 表中以开发者名称唯一代表某个用户，所以需要开发者名称以 ID 替代。但由于开发者名称与 ID 间为多对一的关系，故最终选择每个 ID 最新时间的 follower 数作为其对应的粉丝数。并且在最终，我们仍然需要将这些 ID 对应到算法处理后的新 ID 上。

以上全部操作都没有使用先将数据拉去到本地，再从本地读取然后处理的形式；而是全部通过创建中间表的方式在线运行，处理的记录的数量级可以达到约 10^9 ，但是所需时间却很少，数据获取的速率大致为 $5 * 10^7$ 条/分钟，并且可以在 20 分钟内完成全部数据的获取。

我们通过对用户和仓库分别进行排名的评估从而实现对影响力的分析。排名较高者，则影响力较大。利用 PullRequest 事件的记录，构建了开发者与项目之间的一个复杂网络。在这个网络中，构建三种排名算法。算法分为两类，一类是二分图的排名算法，将开发者节点和项目节点视为两类节点，分别构建了 BiRank 和 BurstBiRank 算法。另一类是单分网络，将开发者和仓库视为同类节点，构建了 PageRank 算法。

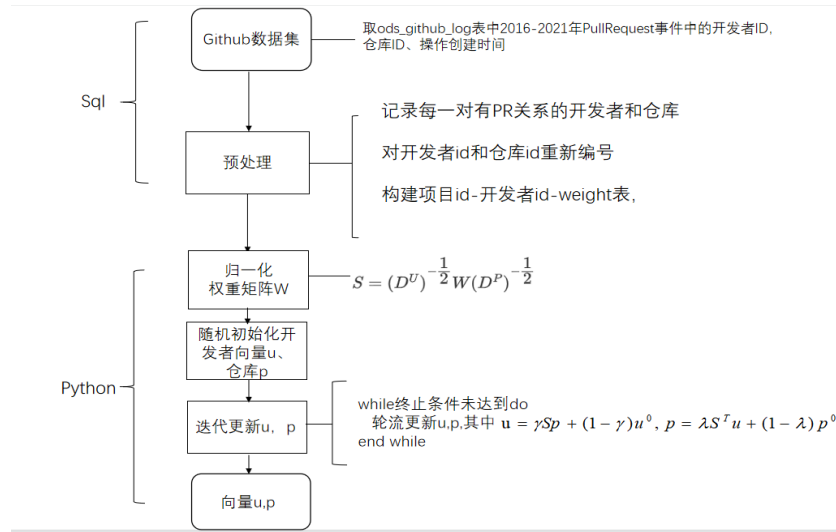


图 3.1

BiRank 算法的构建过程如图 3.1 所示，它迭代地给顶点分配分数，并最终收敛到一个唯一的固定排名。

该算法输入开发者和仓库之间的权重矩阵 W 以及超参数 γ 和 λ ，在权重矩阵中 $w_{ij} = 1$ 则代表开发者 i 在仓库 j 里面有 PullRequest 记录。首先初始化 u_i^0 、 p_i^0 ，分别代表开发者 i 和仓库 j 的初始值。接着对 W 矩阵进行归一化得到矩阵 $S = (D^U)^{-\frac{1}{2}} W (D^P)^{-\frac{1}{2}}$ 。此后迭代计算向量 u 和向量 p 的值，最终收敛到一个唯一的固定值。与传统的基于随机游走的方法相比，BiRank 迭代优化正则化函数，在查询向量的指导下平滑图形。

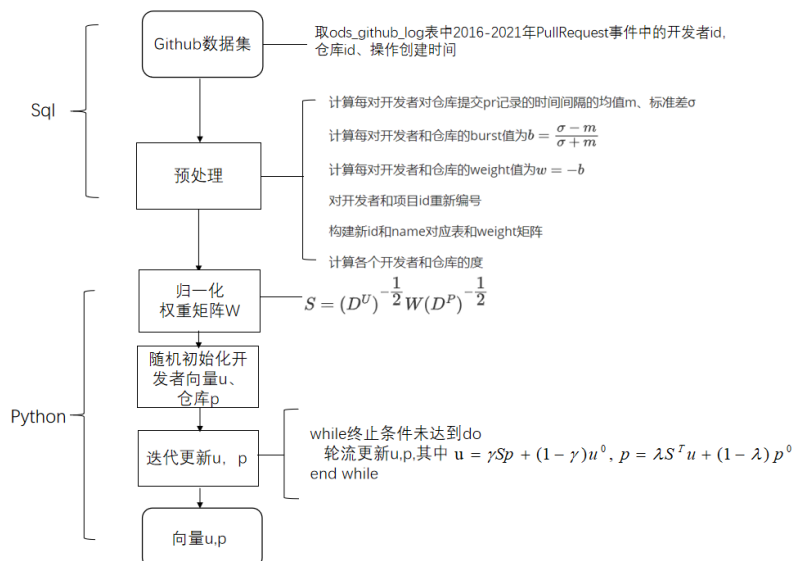


图 3.2

对人类行为模式的研究揭示，人类的行为具有间歇性，人们倾向于在短时间内表现出强烈的活动，然后长时间活动减少甚至没有活动。基于以上事实，BurstBiRank 算法使用突发性来对偏离周期的行为做测量。这种人类行为模式及其背后的规律已经在人类动力学领域得到了广泛的研究。例如，你可能会在空闲时花一整个下午在 YouTube 上观看娱乐视频，但之后在工作日很少访问 YouTube。BurstBiRank 算法的流程图如图 3.2 所示，BurstBiRank 是在 BiRank 的基础上利用突发性行为计算得来初始化 weight 矩阵。

突发值的定义为 $b_{ij} = \frac{\sigma - m}{\sigma + m}$ ，（ b_{ij} 代表开发者 i 在项目 j 上得突发性值） σ 代表时间间隔的标准差， m 代表时间间隔的均值。 b_{ij} 的范围介于 -1 到 1 之间。 b_{ij} 大于 0，代表行为是突发的， b_{ij} 越大，代表行为的突发性越强。当 b_{ij} 为负值时，代表是周期性行为。W 权重矩阵初始化为 Burst 值矩阵的线性函数。接着沿用 BiRank 算法，得出更新最终收敛到一个唯一的固定值。

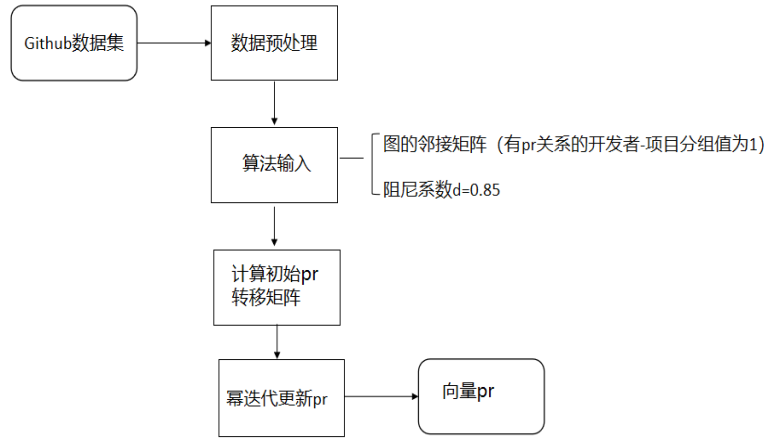


图 3.2

PageRank 的算法流程图如图 3.3 所示，PageRank 将顶点的重要性分数估计为随机行走过程的平稳分布。从一个顶点开始，顶点根据边权重随机跳到相邻顶点。将 PageRank 应用于开发者项目的二分网络，忽略节点的类型，将开发者和仓库视为同类节点。在实验中，阻尼系数 d 设置为 0.85。之后通过幂迭代运算，得到开发者和仓库的 pr 值。

四、结果及评价

通过将三个算法得到的排名结果进行交叉验证，可以从准确率、召回率和 F1 值三个方面对三个结果的相关性进行评估；而通过将根据构建的算法得到的结果与根据简单指标直接排名得到的结果进行 Spearman 相关性分析，我们能够对二者的相对排名的相关性有所了解。

在三个算法的相关性评估中，分别建立影响力 Top1000 开发者集合 A_{bi} 、 A_{page} 、 $A_{burstbi}$ 以及影响力 Top1000 仓库集合 U_{bi} 、 U_{page} 、 $U_{burstbi}$ ，并构建影响力开发者参考集 A_2 以及影响力仓库参考集 U_2 如下：

$$A_2 = (A_{burstbi} \cap A_{bi}) \cup (A_{page} \cap A_{bi}) \cup (A_{burstbi} \cap A_{page})$$

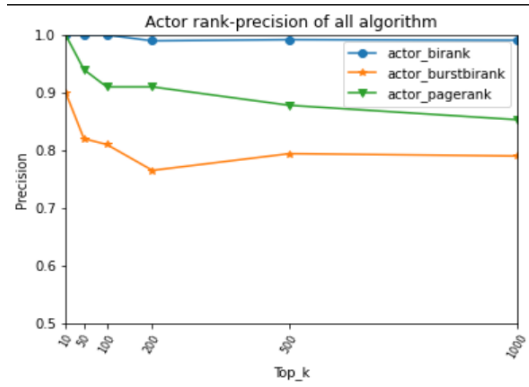
$$U_2 = (U_{burstbi} \cap U_{bi}) \cup (U_{page} \cap U_{bi}) \cup (U_{burstbi} \cap U_{page})$$

以下为开发者在 BiRank 算法中的准确率、召回率以及 F1 值，其余形式上也与之相似：

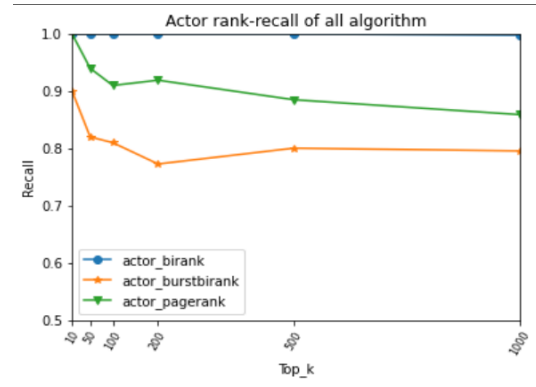
$$Precision_{bi} = \frac{|A_2 \cap A_{bi}|}{A_{bi}}$$

$$Recall_{bi} = \frac{|A_2 \cap A_{bi}|}{A_2}$$

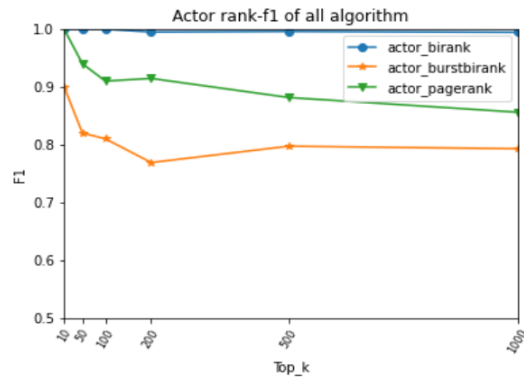
$$F1_{bi} = \frac{2 * Precision_{bi} * Recall_{bi}}{Precision_{bi} + Recall_{bi}}$$



(a)



(b)



(c)

图 4.1

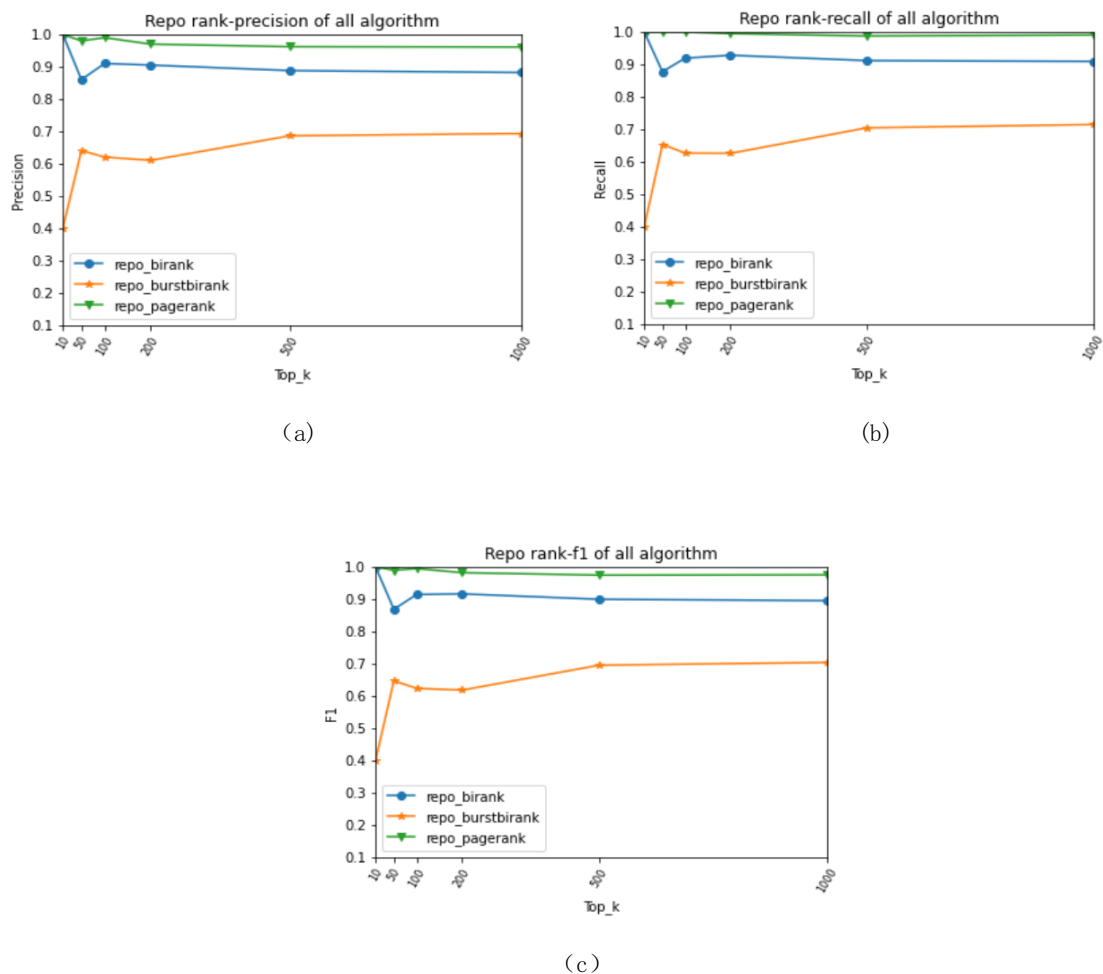


图 4.2

从图 4.1.a 可以看出，对于开发者而言，BiRank 算法的精度最高，其次是 BurstBiRank，PageRank 的精度最低，表明周期性行为并不一定会让开发者更有影响力；关于召回率（见图 4.1.b），BiRank 最高，表明 BiRank 认为有影响力集合和开发者参考集的交集更接近开发者参考集。F1-Measure 是准确率和召回率的综合得分。观察 F1 值图（见图 4.1.C）BiRank 算法优于其他算法，表明 BurstBiRank 算法认定影响力高的开发者在另外两个算法中也具有较高的认可度。对于仓库而言，如图 4.2，PageRank 在准确度、召回率和 F1 值都有较高的分数，表明 PageRank 认定的有影响力的仓库在另外两个算法中也有较高的认可度。

在算法得到的结果与通过指标衡量的结果的相关性分析中，选取的指标为开发者的 follower 数量与仓库的 watch 数量。分别关注在算法中 Top10，Top20，Top50 的开发者和仓库，可以得到其分别在 follower 数量或者 watch 数量的排名。以下表 4.1 中以 BiRank 中开发者排名对比为例，我们将 BiRank 算法排名记为 rank，将 follower 排名记为 follower_rank。

| new_actor_id | actor_name | rank | follower_rank |
|--------------|--------------------------|------|---------------|
| 1389704 | direwolf-github | 1 | 128125 |
| 1065416 | xmo-odoo | 2 | 128816 |
| 2730410 | codacy-badger | 4 | 19847 |
| 1172471 | gitter-badger | 5 | 4982 |
| 2263149 | QualitySoftwareDeveloper | 6 | 159060 |
| 2121899 | idsb3t1 | 8 | 184369 |
| 5148049 | codeserver-service-qa | 9 | 173762 |
| 6575172 | turbocanary-admin | 10 | 180074 |
| 1103564 | slskopytko | 11 | 152899 |
| 6270054 | shiftright-staging | 12 | 174154 |

表 4.1

类似分析所有表格，记 rank 中第 i 个数为变量 x_i ，watch_rank 或 follower_rank 中第 i 个数为变量 y_i ，计算相关系数 $\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum_i (x_i - \bar{x})^2)(\sum_i (y_i - \bar{y})^2)}}$ ，得到结果如下表 4.2 及 4.3 所示：

| | BiRank | BurstBiRank | PageRank |
|-------|--------|-------------|----------|
| Top10 | 0.066 | -0.115 | 0.199 |
| Top20 | 0.180 | -0.311 | 0.067 |
| Top50 | 0.105 | 0.129 | 0.098 |

表 4.2 仓库算法排名与 watch 排名的相关系数

| | BiRank | BurstBiRank | PageRank |
|-------|--------|-------------|----------|
| Top10 | 0.714 | NAN | 0.714 |
| Top20 | 0.314 | -0.999 | 0.348 |
| Top50 | -0.244 | -0.357 | -0.366 |

表 4.3 开发者算法排名与 follower 排名的相关系数

由于开发者的 followers 原始数据覆盖不全，但完全覆盖 follower 数量多的开发者，故在 BurstBiRank 的 Top10 中由于记录数少于 2 而无法计算相关系数，但仍然从侧面反映算法得到的结果与 follower 数量指标的结果有较大差异。此外，根据上表能够发现相关系数均处于 (-0.2, 0.2) 区间内，故三种算法得到的仓库排名与

仓库的 watch 次数指标排名几乎不相关，且三种算法得到的开发者排名与开发者的 follower 数量指标排名也几乎不相关。

同时，可以抽取各个算法共同抽取到的影响力 Top10 的仓库与开发者的特征如下。如表 4.4 所示，通过研究 star 数、fork 数、contributors 数以及 pull requests 数，能够发现被筛选出的仓库多具有贡献者人数多或者 pull request 次数多的特点，与算法本身都为基于仓库与开发者的关系上构建的事实相符合。如表 4.5 所示，共同筛选出的开发者都在一定程度上具有 follower 人数少，仓库内容为无意义的文件或者均为对其余开发者仓库 fork 而来，但具有 repository 数量极大，或者上一年 contribution 很多的特点。依据上述判断基本可确认这些开发者均为机器人。由此可见，在对有影响力的开发者的发现上，三种算法的效果都不佳。

此外，也有部分开发者不符合上述特征，这种例外是由于数据被删除而导致的。以 turbocanary-admin 为例分析，能够发现其在无仓库的情况下在上一年度有 201 次 contribution，但事实上在无仓库时无法做 contribution 操作，所以能够推断该机器人清除了原有仓库。观察其活动情况（见图 4.3），可以发现该开发者在过去一年中突然停止活动。与之类似，其余开发者也有部分数据删除情况。

我们可以从三种算法的构建思路上进行分析，从而找到为何三种算法倾向于认为具有上述特征的仓库和开发者具有较高影响力。BiRank 和 PageRank 构建时，会在具有 Pull Request 关系的仓库与开发者之间构建一条边，且不设置边上权值。而 BurstBiRank 虽在构建时会为行为具有周期性的用户和仓库赋予更大的权值，但是机器人的行为也具有很大的周期性（见图 4.4）。由此，当某个仓库的开发者人数众多时，该仓库的排名会被抬高；相对的，当某个开发者拥有的仓库数量众多时，该开发者的排名也会被抬高，即使这些仓库本身的影响力水平较低，也能通过数量弥补。由此，那些拥有几千个仓库的机器人会被认为具有很大的影响力。然而这一缺陷在对具有影响力的仓库的发现上表现不明显。一方面由于机器人的社交属性缺失的特性，很少有多机器人共同在某一无意义仓库进行 Pull Request 操作；另一方面有价值的仓库有很多度较大的开发者参与其中，由此三种算法共同发现的有

影响力的仓库更有参考意义。这一发现也可以通过将算法发现的有影响力的仓库与其在 watch 数排名中的情况得出。我们统计得到的 watchrank 共有 14800777 个仓库参与排名，在三个算法共同发现的所有仓库中，watchrank 排名最低为 26883，处于该排名的前 0.18%；而 followerrank 共有 7280595 个开发者参与排名，在三个算法共同发现的所有开发者中，followerrank 排名最低为 184369，处于该排名的前 2.5%。客观而言，通过这三个算法所做的影响力分析确实能够剔除大部分影响力低的仓库和开发者。

| new_repo_id | repo_name | BiRank | BurstBiRank | PageRank | watchrank | star | fork | contributors | pullrequests |
|-------------|--|--------|-------------|----------|-----------|-------|-------|--------------|--------------|
| 1614471 | firstcontributions/first-contributions | 1 | 1 | 1 | 437 | 23.4k | 44.9k | 5000+ | 245 |
| 170062 | jlord/patchwork | 2 | 2 | 2 | 24375 | 1k | 36.2k | 5000+ | 2.4k |
| 17203 | octocat/Spoon-Knife | 4 | 4 | 4 | 9653 | 11.1k | 127k | - | 5k+ |
| 3854851 | zero-to-mastery/start-here-guidelines | 5 | 11 | 5 | 16991 | 1.7k | 12.1k | 5000+ | 49 |
| 267844 | JetBrains/swot | 8 | 22 | 9 | 26883 | 1.2k | 13.6k | - | 44 |
| 71898 | DefinitelyTyped/DefinitelyTyped | 9 | 3 | 6 | 236 | 39.3k | 27.3k | 5000+ | 192 |
| 267651 | udacity/create-your-own-adventure | 12 | 46 | 13 | 18689 | 497 | 9.6k | 2839 | 4.8k |
| 12790580 | education/GitHubGraduation-2021 | 13 | 5 | 11 | 18242 | 1.4k | 7.7k | 5000+ | 1.3k |
| 1812687 | MicrosoftDocs/azure-docs | 14 | 7 | 14 | 3357 | 7.1k | 16.3k | 5000+ | 590 |

表 4.4 影响力 Top10 仓库的排名及特征

| new_actor_id | actor_name | BiRank | BurstBiRank | PageRank | follower rank | followers | repositories | last year contribution | 仓库内容 |
|--------------|-----------------|--------|-------------|----------|---------------|-----------|--------------|------------------------|------------------------------------|
| 1389704 | direwolf-github | 1 | 1 | 1 | 128125 | 5 | 1.6k | 2917 | no sense |
| 1065416 | xmo-odoo | 2 | 4 | 2 | 128816 | 3 | 8 | 546 | make sense, but forked from others |

| new_actor_id | actor_name | BiRank | Burst BiRank | Page Rank | follower rank | followers | repositories | last year contribution | 仓库内容 |
|--------------|--------------------------|--------|--------------|-----------|---------------|-----------|--------------|------------------------|------------------------------------|
| 5631549 | codeserver-test1 | 3 | 5 | 3 | - | - | - | - | - |
| 2730410 | codacy-badger | 4 | 2 | 4 | 19847 | 97 | 2.9k | 3800 | make sense, but forked from others |
| 1172471 | gitter-badger | 5 | 3 | 5 | 4982 | 378 | 5k+ | - | make sense, but forked from others |
| 2263149 | QualitySoftwareDeveloper | 6 | 6 | 6 | 159060 | - | - | - | - |
| 4823957 | git2e | 7 | 7 | 7 | - | 0 | 48 | 0 | no sense |
| 2121899 | idsb3t1 | 8 | 8 | 8 | 184369 | 1 | 3.8k | 0 | no sense |
| 5148049 | codeserver-service-qa | 9 | 9 | 9 | 173762 | 0 | 1 | 2708 | no sense |
| 6575172 | turbocanary-admin | 10 | 13 | 10 | 180074 | 0 | 0 | 201 | empty |

表 4.5 影响力 Top10 开发者的排名及特征

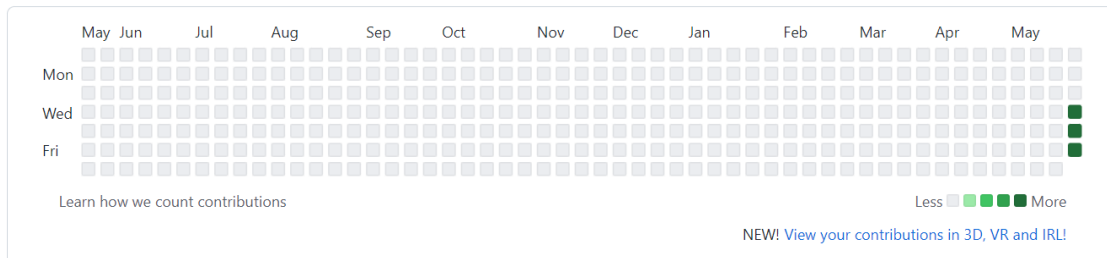


图 4.3 开发者 turbocanary-admin 的活动情况

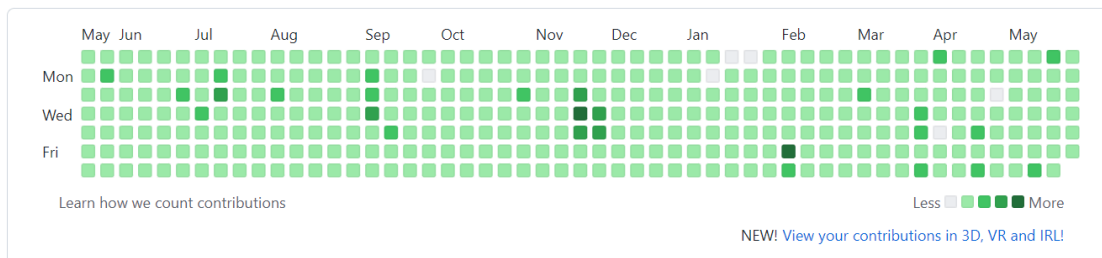


图 4.4 典型机器人开发者的活动情况

五、相关工作

对于复杂网络的影响力节点的分析，许多研究人员都提出了提出了不同的排名算法，以下是对相关工作的一些总结。

在图排序的算法中，PageRank 和 HITS 是最突出的方法。PageRank 是将顶点的重要性分数估计为随机行走过程的平稳分布——从一个顶点开始，然后根据边权重随机跳到相邻顶点。HITS 是假设每个顶点有两个权重：hub 和 authority，将图转换为二分图。如果一个顶点被许多具有中心分数的顶点链接，则该顶点具有高权威分数，如果该顶点链接到许多权威顶点，则该顶点具有高中心分数。

许多算法的变体都是基于 PageRank 和 HITS 的。Haveliwala [1]提出了主题敏感的 PageRank(也称为 personalized PageRank)。Ng 等人[2]研究了 PageRank 和 HITS 的稳定性，发现在某些情况下，HITS 对图结构中的小扰动更敏感。他们提出了两种变体——Randomized HITS and Subspace HITS，则可以产生更稳定的排名。后来 Ding 等人[3]把 PageRank 和 HITS 统一在一个规范化的排序框架下。受 PageRank 的离散时间马氏过程解释的启发，Liu 等人[4]也提出了基于连续时间马氏过程的 BrowseRank，利用用户行为数据进行页面重要性排序。为了将顶点和边上的边信息结合到排序中，Gao 等人[5]通过基于顶点和边上的特征学习转移矩阵，以半监督的方式扩展了 PageRank。

除了对单步图的排序，二分图的排序算法也有很多的相关研究。He 等人[6]提出了一种基于优化的二部网络排序方法 BiRank。Xu 等人[7] 将奇异值分解应用于二分

网络，提出了 SVDRank 和 SVDARank。Moreone 等人[8]将 k-核心分解方法推广到二分网络，指出在生态共生网络中，最大 k-核心节点的灭绝会使生态系统达到崩溃的临界点。

就排名技术而言，它们都通过在图上迭代传播分数来排名，要么通过类似 PageRank 的随机行走，要么通过类似 HITS 的迭代过程。

六、结论

本项目是对 Github 进行影响力分析，即对 Github 中的开发者和项目进行分别进行排序。本项目采用了大批量的日志数据与用户数据进行分析，提取了部分统计数据，并且构建了三种算法，实现了算法结果与统计量之间的相关性分析、算法与算法之间的横向比较。我们在对 Github2016 年至 2021 年大量日志数据的进行预处理后，通过利用开发者在仓库中的 9 千多万条 pullrequest 记录，构建了 PageRank、BiRank 和 BurstBiRank 算法来实现对节点的评估。PageRank 是基于随机游走的方法；BiRank 则是迭代优化正则化函数，在查询向量的指导下平滑图形；BurstBiRank 在 BiRank 的基础上，利用突发性行为初始化权重矩阵。得到三个算法的结果后，我们发现机器人的提交行为对开发者的排名造成了一定的干扰，但对项目的排名影响不大。对三个算法进行横向评估后发现，BiRank 算法分析得出的有影响力的节点同样在另外两个算法中也有较高的认可度，这验证了 BiRank 算法的有效性。此外，将算法结果与统计量进行相关性分析后，可以发现算法的结果与根据统计量得出的结果几乎不具有相关性。这一发现与我们采用的是项目与开发者的关联关系构建图网络具有很大的关联性。在后续工作中，我们考虑首先在预处理中实现机器人检测与剔除，并加入其他开发者和项目的其他关系来综合衡量排名，以提高排名的准确度。

[1] T. H. Haveliwala, "Topic-sensitive PageRank," in Proc. 11th Int. Conf. World Wide Web, 2002, pp. 517 - 526.

- [2] A. Y. Ng, A. X. Zheng, and M. I. Jordan, “Stable algorithms for link analysis,” in Proc. 24th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, 2001, pp. 258 – 266.
- [3] C. Ding, X. He, P. Husbands, H. Zha, and H. D. Simon, “PageRank, hits and a unified framework for link analysis,” in Proc. 25th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, 2002, pp. 353 – 354.
- [4] Y. Liu, et al., “BrowseRank: Letting web users vote for page importance,” in Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, 2008, pp. 451 – 458.
- [5] B. Gao, T.-Y. Liu, W. Wei, T. Wang, and H. Li, “Semi-supervised ranking on very large graphs with rich metadata,” in Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2011, pp. 96 – 104.
- [6] X. He, M. Gao, M.-Y. Kan, and D. Wang, “BiRank: towards ranking on bipartite graphs,” IEEE Transactions on Knowledge and Data, vol. 29, no. 1, pp. 57 – 71, 2016.
- [7] S. Xu, P. Wang, and C. Zhang, “Identification of influential spreaders in bipartite networks: A singular value decomposition approach,” Physica A: Statistical Mechanics and Its Applications, vol. 513, pp. 297 – 306, 2019.
- [8] F. Morone, G. Del Ferraro, and H. A. Makse, “The k-core as a predictor of structural collapse in mutualistic ecosystems,” Nature Physics, vol. 15, no. 1, pp. 95 – 102, 2019.

附录 1. 分工

徐涵钰：

1. 数据预处理：使用 SQL 取数据及处理数据
2. Spearman 相关性分析
3. 交叉验证三个算法的结果

孙文文：

1. 使用 python 实现 birank 算法

任丽伊：

1. 使用 python 实现 burstbirank 算法

廖欣：

1. 使用 python 实现 pagerank 算法