Github 影响力分析

徐涵钰，何雨晴

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

一方面，Github 具有的社交网络特征为我们提供了使用图模型进行影响力分析的可能性；另一方面，Github 本身缺乏对多年来所有的仓库与开发者的综合排名。因此，本项目考虑，通过选取从 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

3.1 数据预处理

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，所以需要将开发者名称以 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。

3.2 算法构建

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。

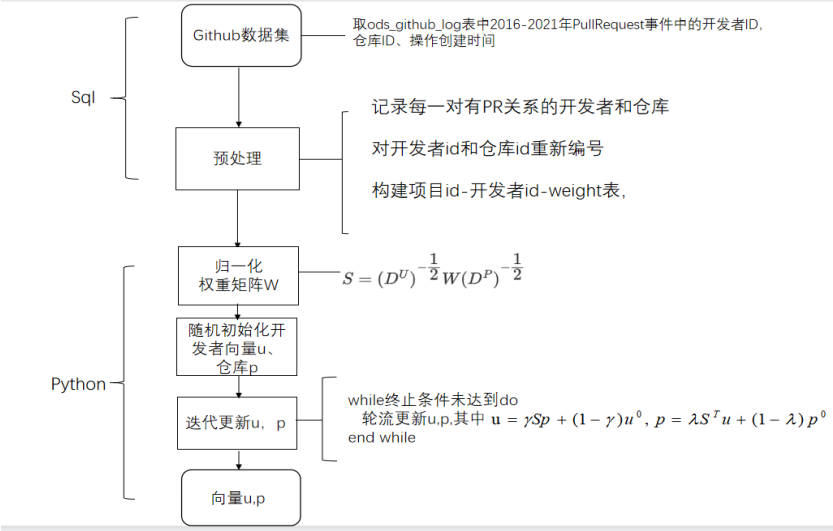


图 3.1

3.2.1 BiRank 算法

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。

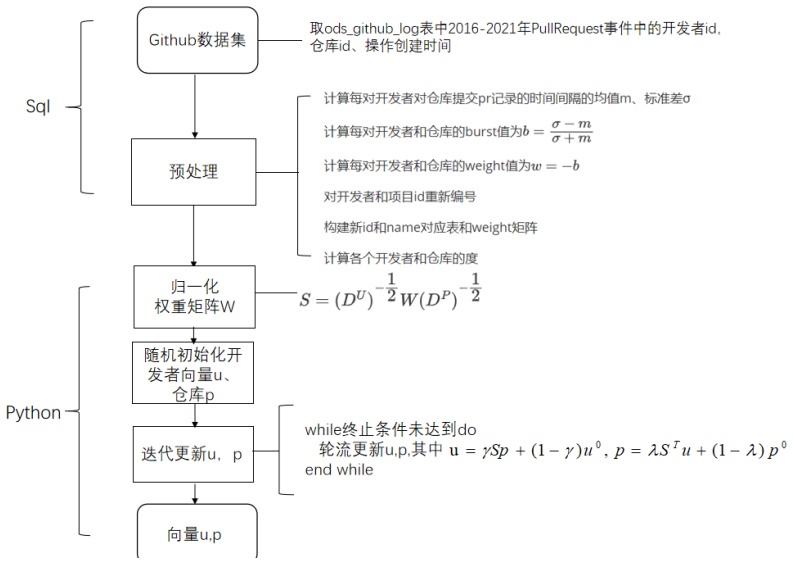


图 3.2

3.2.2 BurstBiRank 算法

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。

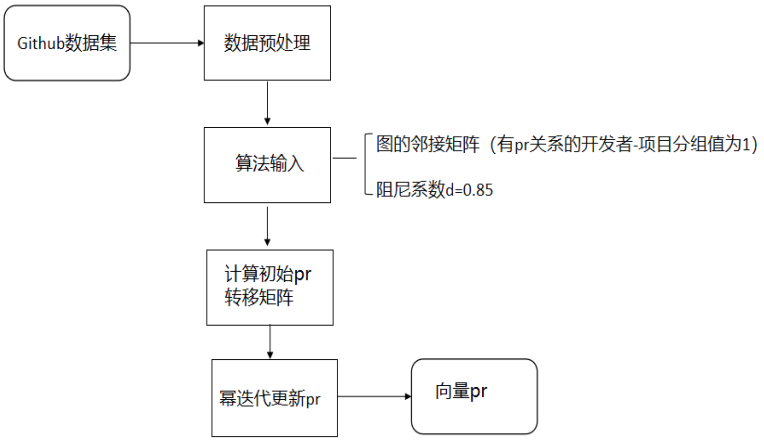


图 3.2

3.2.3 PageRank 算法

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。本项目将 PageRank 应用于开发者项目的二分网络，忽略节点的类型，将开发者和仓库看为同类节点。首先给每个节点赋予初始的 pr 值，第一轮的每个节点的 pr 值是均等的，值为节点个数的倒数。阻尼系数设置为 d=0.85，之后通过幂迭代运算$\mathbf{pr} = \mathbf{d} * (\mathbf{pr} * \mathbf{M}) + (1 - \mathbf{d}) * \mathbf{node_inverse}$，该公式中 M 为转移矩阵，node_inverse 为长度为节点个数且每个值都为节点个数倒数的向量，最终收敛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。

4.1 算法与算法的相关性评估

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：

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

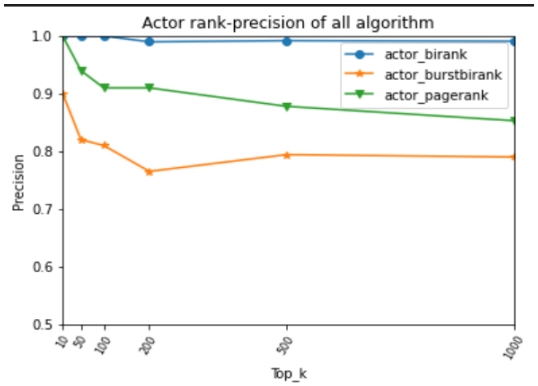
$$U_2 = (U_{burstbi} \cap U_{bi}) \cup (U_{page} \cap U_{bi}) \cup (U_{burstbi} \cap U_{page})$$

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：

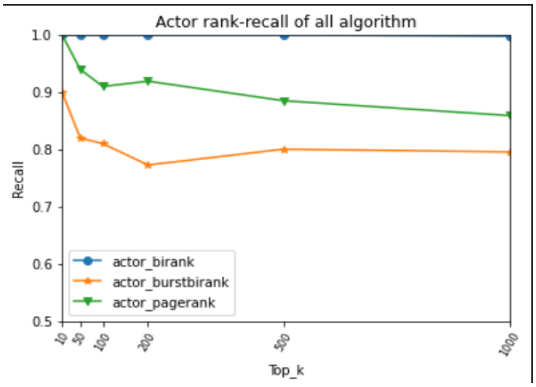
$$Precision_{bi} = \frac{|A_2 \bigcap A_{bi}|}{A_{bi}}$$

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

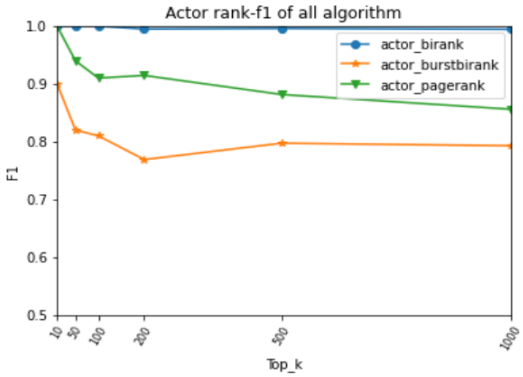
$$F1_{bi} = \frac{2 * Precision_{bi} * Recall_{bi}}{Precision_{bi} + Recall_{bi}}$$



（a）

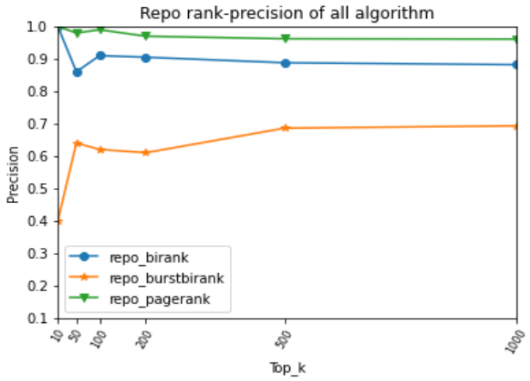


（b）

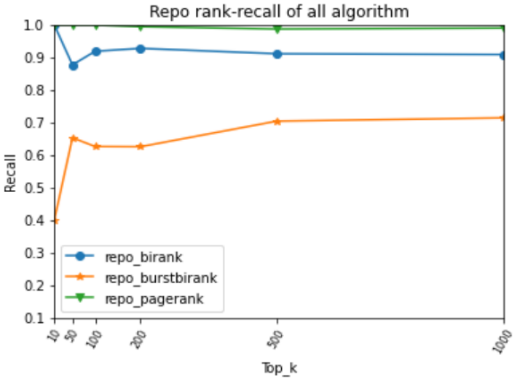

Actor rank-f1 of all algorithm
actor_birank
actor_burstbirank
actor_pagerank
F1
Top_k

（c）

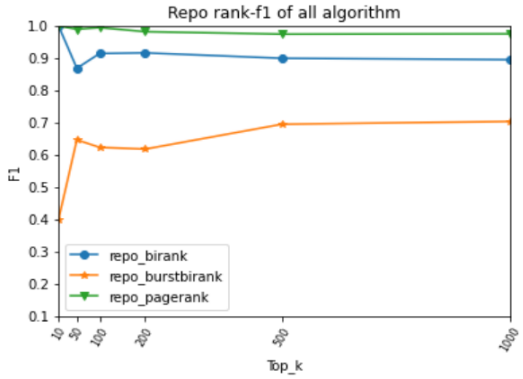
图 4.1


Repo rank-precision of all algorithm
repo_birank
repo_burstbirank
repo_pagerank
Precision
Top_k

（a）


Repo rank-recall of all algorithm
repo_birank
repo_burstbirank
repo_pagerank
Recall
Top_k

(b)


Repo rank-f1 of all algorithm
repo_birank
repo_burstbirank
repo_pagerank
F1
Top_k

（c）

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。

4.2 算法与指标的相关性评估

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	shiftleft-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$\rho = \frac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{(x_i-\bar{x})^2(y_i-\bar{y})^2}}$，得到结果如下表 4.2 及 4.3 所示：

	BiRank	BurstBiRank	PageRank
Top10	0.066	-0.115	0.199
Tpo20	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。

4.3 三个算法共同选择的 Top10 仓库与开发者特征分析

我们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，很少有多个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	pull requests
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

new_repo_id	repo_name	BiRank	BurstBiRank	PageRank	watchrank	star	fork	contributors	pull requests
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	Burst BiRank	Page Rank	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
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

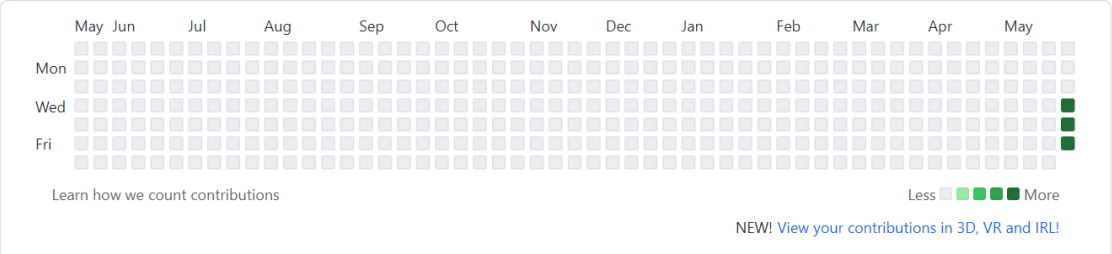


图 4.3 开发者 turbocanary-admin 的活动情况

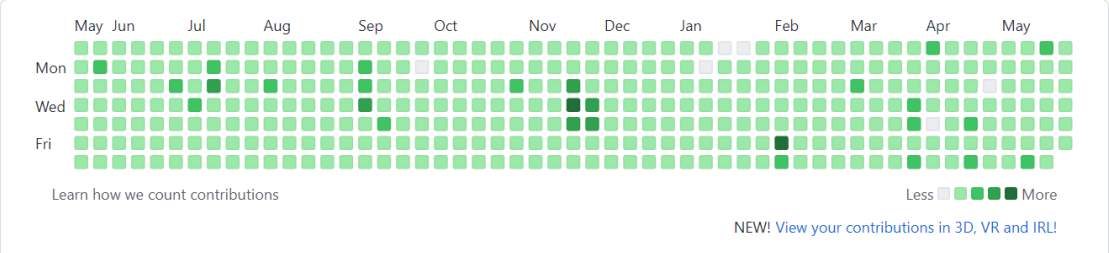


图 4.4 典型机器人开发者的活动情况

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 unifified 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 in“ uential 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. 评价：Spearman 相关性分析

何雨晴：

1. 算法：使用 python 进行排名算法实现

2. 评价：交叉验证三个算法的结果

附录 2.进度

3.20 了解了 openinsight 如何查看表格，阅读了 github 数据集.pdf 文档

3.21 准备开题，向助教了解了开题需要准备的方面

3.22 查阅相关工作

3.23 认真阅读了几篇论文，大致确定了一下路线和方法

3.25 开题答辩，吸取了一些教训，发现对于方法的选取的介绍过于欠缺，重点不明，且花费了过量的精力来做 PPT，而不是更多地关注本身内容

4.13 继续确定题目为影响力分析，阅读 BurstBiRank 论文

4.14 徐涵钰整理 BurstBiRank 论文内容，何雨晴阅读相关论文整理相关工作；

何雨晴查看所有临时表中数据，徐涵钰使用 pandas.DataFrame 尝试预处理了前 1000 条数据，何雨晴首次尝试将所有数据按照关联关系分组

4.15 汇报了相关工作+BurstBiRank 算法的思路和小规模预处理的实现

4.21 确定影响力分析可以被认为是做一个排名，何雨晴提出应当使用 PR 数据构建网络

4.22 开始尝试用临时表解决数据量过大难以处理的问题

4.27 尝试使用 ODPS SQL 获取数据，仍想使用本地 csv 存储数据

徐涵钰完成了使用 sql 进行 BurstBiRank 部分预处理的实现

为减少数据量，讨论了把只有一个开发者的仓库去掉会不会有影响

想要把有关联关系的仓库和开发者进行分组

4.28 讨论确定不要把数据 to_pandas()，应当保持在 odps 中继续预处理

持续讨论分组问题。该分组想要实现的是将有相同的项目或开发者分组，与常规的分组不一样。仍未找到方法。

徐涵钰阅读 odps 相关文档，了解了遍历等操作

继续尝试实现分组，徐涵钰尝试通过构建字典的方式，键值对中值为 dataframe 类型，每次遍历表都新建一个 dataframe 存放新读到的这条数据，如果与之前的某个 dataframe 有交集就合并，如果与之前多个 dataframe 有交集就把多个 dataframe 合并，删除原本的键值对，放入新的键值对中。该方法导致对 odps 的 table 访问频繁，连接被自动关闭，无法运行完成

4.29 尝试加入 options.read_timeout = 3600000，仍无法解决连接被迫关闭的问题

何雨晴开始阅读 BiRank 论文

5.1 先前的 sql 突然报错，无法正常取数据。

5.2 在助教指导下，徐涵钰发现为先前对临时表理解有误，应为首先建立空的表格，随后向其中插入数据，再使用 odps 的 dataframe 从中查看数据，或者用 reader 方法从中取数据后放入某个 table 后循环进行数据量统计工作。

何雨晴尝试在对 table 进行 for record in reader 以实现计算某条数据与上一条数句的时间差。

徐涵钰想通过计算所有数据与 MIN(created_time)的时间差来变相实现两条数据间时间差的计算，公式推导后发现无法实现

双方各自尝试用 read_table 循环遍历方法和使用 odps 的 sql 实现两条数据间时间差的计算，何雨晴首先使用遍历方式实现计算，但运行速度较慢；何雨晴希望将所有仓库与开发者相同的数据行的 created_time 统一到一起形成一个列表，徐涵钰用 sql 方法中的 wm_concat()方法实现该功能

5.3 何雨晴在统一后的新表基础上继续用循环法实现两条数据间时间差的计算

徐涵钰找到方法 lead(created_at,1) over(PARTITION BY repo_id, actor_id ORDER BY created_at) AS next_created_at 能够分区后将后一行的 created_at 数据作为新的一列 next_created_at 加在某一行中，随后使用 datediff 函数实现了对两条数据间时间差的计算，且速度很快，于是抛弃何雨晴实现的遍历方法。

讨论如何处理 NAN 的数据，在分析原论文后，得出结论应当将其以 0 代替

5.5 何雨晴继续研究 BurstBiRank 算法，并且提出需要在 sql 里就对 ID 进行重新编号，否则矩阵过大，提出可以“设置第一行对应新 ID 为 0，如果下一行和前面的仓库重名，则保持不变，否则+1”。然而该思路属于遍历思路。徐涵钰想到可以将所有 distinct 的 repo_id 取到一张表中，然后新增一列，该列设置为行号，最后在把该表与原表连接实现对应。

5.6 徐涵钰发现 ODPS 的 sql 没有直接取行号的功能。阅读相关文档，发现 row_number() over()方法能够变相实现重新编号的功能。

徐涵钰继续写 sql，完成了新编号后的 ID 与 weight 值的一一对应

5.6 何雨晴实现权重归一化，并认真阅读了 BiRank 的论文，实现了 BurstBiRank 的算法，但遇到 dimension mismatch 问题。

5.7 何雨晴发现不使用 numpy 而使用 scipy 不会出现 dimension mismatch 问题

何雨晴发现代码中误将仓库与开发者矩阵位置颠倒了，改正后无报错问题

何雨晴发现算法迭代出现不收敛的问题；徐涵钰猜测原因可能为先前未对其进行分组，导致图不是强连通的。

何雨晴尝试继续以遍历方法解决将有相同的项目**或**开发者分组

5.8 徐涵钰尝试在 pandas.DataFrame 上继续使用先前字典方法实现分组的方法，但是速度非常慢

在助教指导下，向并查集思路靠拢

5.9 何雨晴实现并查集，完成了分组，速度很快

5.12 何雨晴通过重新研究 BurstBiRank 算法伪代码，发现算法不收敛与分组无关，而是因为后续构建的是稀疏矩阵，但是先前却设置 weight = - burst + 1，导致不存在关联的仓库和开发者之间的边上的权值应为 1 而不是 0。考虑到稀疏矩阵便于计算的特性，我们重新选取 weight = - burst，算法在 50 次迭代内收敛了

发现算法结果中机器人占了很多，于是徐涵钰在预处理中首先加入 actor_login NOT LIKE '%bot%'以去除显式为机器人的开发者

认为 BurstBirank 效果不好，开始考虑实现其他算法以提高效果

5.13 徐涵钰实现与 BiRank 配套的数据预处理，何雨晴实现 BiRank 算法构建

5.15 何雨晴主要阅读 BiRank 论文中的结果分析部分，决定实现更多算法进行横向比较

徐涵钰开始对 watch 相关数据与 follower 相关数据进行抽取和预处理

讨论了选取数据的时间范围

5.16 徐涵钰新建两个 ipynb 分别进行对 watch 的处理和对 follower 的处理，首先寻找 watch 数据的所在 table 以及 follower 数据的所在 table，开始时由于在对 type 筛选时误用了 data.head(10)而没有发现有 watch 数据而选择了 star 数据，随后发现该错误并重新使用 watch 数据

讨论了对于 watch 数据的时间选取，讨论了在只查看某一年排名的时候使用 pt<=该年度的最后一天还是使用 pt=该年度。

徐涵钰完成了 watch 相关的 ipynb。

徐涵钰开始阅读 spearman 相关性分析的实现方法，何雨晴开始阅读 SIR 模拟的实现方法。

讨论选取的数据范围为 16 年~20 年

何雨晴构建完成 birank,burstbirank,pagerank 三个算法

徐涵钰在做 follower 相关数据抽取时遇到难以比较两个 datetime 的问题，最终通过使用 TO_CHAR 转变为字符串后与字符串比较的方法解决

5.17 徐涵钰发现 follower 相关数据的爬取时间是从 2021 年 11 月开始的，与原本确定的数据范围截止到的 20 年之间相隔时间过远，follower 数据可能无法很好地反映真实情况，最终选择将数据范围扩大到 16 年~21 年，并选取截止到 21 年年底的 follower 数据。

徐涵钰发现 follower 相关数据没有直接统计好某个开发者的粉丝数的，只有一个字段以字符串格式存放了一个列表，其中每个粉丝的名称以逗号分隔。徐涵钰想到，通过对逗号计数+1，即 CHAR_MATCHCOUNT(followers, ',') + 1 的方法来变相统计粉丝数量，并使用 github 直接搜索了部分用户查看了粉丝数，确定了统计的正确性。

讨论了 pagerank 算法的合理性，何雨晴解释了技术细节

徐涵钰发现，直接由 watch 得到的排名 TopK 与由算法得到的排名 TopK 结果很不一样

徐涵钰发现 pagerank 算法结果有误，何雨晴重新研究后发现先前只对矩阵的右上角进行了赋值而忽视了左下角，重写后结果正确

徐涵钰在跑大批量代码时发现先前隐藏的一些小问题，重新用 distinct 修饰了 repo_id 并且为防止乱序加入了 order by

何雨晴发现徐涵钰的 watch 相关数据中对 ID 的重新编号有误，有相同 ID 由于被筛选出 2 次而被赋予了两个新的编号，徐涵钰重新修改，在代码中加入先取出所有 repo_id 再进行编号，而不是从 repo_id 和 actor_id 的对应表中直接取 repo_id 编号。

徐涵钰发现直接取 ID 与名称的对应关系会有同一 ID 对应多个名称的情况，考虑原因为用户改名/仓库改名，发现一开始就需要将时间因素纳入考虑范围，以便取时间最近的名称来实现与 ID 的唯一对应。

徐涵钰发现最终结果出错，排查完原因，发现是 jupyter notebook 出问题，显示某个部分已经运行，但事实上还是保存的先前未修改时的结果，重跑后问题解决。

徐涵钰继续将 watch 整合到先前实现算法的 ipynb 中，并进行优化，将部分取数据、中间表命名、表与表的连接进行重新对应和修改，并且这些修改能够减少运行时间

由于同一开发者在不同时间有不同的 follower 情况，徐涵钰询问助教，得知了 follower 相关数据中各个时间相关字段的含义，并最终选择 lastupdatedat 即最近更新的时间来判断选择哪条记录作为某一开发者的 follower 情况。

讨论了结果评价，认为可以做算法 VS 指标和算法 VS 算法两种类型的评价。徐涵钰提出可以查看算法中排名前 K 的仓库在 watch 里的排名，以及查看排名前 K 的用户在 follower 中的排名

5.18 徐涵钰继续对 sql 进行优化，在 for record in reader 中以取最大的新编号的 ID 代替 cnt++来查看数据量，进一步减少了运行时间。

何雨晴在写 PPT。

徐涵钰运行整个代码，发现直接从 ods_github_log 中取数据以及 to_pandas()操作都很耗时间，将原本好几次从 ods_github_log 中取不同数据的操作合并为一开始从 ods_github_log 中取所有需要的数据形成中间表，后续再从该中间表中取需要的数据，以减少运行时间，并完成了表与表连接时的重新对应。

徐涵钰实现了查看 birank 中排名靠前的仓库在 watch 中的排名情况，由于 odps 的 sql 似乎没有 x in [列表]的判断方法，导致无法直接将 birank 中排名靠前的 ID 组成的列表写在条件中，只能将每个 ID 分别作为条件的一部分传入。

5.19 徐涵钰继续优化 sql，何雨晴学习 SIR 模拟

徐涵钰发现无法在 follower 排名中找到算法结果 TOP10 的排名，发现原因为：（1）先计算了每个 actor_name 对应的 follower 数，然后采用了 actor_name 与 ID 的唯一对应表，导致有部分 actor_name 无法与 ID 对应，造成了数据缺失。修改 sql 为首先将 actor_name 都对应到 ID 上，再计算 ID 与 follower 数的对应关系表，最后选取时间最近者作为最终的唯一对应结果。（2）follower 数据不全，follower 相关数据的获取是从 2021.11 月开始的，每天只能获取 5000 条数据，而我们原先是取到 2021.12 月为止，很多数据都没有获取到。于是最终选择将所有爬取到的 follower 数据都纳入使用范围，数据面变全了很多，但是仍然有数据缺失的情况存在。

徐涵钰完成了 Spearman 相关性分析。

何雨晴抛弃 SIR 模拟，决定通过对三种算法做交叉验证实现横向比较。徐涵钰将所需数据（即算法结果中 Top1000 的项目/开发者的新编号）取出并保存到 csv 文件中，并交给何雨晴。何雨晴完成了该评价方法的代码书写。

徐涵钰将 BurstBiRank 的 Top1000 的新编号取出。随后发现由于 BurstBiRank 一开始抽取的数据与 BiRank 和 PageRank 要求的不一样，导致新编号事实上与另两种算法得到的新编号不是对应到同一个仓库/开发者上的，需要重新将 BurstBiRank 的新编号再次重新编号才行。但是这一步骤不能在 BurstBiRank 算法的矩阵实现之前进行，因为算法仍然要求新编号是连续并无缺的。

5.20 徐涵钰使用 sql 解决了前一日 BurstBiRank 编号不一致的问题，实现了三个算法新编号的一致，以便比较。

何雨晴使用存放了三个算法的 Top1000 的 ID 的 csv，完成了横向比较

二人一起将 PPT 和演讲稿进行修改

5.23 开始一起书写报告

5.26 为结果评价中能够将三个算法共同选出的开发者与仓库进行分析而将部分数据存入 csv，何雨晴使用 python 做了表的合并

徐涵钰在 github 上分别搜索了前十个共同选出的开发者与仓库，得到了它们的一些特征

5.27 结项答辩