Predicting Long-time Contributors for GitHub Projects using Machine Learning

by Vijaya Kumar Eluri

B.Tech in Computer Science and Engineering, June 1993, Andhra University, India
M.Tech in Computer Science, March 1996, Indian Statistical Institute, India
MBA, December 2017, University of Illinois at Urbana-Champaign, USA
M.S in Supply Chain Management, January 2019, Rutgers Business School, USA

A Dissertation submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Philosophy

August 31, 2021

Dissertation directed by

Shahram Sarkani
Professor of Engineering Management and Systems Engineering

Thomas Mazzuchi
Professor of Engineering Management and Systems Engineering & of Decision Sciences

The School of Engineering and Applied Science of The George Washington University certifies that Vijaya Kumar Eluri has passed the Final Examination for the degree of Doctor of Philosophy as of July 28, 2021. This is the final and approved form of the dissertation.

Predicting Long-time Contributors for GitHub Projects using Machine Learning

Vijaya Kumar Eluri

Dissertation Research Committee:

Shahram Sarkani, Professor of Engineering Management and Systems Engineering, Dissertation Co-Director

Thomas Mazzuchi, Professor of Engineering Management and Systems Engineering & of Decision Sciences, Dissertation Co-Director

Amir Etemadi, Associate Professor of Engineering and Applied Science, Committee Member

Thomas Holzer, Professional Lecturer of Engineering Management and Systems Engineering, Committee Member

Joseph P. Blackford, Professional Lecturer of Engineering Management and Systems Engineering, Committee Member

© Copyright 2021 by Vijaya Kumar Eluri
All rights reserved

Dedication

I dedicate this dissertation to my beloved wife, Vani. Thank you for everything you do.

Acknowledgments

I am grateful to Dr. Thomas A. Mazzuchi and Dr. Shahram Sarkani, Professors of Engineering Management and Systems Engineering at George Washington University, who have guided me throughout this research. I want to thank them for challenging pre-existing conceptions, encouraging new ways of thinking, exploring options, and helping me refine this research work.

To all my Ph.D. program instructors - Dr. Timothy J. Eveleigh, Dr. W M Kroshl, Dr. Amir Etemadi, Dr. Scott Littlefield, Dr. J.P. Blackford, Dr. John Fossaceca, and Dr. Thomas Holzer. Thank you for exceptional quality education.

To my former supervisors: I want to acknowledge my former university supervisor, Prof. Nikhil R. Pal at Electronics and Communication Sciences Unit (ECSU), Indian Statistical Institute, for encouraging me never to give up. I want to thank my former professional boss, Shehzad Mevawalla - Head of Alexa Speech Recognition at Amazon - for instilling self-belief in me.

To my parents, Sree Ramulu and Sarojini Devi: Thank you for believing in education and encouraging me to pursue academics from a young age.

To my in-laws, Radhakrishna and Ratnamala: Thank you for helping whenever we needed help.

To my family, Uma, Prasad, Nirmala, Srinivas, Ramu, Madhavi, Ramesh, Sailaja, Prasad, and Maneka: Thank you for your understanding and helping in various ways while I was busy with work and education.

To my son Pranav and his cousins - Abhi, Akhil, Lohit, Sai, Sanju, Nilesh, Sruti, and Umesh: Thank you for reminding me to be curious and playful.

To my childhood friends Jagan, Padma, and Patnaik: Thank you for all the memories!

Most importantly to my SAI family: Thank you for your support.

Abstract

Predicting Long-time Contributors for GitHub Projects using Machine Learning

Organizations typically develop software systems using non-developmental items and commercial off-the-shelf software. Many organizations are changing the way they create, capture, consume, and commercialize software (Ihara, Monden, and Matsumoto, 2014) by increasingly adopting Open-source Software (OSS) projects (Eluri, Sarkani, and Mazzuchi, 2019). However, many OSS projects do not survive, as the survival of OSS projects depends mainly on retaining new contributors (Eluri, Sarkani, and Mazzuchi, 2019).

While organizations driven by their business needs support some OSS projects, most OSS contributors are volunteers who contribute their work for free (Benbya and Belbaly, 2010). While many join OSS projects, only a few contribute to an OSS project for a considerably long time. A Long-time Contributors (LTC) is defined as a contributor who joins a project and continues to contribute for more than T years; T is generally set to 1, 2, and 3 years (Zhou and Mockus, 2011 and Zhou and Mockus, 2015). LTCs often contribute more code than non-LTCs (Koch and Schneider, 2002 and Mockus, Fielding, and Herbsleb, 2002).

Most new contributors abandon a project without becoming LTCs. The data in this research (Eluri, Mazzuchi, and Sarkani, 2021) shows that 98% of new contributors leave a project before three years. Various factors affect whether a new contributor becomes an LTC, including the new contributor's experience, expertise, the project's maturity, working environment, documentation, and task difficulty. Identifying factors that predict potential LTCs can enable project owners to gain insight into what matters to contributors and take action to retain new contributors for a long time. These actions could include mentoring, quickly responding to questions, providing timely code reviews, and merging contributions.

This research investigates effective predictability of new contributors to OSS repositories becoming LTC based on repository and contributor meta-data collected from OSS

repositories. Compared to state-of-the-art models (Bao et al., 2019), the models built in this research use less than 50% features and produce better results. In 10-fold cross-validation, the precision, recall, F1-score, Mathew's Correlation Coefficient (MCC), and Area Under Curve (AUC) of the best state-of-the-art model are 0.546, 0.041, 0.075, 0.1446, and 0.908, respectively (Bao et al., 2019).

Table of Contents

Dedication iv

Acknowledgments v

Abstract vi

List of Figures xi

List of Tables xii

List of Acronyms xiv

Chapter 1: Introduction 1
- 1.1 Background . . . 1
- 1.2 Research Problem . . . 1
- 1.3 Research Overview, Objectives, Questions, and Hypotheses . . . 3
- 1.4 Overview of Dissertation . . . 5

Chapter 2: Literature Review 6
- 2.1 Open-source Software . . . 6
- 2.2 Contributor Motivation . . . 6
- 2.3 Contributor Attraction . . . 13
- 2.4 Contributor Acceptance . . . 17
- 2.5 Contributor Barriers . . . 20
- 2.6 Contributor Turnover . . . 21
- 2.7 Long-time Contributor Prediction . . . 25
- 2.8 Closely Related Work . . . 28

Chapter 3: Methodology 36
- 3.1 Data Source . . . 36
- 3.2 Dataset . . . 40
- 3.3 Features . . . 43
  - 3.3.1 Feature Set 1 . . . 43
  - 3.3.2 Feature Set 2 . . . 51
  - 3.3.3 Feature Set 3 (with Feature Selection) . . . 57
- 3.4 Prediction Models . . . 60
  - 3.4.1 naive Bayes (NB) . . . 64
  - 3.4.2 Support Vector Machines (SVM) . . . 64
  - 3.4.3 k-Nearest Neighbor (kNN) . . . 68
  - 3.4.4 Logistic Regression (LR) . . . 68
  - 3.4.5 Decision Tree (DT) . . . 68
  - 3.4.6 Random Forest (RF) . . . 69
- 3.5 Evaluation Metrics . . . 70

3.5.1 LTC Precision . . . 71
3.5.2 LTC Recall . . . 71
3.5.3 non-LTC Precision . . . 71
3.5.4 non-LTC Recall . . . 72
3.5.5 F1-score . . . 72
3.5.6 Accuracy . . . 72
3.5.7 Matthews Correlation Coefficient (MCC) . . . 72
3.5.8 Area Under the Curve (AUC) . . . 73
3.6 Model Validation Method . . . 73
3.6.1 K-Fold Cross-Validation (KFCV) . . . 73

Chapter 4: Experiments and Results **76**
4.1 Experimental Results - Research Objective 1 . . . 76
4.1.1 RQ 1.1: Can past history effectively determine an LTC? . . . 76
4.1.2 RQ 1.2: How effective is the past history approach compared to the state-of-the-art approach? . . . 79
4.1.3 RQ 1.3: Which features are the most important in identifying LTCs using past history approach? . . . 82
4.2 Experimental Results - Research Objective 2 . . . 84
4.2.1 RQ 2.1: Can onboarding activity effectively determine an LTC? 87
4.2.2 RQ 2.2: How effective is the onboarding activity approach in determining an LTC compared to the state-of-the-art approach? 89
4.2.3 RQ 2.3: How effective is the feature selection method? . . . 90
4.2.4 RQ 2.4: Which features are the most important in identifying LTCs using onboarding activity? . . . 92
4.3 Threats to Validity . . . 95
4.3.1 Internal Validity . . . 95
4.3.2 External Validity . . . 95
4.3.3 Construct Validity . . . 96

Chapter 5: Discussion **97**
5.1 Evaluating the Validity of the Hypotheses . . . 97
5.2 Recommendations . . . 98
5.2.1 Automate assignment of integrating newcomer contributions. . 98
5.2.2 Allocate resources to evaluate a new contributor and provide core member privilege. . . . 99
5.2.3 Allocate resources for documenting various ways to contribute. 99
5.2.4 Allocate resources to merge and close pull requests from new contributors in the first three months. . . . 102
5.2.5 Allocate resources to onboard newcomers in the first month. . . 102
5.3 Conclusion . . . 103
5.4 Future Work . . . 104

Appendix A: Appendix **114**
A.1 Queries . . . 114

A.1.1 r_Language . . . 114
A.1.2 r_Age . . . 114
A.1.3 r_Number_of_Watchers . . . 114
A.1.4 r_has_Dependents . . . 115
A.1.5 r_Number_of_Core_Members . . . 116
A.1.6 r_Number_of_Contributors . . . 116
A.1.7 r_Min_All_Contributor_Commits . . . 117
A.1.8 r_Max_All_Contributor_Commits . . . 118
A.1.9 r_Min_All_Contributor_Stay . . . 118
A.1.10 r_Max_All_Contributor_Stay . . . 119
A.1.11 r_has_Contribution_Invitation . . . 119
A.1.12 r_has_Contribution_Rules . . . 120
A.1.13 r_Readability_Kincaid_Score . . . 120
A.1.14 c_is_Fake . . . 121
A.1.15 c_is_Deleted . . . 121
A.1.16 c_Age . . . 121
A.1.17 c_is_Core_Member . . . 122
A.1.18 c_Own_Repos_Lang . . . 122
A.1.19 c_Watch_Repos_Lang . . . 122
A.1.20 c_Contribute_Repos_Lang . . . 123
A.1.21 c_History_Commits_Lang . . . 123
A.1.22 c_History_Pull_Request_Lang . . . 124
A.1.23 c_History_Issues_Lang . . . 124
A.1.24 c_History_Followers . . . 125
A.1.25 c_History_Following . . . 125
A.1.26 c_Number_of_Repos . . . 126
A.1.27 c_Min_All_Repos_Commits . . . 126
A.1.28 c_Max_All_Repos_Commits . . . 127
A.1.29 c_Min_All_Repos_Stay . . . 127
A.1.30 c_Max_All_Repos_Stay . . . 128

List of Figures

1.1 OSS Projects on GitHub. 2

2.1 OSS contributor role classification proposed by Ye and Kishida, (2003). . . . 9
2.2 A Multi-Theoretical framework of contributor motivation in OSS proposed by Benbya and Belbaly, (2010). 12
2.3 A socialization and contributor performance conceptual model in OSS proposed by Carillo, Huff, and Chawner, (2017). 15
2.4 Research model of perceived turnover in OSS projects proposed by Yu, Benlian, and Hess, (2012). 23
2.5 Relationship among concepts, dimensions, and measures proposed by Zhou and Mockus, (2015). 26
2.6 Data collection process used by Bao et al., (2019). 30

3.1 Methodology map for research objective 1. 37
3.2 Methodology map for research objective 2. 37
3.3 GHTorrent MySQL database schema from Gousios, (2013). 39
3.4 Data collection process used in this research. 41
3.5 The feature selection process used in this research. 59
3.6 ML model building methodology. 65
3.7 Partitioning the dataset into k-folds, where k = 10. Blue partition represents validation sample. Yellow partition represents training sample. 74
3.8 Code to partitioning the dataset into 10-folds and train the RF model. 75

4.1 Boxplot of contributor age (days) for LTC and non-LTC groups. 85
4.2 Boxplot of repository age (days) for LTC and non-LTC groups. 86

List of Tables

2.1 Criteria for software to be open-source (OSS website, 2020). 7
2.2 Features used in Carillo, Huff, and Chawner, (2017) study. 16
2.3 Features used in Bird et al., (2007) study. 18
2.4 Adaptation of Herzberg, (1966) hygiene features to OSS projects in Yu, Benlian, and Hess, (2012) study. 22
2.5 Features used in Sharma, Hulland, and Daniel, (2012) study. 24
2.6 Features for participant *i* (Zhou and Mockus, 2015). 27
2.7 Features used in Decan et al., (2020) study. 28
2.8 Features used in Bao et al., (2019) study. 31

3.1 Description of database entities used in this study and Bao et al., (2019). . . . 40
3.2 Repositories with repeated/duplicate URLs in Bao et al., (2019) dataset. . . . 42
3.3 Statistics of selected LTC and non-LTC data. 43
3.4 Dataset 1: Structure of the data used in this study. 43
3.5 Feature Set 1: Repository features (r_) used in this study. 44
3.6 Feature Set 1: Contributor features (c_) used in this study. 45
3.7 Feature Set 1: Descriptive statistics for repository and contributor features. . 46
3.8 Dataset 2: Structure of the data used in this study. 51
3.9 Feature Set 2: Developer Profile (DP) features and Repository Profile (RP) features. 52
3.10 Feature Set 2: Developer Activity (DA) features, Repository Activity (RA) features, and Collaboration Network (CN) features. 53
3.11 Dataset 3: Structure of the data used in this study. 58
3.12 Feature Set 3: Selected 55 features for DP and RP dimensions. 61
3.13 Feature Set 3: Selected 55 features for DA, RA, and CN dimensions. 62
3.14 Feature Set 3: Descriptive statistics of the selected 55 features. 63
3.15 Prediction models with different parameters. 70

4.1 Evaluation metrics for the proposed past history approach for all models. . . 77
4.2 Cliff's delta and p-values comparing past history models with RF model. . . 78
4.3 Evaluation metrics for the proposed past history RF model and Bao et al., (2019) RF model. 80
4.4 Percentage gain in evaluation metrics for the proposed past history RF model and Bao et al., (2019) RF model. 80
4.5 Cliff's delta and p-values comparing the proposed past history and Bao et al., (2019) RF models. 80
4.6 Top 10 important features according to the Scott-Knott ESD test for year 1. . 84
4.7 Top 10 important features according to the Scott-Knott ESD test for year 2. . 84
4.8 Top 10 important features according to the Scott-Knott ESD test for year 3. . 85
4.9 Top 10 languages with the highest number of LTCs and total contributors. . . 86
4.10 Evaluation metrics for the proposed onboarding activity approach for all models. 88

4.11 Cliff's delta and p-values comparing onboarding activity models with Bao et al., (2019) RF model. 89
4.12 Evaluation metrics for the proposed onboarding activity RF model and Bao et al., (2019) RF model. 90
4.13 Percentage gain in evaluation metrics for the proposed onboarding activity RF model and Bao et al., (2019) RF model. 91
4.14 Cliff's delta and p-values comparing the proposed onboarding activity and Bao et al., (2019) RF models. 91
4.15 Evaluation metrics for the proposed onboarding RF models - 173 and 55 features RF models. 92
4.16 Cliff's delta and p-values comparing the proposed onboarding activity RF models - 173 feature RF model and 55 features RF model. 92
4.17 Top 10 important features according to the Scott-Knott ESD test for year 1. . 94
4.18 Top 10 important features according to the Scott-Knott ESD test for year 2. . 94
4.19 Top 10 important features according to the Scott-Knott ESD test for year 3. . 95

5.1 Oldest 10 repository details. 101

List of Acronyms

API Application Programming Interface

AUC Area Under Curve

CCB Community Citizenship Behaviors

CN Collaboration Network

DA Developer Activity

DP Developer Profile

DT Decision Tree

EKM Extended Klandermans Model

KFCV K- Fold Cross Validation

kNN k Nearest Neighbours

LR Logistic Regression

LTC Long-time Contributors

MCC Mathew's Correlation Coefficient

ML Machine Learning

NB naive Bayes

OSI Open-Source Initiative

OSS Open-source Software

RA Repository Activity

REST Representational State Transfer

RF Random Forest

RP Repository Profile

SVM Support Vector Machine

TP Task Performance

VIST Valence Instrumentality Self-efficacy and Trust

Chapter 1: Introduction

1.1 Background

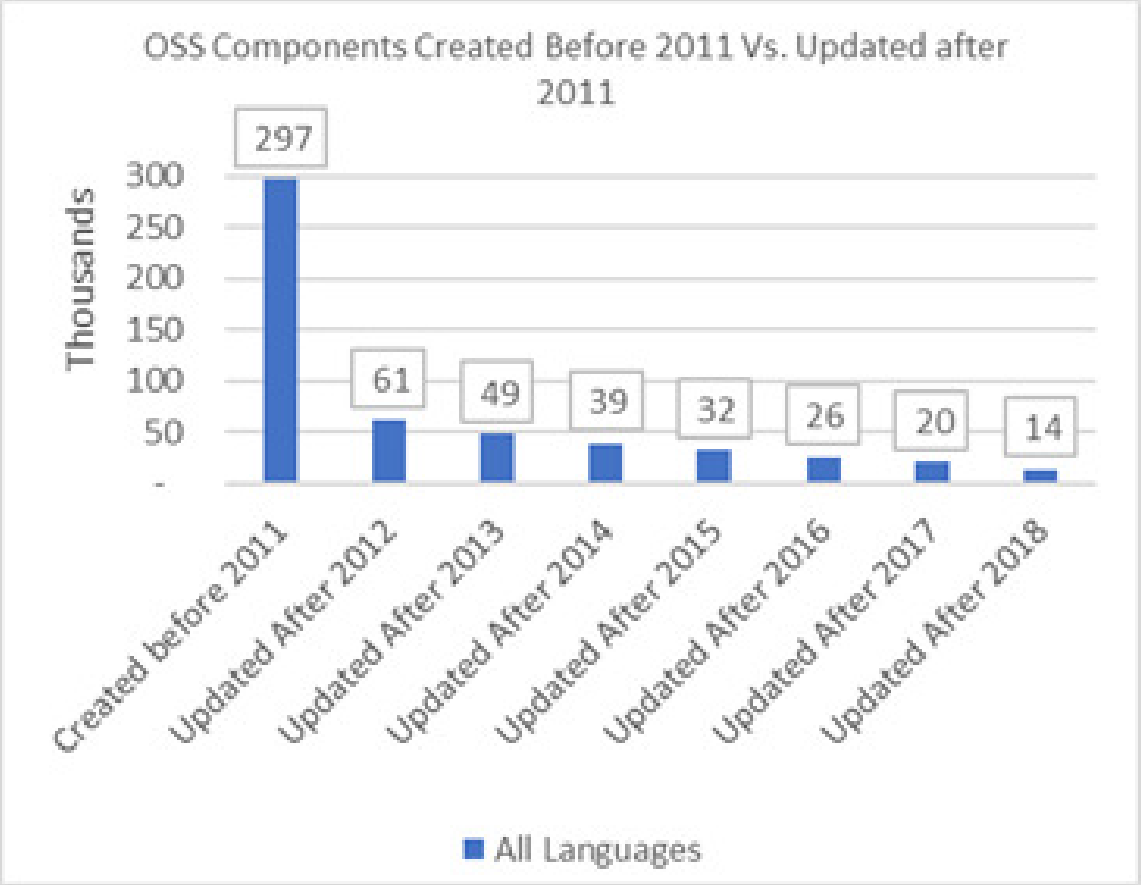
OSS plays a significant role in our daily lives without our knowledge. We use computer systems that depend on thousands of OSS projects globally, such as the Linux operating system, Apache web server, Mozilla Firefox web browser.

Organizations typically develop software systems using non-developmental items and commercial off-the-shelf software. Many organizations are changing the way they create, capture, consume, and commercialize software (Ihara, Monden, and Matsumoto, 2014) by increasingly adopting OSS (Eluri, Sarkani, and Mazzuchi, 2019). According to Octoverse, (2019), 70% of Fortune 50 companies have made contributions to OSS projects, with firms such as Microsoft and Facebook being the most active on GitHub. As pointed out in Eluri, Sarkani, and Mazzuchi, (2019), OSS projects are gaining significant traction, resulting in many applications containing significantly more open-source code than proprietary code (Synopsis, 2018). Software composition analysis report (Synopsis, 2018) conducted by the Black Duck audit team discovered OSS projects in 96% of scanned apps and 257 OSS components per codebase (75% growth between 2017 and 2018).

1.2 Research Problem

OSS project usage continues to experience dramatic growth; however, many OSS projects do not survive, making dependency on OSS projects risky. Eluri, Sarkani, and Mazzuchi, (2019) have analyzed OSS projects and found that OSS projects go out of support rapidly, as displayed in Figure 1.1. The figure shows that 297,000 OSS projects were created before 2011, and contributors updated only 14,000 of these 297,000 thousand projects after 2018.

Figure 1.1. OSS Projects on GitHub.



Data is obtained by querying GitHub.
Query `https://api.github.com/search/repositories?q=created:<=2011-01-01` provides data for projects created before 2011.
Query `https://api.github.com/search/repositories?q=pushed:>2012-01-01+created:<=2011-01-01` provides data for projects created before 2011 and updated after 2012 (Replacing 2012 with 2013 to 2018 in this query gives the component updated data for the rest of the years).

Eluri, Sarkani, and Mazzuchi, (2019) built a Multi-Layer Perceptron ML model using a data set consisting of 449 OSS projects. Each project was tagged as active or inactive (a project is considered inactive if it did not have activity for more than one year; otherwise, it was considered active). Eluri, Sarkani, and Mazzuchi, (2019) extracted five features for each project: Cognitive Complexity, Cyclometric Complexity, Number of Subscribers, Number of Open Issues, and Number of Contributors. Next, the data set was split into a training set consisting of 352 (78.4%) records and a testing set consisting of 97 (21.6%) records as a hold out for final testing. The Multi-Layer Perceptron ML model achieved 81.44% validation accuracy for survivability prediction and has shown that the OSS projects' survival depends on the number of contributors.

Most new contributors abandon a project without becoming LTCs. The data used in this research shows that 98% of new contributors leave a project before three years. Various factors affect whether a new contributor becomes an LTC, including the new contributor's experience, expertise, the project's maturity, working environment, documentation, and task difficulty. Identifying factors that predict potential LTCs can help OSS project owners utilize limited resources to retain new contributors. There is a need to investigate the factors that help effectively predict new contributors to OSS projects becoming long-time contributors.

1.3 Research Overview, Objectives, Questions, and Hypotheses

Contributor activities (code contributions, issue submissions, comments) and repository activities (contribution by all contributors) are critical factors to improve new contributor retention, enabling researchers to predict who will become an LTC. Based on this insight, previous research (Bao et al., 2019) predicted LTCs by developing Machine Learning (ML) models using 63 features representing the development activities of new contributors during the first month from a large OSS dataset.

The initial research objective driving the subject research was based on the lack of predictions using features that capture contributor programming language skills, time to

on-board, and project age. This research was extended to two distinct objectives and seven research questions following feedback from research publication submission. This research is published in Eluri, Mazzuchi, and Sarkani, (2021).

Research Objective 1: Develop a model that predicts LTCs based on the activities of the new contributor and all contributors involved in the repository before the first contribution by the new contributor, leading to the following research questions:

1. **Research Question 1.1:** Can past history effectively determine an LTC?
2. **Research Question 1.2:** How effective is the past history approach compared to the state-of-the-art approach?
3. **Research Question 1.3:** Which features are the most important in identifying LTCs using past history approach?

Research Objective 2: Develop a model that predicts LTCs based on contributor programming language skills, project age, and contributor time to on-board based on the activities of the new contributor and all contributors involved for the most recent one month, two months, and three months before and after the new contributor joins the project, resulting in the following research questions:

1. **Research Question 2.1:** Can onboarding activity effectively determine an LTC?
2. **Research Question 2.2:** How effective is onboarding activity approach in determining an LTC compared to the state-of-the-art approach?
3. **Research Question 2.3:** How effective is the feature selection method?
4. **Research Question 2.4:** Which features are the most important in identifying LTCs using onboarding activity?

The two hypotheses associated with research objectives are :

Research Hypothesis 1: With regard to the problem of predicting LTCs based on standard comparative metrics of accuracy, precision, recall, F1-score, MCC, and AUC, the proposed "past history" based model outperforms the known state-of-the-art approach.

Research Hypothesis 2: With regard to the problem of predicting LTCs based on standard comparative metrics of accuracy, precision, recall, F1-score, MCC, and AUC, the proposed "onboarding activity" based model outperforms the known state-of-the-art approach.

1.4 Overview of Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 presents previous research related to this study. Chapter 3 describes the proposed methodology, including data sources, dataset description, features considered, prediction models, evaluation metrics, and validation method. Chapter 4 explains the results, recommendations to the repository owners, and threats to the validity of this research. Chapter 5 concludes this dissertation with recommendations for future work.

Chapter 2: Literature Review

In this chapter, first, OSS is described. Next, a literature review of articles related to contributor motivations to join OSS project, attracting contributors, accepting contributors, barriers encountered by contributors, reasons for contributor turnover is provided. Next, research on contributor LTC prediction is reviewed. Finally, details of the work that is most closely related to this research are provided.

2.1 Open-source Software

Open-Source Initiative (OSI) is a non-profit organization with a worldwide reach established to raise awareness of the advantages of open-source software, advocate for its use, and create bridges between various constituencies within the community (OSS website, 2020).

One of OSI's most important activities is maintaining the Open-Source Definition as a standards body. According to the (OSS website, 2020), OSS is described as "software that is freely accessible, usable, modifiable, and shareable by anybody (in modified or unmodified form)." OSS is developed collaboratively by a large number of people and distributed under licenses that adhere to the internationally recognized Open-Source Definition, which establishes 10 criteria (reproduced in Table 2.1) that must be met for any software license, and the software distributed under that license, to be labeled "Open-Source software."

2.2 Contributor Motivation

Volunteers contribute to OSS projects without direct economic incentives for their efforts (Robles, Gonzalez-Barahona, and Jesus, 2006). Volunteers have both extrinsic and intrinsic motivations for contributing to OSS projects. Extrinsic motivation is to build a reputation

Table 2.1: Criteria for software to be open-source (OSS website, 2020).

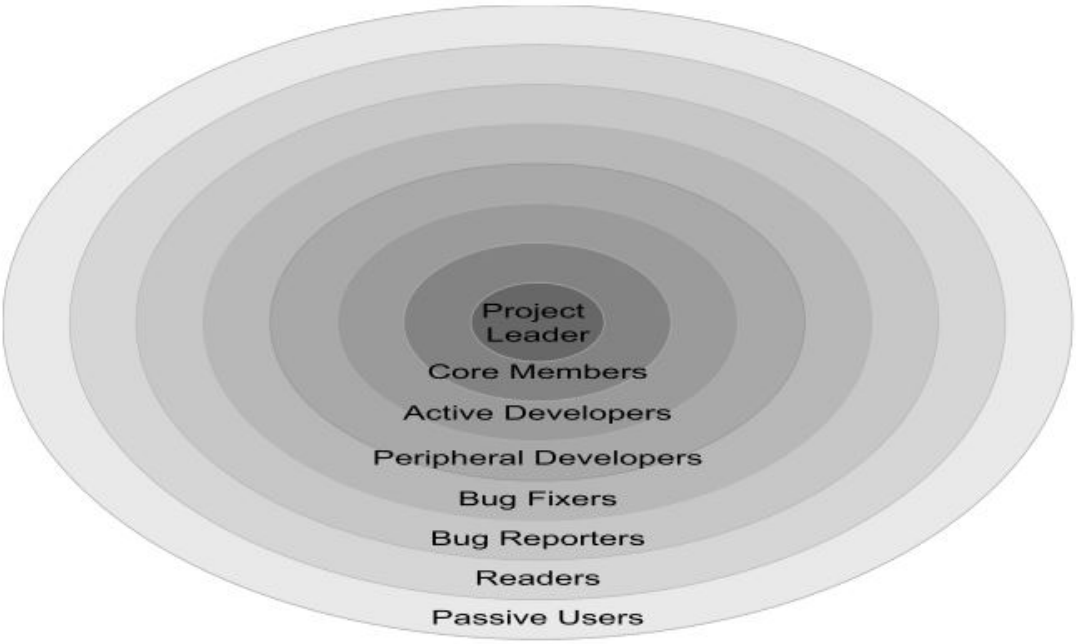
S.No	Criteria	Explanation
1	Free Redistribution	The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2	Source Code	The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
3	Derived Works	The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4	Integrity of The Author's Source Code	The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5	No Discrimination Against Persons or Groups	The license must not discriminate against any person or group of persons.
6	No Discrimination Against Fields of Endeavor	The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7	Distribution of License	The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8	License Must Not Be Specific to a Product	The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
9	License Must Not Restrict Other Software	The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
10	License Must Be Technology-Neutral	No provision of the license may be predicated on any individual technology or style of interface.

within the open-source community, which could open a way to job chances (Crowston, 2011). Intrinsic motivation is associated with a sense of competence and achievement from contribution to OSS projects (Schilling, Laumer, and Weitzel, 2012).

Alexander Hars, (2002) collected data using an online survey that was sent via email to 389 individuals and completed by 81 contributors with a completion rate of 21%. They analyzed the survey data using correlation coefficients and discovered that OSS contributions are motivated by various factors, many of which are based on external incentives. While inner drive, altruism, and affiliation with a group were major variables, external ones such as immediate remuneration and the prospect of future monetary benefits were even more important. Personal desire for a software solution was another important issue for which the research community has not yet gotten enough consideration. According to their study results, developers who took part in OSS projects may be divided into various categories. The most inwardly driven individuals were hobbyists and students. Salaried and contract programmers, on the other hand, were the ones who were mostly driven by external factors. A significant number of developers were compensated for their contributions to OSS projects.

Ye and Kishida, (2003) studied motivations for the use and development of OSS projects. Their study classifies the roles of contributors in the OSS community into eight categories, as shown in Figure 2.1. First, the project leader, who started the project, sets the vision and drives the project forward. Second, the core members, who have been involved with the project for an extended period and have contributed significantly to its progress. The core members guide and coordinate the OSS project's development. Third, active developers, contributors who contribute regularly. Fourth, peripheral developers, occasional contributors who add new functionality to the existing system. Fifth, bug fixers, contributors who fix bugs. Sixth, bug reporters, contributors who report bugs in the current system. Seventh, readers, those who are curious, read the code to learn how the system works—finally, passive users, those who just use the system.

Figure 2.1. OSS contributor role classification proposed by Ye and Kishida, (2003).



Ye and Kishida, (2003) hypothesized that the roles of contributors change by the number of contributions made and use OSS GIMP project's total contributions (code and emails) to classify roles among contributors. GIMP is GNU Image Manipulation Program - a cross-platform image editor available for GNU/Linux, OS X, Windows, and more operating systems. Their investigation showed that 162 members contributed to the project core. Fifty percent, roughly 82 of the 162, contributed once or twice. Only 18 members developed the majority of the system, each making more than 50 contributions with a cumulative contribution of more than 71% of the code. Forty-seven active members that were given write permission to contribute directly to the source tree. There were 111 peripheral developers whose contributions had to be integrated by core or active members. One hundred six members reported bugs via 476 emails. They concluded that core members are most influential as they have the final say in the decision-making.

Ye and Kishida, (2003) concluded that: (1) The primary motivation that attracts software developers to participate and become part of OSS communities is learning. (2) Exploring new ideas and learning through real-time tasks motivates project leaders. (3) For other

participants, the motivation is learning during maintenance, creating new features, fixing bugs, and reporting bugs.

Hertel, Niedner, and Herrmann, (2003) attempted to explain contributor motivations with two models based on a conceptual framework. The first model is named Extended Klandermans Model (EKM) and has its roots in research on voluntary involvement of contributors to promote OSS usage and was considered a social movement to support OSS. In the EKM model, Hertel, Niedner, and Herrmann, (2003) referred to four motivational components "...collective motives, norm-oriented motives, reward motives, and identification processes ...". The second conceptual model mainly focuses on team-based motives, i.e., the motivations behind contributors working in small teams. The second model is named Valence Instrumentality Self-efficacy and Trust (VIST) model and is coined after four attributive components. Valence is similar to collective motives in the EKM model, i.e., a team member's motivation is directly related to their subjective assessment of the team's objectives. The term "instrumentality" refers to the indispensability of an individual's contributions. The more the perceived instrumentality, the greater the incentive to work together to accomplish team goals. Self-efficacy is a term that refers to a team member's perceived ability to demonstrate the appropriate behaviors for team tasks. Trust relates to team members' anticipation that their efforts will be rewarded and not exploited by others.

Hertel, Niedner, and Herrmann, (2003) conducted a web-based survey to test the EKM model and VIST model was taken by the OSS Linux-kernel project members containing 141 participants within age-range of 16-54 from 28 different countries. For EKM-related analyses, the participants were developer and interested reader groups. For VIST model analyses, the participants were only the developer group. The survey revealed that EKM-related and VIST-related factors led to more contributions made to OSS. Those factors include the total number of hours contributors spent on the OSS, which meant a willingness to increase participation. Also, collective motives, which in this case was Linux kernel development. The study also showed that contributors had personal growth by contributing

to OSS community.

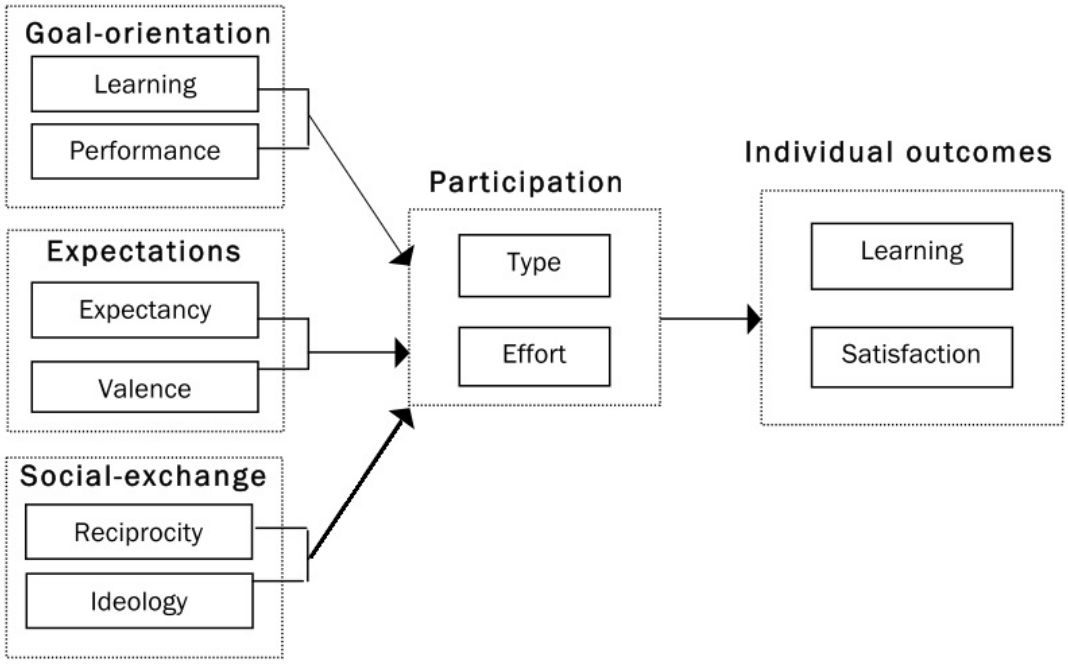
Benbya and Belbaly, (2010) investigated OSS contributor motivation to participation and outcomes with a multi-theoretical approach, as participation in OSS projects is motivated by various factors, and there is no one theoretical framework that can offer a complete explanation of the phenomenon. Different theories of motivation may be more effective at explaining participation and the extent it influences outcomes. Benbya and Belbaly, (2010) proposed a multi-theoretical framework built on goal-orientation theory, expectancy theory, and social exchange theory to explain involvement (both in terms of kind and extent) and how it relates to results (learning and satisfaction).

Goal theory is founded on the assumption that contributors are motivated to achieve their goals. As a result, contributors will aim their behavior toward these goals. There are two kinds of objectives: learning goals and performance goals. Goal orientation distinguishes between them. Learning objectives are concerned with the contributor's desire to further personal growth, while performance objectives focus on the contributor's desire to show competence.

Participants' involvement in OSS projects may be connected to expected benefits, in addition to both learning and performance goals. Individuals who actively engage are more likely to earn "private incentives," mostly linked to work benefits and professional success. Expectancy theory is used to consider these professional expectations that are not considered for by goal orientation. According to expectation theory, a person's anticipation of particular outcomes (expectancy) and their desirability (valence) impact their involvement levels and result outcomes.

Finally, social exchange theory is applied to understand better why contributors share their valuable information and put out effort in OSS project activities for the benefit of others. Social interaction can be classified into two major dimensions: reciprocity and ideology. Reciprocity indicates that contributors are engaged in an exchange relationship because they anticipate receiving something in return for their contribution. When a contributor expresses

Figure 2.2. A Multi-Theoretical framework of contributor motivation in OSS proposed by Benbya and Belbaly, (2010).



a connection to a particular social group, it is assumed that the group membership has some emotional or value importance to the ideology that expressed it.

They study the contribution on two dimensions: type and effort. The type of contribution refers to the kind of work contributed, e.g., code contributed or code reviewed. The effort is the time spent on contributing to OSS.

Benbya and Belbaly, (2010) created the research model shown in Figure 2.2, which explains how variations in goal-orientation (learning vs. performance), expectancies (expectancy and valence), and pro-social conduct (reciprocity and ideology) impact involvement (type and extent) and results (learning and satisfaction).

Benbya and Belbaly, (2010) selected 50 projects hosted on SourceForge.net in 2007-2008 that met the following criterion: (1) the project is in the software development category (code generators, designing tools, and frameworks), (2) the project is in enterprise application development domain, (3) the projects had some activity (such as code contributions to the

project, requests for defect fixes, support, patches, or features, or page views) in the previous week of data collection. A contributor whose role was defined as project administrator was sent two surveys, one for each time period, for each project. Out of 320 contributors, 122 contributors replied to two-times period survey (a total response rate of 38.12%).

Benbya and Belbaly, (2010) gathered information from the surveys conducted throughout two time periods. The motivation and involvement of contributors in OSS projects were evaluated in the first survey period (2007-2008). The results of contributors' involvement in OSS projects were assessed in a second survey conducted in 2008 (seven months after the first study). According to psychology, the temporal difference between motivation, participation, and performance was compatible with the overall model of motivation and performance.

In order to evaluate measures, Benbya and Belbaly, (2010) utilized Partial Least Squares. They discovered that learning objectives, professional expectations, ideology, and reciprocity are all determinants of involvement in OSS and that these variables should be considered together rather than individually when doing future research.

2.3 Contributor Attraction

Yamashita et al., (2016) studied the factors that contribute to attract and retain contributor to OSS projects. Their research intends to evaluate the recruitment and retention of contributors to open-source projects using a set of population migration indicators dubbed Magnet (for recruitment) and Sticky (for retention), to understand better how projects maintain and recruit contributors.

Yamashita et al., (2016) research focused on the characteristics that attract and retain contributors by measuring the attractiveness and retention of contributors in open-source projects using two metrics known as the Magnet and Sticky measures. The Magnet statistic represents the number of new contributors attracted to a project, while the Sticky statistic represents the number of current contributors that remain with the project. These indicators are

designed to represent each project's status in terms of attracting and retaining contributors.

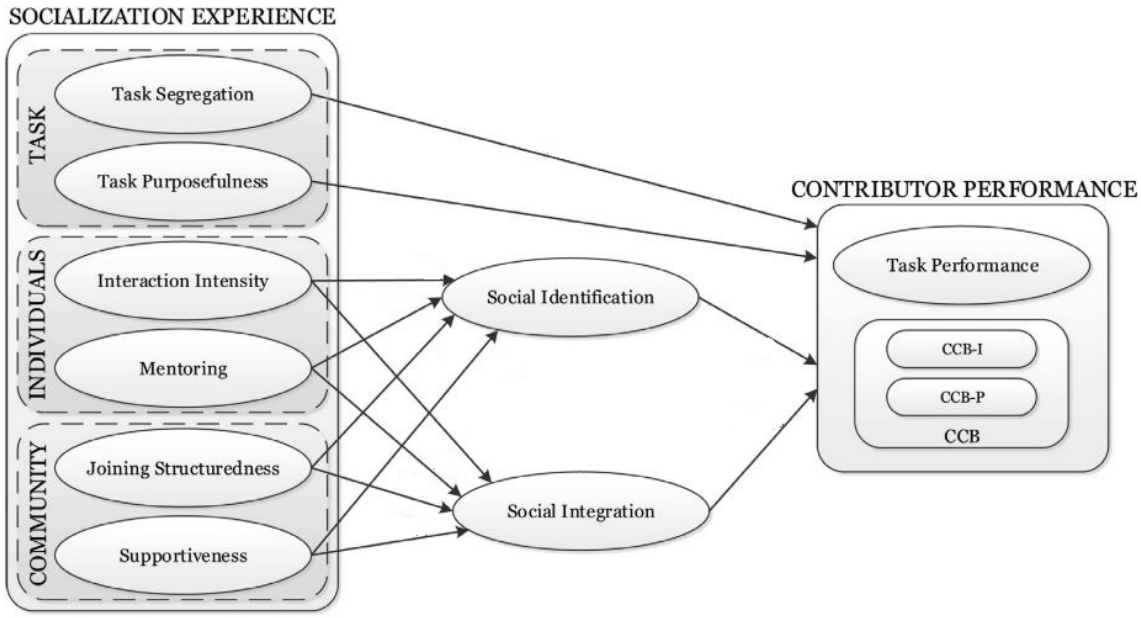
Yamashita et al., (2016) plotted Magnet and Sticky values on two dimensions and created a graph with four quadrants using the median Magnet and Sticky values as the dynamic quadrant thresholds. Projects in each quadrant are given names and described as follows: (1) Attractive projects (with high Magnet and high Sticky values) successfully attract new developers while keeping existing ones, (2) Fluctuating projects (with high Magnet and low Sticky values) are successful in attracting new developers while losing old ones, (3) Stagnant projects (with low Magnet and high Sticky values) keep their current development team but fail to recruit new members, and (4) Terminal projects (with low Magnet and low Sticky values) struggle to retain existing developers while failing to attract new ones.

Yamashita et al., (2016) examined 16,552 GitHub projects with more than 10 forks and developers. They discovered that (1) approximately 23% of contributors stay with the projects to which they contribute, (2) larger projects have a greater chance of attracting and retaining more contributors, (3) 53% of terminal projects ultimately devolve into projects with fewer than 10 contributors, and (4) 55% of attractive projects continue to be attractive.

Carillo, Huff, and Chawner, (2017) applied socialization and citizenship theories drawn from organizational behavior to study the effects of newcomer socialization experience on proximal socialization outcomes and contributor performance in open-source projects. Their investigation concentrated on big and complex OSS projects. Operating system software, desktop environment management systems, and programming language platforms are examples of complex undertakings requiring a high degree of modularity. The word "big" refers to the number of contributors rather than associated with software products.

Carillo, Huff, and Chawner, (2017) identified six unique features that characterize newcomer socialization experience. Two features are associated with the tasks performed by newcomers in a community (task segregation and task purposefulness). Two features are associated with interactions with other community members (interaction intensity and mentoring). The final two features are associated with interactions between newcomers and

Figure 2.3. A socialization and contributor performance conceptual model in OSS proposed by Carillo, Huff, and Chawner, (2017).



the community itself (supportiveness and joining structuredness). These characteristics are a distinct aspect of an individual contributor's experience to become a project participant.

Individual contributor performance was conceived by Carillo, Huff, and Chawner, (2017) as Task Performance (TP) and Community Citizenship Behaviors (CCB). The degree to which a contributor carries out project responsibilities by meeting some minimum level of performance set by the OSS project community is measured in TP. CCB is described as discretionary actions that are not directly or explicitly recognized by the project community's formal incentive system but which, taken together, enhance the community's overall effectiveness in performing its tasks and duties. The CCB construct is a second-order construct with two dimensions: CCB-I (citizenship behaviors oriented toward the benefit of individuals) and CCB-P (citizenship behaviors directed towards the benefit of the project).

The six socialization experience features, two proximal socialization outcomes (social identification and social integration), and contributor performance variables are described in Table 2.2, and the corresponding conceptual model is displayed in Figure 2.3.

Table 2.2: Features used in Carillo, Huff, and Chawner, (2017) study.

Dim	Feature Name	Explanation
Socialization Experience	***Task***	
	Task Segregation	The degree to which a newcomer has performed tasks that are specifically tailored for newcomers
	Task Purposefulness	The degree to which the newcomer has purposefully selected the sequence of tasks performed by a newcomer to learn how to become a contributor
	Individuals	
	Interaction Intensity	The degree to which a newcomer is actively engaged with other newcomers and community members while learning how to become a project contributor
	Mentoring	The degree to which a newcomer has been taken under the wing of one or more experienced members while learning how to become a project contributor
	Community	
	Joining Structuredness	The degree to which a newcomer has to adhere to a step-by-stepwise joining process to become a community contributor
	Supportiveness	The degree to which a newcomer has perceived a community to be supportive while learning how to become a project contributor
Proximal Socializtion Outcomes	Social Identification	Social identification is defined as the extent to which one identifies with an OSS community, leading to viewing the community's successes and failures as one's own
	Social Integration	Social integration is defined as the extent to which an individual perceives themself to be trusted and accepted by the other OSS community contributors
Contributor Performance	Task Performance	Task performance is defined as the extent to which an individual carries out project tasks by meeting some minimal level of performance defined by the OSS community to which they belong
	CCB	
	CCB-I behaviors	OSS community citizenship behaviors oriented towards the benefits of other individuals
	CCB-P behaviors	OSS community citizenship behaviors oriented towards the benefits of the project and its community

Carillo, Huff, and Chawner, (2017) tested the conceptual model on 12 large open-source projects: Debian, Fedora, FreeBSD, Gentoo, KDE, GNOME, Mozilla, NetBSD, OpenSUSE, Ubuntu, and Wikimedia/MediaWiki. They collected data using an online survey that was accessed by 1137 individuals and completed by 367 contributors with a completion rate of 32%.

Carillo, Huff, and Chawner, (2017) analyzed the survey data using Partial Least Squares and presented empirical evidence of a causal relationship between newcomer socialization and contributor performance in his paper. Using socialization variables, social identification, and social integration, the researchers found that they could explain 29% of the variance in task performance and approximately 44% of the variance in CCB. The model had a goodness-of-fit score of 0.433, indicating that it has strong predictive power.

2.4 Contributor Acceptance

Bird et al., (2007) developed a theory of OSS project joining and tested it using data mined from three open-source projects: Apache web server, Python programming language, and Postgres database. They identified three concepts influencing acceptance. First, contributor technical commitment to the project: contributor dedication to the project's success and the duration of the sustained technical engagement. Second, skill level: the contributor's knowledge/skills about the project. Third, individual reputation refers to one's status as a contributor in the community.

Bird et al., (2007) measured the three concepts using features described in Table 2.3 and created a dataset. Each record in the dataset represents one email participant for one month. The features listed in Table 2.3 are computed for each month per participant. The response variable is the time interval between a participant appearing on the project's mailing list and their first commit contribution to the project.

Bird et al., (2007) used survival analysis to understand which features influenced the duration to get accepted to the project. They found that only contributor reputation had a

Table 2.3: Features used in Bird et al., (2007) study.

Feature Name	Explanation
patch	A binary variable that indicates if this contributor has previously submitted a patch
norm_indegree	Measured as the proportion of the total population that has responded to this contributors since his/her first post
success_pct	Measures the percentage of patches submitted by this contributor that were marked as accepted
devs_cum	Measures the total number of messages sent by this contributor prior to the current measurement month
sent_cum	Total number of contributors in the project

substantial effect on contributor status for all three projects.

Ayushi, (2016) investigated the effect of the contributors' geographical location on the code acceptance choices made by integrators. They chose code contributions from the following 17 geographical locations: the United States, the United Kingdom, Germany, France, Canada, Japan, Brazil, Australia, Russia, the Netherlands, China, Spain, India, Switzerland, Sweden, Italy, and Belgium. These 17 geographical areas account for roughly 83% of the overall contributor population on GitHub and at least 1% of all code contributions. A total of 70,740 contributor code submissions from GitHub were selected based on the following criteria: (1) The code submission is from one of the 17 geographical locations, (2) The code contributor did not merge the code themselves, and (3) The data was controlled for known influencing attributes such as project age, project popularity, contributor code contribution size, contributor code quality, and code submission comments received.

Ayushi, (2016) developed the Logistic Regression (LR) model to understand better the effect of geographic location on code acceptance. They discovered that the geographical location substantially impacts code acceptance. In order to verify the findings, Ayushi, (2016) gathered information from a survey sent to 14,979 participants (of whom 2,532 responded, for a response rate of 17% overall). According to survey answers from contributors, contributors from certain nations believe they have been subjected to greater prejudice than those from other countries, which corresponds to the findings of the quantitative LR

mode. These prejudices influence contributors joining the project and becoming LTCs.

El Asri et al., (2017) looked at five OSS projects (AngularJS, Docker, Rails, Symfony, and TensorFlow) with more than 700 contributors to identify what characteristics are essential for new contributors to become core members. They chose five projects with a range of lifespans, programming languages, and domains to get a varied set of historical perspectives. Using GitHub's Representational State Transfer (REST) Application Programming Interface (API), they collected historical information on commit author name, commit date, amount of code churn, count of comments on commits, number of reviews, number of edited files. In addition, they also collect co-editing variables for each edited file that indicates whether or not a collaboration between two contributors occurred.

El Asri et al., (2017) sliced the data by month (time frames) to perform dynamic analysis of data. They constructed three undirected, weighted cumulative networks for each time frame:

1. Files co-edition Network in which vertices denote contributors, and edge weights represent the number of exchanges between those contributors based on the number of files they both edited.

2. Committed Comments Network vertices represent commenters, and edge weights represent the number of commits they engaged.

3. Review's comments Network is built on the interactions between comments on reviews.

El Asri et al., (2017) conducted a dynamic network analysis on monthly sequences of cooperation networks based on the co-edition of files, which was then published. Following one month's worth of activity after another, they created a series of cumulative cooperation networks that enabled them to examine the development of social structures inside each community and the evolution of its contributors, using a core-periphery view of the process.

According to the findings of El Asri et al., (2017), the most significant collaborative activity for joining and remaining in the core team of an OSS project is the number of contributed changes (#commits) submitted. Consequently, the more source code modifications a new contributor produces, the quicker and more likely he will be promoted to the core development team.

2.5 Contributor Barriers

Zanatta et al., (2017) conducted a study to determine the barriers that prevent newcomers from contributing to software crowdsourcing projects in exchange for paid employment in a competitive paradigm. Software crowdsourcing leverages the cloud to offload a portion or all of a software project to an unidentified developer workforce. The authors anticipate that newbies to software crowdsourcing suffer similar challenges to those new to open-source development. They analyzed two types of data: data retrieved from Topcoder, a well-known crowdsourcing site, and comments from newcomers following their first participation in a software crowdsourcing project. Their investigation found that the primary obstacle preventing newcomers from participating in initiatives is a lack of documentation.

Zhou and Mockus, (2011) conducted a study of three OSS projects (JOnAS, JBossAS, and Mozilla) to determine the relationship between OSS projects initial social environment and the probability that a new developer becoming an LTC (a developer contributing for more than 28 months is considered an LTC). OSS project social environment is measured by Project Relative Sociality (PRS), which is defined for a project per time period (generally a month) as the geometric mean of the Relative Sociality (RS) of individual contributors active in the project during that time period. Individual contributor Relative Sociality (RS) is defined as the fraction of the size of a contributor social network to the number of activities the contributor performs. For example, a developer (D1) has been involved in two code changes (C1 and C2) during a particular month. To complete the code change C1, assume D1 interacted with two developers (D2 and D3) and to complete the code change C2, assume

D1 interacted with two developers (D3 and D4). Therefore, the D1's social network is three (D2, D3, and D4), and the number of tasks is two (C1 and C2).

Periods with high PRS are considered to be characteristic of a more social project environment. Each project's timeline is then categorized into three types of periods based on PRS values: the quarter of the time when PRS has the smallest values (low level), a quarter of the time PRS has the largest values (high level), and the remaining half of the time PRS has values in the middle (medium level) Zhou and Mockus, (2011) investigated JOnAS, JBossAS, and Mozilla OSS projects and partitioned each project's timeline into the three level PRS periods. Next, developers are assigned to the level of the period that they were working on when they first joined the project.

Zhou and Mockus, (2011) built an LR model with *glm* function with binomial family from R system representing the equation:

$$L = Initial + Type + Project \tag{2.1}$$

where L is an LTC indicator, $Initial$ represents PRS level at the developer joining time ("Medium," "Low," and "High"), $Type$ is OSS or non-OSS indicator, and $Project$ is the project indicator. Zhou and Mockus, (2011) discovered that the possibility of a newcomer becoming an LTC is greatest for all projects during periods of low PRS. Zhou and Mockus, (2011) concluded that in an OSS project, the periods of low PRS might correspond to lower project social interaction. During these lower project social interaction times, newcomers are more likely to be invested in the project and have more opportunities to get fully engaged and add significant functionality.

2.6 Contributor Turnover

Yu, Benlian, and Hess, (2012) studied the reasons for the turnover of contributors using Herzberg's two-factor theory, which associates factors that result in employee dissatisfaction

Table 2.4: Adaptation of Herzberg, (1966) hygiene features to OSS projects in Yu, Benlian, and Hess, (2012) study.

Herzberg Hygiene Feature	**OSS Hygiene Feature**
Company policy and administration	Project regulation and administration
Supervision	Support
Relationship with boss, peers and subordinates	Working relationships with other members
Salary	Future rewards
	Personal needs for software functionalities

(Herzberg, 1966) and Mobley´s model of job (dis)satisfaction and turnover (Mobley, 1977).

According to Herzberg's two-factor theory (Herzberg, 1966), job satisfaction and dissatisfaction do not exist on a continuum but rather as distinct events that occur regularly (i.e., a gain in satisfaction does not always result in a drop in discontent). In Herzberg, (1966), motivators are defined as factors that lead to satisfaction, while hygiene factors are defined as attributes that lead to unhappiness. Herzberg, (1966) conducted a series of interviews about employee attitudes and their impacts, then did content analysis to determine the most important hygiene elements, including company policy and administration, supervision, pay, interpersonal relationships, and working environment.

Mobley, (1977) demonstrated the link between work (dis)satisfaction and turnover and described a psychological and causal heuristic decision process for quitting. Mobley, (1977) found a strong negative relationship between satisfaction and the frequency with which people consider quitting.

Yu, Benlian, and Hess, (2012) used Herzberg's two-factor theory (Herzberg, 1966) and Mobley´s job (dis)satisfaction and turnover model (Mobley, 1977) to create a research model (Figure 2.3) to understand better the influence of OSS project hygiene features on contributor dissatisfaction and perceived turnover. As indicated in Table 2.4, Yu, Benlian, and Hess, (2012) adopted four critical hygiene criteria, identified their OSS counterparts, and incorporated additional influential components into the research model.

Yu, Benlian, and Hess, (2012) evaluated the research model empirically with data collected from a web-based survey sent to contributors working in sourceforge.net and

Figure 2.4. Research model of perceived turnover in OSS projects proposed by Yu, Benlian, and Hess, (2012).

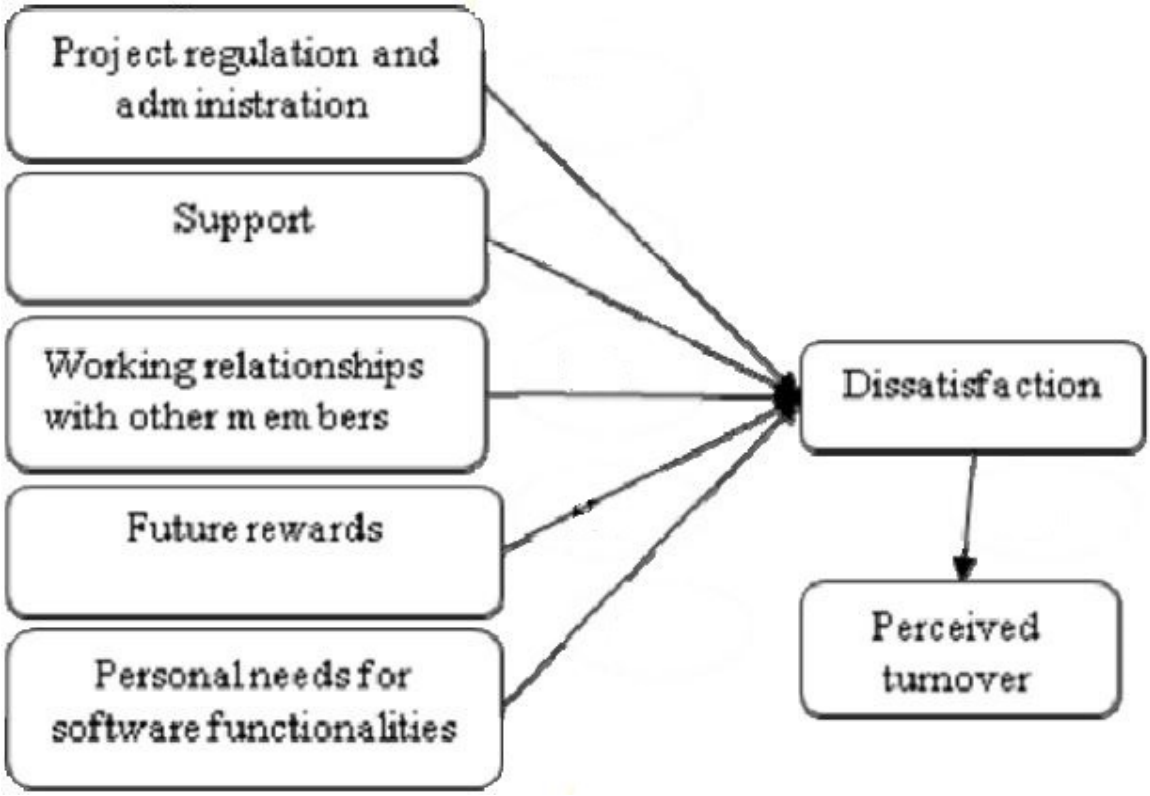


Table 2.5: Features used in Sharma, Hulland, and Daniel, (2012) study.

Level	Feature Name	Explanation
Contributor	Role of the Developer	A project may employ developers for various roles that range in the level and kind of expertise required. Two dummy variables, Developer and Admin with the base group Other (which included all other roles), were created
	Number of Projects	The number of OSS projects undergoing active development that the contributor was involved in
	Past Activity Level	Computed as a binary variable. A contributor was deemed active in the past, coded as 0, if they made at least one commit in the previous 10 month period; otherwise, it is coded as 1
	Tenure	The approximate number of months a contributor joined SourceForge
Project	Project Age	Number of months since the project was registration on SourceForge
	Size of Project	Number of contributors with commit access to the project's code

launchpad.net (121 contributors' responses were usable — an overall response rate of 12%).

Yu, Benlian, and Hess, (2012) used Partial Least Squares to examine measurements. They found that project regulation and administration, future incentives, and personal requirements for software functionality are all highly correlated with discontent. Support and working relationships are insufficiently stable to qualify as hygiene variables.

Sharma, Hulland, and Daniel, (2012) conducted an exploratory study to understand contributor turnover better. Using features from 201 contributors and 40 active projects from SourceForge, they built a Logistic Hierarchical Linear model of OSS turnover behavior.

Sharma, Hulland, and Daniel, (2012) collected features on two levels: the contributor level, which investigates aspects that may impact contributors' inactivity choices, and the project level, which investigates aspects that may impact the rate of project turnover. The features are described in Table 2.5.

Sharma, Hulland, and Daniel, (2012) built a Logistic Hierarchical Linear model on understanding which features influenced contributor turnover and found previous activity, contributor role, project size, and project age to be major turnover factors.

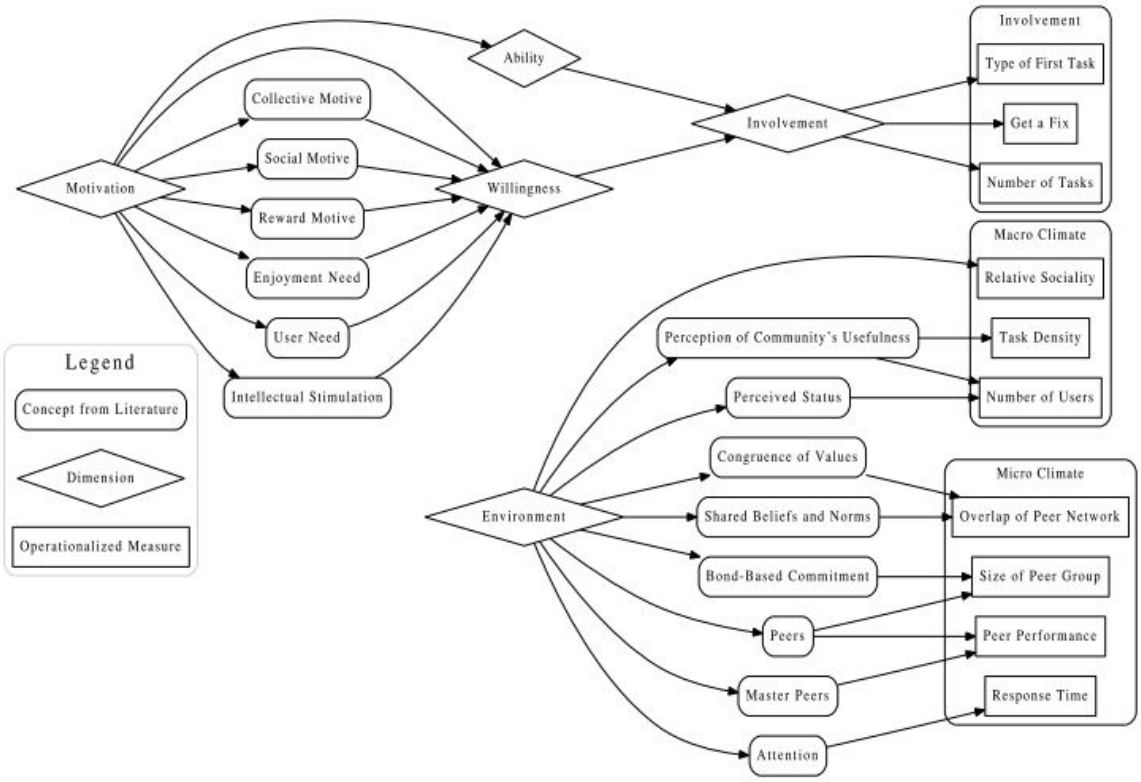
2.7 Long-time Contributor Prediction

In OSS, it is difficult to predict when a contributor may depart an open-source project. Due to the uncertain nature of contributor engagement, open-source projects contain an element of risk (Eluri, Sarkani, and Mazzuchi, 2019).

Schilling, Laumer, and Weitzel, (2012) investigated the predictive power of standard organizational Person-Job (P-J) and Person-Team (P-T) notions on contributor retention. The study examines the behavior of 80 past Google Summer of Code student contributions in terms of Person-Job (P-J), which refers to an individual's suitability for a particular job, and Person-Team (P-T), which refers to an individual's relational compatibility with an existing team, to propose and test and very an objective assessment strategy for evaluating the retention behavior of open-source contributors. The study discovered that P-J and P-T fit are the two notions best adapted to predicting contributor persistence in open-source projects. According to P-J Fit, a student's related development background is directly related to their project retention following the event. The study also discovered that students' familiarity with the project team's coordination procedures substantially correlates with the amount of time they spend on their projects following Google Summer of Code.

Zhou and Mockus, (2015) researched on LTC prediction. They assess an individual's likelihood of becoming an LTC based on their capability, willingness, and ability to contribute at the time of joining. Individual willingness is quantified by the quantity and variety of tasks they initiate and the value of their contribution to the OSS project, as measured by the percentage of reported issues that were finally resolved. The opportunity is modeled using the macro-climate and micro-climate of the projects. Macro-climate refers to a project's general environment, which is consistent for all contributors. Micro-climate is a term that refers to the unique environment that exists for each participant in the project. The relationship among the concepts described in this section and operationalization of the concepts to features is shown in Figure 2.5 and Table 2.6.

Figure 2.5. Relationship among concepts, dimensions, and measures proposed by Zhou and Mockus, (2015).



Zhou and Mockus, (2015) gathered defect reports from Mozilla and Gnome, two projects with over 100,000 contributors each, and developed a LR model based on 10 features from three dimensions: the extent of involvement (contributors' activities), macro-climate (projects as a whole), and micro-climate (the environment of the individual contributor). Their model's average precision and recall for predicting who will remain in the project are claimed to be 31.64% and 24.29%, respectively.

Zhou and Mockus, (2015) discovered that a newcomer's attitude toward the project as demonstrated by their initial contribution of a comment or a bug report using the Bugzilla interface or through a crash-reporting tool doubles her chances of becoming an LTC. Fixing reported issues also increases the individual's chances of becoming an LTC. Zhou and Mockus, (2015) research findings help: (1) individual contributors to become aware of the roles they play and help them contribute most effectively, (2) OSS communities find ways

Table 2.6: Features for participant *i* (Zhou and Mockus, 2015).

Dim	Feature	Refers to	Explanation
Extent of Involvement	nCmt	# of tasks	The logarithm of the number of comments + 1
	GotFix	Get a Fix	At least one of the issues reported by contributor i was fixed
	withBB	Type of First task	First report of contributor i using a crash reporting tool
Macro-Climate	FNotRep	Task density	Contributor i starts participation with a comment
	nUsr	# of users	Number of product users when i joins
	RS	Relative Sociality	Project's relative sociality
Micro-Climate	nPeer	Size of peer group	The logarithm of contributor i's peers' group size + 1
	pShared	Overlap of peer network	Logarithm of the social clustering of contributor i's peer group
	PeerPerf	Peer Performance	Logarithm of the min productivity (issues/-month) of the peers
	LckAttn	Response Time	The longest duration between the contributor i's action until the response is less than one hour

to retain newcomers better. To conclude, the possibility and the probability of an individual becoming an LTC is connected to how valuable his contributions are to his project by commenting, contributing to issue reports, and the project's attention to the new individual.

Wang et al., (2018) used a single GitHub project, the Ruby on Rails (Rails) project, and collected behavior data from 36,661 developers to predict LTCs. They achieved an average precision and recall for LTCs after one year of 50% and 2.2%, respectively, and an AUC of the linear regression model obtained via 10-fold cross-validation of 80.7%. Precision is the fraction of contributors correctly classified as LTCs among those classified as LTCs, and recall is the fraction of LTC contributors correctly classified.

Decan et al., (2020) predicted when contributors would abandon an active project by forecasting the future commit activity of contributors. They developed a probabilistic forecasting model based on survival analysis that uses each contributors' prior and recent contributions to estimate the amount of time before their next contribution.

Decan et al., (2020) used Git Activity Predictor, an open-source tool to build the model. The model had several benefits like the ability to predict without requiring a lot of data, predict for both short-lived projects and long-term projects as only a little prior knowledge is

Table 2.7: Features used in Decan et al., (2020) study.

Dim	Feature	Explanation
Project	longevity	The duration between the first and last project commit
	stars	Number of stars received by the project
	forks	Number of times the project was forked
	contributors	Number of project contributors
	commits	Number of project commits
Contributor	longevity	The duration between the first and last commit to this project
	projects	The number of projects this contributor is contributing.
	commits	Number of contributor commits

needed, and the model can be used in real-time as the analysis only requires a few seconds.

Decan et al., (2020) built the model using a dataset that consisted of 7,528 open-source projects, 5,947 active contributors, and 374,622 unique days of commit activity from all 17,340 contributors/project pairs. Table 2.7 shows and describes the project features and contributor features considered.

Decan et al., (2020) tested the model on over 7,500 git projects representing a wide range of projects based on previous m historical observed intervals between activity days with various values of m. Decan et al., (2020) demonstrated that the estimates produced are usually near to the actual values and discovered that relying on the previous $m = 20$ observed intervals between days of activity yields the least sum of squared difference score (0.24) compared to the other scores, which are between 0.25 (+4%, when $m = 30$) and 0.34 (+41%, when $m = 100$).

2.8 Closely Related Work

Similar empirical LTC prediction studies by Zhou and Mockus, (2015) and Bao et al., (2019) inform this dissertation's research. While Zhou and Mockus, (2015) used only two OSS projects and 10 features to predict LTC, Bao et al., (2019) considered a large number of OSS projects from GitHub. They trained ML models to predict whether a new contributor would remain with a project after a certain period of time. Time periods of 1, 2, and 3 years

were assessed. The research in this dissertation has two new approaches that are closely related to Bao et al., (2019).

Bao et al., (2019) developed prediction models based on a dataset extracted from a GHTorrent MySQL database snapshot on 09/01/2017. The dataset is constructed on a total of 75,046 contributors from 917 projects, where each record has a (project, contributor) pair and 63 features extracted from GitHub project and contributor activity. Details on how 917 projects are selected, 75,046 contributors are considered, and 63 features are extracted are provided.

The 917 repositories are selected using the process shown in Fig. 2.6. The process starts with GHTorrent MySQL database snapshot on 09/01/2017, then selecting the top 1,000 repositories sorted by decreasing number of stars[1] and excluding repositories that do not have a programming language specified, do not use GitHub as the issue tracker, are forked from another repository, or have been deleted from GitHub.

Next, contributors who submitted at least one commit after their GitHub account creation time are considered for each repository. 75,046 contributors are identified. Next, a contributor for a project is regarded as an LTC if the contributor continues to commit to the repository for more than a time T. Three settings for T are used, namely, T = 1 year, 2 years, and 3 years.

Bao et al., (2019) extracted 63 features in five dimensions that may influence a new contributor's decision to become an LTC. Table 2.8 explains the significance of each feature organized in five dimensions: DP, RP, DA, RA, and CN.

DP Dimension. This dimension captures developer information based on the historical activities of the contributor in the GitHub repository. Bao et al., (2019) extracted eight features – user_age, user_own_repos, user_watch_repos, user_contribute_repos, user_history_commits, user_history_issues, user_history_pull_requests, and user_history_followers. These features contain contributor

[1]Developers can keep track of a GitHub repository by starring it.

Figure 2.6. Data collection process used by Bao et al., (2019).

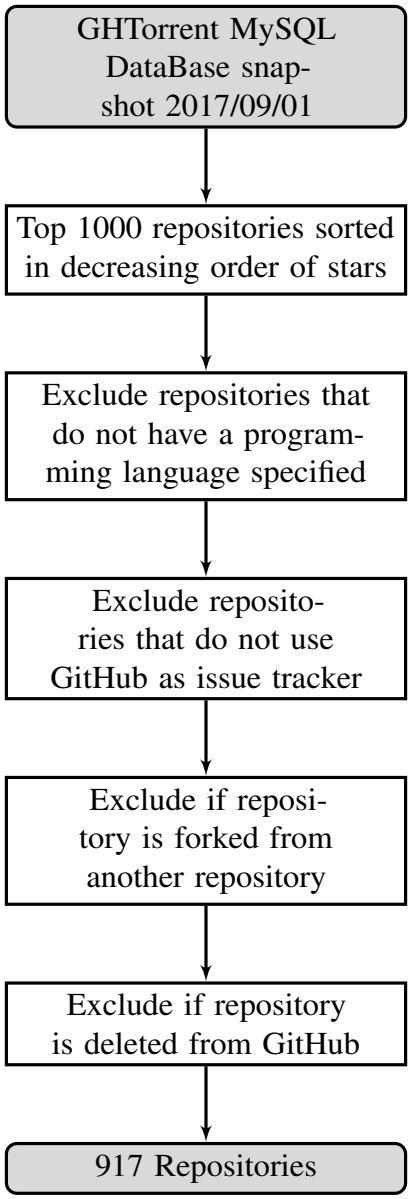


Table 2.8: Features used in Bao et al., (2019) study.

Dim	Feature Name	Explanation
DP	user_age	Number of days between the registration date of the new developer and the date that he/she joins the repository
	user_own_repos	Number of repositories the new developer owns when he/she joins the repository
	user_watch_repos	Number of repositories the new developer watches when he/she joins the repository
	user_contribute_repos	Number of repositories in which the new developer has submitted at least one commit when he/she joins the repository
	user_history_commits	Number of commits the new developer has submitted when he/she joins the repository
	user_history_pull_requests	Number of pull requests the new developer has submitted when he/she joins the repository
	user_history_issues	Number of issues the new developer has submitted when he/she joins the repository
	user_history_followers	Number of users who follow the new developer
RP	Language	Main programming language used by the repository
	before_repo_commits	Number of commits that the repository have when the new developer joins
	before_repo_commit_comments	Number of commit comments that the repository has when the new developer joins
	before_repo_contributors	Number of contributors that the repository has when the new developer joins
	before_repo_contributor_{S}	Statistics of commits of contributors in the repository, where S can be max, min, mean, median, and std
	before_repo_issues	Number of issues that the repository have when the new developer joins
	before_repo_issue_comments	Number of issue comments that the repository has when the new developer joins
	before_repo_issue_events	Number of issue events that the repository has when the new developer joins
	before_repo_issue_events_closed	Number of issue closed events that the repository has when the new developer joins
	before_repo_issue_events_assigned	Number of issue assigned events that the repository has when the new developer joins
	before_repo_pull_requests	Number of pull requests that the repository has when the new developer joins
	before_repo_pull_request_comments	Number of pull request comments that the repository has when the new developer joins
	before_repo_pull_request_history	Number of pull request events the repository has when the new developer joins
	before_repo_pull_request_history_merged	Number of merged pull request events the repository has when the new developer joins
	before_repo_pull_request_history_closed	Number of closed pull request events the repository has before the new developer joins
	before_repo_watchers	Number of watchers the repository has when the new developer joins
DA	month_user_commits	Number of commits that the new developer submits to the repository in the first month
	month_user_commit_comments	Number of comments received in the commits submitted to the repository in the first month
	month_user_issues	Number of issues the new developer submits to the repository in the first month
	month_user_issues_comments	Number of comments received in the issues submitted by the new developer in the first month
	month_user_issue_events	Number of events received in the issues that the new developer submits to the repository in the first month
	month_user_issue_events_closed	Number of closed events received in the issues submitted by the new developer in the first month
	month_user_issue_events_assigned	Number of assigned events received in the issues submitted by the new developer in the first month
	month_user_pull_requests	Number of pull requests submitted by the new developer to the repository in the first month
	month_user_pull_request_comments	Number of comments received in the pull requests submitted by the new developer in the first month
	month_user_pull_request_history	Number of pull request events received in the pull requests submitted by the new developer in the first month
	month_user_pull_request_history_merged	Number of merged pull request events received in the pull requests submitted by the new developer in the first month
	month_user_pull_request_history_closed	Number of closed pull request events received in the pull requests submitted by the new developer in the first month
RA	month_repo_commits	Number of commits submitted to the repository in the first month that the new developer joins
	month_repo_commit_comments	Number of comments received in the commits submitted to the repository in the first month
	month_repo_contributors	Number of contributors who submitted at least one commits to the repository in the first month
	month_repo_contributor_{S}	Statistics of commits of contributors in the first month, where S can be max, min, mean, median, and std
	month_repo_issues	Number of issues submitted to the repository in the first month
	month_repo_issue_comments	Number of comments received in the issues submitted to the repository in the first month
	month_repo_issue_events	Number of events received in the issues submitted to the repository in the first month
	month_repo_issue_events_closed	Number of closed events received in the issues submitted to the repository in the first month
	month_repo_issue_events_assigned	Number of assigned events received in the issues submitted to the repository in the first month
	month_repo_pull_requests	Number of pull requests submitted to the repository in the first month
	month_repo_pull_request_comments	Number of comments received in the pull requests submitted to the repository in the first month
	month_repo_pull_request_history	Number of events received in the pull requests submitted to the repository in the first month
	month_repo_pull_request_history_merged	Number of merged events received in the pull requests submitted to the repository in the first month
	month_repo_pull_request_history_closed	Number of closed events received in the pull requests submitted to the repository in the first month
CN	degree_centrality	These metrics are used to quantify a newcomer's degree of activity in the collaboration structure of an OSS project
	closeness_centrality	
	betweenness_centrality	
	eigenvector_centrality	
	clustering_coefficient	

experience, activeness, and willingness to contribute to OSS in GitHub.

RP Dimension. This dimension is closely related to the macro-climate dimension used by Zhou and Mockus, (2015) and captures repository information based on historical activities in GitHub from all developers before a new contributor submits their first commit to the repository. Bao et al., (2019) extracted 20 features: Language, before_repo_commits, before_repo_commit_comments, before_repo_contributors, before_repo_contributor_max, before_repo_contributor_min, before_repo_contributor_mean, before_repo_contributor_median, before_repo_contributor_std, before_repo_issues, before_repo_issue_comments, before_repo_issue_events, before_repo_issue_events_closed, before_repo_issue_events_assigned, before_repo_pull_requests, before_repo_pull_request_comments, before_repo_pull_request_history, before_repo_pull_request_history_merged, before_repo_pull_request_history_closed, and before_repo_watchers. These features represent the complexity of the repository, since they capture a snapshot of all activity by all developers on the repository from inception to the time the new contributor joins.

DA Dimension. This dimension quantifies new contributor development activity in the repository after joining. Bao et al., (2019) extracted 12 features: month_user_commits, month_user_commit_comments, month_user_issues, month_user_issue_comments, month_user_issue_events, month_user_issue_events_closed, month_user_issue_events_assigned, month_user_pull_requests, month_user_pull_request_comments, month_user_pull_request_history, month_user_pull_request_history_merged, and month_user_pull_request_history_closed. Bao et al., (2019) hypothesized that these features represent the first impression of a new contributor toward the repository. For instance, merging the initial pull request of a new contributor into the repository would give the contributor a sense of accomplishment and increase their willingness to make more contributions to the project.

As part of the research in this dissertation, an issue with the calculation of month_user_issues was identified — both month_user_issues and month_user_issue_comments had the same value for all observations.

RA Dimension. This dimension is closely related to the RP dimension; whereas RP looks at the past, RA looks at the future three months after a new contributor joins the project. This dimension captures repository information based on activity in GitHub from all developers after a new contributor submits their first commit to the repository. Bao et al., (2019) extracted 18 features: month_repo_commits, month_repo_commit_comments, month_repo_contributors, month_repo_contributor_min, month_repo_contributor_max,month_repo_contributor_mean, month_repo_contributor_median, month_repo_contributor_std, month_repo_issues, month_repo_issue_comments, month_repo_issue_events, month_repo_issue_events_closed, month_repo_issue_events_assigned, month_repo_pull_requests, month_repo_pull_request_comments, month_repo_pull_request_history, month_repo_pull_request_history_merged, and month_repo_pull_request_history_closed. These features represent the speed at which the repository is changing, since they capture all activity by all developers on the repository after the new contributor joins.

CN Dimension. This dimension refers to features based on cooperation activities between a new contributor and other contributors in an OSS project during the first month after joining. Bao et al., (2019) build a collaborative network using the comments associated with commits, issues, and pull requests and compute five contributor collaboration activity features.

More specifically, given a project and a new contributor, Bao et al., (2019) first collect all of the project's commits, problems, and pull requests, as well as their corresponding comments, during the first month since the new contributor submitted their first commit. Then,

Bao et al., (2019) create a graph based on the contributors who make commits/issues/pull requests and those who provide comments on the relevant commits/issues/pull requests. Each node in the graph represents a contributor, and for each comment about commits/issues/pull requests, a directed edge connecting each participant to its author is created. Following that, as suggested by Zanetti et al., (2013), five contributor collaboration activity features: degree_centrality, closeness_centrality, betweenness_centrality, eigenvector_centrality, and clustering_coefficient, are computed using the Python program NetworkX.

- **degree_centrality** is defined as the number of connections that pass through a vertex, which may be read either as the possible effect on other vertices or as the quantity of data that a vertex possesses.

- **closeness_centrality** is defined as the inverse of the sum of a node's distances from all other nodes. It quantifies how near a vertex is to all other vertices in a network. A new contributor node with a higher closeness_centrality score is closer to all other contributors in the cooperation network.

- **clustering_centrality** is defined as the proportion of the number of triangular connections to a vertex to the quantity of a pair of edges connecting to the vertex. This characteristic quantifies the degree to which network vertices tend to cluster. A greater value of a new contributor's clustering_centrality suggests a higher density of collaborative activities with their neighbors in the network.

- **betweenness_centrality** is the total number of shortest paths that travel through that node between all feasible pairs of nodes. A greater number for a new contributor's betweenness_centrality suggests they have more power over the collaboration network because more data will pass through their node.

- **eigenvector_centrality** is a metric that awards scores to vertices in a network based on the idea that connecting to nodes with large centrality raises the vertex's centrality.

The eigenvector_centrality of a vertex is as shown in Equation (2.2), in which $DN(n_i)$ denotes the set of immediate neighbors of the node n_i and λ is the highest eigenvalue of the adjacency matrix of the network:

$$Ev(n_i) = \frac{1}{\lambda} \sum_{n_j \in DN(n_i)} Ev(n_j) \tag{2.2}$$

Bao et al., (2019) built naive Bayes (NB), Support Vector Machine (SVM), k Nearest Neighbours (kNN), Decision Tree (DT), and Random Forest (RF) models to predict LTCs using 10-fold cross-validation for the three time intervals. AUC metric is used to measure the effectiveness of models. The results found the models to accurately predict whether a novice will become an LTC soon after submitting their first commit to the project. Bao et al., (2019) found the RF model achieved the best results with AUC greater than 0.8 in all three time intervals and discovered that the number of followers a contributor has when they join the project is the most predictive feature in all time intervals.

Bao et al., (2019) provided the following four insights to project owners: (1) Retain experienced developers by recognizing when an experienced contributor makes an initial contribution and pay attention to the contribution and assign interesting tasks to them. (2) Make a concerted effort to encourage more contributions during a new contributor's first month. If a newcomer is encouraged to make a significant contribution in their first month, there is a good chance that the newcomer will remain on as an LTC. (3) Improve the activity level of a project's community to keep new contributors by innovating and driving the project with new requirements. (4) Use different strategies to retain new contributors for different stages of the project life cycle (like communicating more frequently for newer projects, assigning targeted tasks for mature projects, etc.,).

Chapter 3: Methodology

In this chapter, the experimental setup of the proposed methodology is described, including data sources, dataset description, features considered, prediction models, evaluation metrics, and validation method.

The first research objective is addressed by following the high-level methodology shown in 3.1. Figure 3.1 shows two datasets; one is the state-of-the-art Bao et al., (2019) 63 feature dataset. The other is a new dataset constructed as part of this research that has 31 features extracted from GitHub API and GHTorrent using BigQuery. Next, NB, SVM, kNN, LR, DT, and RF models are built to predict LTCs. Next, the models constructed with the two datasets are compared using Wilcoxon signed-rank test (Wilcoxon, 1946) with Bonferroni correction (Abdi et al., 2007), to determine whether the improvement over the baseline is statistically significant. Finally, Cliff's delta[1] (Cliff, 2014), a non-parametric effect size measure, is used to quantify the difference between the results of the two approaches.

The second research objective was addressed by following the high-level methodology shown in 3.2. Figure 3.2 shows three datasets; one is the state-of-the-art 63 feature dataset by Bao et al., (2019). The second and third are new datasets constructed as part of this research that have 173 and 55 features, respectively, extracted from GitHub API and GHTorrent using BigQuery. The rest of the methodology is the same as described in the previous section.

3.1 Data Source

GitHub,[2] is used as the primary data source. GitHub is an OSS hosting platform. As of August 2019, GitHub is the largest OSS hosting site, with 50 million contributors and 100 million repositories. In addition to source code, GitHub holds rich metadata describing

[1]Cliff defines delta values less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474 and above 0.474 as negligible, small, medium, and large effect sizes, respectively.

[2]GitHub : `https://github.com/about`.

Figure 3.1. Methodology map for research objective 1.

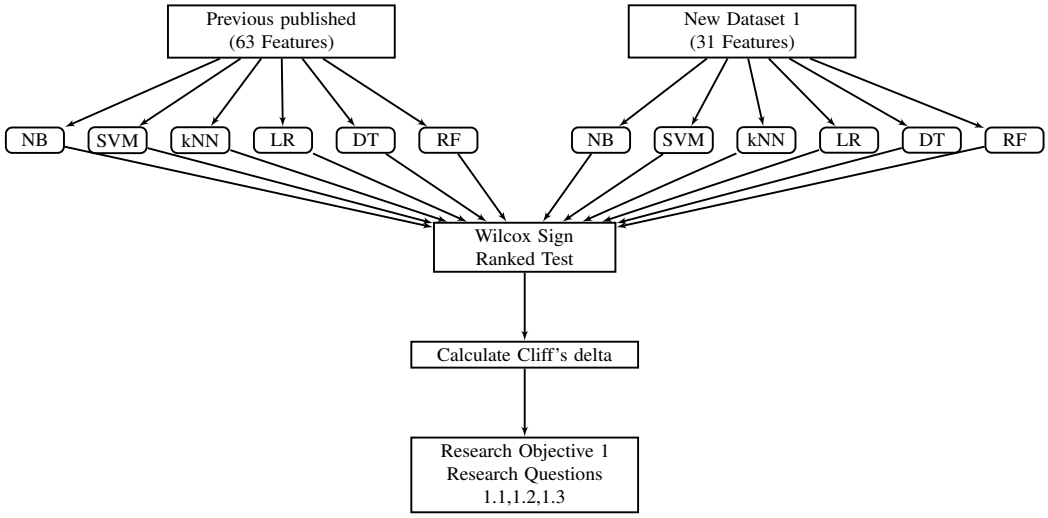
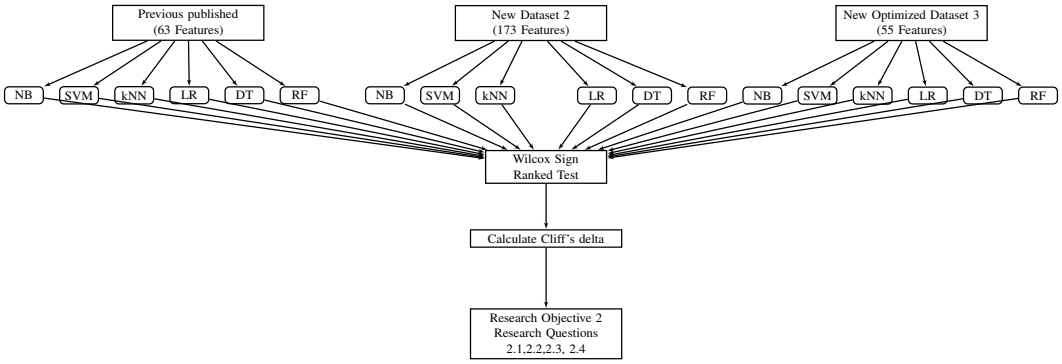


Figure 3.2. Methodology map for research objective 2.



important information, such as repository contributors, repository primary implementation language, contributor commits, and contributor comments.

GitHub metadata can be accessed in various ways, with different trade-offs. One way to access GitHub metadata is via the GitHub REST API[3]. While the API is available for free, GitHub throttles the number of connections and shuts down access if it suspects that the API is being abused. A second way to access GitHub metadata is through the GHTorrent project[4] (Gousios, 2013). The GHTorrent project exhaustively retrieves the contents of all GitHub events and stores them in a relational database using the schema shown in Figure 3.3.

The GHTorrent project provides a scalable query interface to all metadata through the GitHub REST API. While this query interface is accessible without fees, they are killed without warning if they run for more than 100 seconds. A third way is to access GitHub metadata offline using GHTorrent dump files (either as incremental MongoDB dump files or a single MySQL dump file) as used in (Blincoe et al., 2016, Jiang et al., 2019, Jarczyk et al., 2018, and Yu et al., 2016) for repository analysis. The third way to access GitHub requires researchers to download the MongoDB or MySQL dump files and restore them to a local database to access the data. While this approach is free, there is a learning curve to set up databases. A fourth way to access GitHub metadata is to use BigQuery.[5] GHTorrent project metadata files are published through Google BigQuery, a hosted SQL query engine for large datasets. BigQuery is easy to use, responsive, and reliable, but it is a paid service.

In this study, GitHub REST API (the first method described above) and GHTorrent using BigQuery (the fourth method described above) are used to extract features. Features extracted from GHTorrent using BigQuery are backed by relational database schema shown in Figure 3.3, and the entities used in this study are described in Table 3.1.

[3]GitHub API: `https://developer.github.com/v3/`.
[4]GHTorrent project: `http://ghtorrent.org/`.
[5]Google BigQuery: `http://bigquery.cloud.google.com`.

Figure 3.3. GHTorrent MySQL database schema from Gousios, (2013).

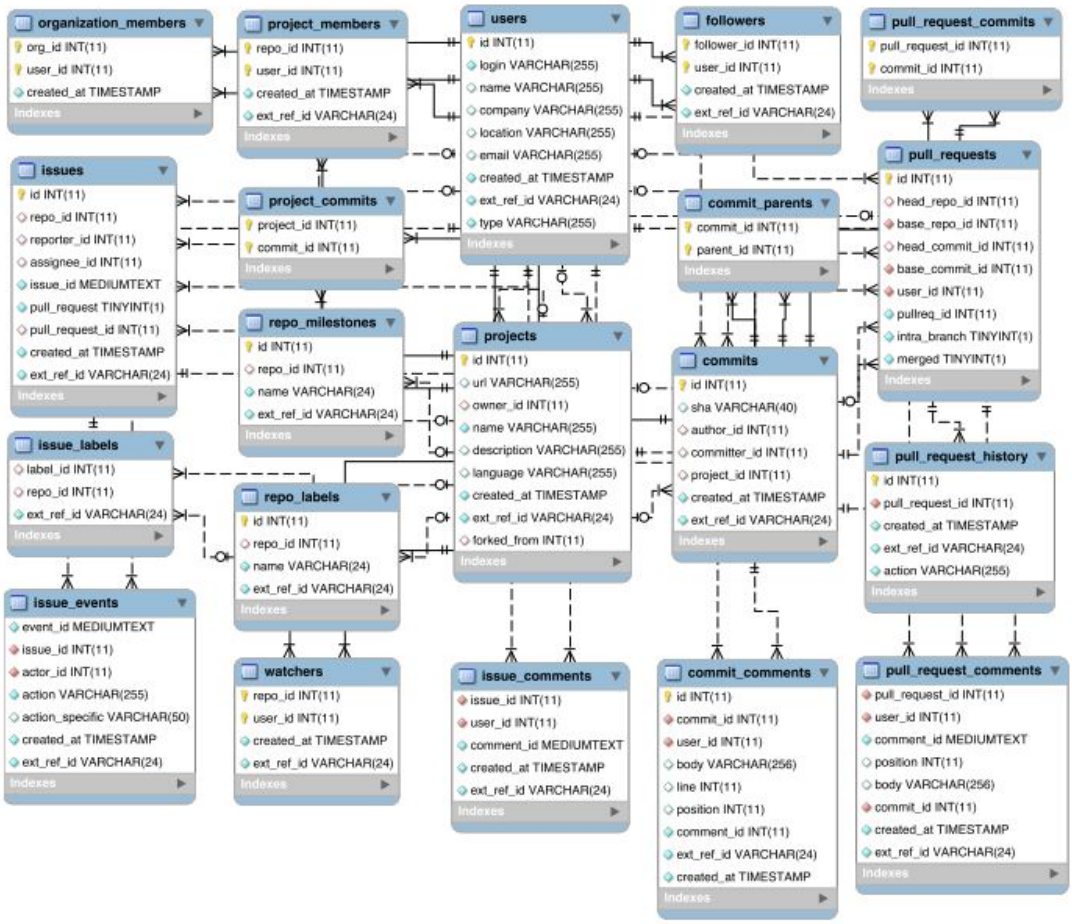


Table 3.1: Description of database entities used in this study and Bao et al., (2019).

Entity	Description
projects	GitHub repository
users	GitHub users
commits	List of all commits for each project
commit_comments	Comments for each commit
issues	Issues recorded for each project
issue_comments	Comments on an issue
issue_events	List of events on an issue, e.g., opened, assigned, closed
pull_requests	List of pull requests for each project
pull_request_comments	Comments on a pull request
pull_request_history	List of events on a pull request, e.g., opened, merged
watchers	Users that have starred a project
followers	Users that are following another user

3.2 Dataset

In this study, the dataset is constructed using the same procedure used by Bao et al., (2019). The dataset in this research has 888 repositories extracted from a GHTorrent MySQL database snapshot taken on 09/01/2017. These 888 repositories are obtained by selecting the top 1,000 repositories sorted by decreasing number of stars[6] and excluding repositories that do not have a programming language specified, do not use GitHub as the issue tracker, are forked from another repository, have been deleted from GitHub, or are duplicate repositories (see Figure 3.4).

Dataset used by Bao et al., (2019) has 917 repositories, while the dataset used in this research has 888 repositories (29 fewer repositories). The 29 repositories with repeated repository URLs are shown in table 3.2. To resolve the repeated URLs issue, four solutions were considered: (1) keep the earliest repository and drop the later repository, (2) keep the later repository and drop the earliest repository, (3) Keep earliest or latest but merge the contributors, and (4) remove all the repeated repositories as there are enough points in the

[6]Developers can keep track of a GitHub repository by starring it.

Figure 3.4. Data collection process used in this research.

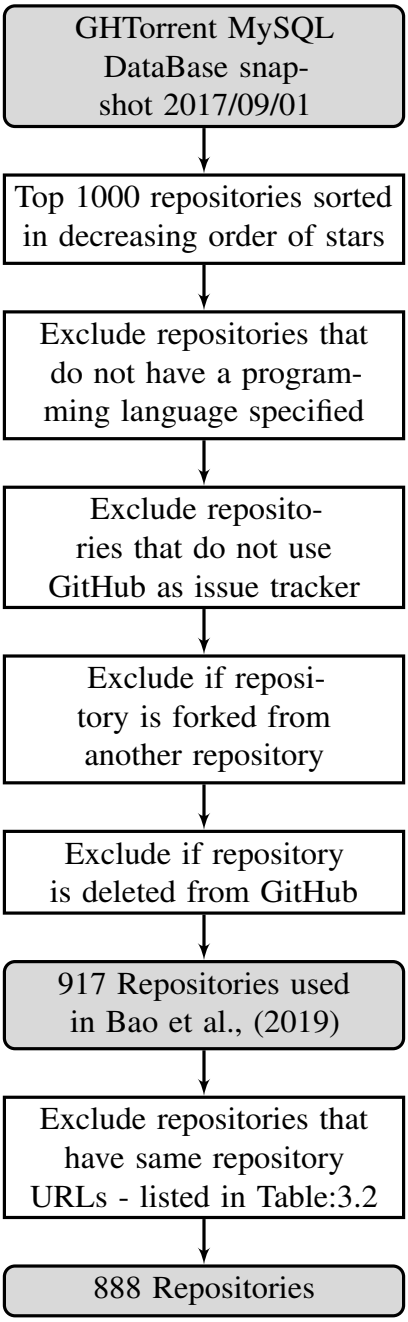


Table 3.2: Repositories with repeated/duplicate URLs in Bao et al., (2019) dataset.

#	repo_id	URL
1 2	9626384 18904813	https://api.github.com/repos/atom/electron
3 4	19676846 23560871	https://api.github.com/repos/MostlyAdequate/mostly-adequate-guide
5 6	28965927 2558	https://api.github.com/repos/zurb/foundation-sites
7 8	2691 36897540	https://api.github.com/repos/d3/d3
9 10	17302594 4095	https://api.github.com/repos/elastic/elasticsearch
11 12	20078281 13345616	https://api.github.com/repos/nodejs/node
13 14	6900230 27601818	https://api.github.com/repos/vuejs/vue
15 16	22091370 10200435	https://api.github.com/repos/kubernetes/kubernetes
17 18	22041940 9861851	https://api.github.com/repos/socketio/socket.io
19 20	22045218 3894	https://api.github.com/repos/jkbrzt/httpie
21 22	1085 34229160	https://api.github.com/repos/pallets/flask
23 24	22101095 32118325	https://api.github.com/repos/reactjs/redux
25 26	40446271 29	https://api.github.com/repos/opencv/opencv
27 28 29	5230 30988985 11072234	https://api.github.com/repos/expressjs/express

dataset. Option four was chosen, as the dataset is large enough for the experiments to be meaningful, leaving with 888 repositories.

Next, for each repository, contributors who submitted at least one commit after their GitHub account creation time are considered. LTC is defined as a contributor who continues to commit to the repository for more than a time T. Three settings for T are used, namely, T = 1 year, 2 years, and 3 years. Table 3.3 shows the dataset class distributions of LTC and non-LTC contributors for the three time intervals. The dataset is imbalanced, with LTC ratios of 12.30%, 8.04%, and 5.98% when the time interval is 1, 2, and 3 years, respectively.

Table 3.3: Statistics of selected LTC and non-LTC data.

Year	Total	# LTC	# non-LTC
1	70,899	8,774 (12.38%)	62,125 (87.62%)
2	70,899	3,823 (5.39%)	67,076 (94.61%)
3	70,899	1,554 (2.19%)	69,345 (97.81%)

Table 3.4: Dataset 1: Structure of the data used in this study.

#	Repo_id	Contributor_id	Join_Date	is_LTC_1	is_LTC_2	is_LTC_3	Feature 1	...	Feature 31
1	6	28	3-12-2012	FALSE	FALSE	FALSE	...	...	...
2	6	32	12-15-2011	TRUE	TRUE	TRUE	...	...	...
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
70,899	...	...	...	...	...	...	...	...	...

3.3 Features

3.3.1 Feature Set 1

A dataset containing 70,899 observations from 888 repositories with 56,766 contributors is constructed. Table 3.4 shows the organization of the dataset. Each observation (row) represents a contributor who joined the repository and is categorized as either an LTC or a non-LTC, depending on whether their project tenure is longer than a time T. Three settings for T are used, namely, T = 1 year, 2 years, and 3 years. For each observation, 31 features related to repositories and contributors from GitHub and GHTorrent are extracted and organized along two dimensions: Repository Features (r_) and Contributor Features (c_). These 31 features comprise 13 repository features and 18 contributor features, as shown in Tables 3.5 and 3.6. Table 3.7 shows descriptive statistics for the numerical features. GitHub REST API is used to extract four features, and BigQuery is used to extract the remaining 27 features.

Table 3.5: Feature Set 1: Repository features (r_) used in this study.

Feature Name	Feature Description	References
r_Language	Main programming language used in the repository	Bao et al., (2019), Jiang et al., (2017), Jiang et al., Krein et al., (2019, 2010)
r_has_Dependents	Has the repository depended on by other repositories	
r_Age	Number of days between repository creation date and new contributor join date	Vasilescu et al., (2015), Borges and Valente, Coelho et al., (2018, 2020), Jarczyk et al., (2018)
r_Number_of_Core_Members	Number of members who have the right to merge and commit to the repository	Nielek et al., (2016)
r_Number_of_Watchers	Number of GitHub members watching the repository	Bao et al., (2019), Dabbish et al., (2012), Borges and Valente, (2018)
r_Number_of_Contributors	Number of contributors to the repository	Lima, Rossi, and Musolesi, (2014), Tsay, Dabbish, and Herbsleb, (2014), Bao et al., (2019), Borges and Valente, (2018)
r_Min_All_Contributor_Commits	Minimum number of commits by any contributor from all contributors to the repository	
r_Max_All_Contributor_Commits	Maximum number of commits by any contributor from all contributors to the repository	
r_Min_All_Contributor_Stay	Minimum time of any contributor from all contributors engaged in the contribution to this repository	
r_Max_All_Contributor_Stay	Maximum time of any contributor from all contributors engaged in the contribution to this repository	
r_has_Contribution_Invitation	The repository has an invitation to contribute in README.md file	Prana et al., (2019), Kobayakawa and Yoshida, (2017), Elazhary et al., (2019)
r_has_Contribution_Rules	The repository has contribution rules documented in CONTRIBUTION.md or README.md files	Prana et al., (2019), Kobayakawa and Yoshida, (2017), Elazhary et al., (2019)
r_Readability_Kincaid_Score	The Flesch—Kincaid score to measure documentation readability of README.md file. The Flesch—Kincaid formula Kincaid et al., (1975), measures readability, with a lower score indicating poor readability.	

Table 3.6: Feature Set 1: Contributor features (c_) used in this study.

Feature Name	Feature Description	References
c_is_Fake	Is the user fake	Jarczyk et al., (2018)
c_is_Deleted	Is the contributor account deleted	
c_Age	Number of days between contributor GitHub join date and repository join date	Bao et al., (2019)
c_is_Core_Member	Does the contributor have the right to merge and commit to the repository	
c_is_Watching_Project	Is the new contributor watching the repository when they joined the repository	Sheoran et al., (2014)
c_Own_Repos_Lang	Number of repositories the contributor owns with the same implementation language as the repository the contributor joined	Hu et al., (2016), Nielek et al., (2016), Coelho et al., (2018)
c_Watch_Repos_Lang	Number of repositories the contributor watches with the same implementation language as the repository the contributor joined	Lima, Rossi, and Musolesi, (2014), Hu et al., (2016)
c_Contribute_Repos_Lang	Number of repositories the contributor contributes to with the same implementation language as the repository the contributor joined	Lima, Rossi, and Musolesi, (2014), Hu et al., (2016)
c_History_Commits_Lang	Number of commits the contributor has submitted to repositories with the same implementation language as the repository the contributor joins	Coelho et al., (2020)
c_History_Pull_Request_Lang	Number of pull requests the contributor has submitted to repositories with the same implementation language as the repository the contributor joins	Cosentino, Izquierdo, and Cabot, (2017),Coelho et al., (2020),Elazhary et al., (2019)
c_History_Issues_Lang	Number of issues the contributor has submitted to repositories with the same implementation language as the repository the contributor joins	Cosentino, Izquierdo, and Cabot, (2017),Coelho et al., (2020)
c_History_Followers	Number of GitHub members who follow the new contributor before the repository join date	Bao et al., (2019), Hu et al., (2016), Nielek et al., (2016), Blincoe et al., (2016)
c_History_Following	Number of GitHub members the new contributor is following before the repository join date	Hu et al., (2016), Nielek et al., (2016), Blincoe et al., (2016)
c_Number_of_Repos	Number of repositories the new contributor contributed	Nielek et al., (2016), Bao et al., (2019)
c_Min_All_Repos_Commits	Minimum number of commits among all repositories the new contributor contributed	Jarczyk et al., (2018), Hu et al., (2018), Coelho et al., (2018)
c_Max_All_Repos_Commits	Maximum number of commits among all repositories the new contributor contributed	Jarczyk et al., (2018), Hu et al., (2018), Coelho et al., (2018)
c_Min_All_Repos_Stay	Minimum time the contributor was engaged with any repository	
c_Max_All_Repos_Stay	Maximum time the contributors was engaged with any repository	

Table 3.7: Feature Set 1: Descriptive statistics for repository and contributor features.

Feature	Quant_5	Mean	Median	Quant_95	Histogram
r_Age	48.00	1027.81	952.00	2240.00	
r_Number_of_Core_Members	0.00	11.91	3.00	37.00	
r_Number_Of_Watchers	96.00	7113.47	4491.00	24693.40	
r_Number_of_Contributors	5.00	323.36	110.00	1531.00	
r_Min_All_Contributor_Commits	1.00	1.48	1.00	1.00	
r_Max_All_Contributor_Commits	23.00	1876.21	884.00	7600.20	
r_Min_All_Contributor_Stay	0.08	0.08	0.08	0.08	
r_Max_All_Contributor_Stay	0.17	2.64	2.08	7.17	
r_Readability_Kincaid_Score	1.20	4.50	4.11	10.14	
c_Age	62.00	1104.25	1057.00	2340.00	
c_Own_Repos_Lang	0.00	9.24	2.00	37.00	
c_Watch_Repos_Lang	0.00	19.84	2.00	88.00	
c_Contribute_Repos_Lang	0.00	14.71	3.00	52.00	
c_newfigs/history_Commits_Lang	0.00	236.33	13.00	910.00	
c_newfigs/history_Pull_Request_Lang	0.00	39.28	1.00	54.00	
c_newfigs/history_Issues_Lang	0.00	13.75	1.00	60.00	
c_newfigs/history_Followers	0.00	76.92	11.00	237.00	
c_newfigs/history_Following	0.00	27.55	4.00	97.00	
c_Number_of_Repos	0.00	53.22	14.00	141.00	
c_Min_All_Repos_Commits	0.00	1.64	1.00	3.00	
c_Max_All_Repos_Commits	0.00	178.81	42.00	648.00	
c_Min_All_Repos_Stay	0.00	0.09	0.08	0.08	
c_Max_All_Repos_Stay	0.00	1.38	0.83	4.33	

3.3.1.1 Repository (r_) Feature Dimension

This dimension captures repository information based on historical activities in GitHub from all contributors before a new contributor submits their first contribution to the repository. This dimension is closely related to the macro-climate dimension used by Zhou and Mockus, (2015) and the repository activity dimension used by Bao et al., (2019). The following 13 features related to the repository are extracted that are hypothesized to contribute to LTC conversion.

Feature r_Language (Bao et al., 2019, Jiang et al., 2017, Jiang et al., 2019, and Krein et al., 2010) captures the main programming language used in the repository. Feature r_has_Dependents evaluates whether other repositories are dependent on the current repository. The insight here is that a repository with dependents would probably have more users who need to contribute.

Feature r_Age (Vasilescu et al., 2015, Borges and Valente, 2018, Coelho et al., 2020, and Jarczyk et al., 2018) calculates the repository age as the number of days between repository

creation date and new contributor join date. Repository age is used as a proxy for project maturity.

Feature r_Number_of_Core_Members (Nielek et al., 2016) counts the number of members who have the right to merge and commit to the repository. The insight here is that more core members will likely translate to more resources to merge contributions by newcomers and convert them to LTCs.

Feature r_Number_of_Watchers (Bao et al., 2019, Dabbish et al., 2012, and Borges and Valente, 2018) tracks the number of GitHub members watching the repository, which is shown to be a useful measure of the popularity of the repository by Borges and Valente, (2018).

Feature r_Number_of_Contributors (Lima, Rossi, and Musolesi, 2014, Tsay, Dabbish, and Herbsleb, 2014, Bao et al., 2019, and Borges and Valente, 2018) counts the number of contributors of the repository before the new contributor joined, which is shown to have a moderate correlation with popularity of the repository by Borges and Valente, (2018).

Feature r_Min_All_Contributor_Commits tabulates the number of commits committed by each contributor to this repository before the new contributor joined and computes the minimum number of commits from all contributors to the repository. Feature r_Max_All_Contributor_Commits tabulates the number of commits committed by each contributor to this repository before the new contributor joined and computes the maximum number of commits from all contributors to the repository.

Feature r_Min_All_Contributor_Stay tabulates the number of years each contributor engaged with this repository before the new contributor joined and computes the minimum number of years from all contributors to the repository. Feature r_Max_All_Contributor_Stay tabulates the number of years each contributor engaged with this repository before the new contributor joined and computes the maximum number of years from all contributors to the repository.

r_Min_All_Contributor_Commits, r_Max_All_Contributor_Commits,

r_Min_All_Contributor_Stay, and r_Max_All_Contributor_Stay features capture the commit and engagement pattern of all contributors to this repository.

Feature r_has_Contribution_Invitation scans the README.md file in the repository to check for documentation related to contribution invitation (Prana et al., 2019, Kobayakawa and Yoshida, 2017, and Elazhary et al., 2019).

Feature r_has_Contribution_Rules scans CONTRIBUTION.md and README.md files in the repository to check for the presence of documentation related to contributions rules (Prana et al., 2019, Kobayakawa and Yoshida, 2017, and Elazhary et al., 2019).

Feature r_Readability_Kincaid_Score computes readability of README.md file in the repository using Kincaid score as metric to evaluate. The Flesch—Kincaid formula (Kincaid et al., 1975), measures readability, with a lower score indicating poor readability.

r_has_Contribution_Rules, r_has_Contribution_Invitation, and r_Readability_Kincaid_Score are proxies for measuring the quality of invitation, instructions to contribute, and clear documentation, respectively.

3.3.1.2 Contributor Feature (c_) Dimension

This dimension captures contributor information based on the historical activities of the contributor in all the GitHub repositories before the new contributor submits their first contribution to the current repository. This dimension is closely related to the developer/developer activity dimension used by Bao et al., (2019). The following 18 features related to the new contributor are extracted that are believed to be indicative of contributor experience, activeness, and willingness to contribute to OSS in GitHub.

Features c_is_Fake (Jarczyk et al., 2018) and c_is_Deleted capture if the contributor is tagged by GitHub as fake or deleted, respectively. These are used with the belief that fake and deleted contributors would stop contributing after a while.

Feature c_Age (Bao et al., 2019) calculates the contributor age as the number of days between contributor GitHub join date and repository join date. Contributor age is used as a

proxy for contributor experience.

Feature c_is_Core_Member identifies if the new contributor has the right to merge and commit to the repository. The insight here is that a new contributor admitted as core member will likely stay with the repository long term.

Feature c_is_Watching_Project (Sheoran et al., 2014) identifies if the new contributor was watching the repository before joining. A contributor watching the repository is a potential indicator that the new contributor was interested in the repository and would probably continue to be interested after joining the repository.

The next six features are extracted in an attempt to capture developer experience and interests in the implementation language of the repository they join - c_Own_Repos_Lang, c_Watch_Repos_Lang, c_History_Issues_Lang,
c_Contribute_Repos_Lang, c_History_Commits_Lang,
c_History_Pull_Request_Lang.

c_Own_Repos_Lang (Hu et al., 2016, Nielek et al., 2016, and Coelho et al., 2018) counts the number of repositories the contributor owns. c_Watch_Repos_Lang captures the number of repositories the contributor watches (Lima, Rossi, and Musolesi, 2014, Hu et al., 2016). c_Contribute_Repos_Lang calculates the number of repositories the contributor contributed (Lima, Rossi, and Musolesi, 2014, Hu et al., 2016).

c_History_Commits_Lang computes the number of commits the contributor has submitted to repositories (Coelho et al., 2020). c_History_Pull_Request_Lang extract the number of pull requests the contributor has submitted to repositories (Cosentino, Izquierdo, and Cabot, 2017, Coelho et al., 2020, and Elazhary et al., 2019). c_History_Issues_Lang counts the number of issues the contributor has submitted to repositories (Cosentino, Izquierdo, and Cabot, 2017, Coelho et al., 2020).

The key insight is that an experienced GitHub contributor is more likely to stay with an OSS project if the implementation language matches their dominant language. This insight is based on this researcher's personal experience (the preference is to contribute to

a Java language project and continue to do so compared to a Python project as Java is my dominant language). This insight is supported by other researchers' previous work (Jiang et al., 2017), where those authors found that contributors prefer forking projects implemented in their familiar programming language. Jiang et al., (2019) defined a programming language similarity measure and showed that unfollowing behavior is correlated with a lower similarity score of contributors. Krein et al., (2010) presented a language entropy metric to characterize developer programming effort over various programming languages and show that language fragmentation is inversely correlated with the code quantity contributed by the OSS community.

The features c_History_Followers and c_History_Following (Bao et al., 2019, Hu et al., 2016, Nielek et al., 2016, and Blincoe et al., 2016) capture the number of developers following the current contributor and the number of other contributors this contributor is following respectively. These features are potential proxies for the social ability of the contributor.

Feature c_Number_of_Repos (Nielek et al., 2016, Bao et al., 2019) counts the number of repositories the new contributor contributed to before joining the current repository — a proxy for experience in contributing to OSS repositories.

Feature c_Min_All_Repos_Commits (Jarczyk et al., 2018, Hu et al., 2018, and Coelho et al., 2018) finds the minimum number of commits the new contributor made to any repository before joining the current repository. Feature c_Max_All_Repos_Commits finds the maximum number of commits the new contributor made to any repository before joining the current repository. These features are proxies for the amount of effort spent by the contributor to OSS repositories.

Features c_Min_All_Repos_Stay computes the minimum number of years the new contributor is engaged with any repository before joining the current repository. Feature c_Max_All_Repos_Stay computes the maximum number of years the new contributor is engaged with any repository before joining the current repository. These features are proxies

Table 3.8: Dataset 2: Structure of the data used in this study.

#	Repo_id	Contributor_id	Join_Date	is_LTC_1	is_LTC_2	is_LTC_3	Feature 1	...	Feature 173
1	6	28	3-12-2012	FALSE	FALSE	FALSE	...	...	...
2	6	32	12-15-2011	TRUE	TRUE	TRUE	...	...	...
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
70,899	...	...	...	...	...	...	...	...	...

for the length of contributor's engagement to OSS repositories.

3.3.2 Feature Set 2

The dataset containing 70,899 observations from 888 repositories with 56,766 contributors is constructed. Table 3.8 shows the organization of the dataset. Each observation (row) represents a contributor who joined the repository and is categorized as either an LTC or a non-LTC, depending on whether their project tenure is longer than a time T. Three settings for T are used: T = 1 year, 2 years, and 3 years. For each observation, the 63 features from previous research (Bao et al., 2019) are augmented with an additional 110 features extracted from GHTorrent using BigQuery.

Features extracted from GHTorrent using BigQuery are backed by relational database schema, and the entities used in this study are described in Table 3.1. The features used in this study are organized along five dimensions: DP, RP, DA, RA, and CN. Refer to Tables [3.9 and 3.10] for a summary of each feature.

3.3.2.1 DP Dimension

This dimension captures developer information based on the historical activities of the developer in the GitHub repository. Bao et al., (2019) extracted eight features – user_age, user_own_repos, user_watch_repos,
user_contribute_repos, user_history_commits, user_history_issues,
user_history_pull_requests, and user_history_followers. These features might indicate developer experience, activeness, and willingness to contribute to OSS in GitHub.

Table 3.9: Feature Set 2: DP features and RP features.

Dim	Factor Name	Explanation
DP	user_age	Number of days between developer GitHub join date and repository join date
	user_own_repos	Number of repositories the developer owns before repository join date
	user_own_repos_lang	user_own_repos with the same implementation language as the repository developer joins
	user_own_repos_ratio	user_own_repos_lang / user_own_repos
	user_watch_repos	Number of repositories the developer watches before repository join date
	user_watch_repos_lang	user_watch_repos with the same implementation language as the repository developer joins
	user_watch_repos_ratio	user_watch_repos_lang / user_watch_repos
	user_contribute_repos	Number of repositories the developer has submitted at least one commit before repository join date
	user_contribute_repos_lang	user_contribute_repos with the same implementation language as the repository developer joins
	user_contribute_repos_ratio	user_contribute_repos_lang / user_contribute_repos
	user_history_commits	Number of commits the developer has submitted before repository join date
	user_history_commits_lang	user_history_commits with the same implementation language as the repository developer joins
	user_history_commits_ratio	user_history_commits_lang / user_history_commits
	user_history_pull_requests	Number of pull requests the developer has submitted before repository join date
	user_history_pull_requests_lang	user_history_pull_requests with the same implementation language as the repository developer joins
	user_history_pull_requests_ratio	user_history_pull_requests_lang / user_history_pull_requests
	user_history_issues	Number of issues the developer has submitted before repository join date
	user_history_issues_lang	user_history_issues with the same implementation language as the repository developer joins
	user_history_issues_ratio	user_history_issues_lang / user_history_issues
	user_history_followers	Number of users who follow the new developer before repository join date
	user_history_following	Number of users followed by the new developer before repository join date
RP	Language	Main programming language used by the repository
	repo_age	Number of days between repository creation date and developer repository join date
	before_repo_commits	Number of commits that the repository has before the new developer joins
	{X}_month_before_repo_commits	Number of commits that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_commit_comments	Number of commit comments that the repository has before the new developer joins
	{X}_month_before_repo_commit_comments	Number of commit comments that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_contributors	Number of contributors that the repository has before the new developer joins
	{X}_month_before_repo_contributors	Number of contributors that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_contributor_{S}	Statistics of commits of contributors in the repository, where S can be max, min, mean, median, and std
	before_repo_issues	Number of issues that the repository has before the new developer joins
	{X}_month_before repo_issues	Number of issues that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_issue_comments	Number of issue comments that the repository has before the new developer joins
	{X}_month_before_repo_issue_comments	Number of issue comments that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_issue_events	Number of issue events that the repository has before the new developer joins
	{X}_month_before_repo_issue_events	Number of issue events that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_issue_events_closed	Number of issue closed events that the repository has before the new developer joins
	{X}_month_before_repo_issue_events_closed	Number of issue closed events that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_issue_events_assigned	Number of issue assigned events that the repository has before the new developer joins
	{X}_month_before_repo_issue_events_assigned	Number of issue assigned events that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_pull_requests	Number of pull requests that the repository has before the new developer joins
	{X}_month_before_repo_pull_requests	Number of pull requests that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_pull_request_comments	Number of pull request comments that the repository has before the new developer joins
	{X}_month_before_repo_pull_request_comments	Number of pull request comments that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_pull_request_history	Number of pull request events the repository has before the new developer joins
	{X}_month_before_repo_pull_request_history	Number of pull request events that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_pull_request_history_merged	Number of merged pull request events the repository has before the new developer joins
	{X}_month_before_repo_pull_request_history_merged	Number of merged pull request events that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_pull_request_history_closed	Number of closed pull request events the repository has before the new developer joins
	{X}_month_before_repo_pull_request_ history_closed	Number of closed pull request events that the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	before_repo_watchers	Number of watchers the repository has before the new developer joins
	{X}_month_before_repo_watchers	Number of watchers the repository has X month(s) before the new developer joins, where X= 1, 2, 3
	{X}_month_after_repo_watchers	Number of watchers the repository has X month(s) after the new developer joins, where X= 1, 2, 3

Table 3.10: Feature Set 2: DA features, RA features, and CN features.

Dim	Factor Name	Explanation
DA	{X}_month_user_commits	Number of commits the new developer submits to the repository in the first X month(s), where X= 1, 2, 3
	{X}_month_user_commit_comments	Number of comments received to commits submitted by the new developer in the first X month(s), where X= 1, 2, 3
	month_user_issues	Number of issues the new developer submits to the repository in the first month
	{X}_month_user_issues_fixed	Number of issues the new developer submits to the repository in the first X month(s), where X= 1, 2, 3
	{X}_month_user_issue_comments	Number of comments received to the issues submitted by the new developer in the first X month(s), where X= 1, 2, 3
	{X}_month_user_issue_events	Number of events received to the issues the new developer submitted to the repository in the first X month(s), where X= 1, 2, 3
	{X}_month_user_issue_events_closed	Number of closed events received to the issues submitted by the new developer in the first X month(s), where X= 1, 2, 3
	{X}_month_user_issue_events_assigned	Number of assigned events received to the issues submitted by the new developer in the first X month(s), where X= 1, 2, 3
	{X}_month_user_pull_requests	Number of pull request submitted by the new developer to the repository in the first X month(s), where X= 1, 2, 3
	{X}_month_user_pull_request_comments	Number of comments received to the pull request submitted by the new developer in the first X month(s), where X= 1, 2, 3
	{X}_month_user_pull_request_history	Number of pull request events received to the pull request submitted by the new developer in the first X month(s), where X= 1, 2, 3
	{X}_month_user_pull_request_history_merged	Number of merged pull request events received to the pull request submitted by the new developer in the first X month(s), where X= 1, 2, 3
	{X}_month_user_pull_request_history_closed	Number of closed pull request events received to the pull request submitted by the new developer in the first X month(s), where X= 1, 2, 3
RA	{X}_month_repo_commits	Number of commits submitted to the repository in the first X month(s) after the developer joins, where X= 1, 2, 3
	{X}_month_repo_commit_comments	Number of comments received to the commits submitted to the repository in the first X month(s) after the developer joins, where X= 1, 2, 3
	{X}_month_repo_contributors	Number of contributors who submitted at least one commits to the repository in the first X month(s) after the developer joins, where X= 1, 2, 3
	month_repo_contributor_{S}	Statistics of commits of contributors in the first month, where S can be max, min, mean, median, and std
	{X}_month_repo_issues	Number of issues submitted to the repository in the first X month(s) after the developer joins, where X= 1, 2, 3
	{X}_month_repo_issue_comments	Number of comments received to the issues submitted to the repository in the first X month(s) after the developer joins, where X= 1, 2, 3
	{X}_month_repo_issue_events	Number of events received to the issues submitted to the repository in the first X month(s) after new developer joins, where X= 1, 2, 3
	{X}_month_repo_issue_events_closed	Number of closed events received in the issues submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
	{X}_month_repo_issue_events_assigned	Number of assigned events received in the issues submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
	{X}_month_repo_pull_requests	Number of pull requests submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
	{X}_month_repo_pull_request_comments	Number of comments received in the pull requests submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
	{X}_month_repo_pull_request_history	Number of events received in the pull requests submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
	{X}_month_repo_pull_request_history_merged	Number of merged events received in the pull requests submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
	{X}_month_repo_pull_request_history_closed	Number of closed events received in the pull requests submitted to the repository in the first X month(s) that the new developer joins, where X= 1, 2, 3
CN	degree_centrality	These metrics are used to quantify a new contributor's degree of activity in the collaboration structure of an OSS project
	closeness_centrality	
	betweenness_centrality	
	eigenvector_centrality	
	clustering_coefficient	

The above eight features are augmented with 13 additional features in an attempt to capture developer experience and interests in the implementation language of the repository they join - user_own_repos_lang, user_own_repos_ratio, user_watch_repos_lang, user_watch_repos_ratio, user_contribute_repos_lang, user_contribute_repos_ratio, user_history_commits_lang, user_history_commits_ratio, user_history_pull_requests_lang, user_history_pull_requests_ratio, user_history_issues_lang, user_history_issues_ratio, user_history_following. The new "_lang" feature (example new feature: user_own_repos_lang) is a restriction of the baseline feature (example baseline feature: user_own_repos) to the implementation language of the repository the developer joined. The new "_ratio" feature is the ratio of the new "_lang" feature and the original feature, indicating developer skill fragmentation. These are the same features added in feature set 1.

The feature user_history_following is the number of other developers this developer is following and is similar to the baseline user_history_followers feature, which is the number of developers following the current developer. These features are potential proxies for the

social ability of the contributor.

3.3.2.2 RP Dimension

This dimension is closely related to the macro-climate dimension used by Zhou and Mockus, (2015) and captures repository information based on historical activities in GitHub from all developers before a new contributor submits their first commit to the repository. Bao et al., (2019) extracted 20 features: Language, before_repo_commits, before_repo_commit_comments, before_repo_contributors, before_repo_contributor_max, before_repo_contributor_min, before_repo_contributor_mean, before_repo_contributor_median, before_repo_contributor_std, before_repo_issues, before_repo_issue_comments, before_repo_issue_events, before_repo_issue_events_closed, before_repo_issue_events_assigned, before_repo_pull_requests, before_repo_pull_request_comments, before_repo_pull_request_history, before_repo_pull_request_history_merged, before_repo_pull_request_history_closed, and before_repo_watchers. These features represent the complexity of the repository, since they capture a snapshot of all activity by all developers on the repository from inception to the time the new contributor joins.

The above 20 features are augmented with 46 additional features: 42 of the 46 features are added in an attempt to augment the snapshot of all activity of all developers with the most recent activity of all developers one month, two months, and three months before the new contributor joins the repository. These features are added to quantify development activity before a new contributor joins the repository. The insight here is that recent repository activity probably captures the churn/maturity of the repository. The new "one_month_" (example: one_month_before_repo_commits), "two_month_" (example: two_month_before_repo_commits), and "three_month_" (example: three_month_before_repo_commits) features are restrictions on the baseline (example: before_repo_commits) feature to one month, two months, and three months before the new

contributor joins the project. In this research, repo_age is added to quantify the number of days between the repository creation date and when the new contributor joins the repository. All of these measures are functions of how long the project has been in existence. In addition, three features one_month_after_repo_watchers, two_month_after_repo_watchers, three_month_after_repo_watchers are added to capture the level of interest in the project after a new contributor joins.

In previous work, Zhou and Mockus, (2015) considered predictors over various time intervals (one week to six months). They found that less than one-month intervals did not yield good predictions and that more than one-month intervals did not substantially improve the predictions. While Zhou and Mockus, (2015) did not improve predictions with longer intervals, features at a month granularity are added in this research. Feature selection method is used to identify important features, as explained later in the feature selection section.

3.3.2.3 DA Dimension

This dimension quantifies new contributor development activity in the repository after joining. Zhou and Mockus, (2015) reported that the initial effort that a new developer contributes to an OSS project is a good predictor of becoming an LTC. Based on the work of Zhou and Mockus, (2015), Bao et al., (2019) extracted 12 features: month_user_commits, month_user_commit_comments, month_user_issues, month_user_issue_comments, month_user_issue_events, month_user_issue_events_closed, month_user_issue_events_assigned, month_user_pull_requests, month_user_pull_request_comments, month_user_pull_request_history, month_user_pull_request_history_merged, and month_user_pull_request_history_closed. Bao et al., (2019) hypothesized that these features represent the first impression of a new contributor toward the repository. For instance, merging the initial pull request of a new contributor into the repository would give the contributor a sense of accomplishment and increase their willingness to make more

contributions to the project.

The above 12 features are augmented with 24 additional features based on the work of Zhou and Mockus, (2015), as stated previously. These 24 features are added to quantify new contributor development activity in the first two and three months after joining the repository. The new "two_" (example: two_month_user_commits) and "three_" (example: three_month_user_commits) features are extensions of the baseline features (example: month_user_commits) to two months and three months after the new contributor joins the project.

As part of this, an issue with the calculation of month_user_issues in Bao et al., (2019) is identified, where,
month_user_issues and month_user_issue_comments had the same value for all observations. This issue was fixed, and a new feature named month_user_issues_fixed is added.

As part of the research in this dissertation, an issue with the calculation of month_user_issues was identified — both month_user_issues and month_user_issue_comments had the same value for all observations. The identified issue was fixed by adding a new feature named month_user_issues_fixed.

3.3.2.4 RA Dimension

This dimension is closely related to the RP dimension; whereas RP looks at the past, RA looks at the future three months after a new contributor joins the project. This dimension captures repository information based on activity in GitHub from all developers after a new contributor submits their first commit to the repository. Bao et al., (2019) extracted 18 features: month_repo_commits, month_repo_commit_comments,
month_repo_contributors, month_repo_contributor_min,
month_repo_contributor_max,month_repo_contributor_mean,
month_repo_contributor_median, month_repo_contributor_std,
month_repo_issues, month_repo_issue_comments,

month_repo_issue_events, month_repo_issue_events_closed,
month_repo_issue_events_assigned, month_repo_pull_requests,
month_repo_pull_request_comments, month_repo_pull_request_history,
month_repo_pull_request_history_merged, and
month_repo_pull_request_history_closed. These features represent the speed at which the repository is changing, since they capture all activity by all developers on the repository after the new contributor joins.

The above 18 features are augmented with 26 additional features to capture all activity of all developers two months and three months after the new contributor joins the repository. The new "two_" (example: two_month_repo_commits) and "three_" (example: three_month_repo_commits) features are extensions of the baseline feature (example: month_repo_commits) from one month to two months and three months after the new contributor joins the project. The reason for extending the RA baseline features is similar to that for extending the RP baseline features.

3.3.2.5 CN Dimension

Bao et al., (2019) extracted five contributor collaboration activity features:
degree_centrality, closeness_centrality, betweenness_centrality,
eigenvector_centrality, and clustering_coefficient, which are used without any modification.

3.3.3 Feature Set 3 (with Feature Selection)

The feature selection method is used to retain relevant features and eliminate irrelevant features. As explained in Alpaydin, (2014), reducing the number of features reduces the model building time complexity, space complexity, and extraction cost. Feature selection is used to reduce the number of features from 173 to 55. Table 3.11 shows the organization of the dataset. Feature selection is widely used in software engineering (Bao et al., 2019, Fan et al., 2018, and Tantithamthavorn et al., 2018) to eliminate correlated features. The feature

Table 3.11: Dataset 3: Structure of the data used in this study.

#	Repo_id	Contributor_id	Join_Date	is_LTC_1	is_LTC_2	is_LTC_3	Feature 1	...	Feature 55
1	6	28	3-12-2012	FALSE	FALSE	FALSE	...	...	...
2	6	32	12-15-2011	TRUE	TRUE	TRUE	...	...	...
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
70,899	...	...	...	...	...	...	...	...	...

selection used in this study has three main steps described below and executed in sequence, as shown in Fig. 3.5.

3.3.3.1 Normality Adjustment

The distributions of the 173 features are right-skewed. Based on the work of Tantithamthavorn et al., (2018), each feature is log-transformed to mitigate skew using $ln(x_f + 1)$, where x_f is the value of feature f.

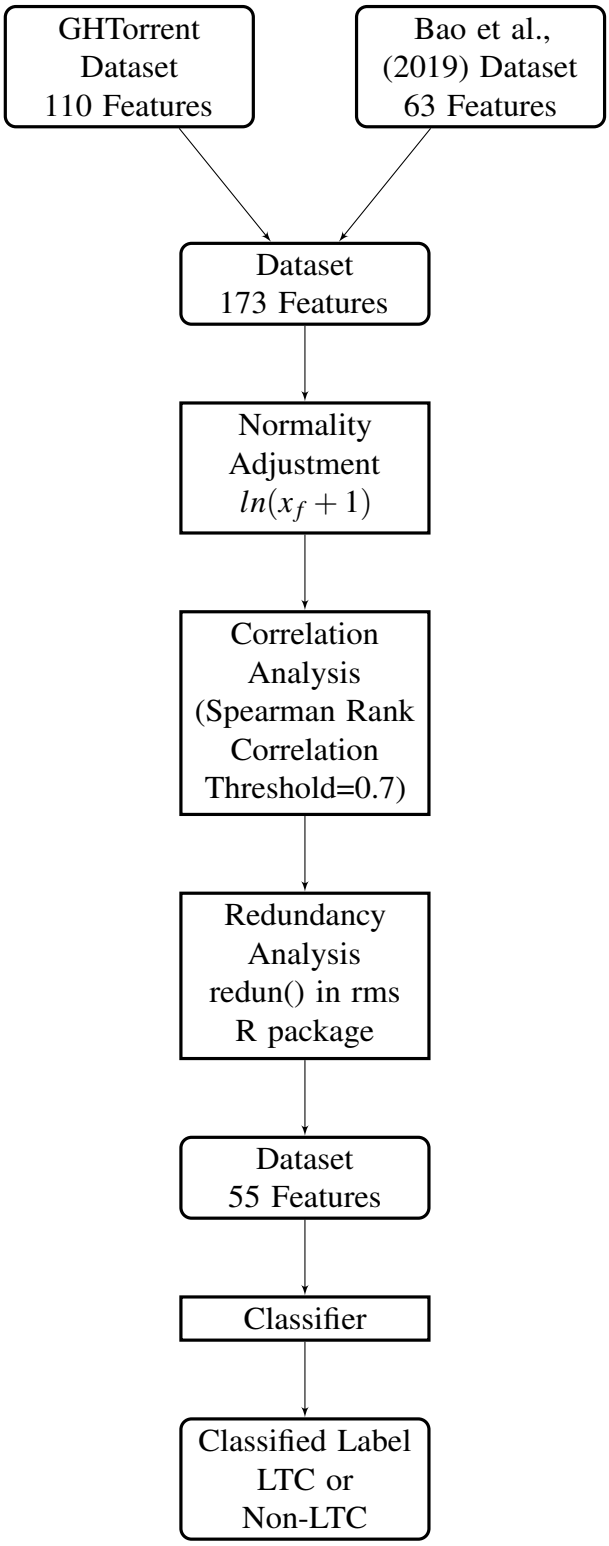
3.3.3.2 Correlation Analysis

As performed in Bao et al., (2019), the correlations among the features are found by constructing hierarchical clustering using the *varclus* function from the *Hmisc* R package. The *varclus* function groups correlated features together as a hierarchical tree. Additionally, the Spearman rank correlation is used as a threshold measure to identify highly correlated groups (threshold set to 0.7). All features, except one randomly selected feature, are removed from each group. This process is repeated until no more features can be removed. After this step, there are 55, 54, and 54 features left for time intervals of 1, 2, and 3 years, respectively.

3.3.3.3 Redundancy Analysis

After removing highly correlated features, *redun* function from the *rms* R package is used to find redundant features. The *redun* function determines how well each feature can be predicted using the remaining features and removes the most predictable feature in an

Figure 3.5. The feature selection process used in this research.



iterative process. The process stops either when any of the remaining features cannot be predicted with an adjusted R^2 greater than a threshold or when removing any feature would cause a previously dropped feature to no longer be predicted at the threshold (the default threshold of 0.9 is used). After this step, 54, 53, and 52 features are left for time intervals of 1, 2, and 3 years, respectively.

To ensure the three time intervals have the same features, the features are combined by taking the set union of 54, 53, and 52 features for time intervals 1, 2, and 3, respectively, yielding 55 features. These 55 features have 33 features from the original Bao et al., (2019) 63 features and 22 from the newly extracted 110 features. The 55 features and descriptive statistics are shown in Tables 3.12, 3.13, and 3.14.

Table 3.12 shows feature set 3 for DP and RP dimensions. These are selected 55 features arranged in two columns, column three has features extracted by Bao et al., (2019) (in normal text), and column four has the additional features extracted in this study (in italics).

Table 3.13 shows feature set 3 for DA, RA, and CN dimensions. These are selected 55 features arranged in two columns, column three has features extracted by Bao et al., (2019) (in normal text), and column four has the additional features extracted in this study (in italics).

These 55 features are used for years 1, 2, and 3 to investigate whether a new contributor will become an LTC and compare the results with those of the original study.

3.4 Prediction Models

In the previous sections, three datasets were extracted. One dataset has 31 features, another has 173 features, and the final dataset has 55 features. The 31, 173, and 55 feature datasets are PastHistory_31, Onboarding_173, and Onboarding_55, respectively. The 63 features from Bao et al., (2019) are referred to as Baseline.

These datasets are used to train ML models using standard ML model building methodology shown in Fig. 3.6 to predict whether a new contributor will continue to be an LTC to

Table 3.12: Feature Set 3: Selected 55 features for DP and RP dimensions.

Dim	Feature	Baseline features	This research features
	user_age	✓	✓
	user_own_repos	✓	✓
	user_own_repo_ratio		✓
	user_watch_repos	✓	✓
	user_contribute_repos	✓	
	user_history_commits	✓	
	user_watch_repos_lang		✓
	user_watch_repos_ratio		✓
	user_contribute_repos_lang		✓
DP	*user_contribute_repos_ratio*		✓
	user_history_pull_requests	✓	✓
	user_history_pull_requests_lang		✓
	user_history_pull_requests_ratio		✓
	user_history_issues	✓	✓
	user_history_issues_lang		✓
	user_history_issues_ratio		✓
	user_history_followers	✓	✓
	user_history_following		✓
	language	✓	
	repo_age		✓
	before_repo_commits	✓	✓
	one_month_before_repo_commits		✓
	before_repo_commit_comments	✓	✓
	one_month_before_repo_commit_comments		✓
	before_repo_contributors	✓	
	before_repo_contributor_max	✓	
	before_repo_contributor_min	✓	✓
	before_repo_contributor_mean	✓	✓
	before_repo_contributor_median	✓	✓
	before_repo_issues	✓	
	before_repo_issue_comments	✓	
	before_repo_issue_events	✓	
	before_repo_issue_events_closed	✓	
RP	*one_month_before_repo_issues*		✓
	before_repo_issue_events_assigned	✓	✓
	one_month_before_repo_issue_events_assigned		✓
	before_repo_pull_requests	✓	✓
	before_repo_pull_request_comments	✓	✓
	before_repo_pull_request_history	✓	
	before_repo_pull_request_history_merged	✓	✓
	before_repo_pull_request_history_closed	✓	
	one_month_before_repo_pull_request_history_merged		✓
	before_repo_watchers	✓	✓
	one_month_before_repo_watchers		✓
	one_month_after_repo_watchers		✓

Table 3.13: Feature Set 3: Selected 55 features for DA, RA, and CN dimensions.

Dim	Feature	Baseline features	This research features
	month_user_commits	✓	✓
	month_user_commit_comments	✓	✓
	two_month_user_commit_comments		✓
	month_user_issues	✓	
	month_user_issue_comments	✓	
	month_user_issue_events	✓	
	month_user_issue_events_closed	✓	
DA	*month_user_issues_fixed*		✓
	month_user_issue_events_assigned	✓	✓
	month_user_pull_requests	✓	✓
	month_user_pull_request_comments	✓	✓
	month_user_pull_request_history	✓	
	month_user_pull_request_history_closed	✓	
	month_user_pull_request_history_merged	✓	✓
	three_month_user_pull_request_history_merged		✓
	three_month_user_pull_request_history_closed		✓
	month_repo_commits	✓	
	month_repo_commit_comments	✓	✓
	month_repo_contributors	✓	
	month_repo_contributor_max	✓	✓
	month_repo_contributor_min	✓	✓
	month_repo_contributor_mean	✓	✓
	month_repo_contributor_median	✓	✓
	month_repo_issues	✓	
RA	month_repo_issue_comments	✓	
	month_repo_issue_events	✓	
	month_repo_issue_events_closed	✓	
	month_repo_issue_events_assigned	✓	
	month_repo_pull_requests	✓	
	month_repo_pull_request_comments	✓	
	month_repo_pull_request_history	✓	
	month_repo_pull_request_history_merged	✓	✓
	month_repo_pull_request_history_closed	✓	
	degree_centrality	✓	✓
	closeness_centrality	✓	
CN	betweenness_centrality	✓	✓
	eigenvector_centrality	✓	✓
	clustering_coefficient	✓	✓

Table 3.14: Feature Set 3: Descriptive statistics of the selected 55 features.

Dim	Feature	Quant_5	Mean	Median	Quant_95	Histogram
DP	user_age	4.14	6.63	6.97	7.76	
	user_own_repos	0.00	2.51	2.64	4.58	
	user_own_repo_ratio	0.00	0.26	0.22	0.69	
	user_watch_repos	0.00	2.56	2.56	5.69	
	user_watch_repos_lang	0.00	1.54	1.10	4.51	
	user_watch_repos_ratio	0.00	0.23	0.15	0.69	
	user_contribute_repos_lang	0.00	1.55	1.39	4.01	
	user_contribute_repos_ratio	0.00	0.28	0.24	0.69	
	user_history_pull_requests	0.00	2.09	1.95	5.05	
	user_history_pull_requests_lang	0.00	1.22	0.69	4.08	
	user_history_pull_requests_ratio	0.00	0.26	0.15	0.69	
	user_history_issues	0.00	2.08	1.95	5.12	
	user_history_issues_lang	0.00	1.24	0.69	4.16	
	user_history_issues_ratio	0.00	0.26	0.13	0.69	
	user_history_followers	0.00	2.52	2.48	5.49	
	user_history_following	0.00	1.82	1.79	4.58	
RP	*repo_age*	3.91	6.48	6.84	7.71	
	before_repo_commits	3.91	7.55	7.81	10.27	
	one_month_before_repo_commits	1.39	4.78	5.03	7.20	
	before_repo_commit_comments	0.00	3.89	4.08	7.17	
	one_month_before_repo_commit_comments	0.00	1.50	1.10	4.26	
	before_repo_contributor_min	0.69	0.72	0.69	0.69	
	before_repo_contributor_mean	1.53	3.03	3.02	4.59	
	before_repo_contributor_median	0.69	1.15	1.10	2.01	
	one_month_before_repo_issues	0.00	3.82	4.08	5.80	
	before_repo_issue_events_assigned	0.00	3.49	3.74	7.15	
	one_month_before_repo_issue_events_assigned	0.00	1.43	1.10	4.49	
	before_repo_pull_requests	2.08	6.32	6.48	9.69	
	before_repo_pull_request_comments	0.00	5.27	5.62	9.57	
	before_repo_pull_request_history_merged	0.00	4.59	4.96	8.25	
	one_month_before_repo_pull_request_history_merged	0.00	1.74	1.39	4.75	
	before_repo_watchers	4.49	8.03	8.41	10.11	
	one_month_before_repo_watchers	0.00	4.96	5.25	7.24	
	one_month_after_repo_watchers	1.10	5.07	5.30	7.20	
DA	month_user_commits	0.69	1.15	0.69	2.64	
	month_user_commit_comments	0.00	0.06	0.00	0.00	
	two_month_user_commit_comments	0.00	0.06	0.00	0.00	
	month_user_issues_fixed	0.00	0.15	0.00	1.10	
	month_user_issue_events_assigned	0.00	0.03	0.00	0.00	
	month_user_pull_requests	0.00	0.09	0.00	0.69	
	month_user_pull_request_comments	0.00	0.05	0.00	0.00	
	month_user_pull_request_history_merged	0.00	0.04	0.00	0.00	
	three_month_user_pull_request_history_merged	0.00	0.09	0.00	0.69	
	three_month_user_pull_request_history_closed	0.00	0.19	0.00	1.10	
RA	month_repo_commit_comments	0.00	1.58	1.39	4.28	
	month_repo_contributor_max	1.39	3.95	4.11	5.95	
	month_repo_contributor_min	0.69	0.73	0.69	0.69	
	month_repo_contributor_mean	0.98	2.16	2.12	3.44	
	month_repo_contributor_median	0.69	1.23	1.10	2.40	
	month_repo_pull_request_history_merged	0.00	1.84	1.61	4.87	
CN	degree_centrality	0.00	0.02	0.00	0.10	
	betweenness_centrality	0.00	0.01	0.00	0.05	
	eigenvector_centrality	0.00	0.02	0.00	0.16	
	clustering_coefficient	0.00	0.04	0.00	0.34	

the project. In this study, NB, SVM, kNN, LR, DT, and RF models are used, as these are used extensively in software engineering research (Tantithamthavorn et al., 2018).

The above data are organized as input pairs $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathscr{R}^d$ is an input instance and y its class label. The entire data are denoted as

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \subseteq \mathscr{R}^d \times \mathscr{I} \tag{3.1}$$

where $\mathscr{R}^d$: d-dimensional feature space; $\mathbf{x}_i$: input instance vector of the i^{th} observation; y_i: class label of the i^{th} observation; and $\mathscr{I} = \{0, 1\}$: labels are in binary space in this research. The goal of the prediction model is to learn a function $h : \mathbb{R}^d \rightarrow \mathscr{I}$, such that $h(\mathbf{x}_i) \approx y_i$ for all $(\mathbf{x}_i, y_i) \in D$. The prediction models used in this study are now briefly described.

3.4.1 naive Bayes (NB)

NB (John and Langley, 1995) is a generative model for classification based on Bayes' theorem that attempts to find class conditional probabilities. Bayes' theorem is used to compute the posterior probability given the prior probability and is stated as

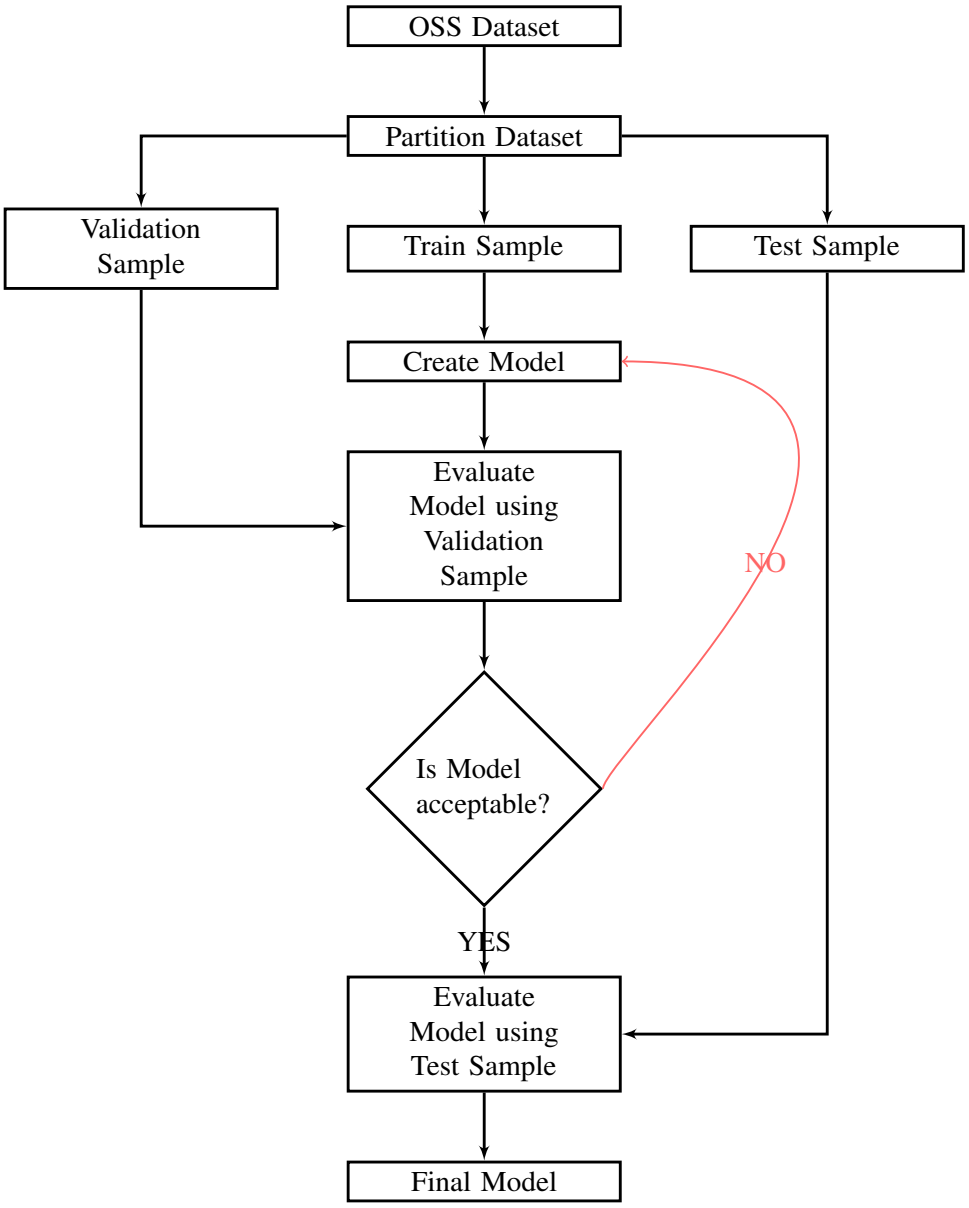
$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y) \times P(y)}{P(\mathbf{x})}. \tag{3.2}$$

NB makes a naive assumption that features are independent, which often is an overly strong assumption for real-world problems. An input $\mathbf{x}$ is classified by computing the posterior probabilities for the two-class labels $y = \{0, 1\}$, and class label 0 is assigned if $P(0|\mathbf{x}) > P(1|\mathbf{x})$; otherwise, class label 1 is assigned to the data instance.

3.4.2 Support Vector Machines (SVM)

SVM (Cortes and Vapnik, 1995) is a classification model based on the concept of finding a separating decision boundary between the two classes of data. It tries to maximize the

Figure 3.6. ML model building methodology.



margin between the hyperplane and the nearest data points. The margin between the two classes of data with labels 1 and -1 are given by the two equations,

$$\mathbf{w}^T\mathbf{x} - b = 1 \tag{3.3}$$

$$\mathbf{w}^T\mathbf{x} - b = -1 \tag{3.4}$$

The maximum margin hyperplane would lie midway between these two parallel hyperplanes and is given by the equation,

$$\mathbf{w}^T\mathbf{x} - b = 0 \tag{3.5}$$

In case of a hard-margin problem, the objective is to find a maximum margin hyperplane to classify all points in the training data correctly; for the i-th data point,

$$\mathbf{w}^T\mathbf{x_i} - b \geq 1, \qquad \text{if } y_i = 1 \tag{3.6}$$

$$\mathbf{w}^T\mathbf{x_i} - b \leq -1, \qquad \text{if } y_i = -1 \tag{3.7}$$

Combining these two gives us,

$$y_i(\mathbf{w}^T\mathbf{x_i} - b) \geq 1, \qquad \text{for all } i \tag{3.8}$$

So the problem of finding the maximum margin hyperplane reduces to a simple optimization problem.

A soft margin SVM may be used in linearly non-separable problems, allowing a few

data points to be misclassified. Although this would reduce the training set accuracy, it improves the generalizability of the classifier.

The kernel trick for SVMs makes it applicable to more complex data that are not linearly separable by a hyperplane in the d-dimensional input space. The kernel trick is done by applying a kernel function to the data points to map them into a higher dimensional vector space where they become linearly separable with a hyperplane, and the problem reduces to that of linear classification. Some widely used kernel functions include polynomial, Gaussian and radial basis function(RBF) kernels.

- Polynomial kernel:

$$K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}\mathbf{x_j} + 1)^d \tag{3.9}$$

- Gaussian RBF kernel:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma||\mathbf{x_i} - \mathbf{x_j}||^2) \quad \textit{where} \quad \gamma = 1/2\sigma^2 \tag{3.10}$$

- Laplace RBF kernel:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{||\mathbf{x_i} - \mathbf{x_j}||}{\sigma}) \tag{3.11}$$

Say $\phi(\mathbf{x_i})$ is the transformed form of $\mathbf{x_i}$ in the higher dimensional feature space, the kernel function is given by,

$$K(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i}).\phi(\mathbf{x_j}) \tag{3.12}$$

However, to get accurate results, the number of training samples should be large enough to fit the classifier in the higher dimensional feature space properly.

3.4.3 k-Nearest Neighbor (kNN)

kNN (Aha, Kibler, and Albert, 1991) is a nonparametric instance model where input $\mathbf{x}$ is assigned the most common label among its k most similar training inputs. Let $S_{\mathbf{x}}$ be the set of k-nearest neighbors of $\mathbf{x}$, where $S_{\mathbf{x}}$ is defined as $S_{\mathbf{x}} \subseteq D$ s.t. $|S_{\mathbf{x}}| = k$ and $\forall(\mathbf{x}_i, y_i) \in D \backslash S_{\mathbf{x}}$,

$$\text{dist}(\mathbf{x}, \mathbf{x}_i) \geq \max_{(\mathbf{x}_j, y_j) \in S_{\mathbf{x}}} \text{dist}(\mathbf{x}, \mathbf{x}_j) \tag{3.13}$$

(i.e., every point in D but not in $S_{\mathbf{x}}$ is at least as far away from $\mathbf{x}$ as the farthest point in $S_{\mathbf{x}}$). Classifier function $h(\cdot)$ is defined to return the most common label in $S_{\mathbf{x}}$:

$$h(\mathbf{x}) = \text{mode}(\{y_l : (\mathbf{x}_l, y_l) \in S_{\mathbf{x}}\}) \tag{3.14}$$

where mode$(\cdot)$ selects the label of the highest occurrence.

3.4.4 Logistic Regression (LR)

LR (Le Cessie and Van Houwelingen, 1992) is a discriminative model of $P(y|\mathbf{x})$, which is assumed to take the exact form

$$P(y|\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^{\mathbf{T}}\mathbf{x} + b)}} \tag{3.15}$$

$\mathbf{w}$ and b are directly estimated via the maximum likelihood estimate or maximum a posteriori.

3.4.5 Decision Tree (DT)

DT (Landwehr, Hall, and Frank, 2005) is a hierarchical, efficient, nonparametric method for classification. A DT contains an inner decision node and leaf outcome (label) nodes. Each inner node has a decision function $f(\mathbf{x})$ with a branch for each outcome. The DT is constructed with the entire training set starting at the root node and is split repeatedly until

one of two conditions is met. The first condition is that all data points in a subset have the same label. If this occurs, splitting stops, and a leaf node is associated with the label y. The other condition is that there are no more features that could be used to split the subset. In this case, a leaf is created and labeled with the most common y. At each node, for all features, all possible splits are tested to identify the split that minimizes the impurity, and splits are obtained as

$$\begin{bmatrix} D^L = \{(\mathbf{x}, y) \in D : x_f \leq t\} \\ D^R = \{(\mathbf{x}, y) \in D : x_f > t\} \end{bmatrix} \tag{3.16}$$

where $f \leftarrow feature$, $x_f \leftarrow feature\ value$, and $t \leftarrow threshold$.

Classification of an input $\mathbf{x}$ starts at the root node. The function at the node is applied, a branch is selected based on the outcome, and this process is repeated until a leaf is reached. The label of the leaf is the class assigned to $\mathbf{x}$.

3.4.6 Random Forest (RF)

RF (Breiman, 2001) is an ensemble of DTs with slightly modified splitting criteria. The model is built as follows. m datasets $D_1, \ldots, D_m$ are sampled from D with replacement. For each D_j, a full DT $t_j()$ (max-depth=∞) is trained with one small modification: before each split $k \leq d$, features are randomly subsampled (without replacement), and only these features are considered for the split. The final classifier is

$$t(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^{m} t_j(\mathbf{x}) \tag{3.17}$$

Table 3.15 summarizes the prediction models and the parameter settings used in this study.

Table 3.15: Prediction models with different parameters.

Model	Parameter	Parameter Description	Parameter Value Used
NB	Distribution Type	TRUE means kernel density estimation is used, and FALSE means normal density estimation is used.	FALSE
SVM	# Sigma	The width of the Gaussian kernels.	0.9
	# Cost	A penalty factor to be applied to the number of errors.	0.25
kNN	# Clusters	The number of nonoverlapping clusters to produce.	17
LR	# M	The maximum iteration count (defaults to -1).	-1
DT	# Iterations	The number of iterations.	21
RF	# Trees	The number of classification trees.	100

3.5 Evaluation Metrics

To evaluate the performance of the models in this research, nine performance measures are commonly used to evaluate models in software engineering (Tantithamthavorn et al., 2018, Zhou and Mockus, 2015, and Fan et al., 2018). One measure, AUC, is a threshold-independent measure and the rest eight are threshold-dependent traditional measures. Threshold-dependent performance measures are calculated by transforming the prediction probabilities into binary classification (LTC or non-LTC) using a default threshold value of 0.5; i.e., if a contributor record has a predicted probability greater than 0.5, they are considered an LTC; otherwise, they are considered a non-LTC. As summarized in Tantithamthavorn et al., (2018), threshold-dependent performance measures have issues because of their dependence on an arbitrarily selected threshold value (Lessmann et al., 2008) and sensitivity to imbalanced datasets (Davis and Goadrich, 2006). This research uses the same threshold (0.5) for these measures to compare the proposed model with the baseline to eliminate bias. Each performance measure is described and defined below.

For each record in the dataset, there are four possible outcomes: a contributor is classified as an LTC when they truly are an LTC (true positive, TP); a contributor is classified as an LTC when they have abandoned the project and become a non-LTC (false positive, FP); a

contributor is classified as a non-LTC when they truly are an LTC (false negative, FN); a contributor is classified as a non-LTC when they have abandoned the project and become a non-LTC (true negative, TN). Based on these four possibilities, the precision, recall, and F1-score are computed to compare the performance of the proposed approaches with that of the baseline.

3.5.1 LTC Precision

LTC Precision is defined as the fraction of contributors correctly classified as LTCs among those who are classified as LTCs, i.e.:

$$P(LTC) = \frac{TP}{TP+FP} \tag{3.18}$$

3.5.2 LTC Recall

LTC Recall is defined as the fraction of LTC contributors who are correctly classified, i.e.:

$$R(LTC) = \frac{TP}{TP+FN} \tag{3.19}$$

3.5.3 non-LTC Precision

non-LTC Precision is defined as the fraction of contributors correctly classified as non-LTC among those who are classified as non-LTC, i.e.:

$$P(non-LTC) = \frac{TN}{TN+FN} \tag{3.20}$$

3.5.4 non-LTC Recall

non-LTC Recall is defined as the fraction of non-LTC contributors who are correctly classified, i.e.:

$$R(non-LTC) = \frac{TN}{FP+TN} \tag{3.21}$$

3.5.5 F1-score

The F1-score combines both precision and recall.

For LTC contributors, the F1-score is the harmonic mean of P(LTC) and R(LTC), i.e.:

$$F1(LTC) = \frac{2 \times P(LTC) \times R(LTC)}{P(LTC)+R(LTC)} \tag{3.22}$$

For non-LTC contributors, the F1-score is the harmonic mean of P(non-LTC) and R(non-LTC), i.e.:

$$F1(non-LTC) = \frac{2 \times P(non-LTC) \times R(non-LTC)}{P(non-LTC)+R(non-LTC)} \tag{3.23}$$

3.5.6 Accuracy

Accuracy is the fraction of correctly classified contributors, i.e.:

$$Accuracy = \frac{TP+TN}{TP+FN+FP+TN} \tag{3.24}$$

3.5.7 Matthews Correlation Coefficient (MCC)

MCC (Matthews, 1975) is computed using the following equation:

$$MCC = \frac{(TP \times TN)-(FP \times FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{3.25}$$

The above equation yields a real value between -1 (perfect disagreement between predictions and actual values) and 1 (perfect agreement between predictions and actual values). A value of 0 indicates random prediction.

3.5.8 Area Under the Curve (AUC)

AUC is computed by plotting the FP rate on the x-axis and the TP rate on the y-axis, where

$$TPR = \frac{TP}{TP + FN} \tag{3.26}$$

$$FPR = \frac{FP}{FP + TN} \tag{3.27}$$

AUC values lie between 0 and 1, with larger values indicating better performance.

3.6 Model Validation Method

In this study, the K-fold cross-validation is used to compare the results of this study with the baseline.

3.6.1 K-Fold Cross-Validation (KFCV)

In K- Fold Cross Validation (KFCV), the dataset is partitioned randomly into K folds. Each fold is approximately equal in size and maintains the same proportions of LTC and non-LTC as in the original data. Of these K folds, (K - 1) folds are used to train the model, while the remaining fold is used to evaluate the performance, as shown in Fig. 3.7 and Code Snippet 3.8. The value of K is set to 10 (K = 10) (i.e., 10-fold cross-validation).

Figure 3.7. Partitioning the dataset into k-folds, where k = 10. Blue partition represents validation sample. Yellow partition represents training sample.

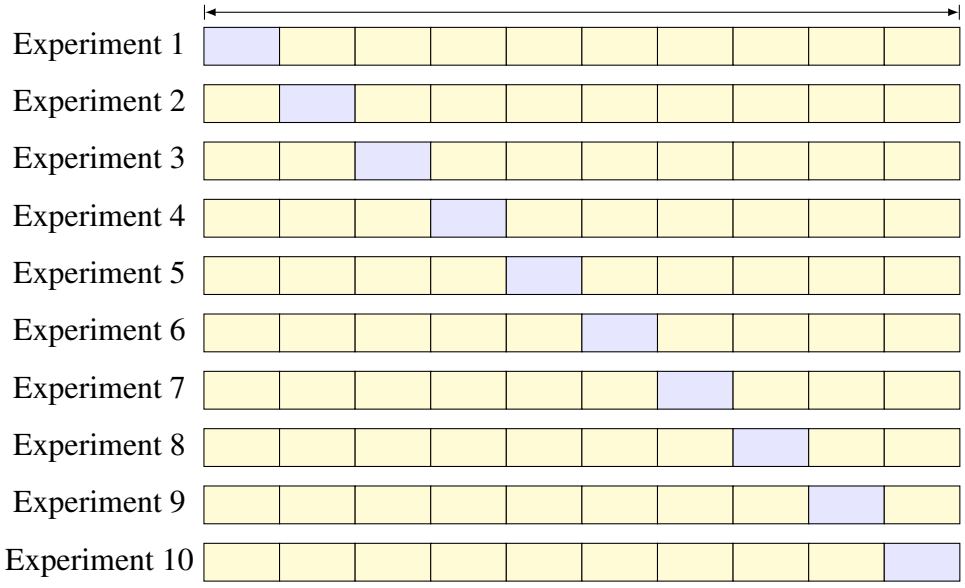


Figure 3.8. Code to partitioning the dataset into 10-folds and train the RF model.

```
predict_LTC_cross_validation <- function(records,
selected_features=NULL, cls="RF", approach=1,
threshold_adaptive=FALSE, imbalance_method=NULL,
outfile="test_time.csv", ...){
    varNames <- names(records)
    varNames <- varNames[!varNames %in% c("is_ltc")]
    varNames1 <- paste(varNames, collapse = "+")
    rf.form <- as.formula(paste("is_ltc", varNames1,
    sep = " ~ "))
    records$is_ltc <- as.factor(records$is_ltc)
    folds <- createFolds(records$is_ltc, k=10, l
    ist=TRUE, returnTrain=TRUE)
    results_frame <- data.frame(
    stringsAsFactors = FALSE)
    for (k in names(folds)) {
     print(paste0("running fold ", k))
     inTrain <- folds[[k]]
     training <- selected_records[inTrain,]
     testing <- selected_records[-inTrain,]
     metrics <- predict_one(rf.form, training, testing,
     cls, approach=approach,
     threshold_adaptive=threshold_adaptive,
     imbalance_method=imbalance_method, ...)
     results_frame <- rbind(results_frame, metrics)
    }
    colnames(results_frame) <- c("accuracy",
    "precision0", "recall0", "f1_score0", "precision1",
    "recall1", "f1_score1", "MCC", "AUC")
    return(results_frame)
}
```

Chapter 4: Experiments and Results

In this chapter, the results of experiments to address the research objectives from Chapter 1 are shared. Results are presented in two sections, one for each research objective. Each research objective section briefly describes the dataset and evaluation methodology used to answer the research questions. Each research question describes the motivation, approach, and results to answer the research question.

4.1 Experimental Results - Research Objective 1

In this section, 31 features extracted from GitHub API and GHTorrent using BigQuery are used. The two tests from Bao et al., (2019) using the same software are applied to compare the proposed approaches. The first test, the Wilcoxon signed-rank test (Wilcoxon, 1946) with Bonferroni correction (Abdi et al., 2007), is used to determine whether the improvement over the baseline is statistically significant. The second test, Cliff's delta[1] (Cliff, 2014), is a non-parametric effect size measure used to quantify the difference between the results of various approaches.

4.1.1 RQ 1.1: Can past history effectively determine an LTC?

Motivation: Capturing, engaging and retaining new contributors is critical for OSS project survival. Effectively determining an LTC as soon as a new contributor joins a repository helps repository owners to utilize limited resources and create an environment that enables LTC conversion. The 31 features are used with five prediction models to investigate the feasibility of building accurate prediction models.

[1]Cliff defines delta values less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474 and above 0.474 as negligible, small, medium, and large effect sizes, respectively.

Table 4.1: Evaluation metrics for the proposed past history approach for all models.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	PastHistory_31-NB	0.685	0.100	0.877	0.934	0.879	**0.996**	0.056	0.521	0.030
	PastHistory_31-SVM	0.677	0.181	0.880	0.935	0.884	0.994	0.129	0.615	0.072
	PastHistory_31-kNN	0.713	0.213	0.880	0.935	0.887	0.989	0.182	0.583	0.108
	PastHistory_31-LR	0.703	0.193	0.880	0.935	0.885	0.991	0.153	0.587	0.088
	PastHistory_31-DT	0.719	0.272	0.885	0.938	0.892	0.989	0.242	0.647	0.149
	PastHistory_31-RF	**0.792**	**0.309**	**0.888**	**0.940**	**0.894**	0.990	**0.274**	**0.701**	**0.171**
2	PastHistory_31-NB	0.751	0.232	0.913	0.954	**0.960**	0.948	**0.275**	0.256	**0.304**
	PastHistory_31-SVM	0.728	0.193	0.947	0.973	0.950	**0.997**	0.130	0.582	0.073
	PastHistory_31-kNN	0.778	0.175	0.947	0.972	0.949	**0.997**	0.119	0.535	0.067
	PastHistory_31-LR	0.781	0.186	0.947	0.972	0.950	0.996	0.133	0.531	0.076
	PastHistory_31-DT	0.793	0.249	0.948	0.973	0.952	0.995	0.204	0.570	0.125
	PastHistory_31-RF	**0.869**	**0.271**	**0.949**	**0.974**	0.952	0.996	0.210	**0.655**	0.125
3	PastHistory_31-NB	0.812	0.210	0.934	0.966	**0.986**	0.946	**0.210**	0.146	**0.395**
	PastHistory_31-SVM	0.737	0.150	0.978	**0.989**	0.979	**0.999**	0.086	0.517	0.047
	PastHistory_31-kNN	0.811	0.158	0.978	**0.989**	0.979	**0.999**	0.082	0.597	0.044
	PastHistory_31-LR	0.851	0.191	0.978	**0.989**	0.980	**0.999**	0.131	0.524	0.075
	PastHistory_31-DT	0.845	0.177	0.978	**0.989**	0.980	0.998	0.120	0.497	0.069
	PastHistory_31-RF	**0.913**	**0.226**	**0.979**	**0.989**	0.980	**0.999**	0.140	**0.695**	0.079

Approach: NB, SVM, kNN, LR, DT, and RF prediction models using 31 features are built. To measure the effectiveness of the proposed approach, the average AUC, MCC, accuracy, precision, recall, and F1-scores for LTC and non-LTC contributors are calculated using 10-fold cross-validation for the three time intervals.

Results: Table 4.1 presents the average AUC, MCC, Accuracy, F1-scores, Precision, and Recall for LTC and non-LTC contributors for the proposed PastHistory_31-NB, PastHistory_31-SVM, PastHistory_31-kNN, PastHistory_31-LR, PastHistory_31-DT, and PastHistory_31-RF models using 10-fold cross-validation with 1, 2, and 3 year time intervals. The best results are in bold. Overall, PastHistory_31-RF model achieves better results than PastHistory_31-SVM, PastHistory_31-kNN, PastHistory_31-LR, and PastHistory_31-DT models across all metrics for all three time intervals. The PastHistory_31-RF model achieves better results than PastHistory_31-NB across all metrics except for precision for non-LTC, F1-scores for LTC, and recall for LTC in time intervals 2 and 3.

Table 4.2 presents the adjusted p-values and Cliff's delta values comparing the AUC, MCC, accuracy F1(non-LTC), and F1(LTC) of PastHistory_31-RF model with PastHistory_31-NB, PastHistory_31-SVM, PastHistory_31-kNN, PastHistory_31-LR, and PastHistory_31-DT using 10-fold cross-validation with 1, 2, and 3 year time intervals. PastHistory_31-RF

Table 4.2: Cliff's delta and p-values comparing past history models with RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	F1(LTC)
1	PastHistory_31-NB	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	PastHistory_31-SVM	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	PastHistory_31-kNN	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	PastHistory_31-LR	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	PastHistory_31-DT	1 (Large) ***	0.84 (Large) **	0.86 (Large) **	0.86 (Large) **	0.76 (Large) *
2	PastHistory_31-NB	1 (Large) ***	0.78 (Large) *	1 (Large) ***	1 (Large) ***	-0.96 (Large) ***
	PastHistory_31-SVM	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	PastHistory_31-kNN	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	PastHistory_31-LR	1 (Large) ***	1 (Large) ***	0.98 (Large) ***	0.96 (Large) ***	1 (Large) ***
	PastHistory_31-DT	1 (Large) ***	0.4 (Medium) ^{i}	0.61 (Large) *	0.6 (Large) *	0.13 (Negligible) ^{i}
3	PastHistory_31-NB	1 (Large) ****	0.28 (Small) ^{i}	1 (Large) ****	1 (Large) ****	-0.82 (Large) ***
	PastHistory_31-SVM	1 (Large) ****	1 (Large) ***	0.73 (Large) **	0.71 (Large) **	0.79 (Large) ***
	PastHistory_31-kNN	1 (Large) ****	0.76 (Large) ***	0.73 (Large) **	0.71 (Large) **	0.79 (Large) ***
	PastHistory_31-LR	1 (Large) ****	0.36 (Medium) ^{i}	0.67 (Large) *	0.68 (Large) *	0.12 (Negligible) ^{i}
	PastHistory_31-DT	1 (Large) ****	0.5 (Large) ^{i}	0.78 (Large) ***	0.81 (Large) ***	0.26 (Small) ^{i}

$^{****}p < 0.001$, $^{***}p < 0.005$, $^{**}p < 0.01$, $^{*}p < 0.05$, $^{i}p > 0.05$ (insignificant)

AUC is statistically significant at the 99.9% confidence level, with a large effect size over all other models in all three time intervals. PastHistory_31-RF accuracy and F1(non-LTC) are statistically significant at the 95% confidence level or higher, with a large effect size over all other models in all three time intervals. PastHistory_31-RF MCC is not statistically significant across all models in all time intervals. PastHistory_31-RF F1(LTC) is not statistically significant over the PastHistory_31-DT model in time intervals 2 and 3. PastHistory_31-RF F1(LTC) is not statistically significant over the PastHistory_31-LR model in time interval 3. PastHistory_31-RF F1(LTC) is statistically significant at the 99.5% confidence level, with a large effect size over all other models in all time intervals.

These results indicate that NB, SVM, kNN, LR, DT, and RF prediction models built using the proposed 31 features are beneficial for detecting LTCs under both threshold-dependent and threshold-independent metrics. In addition, it is found that the RF prediction model outperforms all other models on the AUC metric with a statistical significance of 99.5% confidence level, with a large effect size.

4.1.2 RQ 1.2: How effective is the past history approach compared to the state-of-the-art approach?

The state-of-the-art approach was published by Bao et al., (2019), where they predict LTCs by developing ML models using 63 features. The features are extracted from GHTorrent using BigQuery and are related to contributors and repositories, organized along five dimensions: DP, RP, DA, RA, and CN. Refer to Table 2.8 for a summary of each feature. The features capture data until the first month after a contributor joins the repository. The original dataset consists of 917 repositories with 63 features. Duplicate repositories are removed, and a dataset with 888 repositories with 63 features is used.

Motivation: This research question investigates how well the proposed 31 features perform compared to the 63 features (Bao et al., 2019) across multiple metrics. There are two differences between the proposed feature set compared to Bao et al., (2019) feature set: (1) less than 50% of features (31 vs. 63) are used, and (2) the proposed features have no wait time of one month after the contributor joins. Effectively, this research question is seeking to answer whether the prediction model built based on the proposed 31 features is better than the state-of-the-art model built with 63 features.

Approach: The RF model was the best in the baseline (Bao et al., 2019). It was also found in RQ 1.1 that the RF in the proposed approach performs well. To evaluate the performance, RF models with 31 features (PastHistory_31-RF) and the baseline 63 features (Baseline-RF) for time intervals T = 1, 2, and 3 are built and are compared with respect to the average AUC, MCC, accuracy, precision, recall, and F1-scores for LTC and non-LTC using 10-fold cross-validation.

Results: Table 4.3 presents the average AUC, MCC, Accuracy, F1-scores, Precision, and Recall for LTC and non-LTC contributors for the proposed PastHistory_31-RF and Baseline-RF models using 10-fold cross-validation with 1, 2, and 3 year time intervals. The best results are in bold.

Table 4.3: Evaluation metrics for the proposed past history RF model and Bao et al., (2019) RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	Baseline-RF	**0.813**	**0.319**	**0.889**	0.940	**0.896**	0.987	**0.297**	0.679	**0.190**
	PastHistory_31-RF	0.792	0.309	0.888	0.940	0.894	**0.990**	0.274	**0.701**	0.171
2	Baseline-RF	**0.872**	0.250	**0.949**	**0.974**	0.951	**0.997**	0.184	0.649	0.108
	PastHistory_31-RF	0.869	**0.271**	**0.949**	**0.974**	**0.952**	0.996	**0.210**	**0.655**	**0.125**
3	Baseline-RF	0.908	0.144	0.978	**0.989**	0.979	**0.999**	0.075	0.546	0.041
	PastHistory_31-RF	**0.913**	**0.226**	**0.979**	**0.989**	**0.980**	**0.999**	**0.140**	**0.695**	**0.079**

Table 4.4: Percentage gain in evaluation metrics for the proposed past history RF model and Bao et al., (2019) RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	PastHistory_31-RF	-2.58%	-3.13%	-0.11%	0.0%	-0.22%	0.30%	-7.74%	3.24%	-10.00%
2	PastHistory_31-RF	-0.34%	8.40%	0.0%	0.0%	0.11%	-0.10%	14.13%	0.92%	15.74%
3	PastHistory_31-RF	0.55%	56.94%	0.10%	0.00%	0.10%	0.00%	86.67%	27.29%	92.68%

Table 4.4 presents the percentage gain in AUC, MCC, Accuracy, F1-scores, Precision, and Recall for LTC and non-LTC contributors for the PastHistory_31-RF model compared to Baseline-RF using 10-fold cross-validation with 1, 2, and 3 year time intervals. % gain is computed using (((PastHistory_31-RF - Baseline-RF)/Baseline-RF) * 100).

Table 4.5 presents the adjusted p-values and Cliff's delta values comparing the AUC, MCC, accuracy F1(non-LTC), and F1(LTC) of PastHistory_31-RF model with Baseline-RF using 10-fold cross-validation with 1, 2, and 3 year time intervals.

For a time interval of 1 year, Table 4.3 shows that Bao et al., (2019) RF model performs better than the proposed model. The Baseline-RF model achieves better results than PastHistory_31-RF across all metrics except for three metrics. For F1-scores for non-LTC, both methods produce the same result. For recall for non-LTC and precision for LTC

Table 4.5: Cliff's delta and p-values comparing the proposed past history and Bao et al., (2019) RF models.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	F1(LTC)
1	PastHistory_31-RF	-1.00 (Large) ****	-0.40 (Medium) ^{i}	-0.12 (Negligible) ^{i}	-0.04 (Negligible) ^{i}	-0.58 (Large) *
2	PastHistory_31-RF	-0.32 (Small) ^{i}	0.40 (Medium) ^{i}	0.23 (Small) ^{i}	0.22 (Small) ^{i}	0.63 (Large) *
3	PastHistory_31-RF	0.3 (Small) ^{i}	0.76 (Large) **	0.72 (Large) **	0.7 (Large) **	0.77 (Large) ***

$^{****}p < 0.001$, $^{***}p < 0.005$, $^{**}p < 0.01$, $^{*}p < 0.05$, $^{i}p > 0.05$ (insignificant)

PastHistory_31-RF method produces a better result. Table 4.5 shows that Baseline-RF AUC and F1(LTC) are statistically significant at the 99% confidence level or higher, with a large effect size over the PastHistory_31-RF model. The other results are not statistically significant over the PastHistory_31-RF model.

For a time interval of 2 years, the proposed model performs better than Bao et al., (2019) RF model. Table 4.3 shows that the PastHistory_31-RF model achieves better or the same results than Baseline-RF across all metrics except for two metrics. For AUC and recall for non-LTC Baseline-RF method produces a better result. Table 4.5 shows that only PastHistory_31-RF F1(LTC) is statistically significant at the 99.5% confidence level, with a large effect size over the Baseline-RF model. The other results are not statistically significant over the Baseline-RF model.

For a time interval of 3 years, the proposed model performs better than Bao et al., (2019) RF model. Table 4.3 shows that the proposed models precision, recall, F1-score, MCC, and AUC are 0.695, 0.079, 0.140, 0.226, and 0.913, respectively. These values are 27.29%, 92.68%, 86.67%, 56.94%, and 0.55%, respectively better than the best baseline model as shown in Table 4.4. Table 4.5 shows that PastHistory_31-RF MCC, accuracy, F1(non-LTC), and F1(LTC) is statistically significant at the 99% confidence level or higher, with a large effect size over the Baseline-RF model. PastHistory_31-RF AUC is better but not statistically significant over the Baseline-RF model.

Overall, these results indicate that the proposed model performs better than or equal to the Bao et al., (2019) model when going from time intervals 1 to 3 years, indicating that the features used in this approach effectively predict LTCs. The proposed approach uses less than 50% features (31 vs. 63) and has no wait time of one month after the contributor joins in predicting the true LTC status.

4.1.3 RQ 1.3: Which features are the most important in identifying LTCs using past history approach?

Motivation: Although 31 features differentiate LTCs from non-LTCs, some features are likely to be more predictive than others. This research question seeks to identify the most important features that distinguish LTCs from non-LTCs.

Approach: RF model is used to investigate the most important features because the RF exhibited the best performance in predicting LTCs when using 10-fold cross-validation in RQ 1.1.

Following Fan et al., (2018) and Bao et al., (2019), an RF model via 10-fold cross-validation is built. The RF model is trained using the **randomForest** function with the **importance=TRUE** parameter in the **randomForest** package. Feature importance measures are extracted using the **importance** function in the **randomForest** package.

10-fold cross-validation is executed 10 times, where each execution gives us 10 values per feature for a total of 100 importance values per feature. Applying the Scott–Knott effect size difference (ESD) [2] test (Tantithamthavorn et al., 2018) to these 100 importance values helps pinpoint the most important features for the whole dataset. The Wilcoxon rank-sum test is applied for each feature to check for the statistical significance of the difference between LTC and non-LTC contributors. Cliff's delta is used to measure the effect size of the difference between the two groups.

Results: Tables 4.6, 4.7, and 4.8 show the top 10 features based on the Scott-Knott ESD test, the p-values, and Cliff's delta for the three time intervals, respectively. Columns 1-4 are the group number of the feature, feature name, p-values, and Cliff's delta. The symbol ★ next to a feature name denotes that the feature is in the top 10 features in three time intervals.

r_Language and c_is_Core_Member are non-numeric features and thus do not have

[2] `https://github.com/klainfo/ScottKnottESD`.

a p-value or Cliff's delta. Rows with p-values less than 0.001 and non-negligible effect sizes are in the bold font, and new features are in italics. Four contributor features and six repository features appear in the list of the top 10 important features. Nine features appear in the list of the top 10 important features in all three time intervals.

Considering Table 4.8 that shows the top 10 features for the longest term (year 3), the feature c_History_Followers is the most important feature that is statistically significant with a negative Cliff's delta, indicating that the more followers a contributor has, the more likely they would become an LTC. The features c_Age is the second most important feature that is statistically significant with a positive Cliff's delta, which indicates that experienced contributors do not contribute for the long term and less experienced contributors contribute for the long term as revealed in the c_Age boxplot 4.1. The feature r_Age is the third important feature that is statistically significant with a positive Cliff's delta, indicating difficulty for new contributors to become LTCs in long tenure projects as revealed in the r_Age boxplot 4.2. Software engineering literature (Sen, Singh, and Borle, 2012) points that age of the software is positively related to software maturity and advance to improved software quality. But as the project matures, there might be less interesting work, or the learning curve might be higher. The feature r_Number_Of_Watchers is the fourth important feature that is statistically significant with a positive Cliff's delta, which indicates that it may be difficult for new contributors to become LTCs in projects that have many watchers, as new contributors may not receive resources and attention from the project owners. The feature r_Language is the fifth most important feature to predict LTC and potentially plays a role in LTC conversion as shown in Table 4.9.

The feature c_is_Core_Member is the seventh important feature to predict LTC, indicating that members who have rights to merge and commit are more likely to stay with the project for the long term.

The rest of the top 10 repository features are r_Number_of_Contributors, r_Max_All_Contributor_Commits, and r_Max_All_Contributor_Stay. These all have a

Table 4.6: Top 10 important features according to the Scott-Knott ESD test for year 1.

Groups	Features	P-value	Cliff's delta
1	**r_Number_Of_Watchers ★**	**< 0.001**	**0.233 (Small)**
2	**c_History_Followers ★**	**< 0.001**	**-0.287 (Small)**
3	c_Age ★	< 0.001	0.055 (Negligible)
4	**r_Age ★**	**< 0.001**	**0.199 (Small)**
5	r_Max_All_Contributor_Commits ★	< 0.001	0.076 (Negligible)
6	r_Number_of_Contributors ★	< 0.001	0.110 (Negligible)
7	r_Language ★	–	–
8	c_Max_All_Repos_Commits ★	< 0.001	-0.082 (Negligible)
9	c_Number_of_Repos	< 0.004	-0.052 (Negligible)
10	r_Max_All_Contributor_Stay ★	< 0.001	0.113 (Negligible)

Table 4.7: Top 10 important features according to the Scott-Knott ESD test for year 2.

Groups	Features	P-value	Cliff's delta
1	**c_History_Followers ★**	**< 0.001**	**-0.361 (Medium)**
2	**r_Number_Of_Watchers ★**	**< 0.001**	**0.34 (Medium)**
3	**c_Age ★**	**< 0.001**	**0.165 (Small)**
4	**r_Age ★**	**< 0.001**	**0.298 (Small)**
5	**r_Max_All_Contributor_Commits ★**	**< 0.001**	**0.165 (Small)**
6	r_Language ★	–	–
7	**r_Number_of_Contributors ★**	**< 0.001**	**0.238 (Small)**
8	c_Max_All_Repos_Commits ★	< 0.001	-0.04 (Negligible)
9	c_is_Core_Member	–	–
10	**r_Max_All_Contributor_Stay ★**	**< 0.001**	**0.15 (Small)**

positive Cliff's delta, which indicates that it may be difficult to retain new contributors in mature repositories based on the number of commits, contributors, and long staying contributors.

4.2 Experimental Results - Research Objective 2

This section starts with 173 features composed of 63 features from Bao et al., (2019) and an additional 110 features extracted from GHTorrent using BigQuery. Applying feature selection, linearly correlated features are eliminated, reducing the number of features from

Table 4.8: Top 10 important features according to the Scott-Knott ESD test for year 3.

Groups	Features	P-value	Cliff's delta
1	**c_History_Followers ★**	**< 0.001**	**-0.402 (Medium)**
2	**c_Age ★**	**< 0.001**	**0.29 (Small)**
3	**r_Age ★**	**< 0.001**	**0.399 (Medium)**
4	**r_Number_Of_Watchers ★**	**< 0.001**	**0.355 (Medium)**
5	r_Language ★	–	–
6	**r_Max_All_Contributor_Commits ★**	**< 0.001**	**0.178 (Small)**
7	c_is_Core_Member	–	–
8	**r_Number_of_Contributors ★**	**< 0.001**	**0.301 (Small)**
9	c_Max_All_Repos_Commits ★	< 0.05	0.03 (Negligible)
10	r_Max_All_Contributor_Stay ★	< 0.001	0.134 (Negligible)

Figure 4.1. Boxplot of contributor age (days) for LTC and non-LTC groups.

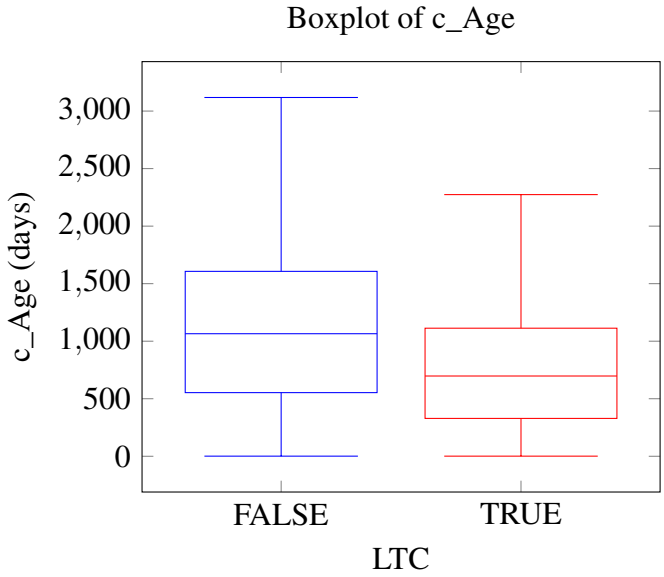


Figure 4.2. Boxplot of repository age (days) for LTC and non-LTC groups.

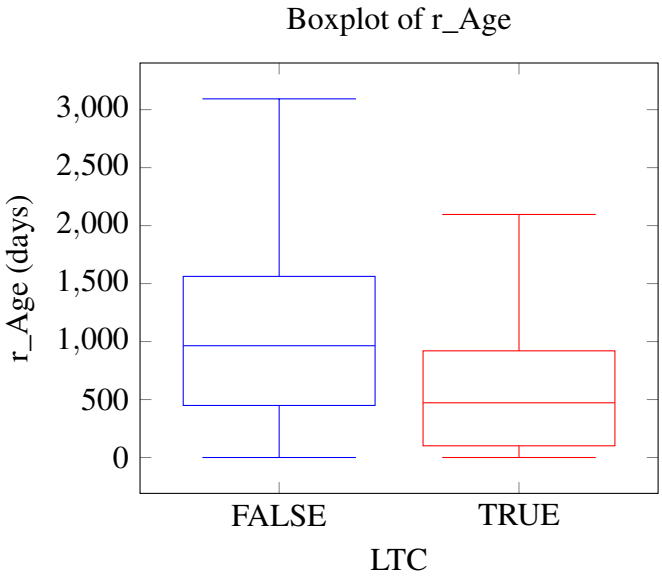


Table 4.9: Top 10 languages with the highest number of LTCs and total contributors.

Language	**# of LTCs**	**Total Contributors**	**% of LTCs**
Ruby	330	13,166	2.5
JavaScript	297	18,852	1.6
Python	211	7,866	2.7
PHP	165	4,454	3.7
C++	141	3,576	3.9
Java	86	3,758	2.3
Scala	49	1,033	4.7
C	41	1,561	2.6
Julia	34	407	8.4
HTML	29	2,036	1.4

173 to 55. The two tests from the benchmark (Bao et al., 2019) using the same software are applied to compare the proposed approach to the benchmark. The first test, the Wilcoxon signed-rank test (Wilcoxon, 1946) with a Bonferroni correction (Abdi et al., 2007), is used to determine whether the improvement over the baseline is statistically significant. The second test, Cliff's delta[3] (Cliff, 2014), is a non-parametric effect size measure used to quantify the difference between the results of both approaches.

4.2.1 RQ 2.1: Can onboarding activity effectively determine an LTC?

Motivation: This research question investigates whether the models using the proposed 55 features (Onboarding_55 approach) perform better than the models using 63 features (Baseline approach) (Bao et al., 2019) with respect to multiple metrics.

Approach: NB, SVM, kNN, LR, DT, and RF prediction models are built using the proposed approach to compare with baseline (Bao et al., 2019). To compare the performance of the proposed approach with NB (Onboarding_55-NB), SVM (Onboarding_55-SVM), kNN (Onboarding_55-kNN), LR (Onboarding_55-LR), DT (Onboarding_55-DT), and RF (Onboarding_55-RF) models with Bao et al., (2019) best result (Baseline with RF model (Baseline-RF)), the average AUC, MCC, accuracy, precision, recall, and F1-scores for LTC and non-LTC contributors are calculated using 10-fold cross-validation for the three time intervals.

Results: Table 4.10 presents the average AUC, MCC, Accuracy, F1-scores, Precision, and Recall for LTC and non-LTC contributors for the proposed Onboarding_55-NB, Onboarding_55-SVM, Onboarding_55-kNN, Onboarding_55-LR, Onboarding_55-DT, Onboarding_55-RF, and Baseline-RF models. Best results are shown in bold. For all metrics, Onboarding_55-LR, Onboarding_55-DT, and Onboarding_55-RF achieve the same or better results compared to the baseline.

[3]Cliff defines delta values less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474 and above 0.474 as negligible, small, medium, and large effect sizes, respectively

Table 4.10: Evaluation metrics for the proposed onboarding activity approach for all models.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	Baseline-RF	0.813	0.319	0.889	0.940	0.896	0.987	0.297	0.679	0.190
	Onboarding_55-NB	0.733	0.211	0.882	0.937	0.884	0.995	0.143	0.711	0.079
	Onboarding_55-SVM	0.729	0.016	0.876	0.934	0.876	**0.999**	0.005	0.280	0.003
	Onboarding_55-kNN	0.782	0.287	0.887	0.939	0.891	0.992	0.236	0.724	0.141
	Onboarding_55-LR	0.837	0.435	**0.901**	0.945	0.910	0.984	0.436	0.731	0.311
	Onboarding_55-DT	0.839	**0.443**	**0.901**	**0.946**	**0.912**	0.982	**0.450**	0.723	**0.327**
	Onboarding_55-RF	**0.844**	0.412	0.899	0.945	0.905	0.988	0.394	**0.764**	0.266
2	Baseline-RF	0.872	0.250	0.949	0.974	0.951	0.997	0.184	0.649	0.108
	Onboarding_55-NB	0.758	0.159	0.947	0.972	0.949	0.997	0.099	0.544	0.055
	Onboarding_55-SVM	0.797	0.033	0.946	0.972	0.946	**0.999**	0.008	0.372	0.004
	Onboarding_55-kNN	0.823	0.218	0.948	0.973	0.950	0.998	0.142	0.668	0.080
	Onboarding_55-LR	0.889	0.372	**0.952**	**0.975**	**0.958**	0.992	**0.350**	0.635	**0.242**
	Onboarding_55-DT	0.887	**0.374**	**0.952**	**0.975**	**0.958**	0.992	0.349	0.643	0.240
	Onboarding_55-RF	**0.896**	0.325	**0.952**	**0.975**	0.954	0.997	0.253	**0.754**	0.152
3	Baseline-RF	0.908	0.144	0.978	**0.989**	0.979	**0.999**	0.075	0.546	0.041
	Onboarding_55-NB	0.746	0.132	0.978	**0.989**	0.979	**0.999**	0.074	0.463	0.041
	Onboarding_55-SVM	0.825	0.039	0.978	**0.989**	0.978	**0.999**	0.006	0.500	0.003
	Onboarding_55-kNN	0.843	0.142	0.978	**0.989**	0.979	**0.999**	0.066	0.620	0.035
	Onboarding_55-LR	**0.938**	**0.326**	**0.979**	**0.989**	**0.982**	0.997	**0.289**	0.579	**0.194**
	Onboarding_55-DT	0.936	0.309	**0.979**	**0.989**	**0.982**	0.997	0.263	0.594	0.170
	Onboarding_55-RF	0.935	0.222	**0.979**	**0.989**	0.980	**0.999**	0.143	**0.648**	0.080

Table 4.11 presents the adjusted p-values and Cliff's delta values comparing the AUC, MCC, accuracy F1(non-LTC), and F1(LTC) of Baseline-RF_55-RF model with Onboarding_55-NB, Onboarding_55-SVM, Onboarding_55-kNN, Onboarding_55-LR, Onboarding_55-DT, and Onboarding_55-RF using 10-fold cross-validation with 1, 2, and 3 year time intervals. Except for accuracy and F1(non-LTC), the results are statistically significant at a 99% confidence level or higher, with a large effect sizes. Compared with only the AUC metric, Onboarding_55-RF outperforms Baseline-RF, which is statically significant at a 99.95% confidence level.

These results indicate that combining the features proposed in this research with features from Bao et al., (2019), removing uncorrelated, redundant features, and building LR, DT, and RF models are beneficial for detecting LTCs under both threshold-dependent and threshold-independent metrics.

Table 4.11: Cliff's delta and p-values comparing onboarding activity models with Bao et al., (2019) RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	F1(LTC)
1	Onboarding_55-NB	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***
	Onboarding_55-SVM	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***
	Onboarding_55-kNN	-1 (Large) ***	-0.82 (Large) **	-0.44 (Medium) ^{i}	-0.28 (Small) ^{i}	-1 (Large) ***
	Onboarding_55-LR	0.96 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	Onboarding_55-DT	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
	Onboarding_55-RF	1 (Large) ***	11 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
2	Onboarding_55-NB	-1 (Large) ***	-1 (Large) ***	-0.87 (Large) **	-0.86 (Large) ***	-1 (Large) ***
	Onboarding_55-SVM	-1 (Large) ***	-1 (Large) ***	-1 (Large) ***	-0.86 (Large) ***	-1 (Large) ***
	Onboarding_55-kNN	-1 (Large) ***	-0.54 (Large) ^{i}	-0.27 (Small) ^{i}	-0.24 (Small) ^{i}	-0.86 (Large) ***
	Onboarding_55-LR	1 (Large) ***	1 (Large) ***	0.93 (Large) ***	0.88 (Large) ***	1 (Large) ***
	Onboarding_55-DT	0.98 (Large) ***	1 (Large) ***	0.76 (Large) *	0.7 (Large) ^{i}	1 (Large) ***
	Onboarding_55-RF	0.92 (Large) ***	0.96 (Large) ***	0.96 (Large) **	0.96 (Large) **	1 (Large) ***
3	Onboarding_55-NB	-1 (Large) ***	-0.1 (Negligible) ^{i}	-0.32 (Small) ^{i}	-0.34 (Medium) ^{i}	0.06 (Negligible) ^{i}
	Onboarding_55-SVM	-1 (Large) ***	-1 (Large) ***	0.2 (Small) ^{i}	0.24 (Small) ^{i}	-1 (Large) ***
	Onboarding_55-kNN	-1 (Large) ***	0.04 (Negligible) ^{i}	0.2 (Small) ^{i}	0.24 (Small) ^{i}	0.03 (Negligible) ^{i}
	Onboarding_55-LR	0.96 (Large) ***	1 (Large) ***	0.55 (Large) ^{i}	0.52 (Large) ^{i}	1 (Large) ***
	Onboarding_55-DT	0.98 (Large) ***	1 (Large) ***	0.62 (Large) *	0.6 (Large) *	1 (Large) ***
	Onboarding_55-RF	0.94 (Large) ***	0.78 (Large) *	0.58 (Large) ^{i}	0.55 (Large) ^{i}	0.83 (Large) *

$^{****}p < 0.001$, $^{***}p < 0.005$, $^{**}p < 0.01$, $^{*}p < 0.05$, $^{i}p > 0.05$ (insignificant)

4.2.2 RQ 2.2: How effective is the onboarding activity approach in determining an LTC compared to the state-of-the-art approach?

Motivation: As part of RQ 2.1, based on the AUC, that RF model using the proposed 55 features performs well. This research question investigates the magnitude of the improvement of RF models using 55 features (Onboarding_55-RF) compared to using 63 features (Baseline-RF approach) (Bao et al., 2019) with respect to multiple metrics.

Approach: To compare the performance of Onboarding_55-RF models with Baseline-RF (Bao et al., 2019), the average AUC, MCC, accuracy, precision, recall, and F1-scores for LTC and non-LTC contributors are calculated using 10-fold cross-validation for the three time intervals.

Results: Table 4.12 presents the average AUC, MCC, Accuracy, F1-scores, Precision, and Recall for LTC and non-LTC contributors for the proposed Onboarding_55-RF and Baseline-RF models using 10-fold cross-validation with 1, 2, and 3 year time intervals. The best results are in bold. Overall, the proposed method achieves better results than Baseline-RF across all metrics.

Table 4.12: Evaluation metrics for the proposed onboarding activity RF model and Bao et al., (2019) RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	Baseline-RF	0.813	0.319	**0.889**	0.940	0.896	0.987	0.297	0.679	0.190
	Onboarding_55-RF	**0.844**	**0.412**	**0.899**	**0.945**	**0.905**	**0.988**	**0.394**	**0.764**	**0.266**
2	Baseline-RF	0.872	0.250	0.949	0.974	0.951	**0.997**	0.184	0.649	0.108
	Onboarding_55-RF	**0.896**	**0.325**	**0.952**	**0.975**	**0.954**	**0.997**	**0.253**	**0.754**	**0.152**
3	Baseline-RF	0.908	0.144	0.978	**0.989**	0.979	**0.999**	0.075	0.546	0.041
	Onboarding_55-RF	**0.935**	**0.222**	**0.979**	**0.989**	**0.980**	**0.999**	**0.143**	**0.648**	**0.080**

Table 4.13 presents the percentage gain in AUC, MCC, Accuracy, F1-scores, Precision, and Recall for LTC and non-LTC contributors for the Onboarding_55-RF model with Baseline-RF using 10-fold cross-validation with 1, 2, and 3 year time intervals. % gain is computed using (((PastHistory_55-RF - Baseline-RF)/Baseline-RF) * 100). The proposed models achieve gains between 3% to 4%, 29% to 54%, 0% to 1%, 0% to 1%, 0% to 1%, 0%, 33% to 91%, 13% to 19%, and 40% to 95% for the AUC, MCC, accuracy, F1-score(non-LTC), precision(non-LTC), recall(non-LTC), F1-score(LTC), precision(LTC), and recall(LTC), respectively, compared to the baseline.

Table 4.14 presents the adjusted p-values and Cliff's delta values comparing the AUC, MCC, accuracy F1(non-LTC), and F1(LTC) of Onboarding_55-RF model with Baseline-RF using 10-fold cross-validation with 1, 2, and 3 year time intervals. All metric results are statistically significant at the 99% confidence level or higher, with a large effect sizes with two exceptions. Accuracy and F1(non-LTC) are not statistically significant for the time interval of 3 years.

These results indicate that combining the proposed features with features from Bao et al., (2019), removing uncorrelated, redundant features, and building RF models is beneficial for detecting LTCs under both threshold-dependent and threshold-independent metrics.

4.2.3 RQ 2.3: How effective is the feature selection method?

Motivation: This research question evaluates whether the prediction model benefits from using all 173 features (Onboarding_173-RF approach) compared to using 55 features

Table 4.13: Percentage gain in evaluation metrics for the proposed onboarding activity RF model and Bao et al., (2019) RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	Onboarding_55-RF	4%	29%	1%	1%	1%	0%	33%	13%	40%
2	Onboarding_55-RF	3%	30%	0%	0%	0%	0%	38%	16%	41%
3	Onboarding_55-RF	3%	54%	0%	0%	0%	0%	91%	19%	95%

Table 4.14: Cliff's delta and p-values comparing the proposed onboarding activity and Bao et al., (2019) RF models.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	F1(LTC)
1	Onboarding_55-RF	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***	1 (Large) ***
2	Onboarding_55-RF	0.92 (Large) ***	0.96 (Large) ***	0.96 (Large) **	0.96 (Large) **	1 (Large) ***
3	Onboarding_55-RF	0.94 (Large) ***	0.78 (Large) *	0.58 (Large) ^{i}	0.55 (Large) ^{i}	0.83 (Large) *

$^{****}p < 0.001$, $^{***}p < 0.005$, $^{**}p < 0.01$, $^{*}p < 0.05$, $^{i}p > 0.05$ (insignificant)

(Onboarding_55-RF approach) because the 173 features may have some nonlinear aspects not captured by the 55 features. This research question seeks to determine if the prediction models built on all features are better than those built with optimized features.

Approach: In RQ 2.1, RF models performed better, making it obvious to build RF models with the same parameters used in RQ 2.1 with all 173 features. Nine measures applied in RQ 2.1 to evaluate the performance of the prediction models by taking the average results of 10-fold cross-validation are used to compare the models built based on all 173 features with those built based on the optimized 55 features.

Results: Table 4.15 presents the average AUC, MCC, accuracy, precision, recall, F1-scores for LTC, and non-LTC contributors for the time intervals of 1, 2, and 3 years obtained via 10-fold cross-validation for Onboarding_173-RF and Onboarding_55-RF. Across all three time intervals and all nine metrics, for a total of 27 comparisons, Onboarding_55-RF has equal or higher values than Onboarding_173-RF for 26 measurements. Table 4.16 presents the adjusted p-values and Cliff's delta to compare the AUC, MCC, accuracy, F1(non-LTC), and F1(LTC) of the two approaches: most results are not statistically significant.

These results indicate that Onboarding_55-RF is better than or equal to Onboarding_173-RF, indicating that the feature selection method was effective in preserving useful informa-

Table 4.15: Evaluation metrics for the proposed onboarding RF models - 173 and 55 features RF models.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	P(non-LTC)	R(non-LTC)	F1(LTC)	P(LTC)	R(LTC)
1	Onboarding_173	**0.847**	0.396	0.897	0.944	0.903	**0.988**	0.377	0.752	0.252
	Onboarding_55	0.844	**0.412**	**0.899**	**0.945**	**0.905**	**0.988**	**0.394**	**0.764**	**0.266**
2	Onboarding_173	0.893	0.290	0.950	0.974	0.953	**0.997**	0.217	0.724	0.128
	Onboarding_55	**0.896**	**0.325**	**0.952**	**0.975**	**0.954**	**0.997**	**0.253**	**0.754**	**0.152**
3	Onboarding_173	0.934	0.181	**0.979**	**0.989**	0.979	**0.999**	0.103	0.614	0.057
	Onboarding_55	**0.935**	**0.222**	**0.979**	**0.989**	**0.980**	**0.999**	**0.143**	**0.648**	**0.080**

Table 4.16: Cliff's delta and p-values comparing the proposed onboarding activity RF models - 173 feature RF model and 55 features RF model.

Year	Approach	AUC	MCC	Accuracy	F1(non-LTC)	F1(LTC)
1	Onboarding_55	-0.28 (Small) ^{i}	0.44 (Medium) ^{i}	0.35 (Medium) ^{i}	0.32 (Small) ^{i}	0.54 (Large) *
2	Onboarding_55	0.24 (Small) ^{i}	0.64 (Large) *	0.54 (Large) *	0.52 (Large) *	0.62 (Large) *
3	Onboarding_55	-0.04 (Negligible) ^{i}	0.60 (Large) ^{i}	0.45 (Medium) ^{i}	0.41 (Medium) ^{i}	0.66 (Large) *

$^{****}p < 0.001$, $^{***}p < 0.005$, $^{**}p < 0.01$, $^{*}p < 0.05$, $^{i}p > 0.05$ (insignificant)

tion for detecting LTC contributors.

4.2.4 RQ 2.4: Which features are the most important in identifying LTCs using onboarding activity?

Motivation: Although 55 features differentiate LTCs from non-LTCs, some features are likely to be more predictive than others. This research question seeks to identify the most important features that distinguish LTCs from non-LTCs.

Approach: RF model is used to investigate the most important features because the RF has the best performance in predicting LTCs when using 10-fold cross-validation in RQ 2.1.

Following Fan et al., (2018) and Bao et al., (2019), an RF model via 10-fold cross-validation is built. The RF model is trained using the **randomForest** function with the **importance=TRUE** parameter in the **randomForest** package. Feature importance measures are extracted using the **importance** function in the **randomForest** package.

Executing 10-fold cross-validation 10 times, where each execution gives 10 values per feature for a total of 100 importance values per feature. The Scott-Knott effect size

difference (ESD) [4] test (Tantithamthavorn et al., 2018) is applied to these 100 importance values to identify the most important features for the whole dataset. The Wilcoxon rank-sum test is applied for each feature to check for the statistical significance of the difference between LTC and non-LTC contributors. Moreover, Cliff's delta is used to measure the effect size of the difference between the two groups.

Results: Tables 4.17, 4.18, and 4.19 show the top 10 features based on the Scott-Knott ESD test, the p-values, and Cliff's delta for the three time intervals, respectively. Columns 1-4 are the group number of the feature, feature name, p-values, and Cliff's delta. The symbol ★ next to a feature name denotes that the feature is in the top 10 features in all three time intervals.

Language is a non-numeric feature and thus does not have a p-value or Cliff's delta. Rows with p-values less than 0.001 and non-negligible effect sizes are in bold font, and new features are in italics.

Nine features appear in the list of the top 10 important features in all three time intervals: three_month_user_pull_request_history_closed, three_month_user_pull_request_history_merged, user_history_followers, language, before_repo_contributor_mean, before_repo_commits, month_user_commits, user_age, and before_repo_pull_requests. Among these features, four are from the 22 newly added features. Two features are in the top 10 in all three time intervals: three_month_user_pull_request_history_closed and three_month_user_pull_request_history_merged, i.e., the number of closed/merged events received in the pull requests submitted to the repository in the first three months after the new contributor joins. The statistical significance test and Cliff's delta values for these two features show that a larger number of submitted pull requests merged and closed in three months contributes to the conversion of contributors to LTCs (this is potentially an indication of positive feedback and satisfaction to contributors). The feature

[4] https://github.com/klainfo/ScottKnottESD

Table 4.17: Top 10 important features according to the Scott-Knott ESD test for year 1.

Groups	Features	P-value	Cliff's delta
1	***three_month_user_pull_request_history_closed*** ★	$<$ **0.001**	**-0.316 (Small)**
2	language ★	–	–
3	**month_user_commits**	$<$ **0.001**	**-0.334 (Medium)**
	user_history_followers ★	$<$ **0.00**1	**-0.287 (Small)**
4	***three_month_user_pull_request_history_merged*** ★	$<$ **0.001**	**-0.213 (Small)**
5	**before_repo_watchers**	$<$ **0.001**	**0.232 (Small)**
6	before_repo_contributor_mean ★	$<$ 0.001	-0.031 (Negligible)
	before_repo_commits ★	$<$ 0.001	0.046 (Negligible)
7	before_repo_pull_requests ★	$<$ 0.001	0.134 (Negligible)
8	user_age	$<$ 0.001	0.055 (Negligible)

Table 4.18: Top 10 important features according to the Scott-Knott ESD test for year 2.

Groups	Features	P-value	Cliff's delta
1	***three_month_user_pull_request_history_closed*** ★	$<$ **0.001**	**-0.408 (Medium)**
2	language ★	–	–
3	**user_history_followers** ★	$<$ **0.00**1	**-0.361 (Medium)**
4	***three_month_user_pull_request_history_merged*** ★	$<$ **0.001**	**-0.291 (Small)**
5	before_repo_contributor_mean ★	$<$ 0.6	-0.006 (Negligible)
6	**before_repo_pull_requests** ★	$<$ **0.001**	**0.29 (Small)**
7	**before_repo_commits** ★	$<$ **0.001**	**0.152 (Small)**
8	***one_month_before_repo_watchers*** ★	$<$ **0.001**	**0.333 (Medium)**
9	before_repo_watchers	$<$ 0.001	0.339 (Medium)
10	***one_month_after_repo_watchers***	$<$ **0.001**	**0.292 (Small)**

one_month_before_repo_watchers is another new feature that is statistically significant with a positive Cliff's delta for the 1-year time interval, which indicates that it may be difficult for new contributors to become LTCs in projects that have many watchers in the month before the new contributor joins, as new contributors may not receive resources and attention from the project owners. repo_age is the other new feature that is statistically significant with a positive Cliff's delta in the 3-year time interval, which indicates that it is difficult for new contributors to become LTCs in a mature project, as there might be less interesting work or the learning curve might be higher. The rest of the 10 most important features are similar to those in Bao et al., (2019).

Table 4.19: Top 10 important features according to the Scott-Knott ESD test for year 3.

Groups	Features	P-value	Cliff's delta
1	***three_month_user_pull_request_history_closed*** ★	$<$ **0.001**	**-0.523 (Large)**
2	***three_month_user_pull_request_history_merged*** ★	$<$ **0.001**	**-0.423 (Medium)**
3	language ★	–	–
4	**user_history_followers** ★	$<$ **0.00**1	**-0.402 (Medium)**
5	before_repo_contributor_mean ★	$<$ 0.5	-0.018 (Negligible)
6	***one_month_after_repo_watchers***	$<$ **0.001**	**0.455 (Medium)**
7	**before_repo_commits** ★	$<$ **0.001**	**0.175 (Small)**
8	**before_repo_pull_requests** ★	$<$ **0.001**	**0.425 (Medium)**
9	**user_age**	$<$ **0.001**	**0.29 (Small)**
10	***one_month_before_repo_watchers***	$<$ **0.001**	**0.423 (Medium)**

4.3 Threats to Validity

4.3.1 Internal Validity

Internal validity is the process of determining if the independent variable causes the observed changes in the dependent variables. A statistical methodology is used to show relationships in this work, but it cannot establish cause-and-effect relationships. A different methodology, such as participatory observation or a qualitative social research method, would be needed to verify causality.

To avoid introducing any additional threats, this study replicates the methodology used by Bao et al., (2019). In addition, the same projects are used (duplicate projects are removed), contributors, and new contributor joining date when extracting new features. Moreover, the same 10-fold cross-validation approach is used. In addition, the same software is used that was shared by the authors Bao et al., (2019) to reproduce the results presented in this research successfully.

4.3.2 External Validity

External validity assesses the generalizability of the outcomes. 888 GitHub projects that cover multiple programming languages and various application domains are used, and hence

the results can be generalizable.

4.3.3 Construct Validity

The construct validity of measurements analyzes whether they accurately measure what they are intended to measure. Based on previous research (Tantithamthavorn et al., 2018), the proposed models' performance is quantified using nine classic threshold-dependent and threshold-independent criteria to generalize and eliminate bias in this research findings. The threshold-dependent performance measures are calculated by converting the prediction probabilities to a binary classification (LTC or non-LTC) with a default cut-off of 0.5; if a contributor record has a predicted probability greater than the cut-off, the contributor record is considered to be LTC; otherwise, the contributor is classified as a non-LTC. As summarized in previous research ([58]), threshold-dependent performance metrics are problematic due to their dependency on an arbitrary threshold and sensitivity to class imbalances. These measures are used with the same threshold (0.5) to avoid introducing bias.

Chapter 5: Discussion

5.1 Evaluating the Validity of the Hypotheses

The hypothesis proposed in Chapter 1 were as follows:

Research Hypothesis 1: With regard to the problem of predicting LTCs based on standard comparative metrics of accuracy, precision, recall, F1-score, MCC, and AUC, the proposed "past history" based model outperforms the known state-of-the-art approach.

Research Hypothesis 2: With regard to the problem of predicting LTCs based on standard comparative metrics of accuracy, precision, recall, F1-score, MCC, and AUC, the proposed "onboarding activity" based model outperforms the known state-of-the-art approach.

Evaluation: Table 4.3, 4.4, and 4.5 present metrics (accuracy, precision, recall, F1-score, MCC, and AUC), gain, and statistical significance for the proposed past history based model and state-of-the-art approach using 10-fold cross-validation with 1, 2, and 3-year time intervals.

Based on the evidence gathered through experimentation, it was found that Hypothesis 1 is partially supported as the state-of-the-art Bao et al., (2019) model performs better than the proposed model for the time interval of 1 year. For time intervals of 2 and 3 years, the proposed model performs better than the state-of-the-art Bao et al., (2019) model.

Table 4.12, 4.13, and 4.14 present metrics (accuracy, precision, recall, F1-score, MCC, and AUC), gain, and statistical significance for the proposed onboarding activity based model and state-of-the-art approach using 10-fold cross-validation with 1, 2, and 3 year time intervals.

Based on the evidence gathered through experimentation, it was found that Hypothesis 2 is supported as the proposed model performs better than the state-of-the-art Bao et al.,

(2019) model for the time interval of 1, 2, and 3-years.

5.2 Recommendations

Capturing, engaging, and sustaining new contributors is critical for OSS project survival. In this study, two data sets are created. The first dataset has 31 factors that contribute to LTC conversion and effectively predicted LTC conversion. The second dataset has 173 features composed of 63 features from Bao et al., (2019) and an additional 110 features extracted from GHTorrent using BigQuery. Feature selection is applied to eliminate linearly correlated features reducing the number of features from 173 to 55.

Based on the experiments and analysis of the results, in this study, five recommendations are offered to OSS project owners:

5.2.1 Automate assignment of integrating newcomer contributions.

It is found that the c_Age feature is an important predictor for retaining new contributors over the long term. The feature c_Age has a positive Cliff's delta, which indicates that experienced contributors do not contribute for the long term and less experienced contributors contribute for the long term, as revealed in the c_Age boxplot 4.1. Making it easier for new contributors to contribute for the long term would be a good investment. A new contributor can submit a pull request, which notifies others of changes the new contributor has pushed to a branch in a GitHub project. However, new contributors will not merge pull requests until the required number of reviewers have approved the request. Once the necessary feedback is provided, the pull request is merged or closed without merging if the changes are no longer needed.

Thus, after a new contributor makes a code contribution and submits a pull request, the project owners can encourage the new contributor by providing feedback and merging or closing the pull request. Assigning an integrator to assess and integrate a pull request

is a laborious and time-consuming process for project owners, resulting in an increasing amount of pull requests remaining to be assessed. Lima Júnior et al., (2018) developed a method to identify a suitable integrator that is 54% better than the state-of-the-art method and can be used by project owners to automate the assignment of integrators to merge/close pull requests within 30 days to encourage new contributors to participate in the long term. Project owners could adopt the automated assignment of integrators to increase the chance of converting new contributors to LTC.

5.2.2 Allocate resources to evaluate a new contributor and provide core member privilege.

The c_is_Core_Member feature is an important predictor for retaining new contributors over the long term. While project owners can automate assigning integrators to merge or close the pull request of new contributors, the assignment is limited by the number of integrators available. Project owners could allocate resources to evaluate a new contributor's experience and provide core member privilege. Providing core member privileges to eligible new contributors would increase the new contributor's chance to convert to LTC and increase the integrator resource pool to help other new contributors commit integration promptly.

5.2.3 Allocate resources for documenting various ways to contribute.

The r_Age, r_Max_All_Contributor_Commits and, r_Number_of_Contributors features are an important predictor for retaining new contributors over the long term. High values for features indicate mature projects. The r_Age, r_Max_All_Contributor_Commits and r_Number_of_Contributors features have positive Cliff's delta, indicating that it is difficult for new contributors to become LTCs in mature projects. To gain further insight, the oldest 10 repositories are analyzed. Table 5.1 shows the oldest 10 repositories, where the first column displays the project name, the second

column shows Date Of Creation (DOC), the third column displays #LTC / #non-LTC (Class), the fourth column shows Documentation Quality (DQ), and the fifth column displayed the Status and Activity in Sept 2020. It was found that one project is deprecated, five projects are in maintenance mode, and four projects are active. All five repositories in maintenance mode have no invitation to contribute and sparse instructions on how to contribute. The four repositories in active status have an invitation and clear instructions on how to contribute. The two repositories with the highest number of LTCs (Spree and Rails) have invited to contribute and provide various ways of contributing to the repository.

For newcomers in mature repositories, the various ways they can contribute to a project may not be clear. Thus, project owners can encourage new contributors by clearly inviting and providing them with instructions on contributing in various ways. The following are a few ways mature repositories can invite newcomers to contribute: releasing code, reporting bugs, translating to new languages, writing documentation, writing specs, writing code, refactoring code, reviewing code, and verifying issues.

Table 5.1: Oldest 10 repository details.

Project	DOC	Class	DQ	Status and Sept 2020 Activity
A tree explorer plugin for Vim.	2008-03-10	0 / 11	Clear documentation. No invitation to contribute. No contribution instructions.	Maintenance — There has not been any commit activity.
An e-commerce solution for Ruby on Rails.	2008-03-10	20 / 447	Clear documentation. Invitation to contribute with clear instructions.	Active — 17 authors have pushed 48 commits to master and 62 commits to all branches.
A Ruby on Rails form builder that uses semantically rich and accessible markup.	2008-04-07	1 / 49	Clear documentation. No invitation to contribute. Contribution instructions were provided.	Maintenance — There has not been any commit activity.
Easy file attachment management for ActiveRecord.	2008-04-10	3 / 140	Clear documentation. No invitation to contribute. Contribution instructions were provided.	Maintenance — There has not been any commit activity.
Ruby on Rails.	2008-04-11	165 / 1558	Clear documentation. Contribution invitation with clear instructions provided.	Active — 55 authors have pushed 115 commits to master and 131 commits to all branches.
Liquid markup language. For versatile online applications, a secure, customer-facing template language.	2008-05-08	3 / 42	Clear documentation. Contribution invitation but unfriendly language is used on what will be and will not be merged.	Active — 1 author has pushed 1 commit to master and 1 commit to all branches.
A library that facilitates the process of creating Ruby objects as test data.	2008-06-03	3 / 61	Clear documentation. Contribution instructions with clear instructions provided.	Active — 1 author has pushed 14 commits to master and 27 commits to all branches.
The code that powers reddit.com.	2008-06-18	8 / 99	Clear documentation. No invitation to contribute.	Maintenance — There has not been any commit activity.
CFNetwork wrapper for HTTP requests that are easy to use.	2008-07-14	0 / 26	Clear documentation. No invitation to contribute.	Maintenance — There has not been any commit activity.
Compass is a stylesheet authoring environment that makes website design simpler to implement.	2008-08-23	0 / 8	Clear documentation. No invitation to contribute.	Deprecated — There has not been any commit activity.

5.2.4 Allocate resources to merge and close pull requests from new contributors in the first three months.

The three_month_user_pull_request_history_closed and three_month_user_pull_request_history_merged are among the top four most important predictors for retaining new contributors over the long term across all the time intervals. Any new contributor can submit a pull request, which notifies others of changes the new contributor has pushed to a branch in a GitHub project. However, new contributors will not be able to merge pull requests until the required number of reviewers have approved the request. Once the necessary feedback is provided, the pull request is merged or closed without merging if the changes are no longer needed.

Thus, after a new contributor makes a code contribution and submits a pull request, the project owners can encourage the new contributor by providing feedback and merging or closing the pull request. This process appears to improve the conversion of new contributors to LTCs.

5.2.5 Allocate resources to onboard newcomers in the first month.

The one_month_before_repo_watchers and one_month_after_repo_watchers are among the top 10 important predictors for retaining new contributors over the long term across time intervals 2 and 3. These features are negatively correlated with LTC status with -0.14 and -0.10, respectively, for the one-year time interval.

Sheoran et al., (2014) discovered that a subset of project observers began contributing to the project and account for a sizable portion of the project's contributors. Due to the notifications they receive from the project, these watchers are aware of the project and its workflow before making their first contribution. Suspecting that project owners have to invest resources to convert watchers to contributors by making sure that the notifications are clear. If a project has many watchers just one month before and one month after a

new contributor joins, project owners must have to balance resource allocation to convert watchers to contributors and convert contributors to LTC. Thus, after a new contributor joins a project, the project owners can allocate time to onboard the new contributor to avoid losing them because the current process impacts the conversion of new contributors to LTCs.

5.3 Conclusion

In this dissertation, using software development data from GitHub, two novel feature approaches are proposed to build ML models to predict whether a new contributor will become an LTC of a project. The state-of-the-art (baseline) approach was published by Bao et al., (2019), where 63 features related to contributor and repository activities until the first month after a contributor joins the repository was used to predict LTCs by developing ML models. The RF model was the best in Bao et al., (2019) with respect to the AUC, MCC, accuracy, F1-score for LTCs and non-LTCs, and precision for LTCs and non-LTCs, and recall for LTCs and non-LTCs.

This research introduced a novel 31 feature past history approach and compared it to the 63 features (Bao et al., 2019) across multiple metrics. There are two differences between the proposed 31 feature set compared to Bao et al., (2019) feature set: (1) less than 50% of features (31 vs. 63) are used, and (2) the proposed features have no wait time of one month after the contributor joins. For a time interval of 3 years, the proposed model performs better than Bao et al., (2019) RF model with the proposed models precision, recall, F1-score, MCC, and AUC values of 0.695, 0.079, 0.140, 0.226, and 0.913, respectively, which are 27.29%, 92.68%, 86.67%, 56.94%, and 0.55%, respectively better than the best baseline model. These results show that the research objective 1: Develop a model that predicts LTCs based on the activities of the new contributor and all contributors involved in the repository before the first contribution by the new contributor more effectively than the state-of-the-art approach was accomplished.

In addition, another novel 55 feature onboarding activity approach was introduced in this

research study and compared to the 63 features (Bao et al., 2019) across multiple metrics. There are two differences between the proposed 55 feature set compared to Bao et al., (2019) feature set: (1) fewer features (55 vs. 63) are used, and (2) the proposed features capture activities that a new contributor performs during the onboarding time. For the time interval of 3 years, the proposed model performs better than Bao et al., (2019) RF model with the proposed models precision, recall, F1-score, MCC, and AUC values of 0.648, 0.080, 0.143, 0.222, and 0.935, respectively, which are 19%, 95%, 91%, 54%, and 3%, respectively better than the best baseline model. These results show that research objective 2: Develop a model that predicts LTCs based on contributor programming language skills, project age, and contributor time to on-board based on the activities of the new contributor and all contributors involved for the most recent one month, two months, and three months before and after the new contributor joins the project more effectively than state-of-the-art approach was accomplished.

5.4 Future Work

Future studies can explore a few areas like models based on ensemble methods, code complexity features, file level change features, technical forum discussion features, and validating important model features.

This research builds NB, SVM, kNN, LR, DT, and RF prediction models using various features. The effectiveness of the models is measured using AUC, MCC, accuracy, precision, recall, and F1-scores metrics. It is found that different models performed better on different metrics. One recommendation for future work is to build ensemble models that aggregate the results of all the individual models and benefit from all models.

This research considers features from development activities to identify the challenges facing new contributors from becoming LTCs. Complexity also arises from code and the number of file-level changes in the project. Another recommendation is to capture code complexity and file-level change features and build models to understand better the role of

code and file level change in converting contributors to LTCs.

There are many technical forums like stack-overflow where members ask questions on OSS projects in the current environment. These questions can provide valuable information into the complexity and challenges of OSS projects. Like previous recommendations, capturing features from technical forums and building models could generate more insights for OSS project owners.

This research uses a statistical methodology to show relationships, but it cannot establish cause-and-effect relationships. A different methodology, such as participatory observation or a qualitative social research method, would be needed to verify causality. The last recommendation for future work is developing and implementing an experiment that would establish cause-and-effect.

Bibliography

[1] Hervé Abdi et al. “Bonferroni and Šidák corrections for multiple comparisons”. In: *Encyclopedia of measurement and statistics* 3 (2007), pp. 103–107.

[2] David W Aha, Dennis Kibler, and Marc K Albert. “Instance-based learning algorithms”. In: *Machine learning* 6.1 (1991), pp. 37–66.

[3] Shaosong Ou Alexander Hars. “Working for Free? Motivations for Participating in Open-Source Projects”. In: *International Journal of Electronic Commerce* 6.3 (2002), pp. 25–39. DOI: 10.1080/10864415.2002.11044241. eprint: https://doi.org/10.1080/10864415.2002.11044241. URL: https://doi.org/10.1080/10864415.2002.11044241.

[4] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, Cambridge, MA, USA, 2014.

[5] Rastogi Ayushi. “Do biases related to geographical location influence work-related decisions in GitHub?” In: *2016 IEEE/ACM 38th IEEE International Conference on Software Engineering*. Vol. 1. IEEE. 2016, pp. 665–667.

[6] Lingfeng Bao et al. “A large scale study of long-time contributor prediction for GitHub projects”. In: *IEEE Transactions on Software Engineering* (2019).

[7] Hind Benbya and Nassim Belbaly. “Understanding developers’ motives in open source projects: a multi-theoretical framework”. In: *Communications of the AIS* 27 (2010), pp. 589–610.

[8] Christian Bird et al. “Open borders? immigration in open source projects”. In: *Fourth International Workshop on Mining Software Repositories (MSR’07: ICSE Workshops 2007)*. IEEE. 2007, pp. 6–6.

[9] Kelly Blincoe et al. “Understanding the popular users: Following, affiliation influence and leadership on GitHub”. In: *Information and Software Technology* 70 (2016), pp. 30–39.

[10] Hudson Borges and Marco Tulio Valente. “What’s in a GitHub star? understanding repository starring practices in a social coding platform”. In: *Journal of Systems and Software* 146 (2018), pp. 112–129.

[11] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[12] K Carillo, S Huff, and B Chawner. “What makes a good contributor? Understanding contributor behavior within large Free/Open Source Software projects – A socialization perspective”. In: *Journal of Strategic Information Systems* 26.4 (2017), pp. 322–359.

[13] Norman Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[14] Jailton Coelho et al. “Identifying unmaintained projects in github”. In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2018, pp. 1–10.

[15] Jailton Coelho et al. “Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects”. In: *Information and Software Technology* 122 (2020), p. 106274.

[16] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine Learning* 20 (1995), pp. 273–297.

[17] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. “A systematic mapping study of software development with GitHub”. In: *IEEE Access* 5 (2017), pp. 7173–7192.

[18] Kevin Crowston. “Lessons from Volunteering and FreeLibre Open Source Software Development for the Future of Work”. In: *Researching the Future in Information Systems*. Springer Berlin Heidelberg, 2011, pp. 215–229. ISBN: 978-3-642-21364-9.

[19] Laura Dabbish et al. “Social coding in GitHub: transparency and collaboration in an open software repository”. In: *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 2012, pp. 1277–1286.

[20] Jesse Davis and Mark Goadrich. “The relationship between Precision-Recall and ROC curves”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[21] Alexandre Decan et al. “GAP: Forecasting commit activity in git projects”. In: *Journal of Systems and Software* 165 (2020), p. 110573.

[22] Ikram El Asri et al. “From Periphery to Core: A Temporal Analysis of Github Contributors’ Collaboration Network.” In: *IFIP Advances in Information and Communication Technology*. Vol. 506. 2017, pp. 217–229.

[23] Omar Elazhary et al. “Do as i do, not as i say: Do contribution guidelines match the github contribution process?” In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2019, pp. 286–290.

[24] Vijaya Kumar Eluri, Thomas A. Mazzuchi, and Shahram Sarkani. “Predicting long-time contributors for GitHub projects using machine learning”. In: *Information and Software Technology* 138 (2021), p. 106616. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2021.106616. URL: https://www.sciencedirect.com/science/article/pii/S0950584921000902.

[25] Vijaya Kumar Eluri, Shahram Sarkani, and Thomas A Mazzuchi. “Open Source Software Survivability Prediction Using Multi Layer Perceptron”. In: *Proceedings of 28th International Conference*. Vol. 64. 2019, pp. 148–157.

[26] Yuanrui Fan et al. “Chaff from the wheat: Characterizing and determining valid bug reports”. In: *IEEE transactions on software engineering* 46.5 (2018), pp. 495–525.

[27] Georgios Gousios. “The GHTorent dataset and tool suite”. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2013, pp. 233–236.

[28] Guido Hertel, Sven Niedner, and Stefanie Herrmann. “Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel”. In: *Research policy* 32.7 (2003), pp. 1159–1177.

[29] F. Herzberg. *Work and the nature of man*. World, 1966.

[30] Yan Hu et al. “Influence analysis of Github repositories”. In: *SpringerPlus* 5.1 (2016), pp. 1–19.

[31] Yan Hu et al. “User influence analysis for github developer social networks”. In: *Expert Systems with Applications* 108 (2018), pp. 108–118.

[32] Akinori Ihara, Akito Monden, and Ken Ichi Matsumoto. “Industry questions about open source software in business: Research directions and potential answers”. In: *2014 6th International Workshop on Empirical Software Engineering in Practice*. IEEE. 2014, pp. 55–59.

[33] Oskar Jarczyk et al. “Surgical teams on GitHub: Modeling performance of GitHub project development processes”. In: *Information and Software Technology* 100 (2018), pp. 32–46.

[34] Jing Jiang et al. “A first look at unfollowing behavior on GitHub”. In: *Information and Software Technology* 105 (2019), pp. 150–160.

[35] Jing Jiang et al. “Why and how developers fork what from whom in GitHub”. In: *Empirical Software Engineering* 22.1 (2017), pp. 547–578.

[36] George H. John and Pat Langley. “Estimating Continuous Distributions in Bayesian Classifiers”. In: *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. UAI’95. Montréal, Qué, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 338–345. ISBN: 1558603859.

[37] J Peter Kincaid et al. *Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel*. Tech. rep. Naval Technical Training Command Millington TN Research Branch, 1975.

[38] Naoki Kobayakawa and Kenichi Yoshida. “How github contributing. md contributes to contributors”. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2017, pp. 694–696.

[39] Stefan Koch and Georg Schneider. “Effort, co-operation and co-ordination in an open source software project: GNOME”. In: *Information Systems Journal* 12.1 (2002), pp. 27–42.

[40] Jonathan L Krein et al. “Impact of programming language fragmentation on developer productivity: a sourceforge empirical study”. In: *International Journal of Open Source Software and Processes (IJOSSP)* 2.2 (2010), pp. 41–61.

[41] Niels Landwehr, Mark Hall, and Eibe Frank. “Logistic model trees”. In: *Machine learning* 59.1-2 (2005), pp. 161–205.

[42] Saskia Le Cessie and Johannes C Van Houwelingen. “Ridge estimators in logistic regression”. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 41.1 (1992), pp. 191–201.

[43] Stefan Lessmann et al. “Benchmarking classification models for software defect prediction: A proposed framework and novel findings”. In: *IEEE Transactions on Software Engineering* 34.4 (2008), pp. 485–496.

[44] Antonio Lima, Luca Rossi, and Mirco Musolesi. “Coding together at scale: GitHub as a collaborative social network”. In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 8. 1. 2014.

[45] Manoel Limeira de Lima Júnior et al. “Automatic assignment of integrators to pull requests: The importance of selecting appropriate attributes”. In: *Journal of Systems and Software* 144 (2018), pp. 181–196.

[46] Brian W Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975), pp. 442–451.

[47] W.H Mobley. “Intermediate linkages in the relationship between job satisfaction and employee turnover”. In: *Journal of Applied Psychology* 62.2 (1977), pp. 237–240.

[48] Audris Mockus, Roy T Fielding, and James D Herbsleb. “Two case studies of open source software development: Apache and Mozilla”. In: *ACM Transactions on Software Engineering and Methodology* 11.3 (2002), pp. 309–346.

[49] Radoslaw Nielek et al. “Choose a job you love: predicting choices of GitHub developers”. In: *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE. 2016, pp. 200–207.

[50] Octoverse. *GitHub Octoverse*. 2019. URL: `https://octoverse.github.com/2019` (visited on 12/30/2020).

[51] Gede Artha Azriadi Prana et al. “Categorizing the content of GitHub README files”. In: *Empirical Software Engineering* 24.3 (2019), pp. 1296–1327.

[52] Gregorio Robles, Gonzalez-Barahona, and M Jesus. “Contributor Turnover in Libre Software Projects”. In: *Open Source Systems*. Springer US, 2006, pp. 273–286. ISBN: 978-0-387-34226-9.

[53] Andreas Schilling, Sven Laumer, and Tim Weitzel. “Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects”. In: *2012 45th Hawaii International Conference on System Sciences*. IEEE. 2012, pp. 3446–3455.

[54] Ravi Sen, Siddhartha S Singh, and Sharad Borle. “Open source software success: Measures and analysis”. In: *Decision Support Systems* 52.2 (2012), pp. 364–372.

[55] P. N Sharma, J Hulland, and S Daniel. “Examining turnover in open source software projects using logistic hierarchical linear modeling approach”. In: *IFIP Advances in Information and Communication Technology* 378 (2012), pp. 331–337.

[56] Jyoti Sheoran et al. “Understanding" watchers" on GitHub”. In: *Proceedings of the 11th working conference on mining software repositories*. 2014, pp. 336–339.

[57] Synopsis. *Synopsis: Open Source Security and Risk Analysis*. ID: 271893. 2018. URL: http://www.sciencedirect.com/science/article/pii/S1353485818300515.

[58] Chakkrit Tantithamthavorn et al. "The impact of automated parameter optimization on defect prediction models". In: *IEEE Transactions on Software Engineering* 45.7 (2018), pp. 683–711.

[59] Jason Tsay, Laura Dabbish, and James Herbsleb. "Influence of social and technical factors for evaluating contribution in GitHub". In: *Proceedings of the 36th international conference on Software engineering*. 2014, pp. 356–366.

[60] Bogdan Vasilescu et al. "Quality and productivity outcomes relating to continuous integration in GitHub". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015, pp. 805–816.

[61] Tao Wang et al. "Who will become a long-term contributor? A prediction model based on the early phase behaviors". In: *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*. 2018, pp. 1–10.

[62] OSS website. *OSS website*. 2020. URL: https://opensource.org/faq%5C#osd (visited on 12/30/2020).

[63] Frank Wilcoxon. "Individual comparisons of grouped data by ranking methods". In: *Journal of economic entomology* 39.2 (1946), pp. 269–270.

[64] K Yamashita et al. "Magnet or sticky? Measuring project characteristics from the perspective of developer attraction and retention". In: *Journal of Information Processing* 24.2 (2016), pp. 339–348.

[65] Yunwen Ye and Kouichi Kishida. "Toward an understanding of the motivation of open source software developers". In: *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE. 2003, pp. 419–429.

[66] Yiqing Yu, Alexander Benlian, and Thomas Hess. “An empirical study of volunteer members’ perceived turnover in open source software projects”. In: *2012 45th Hawaii International Conference on System Sciences*. IEEE. 2012, pp. 3396–3405.

[67] Yue Yu et al. “Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?” In: *Information and Software Technology* 74 (2016), pp. 204–218.

[68] A. L. Zanatta et al. “Barriers Faced by Newcomers to Software-Crowdsourcing Projects”. In: *IEEE Software* 34.2 (2017), pp. 37–43.

[69] Marcelo Serrano Zanetti et al. “Categorizing bugs with social networks: A case study on four open source software communities”. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 1032–1041. DOI: 10.1109/ICSE.2013.6606653.

[70] Minghui Zhou and Audris Mockus. “Does the initial environment impact the future of developers?” In: *Proceedings of the 33rd International Conference on Software Engineering*. 2011, pp. 271–280.

[71] Minghui Zhou and Audris Mockus. “Who Will Stay in the FLOSS Community? Modeling Participant’s Initial Behavior”. In: *IEEE Transactions on Software Engineering* 41.1 (2015), pp. 82–99.

Appendix A: Appendix

A.1 Queries

This appendix provides queries and code to extract the 31 features that belong to feature set 1. Actual values for id's are used in the query, but these can be generalized to join with an id database.

The table extended-method-253503.pu.M_v3 is created with all the 888 repo_ids and corresponding attributes like user_ids, join_date, and repo_start_date.

A.1.1 r_Language

Query :

```
SELECT r.language as r_language FROM
`ghtorrent-bq.ght_2017_01_19.projects` r
WHERE r.id = 29886167;
```

A.1.2 r_Age

Query:

```
SELECT 2017 - EXTRACT(YEAR from r.created_at) as r_AGE
FROM `ghtorrent-bq.ght_2017_01_19.projects` r WHERE
r.id = 29886167;
```

A.1.3 r_Number_of_Watchers

Query:

```
SELECT COUNT(user_id) as r_Watcher_count
FROM `ghtorrent-bq.ght_2017_01_19.watchers`
WHERE repo_id = 29886167;
```

A.1.4 r_has_Dependents

Code:

```
values = []
not_processed= []
for i in data.index:
    url = data['url'][i]
    repo_id = data['repo_id'][i]
    user_id = data['user_id'][i]
    username_reponame = url[29:]
    try :
        page =
        requests.get("https://github.com/"+
        username_reponame+"/network/dependents")
        page_content = soup(page.content, 'html.parser')
        val = retrieve(page_content)
        value = [repo_id,user_id,val]
        print(value)
        values.append(value)
    except :
        not_processed.append(url)

f_record = []
```

```
for i in range(len(values)):
        record = values[i]
        rec = record[:2]
        sub_rec = record[2]
        rec.append(sub_rec[0])
        rec.append(sub_rec[1])
        f_record.append(rec)

cols=['repo_id','user_id','repo_count','package_count']
data_s = pd.DataFrame(f_record,columns=cols)
```

A.1.5 r_Number_of_Core_Members

Query:

```
SELECT repo_id_user_id, count(repo_id_user_id)
as core_members,
count(distinct(member_id)) as distinct_core_members
from extended-method-253503.pu.Project_All_Core_Members
where member_created_at < user_first_date
group by repo_id_user_id;
```

A.1.6 r_Number_of_Contributors

Query:

```
SELECT repo_id_user_id, count(author_id) as num_authors FROM
(SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
```

```
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
extract(MONTH FROM created_at) as month
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```

A.1.7 r_Min_All_Contributor_Commits

Query:

```
SELECT repo_id_user_id, min(total) as commits_min FROM
(SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
extract(MONTH FROM created_at) as month
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```

A.1.8 r_Max_All_Contributor_Commits

Query:

```
SELECT repo_id_user_id, max(total) as commits_max FROM
(SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
extract(MONTH FROM created_at) as month
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```

A.1.9 r_Min_All_Contributor_Stay

Query:

```
SELECT repo_id_user_id,  min(length) as stay_min FROM
(SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
extract(MONTH FROM created_at) as month
```

```
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```

A.1.10 r_Max_All_Contributor_Stay

Query:

```
SELECT repo_id_user_id, max(length) as stay_max
FROM (SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
extract(MONTH FROM created_at) as month
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```

A.1.11 r_has_Contribution_Invitation

Code:

```
def contribute_invitation_check(content):
```

```
    invitation_check =
    ['welcome contributions' , 'for contributing' ,
    'actively welcome pull requests']
    for s in invitation_check:
        if s in content:
            contribution_invite = 'yes'
        else :
            contribution_invite = 'no'
    return contribution_invite
```

A.1.12 r_has_Contribution_Rules

Code:

```
def contribute_rules_check(content):
    rules =
    ['how to contribute','to contribute refer',
    'for contributions',
    'rules to contribute','steps to contribute']
    for s in rules:
        if s in content:
            contribution_rules = "present"
        else :
            contribution_rules = "absent"
    return contribution_rules
```

A.1.13 r_Readability_Kincaid_Score

Code:

```
def document_readability(content):
    result = readability.getmeasures(content, lang='en')
    return result['readability␣grades']['Kincaid']
```

A.1.14 c_is_Fake

Query :

```
SELECT u.fake as Author_is_Fake FROM
‘ghtorrent-bq.ght_2017_01_19.users‘ u WHERE u.id =
(SELECT r.owner_id
FROM ‘ghtorrent-bq.ght_2017_01_19.projects‘ r
WHERE r.id = 29886167);
```

A.1.15 c_is_Deleted

Query :

```
SELECT u.deleted as Author_is_Deleted
FROM ‘ghtorrent-bq.ght_2017_01_19.users‘ u
WHERE u.id =
(SELECT r.owner_id
FROM ‘ghtorrent-bq.ght_2017_01_19.projects‘
r WHERE r.id = 29886167);
```

A.1.16 c_Age

Query:

```
SELECT EXTRACT(YEAR from r.created_at) -
EXTRACT(YEAR from u.created_at) as Author_age
```

```
FROM `ghtorrent-bq.ght_2017_01_19.users` u
INNER JOIN `ghtorrent-bq.ght_2017_01_19.projects` r
ON r.id = 29886167 AND u.id = r.owner_id ;
```

A.1.17 c_is_Core_Member

Query:

```
SELECT repo_id_user_id, count(repo_id_user_id) as count
FROM extended-method-253503.pu.Project_All_Core_Members
WHERE user_id=member_id
AND member_created_at < user_first_date
GROUP BY repo_id_user_id;
```

A.1.18 c_Own_Repos_Lang

Query:

```
SELECT COUNT(r.owner_id) as Author_own_repos
FROM `ghtorrent-bq.ght_2017_01_19.projects` r
WHERE r.owner_id =
(SELECT owner_id FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 29886167)
GROUP BY r.owner_id;
```

A.1.19 c_Watch_Repos_Lang

Query:

```
SELECT COUNT(w.repo_id) as Author_watch_repo
FROM `ghtorrent-bq.ght_2017_01_19.watchers` w
```

```
WHERE w.user_id =
(SELECT owner_id FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 29886167)
AND w.created_at >=
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 29886167)
GROUP BY w.user_id ;
```

A.1.20 c_Contribute_Repos_Lang

Query:

```
SELECT COUNT(c.committer_id)
as Author_contribute_repos_count
FROM `ghtorrent-bq.ght_2017_01_19.commits` c
WHERE c.committer_id =
(SELECT owner_id FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 29886167)
AND c.created_at >=
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 29886167);
```

A.1.21 c_History_Commits_Lang

Query:

```
SELECT COUNT(c.id) as Author_history_commit_count
```

```
FROM `ghtorrent-bq.ght_2017_01_19.commits` c
WHERE c.committer_id =
(SELECT owner_id FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
AND c.created_at <
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358) ;
```

A.1.22 c_History_Pull_Request_Lang

Query:

```
SELECT COUNT(prh.id) Author_pull_request_history_count
FROM `ghtorrent-bq.ght_2017_01_19.pull_request_history`  prh
WHERE prh.actor_id =
(SELECT owner_id FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
AND prh .created_at <
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358) ;
```

A.1.23 c_History_Issues_Lang

Query:

```
SELECT COUNT(i.id) as Author_history_issues_count
FROM `ghtorrent-bq.ght_2017_01_19.issues` i
```

```
WHERE i.reporter_id =
(SELECT owner_id
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
AND i.created_at <
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358);
```

A.1.24 c_History_Followers

Query:

```
SELECT COUNT(f.follower_id )
as Author_history_followers_count
FROM `ghtorrent-bq.ght_2017_01_19.followers` f
WHERE f.user_id =
(SELECT owner_id
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
AND f.created_at <
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358) ;
```

A.1.25 c_History_Following

Query:

```
SELECT COUNT(f.user_id ) as Author_history_following_count
```

```
FROM `ghtorrent-bq.ght_2017_01_19.followers` f
WHERE f.follower_id =
(SELECT owner_id
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
AND f.created_at <
(SELECT created_at
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358) ;
```

A.1.26 c_Number_of_Repos

Query:

```
SELECT COUNT(p.id ) as Author_number_of_repos
FROM `ghtorrent-bq.ght_2017_01_19.projects` p
WHERE p.owner_id =
(SELECT owner_id
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358);
```

A.1.27 c_Min_All_Repos_Commits

Query:

```
SELECT COUNT(c.project_id) as Author_min_all_repos_commit
FROM `ghtorrent-bq.ght_2017_01_19.commits`  c
WHERE c.author_id =
(SELECT owner_id
```

```
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
GROUP BY c.project_id
ORDER BY count(c.project_id) ASC LIMIT 1 ;
```

A.1.28 c_Max_All_Repos_Commits

Query:

```
SELECT COUNT(c.project_id)
as Author_min_all_repos_commit
FROM `ghtorrent-bq.ght_2017_01_19.commits`  c
WHERE c.author_id =
(SELECT owner_id
FROM `ghtorrent-bq.ght_2017_01_19.projects`
WHERE id = 37958358)
GROUP BY c.project_id
ORDER BY count(c.project_id) DESC LIMIT 1;
```

A.1.29 c_Min_All_Repos_Stay

Query:

```
SELECT repo_id_user_id, min(length) as stay_min FROM
(SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
```

```
extract(MONTH FROM created_at) as month
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```

A.1.30 c_Max_All_Repos_Stay

Query:

```
SELECT repo_id_user_id, max(length) as stay_max FROM
(SELECT repo_id_user_id, author_id, count(*) as total,
((Max(year * 12 + month) - MIN(year * 12 + month) + 1)/12)
as length
FROM (SELECT repo_id_user_id, author_id, sha,
extract(YEAR FROM created_at) as year,
extract(MONTH FROM created_at) as month
FROM `ghtorrent-bq.ght_2017_09_01.commits`
JOIN extended-method-253503.pu.M_v3
on project_id = repo_id
WHERE created_at < first_date)
GROUP BY repo_id_user_id, author_id)
GROUP BY repo_id_user_id ;
```