2020

Analyzing User Behavior in Collaborative Environments

Samaneh Saadat
University of Central Florida

ANALYZING USER BEHAVIOR IN COLLABORATIVE ENVIRONMENTS

by

SAMANEH SAADAT
M.S. University of Tehran, 2014
B.S. Iran University of Science and Technology, 2011

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, FL

Fall Term
2020

Major Professor: Gita Sukthankar

ABSTRACT

Discrete sequences are the building blocks for many real-world problems in domains including genomics, e-commerce, and social sciences. While there are machine learning methods to classify and cluster sequences, they fail to explain what makes groups of sequences distinguishable. Although in some cases having a black box model is sufficient, there is a need for increased explainability in research areas focused on human behaviors. For example, psychologists are less interested in having a model that predicts human behavior with high accuracy and more concerned with identifying differences between actions that lead to divergent human behavior. This dissertation presents techniques for understanding differences between classes of discrete sequences. We leveraged our developed approaches to study two online collaborative environments: GitHub, a software development platform, and Minecraft, a multiplayer online game.

The first approach measures the differences between groups of sequences by comparing k-gram representations of sequences using the silhouette score and characterizing the differences by analyzing the distance matrix of subsequences. The second approach discovers subsequences that are significantly more similar to one set of sequences vs. other sets. This approach, which is called contrast motif discovery, first finds a set of motifs for each group of sequences and then refines them to include the motifs that distinguish that group from other groups of sequences. Compared to existing methods, our technique is scalable and capable of handling long event sequences.

Our first case study is GitHub. GitHub is a social coding platform that facilitates distributed, asynchronous collaborations in open source software development. It has an open API to collect metadata about users, repositories, and the activities of users on repositories. To study the dynamics of teams on GitHub, we focused on discrete event sequences that are generated when GitHub users perform actions on this platform. Specifically, we studied the differences that automated accounts

(aka bots) make on software development processes and outcome. We trained black box supervised learning methods to classify sequences of GitHub teams and then utilized our sequence analysis techniques to measure and characterize differences between event sequences of teams with bots and teams without bots. Teams with bots have relatively distinct event sequences from teams without bots in terms of the existence and frequency of short subsequences. Moreover, teams with bots have more novel and less repetitive sequences compared to teams with no bots. In addition, we discovered contrast motifs for human-bot and human-only teams. Our analysis of contrast motifs shows that in human-bot teams, discussions are scattered throughout other activities while in human-only teams discussions tend to cluster together.

For our second case study, we applied our sequence mining approaches to analyze player behavior in Minecraft, a multiplayer online game that supports many forms of player collaboration. As a sandbox game, it provides players with a large amount of flexibility in deciding how to complete tasks; this lack of goal-orientation makes the problem of analyzing Minecraft event sequences more challenging than event sequences from more structured games. Using our approaches, we were able to measure and characterize differences between low-level sequences of high-level actions and despite variability in how different players accomplished the same tasks, we discovered contrast motifs for many player actions. Finally, we explored how the level of player collaboration affects the contrast motifs.

To my amazing mom, Maryam,

and my lovely husband, Ramin.

ACKNOWLEDGMENTS

I would like to express my appreciation to everyone who supported me during my PhD, some of whom are acknowledged below.

First and foremost, I am extremely grateful to my advisor, Dr. Gita Sukthankar, whose insight and knowledge steered me through this research. She has been an amazing mentor and inspiring role model to me. I have always benefited from her valuable advice. I deeply appreciate the freedom she gave me to try my ideas, make mistakes and learn.

I would like to extend my sincere thanks to my dissertation committee members Dr. Steve Fiore for his stimulating questions and for inspiring me to think outside of the box, Dr. Lotzi Bölöni for his thoughtful feedback on my research, and Dr. Ivan Garibay for his consistent support.

I also had the great pleasure of working with two amazing research project teams at UCF: Social-Sim and ASIST. I have learned a lot working alongside smart and accomplished researchers in these teams. I would like to thank them all, in particular, Olivia Newton and Chathika Gunaratne, whom I collaborated with on several research papers; and Brandon Barnes, whose help cannot be overestimated.

I cannot begin to express my thanks to my mother, who is a strong woman and has raised me that way. I feel so lucky to be her daughter. She has always believed in me and has made me to believe in myself. My accomplishments would not have been possible without her support and sacrifices.

Last but not least, special thanks to my best friend and husband, Ramin, for his love, friendship, and encouragement. He has always supported me even when he disagreed with my decisions. I love you, Ramin.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Sequences commonly occur in numerous applications of the physical, biological, and computer sciences. Hence, myriad sequence mining methods have been developed to analyze various types of sequences. For example, in molecular biology, the sequencing of nucleotides in a strand of DNA provides a massive amount of sequence data that has motivated scientists to develop methods to comprehend various aspects of the sequences including sequence assembly, motif finding, and other applications [99, 75, 14]. With the growth of online platforms, discrete event sequences have become even more abundant in the digital world: from user clicks in an online store to actions of users on social media platforms. Developing novel sequence mining approaches that help researchers comprehend the ever-growing body of digital data is imperative.

This dissertation introduces a set of techniques to detect differences between groups of long discrete sequences efficiently. Then we demonstrate how these techniques can be used to analyze the behaviors of users in online collaborative environments. We aim to answer the following research questions:

- What is the difference between software engineering teams that use automated accounts (aka bots) and teams that don't?
- What differences in low-level event sequences create various high-level actions in the Minecraft world?
- What is the difference between highly collaborative players and hardly collaborative players in Minecraft?

In this chapter, first, we introduce sequence mining approaches we developed to analyze groups

of sequences. Then, we present two domains that we applied our sequence mining approaches to analyze event sequences: GitHub and Minecraft.

Sequence Mining

There are many machine learning methods to classify and cluster sequences. However, in some cases, it is hard to explain why those algorithms select a specific data partition—what are the differences in groups of sequences that make them distinguishable? Although in some cases having a black box model is sufficient, there is a need for increased explainability in various research areas such as social sciences. For example, psychologists are less concerned with having a model that accurately predicts human behavior and more eager to identify differences between actions that lead to divergent human behavior. This dissertation presents techniques for understanding differences between groups of discrete sequences. Approaches introduced in this dissertation can be utilized to interpret black box machine learning models on sequences. In this section, we introduce two sequence mining approaches we developed to improve our understanding of users by studying their sequences.

Sequence Groups Comparison

Analyzing event distributions alone is insufficient to reveal the subtle differences in groups of sequences. Our first sequence mining approach is a new analytic framework for characterizing differences between groups of event sequences. Our aim is to be able to answer the following questions:

- How distinct are the sequences of different groups?

- What are the differences between these sets of sequences?

To answer the questions discussed above, we introduce the following approach. First, we compared groups of sequences by converting the sequences to k-gram representation vectors and measuring the silhouette score of the groups. Second, to understand differences between sequences, we created distance matrix between subsequences for all sequences and compared statistics and aggregated matrix profiles of the groups.

As a case study, we trained black box supervised learning methods to classify sequences of GitHub teams and then utilized our sequence analysis techniques to measure and characterize differences between event sequences of teams with bots and teams without bots. In our second case study, we classified Minecraft event sequences to infer their high-level actions and analyzed differences between low-level event sequences of actions.

Although we applied our approach to analyze GitHub and Minecraft event sequences, this approach can be applied to many other sequence mining tasks relevant to socio-technical systems:

- Do click sequences of users who purchase products in an online store differ from those who leave without buying?
- How do event logs of authorized users differ from hackers launching cyberattacks?
- Do the event sequences of GitHub repositories managed by highly productive teams differ from dysfunctional teams?

Contrast Motif Discovery

We introduce a method for analyzing event sequences by detecting contrasting motifs. Our approach is called contrast motif discovery. The aim is to discover subsequences that are significantly

more similar to one set of sequences vs. other sets. Compared to existing methods, our technique is scalable and capable of handling long event sequences.

Motif

Motifs are fairly short subsequences shared between multiple sequences; unlike sequential patterns, motifs are contiguous [14]. Motif is a term borrowed from biological sciences where mutations might occur but in many cases, those mutations do not affect the functions of the genomic sequence.

Contrast Mining

In classification, the goal is to guess the category of an object or data point based on its attribute, as opposed to contrast mining which takes the category of data points and reverse engineers the attributes that mark the data point as a member of a category. Contrast mining attempts to detect meaningful differences between groups of objects. For example, given the attributes of categories of online banking customers, contrast mining would identify dissimilar features between fraudulent and normal users. Discovering contrasts between specific groups of interest is particularly valuable in social science research [7].

Existing contrast mining approaches discover contrasting sequential patterns which are costly to find for large datasets or long sequences. To the best of our knowledge, we are the first to propose a method for discovering *contrast motifs*. Motifs are less general than the patterns extracted from SPM and hence computationally faster to find. We exploit conceptual ideas from motif finding techniques that were designed for time series data and apply them to sequence data.

Given multiple groups of sequences, our algorithm, first, finds a set of candidate motifs for each

group. Then, for each set of candidate motifs, the algorithm selects the subset of motifs that are significantly closer to their own group compared to other groups. These refined motifs are designated as the contrast motifs of the group.

We applied this algorithm to GitHub and Minecraft sequences to achieve a deeper understanding of dynamics of user behaviors in these to environments. To promote shared progress, we have made our code and data publicly available [1].

GitHub

GitHub[2] is a social coding platform for distributed, asynchronous collaborations in open source software (OSS) development. GitHub version control and issue tracking features make it attractive to developers. GitHub software repositories typically have several developers working on the project as a virtual team. Developers make changes to the repository by pushing their content. To be able to push to a repository, users should have push privileges. This push-based development model is mainly used for smaller teams. Large open-source projects use a pull-based development model in which any GitHub user can contribute to a repository by forking (i.e.copying) the repository, making modifications and submitting their contributions by sending a pull request to the original repository. Repository maintainers review pull requests, discuss possible modifications in the comments, and decide whether to accept or reject the requests. GitHub users can report bugs or request new features in the issues section of repositories. Other users can comment on the issues to discuss problem solving approaches or to coordinate on who will work on the issue and how. Users can star a repository to bookmark it or to show their interest in a repository. GitHub also allows users to follow repositories of their interest to get the latest update about them. Users can

[1] https://github.com/SamanehSaadat/ContrastMotifDiscovery

[2] https://github.com/

follow each other on GitHub and get informed about activities of developers they like. Figure 1.1 shows the main page of a repository called scikit-learn with its available functionalities.

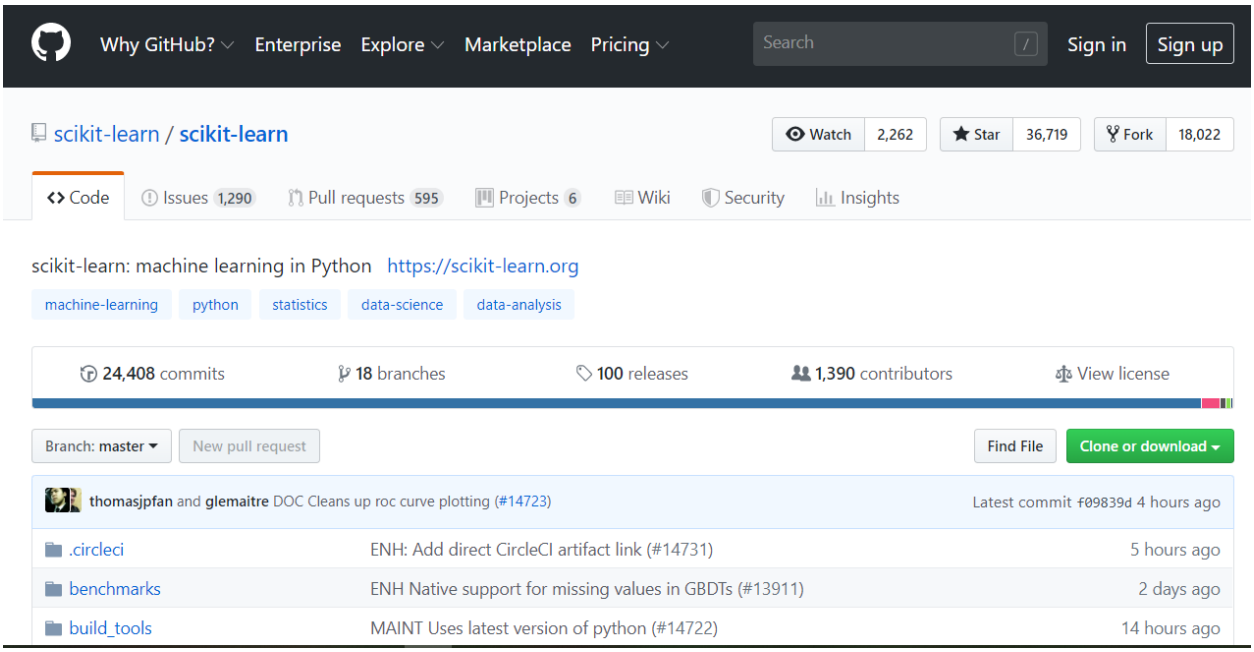


Figure 1.1: An example of a GitHub repository.

GitHub Teams

GitHub is a platform that facilitates software development in virtual teams. In today's connected world virtual teams that collaborate on online platform are becoming more commonplace. Many large technology companies such as Facebook and Google utilize the power of virtual teams and develop open source products on GitHub. These companies are turning to crowdsourcing to solve difficult problems as these online collaboration platforms help them to find the best subject matter experts [66]. For example, Google open sourced Tensorflow[3], a machine learning framework, in 2015 and has thousands of developers contributing to this project on GitHub working remotely as

[3]https://github.com/tensorflow/tensorflow

a virtual team. Hence, studying virtual teams and what affects their performance has an important role in the future of work.

When tasks are too large or too complex for a single individual, teams are useful [18]. However, teamwork is hard because it requires coordination among team members. Challenges of working in teams inspired many organizational researchers to study teams. Studies show that increasing the team size adds to the coordination overhead as larger teams have more possible links among team members [18]. In addition, geographical dispersion of team members also increases coordination complexity since it makes it more challenging for team members to become familiar with each others' skills and expertise and develop tasks [31]. Open source software development teams on GitHub are faced with both of these types of coordination overhead as open projects are virtual teams that are capable of benefiting from a large number of volatile developers with irregular and short-lived participation in the project.

GitHub has been interesting to both social scientists and software engineering researchers since it has an open API to collect metadata about users, repositories, and activities of users on repositories. This API allows researchers to have access to a large dataset of software engineering activities and communications.

Software Bots

Improving the productivity and workflow of software development teams is a key concern of tech companies hence research in this area is gaining traction in recent years [54]. Providing better and smarter tools to help developers simplify and automate tasks is one approach to satisfy that need. As a consequence, software development bots are gaining traction among software engineering teams. In their basic form, bots are interfaces that connect users and services [45] and, in their more advanced form, bots may exploit data analysis or AI techniques. Bots support different soft-

ware development tasks including automating routine tasks (e.g. merging changes across different branches), helping developers stay in the flow (e.g. deferring interuptions until an appropriate time), perform redundant tasks (e.g. answering user questions), and improving decision making (e.g. collecting, analyzing and sharing data relevant to decisions).

Bots are becoming more commonplace in modern teams but little is known about the impact they have in terms of the benefits and disadvantages they bring to team. Our goal in this research is to shed light on the impact of bots on software development processes and outcomes.

GitHub Team Outcome and Processes

First we studied the impact of GitHub bots on the outcome of teams. We investigated the differences in productivity, issue support responsiveness and work centralization of teams with and without bots. Result shows that teams with bots are more productive and the work in these teams are more centralized. Additionally, human-bot teams on GitHub document and track more issues which results in higher issue resolution times.

Second, to study team dynamics, we took a sequence mining approach because sequences contain temporal information about actions taken. We created event sequences of GitHub teams with bots (aka human-bot teams) and teams without bots (aka human-only teams).

We built classification models to predict whether a GitHub team using bots based on their sequences. All models, SVM, LSTM, and LSTM+CNN, were able to distinguish whether an event sequence was generated by a human only or human-bot team, with the neural network models achieving the best F1 score, 0.79, which was significantly better than the baseline. These results validate our hypothesis that the event sequences can be leveraged to detect teamwork differences.

Comparing silhouette score k-gram representation of sequences of teams showed that sequences

of human teams are distinct from sequences of human-bot teams. Comparing aggregated matrix profiles of teams indicated that human-bot teams have more novel event sequences. In contrast, human teams have more repetitive sequences, especially at the early stages of their work.

Finally, we applied a contrast motif discovery method on this data to understand what are the sub-sequences that happen in one type of team at a higher rate compared to the other type of team. This experiment revealed that in human-bot teams, discussions are scattered throughout other works the teams do while in human-only teams the discussions tend to cluster together.

Minecraft

Minecraft is a multiplayer online game where players can explore a 3D world, mine materials, and craft tools and structures. It is a sandbox environment where players are afforded a great deal of freedom in how they interact with the game world. Most Minecraft servers are maintained by players rather than by private companies, making it an ideal laboratory for studying player behaviors and social interactions. Minecraft offers many opportunities for collaboration, including joint crafting and combat. Prior research indicates that players prefer to cooperate with players who have similar action preferences in terms of building, mining, fighting and exploring [57].

Although Minecraft was not explicitly developed for research purposes, it has been used in many learning studies and scientific experiments [60]. It is an ideal laboratory to study collaboration as the game can be modified to become more collaborative, track player activities, and manipulate team compositions [15, 57].

Intelligent agents can be developed in Minecraft which makes Minecraft an excellent platform for studying human-agent teaming. For example, CraftAssist is an implementation of an interactive bot assistant in Minecraft [25].

Game events form sequences that provide valuable information about the play style and high level goals of the players. The observable events are low-level: *move*, *place block*, *consume item*, etc. High-level actions in the game world, such as exploring, mining, fighting, or building, are accomplished by performing chains of low-level actions. Since events are logged multiple times per second, the sequence of low-level game events may be long and filled with superfluous detail. Prior research attempted to classify these low-level event sequences to high-level actions [58].

Our aim is to develop an unsupervised method for detecting common subsequences across different instances of event sequences related to a group, allowing long sequences to be generalized into a few short subsequences. To that end, we developed a contrast mining approach that discovers subsequences that differentiate groups of sequences. We aim to not only discover motifs of a group of sequences but also to refine the motifs to ensure that they represent the differences between classes of sequences.

Minecraft Processes

Understanding event sequences is an important aspect of game analytics, since it is relevant to many player modeling questions. Our second case study analyzes Minecraft action and player sequences.

As a sandbox game, Minecraft provides players with a large amount of flexibility in deciding how to complete tasks; this lack of goal-orientation makes the problem of analyzing Minecraft event sequences more challenging than event sequences from more structured games. Using our approach, we were able to discover contrast motifs for many player actions, despite variability in how different players accomplished the same tasks. Furthermore, we explored how the level of player collaboration affects the contrast motifs. Although this research focuses on applications within Minecraft, our tool, which we have made publicly available along with our dataset, can be

used on any set of game event sequences.

We classified Minecraft event sequences to infer their high-level actions and analyzed differences between low-level event sequences of actions.

Most prior studies in games use sequential pattern mining approaches, which are designed to find frequent exact non-contiguous subsequences. The traditional algorithms for mining sequential patterns [2, 61] are appropriate for short sequences such as supermarket transactions. Consequently, traditional algorithms are ineffective for mining long sequences. Some of the traditional methods are used to process long sequences, but they require extensive run time [49, 61].

In addition, in situations where there is no clear boundaries for sequences (such as the Minecraft dataset used in this dissertation), sequential pattern mining approaches that allow gaps between events might blend two or more actions to create a pattern. One of the advantages of using motifs is that motifs form a continuous subsequence and hence blending is less likely.

This research describes the usage of our contrast motif discovery algorithm to analyze Minecraft event sequences. We created event sequences for different Minecraft actions (fight, mine, explore, and build) and extracted motifs that differentiate actions from each other. Certain actions, such as fighting, yielded more contrast motifs, representing variations in player style. On the other hand, the explore action did not have any motifs that were distinctive to that class.

Moreover, players can be categorized into different groups based on style and characteristics such as collaborative vs. non-collaborative, expert vs. novice, and effective vs. ineffective. Our contrast mining approach can help us achieve a better understanding of the differences between various groups of players. We compared the behavior of *highly* collaborative and *hardly* collaborative players by examining their event sequences. Our investigation revealed that fighting is a common behavior amongst highly collaborative players while there is no behavior shared between less

collaborative players.

Publications

This document is based on peer reviewed publications by the author of this dissertation in collaboration with other researchers. Chapter 3 discusses our sequence mining methods to compare and contrast classes of discrete sequences, which are published in *AIIDE'20* and *WI-IAT'20* conferences [72, 73]. In chapter 4, two of our papers that are published in *SBP-BRiMS'18* and *WI-IAT'20*, are presented which are focused on studying GitHub users and teams [70, 71]. Chapter 5 and 6 includes the applications of our sequence mining methods in GitHub and Minecraft [72, 73].

CHAPTER 2: RELATED WORK

In this chapter, first, we present existing research on mining discrete sequences and discuss their shortcomings. Then we review prior research on GitHub and Minecraft, which are the application domains for our proposed techniques.

Sequence Mining

A sequence is an ordered list of events, where events can be represented by symbols from a specific alphabet. Pattern mining in sequences has countless applications in academia and industry including biological, purchasing, and weblog pattern mining [48]. Because of this, numerous methods have been designed to extract patterns in sequential data, including traditional sequential pattern mining [2, 61], maximal sequential patterns [51], and closed sequential patterns [87].

Explainability

Many of the most accurate machine learning models are constructed as black boxes, meaning that their internal logic is hidden from their users [27]. In the scientific community, there is an increasing interest in explaining decisions made by black box models. For example, Guidotti et al. [26] presented an approach for explaining black box decisions of an image classification model. Despite the abundance of discrete sequences, explaining machine learning models built for sequences has not been well-studied. In this dissertation, we developed glass-box approaches to compare and contrast groups of sequences and explain differences in these groups.

Sequential Pattern Mining

One of the conventional approaches for finding key sequences is sequential pattern mining [23], and researchers have proposed many specific techniques for discovering discriminative patterns that occur at significantly different frequencies across two groups of sequences [30, 29, 16]. One of the major drawbacks of sequential pattern mining approaches is that they either 1) find many patterns if the minimum support threshold is low or 2) generate either none or very short patterns if the minimum support threshold is set high. This makes large-scale sequence mining challenging for sequences for which we do not have prior knowledge. Moreover, interpreting discovered patterns in sequential pattern mining requires subject-matter expertise. Our sequence groups comparison approach summarizes sequences and their differences into a few numbers, reducing the need for domain knowledge to interpret the data. Yet another major drawback of sequential pattern mining approaches is that they are designed for short sequences and are ineffective in processing datasets containing long sequences.

Motif Finding

There are numerous motif discovery approaches that have been developed for time series or biological sequences [56, 5, 4]. Biological methods for finding motifs enforce constraints on the data to ensure that the discovered motifs are scientifically plausible [14]. For example, methods for finding transcription factor binding site motifs extract one and only one motif for each sequence and use relatively short input sequences [98]. These approaches may not be well-suited for user behavior sequences that are long, may contain multiple motifs, and are unconstrained.

Matrix Profile

Since discrete sequences are the categorical analog of time series data, motif discovery approaches designed for time series can be applied to discrete sequences with minor modifications. The matrix profile approach can be utilized to discover motifs of time series. The matrix profile is a vector calculated between two time series (similarity join) or one time series and itself (similarity self-join); for each subsequence of the first time series, the distance is stored to its closest subsequence in the second time series. The distance between two subsequences is their Euclidean distance. Matrix profiles have many applications in time series analysis including motif discovery, discord/anomaly detection, and semantic segmentation [96, 97, 95, 81]. In our work, we created the matrix profile for discrete sequences and use it to summarize differences between groups of sequences.

Yeh et al. [96] extracted matrix profiles for discrete sequences but they convert the sequences to time series first using a method proposed by Rakthanmanon et al.[65]. This method converts sequences to time series by assigning an ordinal number to each symbol in the sequence; then Euclidean distance is used for measuring the distance between time series. This ordinal encoding assumes an ordinal relationship between symbols which may not exist. We developed an approach for extracting the matrix profile directly from discrete sequences. Instead of Euclidean distance, Hamming and LCS distance are employed; these distance measures are designed for sequences and do not require encoding of the input.

Contrast Mining

Understanding the differences between contrasting groups of sequences is an essential task in data mining and has numerous applications including customer behavior analysis and medical diagnosis [6, 93]. Prior research on learning the contrast patterns across groups is primarily focused

on finding contrast sequential patterns. A contrast sequential pattern is a pattern that occurs frequently in one sequence group but not in the others [93]. Contrast sequential pattern mining has the same computational limitations as sequential pattern mining approaches discussed earlier in this dissertation. To overcome this problem, we sought to unify the computational strengths of motif discovery with statistical testing techniques to identify contrast motifs.

GitHub

User Types

Several studies of GitHub have attempted to identify types and roles of users. Badashian et al. [3] detected influential GitHub users and manually identified the role of the top thirty influential users by analyzing their profiles and personal webpages.

Wagstrom et al. [86] focused on a few successful communities for which they could obtain knowledge about the underlying socio-technical practices. The authors manually defined user categories and used a top-down approach based on some heuristics to detect user roles. They divided the user roles in GitHub into two categories: Development Maturity roles and Specialized roles. Development Maturity roles track the progress of an individual through their participation in a project as they move from an interested lurker to a core project member. This role group includes six stages:

1. Lurker: individuals who only monitor activities or issues of a project.
2. Issue: individuals who have been active on the project issue tracker, either by filing new issues or commenting on existing issues or pull requests.
3. Independent: individuals who have created a fork and have worked on it privately.

4. Aspiring: individuals who have submitted pull requests which have not been accepted yet.
5. External contributor: individuals who contribute to the project via pull requests.
6. Internal collaborator: individuals who are a part of the organization and have critical responsibilities such as accepting pull requests.

Specialized roles include roles that a contributor can take depending on their commitment and interest.

1. Prodder: individuals who identify and take on longstanding issues or issues that have idle.
2. Project Stewards: individuals who focus on managing the project. They merge pull requests (from external contributors) into the project, comment on the pull requests, and close a pull request once it has been merged.
3. Code Warriors: individuals who frequently and consistently commit code to a project.
4. Nomad Coders: individuals who have contributed only minor code changes and then move to a new project or individuals who are participating in one project, but make minor contributions to another project.
5. Project Rockstars: individuals who have a high visibility and are significant contributors to a project.

Although the research has been conducted on a small set of projects, it shows the diverse roles of developers on GitHub.

In other studies, Joblin and colleagues [37, 38] used a network-based approach to classify developers into core and peripheral developers. Their approach requires version control tracks in order to reconstruct the developers' network and detect their roles.

In our research, we developed a machine learning based method to discover GitHub users' roles. We have an automatic approach that does not require hand-labeling users or manual investigation of available personal data. Our approach is bottom-up and roles are discovered organically from clustering results. We studied a large data set which contains all GitHub repositories and users that have been active in 32 months. Since our ultimate goal was to simulate the entire GitHub ecosystem, we attempted to identify user roles in all repositories, which are not limited to software development and can include other activities such as curating content, writing books, or doing school projects. Finally, our approach is independent of the developed code and role detection is not dependent on the version control information.

Software Bots

Improving the productivity of developers is a key concern for software engineers. It is expected that providing developers with better tools that automate the software development process helps them to to work more efficiently as a team and to solve larger and more complex problems.

Bots have been developed to support different software development activities including automation of repetitive tasks to reduce workload of software development tasks (e.g. code coverage bots that automatically report what percentage of code is tested after adding new codes), and bridging knowledge and communication gaps in software teams (e.g. mention bots that automatically find the best developers to review a pull request based on working history of developers).

In an effort to support the study of software bots, researchers have identified different classes of bots in software development [76]. There are bots that provide taskwork support for the teams by completing chores that previously would have been completed by human developers. Code bots support code-related activities to make them more efficient. Test bots allow developers to offload the repetitive task of evaluating code. User support bots can communicate with users and provide

answers to frequently asked questions. A few examples of these bots are as follow:

- Hubot[1] automatically updates task items on Trello when code is committed on GitHub to offload the memory overhead of developers.

- BugBot is a Slack[2] app for working with GitHub issues. It allows developers to add and access GitHub issues with no context-switching between Slack and GitHub. This helps developers to track issues and gain awareness about existing issues without being interrupted which improves their productivity by allowing them to stay in the *flow*. Researchers that surveyed software engineers have found that staying in the *flow* without many context-switches is one of the most important factors for developers to have a productive day [54].

- Travis CI[3] is a Continuous Integration service that activates whenever new commits are pushed to that repository or a pull request is submitted. Travis CI automatically builds the software and run tests. When this process is completed, Travis notifies the developers. Figure 2.1 shows the pull requests report of Travis CI in a repository.

- CodeCov[4] is a code coverage tool. Code coverage is a measurement used to express the percentage of lines of code that were executed by an automated test suite. A program with high test coverage has had more of its source code executed during testing and thus has a lower chance of containing undetected software bugs compared to a program with low test coverage. The CodeCov bot can improve code review workflow and quality. Figure 2.2 shows CodeCov.

[1] https://hubot.github.com/

[2] https://slack.com/

[3] https://github.com/marketplace/travis-ci

[4] https://codecov.io/

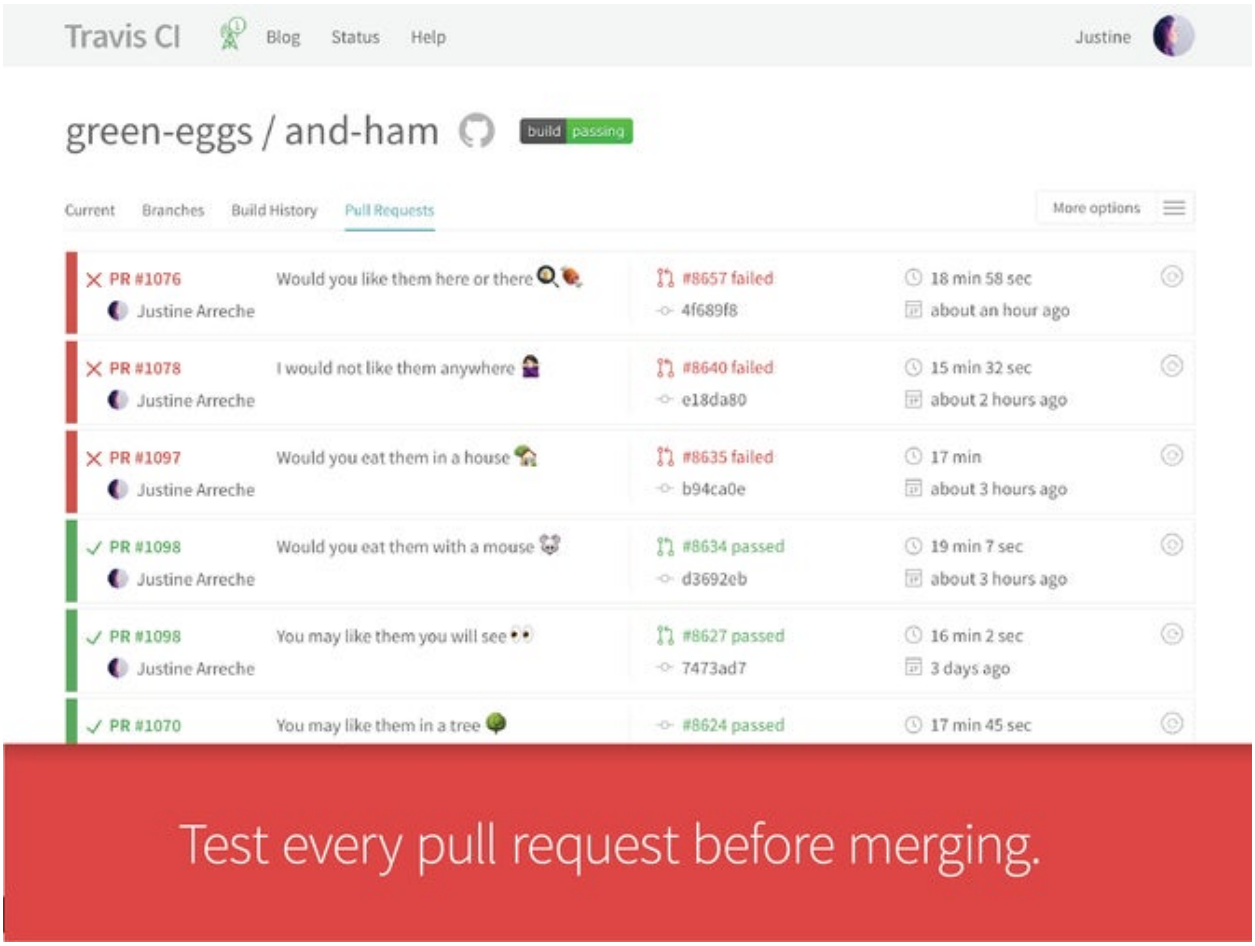


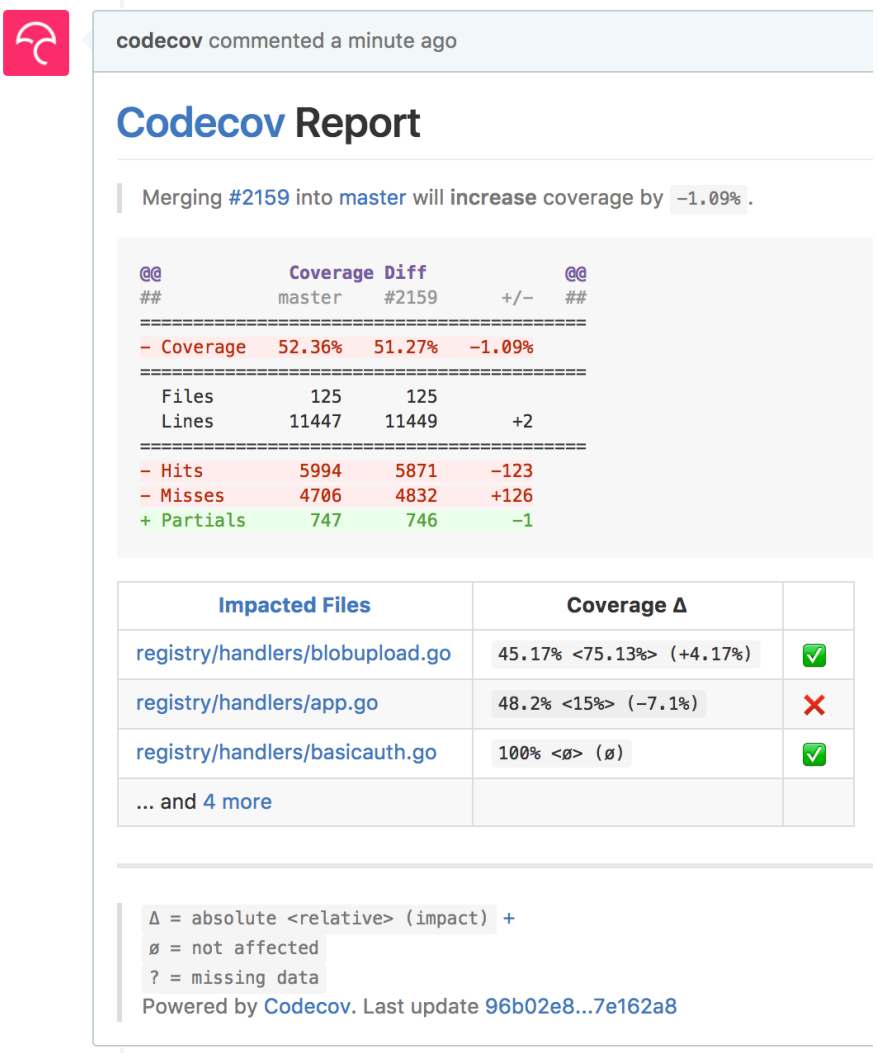
Figure 2.1: Travis CI pull requests report

Image credit https://github.com/marketplace/travis-ci

Hukal et al. [34] studied bots involved in coordinating work in one of the GitHub repositories called Kubernetes[5], which is a system for automating the deployment, scaling, and management of containerized applications[6]. They identified four different classes of bots (listed from simplest to more complex): broker, checker, gatekeeper, and manager bots. This categorization is based on different characteristics of bots such as the level of autonomy, criticality of their task, and the

[5]https://kubernetes.io/

[6]Application containerization is an OS-level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each app.



codecov commented a minute ago

Codecov Report

Merging #2159 into master will increase coverage by -1.09%.

```
@@            Coverage Diff             @@
##           master    #2159      +/-   ##
==========================================
- Coverage   52.36%   51.27%   -1.09%
==========================================
  Files         125      125
  Lines       11447    11449       +2
==========================================
- Hits         5994     5871     -123
- Misses       4706     4832     +126
+ Partials      747      746       -1
```

Impacted Files	Coverage Δ	
registry/handlers/blobupload.go	45.17% <75.13%> (+4.17%)	✅
registry/handlers/app.go	48.2% <15%> (-7.1%)	❌
registry/handlers/basicauth.go	100% <ø> (ø)	✅
... and 4 more		

Δ = absolute <relative> (impact) +
ø = not affected
? = missing data
Powered by Codecov. Last update 96b02e8...7e162a8

Figure 2.2: Codecov

Image credit https://docs.codecov.io/docs/pull-request-comments

impact of their task on the workflow. Brokers scan and repost information, following a simple if-this-then-that procedure. They automate non-critical tasks that requires little or no human involvement. Checkers evaluate the information and notify the most relevant human members to take actions. Gatekeepers confirm that the submitted work satisfies the pre-defined requirements and can proceed to the next step. For example, gatekeeper bots check if developers signed the Contributor Licensing Agreement (CLA) and redirect them to the compliance page if not. Manager

bots perform interactive coordination and supervision. In general, this study illustrates the role of bots in open source software development; bots facilitate maintenance and the reinforcement of order in the project. They extend ability of developers to coordinate and take over mundane tasks to free up human resources to perform more complex work. The importance of bots increases in OSS projects as the project grows in size. The authors argued that bots help to enforce procedural rules to implement predefined workflows and increase the reliability and stability of the project.

In addition to their role in improving individual productivity, bots are known for their support in team cognition. For example, *Situational awareness*, which is known to help teams to be more effective [63], can be enhanced by deployment of bots. Software bots enhance team communication as well by initiating necessary conversations or reducing the amount of communication by automating tasks [77]. Although bots are created with the purpose of enhancing individual and team productivity, they may have negative impact on user experiences [50]. Thus, it is indispensable to study the changes that bots create [89, 55]. In recent years, there has been a growing body of research studying GitHub bots. While many studies focus on the outcome of the project when studying GitHub bots [88, 89], others reviewed the experience of developers in such environments [50]. While myriad of GitHub bots are adopted by Open Source Software (OSS) community, relatively little has been done to study how they impact experience of human members. In this research, we intend to shed light on the experience of team members in human-bot teams.

Many human-bot interaction studies mainly consider surveys as the source of their input [50, 88]. While surveys are valuable source of information to get us into the head of developers, it is challenging to survey thousands of developers. As a consequence, most of these research papers studied a small fraction of active GitHub repositories. In this paper, with the use of archival data of GitHub activities, we studied thousands of GitHub repositories with and without bots.

In our research, we aim to create new techniques that enable us to study the performance and

cognitive processes of GitHub teams from analyzing their event data.

A study on a sample of popular repositories before and after bot adoption found no significant difference in various metrics such as time to pull request before and after bot adoption [88]. However, a more recent study found that adoption of code review bots increases the number of merged pull request and decreases communication among developers [89]. In addition to effectiveness of bots in improvement of the task work, prior studies examined the social interactions of bots with humans. Liu et al. found that *Stale* bot, which helps maintainer to triage abandoned issues and pull request by marking them based on the period of inactivity, can create negative experience for contributors [90, 50]. While bots are developed with the purpose of improving the user experience, looking at prior research demonstrate that how crucial it is to study impact of bots on individuals as well as team processes.

There is a growing body of research studying human-bot interaction and providing suggestion on the best practices for development of effective software bots [77, 11, 88]. The suggestions include improved social interactions, better management of the workflow, and increase the awareness of the developers regarding useful tools. Erlenhov et al. goes one step further and describe an idea software bot as ”an artificial software developer which is autonomous, adaptive, and has technical as well as social competence” [17].

Team Sequences

Team researchers have recognized the dynamic that teams and their tasks have. Analyzing temporal sequences of teams can be used to provide novel insights about these patterns, and sequence mining methods can be utilized to to study research questions concerning the dynamic nature of teamwork and teams [32]. The advantage of sequence-based methods is that they do not isolate a single event, but instead examine the events ”in their continuity’ [32].

In the field of organization science, [8] has exploited sequence mining methods to explain differences in performance of organizations by examining sequence patterns of the learning processes. By analyzing sequential changes in work processes of an organization over a period of time, [62] discovered that variability was negatively related to the performance.

A number theories on team development include sequence-based concepts, and sequence mining methods can be developed to evaluate these hypotheses. For instance, there is a body of work on group development that studies the progress of teams through stages; patterns of these sequences are related to the future operation and performance of the team [82, 91].

Herndon et al. state that sequence mining approaches have myriad applications in team research [32]. The authors argue that *optimal matching* approaches can be used to compare team process sequences for the purpose of examining the impact of weak or strong ties on team processes. *Optimal matching* algorithms measure the distance between two sequences by calculating the cost of transforming one sequence into the other. Although optimal matching techniques have widespread use in sociology, their main drawback is that the cost function, which is defined by the researchers, can influence the outcome [67].

There are studies that were not able to justify some observed behavior of the subjects based on the mere condition of subjects. For instance, in [47], which studied the impact of Transactive Memory System (TMS) on team performance, the authors hypothesize that enabled TMS in team will translate into higher performance. Although they observed higher performance for teams with TMS compared to teams without TMS, they could not justify the highest performance of teams with TMS disabled first and then enabled. [32] argues that the difference in the performance of these teams is probably caused by their team processes rather than their mere condition and suggests that sequence methods can be used to further analyze the differences.

These studies manually coded states of teams and their activities in a pre-defined time interval (e.g.

1 minute). However, with GitHub, a large corpus of team activities is accessible to us effortlessly.

Minecraft

[58] collected Minecraft data and studied players' actions using the frequencies of low-level events. Additionally, they constructed a classifier that predicts the high-level action from the low-level event log. To study collaboration in Minecraft, [57] defined various types of collaboration graphs such as contact, chat, and build graphs. They introduced the collaboration index as a universal metric to assess and compare the collaborativeness of players. Their study also identified predictors of collaboration in this game, including player familiarity and similarity. This research leverages these collaboration metrics and data structures to conduct a study of how collaboration affects contrast motifs.

Game Sequences

Analyzing player behaviors can be beneficial for myriad purposes including improving user experience, supporting administrative tasks, and assisting social science studies [58]. Sequence mining techniques have many potential applications in game analytics as they allow researchers to investigate patterns of player behavior [52].

Event sequences are a valuable data type for game analytics as they provide not only the frequency of events but also the temporal order in which those events occurred. The most popular unsupervised approach for analyzing player action sequences is Sequential Pattern Mining (SPM) [52, 42, 46]. However, these techniques are memory- and time-consuming for large datasets or long sequences [74].

A combination of frequent sequence mining and clustering can be used to visualize common subsequences of player actions [42]. [40] created sequences of user keyboard input and mouse movement and extracted the repetitive patterns within games using Lempel–Ziv–Welch (LZW) compression-based algorithms. [46] applied sequential pattern mining in *Starcraft: Brood War* at both the micro and macro level to discover short-term and long-term patterns of player behavior.

In this research, we present a more efficient approach for finding common subsequences of a sequence group that can handle long sequences with minimal memory consumption and reasonable execution time. This research demonstrates the usage of our technique to analyze Minecraft player action sequences.

CHAPTER 3: SEQUENCE MINING

Sequence Group Comparison

This section describes our two part approach to analyzing event sequences: 1) distinction measurement, and 2) difference detection. The code for our analytic pipeline is publicly available at [79].

Sequence Distinction Measurement

Given two groups of sequences, our goal in this section is to measure distinction of these two groups from each other (i.e. whether they belong to two different clusters) and understand why they are distinct. Although we explain our analysis procedure for two groups, it is easily generalizable to more than two groups of sequences.

Sequence to Vector

Our first step in making sequences comparable is converting them to equal length vectors. For vectorization, we extract a k-gram representation of sequences by moving an overlapping window with fixed size of w along the sequence to generate $n = l - w + 1$ subsequences where l is the length of the sequence. This transformation converts each sequence to a set of ordered subsequences of length w. To be able to compare a k-gram representation of sequences with each other, we convert them to vectors with equal lengths using the *Term Frequency-Inverse Sequence Frequency* (*TF-ISF*) model. *TF-ISF* is analogous to the *TF-IDF* procedure that is used for vectorizing textual documents (i.e. an ordered list of words) [53]. Term frequency (*TF*) measures the

frequency of every subsequence in a sequence. Higher frequency subsequences tend to contribute noise to the similarity computation [1]. One way to avoid this noise is by lowering the weighting subsequences with higher frequency using inverse sequence frequency (*ISF*). If there is a subsequence that is shared between most of the sequences, this subsequence may be less important in understanding differences between groups of sequences. The inverse sequence frequency ISF_i of the *i-th* subsequence is calculated using Equation 3.1.

$$\text{ISF}_i = \log(N/N_i) \tag{3.1}$$

where N is the total number of sequences and N_i is the number of sequences that contain the *i-th* subsequence. Note that ISF_i is a decreasing function of the number of sequences in which it occurs.

To summarize, in order to create the vector of sequences using *TF-ISF* model, we first measure the frequency of each subsequence i (TF_i) and then multiply them by ISF_i. A subsequence has a high *TF-ISF* for a sequence if it appears many times in that sequence and does not appear in many other sequences [64].

In addition to the *TF-ISF* model for vectorization, we have a simple binary vectorization method. In this model, each vector has zero and one values where one indicates the existence of a subsequence in the sequence and zero otherwise (i.e. frequencies of subsequences are ignored). Comparing this model with the *TF-ISF* model is helpful in understanding whether the difference between two groups of sequences is solely due to difference in the frequencies or whether the subsequences also differ.

Sequence Comparison

To compare two groups of sequences, we need a distance measure and also a metric that tells us the amount of similarity (or dissimilarity) between two groups. We use cosine similarity and silhouette score for these purposes, respectively.

Silhouette score is mainly used to measure how well a set of samples is clustered and to compare the results of different clustering methods or configurations [69]. Silhouette score is used to measure the relative distinctiveness of two groups of sequence vectors. The silhouette score (Equation 3.4) is calculated using the mean intra-group distance ($a(i)$ in Equation 3.2) and the mean nearest-group distance ($b(i)$ in Equation 3.3) for each sequence i. To find the distance between sequence i and group j, the distances between i and all sequences in group j are found, and their average is calculated.

$$a(i) = \frac{1}{|G_i| - 1} \sum_{j \in G_i, i \neq j} d(i, j) \tag{3.2}$$

$$b(i) = \frac{1}{|G_i|} \sum_{j \in G_i} d(i, j) \tag{3.3}$$

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{if } |G_i| > 1 \tag{3.4}$$

and $s(i) = 0, \text{if } |G_i| = 1$.

The best value of silhouette score is 1 and the worst value is -1. A score of 1 indicates that two groups are completely separate. Values near zero show that groups are overlapping. Negative values generally indicate that sequences of one group will be incorrectly assigned to the other group. If the silhouette score between the vectors of the two groups is higher, they are more distinct from each other. We calculate silhouette score to measure the separation between groups of event sequences.

Sequence Difference Detection

The vectorization method explained in Section 3 is good for large-scale comparison of sequences and reveals the differences in short and exact subsequeneces. However, this vectorization method is computationally expensive for comparing large subsequences. This section illustrates how we use matrix profiles to summarize differences between two groups of sequences based on large subsequences. The matrix profile, introduced by Yeh et al. [96], is used for time series analysis. The matrix profile is a vector calculated between two time series (similarity join) or one time series and itself (similarity self-join); for each subsequence of the first time series, the distance is stored to its closest subsequence in the second time series. The distance between two subsequences is their Euclidean distance. Matrix profiles have many applications in time series analysis including motif discovery, discord/anomaly detection, and semantic segmentation [96]. In this study, we created the matrix profile for discrete sequences.

Yeh et al. [96] applied the matrix profile to DNA sequences but they convert the sequences to time series first using a method proposed by Rakthanmanon et al. [65]. We extract the matrix profile directly from discrete sequences. The main difference between matrix profiles on sequences versus time series is the distance calculation function. Euclidean distance is defined on real values and thus cannot be applied to sequences. Therefore, we use distance functions designed for discrete sequences: Hamming distance and Longest Common Subsequence (LCS).

Hamming distance is a simple distance function that calculates the number of mismatching positions between two sequences of equal length. The Hamming distance calculation is fast with a time complexity of $O(w)$ where w is the length of its input sequence. The distance matrix calculation time complexity is $O(n^2w)$ if Hamming distance is used.

Longest Common Subsequence (LCS) is a classic dynamic programming algorithm that finds the

longest subsequence common between two sequences; $w - \text{len}(LCS)$ is the LCS-based distance metric between two subsequences of length w. LCS time complexity is $O(w^2)$ where w is the length of LCS input sequences. Hence the distance matrix calculation time complexity is $O(n^2w^2)$.

Using these distance measures, we create the matrix profile of a sequence and itself (similarity self-join). The first step in the creation of the matrix profile is constructing the distance matrix D which is a $n \times n$ matrix where $D[i, j]$ represents the distance between ***i-th*** and ***j-th*** subsequences. For efficiency reasons, Yeh et al. [96] skip calculating the 2-dimensional distance matrix. However in this study, our main purpose is providing tools that facilitate pattern discovery within and across sequences, hence it is helpful to store and visualize the 2-dimensional distance matrix. Algorithm 1 shows the distance matrix calculation procedure; distance(i, j) is the distance between subsequence i and subsequence j which can be calculated using Hamming or LCS-based distance.

ALGORITHM 1: Distance matrix calculation

Input: s: input sequence and w: window size
Output: D: pairwise distance between subsequences

```
l = length(s)
n = l − w + 1
D = n × n matrix with default values of w
r = w/2
for i = 0 to n do
    for j = i + r to n do
        d = distance(i, j)
        D[i, j] = d
        D[j, i] = d
    end for
end for
```

The algorithm takes one sequence s and a window size w as input and generates a distance matrix D. The matrix D is initialized with window size w because this is the maximum distance possible between two subsequences. To avoid trivial matches for each subsequence s_i, [96] suggests excluding a region of length w centered on the starting position of s_i. Since the distance between subsequence i and j ($D[i, j]$) is the same as the distance between subsequence j and i ($D[j, i]$), our

distance matrix is symmetric. For time efficiency, we only calculate $D[i, j]$ and assign it to $D[j, i]$.

After the calculation of distance matrix D, the matrix profile is generated by considering the lowest value of each row as the matrix profile value of that row (Equation 3.5).

$$P[i] = \min_{\forall j \in n} D[i, j] \tag{3.5}$$

Figure 3.1 shows an example of the matrix profile calculation for a toy sequence. In this example, the sequence length is 13, and the window length is 3. Therefore, distance matrix is an 11×11 matrix ($n = l - w + 1 = 13 - 3 + 1 = 11$), summarizing the pairwise distance between subsequences. The matrix profile is calculated by finding the minimum distance to other subsequences for all subsequences.

We calculate the similarity self-join matrix profile for all sequences in the dataset. Characteristics of these matrix profiles such as minimum, maximum, and variance reveal interesting properties of the sequences including motif positions, discord positions, and sequence complexity [96].

Contrast Motif Discovery

This section introduces our approach for finding the top c most abundant motifs for the group of sequences. Then, we discuss our procedure for refining motifs to discover the contrasting ones.

Motif Finding

In order to find the initial set of candidate motifs, we use a modified *SnippetFinder* algorithm, which was designed to detect snippets in time series [35]. The *SnippetFinder* algorithm uses a

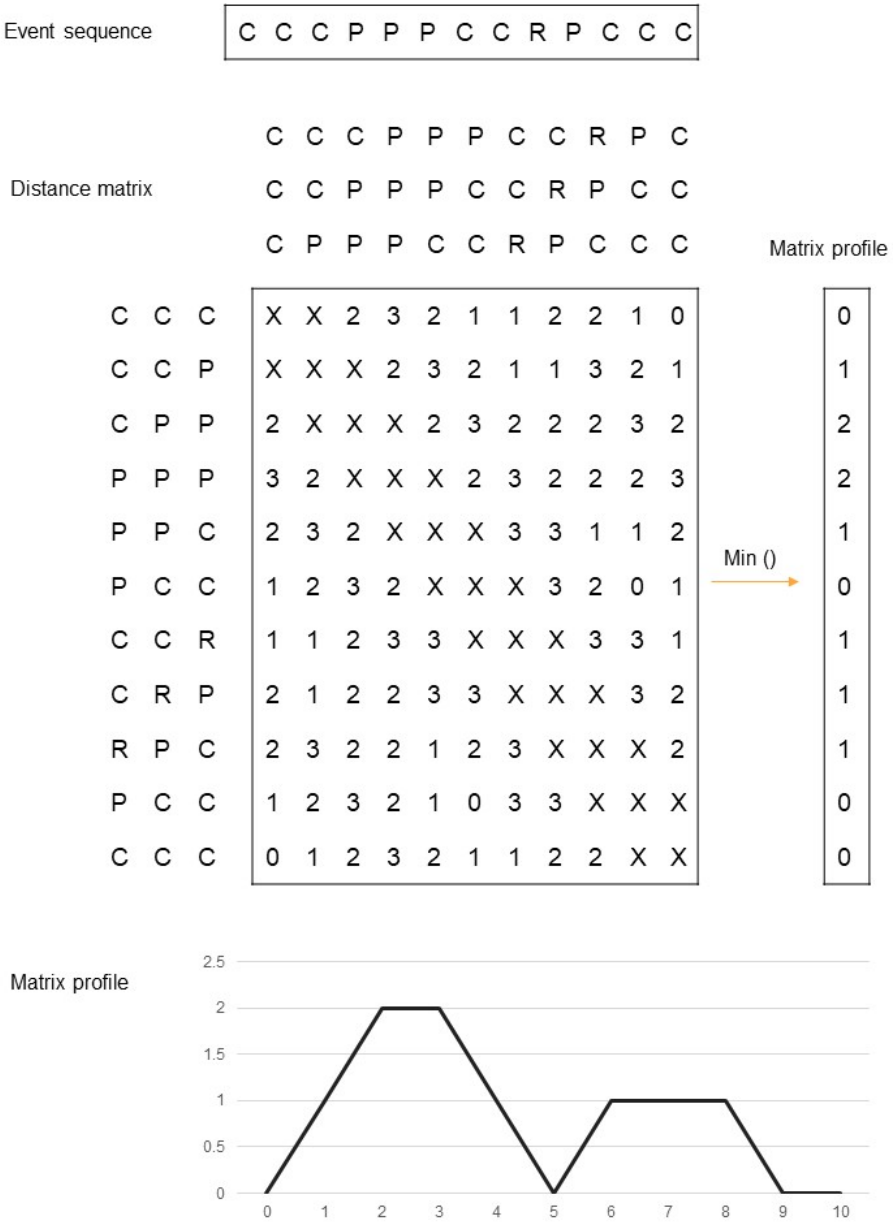


Figure 3.1: Matrix profile calculation example. From the event sequence, first the distance matrix is calculated. This is a 2-dimensional matrix representing the distance between each pair of subsequences of length 3 (= window length). The matrix profile is calculated by selecting the minimum value of each row in the matrix which represents the distance to the closest subsequence.

single time series, rather than a discrete event sequence. The main building block of this algorithm is the matrix profile, a vector of real-valued numbers representing the pairwise distance between subsequences of two sequences [96]. Since our goal is to extract common subsequences shared across multiple discrete sequences, we made modifications to the *SnippetFinder* algorithm. Our motif finding approach is presented in Algorithm 2. The set of subsequences that are produced by the algorithm are used as the candidate motifs.

ALGORITHM 2: Candidate motif finding algorithm

Input: S: input sequences, c: number of candidate motifs and w: window size
Output: M: top c snippets of S

Remove short sequences from S
$pm =$ list of all possible motifs of length w in S
$P =$ collection of profiles between pm and S
$m = size(pm)$ (i.e. number of possible motifs)
$l =$ profile length
$Q =$ array of length l initialized with inf
$M =$ empty array to store candidate motifs
while $size(M) < c$ **do**
    $min\_area, min\_idx, min\_motif = inf, -1, None$
    for $i = 1$ to m **do**
        $cur\_motif = pm[i]$
        $e =$ element-wise min between $P[i]$ and Q
        $cur\_area = sum(e)$
        if $cur\_area < min\_area$ **then**
            $min\_area = cur\_area$
            $min\_idx = i$
            $min\_motif = cur\_motif$
        end if
    end for
    $Q =$ element-wise min between $P[min\_idx]$ and Q
    Add min_motif to M
end while

The algorithm takes the list of sequences (S), desired number of motifs (c) and window size (w) as input and outputs a list (M) containing c candidate motifs. As a data preparation step, sequences shorter than the window size are removed from the list, since it is impossible that they contain motifs of length w. The list of all possible motifs (pm) is created by moving an overlapping

window of size w along all sequences. Then the matrix profile of every possible motif is created. This matrix profile contains one value for every sequence which represents the distance between the possible motif and the sequence. The matrix profile length (l) is the length of the profile for each possible motif which is equal to the number of sequences. The distance between one sequence and a possible motif is the minimum value of the sequence profile (i.e. the minimum distance between the possible motif and subsequences of the sequence). In this study, LCS distance is used to measure the distance between subsequences. LCS distance ($LCS\ distance = window\ size - LCS\ score$) is based on the well-known Longest Common Subsequence (LCS) algorithm.

After creating the collection of profiles (P), the next step is to find the candidate motifs. The algorithm iteratively finds candidate motifs and continues to look for new candidates until it reaches the user-specified number of motifs. In Algorithm 2, Q is an array that stores the element-wise minimum values of the profiles related to candidate motifs so far. This array is initialized with inf values and is used in successive iterations to find candidate motifs in new sequences. In each iteration, the algorithm loops over possible motifs to determine which of them has the minimum area under the e curve. The e curve shows how close the possible motif is to the sequences for which no motif has been discovered. The algorithm stores the candidate motifs in a list (M), which is the output of the algorithm.

Running this algorithm requires $O(w^2n^2)$ time; where n is the total length of input sequences. Since w, which is the motif length, is a small constant number, it can be disregarded. Therefore, the time complexity of this algorithm is $O(n^2)$.

Motif Refinement

Algorithm 2 generates a list of candidate motifs for a group of sequences. Since our goal is to discover contrasting motifs that are more similar to their group while being distant from other groups,

we need to filter candidate motifs of each group. This is a subgroup discovery task: identifying interesting subgroups of objects with respect to a particular feature. A subgroup of objects is interesting when the feature values within the subgroup differ in a statistically significant way from the feature values of the other objects [44]. In this study, we constructed interesting subgroups of motifs using a *Mann–Whitney U test*. Motifs are only selected 1) if the average distance to the sequences in their group is lower than the distances to the sequences from other groups and 2) this difference is statistically significant (i.e. the *Mann–Whitney U test* has a *p-value* lower than 0.05). Our algorithm does not assume that motifs exist in all sequences of the group, but it detects the motifs that are closest to all sequences of that group.

Chapter Summary

In this chapter, we introduced two novel sequence mining approaches that can be utilized to study differences between groups of discrete sequences. The first approach is based on analyzing silhouette score and various statistics related to aggregated matrix profiles of sequences. The second approach is capable of finding short subsequences that are significantly closer to one group of sequences compared to others. Comparing these short subsequences gives us a better understanding of what distinguishes the groups of sequences. In Chapters 5 and 6 we discuss the applications of these approaches in GitHub and Minecraft.

CHAPTER 4: GITHUB DEVELOPERS AND TEAMS

Discovering GitHub Developer Archetypes

GitHub repositories typically have several developers working on the project as a virtual team. However open source projects hosted on GitHub can be downloaded and copied thousands of times, spawning an ecosystem of related repositories. Agent-based models are a powerful tool for predicting population level behaviors; however their performance can be sensitive to the initial simulation conditions. To create a versatile agent-based model for simulating large-scale usage trends of the GitHub collaborative development tool we proposed a procedure for initializing agent-based simulations in which the population is abstracted into a set of archetypes. Although it is possible to predict usage trends on GitHub using purely machine learning approaches [10], we believe that a hybrid approach of agent-based modeling and data mining is more promising, enabling us to explore a richer range of community interactions.

One challenge of modeling developer behavior is that GitHub has become popular as a general purpose hosting and communication tool for myriad types of efforts, ranging from personal software archives to large open source projects with millions of users. Many repositories are not directly related to software development but are instead used to curate document collections [94]. Previous studies of computer-supported cooperative software development have attempted to survey the developers to understand the differences between individual contributors vs.rockstar programmers and popular curators [9]. There is a large amount of variability in the usage rates of GitHub, with some developers submitting hundreds of changes in a month, but with most users remaining completely dormant or passively observing.

Expressing the diversity of the user population within a single agent-based framework is demand-

ing. Rather than relying on existing taxonomies of user behavior created from survey data, we extracted the archetypes from the user's contribution history from the most stable clusters found by k-means clustering. This approach also has the advantage of simultaneously producing the relative distribution of each archetype across the developer population, along with the monthly activity. Our results show that our archetype extraction and simulation initialization procedure produces more accurate predictions of population behavior, as measured by the Gini coefficient of contributor activities.

Our experiments examine three questions:

1. do stable clusters exist across consecutive months in the partitioned GitHub data?
2. does the ABM configured with the extracted archetypes outperform the simple mean model?
3. are archetypes extracted from more stable clusters better than those from less stable clusters?

Our dataset consists of all GitHub users and repositories created before March 2017 along with the activity data from January 2015 to February 2017. We divided 26 months of data into 20 months for training the clustering and 6 months for testing the simulation. There are approximately seven million users with at least one activity during the training period, but we restrict our analysis to the three million users with greater than ten total activities. The GitHub activity dataset consists of 14 event types: *CommitComment*, *Create*, *Delete*, *Fork*, *Gollum*, *IssueComment*, *Issue*, *Member*, *Public*, *Pullrequest*, *PullrequestReviewComment*, *Push*, *Release*, and *Watch*.

We created activity profiles of GitHub users using the average monthly activity per event type to be used as clustering features. Since GitHub users have a wide range of activity levels, first we partitioned users based on their average monthly activity and then clustered each partition separately. Table 4.1 shows the number of users in each partition.

Table 4.1: GitHub user partitions

Partition	Average monthly activity	Number of users
1	(0,10]	1.4M
2	(10, 100]	1.5M
3	(100, 1K]	44K
4	(1K, 10K]	741
5	(10K, inf]	69

Since the range of values for different event types varies widely, we normalized features by scaling them to lie between zero to one. We clustered each partition separately using k-means but restricted our analysis to partitions with greater than one hundred users.

Cluster Stability

One question is whether clustering the data from different time periods yields the same archetypes. Are the data-driven archetypes more sensitive to monthly activity fluctuations than archetypes described in survey studies? To examine this question, we performed a cluster stability analysis to measure whether similar clusters are observed from month to month. Similarity is computed between clusterings of consecutive months, and the stability score is the average similarity score of all consecutive months [85]. The Adjusted Rand Index (ARI) is used to measure similarity between clusterings. ARI is 1.0 when clusters are identical and close to 0.0 for random labeling [33]. The following procedure is used to calculate stability score for each k value:

Given a set $M = m_1, m_2, ..., m_n$ of monthly user activity profiles in the training months, the k-means algorithm takes the number of clusters, k as input:

1. For $k = 2, ..., k_{max}$

   (a) For $i = 1, ..., n$

   Cluster data of m_i into k clusters to obtain model CM_i and clusters C_i

   (b) For $i = 2, ..., n$

   Cluster data of m_i using CM_{i-1} to obtain clusters C'_i

   (c) Compute stability as the mean similarity between clustering C_i and C'_i

$$Stability(k) = \frac{1}{(n-1)^2} \sum_{i=2}^{n} Similarity(C_i, C'_i) \tag{4.1}$$

2. Choose the parameter k that gives the highest stability:

$$K = \arg\max_k Stability(k) \tag{4.2}$$

We examined the stability of the clustering across consecutive months. Table 4.2 shows the stability score for k-means clustering with $k = 3, ..., 9$ in all 4 partitions. k values of 4, 3, 4, and 3 generate the most stable clusters for partition 1, 2, 3, and 4 respectively. Partitions 1, 2, and 3 definitely exhibit stable clusters as their best stability scores exceed 0.9.

Figure 4.1 illustrates cluster centroids for the partition of users that have between 1K and 10K average monthly activity. The cluster on the right contains users users who perform most of their tasks individually as their dominant activity is push event. The middle cluster corresponds to the users who not only perform individual work by submitting push event, but also participate in collaborative activities such as commenting on issues and pull requests. The cluster on the left is related to users who have managerial roles as they mostly engage in coordinating and administrative activi-

Table 4.2: Stability score for different partitions

# Clusters	0-10	10-100	100-1K	1K-10K
3	0.931	**0.996**	0.920	**0.685**
4	**0.949**	0.988	**0.928**	0.671
5	0.912	0.919	0.841	0.557
6	0.825	0.989	0.867	0.604
7	0.795	0.934	0.751	0.594
8	0.791	0.974	0.796	0.577
9	0.769	0.973	0.617	0.571

ties such as reviewing pull requests, releasing the software, and creating development branches in addition to discussing issues and pull requests.

Archetype Model

The clustering results were then used to initialize the archetypes included in our agent-based model of GitHub repository contribution, developed on NetLogo 6.0.2 [92]. *General User* archetypes were created using the best and second best clusters from partitions 1 to 4 in Table 4.1. *Hyperactive Users* were defined by aggregating the event activity profiles of the 69 users in partition 5 into mean frequency per event type. Accordingly, the cluster size for *Hyperactive Users* was set equal to the count of users in partition 5.

Using the above archetypes, a scaled-down agent-based model was constructed. Two agent breeds were modeled: 1) *Users* and 2) *Repositories*. *Repositories* were considered a non-active breed of agents that kept track of contributions made by *User* agents. *User* agents were allowed to perform one out of a set of actions, U, that reflected actual GitHub events plus the event *Idle* for the case that a User did not perform an event during that time step. Since activity was partitioned on monthly basis, the per minute frequency of a GitHub event ($U_j^{GH} : U^{GH} = U - \textit{Idle}$) being triggered by a

General User ($a_i : i <= 16$) was referred to as $F_{U_j^{GH},a_i}$ and calculated as:

$$F_{U_j^{GH},a_i} = \frac{A_{U_j^{GH},a_i}}{43200} \tag{4.3}$$

For *General Users*, $F_{U_j^{GH},a_i} < 1$ and was modeled as the probability that a user of archetype a_i would trigger a GitHub event U_j^{GH} in a discrete simulation time step. Accordingly, each discrete time step was used to represent a minute. *Hyperactive Users* were modeled as triggering $F_{U_j^{GH},a_17}$ where $F_{U_j^{GH},a_17} > 1$ such that many events of type U_j^{GH} were generated per simulation time step. This difference in event triggering could have been accommodated by modeling simulation time steps as milliseconds but was performed to reduce the runtime of the simulations to a computationally feasible limit. The model was scaled down by 1000th of the population size of GitHub in the training data and cluster sizes.

In addition to the rate at which *Users* performed events of different types which was informed by the clustering results, there was the question of modeling the target repository for each event. To handle this, we obtained the mean number of repositories a user would interact with during a month from the data (mean = 3.6, st.dev = 37.046). These values were used to calculate the maximum number of repositories a user would work with in the simulated month, μ, through a gamma distribution ($\alpha = \frac{3.6^2}{37.046^2}$, $\lambda = \frac{3.6}{37.046^2}$).

Users maintained a list of familiar repositories. Each simulation time step, a user agent selected a behavior to perform based on the event frequency defined by its archetype. If this event was a contribution event, one of the repositories in its contribution list would be selected as the target of this event. If, in a simulation time step, a user decided to perform a *Watch* or *Fork* event, the user chose σ repositories at random from the repository population and selected the repository with the highest sum of *Fork*s and *Watch*es from this subset as the target for this action. This repository

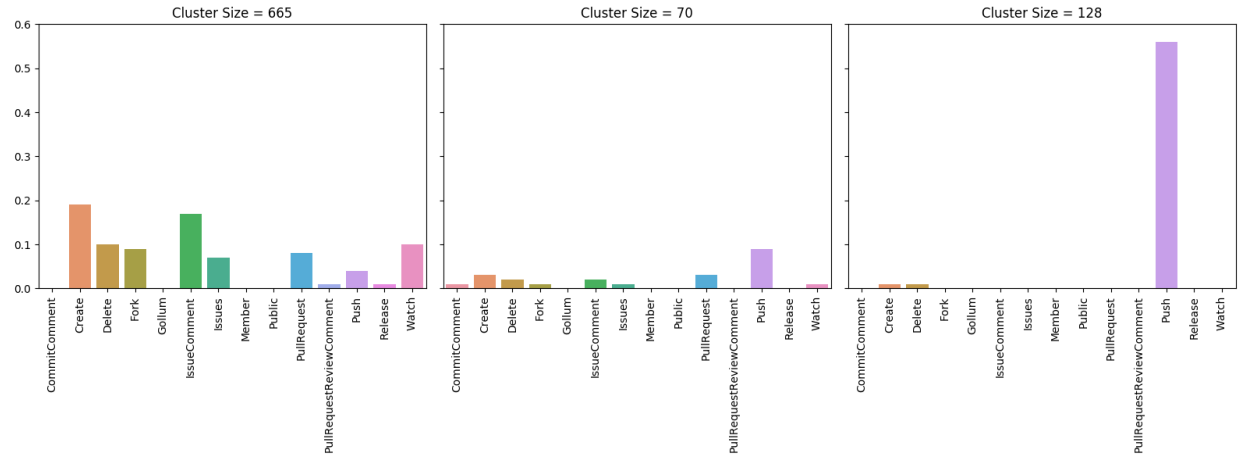


Figure 4.1: Cluster centroids for users with more than 1000 and less than 10,000 average monthly activity

would then be added to its list of familiar repositories, displacing a repository already in this list at random if the list was already at capacity μ.

Some key ABM parameters were directly inferred from statistics of the training data. New *Users* were injected into the simulation at a probability of 0.008 per time step. *Create* events generated new repositories at a probability of 0.481955 as actual create events can result in repositories, branches and tags.

GitHub events can be grouped into three general categories: contributions, watches, and forks (copies). To evaluate the simulation performance of our extracted archetypes, we ran several configurations of the agent-based model. The baseline (Configuration 0) models the entire population using the mean event frequencies. Configuration 1 uses the archetypes from the most stable clustering result on each partition, yielding 15 archetypes (14 *General Users* and 1 *Hyperactive User*). Configuration 2 used the second best clustering results yielding 17 archetypes. The simulations were run for 43200 time steps, simulating a month of GitHub activity with $\sigma \in 1, 2, 4, 8, 16, 32$. Each configuration was repeated ten times to obtain aggregate simulation results.

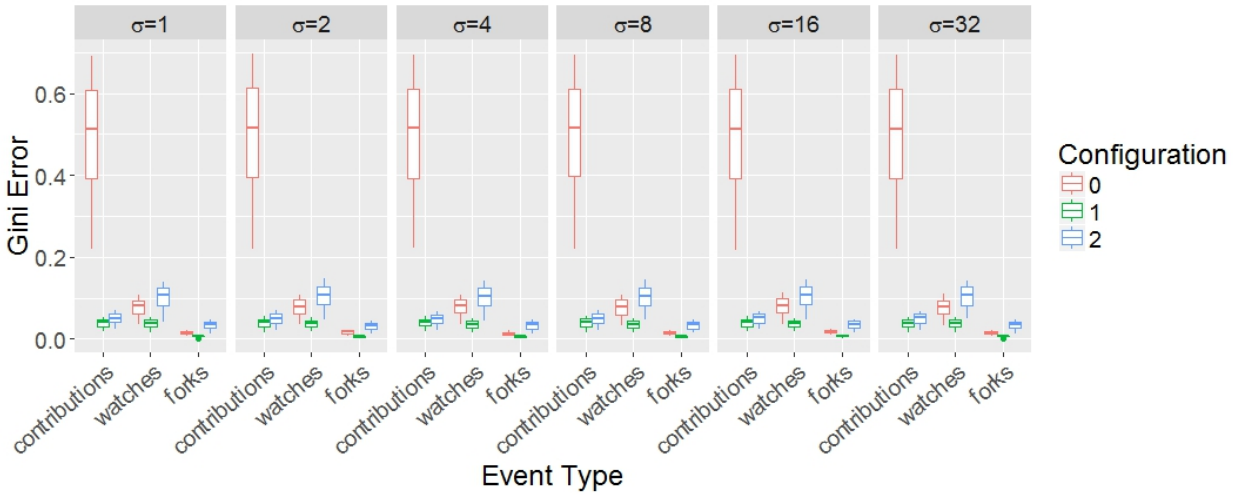


Figure 4.2: Error of Gini Coefficient for Users. Configuration 0 (without cluster information) performs badly at predicting the dispersion of contributions across users. Configuration 1 (most stable cluster) is the best performer yielding a small improvement vs. using the second most stable cluster to initialize the archetypes.

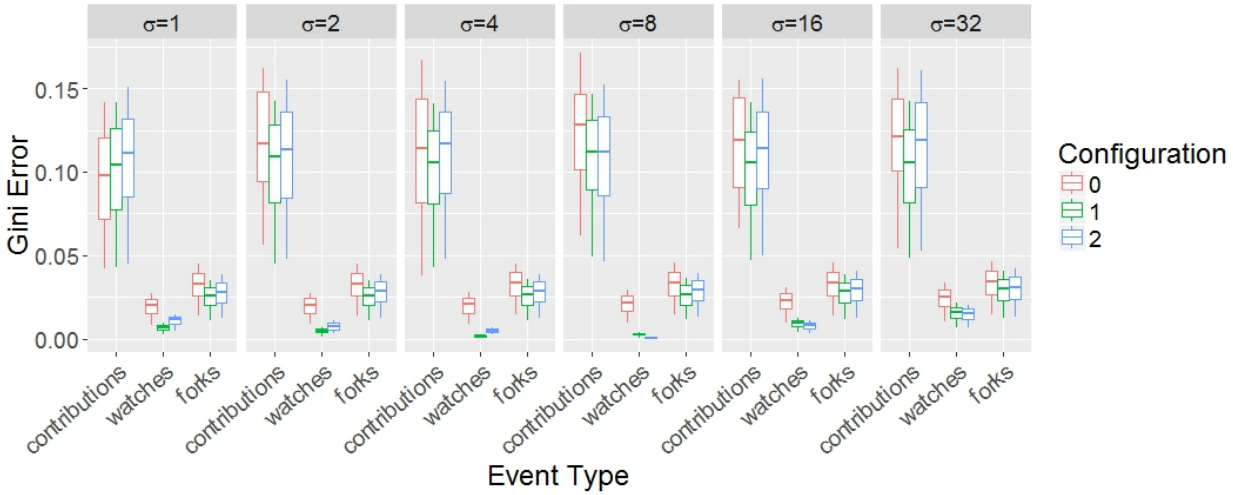


Figure 4.3: Error of Gini Coefficient for Repositories. Configurations 1 and 2 (cluster based archetypes) yield slightly better performance. However, the ad hoc heuristics used by the simulation for repository assignment do not perform as well at allocating events across repos.

Although our ABM is designed to answer questions about many types of GitHub trends, we are particularly interested in accurately modeling the relative activity levels of users and repositories since these are core aspects of the ABM that affect many population-level trends. Our experiments measure the absolute error of different initial archetype populations at predicting the Gini coefficient over one month of test data. Rather than looking at the errors of specific event types, we group the events into meaningful action categories: 1) contributions, 2) watches, and 3) forks. Figure 4.2 shows the performance of the cluster-based archetypes at predicting the Gini coefficient over user contributions for one month of test data. We also study the performance of our repository allocation heuristics at predicting the Gini coefficient over repository activity (Figure 4.3).

The stable cluster-based user archetypes outperform the baseline and the less stable clusters at predicting the dispersion of activity across users. These archetypes offer slight improvements in calculating the dispersion across repos, however the heuristics for repo assignment do not perform as well.

GitHub Team Productivity

Data Set

The data set used in this study contains all GitHub events from January 2016 to June 2017. From this data set, we selected software repositories created in January 2016 and included only those that had more than 20 work events and at least two members in the team formation phase. We measured size of the teams, once more, in the evaluation period and removed the repositories with less than two members. Our final data set included 20,370 active repositories and the 59,178 unique GitHub users contributing to those repositories in the evaluation period.

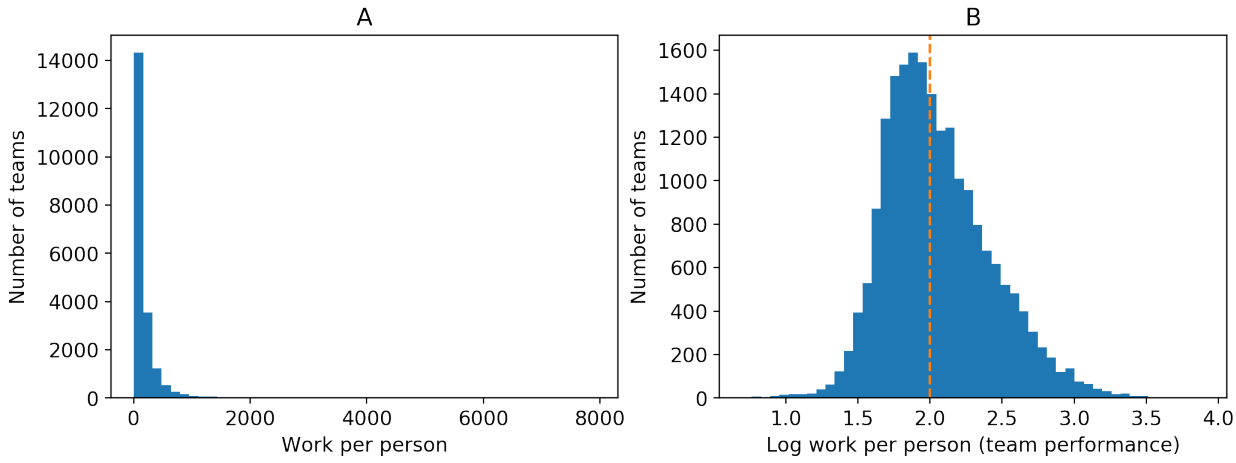


Figure 4.4: Distribution of work per person per month (A) and transformed distribution of work per person per month (B).

Team Performance Evaluation

First, we examine the characteristics of the work event data during the performance evaluation period. Performance is defined as the amount of the work teams completed per person in the evaluation period. Figure 4.4A shows that the distribution of work per person is a heavy-tailed distribution. A log transformation on work per person was applied to decrease the variability of this measurement. The transformed distribution is a normal distribution with a mean and variance of 2.05 and 0.37, respectively (Figure 4.4B). We considered the log transformed values of work per person representative of team productivity. To categorize teams based on their productivity, we used the median of team productivity (= 2.0) as our threshold and labeled teams as high-performing if their productivity was greater than the threshold and low-performing otherwise.

Our unit of analysis is based, in part, on the size of the team during the performance evaluation period. Figure 4.5 provides the histogram of team sizes in our data during the performance evaluation period.

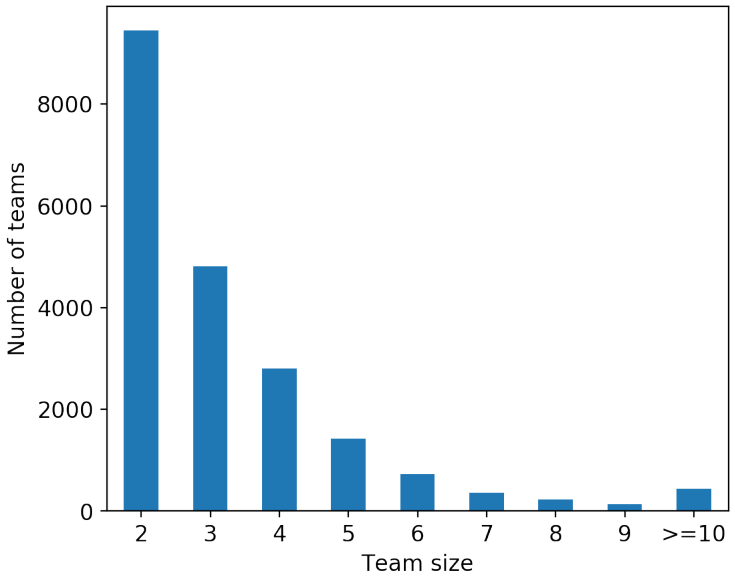


Figure 4.5: Team sizes in the performance evaluation period.

Evaluation Period Team Sizes

For additional analyses, we grouped teams based on their size: teams with three or fewer members were considered small, teams with between four and six members were considered medium, and teams with seven or more members were categorized large. Vasilescu et al. [83] used higher thresholds to categorize teams in software repositories on GitHub. However, given the short life span of the repositories we studied, we determined these thresholds were more appropriate to capture differences in processes and outcomes across team sizes. The number and the proportion of teams in each group is provided in Table 4.3. In the performance evaluation period, more than half of the repositories in our data set were maintained by small teams and less than a quarter were maintained by medium-sized teams. Large-team repositories made up less than five percent of the data.

Figure 4.6A shows the performance of teams of different sizes; teams with two members have

Table 4.3: Proportion of teams by team size.

Group	Size	Proportion	Frequency
Small	[2, 3]	70%	14261
Medium	[4, 6]	24%	4951
Large	[7, inf)	6%	1157

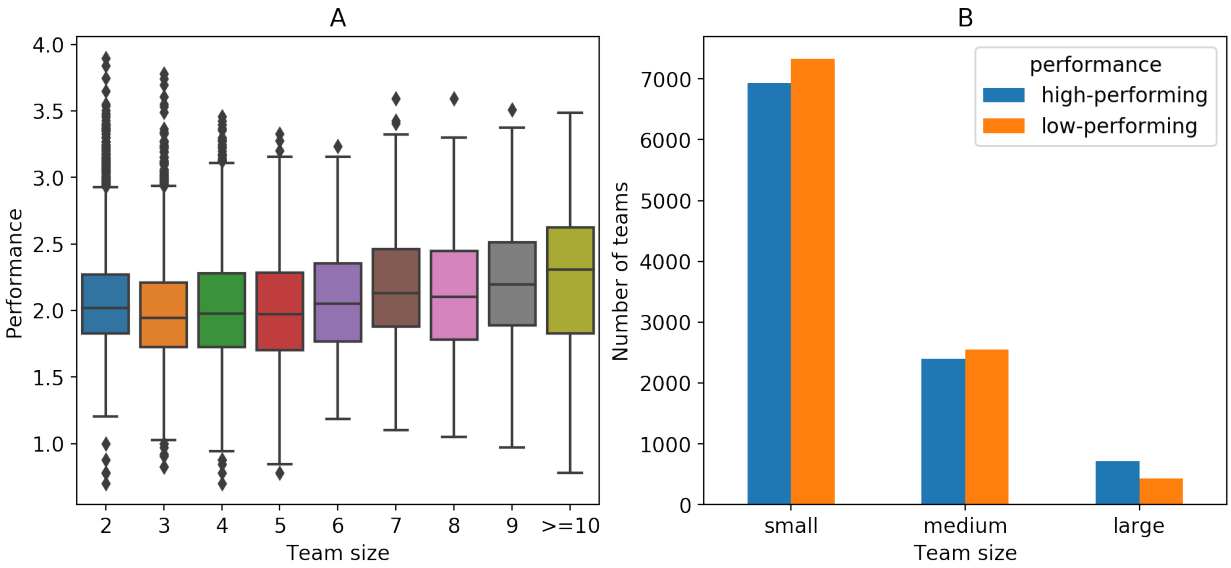


Figure 4.6: Performance of teams by size (A) and high- and low-performing teams by team size groups (B).

slightly higher performance compared to teams with three, four, or five members. Given that most contributors do very little work in GitHub repositories [84], this finding is not particularly surprising. Figure 4.6B shows the number of high- and low-performing teams in each team size category. Interestingly, there are more high performers than low performers in the large team size group. This suggests that, in our data set, large teams are likely engaged in more frequent interactions and potentially more effective collaborations leading to the production of more communications and code artifacts (i.e., more work events per person).

Work Centralization

To examine the distribution of work underlying the performance differences observed across team sizes, we calculated the amount of work centralization in teams using the Gini coefficient. The Gini coefficient is a measure of inequality, where 0 represents perfect equality and 1 represents maximum inequality. We use it to analyze the inequality of work events per person. More specifically, a higher Gini coefficient suggests that a smaller set of team members do most of the work; that is, work is centralized to a fewer number of members in the repository. Figure 4.7 provides distributions for the Gini coefficient values of teams in the performance evaluation period. High-performing teams have a higher Gini coefficient compared to low-performing teams, regardless of team size. The difference between the Gini coefficient of low- and high-performing teams is greater in large teams compared to small and medium teams. This finding provides support for the importance of work centralization for performance in OSSD in GitHub [36]. Our results suggest that work centralization is linked to increased overall productivity in software repositories and is particularly important for collaborations in large teams.

Work Style Clusters

GitHub users and teams behave differently and clustering them can express their diversity [70]. To find pattern of activities of teams and discover various work styles on GitHub, we clustered teams using the k-means clustering algorithm applied to the proportion of different types of work events. The number of teams in each of the three work style clusters is provided in Table 5.3 and the cluster centroids are plotted in Figure 4.8. Based on the proportion of events in work style clusters, we labeled them as: *toilers*, *communicators*, and *collaborators*.

Toilers produce a higher proportion of push events compared to *communicators* and *collaborators*

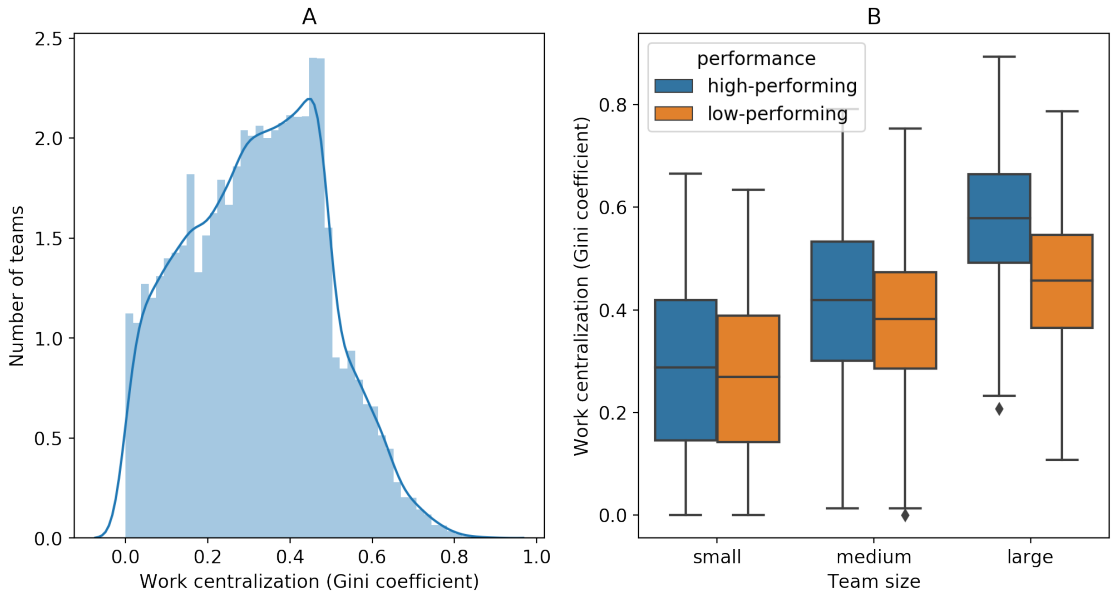


Figure 4.7: Distribution of Gini coefficients for all teams (A) and distribution of Gini coefficients by team size and performance (B).

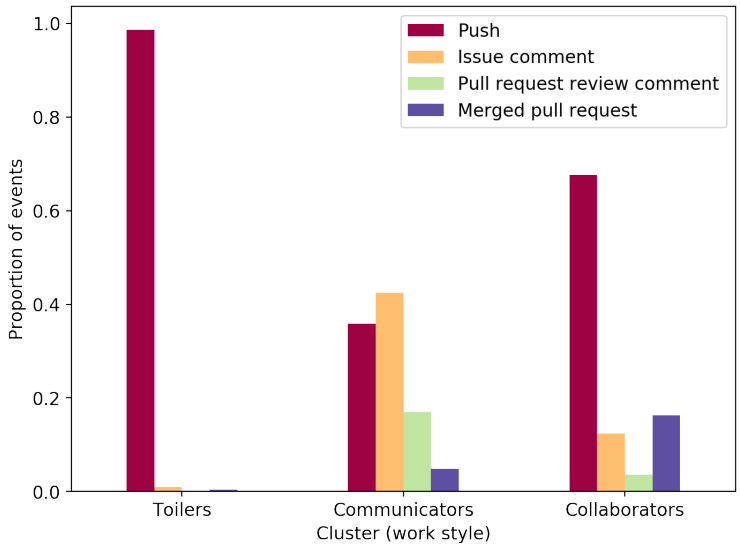


Figure 4.8: Proportions of events by work style cluster.

Table 4.4: The mean and standard deviation of different work events for toilers, communicators, and collaborators.

Work style cluster		*Toilers*	*Communicators*	*Collaborators*
Number of teams		15264	3219	1886
Push	mean	79.61	68.61	91.60
	std	118.19	143.40	128.98
Merged PR	mean	0.55	14.93	23.12
	std	4.53	47.79	45.15
Issue comment	mean	0.86	109.72	17.31
	std	4.61	477.73	35.62
PR review comment	mean	0.09	41.12	5.65
	std	1.51	142.60	19.54

but have limited, if any, communication taking place in GitHub. This may reflect a failure to engage in explicit coordination but it is also possible that these teams use alternative communication channels, like Slack [1] and Discord [2], to coordinate their contributions, as past research has shown [78]. Although *toilers* focus on code contributions, they generally do not accept external contributions, unlike the other two work style groups. *Communicators* produce a higher proportion of comment events, and *collaborators* produce a higher proportion of pushes and merged pull requests.

For additional analysis, we calculated the average number of each event type for different work styles (Table 5.3). Overall, *collaborators* have the highest level of productivity in terms of internal and external code contributions (pushes and merged pull requests, respectively) whereas *communicators* have the highest level of productivity in terms of explicit coordination. Although *toilers* devote the majority of their effort to code contributions, they still have, on average, a lower number of pushes compared to *collaborators*.

[1] https://slack.com/

[2] https://discordapp.com/

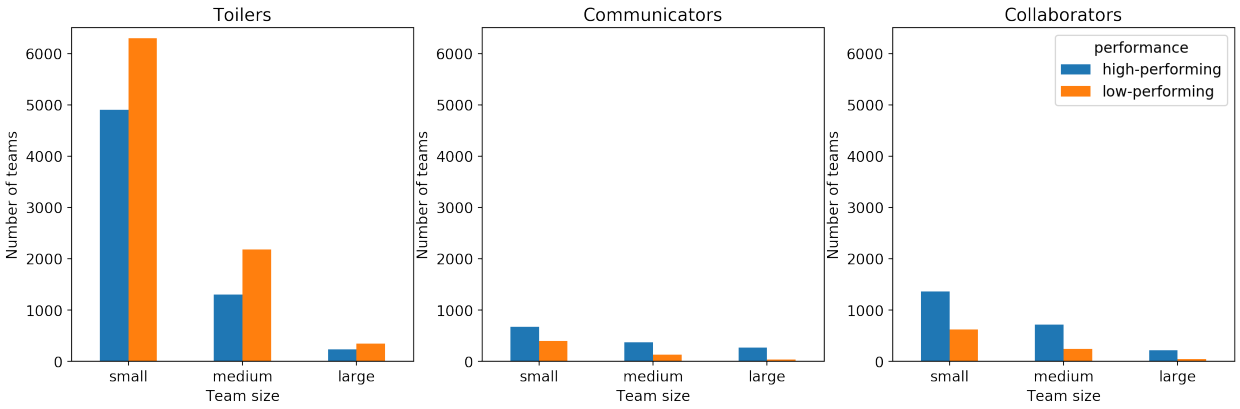


Figure 4.9: The performance of different work styles and team size groups.

Work Style and Performance

Figure 4.9 shows the number of high- and low- performing teams in each work style cluster. Nearly 75% of the teams in our data are in the *toilers* cluster. Of this subset, the majority of them were low-performing teams. This suggests that a lack of communication within GitHub is indicative of poor performance. Although *toilers* could be communicating outside the GitHub ecosystem, the fact that they did not perform relatively well suggests that direct communication within GitHub (e.g., issue comments) may help overall team performance. This is illustrated when we look at the *communicators* and *collaborators* clusters. For the *communicators*, we see a relatively more equal distribution of comments proportional to the amount of work done. For the *collaborators*, there is relatively more communication compared to the toilers. And, proportionate to the amount of other activity, they also accept more pull requests than any of the other clusters. In the *communicators* and *collaborators* clusters, the majority of teams are high performers. This effect is more pronounced for large teams and, to a lesser extent, medium teams compared to small teams.

Team Feature

We extracted team features based on event data in the team formation phase.

- **Relative frequency of work events** is the ratio of the number of each work event to the total number of work events. This feature allows us to evaluate the type(s) of work undertaken during team formation.

- **Work events per person** is the number of each work event divided by the number of team members. We used this feature to evaluate the amount of work, or productivity of contributors.

- **Burstiness** measures the temporal correlation of activities within a team and defined as equation 4.4; where μ_τ and σ_τ are mean and standard deviation wait times $P(\tau)$.

$$Burstiness = \frac{\sigma_\tau - \mu_\tau}{\sigma_\tau + \mu_\tau} \tag{4.4}$$

  An analysis of burstiness reveals the presence of increased, synchronized activity in a team and is linked to effective collaborations and improved team performance [66].

- **Issue labeled proportion** is the proportion of issues in the repository that are labeled. Because issue labels are one way that developers can organize their work and communication on GitHub, we use this feature to evaluate the use and organization of cognitive artifacts by the team.

- **Team size** is the number of team members in the repository during the team formation phase. This feature was used to assess the importance of the team's size early on in its development.

Our goal is to understand the impact of team formation phase on the evaluation period. Figure

4.10 illustrate the team formation phase features that are linked to performance of teams in the evaluation period; that is, the features predictive of success. Across each of these, t-tests show that the *p-value* is less than 10^{-10} in all features represented in figure 4.10. More specifically, these features identified during the formation phase, and indicative of the type of work norms created during formation, are significantly related to higher performance during the evaluation phase.

Types of Activity

Large-team repositories have a higher proportion of coordination events (i.e., issue comments in figure 4.10B) and collaborative work events (merged pull requests in figure 4.10C). Conversely, these repositories have a lower proportion of internal contribution events (pushes in figure 4.10A). High-performing teams have a lower proportion of push events and a higher percentage of coordination events. In other words, high-performing teams exhibit higher levels of coordination. Issues in GitHub repositories can be used to develop an understanding of the problem at hand and the work needed to resolve the problem, and also offer a space for the discussion and evaluation of potential solutions and strategies to address needs of users and the project. The larger proportion of issue comments in high-performing teams shows that dissections of issues are helpful in improving the productivity of the teams. In contrast, low-performing teams have a lower proportion of coordination events and a higher proportion of contribution events. This suggests that these teams are primarily focused on their work output and spend less time on coordination and communication. Fitting with the team cognition literature [20], this may lead to a failure to evaluate alternatives and the premature selection of solutions.

Burstiness

Overall, we observe a higher amount of burstiness in high-performing teams and a lower amount of burstiness in low-performing teams. Interestingly, this difference in high- and low-performing teams is consistent across team sizes. This suggests that high-performing teams, in general, interact more frequently and their activities are highly-synchronized.

Issue Labels

Figure 4.10F shows that proportion of labeled issues is generally higher for high-performing teams than it is for low-performing teams. This is particularly true for medium and large teams and less noticeable in small teams. Again, this finding supports the claim that larger teams require more coordination mechanisms to function effectively. This suggests that, as a classification system for artifacts, the consistent use of issue labels may scaffold collaborative problem-solving processes [21] and thus support the productivity of software development teams in GitHub. More specifically, in line with team cognition theory on complex problem solving [19], issue labels provide support for information-gathering and knowledge-building activities of current and prospective team members.

Chapter Summery

In this chapter, we conducted an in-depth analysis of GitHub users and teams. To enhance our understanding of GitHub users, we aimed to discover user archetypes by finding stable clusters over time. Additionally, we studied GitHub teams by analyzing various aspects of their performance such as productivity and work centralization. We discovered GitHub team work styles based on the proportion of their work activity.

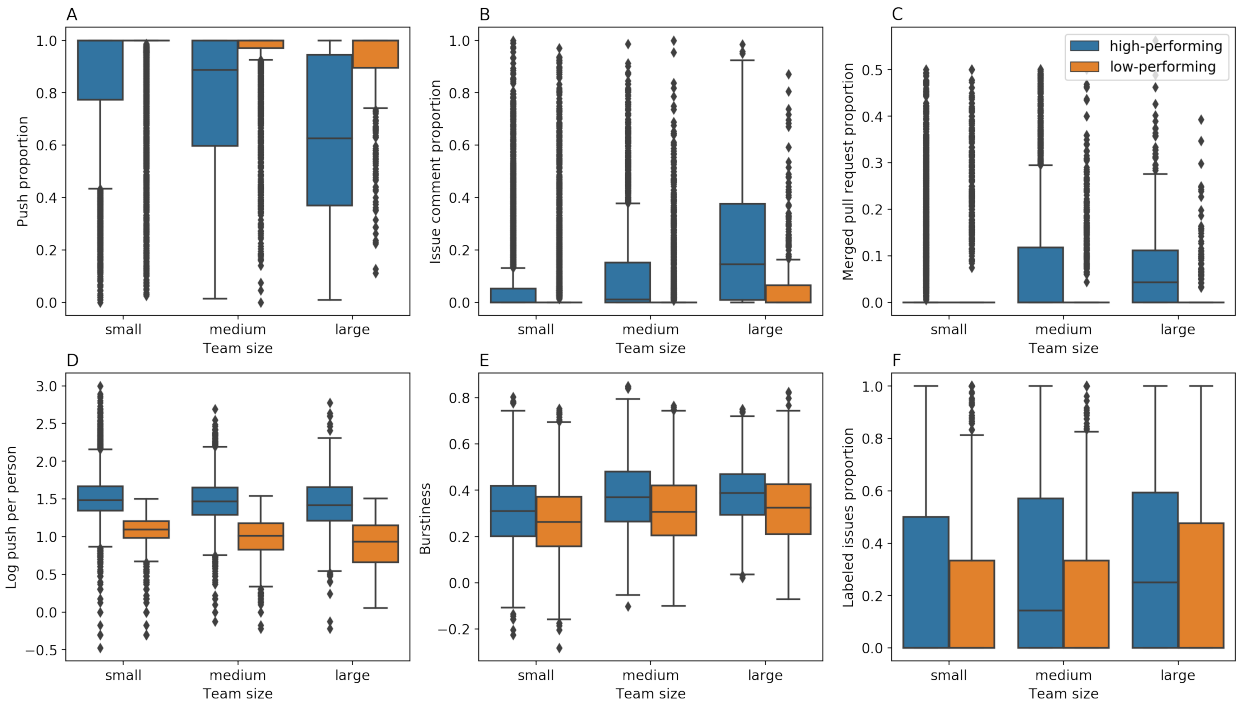


Figure 4.10: Team formation phase features by team size and performance group.

CHAPTER 5: GITHUB HUMAN-BOT TEAMS

This chapter presents research examining the effects of bots on software development team processes and outcomes through an analysis of the event distributions and sequences generated in GitHub repositories. We studied human-bot and human-only teams on GitHub from different aspects including outcome of these two types of teams and their processes.

GitHub Human-Bot Teams Outcome

In this section, we explain our dataset of human-only and human-bot teams, and discuss the effect of bots on the relative productivity level of teams, both in terms of work events completed and speed of issue closure. In addition, we examined differences in work centralization and event distributions between bot-human teams and human only teams. Finally, to investigate how the presence of bots within the team affects the style of work completed by humans on the team we clustered the teams based on their activities.

Data Set

Many studies of software development in GitHub focus on large, mature projects. We contribute to this body of work by, instead, considering the effects of bots in projects with different sizes of the same age. To do this, we selected software repositories created in January 2016 if they were active (at least 20 work events) in the first six months after their creation and had more than two human team members. To evaluate these teams, we examined 13 months of their work events after creation. We considered a GitHub user a member of a team if they have completed at least one of the following: one push event; five accepted pull requests; ten issue comments; or ten pull request

review comments. This dataset contained a total of 20,119 software development repositories.

Past research establishes the relationship between team size and performance differences in OSS projects [68, 38]. Because of this, we grouped teams according to size prior to analysis. We classified teams as small if they had two or three members, medium if they had between four and six contributors, and large if they had seven or more members. In our sample, we identified 304 (1.5%) teams that had at least one bot. Of this subset, 280 teams had 1 bot, 21 teams had 2 bots, and 3 teams had 4, 8, and 12 bots.

Bot Identification

It is not possible to detect all automated activity on GitHub (e.g., if they use regular user accounts to perform actions) and it is beyond the scope of this research to perform an exhaustive search for all automation in work events. In our data set, an account is considered a bot if its type is set to Bot, its name ends with '-bot', and/or it has repeated identical comments. Some of these bots are those that are provided for developers through the GitHub Marketplace[1], the platform's online store for development tools. GitHub Apps, which are official GitHub bots, allow users to automate and improve their workflow. Users can build, share, or sell their Apps on GitHub Marketplace. GitHub Apps can be used for different phases of the development process, from continuous integration to project management and code review.

Control for Developers Expertise

Since our research is focused on studying the impact of bots, we control for expertise to make sure the obtained results are not due to team expertise. To control for the expertise level of the teams,

[1]https://github.com/marketplace

Table 5.1: Number and percentage of human bot teams across team sizes

Team Size	# human teams	# human-bot teams
Small	153	128
Medium	96	84
Large	55	92

we extracted an expertise vector for every team member comprised of 1) number of followers 2) number of following 3) number of public repositories owned by the developer 4) GH-impact score (a measure of influence on GitHub). A developer has a GH-impact score of n if they have n repositories with n stars. To compare teams, we summed the expertise vectors of team members to measure total expertise of the team. To balance our dataset in a way that human only teams have the same level of expertise as human-bot teams, for each human-bot team, we found the most similar human only team with respect to their expertise vector and downsampled human teams to 304 teams corresponding to the 304 human-bot teams. To avoid dominance of variables with large values to be determinant of similarity, we normalized expertise vectors to lie between zero and one before similarity calculation.

Table 5.1 shows the number of teams for different team sizes in addition to the number and percentage of human-bot teams after downsampling human teams.

Team Productivity

Prior studies have measured the amount of work performed by GitHub teams by counting the number of push events [59]. This approach works for repositories that have the *shared repository* development model in which team members are granted push access. However, many large open-source repositories have the *fork and pull* model in which all GitHub users are allowed to fork the repository, make changes and send a pull request to submit their contributions. To account for

these repositories, in addition to push events, we included the number of accepted pull requests in the work of the team. Moreover, we believe that the communications between team members about code development are an important component of teamwork. Therefore, we included these types of comments in the tally of work. Thus in order to measure the total work of a team, we tabulated a subset of the GitHub events for the team repository: push, issue comment, pull request review comment, and accepted pull request.

For each team, we then calculated work per human (i.e. total work divided by the team size) to measure the productivity of the team. For this analysis, we removed the events associated with the bot accounts. Teams were then grouped according to size. We considered teams as small if they had two or three members, medium if they had between four and six contributors, and large if they had seven or more members. Figure 5.1 illustrates differences between productivity of human-bot teams versus human teams. Human-bot teams are more productive regardless of their team sizes. A Mann-Whitney U test shows that the *p-value* is less than 10^{-2} for team productivity in Figure 5.1. Therefore, as measured by the generation of GitHub events, the productivity of human-bot teams is significantly higher than the productivity of human teams. This indicates that the bots are affecting the work process by modifying the type of events executed by human team members rather than reducing their overall quantity.

In order to understand the role of bots in teams we looked at the events they perform. Figure 5.2 shows that a high proportion of the bot generated events are issue comments. This indicates that the bots serve the important purpose of documenting the activity of the human team members.

Figure 5.3 shows the event distribution of teams. Human-bot teams not only perform more push events but also have more issue comments. Overall this suggests that the bots function as facilitators for the software release process (through push events) while documenting the code changes through comment events.

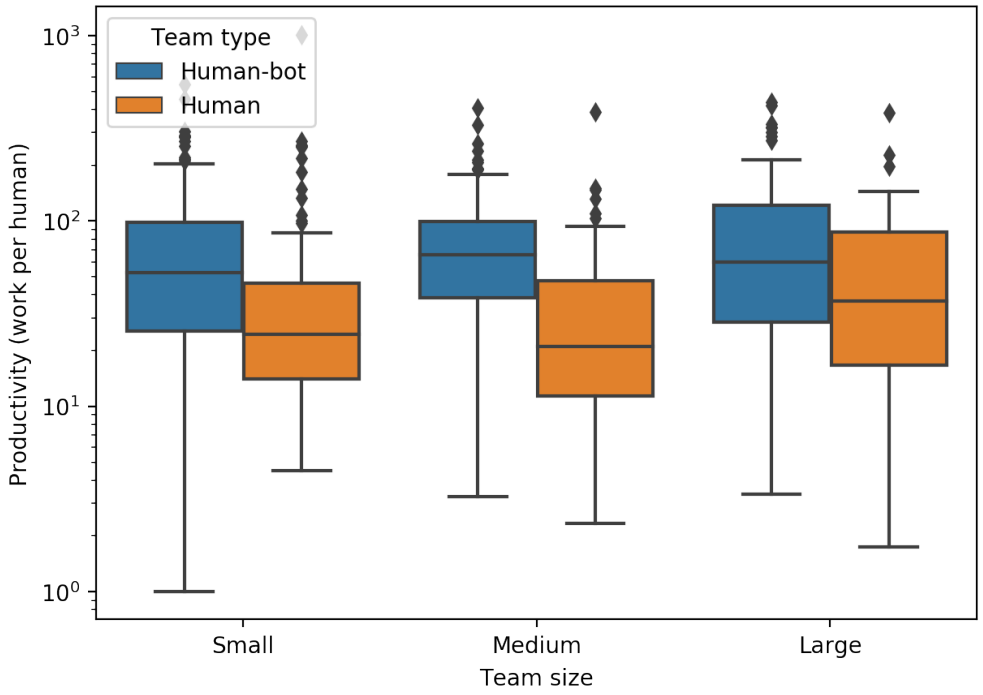


Figure 5.1: Human-bot teams are more productive than human only ones, as measured by generation of work events.

Work Centralization

We aim to understand how distribution of work among team members is associated with team productivity. To quantify work centralization among team members, we computed the Gini coefficient. The Gini coefficient is one of the most commonly-used metrics to capture inequality of income distribution in economics [13], but has been adapted to study inequalities more generally. There are different ways to compute the Gini Coefficient. We computed the Gini coefficient using equation 5.1, which is Relative Mean Absolute Difference; where x_i is the work of person i, and there are n persons.

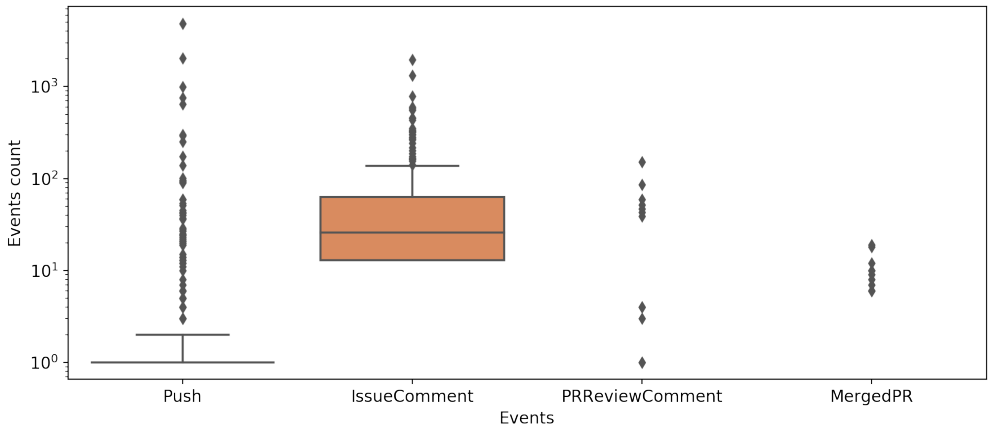


Figure 5.2: The average number of events performed by each bot, broken down by event type.

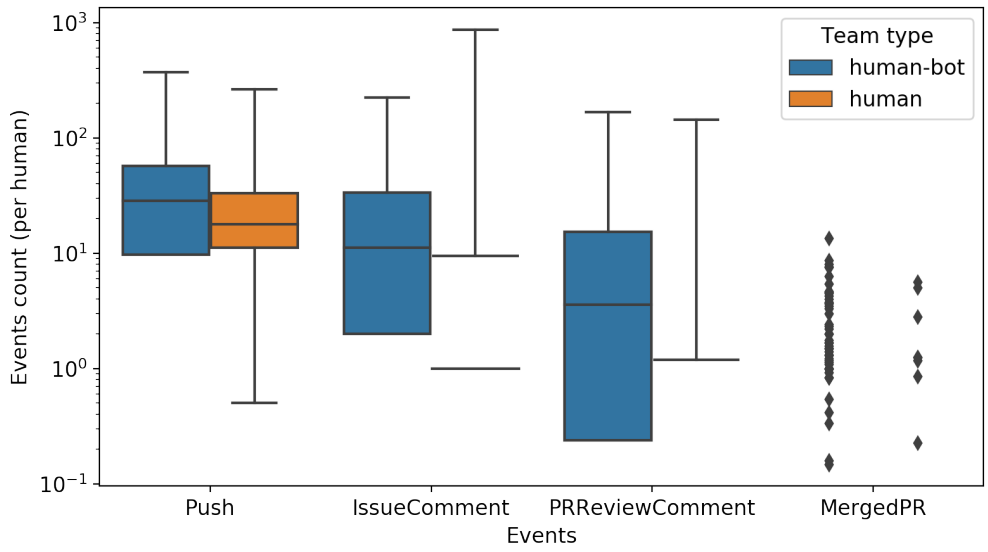


Figure 5.3: Difference between events that teams with and without bots perform.

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2 \sum_{i=1}^{n} \sum_{j=1}^{n} x_j} \tag{5.1}$$

The Gini coefficient, represented in the range of $[0, 1]$, is 0 if all team members perform an equal amount of work and the coefficient increases with the increase of the skewness in the work distri-

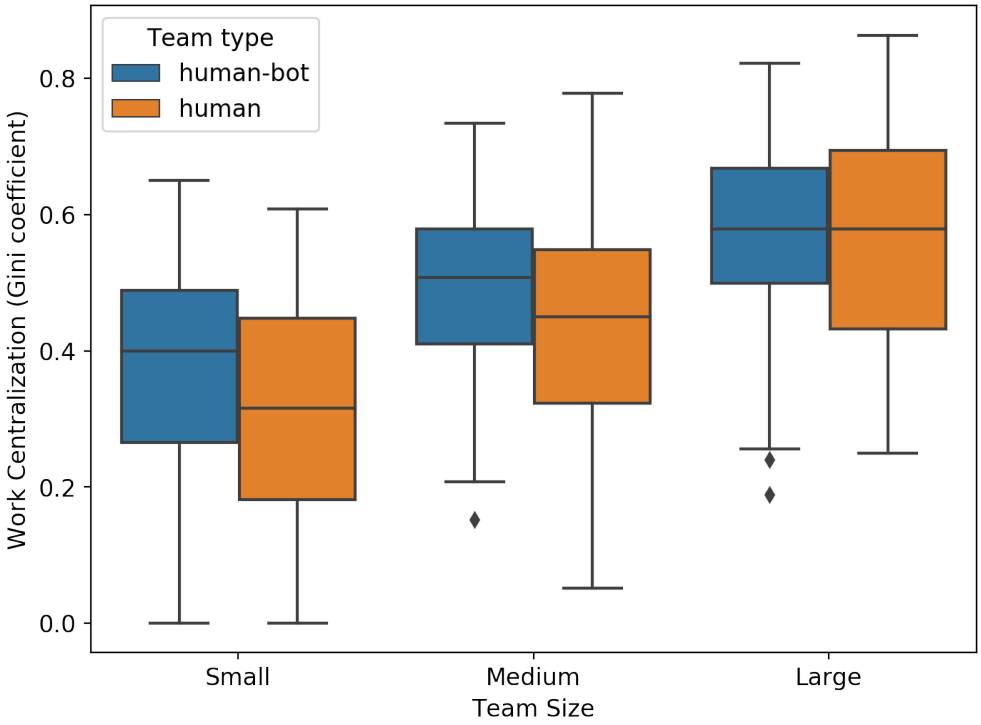


Figure 5.4: Work centralization of human-bot teams versus human teams

bution among team members.

Figure 5.4 illustrates work centralization of teams. When examining the distribution of work across human team members, we observe differences between human-bot teams and human teams, and that difference is consistent for small and medium team sizes. That is, human-bot teams have a higher Gini than human teams in small and medium team sizes. Mann-Whitney U test shows that this difference is significant (p-value < 0.003) for small and medium teams. The presence of bots is related to greater disparity in work distribution among contributors. This indicates that the bots are not serving the function of spreading the work more evenly across the developers. However, for large teams, there is no significant difference between work centralization of teams with or without bots.

Survival Analysis of Issue Closure

GitHub repositories include an issue handling infrastructure. Within this, developers and end users can report a bug, provide a feature request, etc. An issue is opened when further discussion is needed. This includes requesting more information and responding to questions posed in the issue. GitHub users engage in discussion around the issue by commenting on it (i.e., creating issue comments linked to the issue in question). The issue is closed when the problem or request is resolved or otherwise addressed. Issue closure rates thus reflect the speed with which teams resolve problems and can be used to assess issue support quality, an important indicator for process performance [36]. We performed survival analysis on issues to evaluate the issue support quality provided by teams in our sample of software repositories.

The repositories we studied were active at the time of data collection. As a result, it was likely that they had issues that were open when the data was collected but these issues may have been closed after our data collection. Therefore, these issues are considered to be censored. Survival analysis is designed to make use of censored data to make inferences. Survival analysis is based on the expected time duration until the event of interest happens.

Consider T to be a random variable that represents the time that it takes for an issue to be closed in a specific repository. Probability density function (pdf) of $f(t)$ and cumulative distribution function (cdf) of $F(t)$ can be used to characterize the distribution of random variable T; where $f(t)$ is the number of issues closed at time t and $F(t)$ is the number of issues closed until time t. The cdf function can be defined as $F(t) = P(T < t)$, which is the probability that the issue is closed in t days. $F(t)$ gives us the proportion of the issues that are closed in less than t days. Survival function $S(t)$ (equation 5.2) gives us the probability that the issue has not been closed until time t. In other

words, $S(t)$ gives us the proportion of the issues that are closed after t days.

$$S(t) = 1 - F(t) = P(T \geq t) \tag{5.2}$$

Survival function S can be inferred from function f but since f is not available, we have to estimate S from data. We used a non-parametric method called the Kaplan-Meier to estimate the survival curve [41]. Equation 5.3 is how survival function estimated in the Kaplan-Meier method.

$$\hat{S}(t) = \prod_{i:t_i \leq t} \frac{n_i - d_i}{n_i} \tag{5.3}$$

where d_i is the number of issues closed at time t_i and n_i is the number of issues still open just prior to time t_i.

Among the 238 teams that had at least five issues, 126 of them had bots and 112 of them did not have bots. Figure 5.5 compares the issue closure response time in human-bot teams vs. human teams. Human-bot teams have similar issue closure rate across different team sizes. In small and medium size teams, human-bot teams are slower at closing their issues.

To further investigate the reason for slower issue support in human-bot teams, we looked at the median number of issues. Our results show that human-bot teams have significantly higher number of issues compared to human teams. Table 5.2 shows the average number of issues for each team type across all team sizes. The reason for having higher response time to issues could be having more issues to address.

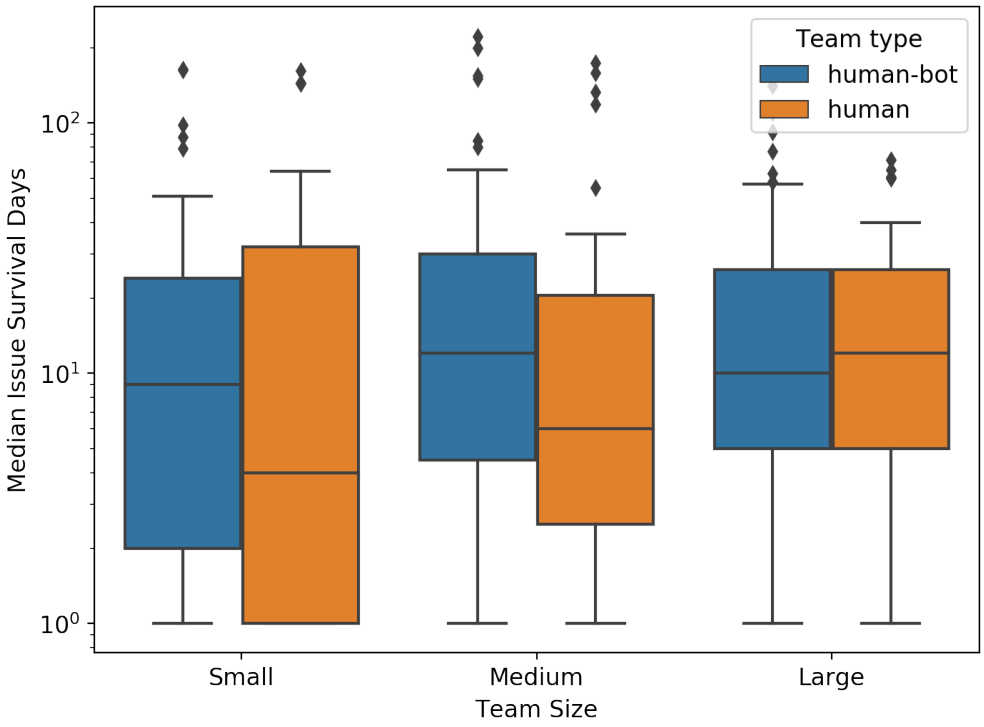


Figure 5.5: Issue closure of human-bot teams versus human teams

Work Style Clusters

There is large variability in event distributions of different users and teams on GitHub [70]. To summarize activity patterns of teams, we clustered them and discovered different work styles for these teams. For each team i, we extracted four features, which are the proportion of each work event performed by human members of the team, f_{ij},

$$f_{ij} = \frac{w_{ij}}{T_i} \tag{5.4}$$

where $j \in \{Push, IssueComment, PRReviewComment, MergedPR\}$, w_{ij} is the number of work event j in the team i, and T_i is the total amount of work in that team. Based on these features,

Table 5.2: Average number of issues and median of issue survival days for different team types and team sizes

Team size	Team type	# of teams	Avg. # of issues	Median of issue survival days median
Small	Human	39	42	3
	Human-bot	65	48	8
Medium	Human	43	53	5
	Human-bot	55	103	11
Large	Human	30	233	11
	Human-bot	71	432	9

we clustered teams into 3 clusters using k-means algorithm. k-means is a representative-based clustering algorithm that relies on distance to cluster data points.

Software development teams employ different work styles on GitHub. In order to discover various work styles on GitHub, we clustered teams using the k-means clustering algorithm applied to the proportion of different types of work events. The number of teams in each of the three work style clusters is provided in Table 5.3 and the cluster centroids are plotted in Figure 5.6A.

Based on our clustering analyses, these GitHub teams could be characterized by the relative distribution of events performed by the team and divided into three groups: toilers, communicators, and collaborators. *Toilers* produce a higher proportion of push events compared to *communicators* and *collaborators* but have limited communication taking place in GitHub. This may reflect a failure to engage in explicit coordination but it is also possible that these teams use alternative communication channels. Although *toilers* focus on code contributions, they generally do not accept external contributions, unlike the other two work style groups. *Communicators* produce a higher proportion of comment events, and *collaborators* produce a higher proportion of pushes and merged pull requests.

We studied the relative proportion of bot-human teams that fall into each of these work style cat-

Table 5.3: Number of bot-human teams in each work style.

Work style	# teams	# human-bot teams
Toilers	224	148 (40%)
Communicators	61	107 (64%)
Collaborators	19	49 (72%)

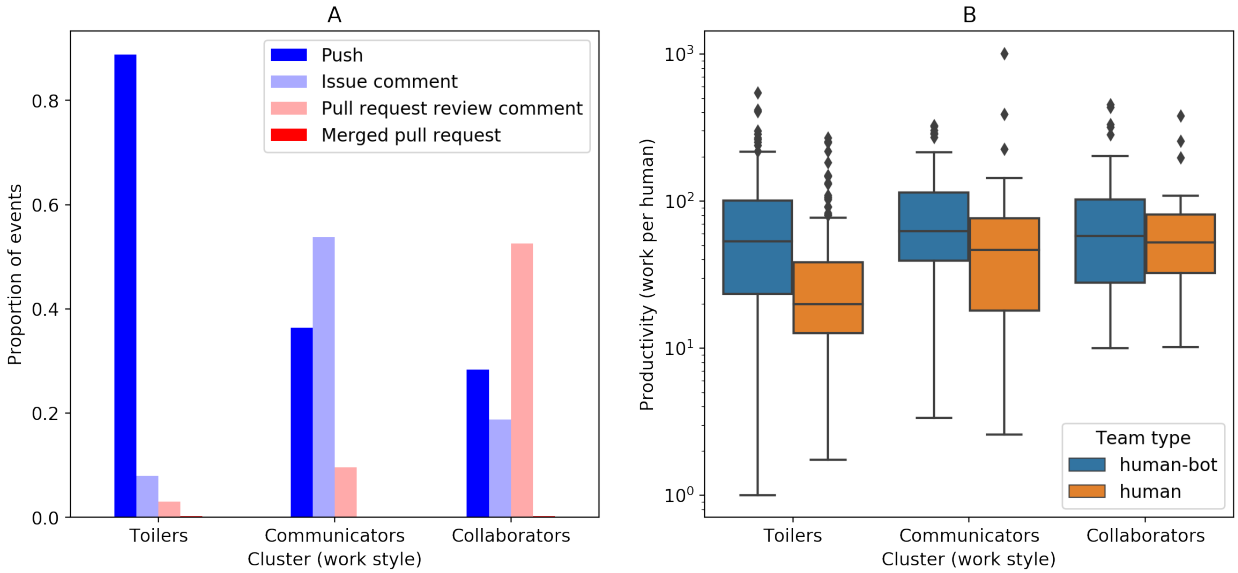


Figure 5.6: Clustering analysis of the relative event type distributions of toilers, communicators, and collaborators (A) and productivity of teams with and without bots, separated by team type (B)

egories. Bot-human teams comprise a higher percentage of the collaborator and communicator teams than toiler teams. This is unsurprising given that a key difference between the toilers vs. the other team types is the relative proportion of issue comments. Across work styles, human-bot teams exhibit higher levels of productivity relative to teams that do not make use of bots (Figure 5.6B).

Impact of Bots on the Outcome of Teams

The presence of bots within a team appears to change the nature of the work that the software engineers perform in the following ways:

- According to our productivity measure (work-events per human), bot-human teams are more productive than human only teams.
- Bots are generating a high number of issue comment events. Based on our topic analysis, these comments can be divided in three categories: actuarial, greeting/guiding, and efficiency.
- Bots are not serving the purpose of distributing the workload across the team, since the bot-human teams have a high work centralization.
- The effect that the bots have on the issue closure process is complex; the presence of bots does not yield a clear speedup in issue closure. More issues are documented in bot-human teams as compared to human only teams.
- According to our team taxonomy, bot-human teams are more likely to be communicators and collaborators rather than toilers.

Interestingly, bots appear to be serving as an aid to both taskwork (by performing build automation) and teamwork. Specifically, in addition to automating repetitive tasks, bots supported team knowledge building and shared knowledge structures among human team members. Bots also interacted directly with contributors to provide guidance for working on a project. These findings support Fiore and Wiltshire theorizing about the need to disentangle teamwork from taskwork in human-machine teams [22].

Although there are limitations in how much can be inferred from a statistical analysis of event data without interviewing programmers or analyzing code, we believe that event analysis is an accurate reflection of programmer interactions within social coding platforms.

Encouragingly, the programmers' production of work-related events is higher in bot-human teams. The presence of bots modifies how human team members report and handle problems. Teams with bots are more diligent about documenting issues, leading to more issues in total being reported, with bots generating a large numbers of issue comments.

Though many of the bots are relatively simple, there is evidence that bots aid both taskwork and teamwork, removing the tedium of build management and documenting version changes. However, it is possible that bots also contribute to information overload for the human users by generating superfluous notifications of trivial code changes. This may reduce the quality of experience for software engineers immersed in the coding process. We believe that in order to enjoy the productivity benefits of bots, without being overloaded by messages, programmers must employ an intelligent filtering strategy to eliminate unwanted notifications. A fertile area for future research is to transition these systems from simple scripts to bots with *artificial social intelligence* who are more capable at offering decision-making support.

Sequence Group Comparison

This section describes the results of our analysis procedure for studying the differences between event sequences of human-bot versus human-only teams on GitHub.

Dataset

We used the same subset of GitHub data that explained in previous section for doing our sequence analysis. We created the team event sequences using all events, sorted by time, performed within a year of repository creation. Figure 5.7 shows the distribution of sequence lengths for human-bot teams and the downsampled group of human teams. Human only teams have shorter average sequence lengths; the variance of the sequence lengths is also smaller in human teams. As demonstrated in Section 5, sequence length alone is not a predictor of the team type that generated that sequence.

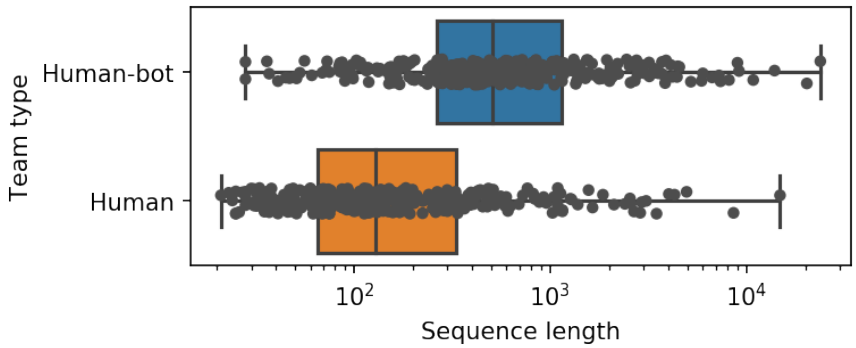


Figure 5.7: Human-bot teams have longer sequences on average; however this is not a strong predictor of team type.

Classifying Team Type

In our research, we need to predict which group each sequence of repository events belongs to. Creating a predictive model requires discriminative features. The approaches discussed in Section 3 and 3 reveal whether the two groups of sequences are different from each other. For a dataset, if we infer, from the above methods, that two groups are distinct, a model can be trained to predict the group from the sequences themselves.

One of the challenges of the sequence classification problem is that sequences can vary in length. To use classic machine learning algorithms, we need equal length real-valued vectors. One way to convert sequences to vectors is using the vectorization methods discussed in Section 3. We use a k-gram representation for vectorizing the sequences in the dataset before employing a support vector machine (SVM) model to construct a predictive model.

Our second approach for sequence classification is using deep neural network models. Deep learning models, unlike classic algorithms such as SVM, do not always require direct vectorization. These models can have an embedding layer that converts the sequences to real-valued vectors. We use a Long Short Term Memory (LSTM) model to learn and classify representations for sequences. LSTMs have achieved notable success in natural language processing tasks such as machine translation [80]. Additionally, we test 1-Dimensional Convolutional Neural Networks (1D CNNs) which excel at learning the spatial structure in input data. CNNs are also used in many sequence models such as sentence classification and language translation [43, 39].

Our baseline is a simple Logistic Regression model. This model considers the length of the input sequence as the only feature for the classification task.

We trained and tested three different machine learning models for the team type prediction task.

SVM

We used the k-gram representation of sequences and vectorized them using *TF-ISF* model. Then, we trained a SVM model using the implementation available in *scikit-learn* machine learning library.

LSTM

We used the LSTM recurrent neural network models implemented in *Keras* deep learning library [12]. Each event was mapped onto a 32 length real-valued vector. Only the first 1000 events of each team were considered; long sequences were truncated and short sequences were zero padded. As discussed in Section 5, the majority of the teams generate less than 1000 events. The first layer of our neural network is an embedded layer that uses length 32 vectors to represent each event. The next layer is an LSTM layer with 100 neurons. Finally, we added a dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes: human team or human-bot team. Because it is a binary classification problem, we used log loss as the loss function. The efficient ADAM algorithm is used for optimization.

CNN+LSTM

Our CNN+LSTM model used the same architecture as our LSTM model, but with a 1-dimensional CNN layer and a max pooling layer before the LSTM layer.

For the neural network models we hold out 20% of data for testing and trained the models on rest of the data. 10% of the training data was used as validation set for tuning parameters. For the SVM and Logistic Regression models we used 5-fold cross-validation. Figure 5.8 shows the performance of the different classification models. Our neural network models achieved the highest F1 scores of 0.79 (precision=0.77, recall=0.82) and 0.77 (precision=0.75, recall=0.80) for CNN+LSTM and LSTM, respectively; while the SVM model achieved F1 score of 0.74 (precision=0.66, recall=0.82). Both models have significantly higher F1 score than our baseline model with F1 score of 0.54 (precision=0.77, recall=0.42), showing that predicting the type of teams is a non-trivial task and that the sequence of events is helpful in distinguishing team types.

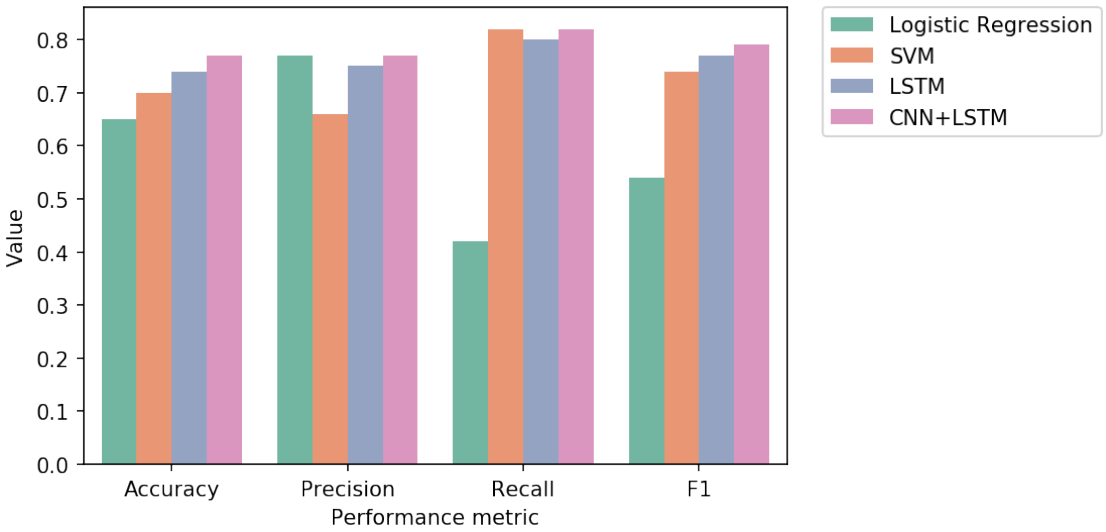


Figure 5.8: Classification performance of the different models (logistic regression, SVM, LSTM, CNN+LSTM) at recognizing human-bot versus human only team sequences

Sequence Differences

A k-gram representation of GitHub team event sequences was created using different window sizes $w \in \{2, 3, 4, 5\}$. Then *TF-ISF* vectors were extracted for these sequences.

To compare the sequences of human teams with the sequences of human-bot teams, we calculated the silhouette score between human-bot team vectors and the downsampled set of vectors of human teams. Figure 5.9 illustrates the amount of distinction between human teams versus human-bot team sequences for different window sizes. Positive values of the silhouette score show that these two groups of sequences are relatively distinct, although they are not completely separate.

The distinction between human and human-bot teams decreases as w increases, where w is the length of the window for constructing subsequences of each sequence. This occurs because when subsequences become longer, the number of shared subsequences between the sequences

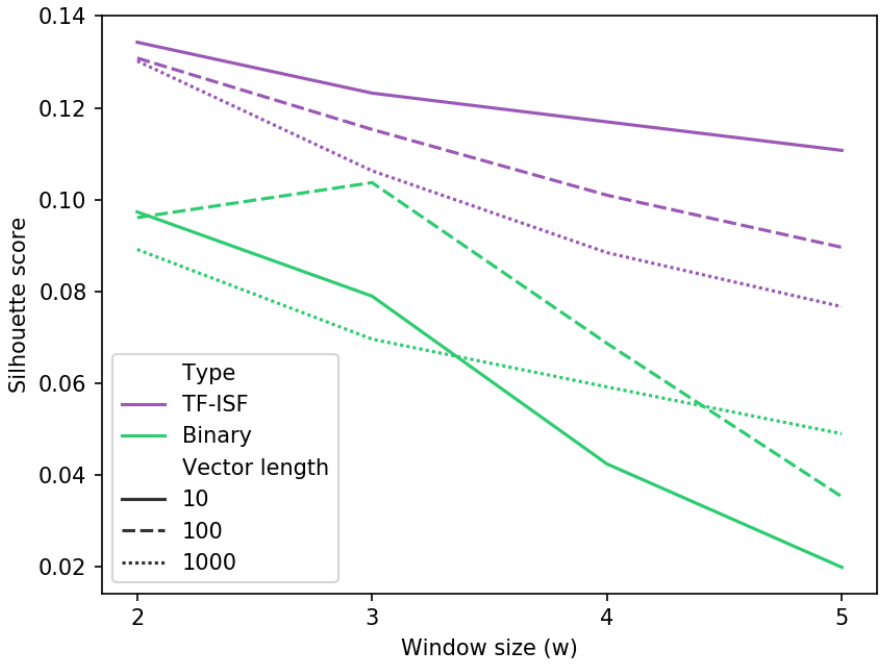


Figure 5.9: Human vs. human-bot teams silhouette score considering different vector lengths (line style) and different vectorization models (line color). The best method for detecting differences at all window sizes is TF-ISF with a vector length of 10. However even the binary vectorization model detects differences between the two groups of sequences.

decreases.

In our *TF-ISF* model, which creates real-valued vectors from an ordered list of subsequences, the size of the vectors is the number of unique subsequences in all sequences. Since there are many subsequences that are rare, there is an option to limit the size of the vector to only consider most frequent subsequences. We measured the silhouette score between human and human-bot teams considering vector length to be 10, 100, and 1000. Vector length x denotes that a vocabulary of subsequences is constructed that only considers the top x subsequences ordered by term frequency across the sequences. Figure 5.9 illustrates the impact of vector length on measuring the

distinction. The distinction between two groups increases when a shorter vector length is used. This means that the main difference between human only and human-bot sequences is in the most frequent subsequences.

We also calculated silhouette scores for the binary vectorization model. This model ignores the frequency of subsequences in order to understand if the distinction between human versus human-bot team sequences occurs because of the difference in frequencies or whether the subsequences themselves also differ. Green lines in Figure 5.9 correspond to the binary model. The results show that although using the binary model makes the groups less separated, this model still reveals the distinction between human only and human-bot teams. This indicates that it is not only the frequency of the subsequences that differs between human only versus human-bot teams but also that different subsequences exist in the these two groups.

To make sure that the observed differences are due to teamwork differences rather than the additional bot event activity, we also measured the silhouette score after removing bot activity from human-bot team sequences. We still observed positive, although slightly lower, silhouette scores between team sequences. This shows that using bots in GitHub teams not only affects the sequences of the teams but also changes the activities of the human members of the teams.

Matrix Profile Analysis

To understand what is different about human only team event sequences as compared to human-bot teams, we constructed distance matrices and matrix profiles for all sequences. For the distance calculation between two subsequences, we used Hamming distance due to its time efficiency.

Distance matrices for a randomly selected sequence are visualized in Figure 5.10 for different values of window sizes. For every subsequence, the distance matrix shows the position of all

similar subsequences. In the heat-map plots of Figure 5.10 similarity is represented by redness. So for every row of the distance matrix, red blocks demonstrate repeats of similar subsequences. For smaller window sizes, blocks are smaller but the similarity is higher. For a window size of 20, there are subsequences that perfectly match (zero distance).

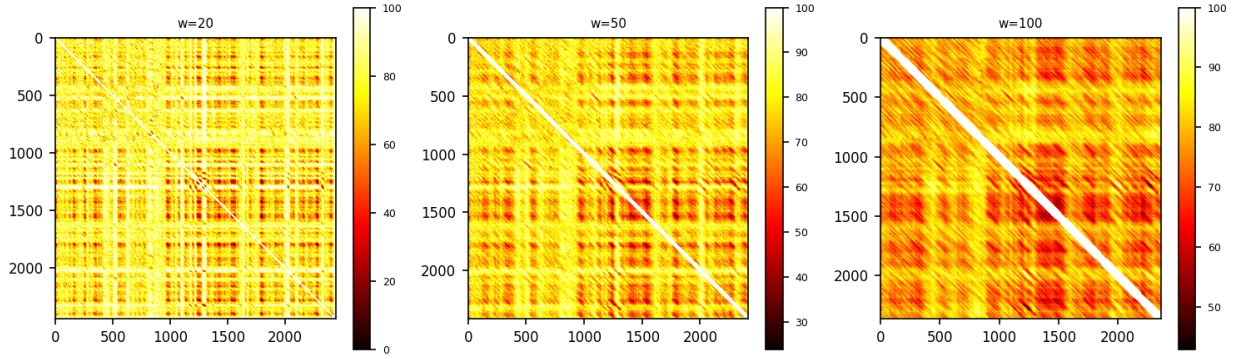


Figure 5.10: Distance matrix of a randomly selected sequence for different values of window size. For every subsequence, the distance matrix shows the position of all similar subsequences. Similarity is represented by redness; red blocks demonstrate repeats of similar subsequences. For smaller window sizes, the blocks are smaller but the similarity is higher. Note that heat maps are scaled to lie between 0 to 100 to make comparison easier.

From distance matrices of sequences in our dataset, we created their matrix profiles. Figure 5.11 shows the corresponding matrix profile of the sequence in Figure 5.10 for different window sizes. The trend of the matrix profile is independent of the window size, i.e. minimums and maximums occur at the same positions for different window sizes. Minimums of the matrix profile represent motifs of the sequence and maximums are related to discord or anomalies. Increasing the window sizes enlarges the distance between the most similar subsequences (e.g. distance between most similar subsequences is 40 for the window size of 100). For our remaining experiments, we choose $w = 20$ as it finds more similar subsequences.

Table 5.4 shows the summary of statistics of matrix profiles of human versus human-bot teams. The matrix profile consists of distances to the closest subsequence for every subsequence. The

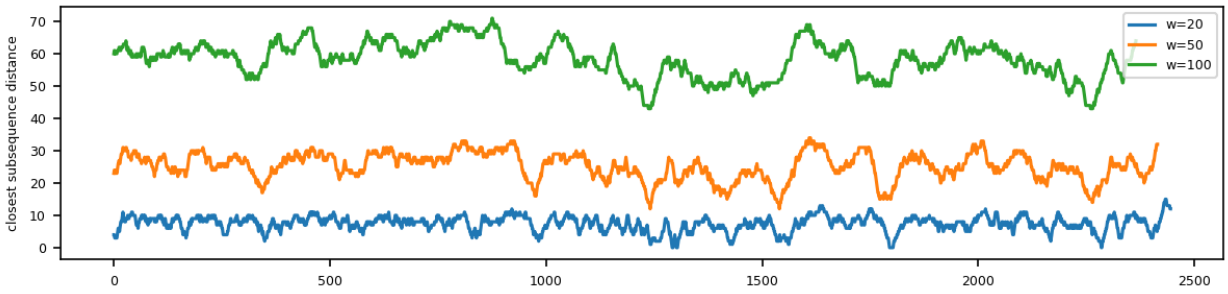


Figure 5.11: Matrix profile of the sequence in Figure 5.10. The trend of the matrix profile is independent of the window size. Increasing the window sizes enlarges the distance between the most similar subsequences.

higher average for average of matrix profile values for human-bot teams shows that human-bot teams subsequences are less similar to each other compared to human teams. That indicates that human teams have more repetitive groups of actions. A Mann-Whitney U test shows that the average matrix profile value is significantly different in human teams compared to human-bot teams ($p = 0.02$). The Mann-Whitney U test was chosen to test the null hypothesize because the data does not follow a normal distribution and a non-parametric statistical test is needed. Human-bot teams may have less repetitive sequences of human actions if the bots perform repetitive tasks, leaving fewer repetitive series of actions for humans to perform.

Yeh et al. [96] considered the variance of matrix profile to be representative of the complexity of its underlying time series. Although human-bot teams have a higher matrix profile variance, the Mann-Whitney U test shows that this difference is not significant. Therefore, we cannot conclude that human-bot teams have more complex sequences.

The absolute minimum value in the matrix profile is related to the best motif of the sequence and the maximum value shows the anomaly [96]. There is no significant difference between minimum or maximum values of matrix profiles in human teams versus human-bot teams ($p = 0.1$).

Table 5.4: Human versus human-bot teams matrix profile summary. The human-bot and human columns show the average values across all matrix profiles for each team type. The last column shows the p-value of the Mann-Whitney U test between team types.

Metric	Human-bot	Human	p-value
Variance	4.8	4.6	0.1
Average	6.9	6.1	0.02
Minimum	2.0	2.2	0.1
Maximum	11.8	11.0	0.1

Figure 5.12 shows the profiles of human and human-bot team sequences. We created matrix profiles for full length team sequences but for better visualization, we plotted only 1000 first positions of matrix profiles in Figure 5.12. Since 97% of teams have sequences shorter than 1000, this visualization contains the full length matrix profile of the majority of the teams. The matrix profile of human-bot teams clearly have higher values compared to human teams, indicating the higher novelty of sequences in human-bot teams.

Figure 5.12 shows that although human teams have lower matrix profile values at the beginning due to simpler, repetitive groups of actions, as the human team projects progress, the value and fluctuation of their matrix profiles increases which indicates that they are becoming more complex. However, human-bot teams seem to be complex from the beginning, and they maintain the complexity of their sequences as the projects progress.

Contrast Motif Discovery

This section describes our work on discovering contrast motifs of human-bot and human-only teams. For this section we introduce a new dataset that we collected which is larger and more recent. In addition, a more advanced approach, which is based on machine learning techniques, is exploited to detect bots in this dataset. First, we give a brief introduction about the dataset and

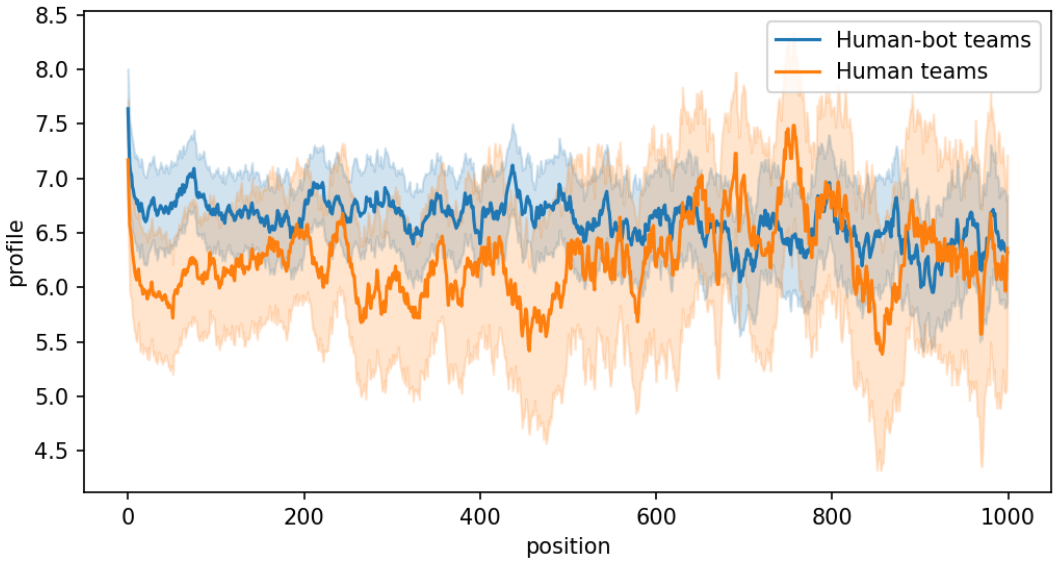


Figure 5.12: Aggregate matrix profile for human-bot teams vs. human only teams. The human-bot teams have higher values compared to human teams, indicating the higher novelty of sequences in human-bot teams.

then we present the results of applying the contrast discovery method on this dataset.

Dataset

We obtained the most recent data month available from the GHTorrent[2] dataset (June 2019). GHTorrent is an offline mirror of GitHub public event data.

Bot Detection

Existing research on software bots either manually detects bots in a small set of repositories or investigates the role of several well-known bots. As the variety and use of bots increases, large-

[2]https://ghtorrent.org/

scale automatic bot detection becomes necessary. In a recent study, Golzadeh et al. used repetitive comments to detect bots in one specific domain (e.g. pull request) [24]. However, there are many bots that do not make comments. To address this issue, we implemented a more comprehensive approach to automatically detect GitHub bots in a large scale. We exploited various aspects of an account, in addition to the comment similarity, to identify whether they are bots. We manually labeled hundreds of accounts and used it as training data for a classifier to detect bots.

Labeling

4815 accounts with "bot" in their username were found. We investigated 612 randomly selected accounts and labeled them with 1 if their GitHub profile or our web search indicated that they are bots and 0 otherwise. In these 612 GitHub accounts, we discovered 424 bots which correspond to about 70% of the accounts.

Features

The following features were used to identify bots:

- Comment similarity: Average cosine similarity of comments (Cosine similarity of -1 denotes no comments to compare).
- Organization owned: If the bot belonged to an organization at any moment (0 for no and 1 for yes)
- Unique event types: The number of types of events the bot account contributed to (IssueCommentEvent, IssuesEvent, PullRequest, PullRequestReviewComment, CommitCommentEvent, PushEvent).

- ”bot” placement: The placement of the string “bot” in the name (beginning, middle, or end).

Bot Detection Classifier

We trained and tested various classifiers on the labeled dataset including *Logistic Regression*, *Random Forest*, and *Gradient Descent Boosting*. We evaluated the classifiers using cross validation to ensure the model is not overfitted. Since the dataset is imbalanced, we used stratified k-fold cross validator to preserve the percentage of samples for each class. k was set to 5 in our experiments.

Event Dataset

More than 46 million events were found during the month of June 2019. These events were executed by more than 420,000 active users on over 285,000 repositories. We call a user or repository active if they perform at least one push or pull request.

We considered a GitHub user a member of a team if they contributed at least one push or one pull request event in the repository. In this dataset, we removed events related to bots and non-members. Then, we extracted teams which are defined as repositories with two or more members. There are two types of teams: (1) Human-bot teams: teams that are using automated accounts. (2) Human-only teams: teams that are not using any automated services.

Team Sequences

We ordered the events corresponding to each team based on the time that event is performed and created event sequences for each team.

Table 5.5 shows the percent of different event types in our dataset. *Push* and *pull request* are the

Table 5.5: Proportion of various events in our dataset.

Event Type	**Proportion**
PushEvent	0.441458
PullRequestEvent	0.156276
CreateEvent	0.113361
IssueCommentEvent	0.091894
PullRequestReviewCommentEvent	0.056228
DeleteEvent	0.047691
IssuesEvent	0.040053
WatchEvent	0.024057
ForkEvent	0.013839
ReleaseEvent	0.005697
GollumEvent	0.003802
MemberEvent	0.003178
CommitCommentEvent	0.001672
PublicEvent	0.000796

most frequent events. These two events are used to submit changes made on a local repository to the main repository.

Since events with low frequency will not be in final motifs, we removed the least frequent events to make the algorithm run faster. Additionally, we combined *issue event* and *issue comment* as they are referring to the same kind of activity. Our final list contained six event types including: *push*, *pull request*, *issue*, *pull request review comment*, *create*, and *delete*.

After transforming sequences, we removed sequences shorter than five because the sequence length should be longer than window length and we wanted to test window sizes longer than two. The reason for using a higher minimum sequence length is that as the threshold goes higher, more and more teams are omitted from the data set. We chose the threshold to be five as it removes very short sequences without discarding a large number of teams.

Table 5.6: Median value of event frequencies before and after sampling.

Event Type	H-B	Original H-only	Downsampled H-only
PushEvent	11.0	9.0	12.0
PullRequestEvent	9.0	4.0	9.0
IssuesEvent	1.0	0.0	1.0
IssueCommentEvent	6.0	0.0	5.0
CreateEvent	2.0	3.0	2.0
DeleteEvent	1.0	1.0	1.0

Team Sampling

Our dataset is imbalanced with 201,204 human-only teams and 4,205 human-bot teams. For each team, we create a vector that contains the frequency of the different event types in that team. For each human-bot team, we found the most similar human-only team with respect to their event frequency vector and downsampled the human teams to 4,205 teams corresponding to the 4,205 human-bot teams.

Table 5.6 shows the median event frequencies. Before downsampling, the median event frequencies are much lower for human-only teams. However, after downsampling human-only teams, their medians move closer together.

Contrast Motifs

We ran the Contrast Motif Discovery tool on the sequences we created for GitHub teams. We tested window sizes from 2 to 5. As the window size becomes larger, repetitive patterns emerge within motifs. Window size 4 was the largest window size with the least repetition. Therefore, in this dissertation, we present our results for a window size of 4.

Figure 5 shows the graph representation of the contrast motifs for each team type. We observe

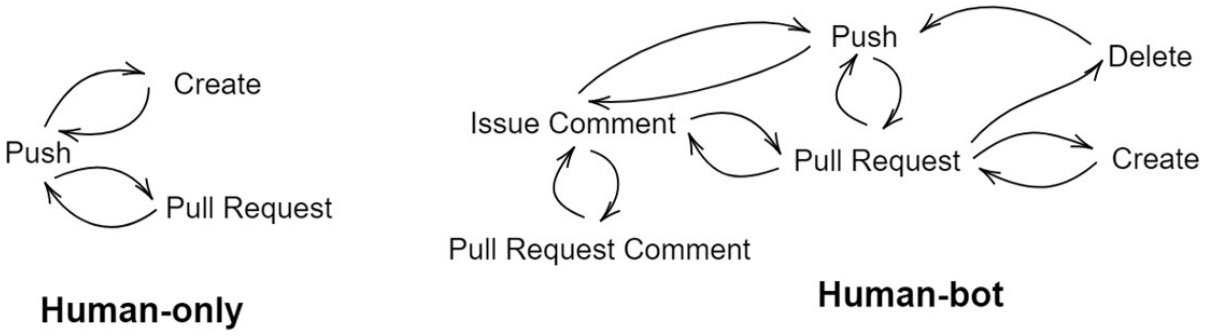


Figure 5.13: Motif graph for human-only and human-bot teams.

a more complex graph structure for human-bot teams while human-only teams have a simpler structure. This observation means that the repetitive patterns in human-bot teams are more complex than human-only teams. Note that this does not indicate that these patterns do not occur in human-only teams, it means that these patterns are significantly more frequent in human-bot teams.

Another observation about the graphs shown in Figure 5 is that in human-bot teams, issue comments occur before and after each event type. This does not mean that human-bot teams make more issue comments since we selected repositories in a way that have similar event frequencies. We hypothesize that this pattern occurs because in human-bot teams issue comments are intermixed with other events rather than being clustered together. To confirm this hypothesize, we examined the length of consecutive issue comment events in human-only and human-bot teams. First, we measured the average length of consecutive issue comment events in all sequences. Then, we ran a *Mann-Whitney U-test* analysis to investigate whether there is a difference between two types of teams. Our *U-test* indicates that the average length of consecutive issue comment events in human-only teams is higher than this value in human-bot teams and the difference is statistically significant (*p-value*$= 10^{-7}$). This finding means that bots force human members of the team

discuss issue between different stages of their work.

In summary, in human-bot teams, contrast motifs are more complex and issue comments are scattered throughout the event sequences while in human-only teams there are more simple contrast motifs and issue comments tend to be clustered together.

Chapter Summary

In this chapter, we studied human-bot teams on GitHub and compared them with human-only teams. We aimed to enhance our understanding of the differences between two groups of teams by comparing their performance, work distribution and issue support quality. Our analyses show that human-bot teams are different from human-only teams in terms of work processes and outcome.

CHAPTER 6: MINECRAFT

In this chapter, we discuss Minecraft action sequences and the results of applying our sequence mining approaches in this domain. We also find contrast motifs of Minecraft players in this chapter to compare highly collaborative players with hardly collaborative ones.

Dataset

Our study uses a dataset collected by the Heapcraft project across multiple servers [58]. The dataset contains two months of data from 45 players, forming 14 person-days worth of active gameplay. The benefit of this dataset is that it provides ground truth Minecraft actions for collections of raw events. At random intervals, players were asked to specify the high-level actions they are performing: explore, mine, build, and fight. The four action types used in their data collection were inspired by player types in Bartle's study: killers, explorers, achievers, and socializers [6]. For the fight, explore, mine, and build actions there are 37, 124, 186, and 297 data points respectively which creates a dataset of size 644.

Several of the events were excluded by [58] from the event log due to low frequency, correlation to other events, and redundancy. Moreover, move, sprint and sneak events were transformed to their corresponding distance or duration. We followed the event cleaning procedure presented by [58] except move, sprint and sneak events were also removed as distance and duration cannot be easily converted to symbols in sequences.

The original study considered the duration of each action to be two minutes centered around the time of response received. These two-minute intervals (labeled with high level actions) were used to construct our action sequence dataset. The sequence dataset of players was created by consid-

ering all the events performed by players during the data collection period. We created the event sequences by assigning a symbol to each Minecraft event and creating an ordered list of symbols for each data point. Since our method relies solely on the order of events rather than their frequencies, consecutive repetitive events are replaced by one event. For example, *aaabbcccd* is transformed to *abcd*.

Minecraft Action Sequence Comparison

In this section, we present the results of applying our sequence comparison approach to Minecraft action sequences.

Minecraft Action Classification

Similar to GitHub sequence classification, we classified Minecraft actions using three different classifiers (i.e. SVM, LSTM, CNN+LSTM) and a baseline (i.e. Logistic Regression based on sequence length). Figure 6.1 demonstrates the performance of the classifiers. On this dataset, SVM performed better than other classifiers with an F1 score of 0.61 (precision=0.64, recall=0.60). The SVM performance is slightly higher than CNN+LSTM with an F1 score of 0.60 (precision=0.71, recall=0.52). Note that this is multi-class classification problem with four possible outputs. Therefore, a classifier that randomly assigns label to sequences will achieve an accuracy of 0.25.

Minecraft Actions Silhouette Score Analysis

We vectorized Minecraft action sequences using *TF-ISF* approach as it is superior to the binary vectorization in detecting differences. Silhouette scores for Minecraft action vectors were calcu-

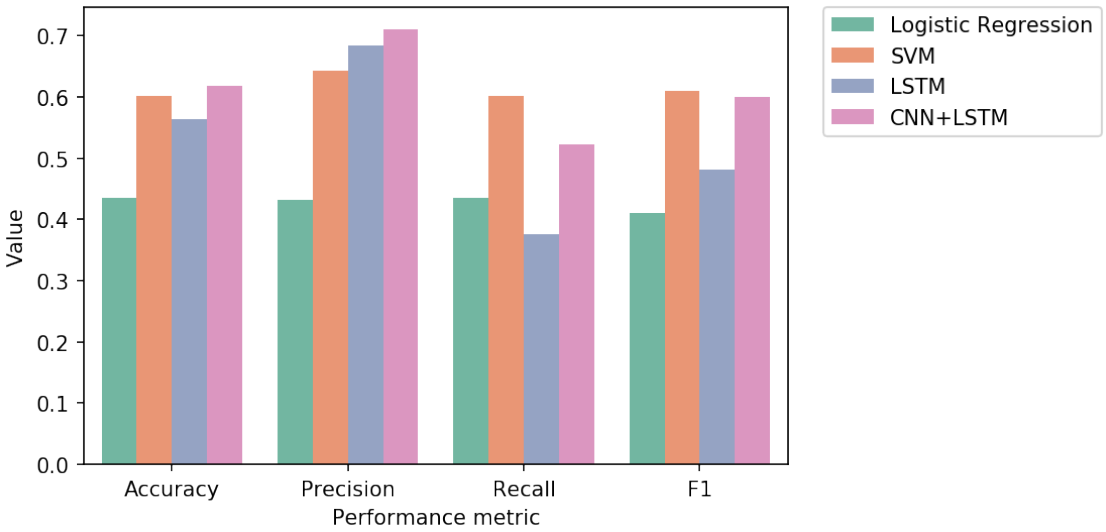


Figure 6.1: Classification performance of the different models at recognizing Minecraft action sequences.

lated using various vector sizes and window lengths. A window length of 2 and vector size of 50 delivered the highest silhouette score of 0.052. This silhouette score is less than silhouette score of GitHub sequences (0.13). This indicates that the classification of Minecraft actions is more challenging than identifying GitHub repositories with bots, hence the lower performance is unsurprising.

Minecraft Actions Matrix Profile Analysis

We created matrix profiles for all sequences, using a window size of 5. Figure 6.2 illustrates aggregate matrix profiles for each type of action. Since the matrix profiles of the build and mine actions have larger values at the beginning, we can infer that these two actions start with a subsequence that does not repeat later.

We calculated minimum, maximum, variance, and mean for matrix profiles of Minecraft action

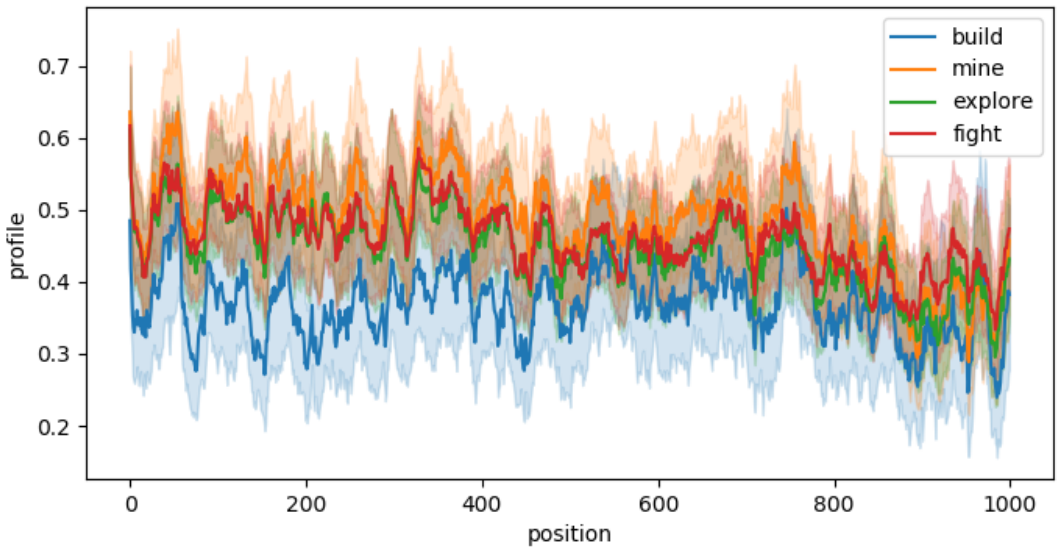


Figure 6.2: Aggregate matrix profile for Minecraft actions

sequences. Table 6.1 shows the average of these statistics for different actions. The variance of matrix profiles is similar across Minecraft actions. This means that sequences of different actions have similar level of complexity. Minimum, maximum, and mean are different in build compared to other three actions, and this difference is statistically significant (p-*value* < 0.05). The same holds true for mine action. The higher matrix profile mean in build and mine indicates that these two actions have less repetitive subsequences. According to Table 6.1, explore and fight have the lowest average minimum, suggesting that these two actions have extremely strong motifs. The highest average maximum in build action indicates that discords in build sequences are exceedingly distinct from the rest of the sequence.

Table 6.1: Matrix profile statistics of Minecraft actions.

Metric	Build	Explore	Fight	Mine
Variance	0.5	0.6	0.6	0.5
Mean	2.6	0.9	0.9	2.1
Minimum	1.9	0.3	0.3	1.5
Maximum	3.5	2.6	2.7	3.2

Minecraft Action Contrast Motifs

This section describes the application of our proposed algorithm for analyzing Minecraft action sequences. From the HeapCraft dataset, we extracted sequences labeled with the ground truth action. Each action type is considered to be a group; sequences labeled as that action belong to that group. Therefore, there are four groups of sequences for actions: *fight (f), explore (e), mine (m),* and *build (b)*.

Contrast Motif Distances

We ran our contrast motif finding algorithm for window sizes ranging from 3 to 9. The number of sequences larger than the window size dramatically decreases by increasing the window size. In order to avoid discarding short sequences, a window size of 5 was selected. Using this window size, 567 sequences were considered (34, 102, 171, and 260 for fight, explore, mine, and build, respectively).

The c parameter in Algorithm 2 is a user-specified input determining the desired number of motifs. This parameter has been set experimentally in our analysis. We started from a low value for c parameter and then we increased this parameter until the number of contrast motifs in each group is less than the c parameter.

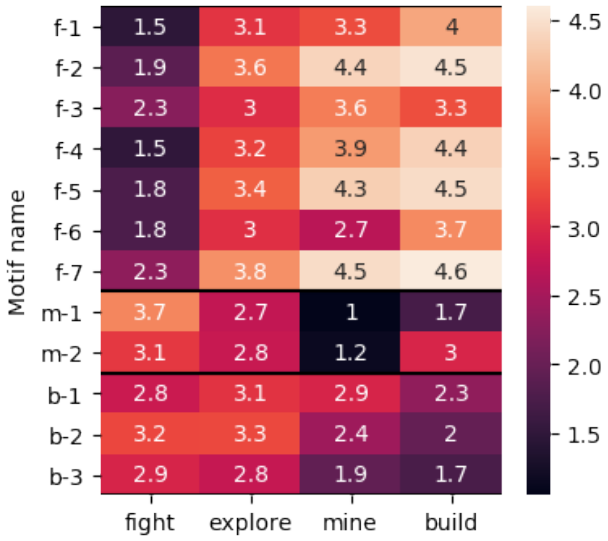


Figure 6.3: The average distance between motifs and sequences of actions. Rows represent motifs, and columns denote the action labels. For example, row **f-1** and column **fight** shows the average distance between motif **f-1** and **fight** sequences. Motif names are comprised of the action symbol (**f**, **m**, and **b** for fight, mine and build, respectively) and an ordinal number. Darker colors on the heatmap denote a lower distance between the motif and sequences of that action.

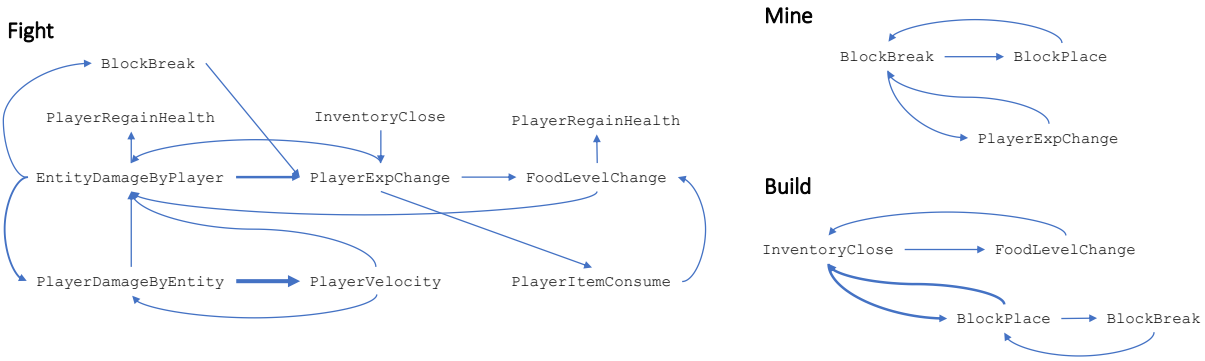


Figure 6.4: Contrast motifs of different actions represented in directed graphs. Nodes are events, and there is an edge between two events if they appeared consecutively in at least one motif. The thickness of edges represents the number of times that relationship was observed in the motif set.

Running the algorithm on the HeapCraft action sequence data with window size 5 results in 7, 2, and 3 contrast motifs for fight, mine, and build actions, respectively. Figure 6.3 shows the average

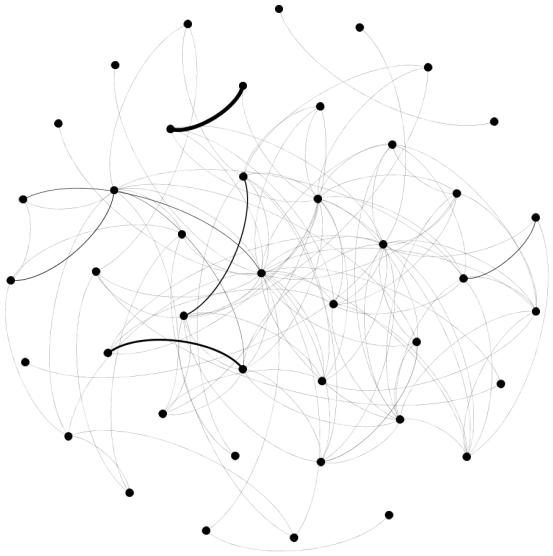


Figure 6.5: Collaboration graph between players. Nodes are players, edges show collaboration, and the thickness of the edge represents the duration of the collaboration.

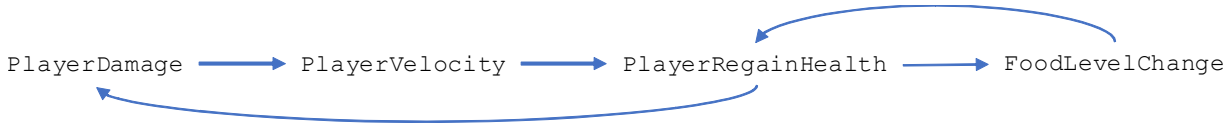


Figure 6.6: The highly collaborative players contrast motif. This motif is similar to the fight motif showing that fighting is the action shared amongst highly collaborative players and fighting is what distinguishes these collaborative players from less collaborative players.

distance between motifs and sequences of different actions. This heatmap illustrates that motifs of an action are similar to sequences of the same action class and distant from other action classes.

More contrast motifs were discovered in the fight action compared to mine and build. This may occur because there is more variety in player fighting styles compared to other actions. Unfortunately, no motif was found for the explore action. In the [58] study, which attempts to classify actions, the majority of classifier errors involve the explore action. As mentioned by the authors,

this could be due to the nature of the game or could be a result of their data collection procedure. In addition, Figure 6.3 shows that fight motifs are less likely to appear in other sequence actions. This shows that fighting is a more distinctive behavior.

The mining action has the most conservative motifs. Motifs of mine, with an average distance of 1.1, are closest to mine sequences as compared to fight and build with average distances of 1.9 and 2, respectively.

Although m-1 is closer to build sequences than some of the build motifs and can be considered a motif of build, it is not a contrast motif for the build action as it is closer in distance to mine sequences than build sequences.

Minecraft Events of Contrast Motifs

To have a visual representation of the motif set for each action, we constructed a graph for each action. Figure 6.4 illustrates the contrast motifs discovered for various actions in a graph format.

To ensure that our results are not entirely dependent on the window size parameter, we examined the motifs generated by other window sizes. It appears that smaller or larger window sizes create motifs that are subsequences or supersequences of the motifs identified for this window size.

Our approach provides detailed insight into player Minecraft actions. [58] conducted a frequency based analysis of Minecraft actions and found that the build action is highly correlated with the frequency of BlockPlace, but our approach reveals that other events such as InventoryClose and BlockBreak also occur in the building process. Unlike prior work, our approach also extracts their temporal ordering.

Players Contrast Motifs

There are various types of collaboration in Minecraft including building together, sharing building/farming infrastructure, mutual protection, and practice fights to hone skills [57]. Prior research on Minecraft quantifies collaboration as the duration of the time players spend in contact with each other; two active players are considered to be in contact if their distance is less than 15 blocks [57]. Figure 6.5 illustrates the collaboration graph of the players in the dataset.

In the collaboration graph, nodes represent players and edges indicate collaboration. There are 42 nodes, and 123 edges in this graph, demonstrating that 3 players have no collaboration at all and there are many pairs of players who don't collaborate with each other. The collaboration graph has an average degree of 5.8, showing that each player collaborates with 6 other players on average.

To make the collaborativeness of players comparable, [57] normalizes the amount of collaboration of players by the total active time of the player and calls it the collaboration index. We calculated the collaboration index for players in the dataset. The collaboration index of players in this dataset has a skewed distribution with minimum, median, and maximum of 0, 0.014, and 0.79, respectively.

We considered players to be *highly* collaborative if their collaboration index is higher than 90% of players, and *hardly* collaborative if their collaboration index is in $10th$ percentile. Collaboration index is between zero to one with a $10th$ percentile and $90th$ percentile of 0.002 and 0.19, respectively.

To compare players who are more collaborative with less collaborative players, we applied our algorithm to find contrast motifs of these two groups of players. The median length of player sequences was 2011 in our dataset. We removed players who barely played the game (with sequences shorter than 10) and conducted the experiments with remaining 41 players. We tested window sizes from 3 to 15 for the motif. A contrast motif was discovered for window size 8 for highly collabora-

tive players, but no contrast motif was found for players with lower collaboration index. Figure 6.6 shows the graph visualization of the contrast motif of highly collaborative players.

Comparing the motif illustrated in Figure 6.6 with the action motifs of Figure 6.4 shows that the motif of highly collaborative players has the most intersection with the fighting behavior. In other words, the behavior that is shared between highly collaborative players is fighting. This finding is aligned with prior research that indicates a strong correlation between fighting and collaborativeness in Minecraft [57].

Additionally, we attempted to apply PrefixSpan algorithm [28], which is a classic sequential pattern mining algorithm to find sequential patterns of these two groups of players. On a computer with 16GB RAM and a 4-core 1.9 GHz CPU, the PrefixSpan algorithm ran out of memory and failed while our motif discovery approach successfully completed within two hours.

Chapter Summary

In this chapter, we studied various aspects of low-level event sequences of high-level Minecraft actions: fight, build, mine, and explore. Additionally, we looked at two groups of players: highly collaborative and hardly collaborative. We discovered that highly collaborative players have similar behaviors while no common behavior was found among hardly collaborative players.

CHAPTER 7: CONCLUSIONS

By making data collection inexpensive and convenient, games such as Minecraft and socio-technical platforms such as GitHub advance our understanding of social science. Sequence mining can assist in this endeavor by summarizing a large volume of user data into a more intuitive format. We presented two sequence mining approach to facilitate the analysis of users' event sequences in Minecraft and GitHub.

We proposed an approach to explain classification results of groups of discrete sequences by quantifying the differences between the groups. We presented case studies of how our approach can be used to understand GitHub teams and Minecraft actions. We used our approach to study two different GitHub repository groups: those who use automated accounts (bots) and those who don't. Our analytic approach reveals subtle differences in teamwork patterns that are difficult to distinguish from event distributions. We believe that sequences of GitHub events can be mapped to team cognitive processes such as knowledge-building, information sharing, and problem-solving; normally in psychology experiments this mapping is accomplished by human observers but we aim to do it with machine learning.

Our experiments reveal that human team event sequences are relatively distinct from human-bot teams in terms of the existence and frequency of short subsequences. This shows that the cadence of activity in human-bot teams is different than human only ones. The matrix profile analysis shows that human-bot teams exhibit differences in both average and absolute maximum values. By analyzing the matrix profile of teams, we see that human-bot teams are less likely to repeat event subsequences than human only teams. Although it is unsurprising that human developers avoid repetition, it is interesting that the usage of bots can be detected from the event sequences alone, without using features from the comments, repository profiles, or code. Moreover, we found

that in human-bot teams, issue comments are scattered through out the event sequences while in human-only teams issue comments are tend to cluster together.

We utilized our approach to study Minecraft action sequences. Our analysis shows that build and mine actions have less repetitive subsequences compared to fight and explore. Sequences of various actions have similar level of complexity. Our experiment reveals that improving the performance of Minecraft action classification is challenging because these groups of actions are extremely similar in terms of the existence and frequency of subsequences.

We applied our new contrast motif discovery technique to a Minecraft dataset. First, we analyzed the low-level sequences of high-level actions by extracting contrast motifs that distinguish actions from one another. The motifs of each action were visualized in a graph to facilitate the comparison between motifs of different actions. Many of the events shown in the graphs are consistent with our intuitions on how players would achieve the tasks. Some of them are aligned with prior research while others were uncovered only through the use of our algorithm.

Finally, we employed our algorithm to compare highly collaborative Minecraft players with hardly collaborative ones. We created a collaboration graph across players and calculated the collaboration index of every player. Our method discovered a motif that is shared between more collaborative players but that does not occur in the sequences of less collaborative users. Comparing the graph of this motif with the action motif graphs shows that the behavior shared between highly collaborative players is fighting. Our sequence mining approach can be used for the implementation of agents who possess theory of mind about their human teammates while also providing a glass box for social scientists to enhance their interpretations of human behavior.

LIST OF REFERENCES

[1] C. C. Aggarwal. *Data Mining: The Textbook*. Springer, 2015.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 3–14, 1995.

[3] A. S. Badashian and E. Stroulia. Measuring user influence in github: the million follower fallacy. In *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*, pages 15–21. ACM, 2016.

[4] T. L. Bailey. Dreme: motif discovery in transcription factor chip-seq data. *Bioinformatics*, 27(12):1653–1659, 2011.

[5] T. L. Bailey, M. Boden, F. A. Buske, M. Frith, C. E. Grant, L. Clementi, J. Ren, W. W. Li, and W. S. Noble. Meme suite: tools for motif discovery and searching. *Nucleic Acids Research*, 37(suppl_2):W202–W208, 2009.

[6] R. Bartle. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD Research*, 1(1):19, 1996.

[7] S. D. Bay and M. J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.

[8] C. B. Bingham and J. P. Davis. Learning sequences: Their existence, effect, and evolution. *Academy of Management Journal*, 55(3):611–641, 2012.

[9] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology*, 70:30–39, 2016.

[10] H. Borges, A. Hora, and M. T. Valente. Predicting the popularity of GitHub repositories. In *Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering*, 2016.

[11] C. Brown and C. Parnin. Sorry to bother you: designing bots for effective recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 54–58. IEEE, 2019.

[12] F. Chollet et al. Keras. `https://keras.io`, 2015.

[13] C. Dagum. The generation and distribution of income, the lorentz curve and the gini ratio. *Economie Appliquée*, 33(2):327–367, 1980.

[14] M. K. Das and H.-K. Dai. A survey of DNA motif finding algorithms. In *BMC Bioinformatics*, volume 8, page S21. Springer, 2007.

[15] D. Debkowski, A. Marrero, N. Yson, L. Yin, Y. Yue, S. Frey, and M. Kapadia. Contained: Using multiplayer online games to quantify success of collaborative group behavior. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[16] H. Du, Y. Su, and C. Li. Discriminative sequential pattern mining for software failure detection. In *Proceedings of the International Conference on Informatics and Systems*, pages 153–158, 2016.

[17] L. Erlenhov, F. G. de Oliveira Neto, R. Scandariato, and P. Leitner. Current and future bots in software development. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 7–11. IEEE, 2019.

[18] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb. Familiarity, complexity, and team performance in geographically distributed software development. *Organization Science*, 18(4):613–630, 2007.

[19] S. M. Fiore, M. A. Rosen, K. A. Smith-Jentsch, E. Salas, M. Letsky, and N. Warner. Toward an understanding of macrocognition in teams: predicting processes in complex collaborative contexts. *Human Factors*, 52(2):203–224, Apr 2010.

[20] S. M. Fiore, K. A. Smith-Jentsch, E. Salas, N. Warner, and M. Letsky. Towards an understanding of macrocognition in teams: developing and defining complex collaborative processes and products. *Theoretical Issues in Ergonomics Science*, 11(4):250–271, July 2010.

[21] S. M. Fiore and T. J. Wiltshire. Technology as teammate: Examining the role of external cognition in support of team cognitive processes. *Frontiers in Psychology*, 7:1531, 2016.

[22] S. M. Fiore and T. J. Wiltshire. Technology as teammate: Examining the role of external cognition in support of team cognitive processes. *Frontiers in psychology*, 7:1531, 2016.

[23] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[24] M. Golzadeh, D. Legay, A. Decan, and T. Mens. Bot or not? detecting bots in github pull request activity based on comment similarity.

[25] J. Gray, K. Srinet, Y. Jernite, H. Yu, Z. Chen, D. Guo, S. Goyal, C. L. Zitnick, and A. Szlam. CraftAssist: A framework for dialogue-enabled interactive agents. *arXiv preprint arXiv:1907.08584*, 2019.

[26] R. Guidotti, A. Monreale, S. Matwin, and D. Pedreschi. Black box explanation by learning image exemplars in the latent feature space. *CoRR*, abs/2002.03746, 2020.

[27] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 51(5):1–42, 2018.

[28] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the International Conference on Data Engineering*, pages 215–224, 2001.

[29] Z. He, S. Zhang, F. Gu, and J. Wu. Mining conditional discriminative sequential patterns. *Information Sciences*, 478:524–539, 2019.

[30] Z. He, S. Zhang, and J. Wu. Significance-based discriminative sequential pattern mining. *Expert Systems with Applications*, 122:54–64, 2019.

[31] J. D. Herbsleb and R. E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, 16(5):63–70, 1999.

[32] B. Herndon and K. Lewis. Applying sequence methods to the study of team temporal dynamics. *Organizational Psychology Review*, 5(4):318–332, 2015.

[33] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[34] P. Hukal, N. Berente, M. Germonprez, and A. Schecter. Bots coordinating work in open source software projects. *Computer*, 52(9):52–60, 2019.

[35] S. Imani, F. Madrid, W. Ding, S. Crouter, and E. Keogh. Matrix profile XIII: Time series snippets: A new primitive for time series data mining. In *IEEE International Conference on Big Knowledge (ICBK)*, pages 382–389, 2018.

[36] O. Jarczyk, S. Jaroszewicz, A. Wierzbicki, K. Pawlak, and M. Jankowski-Lorek. Surgical teams on github: Modeling performance of github project development processes. *Information and Software Technology*, 100:32 – 46, 2018.

[37] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 164–174. IEEE, May 2017.

[38] M. Joblin, S. Apel, and W. Mauerer. Evolutionary trends of developer coordination: a network approach. *Empirical Software Engineering*, 22(4):2050–2094, August 2017.

[39] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

[40] S. J. Kang, Y. B. Kim, and S. K. Kim. Analyzing repetitive action in game based on sequence pattern matching. *Journal of Real-time Image Processing*, 9(3):523–530, 2014.

[41] E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

[42] E. E. Kastbjerg. Combining sequence mining and heatmaps to visualize game event flows. *Master's thesis, IT University of Copenhagen*, 2011.

[43] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[44] L. Langohr, V. Podpečan, M. Petek, I. Mozetič, and K. Gruden. Contrast mining from interesting subgroups. In *Bisociative Knowledge Discovery*, pages 390–406. Springer, 2012.

[45] C. R. Lebeuf. *A taxonomy of software bots: towards a deeper understanding of software bot characteristics*. PhD thesis, 2018.

[46] M. A. Leece and A. Jhala. Sequential pattern mining in Starcraft: Brood War for short and long-term goals. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[47] K. Lewis, D. Lange, and L. Gillis. Transactive memory systems, learning, and learning transfer. *Organization Science*, 16(6):581–598, 2005.

[48] V. C.-C. Liao and M.-S. Chen. Efficient mining gapped sequential patterns for motifs in biological sequences. *BMC Systems Biology*, 7(S4):S7, 2013.

[49] T. Y. Lin. Granular computing. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 16–24. Springer, 2003.

[50] D. Liu, M. J. Smith, and K. Veeramachaneni. Understanding user-bot interactions for small-scale automation in open-source development. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2020.

[51] C. Luo and S. M. Chung. Efficient mining of maximal sequential patterns using multiple samples. In *Proceedings of the SIAM International Conference on Data Mining*, pages 415–426, 2005.

[52] S. Makarovych, A. Canossa, J. Togelius, and A. Drachen. Like a DNA string: Sequence-based player profiling in Tom Clancy's The Division. In *Artificial Intelligence and Interactive Digital Entertainment Conference*. York, 2018.

[53] C. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[54] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 19–29. ACM, 2014.

[55] S. Mirhosseini and C. Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 84–94. IEEE, 2017.

[56] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover. Exact discovery of time series motifs. In *Proceedings of the SIAM International Conference on Data Mining*, pages 473–484, 2009.

[57] S. Müller, S. Frey, M. Kapadia, S. Klingler, R. P. Mann, B. Solenthaler, R. W. Sumner, and M. Gross. Heapcraft: Quantifying and predicting collaboration in Minecraft. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[58] S. Müller, M. Kapadia, S. Frey, S. Klinger, R. P. Mann, B. Solenthaler, R. W. Sumner, and M. Gross. Statistical analysis of player behavior in Minecraft. In *Proceedings of the International Conference on the Foundations of Digital Games*, 2015.

[59] G. Murić, A. Abeliuk, K. Lerman, and E. Ferrara. Collaboration drives individual productivity. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019.

[60] S. Nebel, S. Schneider, and G. D. Rey. Mining learning and crafting scientific experiments: A literature review on the use of Minecraft in education and research. *Journal of Educational Technology & Society*, 19(2):355–366, 2016.

[61] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[62] B. T. Pentland. Sequential variety in work processes. *Organization Science*, 14(5):528–540, 2003.

[63] M. Poppendieck and T. Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, 2003.

[64] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[65] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270, 2012.

[66] C. Riedl and A. W. Woolley. Teams vs. crowds: A field test of the relative contribution of incentives, member ability, and emergent collaboration to crowd-based problem solving performance. *Academy of Management Discoveries*, 3(4):382–403, 2017.

[67] N. Robette and X. Bry. Harpoon or bait? a comparison of various metrics in fishing for sequence patterns. *Bulletin of Sociological Methodology/Bulletin de méthodologie sociologique*, 116(1):5–24, 2012.

[68] D. M. Romero, D. Huttenlocher, and J. M. Kleinberg. Coordination and efficiency in decentralized collaboration. In *Proceedings of the 9th International AAAI Conference on Web and Social Media*, pages 367–376. AAAI Press, 2015.

[69] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[70] S. Saadat, C. Gunaratne, N. Baral, G. Sukthankar, and I. Garibay. Initializing agent-based models with clustering archetypes. In *International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation*, pages 233–239. Springer, 2018.

[71] S. Saadat, O. B. Newton, G. Sukthankar, and S. M. Fiore. Analyzing the productivity of github teams based on formation phase activity. *arXiv preprint arXiv:2011.03423*, 2020.

[72] S. Saadat and G. Sukthankar. Contrast motif discovery in minecraft. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 266–272, 2020.

[73] S. Saadat and G. Sukthankar. Explaining differences in classes of discrete sequences. *arXiv preprint arXiv:2011.03371*, 2020.

[74] P. Saraf. Improved prefixspan algorithm for efficient processing of large data. 2015.

[75] M. H. Schulz, D. R. Zerbino, M. Vingron, and E. Birney. Oases: robust de novo rna-seq assembly across the dynamic range of expression levels. *Bioinformatics*, 28(8):1086–1092, 2012.

[76] M.-A. Storey and A. Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, pages 928–931, 2016.

[77] M.-A. Storey and A. Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931, 2016.

[78] M.-A. Storey, A. Zagalsky, F. F. Filho, L. Singer, and D. M. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43(2):185–204, February 2017.

[79] Supplementary materials. `https://bit.ly/30xrLV5`.

[80] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[81] S. Torkamani and V. Lohweg. Survey on time series motif discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(2):e1199, 2017.

[82] B. W. Tuckman. Developmental sequence in small groups. *Psychological bulletin*, 63(6):384, 1965.

[83] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3789–3798. ACM, 2015.

[84] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload–a case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, Aug. 2014.

[85] U. Von Luxburg et al. Clustering stability: An overview. *Foundations and Trends in Machine Learning*, 2(3):235–274, 2010.

[86] P. Wagstrom, C. Jergensen, and A. Sarma. Roles in a networked software development ecosystem: A case study in github. 2012.

[87] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings. International Conference on Data Engineering*, pages 79–90. IEEE, 2004.

[88] M. Wessel, B. M. De Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–19, 2018.

[89] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa. Effects of adopting code review bots on pull requests to oss projects. In *IEEE International Conference on Software Maintenance and Evolution. IEEE Computer Society*, 2020.

[90] M. Wessel, I. Steinmacher, I. Wiese, and M. A. Gerosa. Should i stale or should i close? an analysis of a bot that closes abandoned issues and pull requests. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 38–42. IEEE, 2019.

[91] S. A. Wheelan. *Group processes: A developmental perspective.* Allyn & Bacon, 1994.

[92] U. Wilensky. Netlogo. Technical report, Center for Connected Learning and Computer-based Modeling, Northwestern University, Evanston, IL, 1999.

[93] R. Wu, Q. Li, and X. Chen. Mining contrast sequential pattern based on subsequence time distribution variation with discreteness constraints. *Applied Intelligence*, 49(12):4348–4360, 2019.

[94] Y. Wu, J. Kropcznyski, R. Prates, and J. M. Carroll. Rise of curation in GithHub. In *AAAI Conference on Human Computation and Crowdsourcing*, 2015.

[95] C.-C. M. Yeh, N. Kavantzas, and E. Keogh. Matrix profile iv: using weakly labeled time series to predict outcomes. *Proceedings of the VLDB Endowment*, 10(12):1802–1812, 2017.

[96] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *IEEE International Conference on Data Mining (ICDM)*, pages 1317–1322, 2016.

[97] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, Z. Zimmerman, D. F. Silva, A. Mueen, and E. Keogh. Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Mining and Knowledge Discovery*, 32(1):83–123, 2018.

[98] F. Zambelli, G. Pesole, and G. Pavesi. Motif discovery and transcription factor binding sites before and after the next-generation sequencing era. *Briefings in Bioinformatics*, 14(2):225–237, 2013.

[99] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.