基于加权 PageRank 的 Github 项目活跃度分析

孙印政 陈丘轲

1、引言

在 GitHub 平台上，用户的所有行为都是公开透明且可追溯的，对于某一开发者，可以通过分析开发者的行为数据（主要为 Commit 记录）来评估其在项目中的活跃情况，进而确定开发者的活跃程度。

对于某一项目的活跃程度，一种基本思路为通过开发者行为数据的数量统计以及加权和的方式来直接计算项目的活跃度。但其实这种方式存在一个明显的问题，如果有人反复去刷一些行为事件，则可以人为的提升项目的活跃度，那么基于此方法预测项目活跃度的真实情况不一定准确。本次研究基于给定的 GitHub 数据集，拟采用合适的方式，确定 GitHub 区域中某一项目的活跃程度。

2、问题描述

基于 GitHub 数据，通过构造协作网络，实现对于 GitHub 社区内的项目活跃度进行分析。对于一段期间内的 GitHub 项目活跃度进行排名，并最终以一定可视化的形式展示和分析。

3、方法

本次实验中拟采用如下思路解决问题：由于 Github 是一个社交平台，行为数据不仅可用于进行一般性的统计分析，还可以用于构建协作网络，通过网络再进行进一步分析，确定项目的活跃程度。

本次实验中，先通过全域行为事件数据构造一个全域协作网络，再通过一些经典的社交网络算法来进行分析。采用开发者的行为数据构建网络，根据构建网络结果确定项目之间的关联度，再通过 PageRank 等算法计算项目的活跃度。

针对海量的 github log 日志数据，我们采用 Spark 分布式并行计算框架来计算项目与项目之间的协作关联度。

具体流程如下：

首先，GitHub 主要是以 Issue + PR 的方式进行沟通与协作的。对于开发者的数据，根据 Issue 和 PR 的信息，计算某开发者在某项目上一段具体时间内的活跃度。活跃度计算公式$A_d = \sum w_i c_i$。对于开发者的的 GitHub 行为数据计算其对项目的贡献度，由于关系类型包含多种，需要对其降维，对于某一开发者对于某一项目的所有操作进行一个综合的评价，实际操作中选取了 Issue 评论，Issue，PR 操作，PR Review 四类事件并分别赋分 1，2，3，4。对于每一条 log 日志，根据事件类型类型输出由 contributor_id，repo_id，score 构成的三元组，然后在 contributor_id，repo_id 上做聚合 Group 操作，累加得到开发者对某一项目的总贡献度，结果保存在数据库中间表中。该步骤流程如图 1 所示。

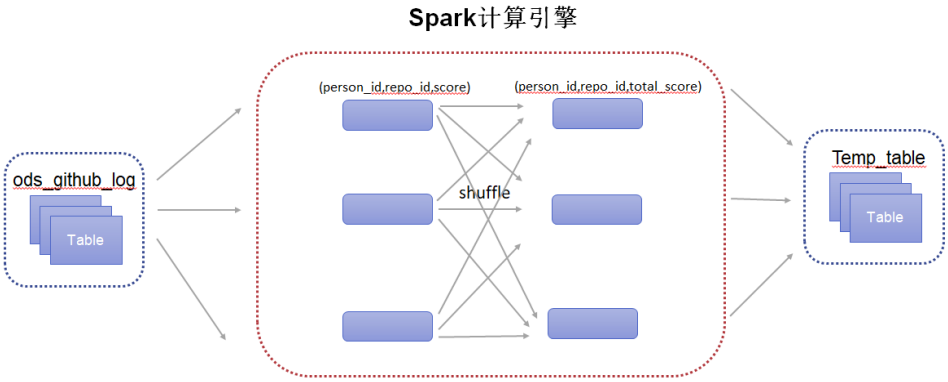

图 1 计算开发者对项目的贡献度

其次，在本次实验中假设，若两个项目上有相同的活跃开发者，则这个开发者在某种程度上就关联了这两个项目。使用开发者在两个项目上的活跃度的调和平均作为该开发者对两个项目关联度的贡献，两个项目之间的关联度即两个项目上所有开发者对两个项目关联度的贡献之和，即：$R_{ab} = \sum_{dev} \frac{A_{da}A_{db}}{A_{da}+A_{db}}$。这一步实际上是对于节点类型进一步降维，去除掉所有开发者节点，通过同一开发者在两个项目的影响，确定项目之间的协作关联度。同样，计算任务由 Spark 框架完成，上文提到的元组根据 contributor_id 进行聚合，每两个元祖通过上述公式计算得到项目间的协作关联度 Collaborative relevance，并输出(repo_id1，repo_id2，collaborative_relavance)，结果保存在 SYZ_CQK_Result 表中，同时该条边的信息也会被记录在图数据库 Neo4j 中。Neo4j 用于存储项目节点和边的信息，同时实现了协作网络的可视化。该步骤流程如图 2 所示。

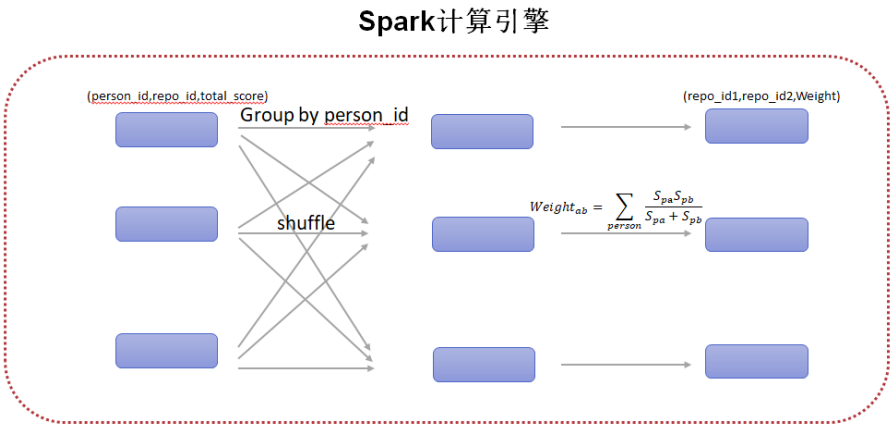

图 2 计算项目间协作关联度

最后，对于项目网络，采用了加权 PageRank 算法（Weighted PageRank，即 WPR）进行计算。PageRank 算法的每次迭代，将每个网页的排序值平均分给了它指向的网页。但在实际中，同一网页指向的不同链接的重要性可能不同，分到的排序值也应不同。本次研究中引入了加权 PageRank 算法 （WPR），这是对标准 PageRank 算法的扩展，WPR 还考虑了两个项目间的重要性。一个网页越受欢迎，连向它的网页和它连出的网页越多。所以 WPR 根据 v 指向网页集合中点的出度和入度，分配 v 的排序值。这样 v 排序值会更多地分配到更受欢迎的网页上，而不是均分。

4、评价

处理数据分析

本次研究针对于 GitHub 数据集进行分析，选取 2020 年第一周，即 1 月 6 日到 1 月 12 日一周中的 ods_github_log 信息，统计 Issue+PR 数量，选取数量大于等于 5 的项目作为活跃项目，并根据这些项目构建网络，选取活跃仓库数量 67 万个。

可视化结果

本实验由于数据量庞大，在可视化时仅选择 10000 条数据进行可视化展示，结果如图 3 所示。为了将图谱进行精简，实验中又设置 pagerank 的阈值，筛选

大于阈值的重要核心节点，减掉小于阈值的节点，结果如图 4 所示。两图中圆圈大小代表了项目的重要程度，两个项目间线长粗细代表了项目间关联度高低。

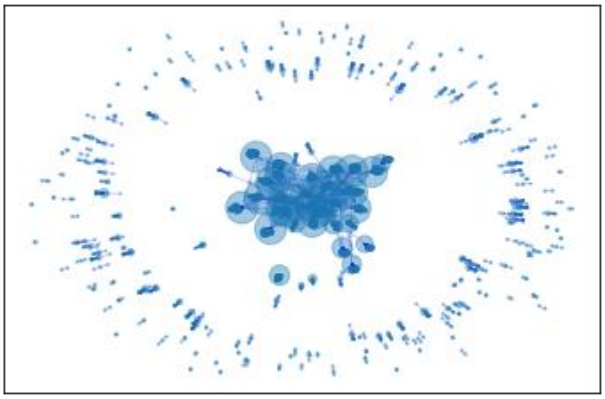

图 3 WPR 计算结果

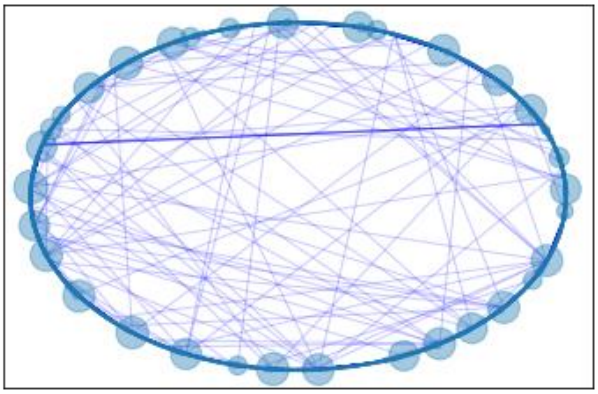

图 4 阈值筛选 WPR 计算结果

将计算结果存入图数据库 neo4j，并将结果中的 10000 条数据进行可视化展示，结果如图 5 所示。

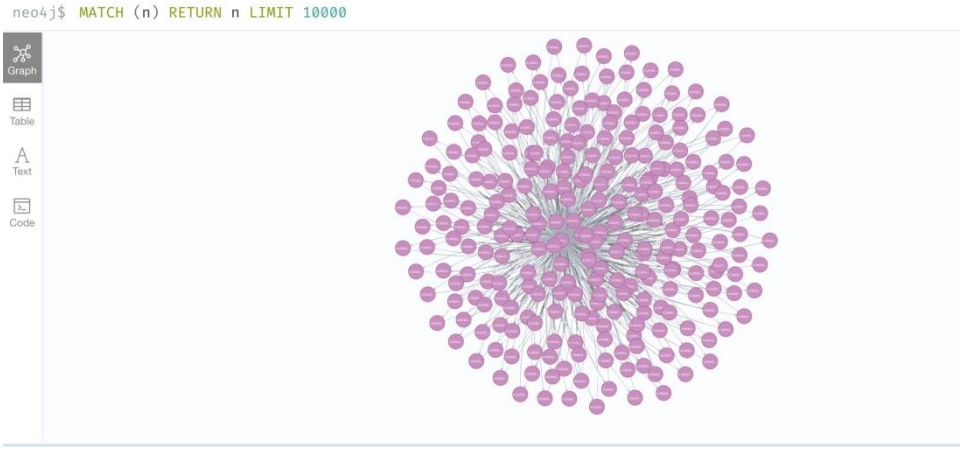

图 5 neo4j 存储数据可视化展示

高活跃度项目分析

使用 WPR 算法计算，获得 2020 年第 1 周中份活跃度最高的 10 个项目，将该结果与 X-lab 开放实验室发布的《GitHub 2020 数字洞察报告》中构建的 Github 全域项目协作关系网络 OpenGalaxy 中 2020 年项目活跃度 Top10 进行对比，结果如表 1 所示。

排序	实验计算结果	OpenGalaxy 结果
1	microsoft/vscode	microsoft/vscode
2	flutter/flutter	flutter/flutter
3	kubernetes/kubernetes	tensorflow/tensorflow
4	DeflitelyTyped/DefinitelyTyped	microsoft/TypeScript
5	microsoft/TypeScript	DeflitelyTyped/DefinitelyTyped
6	facebook/react-native	microsoft/WSL
7	gatsbyjs/gatsby	golang/go
8	helm/charts	gatsbyjs/gatsby
9	rust-lang/rust	kubernetes/kubernetes
10	tensorflow/tensorflow	vercel/next.js

表 1 实验计算结果与 OpenGalaxy 数据对比

VSCode 的影响力在《GitHub 2020 数字洞察报告》中基于开发者活跃度定义确定的项目活跃度的方式中排名 11，而在协作网络的计算结果中排名第一，且远高于第二名，成为活跃度最高的项目。经分析，这主要是因为 VSCode 更容易与其他各领域的项目产生了大量的协作关联，有很多顶级的开发者都不仅仅会单独对 VSCode 有贡献，一般同时也会对其他项目有较高贡献，从而在社交网络中两者之间会产生较大的关联度，进而通过社交网络算法计算的结果中 Vscode 权重占比较高。

关于机器人账号的过滤

由于大量项目中存在协作机器人，可用于对项目进行协作管理，而机器人可以服务多个的项目，从而导致在大量的项目之间建立起关联。尤其是高活跃度的项目往往机器人的活跃度也很高，会导致这些项目之间高度相关，对最终计算结果显示不利。在本次实验中，为了对机器人账号进行过滤，在构建二分图过程中，根据开发者活跃的项目数量，剔除了活跃数量大于 10 的开发者账号，一定程度上减少了高活跃度机器人的存在。

5、相关工作

在已有工作中，对于 GitHub 区域活跃度计算，普遍的做法是采用了基于数值统计的方式实现的，即根据开发者的行为数据的数量统计以加权和的方式来获得项目的活跃度，目前该方式也受到普遍认可。

本实验基于博客 https://blog.frankzhao.cn/github_activity_with_wpr 的思路，尝试通过构建社交网络实现确定项目之间活跃度的分析。

6、结论

对于 Github 项目活跃度的评价，本次实验给出了另一种方式。由于 Github 是一个社交平台，行为数据不仅可用于进行一般性的统计分析，还可以用于构建社交网络，通过网络再进行进一步分析，确定项目的活跃度程度，这种分析方法相较传统方式的算法稳定性更好。本次实验中，通过构建了项目之间的协作关系网络，并采用加权 PageRank 算法计算，获得网络中项目的活跃度值，并以一定可视化形式实现展示。

本次研究中由于数据量过大的原因，只处理了一周的数据。而数据的计算结果 Top10 与 OpenGalaxy 给出的项目活跃度 Top10 较一致，一定程度上说明了一周的时间段内 Github 数据对于整个网络中心度的判断较准确。

结合研究结果，我们认为协作网络的评价方式更适用于对一些联系紧密，关联度强或者对某一特定领域或区域内的 GIthub 项目进行活跃度与中心度的分析。这种分析方法更关注于整个区域的中心位置，对于一个 Github 社区中的核心项目判断会比较准确，但不适用于对于区域中每一个项目给出明确的活跃度指标。

参考文献

[1]王锦坤,姜元春,孙见山,孙春华.考虑用户活跃度和项目流行度的基于项目最近邻的协同过滤算法[J].计算机科学,2016,43(12):158-162.

[2]杨欣捷,田蜜,江一鸣.开源项目活跃度模型构建及实证[J].信息技术与网络安全,2021,40(07):27-33+51.DOI:10.19358/j.issn.2096-5133.2021.07.005.

[3]叶培根,毛建华,刘学锋.基于大数据的 GitHub 开源社区开源项目量化分析[J].电子测量技术,2017,40(08):84-89.DOI:10.19651/j.cnki.emt.2017.08.019.

[4] Frank Zhao.加权 PageRank 下的 GitHub 全域项目活跃度分析[EB/OL]. https://blog.frankzhao.cn/github_activity_with_wpr, 2020‐10‐07.

[5]任晓龙,吕琳媛.网络重要节点排序方法综述[J].科学通报,2014,59(13):1175-1197.

[6]王德广,周志刚,梁旭.PageRank 算法的分析及其改进[J].计算机工程,2010,36(22):291-293.

附录

分工

陈丘轲：算法设计与实现，数据可视化，结果分析与报告撰写

孙印政：模型构建，数据处理，协作网络构建

进度

3-4 周：确定选题，查找相关资料

5-6 周：确定研究方法，确定模型

7-8 周：初步尝试获取数据，进行数据处理

9-10 周：构建协作网络，算法设计

11-12 周：算法实现，数据批量运行，数据可视化

13-14 周：数据结果分析，汇报准备，报告撰写

项目链接

https://github.com/YinZheng-Sun/SocialComputing