四川大學

本科生毕业设计（学术论文）

题　　目 开源背景下开发者的贡献评估和特征分析

学　　院 软件学院

专　　业 软件工程

学生姓名 韦子健

学　　号 2016141463089 年级 2016

指导教师 杨秋辉

教务处制表

二〇二〇年四月三十日

开源背景下开发者的贡献评估和特征分析

软件工程

学生 韦子健 指导老师 杨秋辉

［摘要］ 开源社区发展迅速，越来越多的公司、独立开发者参与到开源的项目中，极大地促进软件产业的进步与发展。开发人员的贡献评估和行为特征的研究是软件演化领域的重要研究内容，也是经验软件工程的研究热点。通过对开发人员的贡献评估和特征分析，可以发现一系列具有价值的项目演化规律，进而帮助项目管理人员合理配置开发人员，提高软件的开发效率。

本文基于 Github 开源社区，对其中的开发人员进行贡献评估和特征分析。包括数据爬取、预处理，并将数据进行分析，以可视化方式展示分析结果。

在已有的相关研究中，对于工程领域的代码库分析相对较少，对开发者的各类特征的深入挖掘和解释说明也相对较少。本文重点引入一种贡献度计算方式，同时定义多类指标，使用相关分析、回归分析、主成分提取等技术研究各个指标之间的数学关系。之后对贡献度较为高的开发人员以时间、合作、地域、域等特征进行可视化分析，得到一系列结论。

通过对 Github 网站上热门的 17 个开源项目研究，开源社区下贡献者的代码提交次数和活跃时间两个指标可以更好地描述一个开发者的贡献程度。对于特征分析方面，发现了一些有价值的规律。如开源项目往往由少数精英主导；大部分项目在中期的提交行为活跃，初期缓慢增长，末期提交行为趋于稳定；开发过程中的各开发者更倾向于单独完成自己的开发任务；在贡献较多的开发者中，存在不同形式的贡献规律，贡献偏向有不同的类型；commits 较为活跃的地区是北美（尤其美国）、欧洲、东亚，北亚、中亚、南美表现则较为一般，非洲地区 commits 较为贫乏；由公司主导开发的软件项目，公司本身的域占据了开发人员的大多数。

［主题词］ 开源社区；特征分析；贡献度；可视化；数据挖掘

Developer contribution assessment and feature analysis in the context of open source

Software Engineering

Student： Zijian Wei Adviser: Qiuhui Yang

[Abstract] The open source community is developing rapidly. More and more companies and independent developers are participating in open source projects, which greatly promotes the progress and development of the software industry. Developers 'contribution assessment and behavioral characteristics research are important research contents in the field of software evolution, and are also research hotspots in empirical software engineering. Through the evaluation of the developer's contribution and feature analysis, you can find a series of valuable project evolution laws, and then help the project management staff to rationally allocate developers and improve the efficiency of software development.

This article is based on the Github open source community, and evaluates and analyzes the contributions of developers. Including data crawling, preprocessing, and analysis of the data, using visual technology for analysis.

In the existing related research, there is relatively little analysis of the code base in the field of engineering. This article focuses on introducing a method of calculating the contribution, defining multiple types of indicators, and using correlation analysis, regression analysis, principal component extraction and other techniques to study the mathematical relationship between the various indicators. Afterwards, the developers with a high contribution rate were visually analyzed with time, cooperation, region, region and other characteristics, and a series of conclusions were obtained.

Experimental results show that through research on 17 popular open source projects on the Github website, the two indicators of code submission times and active time of contributors in the open source community can better describe the contribution of a developer. For feature analysis, some valuable laws were found.

[Key Words] open source comunity；feature analysis；contribution；visualization；data mining

目 录

1 绪论

1.1 研究背景

随着开源软件社区迅速发展，社区中与软件开发相关的活动愈加频繁，开发人员利用开源社区平台相关功能进行协作开发，如进行版本控制、交流代码问题、分享经验，因此开源社区积累了大量的开发数据资源（包括用户信息、代码信息、组织信息等）。近年来，软件工程领域研究者已经认识到了开源社区中历史数据存在的潜在价值，通过对这些历史数据进行深入挖掘和研究分析，可以得到一些与软件开发相关的一般规律，达到解决软件工程领域相关问题的目的[1][5]。Github 是一个带有社交和协作开发功能的代码仓库。如今，Github 已经成为最火热的代码仓库平台，来自世界各地的开发者可以无视地域限制、语言限制、时间限制，通过 Github 在线平台，进行软件的合作开发。由此，近几年涌现出了许多高质量的软件和架构，为全世界开发者进行代码复用提供了极大便利，避免了重复造轮子的烦恼。同时，开源行为极大地提高了工作和开发效率，从而加快软件的演化速度，软件演化的节奏由慢到快，敏捷开发有了更有力的支持[3][6]。越来越多的代码项目（包括各大公司的开源项目、个人开发者的项目）纷纷保存在 Github，这些项目涉及的编程语言和领域也十分丰富，几乎覆盖了整个计算机领域。通过 Github 所提供的相关服务，世界各地的开发者为自己参与项目提供自己力所能及的贡献，如新增功能、修复代码缺陷、提供或完善测试等等[7][8]。

在开源环境下，开发人员的行为特征和贡献评估成为了软件工程的研究热点。本次课题研究意义在于通过对项目开发人员的贡献进行评估，各类行为特征的进行挖掘，掌握项目的角色配置结构，从而提高软件开发效率和减少开发成本，对未来项目发展走向具有指导意义[4]。

1.2 国内外研究现状

对于开发人员贡献度量评估和行为特征分析，Gousions 等人以代码相关和非代码相关两个维度出发，从代码仓库、邮件列表和论坛、缺陷数据库、Wiki 等数据源中捕获了近 30 个度量指标[2]，并将这些度量指标以对项目本身是否产生积极影响进行二分类，进而提出了一套完整的贡献度量体系以及开发者贡献度计算方法。但他们的实验存在很多局限性和缺陷，如 Gousions 只是总结出了诸多指标，但指标的可用性、显著性没有得到严格保证，需要进行进一步地计算验证。同时，对于部分指标存在难以获取的情况，Gousions 等人也仅仅才用了和代码相关、且易于获取的数据进行了实验[9][12]。

对于评估开发者在代码仓库中的贡献，Jalerson Lima 以经验启发式提出了对开发者贡献评估的多个指标[3]，并通过对话交流的方式，搜集了 7 个软件开发团队的相关信息，从而得出一些结论。指标包括四大方面，即代码贡献（Code Contribution）、方法评价复杂度（Average Complexityper Method）、开发者引入缺陷的情况（Introduced Bugs）、缺陷修复贡献（Bug Fixing Contribution），软件项目管理者认为代码贡献和方法评价复杂度两个指标更适合用于评估一个开发者的代码贡献。

对于开发者的贡献特性研究，Minghui Zhou 等人提出了基于时间维度的开发者贡献特性研究方法。从开发者开发动力因素、个人能力因素、所在环境因素与动力因素相关联意愿因素四个方面进行扩展分析[4]，启发式地总结出了多个贡献度评估指标。Minghui Zhou 等人发现一些时间规律，如对于加入项目的第一个月内的长期贡献者（LTS）在开源社区中表现会更加活跃，并且个人能力特征受时间影响很大，随着时间迁移，个人能力往往可以得到提高。Minghui Zhou 等人比较创新的地方在于实验方法，即采用了机器学习的分类方法。他们将搜集到的开发者信息按照时间进行分类，以开发者刚加入项目时的相关评估指标和最终是否成为长期开发者（LTS）作为学习标签，利用逻辑回归训练了一个预测模型，该模型可以接受一个开发者的行为特征，输出一个新开发者是否会成为长期开发者的概率值。但是该模型准确率和召回率都比较低，需要就算法本身和数据集本身进行进一步的研究[14]。

在开源软件社区相关研究上，国内主要针对开源环境背景下开发者的贡献评估方法的研究和开发者行为特征方面的挖掘。

在开发者的贡献评估方法的研究上，甘谊昂[27]等人从Github、本地Git仓库两种方式进行数据集的搜集，得到了软件项目开发者的相关信息，并以主观方式定义了一些评估开发者贡献的度量指标，如代码行数、修复代码次数、提交代码次数等等。从代码相关（如代码贡献）、非代码相关两大类指标进行开发者贡献评估，非代码相关指标如开发者在社区发布issue、评论、代码review等行为。甘谊昂等人研究的内容主要是：首先对搜集到的数据集进行数据挖掘，选择出用于评估开发人员贡献行为的相关指标，相比之前的研究，它们通过统计分析的方式，保证了指标的可用性和显著性。结合数据可视化技术和数据挖掘技术，甘谊昂等人研究开发人员在时间维度上的贡献特证，如个体开发者的惰性时间、项目开发人员前后期贡献的关系等，进而提取科学的软件演化模式。根据演化模式的不同，对开发者按照角色进行分类，并对具有不同角色身份的开发者进行对比，总结各个时间周期的开发者的贡献特点。局限性上讲，其研究结果适用于大型开源项目，尤其适用拥有众多开发者的开源项目，但是不能保证在中小型项目上的结论，如果要一个通用规律，需要选择更多类型的开源项目然后进行分析。

在行为特征分析上，李存燕等人以 Github 开源社区的项目数据为研究对象，通过把代码项目克隆到本地[28]，即采用 git clone 命令，之后在命令行中执行 git log 命令，再加以相关 shell 脚本处理，获取了三个典型项目的相关数据(如开发人员 Commits 次数和修改代码行数等)；实验内容主

要为：以开发人员代码特征、文件特征和开发人员间的协同关系等方面进行研究，结合可视化技术，揭示了开发人员在 Github 上的部分行为特征。

1.3 论文主要工作

论文主要采用经验软件工程方法完成研究过程。

首先是对数据集的获取。对于开发人员的代码贡献信息，没有公开的数据集，需要自己进行搜集和相关处理[15]。开源项目来源主要是 Github 上托管的热门项目，数据获取采取两种方式，一是通过编写爬虫程序（Github REST Api V3）[16]，另一个是命令行运行 git log 命令，并结合 shell 文本处理技术[17]，最终将提取的数据进行 json 结构化保存。

有了 json 结构化的数据集后，会通过以下方面进行研究。首先是针对数据集中各种贡献指标自身的研究。分析方法主要为主成分分析、相关性分析、回归分析三部分，保证贡献指标的可用性和显著性。

在研究完变量之后，会继续在此基础上研究开发人员的具体特征。如合作关系、地域特征、邮域特征、贡献时间特征，结合可视化技术进行结果展示和分析，从而得出一些结论。

在研究过程中，对于数据集的获取，构建相关的可执行工具，尽可能做到自动化分析开源项目的相关信息。同时对相关的算法进行代码封装，便于后续的使用。

1.4 论文组织与结构

全文一共有五章组成，每一章的组成如下：

第一章绪论。这一部分主要介绍了开源贡献人员特征分析的研究背景、主要研究方向和工作以及国内外的研究现状。

第二章相关技术。解释了本课题中采用的研究方法（实证研究方法）和数据挖掘技术的理论基础。它主要包括四个方面：经验软件工程，相关性分析，主成分分析和回归分析。

第三章系统架构及研究方法。这一部分说明全文的研究框架以及具体的研究方法。主要包括数据的采集抽取、数据的预处理、贡献度的计算方式、观测指标的定义以及研究方法、贡献者的特征分析方法。

第四章实验结果及分析。这部分结合可视化技术来显示实验结果，进而分析，并给出相关结论。

第五章总结与展望。本部分对全文的工作和研究进行了总结，并讨论了该实验的缺点和今后的工作前景。

2 相关技术

2.1 经验软件工程

2.1.1 定义

经验研究方法（Empirical research method，也称为实证研究方法）在很多研究型学科中得到广泛应用，如物理学、心理学、医学以及各类社科领域。软件工程不仅涉及软件的技术解决方案，同时也与组织、管理、人类行为等多种问题有关。1986 年，Basili 等人首次将实证研究方法引入软件工程领域，是软件工程领域实证研究的鼻祖[33]，之后，经验软件工程（Empirical software engineering）成为了很多软件工程问题的经典研究方法。2005 年，Kitchenham 等人提出了基于证据的软件工程，并给出了相关研究方法，如系统文献综述法[34]。近年来，几乎所有期刊中都有涉及经验软件工程的论文。2016 年，在美国举办的 ICSE 上，程序委员会宣布，在接受的论文中，共有 32 篇关于经验研究的论文，可见，实证研究成为了软件工程研究问题的重要手段。

2.1.2 研究方法

软件工程中的经验研究方法主要包括实验方法、案例研究、调查研究、仿真实验和试点研究等方法。其中，实验方法主要用于软件质量，软件维护，软件测试，软件构建，软件工程模型和方法等研究方向。在这些研究方向中，可变因素通常采用易于控制和易于访问的历史数据（例如代码库，代码报告等），以保证实验的可用性和显著性。例如，Raja U 等人使用文本挖掘技术，对开源软件的缺陷报告进行挖掘分析，研究并统计了开源项目在软件生命周期中重复提交代码 bug 的数量，得到相关结论[35]。考虑本次论文课题的研究问题，即开源贡献者的贡献评估和特征分析，采用实验的方法进行研究，并得出相关结论。

对于实验方法的经验研究，通常包括数据来源、数据收集、数据分析（包括使用分析工具）三个步骤。

在数据来源上，开源软件成为软件工程领域各个研究方向常用的数据来源，本文亦采用 Github 为数据来源。

在数据采集上，相关研究中采用问卷、归档数据、访谈采集数据、爬虫等手段。对于前三类数据，特点是易于获取、易于结构化，但成本可能较高。考虑时间成本，本次研究采取爬虫手段来获取实验数据。

在数据分析上，实验研究一般采用统计图表和数理统计两种方式。统计图表通常作为分析的第一步，将收集的数据可视化；数理统计方法采用相关分析、回归分析、参数检验等统计方法对实验数据进行进一步的挖掘，以便于得出结论。本次课题采用统计图表和数理统计相结合的方法进行实验。

2.2 相关性分析

在实际的生产和生活中，许多事物密不可分，其中有些是紧密相连的，有些是稀疏的。有两种表达相互联系的事物相互依存的方法：相关性和回归（函数关系）。相关关系不是确定关系，当一个或几个事物的取值发生变化时，与它（它们）有联系的事物的取值也会发生变化，但变化值不是确定的数值。基于这些区别，在数据分析中，一般先做相关关系的分析，待相关关系清楚以后，再进一步确定不同变量之间的函数关系（回归关系）[18]。

2.2.1 相关分析

在本次课题中，我们通过爬虫等方式可以获得开发人员的相关数据，如提交代码行数、提交次数、增加代码行数、删除代码行数等，通过对这些指标进行相关性分析，可以更加清楚的了解开发者贡献评估指标的相关性特征，如增加代码行数可能和提交次数有着较强的关联性，从而为我们之后进一步的行为特征分析提供方向。

具体而言，两个固定范围变量和固定变量之间的相关性是用 Pearson（皮尔逊）相关系数来判定，属于参数检验的方法；两个有序或固定变量之间的相关性用 Spearman 等级相关系数和 Kendall's tau-b 等级相关系数来判定，这两种方法是非参数检验[19][20]。

2.2.2 Pearson 积矩相关

Pearson 相关评估两个连续变量之间的线性关系。当一个变量中的变化与另一个变量中的成比例变化相关时，这两个变量具有线性关系。例如，您可能使用 Pearson 相关来评估您生产设施温度的升高是否与巧克力涂层的厚度减少有关。其中，Pearson 相关系数公式如下：

$$COR(X,Y)=\frac{COV(X,Y)}{\delta_x\delta_y}=\frac{\sum_1^n(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_1^n(X_i-\bar{X})^2(Y_i-\bar{Y})^2}} \quad (2\text{-}1)$$

从上述公式可以看出，Pearson 相关系数是通过协方差除以两个变量的标准差得到的。协方差能反映两个随机变量的相关程度（当协方差大于 0 的时，表示两者之间为正相关，当协方差小于 0 的时，表示两者之间为负相关），但受变量的量纲的影响很大，不能简单地根据协方差的大小判断变量之间的相关程度。为了消除量纲的影响，于是诞生了相关系数的概念。当两个变量的方差都不为零时，相关系数才有意义，相关系数的取值范围为$[-1,1]$。当相关系数为 1 时，变量关系为完全正相关；当相关系数为-1 时，变量关系为完全负相关；相关系数的绝对值越大，变量相关性越强；相关系数越接近于 0，变量相关性越弱[21]。

2.2.3 Spearman 秩次相关

Spearman 相关评估两个连续或顺序变量之间的单调关系。Spearman 相关系数基于每个变量的秩值（而非原始数据）。Spearman 相关通常用于评估与顺序变量相关的关系[22]。

假设两个随机变量分别为X、Y（也可以视为两个集合），并且其元素数均为N，两个随即变量取的第$i(1 \le i \le N)$个值分别用X_i、Y_i表示。对X、Y进行排序（同时按升序或降序进行排序），得到两个元素排名集合x、y，其中元素x_i、y_i分别为X_i在X中的排名以及Y_i在Y中的排名。对应元素地将集合x、y中的元素进行相减，得到一个排行差分集合d，其中$d_i = x_i - y_i$，$(1 \le i \le N)$。随机变量X、Y之间的 Spearman 等级相关系数可以由x、y或者d计算得到，其计算方式如下：

由排行差分集合d计算而得：

$$\rho = 1 - \frac{6\sum_1^N {d_i}^2}{N(N^2-1)} \tag{2-2}$$

由排行集合x、y计算而得（斯皮尔曼等级相关系数在另一个角度上也可以看作是经过排名的两个随即变量的 Pearson 相关系数，以下实际是计算x、y的 Pearson 相关系数）：

$$\rho = \frac{\sum_1^n (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_1^n (x_i-\bar{x})^2 (y_i-\bar{y})^2}} \tag{2-3}$$

与 Pearson 相关系数相比，Spearman 相关系数对于数据异常值和极值的反应不敏感。只要两个变量的观测值是成对的等级评估数据，或从连续变量观测数据转换而来的等级数据，则无论两个变量的总体分布如何，都可以采用 Spearman 相关系数进行研究，因此其所需的数据条件比 Pearson 相关系数少。

2.3 特征提取技术

在多元统计分析中，主成分分析（Principal components analysis，PCA）是一种用于分析和简化数据集的技术。主成分分析通常用于降低数据集的维数，同时保持数据集中最大方差贡献的特征。这是通过保留低阶主成分而忽略高阶主成分来完成的。这种低阶成分通常可以保留数据的最重要方面[25]。

在对开发者进行贡献评估时，我们将对提取出多个贡献指标，对于这些贡献指标，哪一些更加重要，并且可以更好的描述一个开发者的贡献程度，是我们想要得到的结果。通过主成分分析技术，我们可以提取出较为重要的开发者贡献指标，以方便实验的后续研究。

PCA 方法主要分解协方差矩阵以获得数据的主成分（即特征向量）及其权重（即特征值）。结果可以理解为原始数据贡献的解释：哪些部分对原始数据值的贡献最大？换句话说，PCA 提供了减少数据量的有效方法。

PCA 的目标是找到一组新的正交基$\{u_1, u_2, \cdots, u_k\}$（从$n$维下降到$k$维），使得数据点在该正交基构成的平面上投影后，数据间的距离最大，即数据间的方差最大。如果数据在每个正交基上投影后的方差最大，那么同样满足在正交基所构成的平面上投影距离最大。

设正交基u_j，数据点x_i在该基底上的投影距离为${x_i}^T \cdot u_j$，所以所有数据在该基地上的投影的方差J_j为：

$$J_j = \frac{1}{m}\sum_{i=1}^{m}({x_i}^T u_j - {x_{center}}^T u_j)^2 \tag{2-4}$$

其中：m为样本数量，在数据运算之前对数据 x 进行 0 均值初始化，即$x_{center} = 0$，从而

$$J_j = \frac{1}{m}\sum_{i=1}^{m}({x_i}^T u_j)^2 = \frac{1}{m}\sum_{i=1}^{m}({u_j}^T x_i \cdot {x_i}^T u_j) = {u_j}^T \cdot \frac{1}{m}\sum_{i=1}^{m}(x_i {x_i}^T) \cdot u_j \tag{2-5}$$

所以：

$$J_j = {u_j}^T \cdot \frac{1}{m}(x_1 {x_1}^T + x_2 {x_2}^T + \cdots + x_m {x_m}^T) \cdot u_j = {u_j}^T \cdot \frac{1}{m}\left([x_1 \cdots x_m]\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}\right) \cdot u_j =$$

$$\frac{1}{m}{u_j}^T X X^T u_j \tag{2-6}$$

由于$\frac{1}{m}XX^T$为常数，这里假设$S = \frac{1}{m}XX^T$，则：

$$J_j = {u_j}^T \cdot S \cdot u_j \tag{2-7}$$

此时根据 PCA 目标，我们需要求解J_j最大时对应的u_j。以下根据 Lagrange 乘数法求极值进行求解：

$$J_j = {u_j}^T \cdot S \cdot u_j, s.t. {u_j}^T u_j = 1 \tag{2-8}$$

则构造函数：

$$F(u_j) = {u_j}^T \cdot S \cdot u_j + \lambda(1 - {u_j}^T u_j) \tag{2-9}$$

求解$\frac{\partial F}{\partial u_j}$，得：

$$2S \cdot u_j - 2\lambda_j \cdot u_j = 0 \tag{2-10}$$

即：

$$S \cdot u_j = \lambda_j \cdot u_j \tag{2-11}$$

当λ_j，u_j分别为S矩阵的特征向量、特征值时，J_j有极值，把上述结果带回公式得：

$${J_j}_m = {u_j}^T \cdot \lambda_j \cdot u_j = \lambda_j \tag{2-12}$$

所以对于任意满足条件的正交基，对应的数据在上面投影后的方差值为S矩阵的特征向量，从而：

$$J_{max} = \sum_{j=1}^{k} \lambda_j , \lambda_j \ from \ big \ to \ small \ order \tag{2-13}$$

所以投影正交基为S的特征向量中的前k个最大特征值对应的特征向量。

接下来对S进行特征分解，S的特征向量集合：

$$U = U \ of \ svd(S) = U \ of \ svd(\frac{1}{m} XX^T) \tag{2-14}$$

另外，由于$S = \frac{1}{m} XX^T$，由于 X 已 0 均值处理，根据协方差矩阵定义：S为数据集X的协方差矩阵。

综上，即可得到满足投影后数据距离最大的新的正交基$\{u_1, u_2, \cdots, u_k\}$

因此：

$$X_{new_{k*m}} = \begin{bmatrix} u_1{}^T \\ u_2{}^T \\ \vdots \\ u_k{}^T \end{bmatrix}_{k*n} \cdot X_{n*m} \tag{2-15}$$

在对 PCA 结果分析时，数据的方差可以表示数据波动的变化信息，从上述公式可知，数据的方差等于协方差所有特征值之和，第 i 个主成分的方差等于协方差矩阵的第 i 个特征值。

2.4 回归分析

在统计学中，线性回归（Linear Regression）是一种回归分析，它使用称为线性回归方程的最小二乘函数对一个或多个自变量和因变量之间的关系进行建模。此函数是一个或多个模型参数（称为回归系数）的线性组合。仅具有一个自变量的情况称为简单回归，而具有多个自变量的情况称为多重回归（Multivariable Linear Regression）[10]。

对于本次研究课题，通过选定一个代表性贡献指标，然后对其他指标进行回归分析，可以建立多元回归模型。对模型的回归系数进行分析，可以判断对原始数据集拟合效果的好坏；对模型的决定系数进行分析，可以获取更具有代表性的贡献指标。

对于一个给定的随机样本$(Y_i, X_{i1}, X_{i2}, \cdots, X_{ip}), i = 1,2, \cdots, n$，我们假设因变量$Y_i$和自变量$X_{i1}, X_{i2}, \cdots, X_{ip}$之间的关系除了包括$X$的影响以外，还有其他影响存在。因此加入一个误差项$\varepsilon_i$来描述除了$X_{i1}, X_{i2}, \cdots, X_{ip}$之外任何对$Y_i$的影响。所以一个多变量线性回归模型可以表示为以下的形式：

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} + \varepsilon_i, i = 1,2, \cdots, n \tag{2-16}$$

向量形式为：

$$Y = X\beta + \varepsilon \tag{2-17}$$

使用最小二乘法估计为：

$$\hat{\beta} = (X^T X)^{-1} X^T Y \tag{2-18}$$

为了评估回归模型的效果，即因变量的波动有多少可以由自变量的波动所描述，我们引入决定系数来评估。

$$SS_{res} = \sum_{i=1}^{n} (y_i - f_i)^2 \tag{2-19}$$

$$SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \quad (2\text{-}20)$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad (2\text{-}21)$$

决定系数的范围在[0,1]，决定系数越大，拟合优度越大，自变量对因变量的解释程度也越大。

2.5 本章小结

在本节中，主要介绍了数据挖掘中常用到的相关性分析、PCA 特征提取技术、回归分析的相关概念以及它们的数学理论，在之后研究变量过程中将会使用相关方法。

3 研究过程

本次课题的研究过程按照实证研究的一般流程来进行。首先是开源社区开发者数据集的获取；其次是对数据集进行预处理、提取；接下来将对数据集进行项目和开发者两个维度上的分析：使用贡献度和开发者贡献模型指标，对这些指标进行相关分析、特征提取、回归分析；然后对高贡献度的开发者进行时间维度上的特征分析、地域特征分析、合作关系特征分析等。

整个研究过程如图 3.1 所示。数据集来源为两个部分，一部分来自 github 本地克隆后通过 git log 命令获取，另一部分来自于 Github Rest Api V3。之后我们对数据全部进行 json 结构化存储，然后编写 Python 脚本对数字型数据进行数据预处理，最后进行数据分析和可视化，得出相关结论。

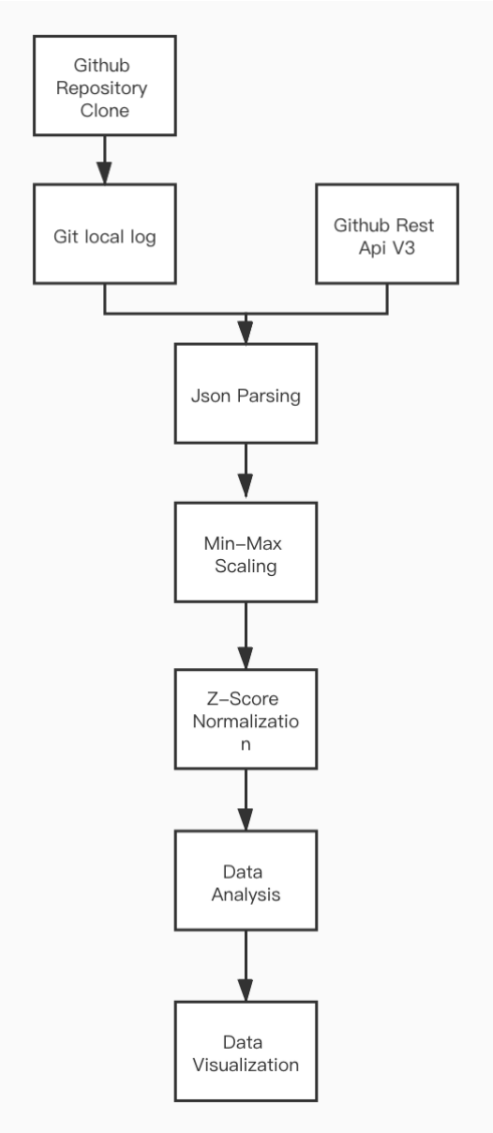

图 3.1 研究过程

3.1 数据收集

3.1.1 版本控制系统

版本控制的主要功能是跟踪文件更改。它将详细记录何时以及谁更改了文件的内容。每次更改文件时，文件的版本号都会增加。除了记录版本更改外，版本控制的另一个重要功能是协作开发。软件开发通常是多人的协同操作，版本控制可以有效地解决版本同步和不同开发人员之间的开发沟通问题，提高协同开发效率。协同开发中最常见的不同版本软件的错误(Bug)修正问题也可以通过版本控制中分支与合并的方法有效地解决[17][18]。

具体来说，在每个开发任务中，开发者需要首先设置一个开发基准，并确定每个配置项的初始开发版本。在开发过程中，开发人员根据开发基线的版本来开发所需的目标版本。发生需求变更

时，通过评估变更，确定变更的范围，并修改受影响的配置项的版本。根据更改的性质，配置项的版本树会继续扩展，或者会生成新分支以形成新的目标版本，并且不受更改影响的配置项不应更改。同时，它应该能够记录和跟踪更改对版本的影响。如有必要，开发者可以返回以前的版本。

目前有两种主流的版本控制系统，分别为：集中式的版本控制系统 SVN 和分布式的版本控制系统 Git[7][13]。Git 是目前业界最流行的版本控制系统（Version Control System），而 GitHub 是开源代码托管平台的翘楚。越来越多的从业者、从业团队以及开源贡献者首选 GitHub 用于管理项目代码。

GitHub 里面的项目可以通过标准的 Git 命令进行访问和操作。同时，所有的 Git 命令都可以用到 GitHub 项目上面。网站提供了一系列社交网络具有的功能[29]，例如点赞（star）、关注（follow）、评论。用户可以通过复刻（fork）其他开发者的项目参与开发。此外 GitHub 还有 Wiki 等功能。GitHub 同时允许注册用户和非注册用户在网页中浏览项目，允许以 ZIP 格式打包下载项目代码文件。

本次课题的数据集通过本地克隆分析和 Github Api V3 两个方式获取。

3.1.2 用本地克隆技术获取数据

本地克隆的项目来自于 github.com/topics（Github 提供的关于热点仓库的搜索功能）中热门的主题仓库。在衡量每个仓库的实际活跃情况以及涵盖度，最终选择了以下 17 个仓库作为数据来源。

- FFmpeg：FFmpeg 是一个用于快速处理音频、视频等文件的命令集合和框架，具有免费、易于使用等特点。
- Babel：Babel 是一个编译工具，由于历史原因，部分浏览器不支持较新版本的 JavaScript 语法，因此对于前端开发人员，需要将 ECMAScript 2015 以上版本的代码转换为浏览器支持的低版本的 JavaScript 语法。
- Bitcoin：比特币（Bitcoin, 简称 BTC）是世界上第一个数字货币，最早由日本开开发者“中本聪”以开源软件的形式推出，实现了一种 P2P 网络支付系统。
- Bootstrap：Bootstrap，由 Twitter 公司开发，是目前最为流行的前端框架。它由 Twitter 的 Mark Otto 和 Jacob Thornton 合作开发，已在世界诸多知名网站得以应用，受到前端开发人员的青睐。
- Cocos2d-x：Cocos2d-x 是一个移动游戏框架，帮助开发者可以快速开发一款 2D 游戏。
- Flutter：Flutter 是由 Google 推出并开源的移动应用开发框架，具有跨平台、高保真、高性能等特点。开发者可以通过 Dart 语言开发 App，编写一套代码，便可以同时在 web、iOS、Android 多种平台运行。

- gRPC：gRPC 是一个高性能、通用的开源 RPC 框架，由 Google 公司开发，基于 HTTP/2 协议标准而设计，使用协议为 ProtoBuf(Protocol Buffers)，支持 Java、Golang、Python 等多种开发语言。
- Kubernetes：Kubernetes（常简称为 K8s）是用于自动部署、管理容器的开源系统，多用于管理 docker 容器，极大地促进了容器化普及。
- Pytorch：PyTorch 是一个开源机器学习库，用于计算机视觉、nlp 等研究任务。由 Facebook 公司人工智能实验室开发，可以方便地实现 GPU 加速，还支持动态神经网络。
- Spark：Spark 是一个 Apache 顶级开源项目，性质是一个快速、通用的大规模数据处理引擎， 具有可伸缩、基于内存计算、直接读写 Hadoop 等特点，Spark 已经成为轻量级大数据应用的统一平台，可以完成实时流处理、交互式查询、垃圾信息过滤等各种应用。
- Spring-framwork：Spring 是一个开源 Java web 框架，帮助开发者可以更快速创建基于 Java 的网站应用。
- Three.js：Three.js 是一个炫酷的 Javascript 3D 库，以 WebGL 为标准，简化了 WebGL 开发复杂度和降低了入门难度。
- TiDB：TiDB 由 PingCAP 公司开发设计，该数据库统一了传统的 RDBMS 和 NoSQL 的最佳特性，因其支持无限的水平扩展，具备强一致性和高可用性，在各大商业后台得到广泛应用。
- Redis：Redis 是一个开源的、极高读写速度的、基于内存实现的键值对存储系统。
- openCV：OpenCV 主要实现了实时计算机视觉相关算法和框架，具有跨平台的特点，是如今计算机视觉领域最为常用的第三方库。
- Flask：Flask 是一个轻量级 Web 应用框架，使用 Python 语言编写，适用于创建小型网站项目。
- Vue：Vue 是一套用于快速构建用户界面和实现交互能力的 JavaScript 框架，采用自下而上增量开发的设计的渐进式框架。

可以看到，这 17 个项目涵盖了开发者的大多数活动领域，如：web 前端框架、web 后端框架、网络通信、编译软件、游戏开发、计算机视觉、机器学习、数据库、缓存、大数据引擎、渲染框架、客户端开发、音视频多媒体处理、电子货币等等。从而保证本次课题数据来源可靠，泛化性强。

在将项目克隆到本地后，使用 git log 命令进行特定数据获取。如统计仓库个人开发量、个人增删代码量、仓库提交者信息（排序）、开发者数量统计、代码提交数量统计、仓库代码总行数等。

```
# 统计仓库个人开发量
git log --author="username" --pretty=tformat: --numstat | awk '{ add += $1; subs += $2; loc += $1 - $2 } END
# 统计每个人的增删代码量
git log --format='%aN' | sort -u | while read name; do
  echo -en "$name\t"
  git log --author="$name" --pretty=tformat: --numstat | awk '{ add += $1; subs += $2; loc += $1 - $2 } END
done
# 查看仓库提交者排名前 5
git log --pretty='%aN' | sort | uniq -c | sort -k1 -n -r | head -n 5
# 贡献者数量统计
git log --pretty='%aN' | sort -u | wc -l
# 提交数量统计
git log --oneline | wc -l
# 统计代码总行数:
# shellcheck disable=SC2038
find . -name "*.m" -or -name "*.h" -or -name "*.xib" -or -name "*.c" | xargs grep -v "^$" | wc -l
```

图 3.2 git log shell 命令获取数据

同时对 git log 进行解析，获取 json 化 commit 数据。对于每一条 commit 数据，得到 json 字典，其中 key 包括：commit ID、tree、parents commit、author（name、email、date、timezone）、committer、message、changes（add、del、file）。样本数据如下图所示：


```
{
  "commit":"420f2ee69aad0316c6f49e9dfdc6a02c8f9d4fc8",
  "tree":"5b8066743e0833e2dd1e99c4b34b9c18cc1fdf75",
  "parents":[
    "c831a2450dbf252c75750a455c63e1016c2f2244"
  ],
  "author":{
    "name":"Babel Bot",
    "email":"babel-bot@users.noreply.github.com",
    "date":1584039068,
    "timezone":"+0000"
  },
  "committer":{
    "name":"Babel Bot",
    "email":"babel-bot@users.noreply.github.com",
    "date":1584039068,
    "timezone":"+0000"
  },
  "message":"Add v7.8.8 to CHANGELOG.md [skip ci]",
  "changes":[
    [
      26,
      0,
      "CHANGELOG.md"
    ]
  ]
}
```


图 3.3 git log commit json 化数据样本

3.1.3 用 GitHub REST API V3 获取数据

GitHub REST API v3 是一个基于 Restful 风格的调用接口。通过发送 http 请求，可以获取到对应的 json 数据。本次课题主要用到以下请求：

- 个人所有 repo：/users/用户名/repos，返回一个 repo 的 Json 格式列表。
- 个人主要信息：/users/用户名，返回个人的登录信息。
- repo 详细信息： /repos/用户名/仓库名。repo 的路径就开始和个人信息不同了。
- 获取某个 repo 的内容列表：/repos/solomonxie/gists/contents，注意这只会返回根目录的内容。
- 获取 repo 中某文件信息（不包括内容）：/repos/user/gists/contents/文件路径。文件路径是文件的完整路径，区分大小写。只会返回文件基本信息。
- repo 中所有的 commits 列表： /repos/用户名/仓库名/commits。
- 某一条 commit 详情： /repos/用户名/仓库名/commits/某一条 commit 的 SHA
- issues 列表： /repos/用户名/仓库名/issues。

获取一个样本用户的个人信息如下，其中包括登录名、blog、位置、邮箱、follower 数、following 数等用户信息，也包括组织（organization）、仓库（repos）、事件（events）等社交信息：


```
{
  "login": "sebmck",
  "id": 853712,
  "node_id": "MDQ6VXNlcjg1MzcxMg==",
  "avatar_url": "https://avatars0.githubusercontent.com/u/853712?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/sebmck",
  "html_url": "https://github.com/sebmck",
  "followers_url": "https://api.github.com/users/sebmck/followers",
  "following_url": "https://api.github.com/users/sebmck/following{/other_user}",
  "gists_url": "https://api.github.com/users/sebmck/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/sebmck/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/sebmck/subscriptions",
  "organizations_url": "https://api.github.com/users/sebmck/orgs",
  "repos_url": "https://api.github.com/users/sebmck/repos",
  "events_url": "https://api.github.com/users/sebmck/events{/privacy}",
  "received_events_url": "https://api.github.com/users/sebmck/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Sebastian",
  "company": null,
  "blog": "",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": null,
  "public_repos": 3,
  "public_gists": 32,
  "followers": 3990,
  "following": 102,
  "created_at": "2011-06-16T09:33:55Z",
  "updated_at": "2020-03-23T03:42:27Z"
}
```


图 3.4 github rest api v3 用户数据样本

3.2 数据预处理

在机器学习领域，不同的评估指标（即特征向量中的不同特征是不同的评估指标）往往具有不同的维数和维数单位，这种情况会影响数据分析的结果。因此需要进行数据标准化处理，使得数据

指标之间具有可比性。原始数据经过数据标准化处理后，指标处于同一数量级，适合进行综合比较评价[19]。最常用的方法主要有以下两种:

线性函数归一化（Min-Max Scaling）。该数据归一化方法对原始数据施以线性变换，使结果映射到$[0,1]$的范围，得到原始数据的等比缩放后的归一化数据。归一化公式如下:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

其中，X为原始数据，X_{max}、X_{min}分别为数据最大值和最小值。

零均值标准化（Z-Score Normalization）。该数据标准化方法会将原始数据映射成为均值为 0、标准差为 1 的分布。具体来说，若原始特征的均值为μ、标准差为δ，则标准化公式为:

$$z = \frac{X - \mu}{\delta}$$

3.3 数据分析

3.3.1 贡献度评估指标

在考虑实际开发者的贡献度量时，需要考虑的因素很多。在本次课题中，从代码和项目时间两个维度，定义了增加代码行数（add-line）、删除代码行数（del-line）、commits 数量（commits-count）、在项目中年限（age）、活跃天数（active-days）五个指标来作为开发者的贡献指标，并进行研究[3][4][28]。

基于 Gousios 等人以代码相关和非代码相关对贡献指标进行分类，进而提出的一套较复杂的开发人员贡献行为度量标准[2]，结合本文研究的实际情况 ，采取如下的计算公式。

针对代码贡献，采用 LOC 和 commit 次数相结合的方式[23][27]。commit 次数是对每个开发人员 commit 的项数统计，而 LOC 则是每个开发人员所有 commits 的修改行数累加。另外，假设 LOC 和 commit 次数有相同的权值，分别计算各自的贡献度后求平均[28]。具体的第 i 个贡献者的度量度公式如下[2]:

$$contri(i) = \frac{LOC(i)}{\sum_{k=1}^{n} LOC(k)} + \frac{Commit(i)}{\sum_{k=1}^{n} Commit(k)}, 1 \le i \le n$$

可见$contri(i)$的值域范围为$[0,2]$，越接近 2 则说明对项目的贡献程度越大。

对以上贡献度评估指标和贡献度进行数据分析主要采取 2.1 至 2.3 部分的相关算法。对于相关分析，由于原始数据大多为等级资料，因此最终选择采用 Spearmanr 等级相关；另外使用多元线性回归、PCA 算法对贡献指标进行分析。相关代码如下图所示:

```
def spearmanr_rank(data):
    corr, p_value = spearmanr(data)
    return corr, p_value


def pca_analysis(x, n=2):
    pca = PCA(n_components=n)
    principal_components = pca.fit_transform(x)
    print("pca.pca.explained_variance_|", pca.explained_variance_)
    print("pca.explained_variance_ratio_|", pca.explained_variance_ratio_)
    pca.components_ = np.around(np.asarray(pca.components_), decimals=3)

    np.set_printoptions(suppress=True)
    print("pca.components|", pca.components_)
    return principal_components


def linear_regression(x, y):
    # Note the difference in argument order
    x = sm.add_constant(x)
    model = sm.OLS(y, x).fit()
    predictions = model.predict(x)  # make the predictions by the model
    # Print out the statistics
    return model.summary()
```

图 3.5 数据分析相关代码

3.3.2 合作关系分析

合作关系是指开发社区中开发者使用 git 方式进行协作开发时出现的行为。如共同修改了文件、git 产生冲突、解决冲突等行为。

通过对上述 17 个项目的 git log 日志可以获取开发人员们在修改文件上的合作关系。在对 git log 进行 json 格式化的数据中可以看到，change 项中包含了修改的文件名称，以及对应的添加和删除行数，而在 committer 项中，可以拿到此次 commit 提交的作者名。通过统计每个文件对应的修改次数以及修改的作者名称，可以构建基于修改文件的开发人员合作关系模型。

```
def parse_log_json_and_save_file(path="data/git-log-json/grpc_gitlog.json",
                                 out_path="data/git-log-json/grpc-file-analysy.json"):
    file_count_dict = {}
    file_user_dict = defaultdict(set)
    with open(path, "r", encoding="utf-8") as f:
        with open(out_path, "w") as fw:
            data = json.load(f)
            for e in data:
                for change in e["changes"]:
                    if change[2] in file_count_dict:
                        file_count_dict[change[2]] += 1
                    else:
                        file_count_dict[change[2]] = 1
                    file_user_dict[change[2]].add(e["author"]["name"])
            sort_file_count = sorted(file_count_dict.items(), key=lambda item: item[1])
            fw.write(json.dumps(sort_file_count))
            fw.write("\n")
            fw.write(str(dict(file_user_dict)))
```

图 3.6 合作关系特征分析相关代码

3.3.3 地域、邮箱域特征分析

地域特征分析是指对开源项目的开发人员所处地域进行统计分析，从而发现一些地域上的特征。邮箱域分析是指对开发人员每次提交代码时，所使用的邮箱用户名的网络域进行分析，从而发现相关特征。

对于地域信息，可以查看开发人员的 email 信息、Github 提供的定位信息，并进行相关推断，可以得到开发人员的地域信息。最后以国家为单位 ，进行统计。

对于邮箱域，采取对 git log（json 格式化后）进行分析，对每次的 commit 作者的 email 进行域名统计分析。

```
def parsing_data(path="data/spark_users_data.json", field="company"):
    count_list = []
    with open(path, "r", encoding="utf-8") as f:
        data = json.load(f)
        for e in data:
            if e[field] is not None:
                count_list.append(e[field])
            else:
                count_list.append("default")
    _, l1, l2 = use_list_count(count_list)
    return l1, l2
```

图 3.7 地域特征分析相关代码

3.3.4 时间特征分析

时间特征分析是指在研究开发人员贡献行为时，加入时间维度进行分析。如不同开发人员提交代码量、提交代码次数等行为随时间变化的情况。

在对开发人员进行时间序列上的分析时，主要从项目总体的活跃情况和高贡献度作者在时间维度上的添加代码行数、commit 提交次数等方面进行可视化分析。

实现上本文采用开源社区开发的一款命令行工具 Gitstats 进行分析。Gitstats 是针对 git 版本管理开发的历史信息统计工具，非常适用于在时间维度上对 git 仓库进行信息统计。

3.4 数据可视化

可视化采用 JavaScript Echarts 和 Google Map Api 进行展示。

ECharts，是一个纯 Javascript 的图表库，底层依赖轻量级的 Canvas 类库 ZRender，提供直观，可交互，支持个性化定制的可视化图表。ECharts 提供了常规的折线图、条形图、散点图、饼图，盒形图，用于地理数据可视化的地图、热力图、线图，用于数据关系可视化的关系图，树图，多维数据可视化的平行坐标等等多种绘图接口。

对于使用 Echarts，我们将之前数据分析的结果保存为 json 结构的 list、dict 等，然后调用对应的 JavaScript 接口，将 json 数据反序列化，传入对应变量，进行图形绘制。

Google Map Api 是由 Google 公司提供的一组定位服务 Api。使用其中的 Maps JavaScript API 进行地域特征的展示。通过 Maps JavaScript API，可以使用自己的内容和图像来自定义地图，以显示在网页上。Maps JavaScript API 具有四种基本地图类型（路线图、卫星图、混合图和地形图），并且可以使用图层和样式，控件和事件以及各种服务和库进行修改。

对于使用 Google Map Api，具体使用步骤和相关代码如下：

1、通过<script>标签加载 Google Map JavaScript Api；

2、定义<body>标签的类属性为 map；

3、创建一个 google.visualization.GeoChart 对象；

4、创建 data_table 表对象进行绘制数据的存储，并使用 addRow()方法添加数据；

5、将 data_table 作为参数传入 GeoChart 对象，并调用 draw()绘制;

```
<body class='map'>
  <div id='chart_markers'>
    <script type='text/javascript'>
      google.load('visualization', '1', {
        packages: ['geochart'], callback: function () {
          var data_table = new google.visualization.DataTable();
          data_table.addColumn({ "type": "string", "label": "Location" });
          data_table.addColumn({ "type": "number", "label": "Contributors" });
          data_table.addRow([{ v: "United States" }, { v: 1709062 }]);
          var chart = new google.visualization.GeoChart(document.getElementById('chart_markers'));
          chart.draw(data_table, { displayMode: "region", region: "world", legend: "none", colors: ["A6E3BE","18A851"] });
        }
      });
    </script>
  </div>
</body>
```

图 3.8 Google map 可视化相关代码

3.5 本章小结

在这一部分，系统的介绍了整个实验流程和实验各个部分的实现方案。包括数据获取、抽取信息、预处理、贡献度计算方式定义、相关观测变量的定义、特征分析方案及可行性、可视化技术的选择。

4 实验结果及分析

4.1 实验环境

本文的实验均在 macOS Catalina v10.15.3 系统中进行，具体环境如下所示：

表 4.1 实验环境

CPU	RAM	GPU	Language
2.3 GHz 四核 Intel Core i5	16 GB 2133 MHz LPDDR3	Intel Iris Plus Graphics 655 1536 MB	Python 3.7 JavaScript

4.2 实验数据

实验数据来自于 github.com/topics（Github 提供的关于热点仓库的搜索功能）中热门的主题仓库。在衡量每个仓库的实际活跃情况以及涵盖度，选择了以下 17 个项目。这 17 个项目涵盖了开发者的大多数活动领域，如：web 前端、web 后端、网络通信、编译软件、游戏开发、计算机视觉、机器学习、数据库、缓存、大数据引擎、渲染框架、音视频多媒体处理、电子货币等等。从而保证本次课题数据来源可靠，泛化性强。本次课题数据集基于 17 个项目，共计 19437 名开发人员，500979 次 commit 贡献，平均采集周期为 10 年。

表 4.2 实验项目信息

仓库项目名称	版本数（tag）	开发人员数	Commits 数	当前版本文件数	数据采集周期	项目领域
Babel	649	924	13607	18774	2012.09-2020.03	编译工具链
Bitcoin	231	870	23248	1924	2009.08-2020.03	电子货币
Bootstrap	57	1302	19449	515	2011.04-2020.03	Web 前端
Cocos2d-x	98	870	37344	4254	2010.07-2020.03	游戏开发
FFmpeg	324	1839	97014	7264	2000.12-2020.03	多媒体处理
Flask	31	670	3845	221	2010.04-2020.03	Web 后端
Flutter	295	590	18083	4599	2014.10-2020.03	移动开发
Grpc	180	637	43604	8191	2014.11-2020.03	Rpc 框架
Kubernetes	631	3062	89445	22109	2014.06-2020.03	容器治理

Opencv	90	1507	28272	6139	2010.05-2020.03	计算机视觉
Pytorch	32	1790	24930	6623	2012.01-2020.03	机器学习
Redis	228	426	9017	801	2009.03-2020.03	缓存
Spark	117	2097	26675	16170	2010.03-2020.03	大数据
Spring	173	533	20419	8485	2008.07-2020.03	Web 后端引擎
Three.js	106	1543	32379	4007	2010.03-2020.03	渲染框架
Tidb	95	463	10556	1216	2015.09-2020.03	数据库
Vue	249	314	3092	549	2016.04-2020.03	Web 前端

4.3 实验结果

4.3.1 贡献指标主成分分析

首先对各个观测变量进行主成分分析，找出各个观测变量对贡献的影响程度。

提取 PCA 信息时，首先将各个观测变量的系数（loadings，即公式（2-15）中的系数项）进行提取，如下表所示：

表 4.3 PCA loadings

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
Commit	0.479	0.241	0.598	0.575	-0.158
Add-line	0.447	-0.525	-0.178	-0.155	-0.685
Del-line	0.448	-0.53	-0.119	0.162	0.692
Age	0.349	0.507	-0.743	0.262	-0.027
Active-day	0.499	0.36	0.212	-0.742	0.163

因此可以推导出 Comp. 1 和 Comp. 2 的表达式：

$$comp_1 = 0.479 * commit + 0.447 * addline + 0.448 * delline + 0.349 * age + 0.499 * activeday$$

$$comp_2 = 0.241 * commit - 0.525 * addline - 0.53 * delline + 0.507 * age + 0.36 * activeday$$

其次，主成分分析模型信息如下表：

表 4.4 PCA 模型信息

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
Explained variance	3.052	1.244	0.553	0.100	0.067

Explained variance ratio	0. 609	0. 248	0. 110	0. 020	0. 013
Cumulative proportion	0. 609	0. 857	0. 967	0. 987	1. 000

其中 Comp.1～Comp.5 是由原来五个观测变量（即增加代码行数（add-line）、删除代码行数（del-line）、commits 数量（commits-count）、在项目中年限（age）、活跃天数（active-days））线性组合变换后的新特征。第一行 Explained variance 表示的是主成分的标准差，第二行 Explained variance ratio 表示的各个主成分的贡献率，第三行 Cumulative proportion 表示的累积贡献率，最终主成分的累积贡献率为 100%，即完全解释原数据包含的信息。

将表 4. 4 中的 Explained variance ratio（表示的各个主成分的贡献率）绘制成饼状图如下：

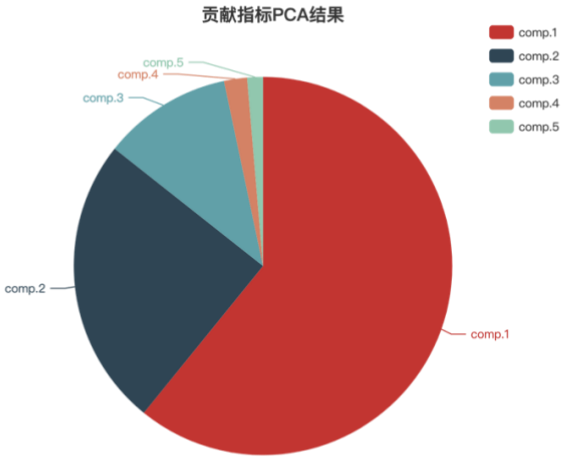

图 4. 1 PCA 贡献率饼状图

通过观察 PCA 贡献率的饼状图，可以看到第一主成分、第二主成分和第三主成分占据了 96. 7% 的信息表示，因此可以很好表示各个开发者在每个观测维度上的综合信息。对于第一主成分，各个观测变量对应的系数符号相同，且值均在 0. 4 左右，它反映了开发者的贡献程度。对于高贡献者，在观测指标上的值都比较大，因此，第一主成分值比较小；而低贡献者，在观测指标上的值比较小，因此，第一主成分的绝对值比较大。

通过对每个项目 commit 总量排行前 20 的开发者结合第一主成分和第二主成分进行散点图的绘制。

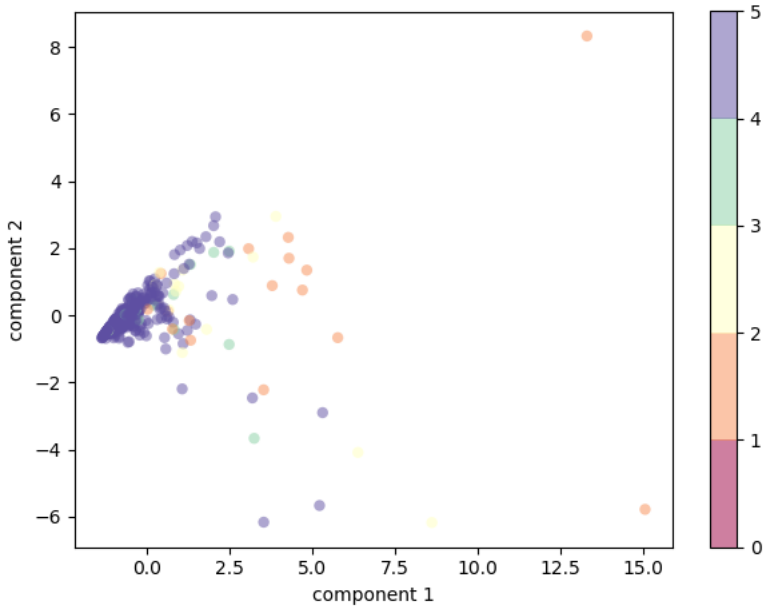

图 4.2 PCA Comp.1&2 数据分布散点图

从上图可知，高贡献者的第一主成分取值范围和第二主成分的取值范围大概为$(-1,2)$。

4.3.2 贡献指标相关性分析

通过 PCA 分析各个观测指标后，可以使用经过线性变换后的新指标来描述一个开发者的贡献程度，并作了一些具体的分析。但还要关心变量之间的关系，下面是相关性分析的结果。

采用 Spearman 相关系数计算变量之间的相关性，从而避免数据的分布问题。

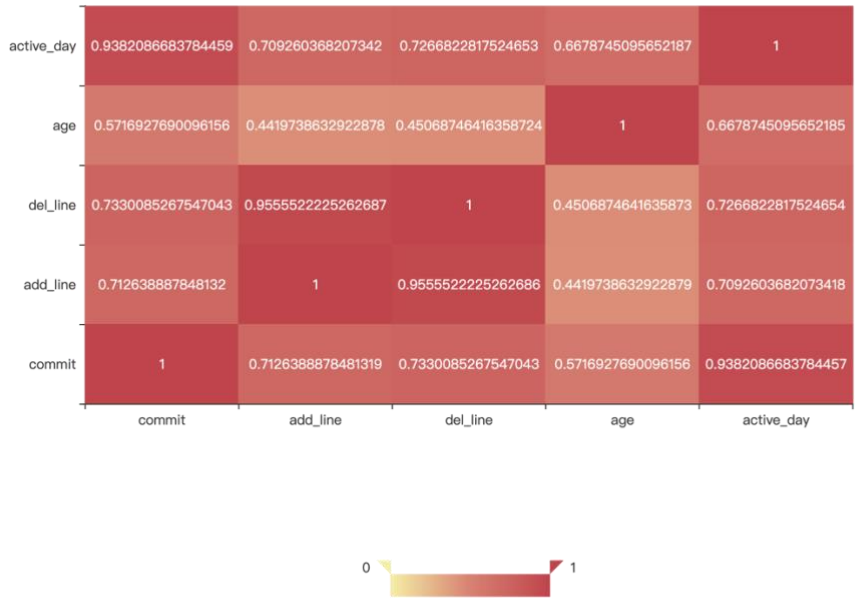

图 4.3 关系矩阵热点图

通过关系矩阵热点图可以发现，commits 次数对于剩余指标有着非常大的相关性，因为 git 版本控制的贡献主要方式便是 commit。可见，贡献度的计算公式中采用 commits 次数作为其中一项，可以很好描述贡献度。

在时间维度上，一个开发者经历项目的时间与其他指标相关性差，但是活跃天数相关性强。因此活跃天数也可以作为评估开发者贡献程度大小的重要指标。

4.3.3 贡献指标回归分析

本文继续引入回归分析，让 commit 次数和其他指标进行单变量的回归分析。其结果如下，其中，Coef 表示回归方程的参数估计，std-err 表示回归参数的标准差，R-square 表示决定系数，用于度量因变量的变异中可由自变量解释部分所占的比例。

表 4.5 回归分析矩阵

	Coef	Std-err	R-square
Add-line	0.4334	0.049	0.188
Del-line	0.4562	0.048	0.208
Age	0.4293	0.049	0.184
Active-day	0.8606	0.028	0.741

通过回归分析矩阵中 R-square 的观测，取得最大值是 Active-day 这一观测因素，在拟合贡献上，这一指标具有比较强的代表性。最低的是 Age，即开发人员参与时间。

4.3.4 开发人员贡献数据分布

根据 3.2 中定义的贡献度评估公式，计算出每个项目 top20 的高贡献开发人员。

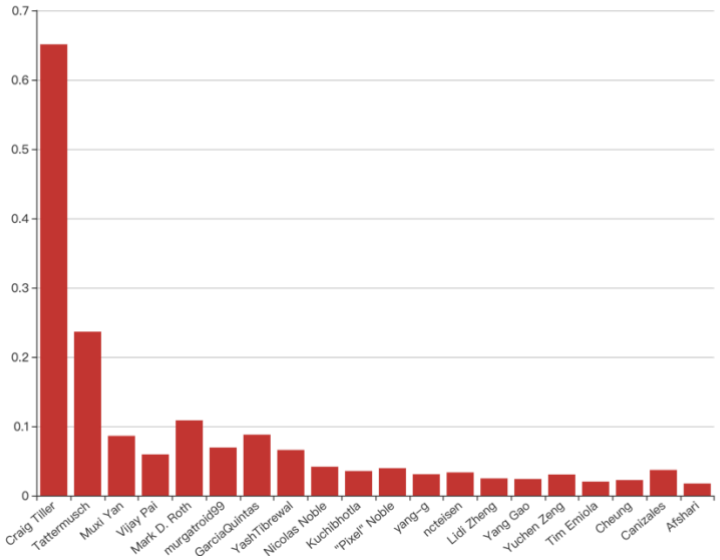

图 4.4 gRpc 贡献度 top20

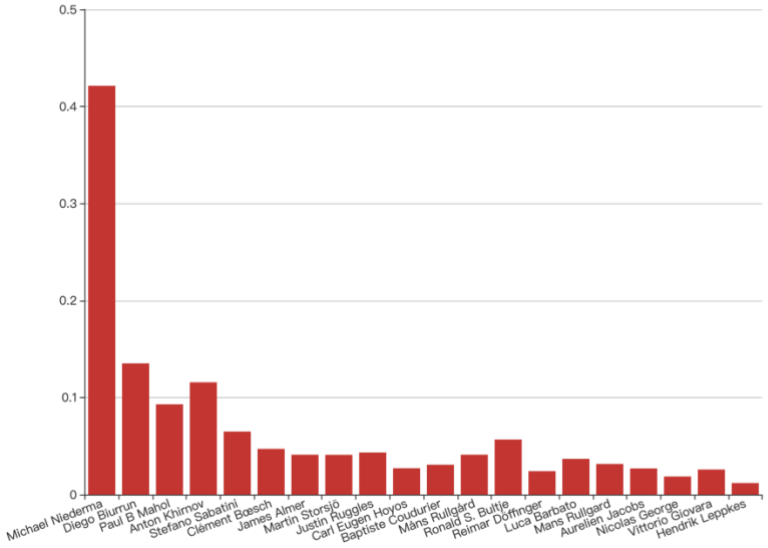

图 4.5 FFmpeg 贡献度 top20

通过对 gRPC 和 FFmpeg 高贡献度开发人员的分析，可以看到在软件项目中，极少数开发人员的贡献度很高，即整个开发过程中，由少数精英主导。在 gRPC 项目中，作者 Craig Tiller 的贡献度高达 0.65，在 FFmpeg 项目中，作者 Michael Niedermayer 的贡献度也高达 0.42，他们的贡献度超过了第二名的两倍之多。同时，对于剩余的 80%开发人员，他们对项目的贡献度不足 0.05。该现象

也直接反映了 Mockus 等人发现的在软件生命周期中存在的“二八原则”，即软件项目开发 80%的工作由不到 20%的开发人员完成。

对于高贡献度的开发人员，往往是项目前中期的开发主力。分析上述两个项目在历史提交中，按月对开发者提交次数排序，可以发现，对于 FFmpeg 项目，作者在 2001～2007 年大部分月份均名列贡献次数第一名。部分结果如下：

表 4.6 FFmpeg 项目初期贡献情况

Time	Author	Commits	Next Top5
2001-07	Fabrice Bellard	22 (91.67% of 24)	Nick Kurshev
2001-08	Fabrice Bellard	76 (86.36% of 88)	Nick Kurshev, Arpi, Juanjo
2001-09	Fabrice Bellard	34 (94.44% of 36)	Nick Kurshev, Arpi
2001-10	Michael Niedermayer	54 (55.10% of 98)	Nick Kurshev, Arpi, Zdenek Kabelac, Juanjo, Pierre Lombard
2001-11	Michael Niedermayer	52 (64.20% of 81)	Nick Kurshev, Juanjo, Arpi, Zdenek Kabelac, Pierre Lombard
2001-12	Michael Niedermayer	13 (68.42% of 19)	Arpi, Johannes Feigl
2002-01	Michael Niedermayer	40 (72.73% of 55)	Nick Kurshev, Arpi, Felix Bünemann, Zdenek Kabelac, Vladimir Dergachev

表 4.7 FFmpeg 项目中期贡献情况

Time	Author	Commits	Next Top5
2004-07	Michael Niedermayer	62 (64.58% of 96)	Roman Shaposhnik, Todd Kirby, Mike Melanson, Alex Beregszaszi, Patrice Bensoussan
2004-08	Michael Niedermayer	40 (65.57% of 61	Alex Beregszaszi, Roman Shaposhnik, Mike Melanson, Todd Kirby, Roberto Togni
2004-09	Michael Niedermayer	84 (71.19% of 118)	Alex Beregszaszi, Roman Shaposhnik, Loren Merritt, Mike Melanson, Roberto Togni
2004-10	Michael Niedermayer	75 (61.98% of 121)	Mike Melanson, Roman Shaposhnik, Michel Bardiaux, Aurelien Jacobs, Wolfram Gloger
2004-11	Michael Niedermayer	42 (67.74% of 62)	Loren Merritt, Mike Melanson, François Revol, Milan Cutka, Maarten Daniels
2004-12	Michael Niedermayer	49 (73.13% of 67)	Panagiotis Issaris, Roberto Togni, Loren Merritt, Sigbjørn Skjæret, Roine Gustafsson
2005-01	Michael Niedermayer	70 (56.00% of 125)	Roine Gustafsson, François Revol, Roberto Togni, anonymous, Loren Merritt

4.3.5 开发人员贡献时间特征

从开发者每天的活跃情况分析，开发人员更偏向于在 16:00～24:00 提交自己的代码贡献。

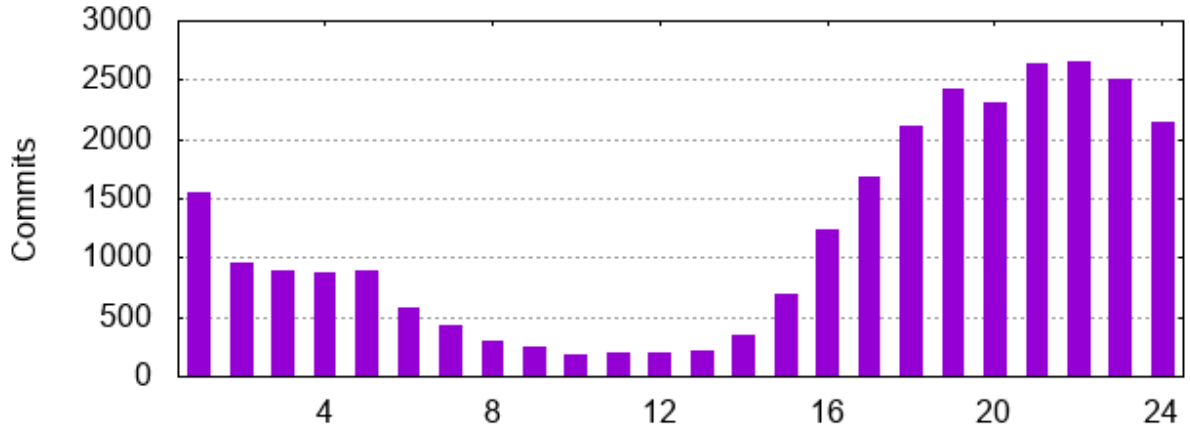

图 4.6 开发者活跃情况小时图

从周分布来来看，高贡献的开发人员的高活跃区域位于下图的右上角，即工作日的下午至凌晨。因此，综合周分布和每天的分布来看，贡献行为和开发人员的工作情况有着比较密切的关系。

Weekday	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Mon	65	82	92	70	88	71	54	37	22	22	22	21	26	52	118	281	328	405	495	484	466	505	498	394
Tue	308	160	164	165	145	106	52	42	43	41	22	34	27	53	142	228	284	417	461	368	491	500	510	445
Wed	285	146	138	150	119	77	77	38	48	36	52	39	42	76	146	224	295	377	460	470	479	526	438	391
Thu	245	168	149	141	178	79	68	47	47	35	29	31	46	69	122	241	300	386	392	420	517	475	441	396
Fri	300	181	138	134	154	89	68	65	45	26	39	37	32	60	111	181	297	365	441	395	490	478	450	363
Sat	249	123	111	140	128	76	64	26	22	13	13	20	20	29	31	50	130	71	88	99	128	98	88	99
Sun	102	99	101	70	85	73	45	44	24	15	14	15	27	14	23	38	51	91	80	75	62	79	79	54

图 4.7 开发者活跃情况周分布

针对项目按年分析贡献行为，大部分项目在中期的提交行为活跃，初期缓慢增长，末期提交行为趋于稳定。

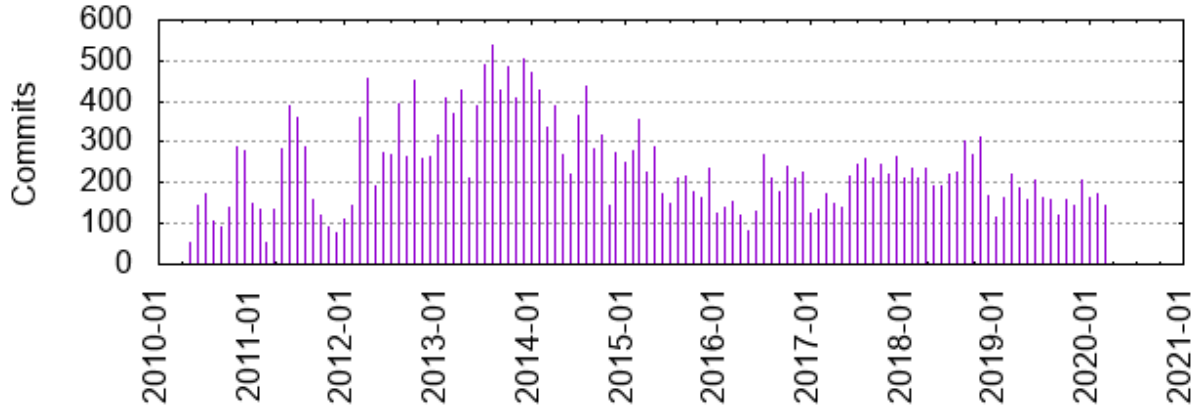

图 4.8 opencv 开发者活跃情况年分布

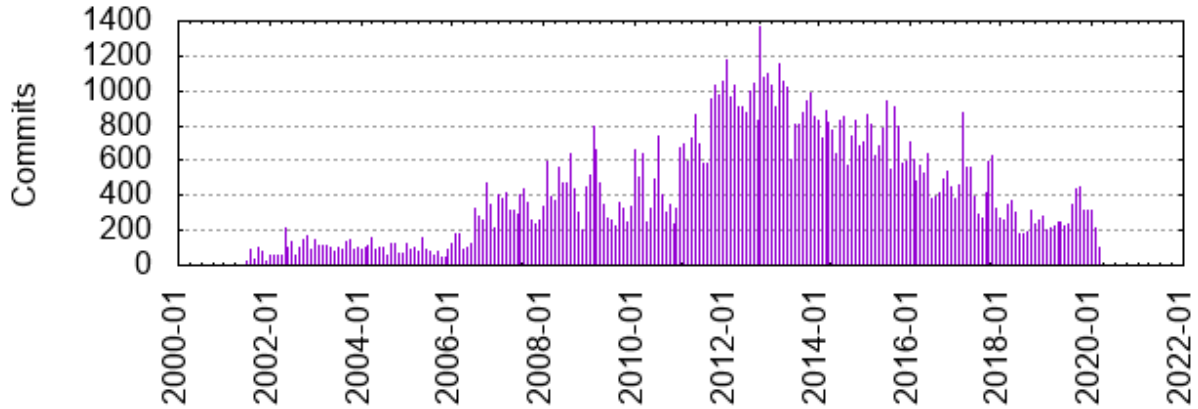

图 4.9 FFmpeg 开发者活跃情况年分布

对于单个开发者，在时间上也具有相类似的特征，即一个贡献者存在明显的活跃时期和惰性时期。以 FFmpeg 项目为例，观察作者 Michael Niedermayer 的添加代码行数曲线可知，在 2004～2008、2013～2015 年，即项目的初期、中期阶段，添加代码行数居多。观察作者 Michael Niedermayer 的贡献次数可知，在 2012～2016 年，commits 行为，commits 行为暴增。可见，添加代码对于评估项目早期贡献度更具有代表性，而贡献次数对于评估项目中期贡献度更具有代表性。

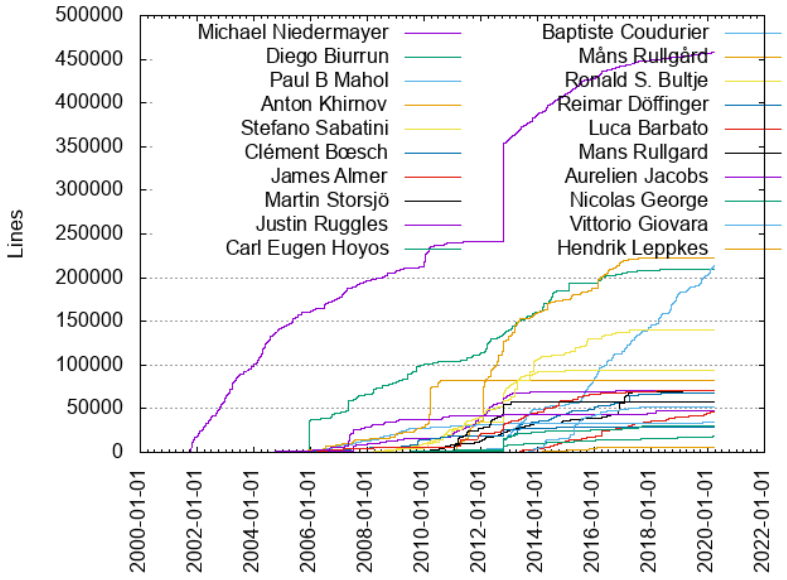

图 4.10 FFmpeg 开发者添加代码行数演变图（top20）

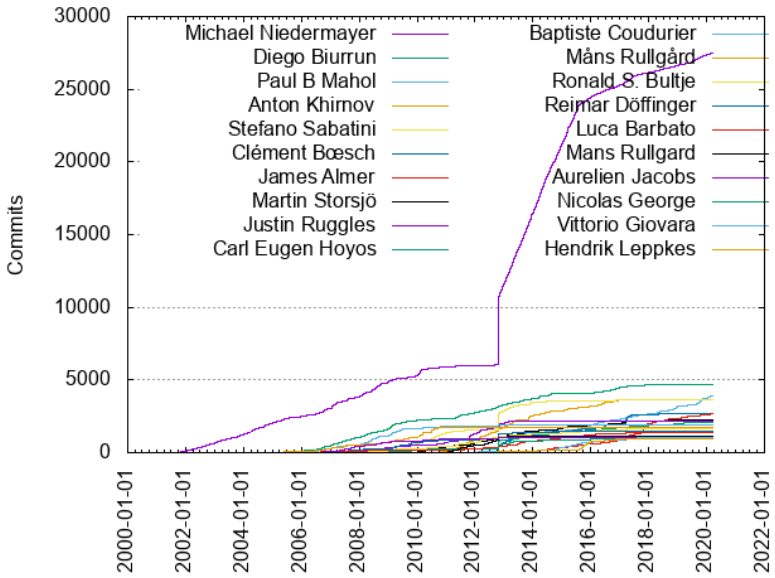

图 4.11 FFmpeg 开发者贡献次数演变图（top20）

4.3.6 开发人员贡献合作特征

统计每个项目的修改文件趋势，即被*i*个人修改过的文件数。部分数据如下图所示，其中横坐标*i*代表文件被*i*个人前后修改过，对应的纵坐标表示为此状态下的文件数。

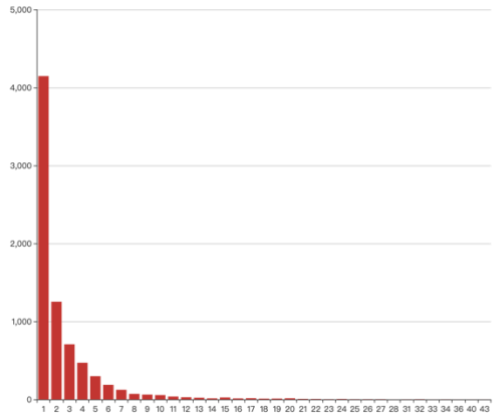

图 4.12 Flutter 修改文件趋势

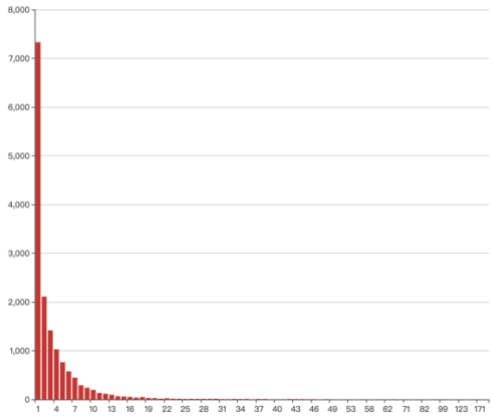

图 4.13 Pytorch 修改文件趋势

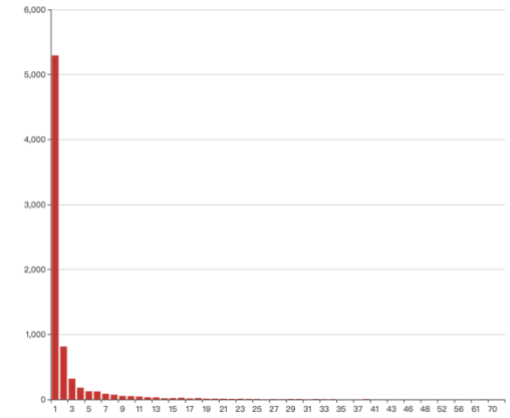

图 4.14 TiDB 修改文件趋势

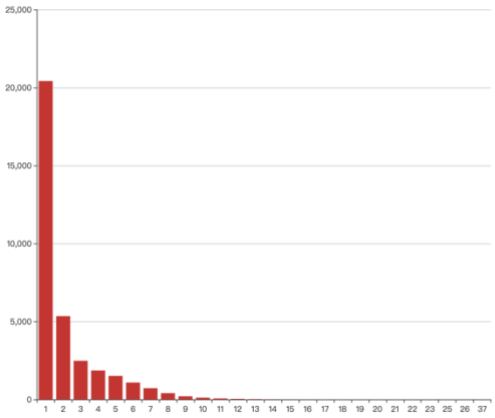

图 4.15 Spring 修改文件趋势

(1) 在开发过程中，绝大多数文件只被一个人修改过，说明开发过程中的各开发者更倾向于单独完成自己的开发任务，如新建独有文件、私有化配置等。这也符合软件开发的开闭原则，即当软件需要变化时，尽量通过扩展软件实体的行为来实现变化，而不是通过修改已有的代码来实现变化。

(2) 另外，除了项目中开发单独开发的文件外，两个开发者共同合作完成的文件个数最多。特别是在偏向于工程化的仓库中，如 Flutter、Spring，这一部分协作关系大约占总项目文件的 20%～30%，相当于单独开发文件数目的1/4。对于频繁参与协同开发的开发者，贡献度也相对较高。

(3) 在贡献较多的开发者中，存在不同形式的贡献规律，贡献偏向有不同的类型。主要分为以下两类：一是在源代码重构方面贡献较多的开发者，对于这类开发者，会接触到大量的项目文件（往往修改一些变量名称、代码逻辑、公共组件，会引起大量文件修改）；二是倾向于贡献少数核心文件的开发者，并且对维护这些核心文件有重大责任、作用。

其次，本文继续针对每个文件被 commit 提交的次数，进行统计分析。

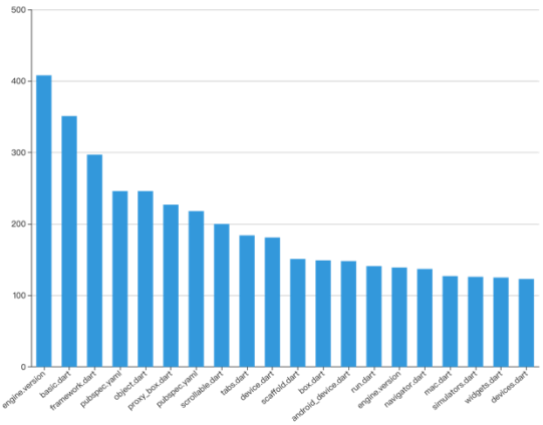

图 4.16 Flutter 文件提交统计

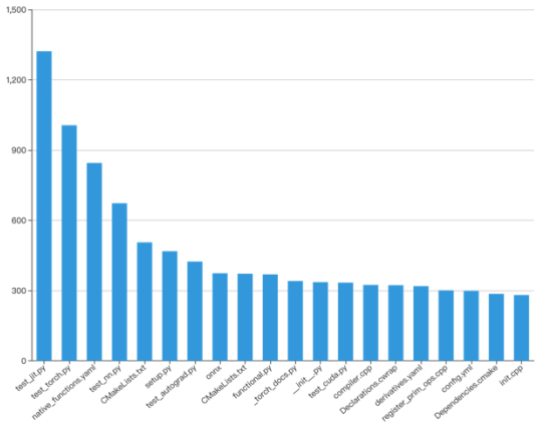

图 4.17 Pytorch 文件提交统计

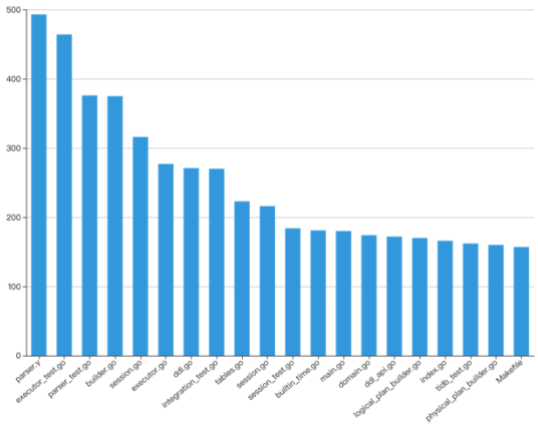

图 4.18 TiDB 文件提交统计

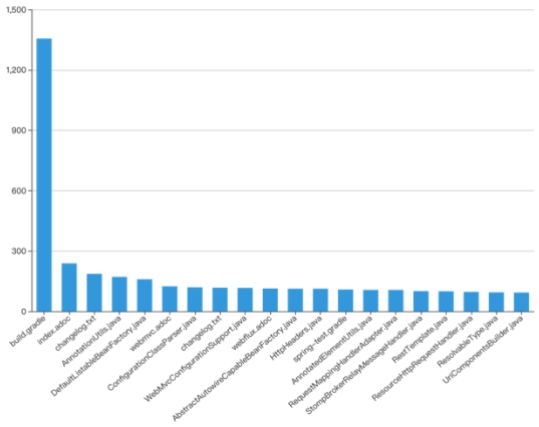

图 4.19 Spring 文件提交统计

对于开发者多次提交的文件，本文发现文件性质和项目类型有比较大的关系。对于 Spring、Flutter 这类偏向于工程的项目，在环境配置、基础框架代码（index、basic）方面的文件被提交次数居多；对于 Pytorch 这类偏向于算法层面的项目，在测试用例、安装脚本方面的文件被提交次数居多。

综合修改文件趋势和文件提交统计，可以发现少数文件被开发者频繁修改。例如 Flutter 项目中的 basic. dart 文件被 351 次提交（占比大约为10%），被 39 个人进行修改；在 TiDB 项目中，有 1.3%的文件数被超过 6 个开发人员进行修改。

4.3.7 开发人员贡献地域特征

在本节中，继续分析了开发人员的地域特征。通过对 git log 日志中每条 commit 的作者 email，以及通过 Github Api V3 对该作者进行定位（location 字段），再按国家对 commits 进行聚合统计，得到以下数据。

表 4.8 commits 数量国家排行 top13

国家	Commits 数量
United States	194594
Germany	36411
United Kingdom	34699
China	28545
France	17253
Canada	16374
India	12504
Japan	12403
Brazil	11932
Russia	11537
Australia	10267
Netherlands	8562
Spain	6894

从上表可以看到，commits 排名第一的国家是美国（United States），且 commits 数量是第二名德国（Germany）的五倍之多，之后排名的国家 commits 差距相比较小。其中中国（China）的 commits 位居第四。

对 commits 全部数据进行聚合统计，并使用 google map 可视化，结果如下图：

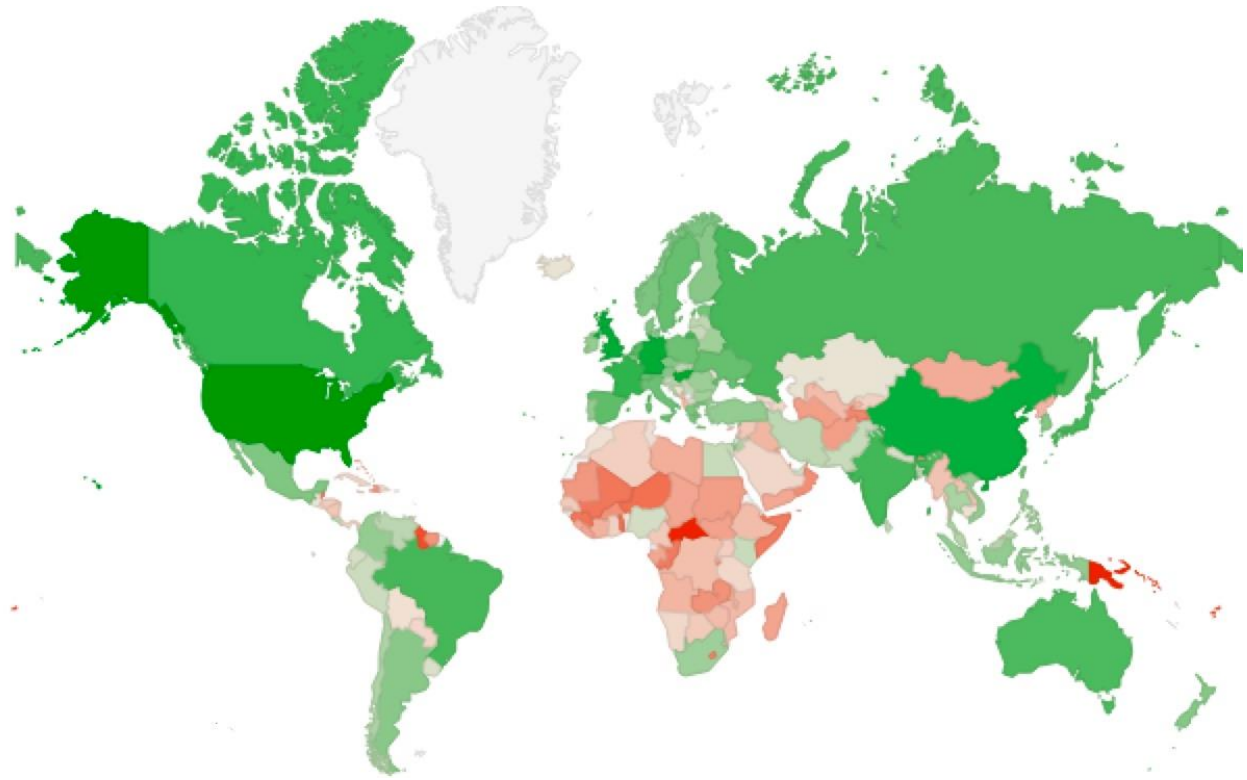

图 4. 20 commits 地域分布

针对上图可以发现，commits 较为活跃的地区是北美（尤其美国）、欧洲、东亚，北亚、中亚、南美表现则较为一般，非洲地区 commits 较为贫乏。

之后，本文继续做了一个有趣的实验：计算上述 commits 和对应国家的人口比例，并进行排序。结果如下：

表 4. 9 commits/pop 比例国家排行 top13

国家	Commits	人口（pop in 2016）	Rate(Commits/pop*10000)
Switzerland	6108	7581000	8.06
Norway	3376	5009150	6.74
New Zealand	2741	4252277	6.45
United States	194594	310232863	6.27
Luxembourg	306	497538	6.16
Sweden	5873	9555893	6.15
Iceland	183	308910	6.06
United Kingdom	34699	62348447	5.57
Denmark	2839	5484000	5.18
Netherlands	8562	16645000	5.14

Canada	16374	33679000	4.86
Finland	2515	5244000	4.80
Australia	10267	21515754	4.77

在考虑到人口因素，计算比例之后，由表 4.8 和表 4.9 对比可以发现，美国在 commits/pop 中的排名下降到了第四名，瑞士、挪威、新西兰位居前三。观察比例得分发现，除第一名瑞士之外，第二至八名的得分分布在 6.0 上下，且差距不明显，意味着这些国家在该项实验中几乎位于同一水平。

对 commits/pop 全部数据再次进行聚合统计，使用 google map 可视化，结果如下图：

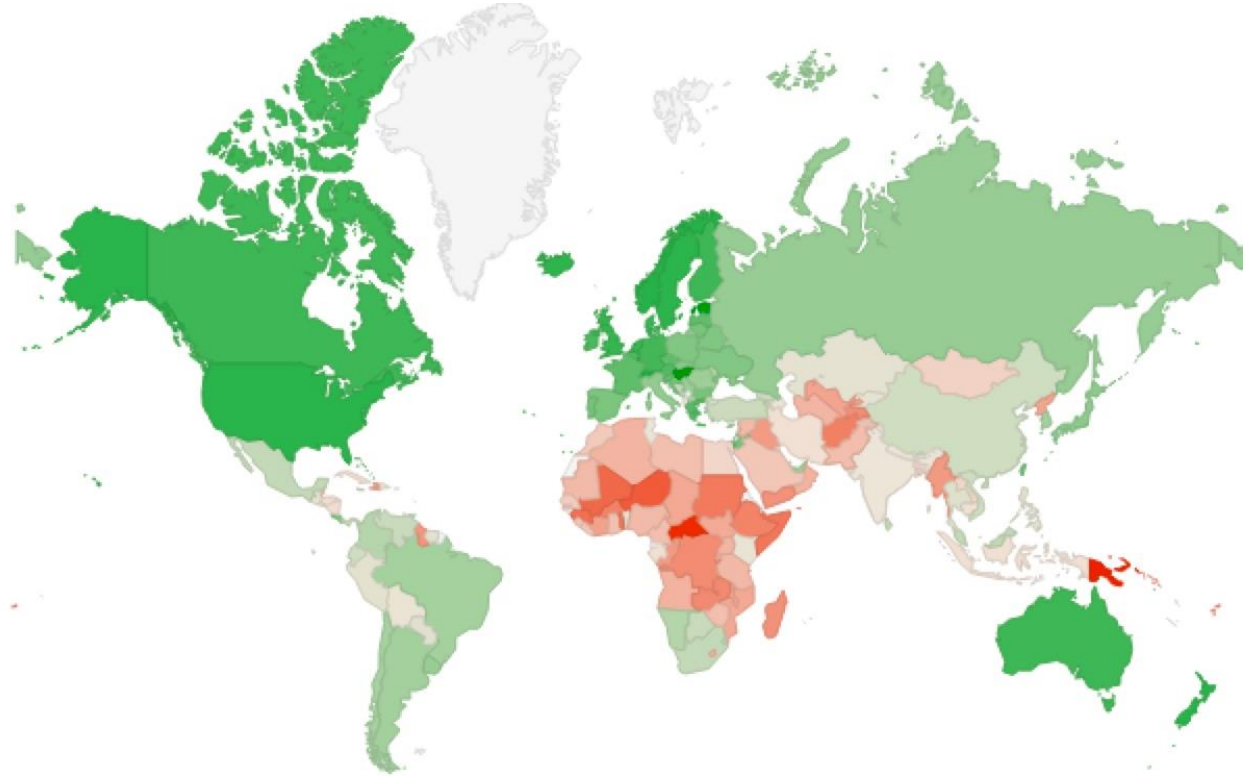

图 4.21 commits/pop 地域分布

对于 commits/pop 地域分布，并对比 commits 总量地域分布可以发现，北美、欧洲地区表现最为突出，且有所提高。亚洲、南美在考虑人口因素后，贡献能力有所下降，非洲表现依然相对较弱。

4.3.8 开发人员贡献域特征

域特征，即开发人员在提交代码时所登陆的邮箱域。根据每个项目仓库的 git log 日志，对邮箱进行提取，统计每个域的数量。其中 Pytorch、gRPC、Three.js 的统计结果如下：

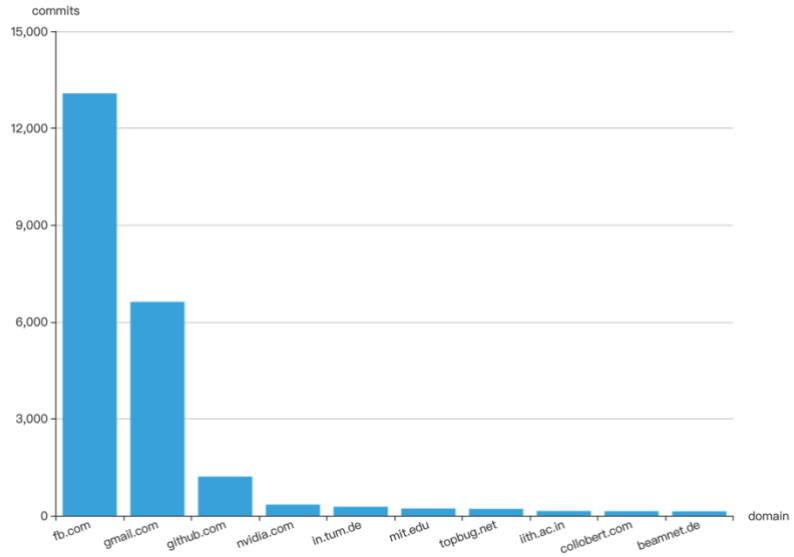

图 4.22 Pytorch 贡献域特征

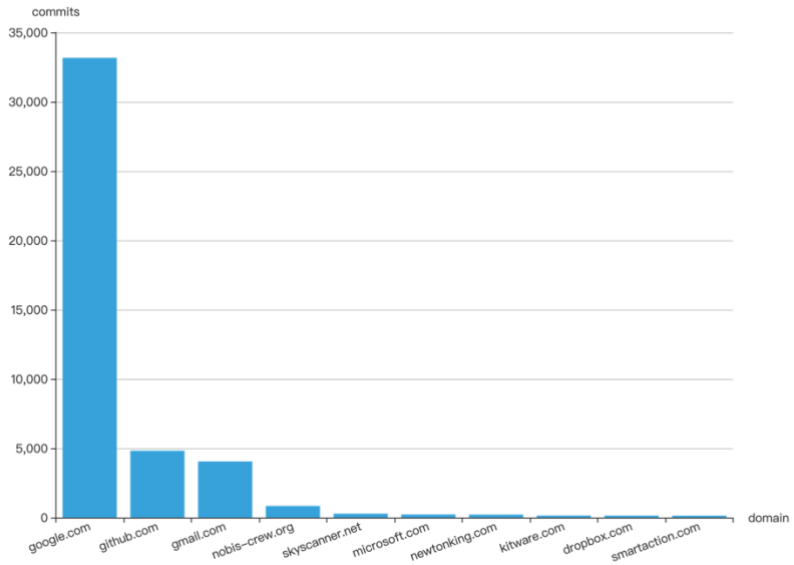

图 4.23 gRPC 贡献域特征

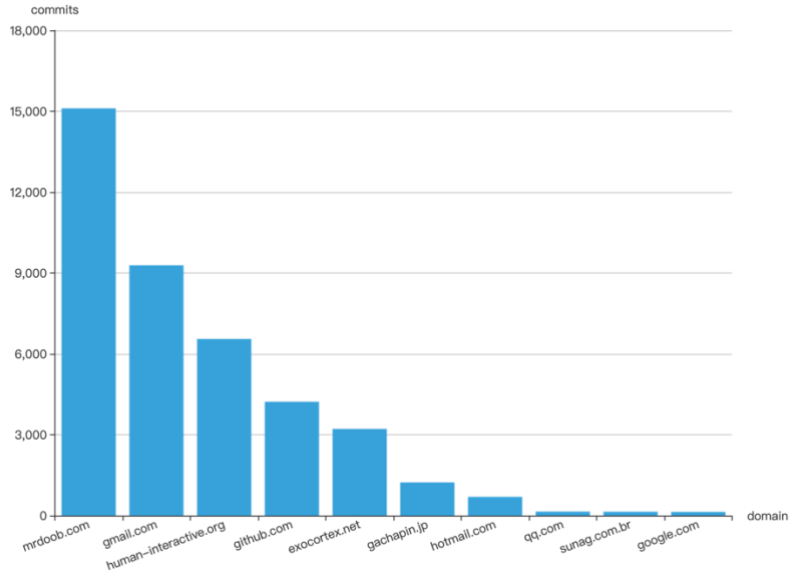

图 4.24 Three.js 贡献域特征

观察上面三个项目的统计结果，可以得到如下结论：

1、对于邮箱域总量而言，gmail.com、github.com 在个人开发者的邮箱域占比最大；

2、对于 Pytorch（来自 FaceBook 的机器学习开发框架）和 gRPC（来自 Google 的远程函数调用框架）这类由公司主导开发的软件项目，公司本身的域占据了开发人员的大多数，即 fb.com 和 google.com；对于 Three.js 这类开源程度较高，由开源社区主导开发的软件项目，域分布较为更为均匀；

3、除公司域、个人邮箱域外，还有一些学校的域名，如 mit.com，由此可见，对于开源项目，一些学校的师生也做出了较大的代码贡献。

4.4 本章小结

在本节中，结合多种图表对实验结果进行了展示，并分析结果，得到一些有价值的规律。首先，描述了本次课题所处的实验环境；之后对获取到的项目数据进行了展示，并利用相关性分析、回归分析、PCA 特征提取对数据进行了统计分析，得出相关符合实际的结论；最后，针对贡献者就合作关系、时间、地域、邮箱域等方面进行了特征分析，得出一些结论，并进行汇总。

5 总结

5.1 工作总结

本次论文主要对开源社区中的热点项目以及项目中的开发人员进行了研究，进而研究了针对开发人员的贡献评估方法和开发者行为特征、合作特征、地域特征等方面的挖掘。在研究过程中，利用了相关性分析、回归分析、主成分分析等技术对定义的观测指标进行了详细的分析；在特征分

析上，利用直方图、饼图、地域图、散点图等多种可视化技术分析的数据结果，并得到一些有意义的结论。本文的具体工作如下：

（1） 开源社区项目、开发者研究数据集的构建。数据来源为异构源，主要来自于两部分。一部分来自 Github Api，另一部分来自于本地的 Git 仓库（git log）。通过爬虫、git 命令进行数据的采集与抽取。

（2） 开源贡献者贡献度的计算以及相关度量指标研究。基于开发人员的 git 活动（commit、add、del 等），从已有的文献并结合实际经验，定义了贡献度的计算方式和有价值的观测指标。为了研究贡献度的相关因素，采用了大量相关性分析的方法，并结合主成分分析，将多个指标化为少数几个综合指标。在相关性分析方面，采用斯皮尔曼等级相关性分析，对观测变量之间的线性关系进行了研究。结果表明 commits 次数最能代表一个开发者的贡献程度。对于回归分析，发现一个开发者的活跃天数和 commits 数量关联最为紧密。

（3） 开源贡献者的特征分析。本文将特征分析主要分为时间特征、合作特征、地域特征、域特征四个部分。通过对采集的开发者信息进行聚合分析，在四个特征分析上分别给出了相应的结果，以及可视化展示。

（4） 工具原型的开发。在研究过程中，主要针对数据采集和抽取，本次研究开发了一些可执行工具。如 git2json 工具，可以将 git log 的日志信息转化为易于分析的 json 格式；github 请求包，这是一个 python 包，主要封装了一些针对本次项目的 github restful api，包括获取项目信息、用户信息、commits 信息等。

在工程领域的开发项目上，很少有研究者对开发者贡献指标和行为特征进行深入挖掘和解释说明。本文利用数据挖掘和可视化技术，研究了开发者在开源场景下的大量特征，并在研究过程中证实和发现了一些规律。

5.2 实验局限性

在本次实验中，虽然获得了一些科学的规律，但仍有一些不足之处。主要为以下几个方面：

（1） 数据集的选择上存在主观性。在数据集的选择上，主要通过 github 上的热门项目进行获取。在这一过程中，带有主观性因素。

（2） 本文的研究结果主要适用于大型项目，即 commits 次数较对、协同开发者较多的大项目。

（3） 在观测指标方面，观测指标倾向于代码指标，未考虑到非代码相关的指标，如社区活跃指标，即在开源社区的 issue、Pull Request 等社交活动。

5.3 展望

由于研究时间的限制，本文的研究课题还有继续研究和改进的方面。

（1） 在研究过程中，构建了一些针对开发者贡献研究的算法代码和可执行工具，这些代码和工具可以在后续过程中进一步抽象封装，增加自动化流程。

（2） 在数据集方面，可以统一格式规范，将数据集进行开源，便于后续的研究。

（3） 在可视化方面，可以通过 web 前端技术，进行可交互式的展示和分析。

参考文献

[1] Milewicz R, Pinto G, Rodeghero P. Characterizing the roles of contributors in open-source scientific software projects[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, 2019: 421-432.

[2] 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. 2008: 129-132.

[3] Lima J, Treude C, Figueira Filho F, et al. Assessing developer contribution with repository mining-based metrics[C]//2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2015: 536-540.

[4] Zhou M, Mockus A. What make long term contributors: Willingness and opportunity in OSS community[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 518-528.

[5] Nagwani N K, Verma S. Rank-me: A java tool for ranking team members in software bug repositories[J]. 2012.

[6] Schuler D, Zimmermann T. Mining usage expertise from version archives[C]//Proceedings of the 2008 international working conference on Mining software repositories. 2008: 121-124.

[7] Ben X, Beijun S, Weicheng Y. Mining developer contribution in open source software using visualization techniques[C]//2013 Third International Conference on Intelligent System Design and Engineering Applications. IEEE, 2013: 934-937.

[8] Linstead E, Rigor P, Bajracharya S, et al. Mining eclipse developer contributions via author-topic models[C]//Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007). IEEE, 2007: 30-30.

[9] Lima J, Treude C, Figueira Filho F, et al. Assessing developer contribution with repository mining-based metrics[C]//2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2015: 536-540.

[10] Gousios G, Spinellis D. Alitheia core: An extensible software quality monitoring platform[C]//2009 IEEE 31st International Conference on Software Engineering. IEEE, 2009: 579-582.

[11] Amor J J, Robles G, Gonzalez-Barahona J M. Effort estimation by characterizing developer activity[C]//Proceedings of the 2006 international workshop on Economics driven software engineering research. 2006: 3-6.

[12] Honsel V, Honsel D, Herbold S, et al. Mining software dependency networks for agent-based simulation of software evolution[C]//2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). IEEE, 2015: 102-108.

[13] Zhou M, Mockus A. What make long term contributors: Willingness and opportunity in OSS community[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 518-528.

[14] Baysal O, Godfrey M W, Cohen R. A bug you like: A framework for automated assignment of bugs[C]//2009 IEEE 17th International Conference on Program Comprehension. IEEE, 2009: 297-298.

[15] Honsel V, Herbold S, Grabowski J. Hidden markov models for the prediction of developer involvement dynamics and workload[C]//Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering. 2016: 1-10.

[16] Hong Q, Kim S, Cheung S C, et al. Understanding a developer social network and its evolution[C]//2011 27th IEEE international conference on software maintenance (ICSM). IEEE, 2011: 323-332.

[17] Long J. Understanding the Role of Core Developers in Open Source Software Development[J]. Journal of Information, Information Technology & Organizations, 2006, 1.

[18] Honsel V. Statistical learning and software mining for agent based simulation of software evolution[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 2: 863-866.

[19] Weissgerber P, Pohl M, Burch M. Visual data mining in software archives to detect how developers work together[C]//Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007). IEEE, 2007: 9-9.

[20] Shrestha N, Barik T, Parnin C. It's Like Python But: Towards Supporting Transfer of Programming Language Knowledge[C]//2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2018: 177-185.

[21] Pinto G, Wiese I, Dias L F. How do scientists develop scientific software? an external replication[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 582-591.

[22] Ma Y, Bogart C, Amreen S, et al. World of code: An infrastructure for mining the universe of open source vcs data[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, 2019: 143-154.

[23] Bangerth W, Heister T. What makes computational open source software libraries successful?[J]. Computational Science & Discovery, 2013, 6(1): 015010.

[24] Baltes S, Dumani L, Treude C, et al. Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts[C]//Proceedings of the 15th international conference on mining software repositories. 2018: 319-330.

[25] Lin B, Robles G, Serebrenik A. Developer turnover in global, industrial open source projects: Insights from applying survival analysis[C]//2017 IEEE 12th International Conference on Global Software Engineering (ICGSE). IEEE, 2017: 66-75.

[26] 廖志芳,李斯江,贺大禹,赵本洪. GitHub 开源软件开发过程中关键用户行为分析[J]. 小型微型计算机系统,2019,40(01):164-168.

[27] 甘谊昂. 开源软件社区开发者的贡献评估方法研究[D]. 国防科学技术大学, 2016.

[28] 李存燕,洪玫. Github 中开发人员的行为特征分析[J]. 计算机科学,2019,46(02):152-158.

[29] 袁霖,王怀民,尹刚,史殿习,李翔. 开源环境下开发人员行为特征挖掘与分析[J]. 计算机学报,2010,33(10):1909-1918.

[30] 徐奔. 开源软件开发人员行为特征的可视化挖掘[D].上海交通大学,2013.

[31] 张莉,蒲梦媛,刘奕君,田家豪,岳涛,蒋竞.对软件工程中经验研究的调查[J].软件学报,2018,29(05):1422-1450.

[32] Wohlin C, Höst M, Henningsson K. Empirical research methods in software engineering[M]//Empirical methods and studies in software engineering. Springer, Berlin, Heidelberg, 2003: 7-23.

[33] Basili V R, Selby R W, Hutchens D H. Experimentation in software engineering[J]. IEEE Transactions on software engineering, 1986 (7): 733-743.

[34] Dyba T, Kitchenham B A, Jorgensen M. Evidence-based software engineering for practitioners[J]. IEEE software, 2005, 22(1): 58-65.

[35] Raja U. All complaints are not created equal: text analysis of open source software defect reports[J]. Empirical Software Engineering, 2013, 18(1): 117-138.

声　明

本人声明所呈交本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得四川大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

本学位论文成果是本人在四川大学读书期间在导师指导下取得的，论文成果归四川大学所有，特此声明。

学位论文作者（签名）_____

论文指导教师（签名）_____

年　月　日

致 谢

时间如水，总是无言。我在四川大学度过了我人生中最难忘的四年，这四年中，走的艰辛却也收获颇丰，我的成长源于自身的努力，伴随着老师们的指点，更离不开四川大学的栽培和熏陶。

首先，最要感谢的是我的导师杨秋辉老师。在进行毕设的这几个月中，我得到了杨老师无微不至的关心和教导，杨老师知识渊博，视野宽阔，对我严格要求，耐心地为我答疑解惑，我向杨老师表示深深地谢意；

其次就是为我提供帮助，和我一起并肩作战陪伴我的同学们；

最后要向教导我，栽培我的爸爸妈妈表示感谢！

毕设设计的完成，为我大学的学习生涯画上了一个圆满的句号。我在四川大学度过了我人生中最难忘的四年，他让我受益匪浅，让我适应了从一个学生到长大成人的转变，让我成长，也留住了我人生中最美好的四年！