Cracking the Code: Unraveling Gender Disparities in Open-Source Contributions

by

Norhan Abbas

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Abstract

Within the world of open source software (OSS) development, previous research has shown that the success rate of pull requests (PRs) may exhibit gender-related imbalances. In this work, we seek to examine which factors may contribute to this imbalance; we do so by performing a comprehensive study on a corpus of over 50,000 accepted PRs taken from a set of well-known Python projects. We perform both stylometric and quality-based analyses on the submitted PRs by both female and male developers, and we find that the results vary across gender. For example, we found that code written by male developers is more prone to both bugs and blocker issues. Based on our experiences, we propose a set of actionable recommendations, aimed at fostering diversity and equal opportunities within the OSS ecosystem.

Acknowledgements

I deeply appreciate my supervisors Diogo Barradas and Mei Nagappan for their significant contributions to this thesis. Mei's guidance was crucial in directing the research, and Diogo's support, feedback, and open-door policy were instrumental in the thesis's successful completion.

Moreover, the sense of community and shared aspirations within our CrySP lab, combined with the support and encouragement of friends Menna, Mona, Diaa, Lucas, Nils, Vecna, Adrian, Thomas, and Bailey have collectively enriched my experience and emboldened my efforts. I am truly grateful for your presence and contributions, which have played an integral role in the achievements of this journey. In a personal note, I must express my profound gratitude to my mother for her unconditional love and unwavering support throughout my life's endeavors. I also wish to express my gratitude to my sister, who has been by my side through thick and thin. I genuinely wish her all the success she deserves in her endeavors.

Last but not least, I'd like to thank my readers, Joanne Atlee and Michael Godfrey, for their interest and engagement with this work. Your time and insights are deeply valued and appreciated. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In recent years, the open-source software (OSS) community has emerged as a dynamic and thriving landscape, uniting developers from various origins/diverse backgrounds to collaborate on the development and improvement of software applications. The OSS ecosystem has long been marked by a significant gender imbalance, with male developers comprising the majority of participants engaged with the community [9]. Indeed, this gender disparity is reflected in different aspects, including sheer developer demographics and a skew of participation levels towards male developers.

Previous research has revealed numerous challenges encountered by female developers within this predominantly male-dominated environment at large. This encompasses instances of gender bias, discrimination, and even exclusion. To illustrate, a study [54] conducted on collaborative online software development platforms like GitHub have unveiled a distinct degree of negative treatment experienced by female developers upon their gender being disclosed to the community. A tangible case in point arises from findings of Terral et al.[65], which exposed that when a female developer's gender is revealed during the submission of pull requests (e.g., due to a clearly marked female nickname, or gender information contained in public user profiles), the integration of these contributions into upstream code repositories tends to encounter a higher rate of rejection.

In light of these gender-related biases, two intriguing research questions arise:

- *Are there substantial disparities in coding styles and patterns between female and male developers?*

- *Does a potential bias against gender lead to a higher rate of rejection for pull requests submitted by females?*

In this thesis, and towards addressing these questions, we develop a study that aims to explore whether code stylometry characteristics and code quality indicators can help us shed light over the actual differences (if any) on how male and female developers craft computer code. While code stylometry techniques [40, 12] offer a means to examine and quantify various aspects of code (e.g., indentation preferences, naming conventions, and usage of comments), other static code analyzers help us define the overall quality of code (e.g., resorting to standard-conformance checks, verifying the absence of vulnerabilities, etc.). We employ these techniques to analyze the underlying structures and patterns that shape code and potentially uncover distinguishing features between code written by male and female developers. This analysis also holds the potential to reveal any implicit biases or gender-specific coding practices that may exist within the open-source community.

To assess the presence of gender-based differences in code written by male and female developers, we leveraged machine classification tools, fueled with the above sets of features, and quantified the success of these classifiers in distinguishing male from female code. To conduct our experiments, we leveraged a fraction of a prominent dataset on the pull-based development model [71], and which includes annotations describing the gender of the developer that submitted a given accepted pull request. Specifically, we curated a dataset that spans over 50 000 accepted pull requests from four relevant OSS projects written in the Python language, including a total of 40 500 contributors.

Our findings show that training XGBoost classifier on a balanced dataset offers valuable insights into the disparities within pull request acceptance rates among female and male developers in the OSS community. Particularly noteworthy is the classifier's ability to achieve a robust recall rate of 0.55 for female developers, effectively capturing more than half of the actual female-initiated pull requests. Although precision for the female class stands at 0.34, the balanced interplay between these metrics yielded an F1 score of 0.42.

Our analysis uncovered a distinct coding divergence, as male developers' pull requests exhibited a notably higher frequency of bugs and blocker issues, accompanied by discernible differences in Python keywords usage. These quantitative insights, along with our comprehensive exploration, underscore the importance of reevaluating gender-associated assumptions and fostering inclusive coding culture in OSS community.

Overall, our findings support the hypothesis that there are indeed significant stylistic differences in how male and female developers produce code. We argue that, through the examination and understanding of disparities such as the above, this study can contribute to the ongoing discussions surrounding gender inclusivity and diversity within the open-source software community. Specifically, our study highlights the need to recognize and address gender biases towards dismantling preconceived notions about coding abilities and encouraging a more meritocratic evaluation of developers' skills and contributions. This, in turn, promotes inclusivity by valuing and respecting the diverse range of coding styles and approaches that individuals, regardless of gender, can bring to the open-source community.

1.2 Contributions

We initiate an investigative expedition into the domain of gender-associated coding patterns within the context of open-source, aiming to illuminate the intricacies that influence the approval of pull requests, in light of gender revelation.

Our list of contributions is as follows:

1. We construct a dataset characterized by individual records, each intricately detailed with descriptive attributes pertaining to pull requests. These records are meticulously annotated with the respective gender of the contributing code author. Our dataset synthesis draws inspiration from the work of Zhang et al. [71], encompassing a compilation of over 50,000 approved pull requests sourced from prominent Python projects. This compilation spans across four popular projects and involves the collective contributions of 40,500 distinct contributors. Our feature ensemble comprises three distinct sets of code stylometry and quality indicators, consisting of one adapted from the seminal study by Zhang et al. [71], and two uniquely devised by our team.

2. Our analysis peels back the layers of code stylometry and quality, unveiling the nuances that distinguish coding contributions from female and male developers. We uncover that stylistic features, along with some code quality indicators, and hold the key to distinguishing these coding habits.

3. Our study goes beyond the realm of traditional research, serving as a compelling call for awareness and transformation within the open-source software landscape. By spotlighting the presence of gender biases and disparities, we prompt a collective reckoning. Through practical recommendations grounded in our findings, we advocate for diversity, equality, and inclusivity as essential drivers of advancement in open-source software ecosystem.

1.3 Organization

This thesis is organized as follows. In Chapter 2, we provide background knowledge on existing gender diversity issues in the open-source software community and discuss techniques that can help perform the identification of developers of open-source software. In Chapter 3, we detail the methodology of our study, providing details on data collection, processing, and analysis methods. In Chapter 4, we present the results of our analysis. Lastly, in Chapter 5, we summarize the main takeaways of our study, discuss the limitations of our analysis, and introduce directions for future work.

Chapter 2

Related Work

Over the past few decades, researchers have devoted significant attention to investigate the concept of diversity in the field of Software Engineering. In a study conducted by Miller et al [50], cognitive diversity was defined as the variation in beliefs and preferences among team members, which contributes to shaping the objectives of the organization. Cognitive diversity encompasses both observable distinctions, such as race, as well as less apparent differences, including disparities in background and levels of experience [49]. Examining the relationship between diversity among team members and the team outcomes has yielded some remarkable findings, indicating that diverse teams are more inclined to outperform teams with more homogeneous cognitive team [37]. These findings highlight the significance of diversity within Software Engineering, showing that a wider range of perspectives have a greater likelihood of improving team performance and achieving favorable outcomes.

A team is a collection of individuals [64], with different tasks that can require synchronization, who share responsibility for the project outcomes. A positive correlation has been established in research between team diversity and team performance. To illustrate, it was found out that functional diversity could be a catalyst for faster product development in the computer industry [25]. Various stages of product development require different multifunctional expertise, for that reason multifunctional involvement was correlated to successful projects [55][16]. Additionally, Keller [41] found out that functional diversity had indirectly improved scheduling and budgeting and served as a job distressor. Since job stress could hinder the team from moving forward and interfere with the development process, functional diversity is useful in this case [41]. From a financial point of view [36], it has been noticed that gender-diverse teams could perform better, especially when a significant proportion of management is women. Emphasizing on the previous, a business case study [34] has proved that gender-diverse teams have better team dynamics, collegial relationships, and productivity. Even though diversity in teams could result in knowledge increase [47], it could be a source of conflict due to different team members' perception and gaps between their interpretation [18] [53].

As researchers have made progress in studying cognitive diversity in Software Engineering, the lack of gender diversity has emerged as a challenge. In response to that, the workforce has started prioritizing diversity and inclusion polices [21], and some organizations, such as Google and IBM, have even devoted to create a diversity and inclusion office [68]. Despite the corporate effort to diversify their workforce and broaden the profiles of their employees, the proportion of female developers in the industry has barely seen any change. A study [46] in 2014 documented that only 18% of bachelor's degrees in Computer Science were earned by women; in addition, a labor market consensus in 2015 revealed that men still dominate the software industry, accounting for 79% of developers [68]. One prominent obstacle that is regarded as a barrier to altering gender ratio, which has been extensively documented, is gender-bias. Female underrepresentation in the Software Developer workforce is attributed to being subjected to judgement and prejudice based on gender. Consequently, around 41% of females leave the tech industry after 10 years [69].

2.1 Gender Diversity in Software Development

Gender stereotypes pertaining to women and men could potentially strengthen and solidify particular gender roles, ultimately influencing women's role within the information technology (IT) industry. To illustrate, studies [7] [59] [23] sought to identify and analyze traits attributed to female and male gender roles. It was found out that empathy, emotionality, and dependency were commonly affiliated with femininity, while decisiveness, dominance, and aggression were associated with masculinity. Consequently, that perception unintentionally contributed to organizational segregation as some managers [57] lean more towards hiring female employees in socially oriented positions, such as project management. Hence, a gender gap [46] [67] [24] has been noticed in the field of Computer Science (CS). Other studies [17] dived into possible factors that affects gender gap in IT, they found out that the lack of female role models is one of the obstacles in the way of more female entering the software industry. For example, young women are usually attracted to fields where they have some female icons and can see women succeeding.

2.2 Gender-Bias in Open Source Software (OSS)

Although gender bias behaviour is not a recent occurrence within the software industry, it has reached an alarming level in the Free and Open-Source Software (FOSS) community. The severity of the situation became apparent through a survey [29] that was carried out within the FOSS community. The survey showed that female participation was remarkably low, ranging from 2% to 5%. Consequently, some women [51] tried to protect themselves from potential discrimination as they used gender-neutral names when participating in online forums.

Terrell et al. [65] studied biases against women in Open Source Software (OSS); they compared women's and men's contribution to OSS as they examined different pull requests submitted by women and men on GitHub. To their surprise, they found out that it is much likely for pull requests submitted by women to be merged unless the gender is visible. Possible justifications for Terrell's findings could include i) survivorship bias, where men could possibly be perceived as always successful developers, as a result this type of bias predicts the future performance of any male developer ii) self-selection bias, where repositories' owners choose who could contribute to their projects iii) women could possible be held to higher performance.

Later on, Bosu and Sultana [9] studied the level of gender diversity in some popular OSS projects, and even though they noticed how badly gender diversity was in some of the OSS projects they investigated, they found no significant differences between female and male developers, in terms of productivity. By the same token, when Canedo et al. [13] studied the vertical segregation in OSS, they found no substantial differences between female and male developers, work practice wise. Moreover, another study investigated how biased code review processes could possibly be, and it was noted that female developers have way lower code participation than male ones, which indicated that that the code reviewer selection has potential affinity biases [63]. Due to such biases, promoting diversity and inclusion has been so challenging that some women switch careers due to their negative experiences.

2.3 Promoting Gender Diversity in CS

In order to address the underrepresentation of women in the software industry, several interventions have been proposed [8]. One factor contributing to the lower representation of women in Computer Science (CS) is the lack of pre-college experience in the field of computing. Accordingly, some of these interventions have focused on college students, specifically targeting first year students who have no prior exposure to programming. These interventions have taken the form of free and informal workshops, designed to provide hands-on experience with some simple tasks in a stress-free and dynamic learning environment. In these workshops, students actively participate in a range of hands-on programming activities. Another suggested approach is peer-to-peer interaction, through pairing more experienced students with less experienced ones. This setup encourages teamwork and cooperation among students as they collaborate on tasks and assignments that involve programming concepts.

Moreover, "live coding" sessions have emerged as a recommended strategy to bridge the gender diversity gap in the field of computing. During these sessions, experienced CS tutors guide students to follow examples and gain hands-on experience in real-time. These sessions are combined with an introductory course in programming language and can be conducted on a weekly basis. Ultimately, the goal of these interventions is to increase gender diversity within the field of computing by equipping students with the necessary skills and boosting their confidence to thrive in the field.

Another research [20] conducted at Cardiff University in UK proposed the review of CS course materials with the aim of identifying potential examples of unconscious bias or stereotypes. This study highlighted the importance of reviewing instances because they can unintentionally reinforce biased or hetero-normative ideas. For example, some books use phrases like every man loves a women, which promotes a stereotype that could negatively affect the confidence and inclusivity of individuals from minority groups.

2.4 Techniques for Gender Identification in OSS

Currently, heterosexual men have a strong presence in the software development organizations, which has resulted in a disparity as it comes to gender diversity [62]. Consequently, individuals from other demographic backgrounds, such as women witness gender-bias and discrimination within this realm. As a result, many women try to seek other opportunities in different career paths, where the work environment is more inclusive. Accordingly, it is important to recognize biases, identify and understand the impact of them. For the previous reasons, some researchers have studied gender identification. It is an approach to predict the class to be "female", "male", or "unknown" when the gender cannot be determined.

2.4.1 Name-Based Approach

This approach is dependent on gender resolution [66], based on some database with previously collected names labeled as either "female" or "male". In some cases, if the name in question is not disclosed (aka not included in the database), it gets categorized as "unknown". For example, several studies [44], [33] based their name-based gender approach on the public data provided by the US Census Bureau and US Social Security Administration. When some common names were found to be given to both women and men, the authors considered the frequency of use for each gender, and assign the gender with the most usage to that common name.

2.4.2 Image Processing Technique

Lu et al. [48] introduced image processing as a method to identify gender. The following approach offers a potential alternative to the conventional avatar analysis. However, it is crucial to note that a significant number of avatars, which are used in various platforms, are not actual facial images but rather symbolic representation, cartoon, or even isolated body parts. The previous poses a challenge to the proposed technique, as it primarily relies on facial images for accurate gender identification. Additionally, the implementation of this technique involves a series of manual preprocessing steps, including cropping, resizing, and rotation of images. These manual preprocessing adds another layer of complexity to the overall process as it requires human intervention.

2.4.3 Stylometry

Stylometry, a "metric" that could determine the originality level of a writing style [22], has been regarded as a discriminator to detect plagiarism in early age. It quantifies and characterizes different styles using a set of various features, which could later point fingers to a pool of candidates with similar style, based on previously investigated samples. Some studies have shown that style could be quantifiably measurable using some statistical and quantitative description of the disputed work, whether it was text, code, etc. Tools using a stylometric approach [31] usually compare the piece in question against a collection of other referenced pieces in that tool, under the assumption that they stem from different authors.

Code Stylometry

The pursuit of attributing code to its creators has demonstrated its potential in various aspects of software analysis, including the identification of potential cyber threats through forensic examination, as highlighted by the research conducted by Amuchi et al.[3]. Beyond its application in cybercrime detection, this methodology extends its significance to the realm of addressing cyberbullying incidents. Nevertheless, this endeavor introduces a complex ethical dilemma: while authorship attribution serves as a tool for unveiling individuals involved in malicious online activities, it concurrently encroaches upon their privacy by revealing their identities without explicit consent.

This intricate scenario underscores the critical importance of unraveling the origins of code within the domain of software investigation. For instance, it proves instrumental in tackling the intricate challenge of determining whether a specific set of malicious software can be attributed to a particular programmer. This feat presents a formidable endeavor in the broader context of classifying software entities globally. The illustrative example magnifies the value of associating code with its authorship, illuminating its potential in unraveling intricate enigmas within the realm of software intricacies. In a bid to reconcile this limitation, Caliskan-Islam et al.[12] adopt a balanced approach by leveraging the capabilities of the fuzzy parser Joern, meticulously designed to navigate incomplete code fragments[70]. This dynamic parser generates abstract syntax trees from code segments, prioritizing viable parsing while bypassing fragments requiring additional context.

As our study transitions to the methodology chapter, we leverage these foundational insights to embark on an exploration of gender-based disparities in pull request acceptance rates and coding patterns.

Chapter 3

Methodology

This chapter introduces different dimensions of the methodology followed in our study. We start by providing details about how we collected the necessary data for conducting our study (Section 3.1), and describe our feature selection process (Section 3.2). Then, we describe our experimental setup (Section 3.3) and introduce the machine learning models and metrics used in our evaluation (Section 3.4).

3.1 Data Collection and Scaling

In this section, we describe the source of pull requests analyzed in the context of our study (Section 3.1.1) and describe the techniques we used to tackle the class imbalance identified in our dataset (Section 3.1.2).

3.1.1 Pull Requests Dataset

Data sources and acquisition of pull request data. A fundamental requirement for our study is that of selecting a dataset comprised of pull request data that is annotated with information about the gender of the pull request submitter. Recently, Zhang et al. [71] have investigated the pull-based development model and created a large dataset that includes records tied to pull requests across multiple open-source projects hosted in GitHub. This dataset aggregates different pieces of information about these projects, including various aspects about contributors, pull requests, and meta-information about the projects

Table 3.1: Summary statistics about our dataset containing pull requests in Python.

Projects	Contributors	Total Pull Requests	Pull Requests by Females (%)	Pull Requests by Males (%)
4	40 500	56 126	13.08%	86.92%

themselves. Fortunately, this dataset includes the gender of project contributors as a feature.

In essence, the above dataset enables us to use the gender feature as a ground-truth class label and to link the gender of a contributor to a specific pull request. In our study, we will gather multiple features that characterize the code of a pull request and then apply machine learning classifiers toward understanding whether these features can help us distinguish the way male and female programmers write code. In Section 3.2, we elaborate on our approach to extract features from our pull requests dataset.

It should be noted that the dataset of Zhang et al. does not include the code for the pull requests themselves, albeit it includes an identifier for each pull request. During the preliminary steps of our analysis, we use these identifiers to fetch the corresponding pull requests from GitHub and then build a dataset where each record contains different features for each pull request and are labeled according to the pull request submitter's gender, as annotated by Zhang et al..

Data preprocessing. We preprocessed and categorized the pull requests present in Zhang et al. [71]'s dataset based on the programming language utilized to write code. We placed particular emphasis on pull requests submitted to projects written in Python, a popular programming language which emerged as the second most prominent one amongst the set of considered open-source projects in terms of size, including more than 700k pull requests. Yet, it is crucial to note that our study's primary focus revolves around the exploration of coding styles related to gender attributes. Since we found that not all contributors in Zhang et al.'s dataset are tagged with a specific gender (some records are effectively marked as *unknown*), we took the necessary step of excluding from our analysis all pull requests whose submitter's gender attribute was missing.

We focused our analysis on a subset of the projects considered by Zhang et al. and that adhered to the constraints spelled out in the previous paragraph. We selected the four largest projects based on the number of pull requests. This decision was motivated by the complexity involved in analysing a large number of pull requests for extracting code quality features (see more details on our feature extraction procedures in Section 3.3.3). Table 3.1 presents a summary of the composition of our dataset. As shown in the table, (and after conducting the above preprocessing operations) our dataset is composed of a total of 56 126 pull requests (∼13% submitted by females and ∼87% submitted by males), split amongst 4 large projects, and including a total of 40 500 unique contributors, out of which 5,297 were identified as female developers and 35 203 were identified as male developers. We can observe that there exists a large majority of male coders, further revealing the need for a deeper look into whether there are meaningful differences in the style of code written by male and female developers.

3.1.2 Dealing with Data Imbalance

As seen in the previous section, the dataset we obtained is unbalanced and largely skewed toward male developers. In the realm of data analysis, imbalanced datasets refer to the fact that there might be a minority class with too few samples or that one specific class outweighs the others. This imbalance may present a challenge for learning algorithms, possibly leading to biased model performance. In our evaluation, we present results both when learning over this imbalanced data (Section 4.1.1), and after applying data oversampling techniques (Section 4.1.2) that help us close the gap between the number of pull requests produced by male and female developers, as considered in our training dataset.

Strawman approaches to learn over imbalanced datasets rely on data oversampling and undersampling techniques. For instance, random oversampling relies on duplicating some of the data points of the minority class, while undersampling could simply rely on trimming the number of instances in the majority class. However, these methods do not come without their own specific drawbacks; for oversampling, our model is not being fed with entirely new information, while for undersampling we might potentially discard records that bring important information for training a machine learning model.

A popular oversampling technique for dealing with imbalanced datasets is called Synthetic Minority Over-Sampling Technique (SMOTE), having been introduced by Chawla et al. [14] in 2002. SMOTE aims to balance the dataset of the minority class as it increases its representation through selection of an instance, finding its k nearest neighbors, then creating a synthetic data point from the shared feature space. This helps the model trained in the SMOTE-produced data to be less biased.

In our evaluation, we will also gauge the utility of using SMOTE to compensate for data imbalance during training and assess whether the use of oversampling can help us better identify the differences between the ways male and female developers write computer code.

Table 3.2: Number of attributes considered in each feature set.

Feature set	**Pull request characteristics**	**Code quality indicators**	**Keyword frequency**
Number of features	17	94	35

3.2 Feature Selection

In this section, we describe the feature sets used in our study (Section 3.2.1), and how we pre-processed these features prior to conducting our analysis (Section 3.2.2).

3.2.1 Feature sets

Here, we describe the three different feature sets we used to gather insights on how male and female programmers write code. These features rely on a) general pull request characteristics, b) code quality indicators, and c) language keywords' frequency. All the considered feature sets rely on information that is statically obtained from the code of the pull requests considered in our dataset. In our evaluation, we analyse the contribution of each feature set in our ability to distinguish male and female coders, and analyse how the combination of different feature sets impacts our results. Table 3.2 depicts a summarized breakdown of the number of attributes included in each feature set, comprising a total of 152 features.

Feature Set 1 – Pull request characteristics. The first feature set that we leveraged in our work corresponds to the set of pull request attributes previously documented by Zhang et al. [71], who focused on the analysis of the pull-based development model. While the original work of Zhang et al. relied on a more comprehensive set of features (which also included project and contributor characteristics), we filtered the original feature set to exclusively retain the features directly associated with pull requests' data. This filtering step aimed to produced a feature set that is exclusively focused on code-related attributes, excluding features that provide details related to the project itself or its contributors. Thus, the pull request characteristics we retain are intended to provide us with information regarding different coding styles, allowing us to explore if women and men do indeed code differently.

In particular, we retain information about properties that describe each pull request, such as its complexity, the number of lines added, whether it fixes a bug, or whether it includes any tests. Other features that we ignore comprise a) contributor characteristics, which are factors associated with both the pull request submitter and integrator, including country of origin, gender, experience level, and response time, or b) project characteristics, which primarily focus on attributes such as the programming language used, the project's popularity, and the workload it entails.

We extracted a total of 17 features based on pull request characteristics. The full listing of considered features is shown in Appendix A.1.

Feature Set 2 – Code quality indicators. The second set of features considered in our study focuses on code quality features, as measured by automatic static code analysis tools which analyse code and provide developers with suggestions on how to improve their programs across different metrics like readability, extensibility, or security. Specifically, we used SonarQube, a software tool designed for static code analysis that can evaluate code quality and identify potential software vulnerabilities. Developers use SonarQube to automate code reviews, detect code smells, evaluate code duplication, and provide suggestions based on coding standards and best practices. SonarQube provides a wide range of features for analysing source code in different programming language.

Amongst the features extracted by SonarQube, we highlight the identification of potential vulnerabilities (and their type), the recognition of code smells, and general statistics about the analysed code such as the number of lines of code found in a given pull request.

We extracted a total of 93 features based on the results of the analysis conducted by SonarQube. The full listing of considered features is shown in Appendix A.2.

Feature Set 3 – Keyword count. The third feature set considered in our study is based on the quantification of different language-specific keywords which are included in the pull requests submitted by programmers. More specifically, we counted how often Python keywords (such as `if`, `with`, or `try`) appeared in each pull request.

We counted all 35 Python keywords. The full listing of the keywords is shown in Appendix A.3

3.2.2 Feature pre-processing

After obtaining the features for each of the feature sets described in the previous section, we observed that a substantial number of features associated with code quality indicators were not available for all the pull requests included in our dataset. This was due to the kind of information SonarQube requires to generate values for some of its quality indicators. For instance, the number of classes per source code file is a feature that will be missing when a pull request's modifications consist of only a few lines which do not include class declarations. In such cases, SonarQube does not make the number of classes available for that specific sample. Another example is the cognitive complexity indicator, which assesses the difficulty of understanding the code's control flow. This feature often has a large number of missing values since our pull request data mostly comprises small snippets of code over which SonarQube's control flow analysis cannot be performed.

To prevent potential biases and inaccuracies that could arise from the absence of feature values across the dataset [2], we undertook a preprocessing step with the goal of tackling the issue of missing feature values. This procedure took place as follows.

First, we assessed the percentage of missing values for each feature in the dataset. By quantifying the ratio of missing values for each feature, we could determine the usefulness of each feature included in one of the considered feature sets.

Second, we undertook a feature filtering process where if the number of records missing a given feature exceeded a threshold percentage t, we would exclude that feature from our analysis. (For instance, if a feature is missing in 60% of the records in the dataset, it would be excluded if $t \geq 0.6$.) In our evaluation, we analyse the impact of the choice of this threshold in our ability to distinguish female from male coders.

Lastly, for the remaining samples including features with missing values (but whose overall ratio is not above threshold t), we filled missing feature values resorting to well-known methods [2]. Specifically, we filled numerical attributes with the average value for that feature amongst all non-empty records sharing the same class in the dataset, and similarly filled categorical attributes with the mode of that feature amongst all non-empty records that share a same given class.

3.3 Experimental Setup

In this section, we describe the laboratory testbed we used to perform our experiments (Section 3.3.1), detail the process through which we identified and acquired the pull requests of interest to our study (Section 3.3.2), and explain how we extracted each of our feature sets from the pull request code we collected (Section 3.3.3).

3.3.1 Laboratory testbed

To conduct our research, we benefited from the RIPPLE Facility at the University of Waterloo. The RIPPLE machines facilitated the collection of data, the generation of feature sets, and were used to speed up the generation of machine learning models during our analysis.

3.3.2 Data Extraction

Our analysis pipeline was run in the RIPPLE machines and includes various stages related to data extraction, starting with the retrieval of pull requests from GitHub. Then, we retrieve the most meaningful changes made by each pull request, which serves as valuable input for our analysis. Below, we provide additional details on each step of this data extraction pipeline.

Download of pull requests' code. We used the RIPPLE machines to access GitHub and retrieve the set of pull requests we were ultimately interested in for our analysis. We used the PyGitHub library to streamline the fetching of pull requests from GitHub given their identifier. PyGitHub is a valuable Python library that is regarded as a tool for developers to interact with GitHub. It offers a wide range of functions that enable developers to create and modify repositories, retrieve data, and manage pull requests.

Identification and extraction of useful pull request data. As part of our study, we are mostly interested in analysing the contributions that male and female developers submit through pull requests, as annotated in Zhang et al.'s dataset. Thus, our initial task was then to access and retrieve each pull request we considered as part of our dataset. To facilitate this process, we required the pull request ID (mentioned in Zhang et al.'s paper, but not included in their public dataset), which would allow us to access these pull requests effectively in GitHub. To this end, we reached out to the authors and were given access to the private version of the dataset [73] containing the necessary identifiers. However, the code contained in the pull requests also includes existing and deleted lines that are largely uninformative. Thus, once we fetched the pull requests, we ran a script aimed at extracting the added and modified lines for each pull request, keeping those lines of code as the defining data of a pull request in the context of our work. We accomplished this by extracting the difference between files included in the pull request right before and after the pull request was submitted.

3.3.3 Feature Extraction

Once the pull request changes were acquired, we used different methods to extract features about the coding style and quality of each pull request.

Feature Set 1 – Pull request characteristics. We leveraged Zhang et al.'s [71] study, which has already characterized the features related to pull requests in the dataset. The dataset they provided [72] encompassed various attributes, such as features about contributors, submitters, projects, and pull requests. As our investigation solely concerns pull requests, we concentrated only on the features directly related to pull requests.

Feature Set 2 – Code quality indicators. To utilize SonarQube's static code analysis [58], we uploaded each pull request as an individual project to be scanned on SonarQube. Through the use of a REST API, we were able to extract SonarQube's analysis for each pull request. Since we were using SonarQube's community edition, obtaining the tool's code quality analysis report containing the full set of features we were interested in collecting was not immediately possible. Instead, we had to submit 99 separate requests via REST API to extract each feature individually.

We observed that the community edition of SonarQube does not seem to be optimized for handling large amounts of concurrent code analysis conducted over different projects. Therefore, to handle our substantial dataset of 56 126 pull requests (where again, each had to be submitted as an individual project to SonarQube), we opted to run multiple SonarQube instances on separate machines in the RIPPLE infrastructure and craft multiple Python scripts to fetch and parse the necessary feature values from each running instance of SonarQube.

Feature Set 3 – Keyword count. For extracting keyword-based features, we leveraged the `keyword` module in Python to obtain a comprehensive list of reserved keywords in Python (by inspecting `keyword.kwlist`, a read-only attribute that returns a list of all keywords reserved for the Python interpreter). Next, we iterated over the added and modified lines of each pull request to count the occurrences of these keywords.

3.4 Models and Metrics

In this section, we describe the machine learning models we used to analyse pull request data, and introduce the metrics we relied on to assess how effectively these models can differentiate between code submitted by female and male developers.

3.4.1 Machine Learning Models

We now present a description of the decision-tree based algorithm we chose for conducting our experiments, and detail how we train and evaluate this model on our dataset.

Model selection. We chose to use the extreme gradient boosting (XGBoost)[15] classifier for the analysis conducted in our study. XGBoost is an optimized version of the gradient boosting algorithm, and it is particularly adept at solving classification and regression problems. It has recently become a popular machine classification model due to its excellent performance (comparable to deep learning classifiers in multiple different settings) and efficiency (typically easier and less expensive to train than deep learning architectures). The main advantage of XGBoost over other conventional decision-tree based techniques like Random Forests hinges on the fact that the former relies on a gradient boosting technique. To illustrate, Random Forest trains multiple decision trees independently on a subset of the training data, which is selected randomly. Then, a majority voting among the tress determines the final prediction. On the other hand, XGBoost builds trees iteratively such that each tree improves on the mistakes made by the previous one, and the final prediction is determined by summing up the outputs from all trees. Additionally, XGBoost mitigates the risk of overfitting through its regularized learning.

Training and evaluating the model. For training and evaluating our model's effectiveness, we partitioned the dataset into an 80% training subset and a 20% testing subset. This strategic division carries notable benefits, especially when tackling imbalanced datasets. When dealing with imbalanced datasets, this separation strategy ensures that both the majority and minority classes are well-represented in both the training and testing stages. This safeguards against undue bias towards the majority class during the model's training phase, promoting learning from a more equitable range of examples. Subsequently, during testing, the model is subjected to a varied array of instances, encompassing those from the minority class. This method furnishes a comprehensive gauge of the model's capacity to generalize across different categories, thus enhancing the dependability of assessing its performance on both majority and minority classes. To guarantee a thorough assessment, we meticulously retained the original class distribution in both the training and test sets. This meticulous choice takes on heightened significance within the context of imbalanced datasets. By upholding the proportion of instances from both the majority and minority classes, we aimed to construct training and test subsets that accurately mirror the traits of the primary dataset. This approach amplifies the model's exposure to the intricacies inherent in imbalanced class distributions during its training phase and establishes a robust foundation for appraising its performance across a diverse range of instances, including those from the minority class, during the testing phase.

3.4.2 Evaluation Metrics

We leverage well-known metrics for assessing the performance of our classifiers and to better understand the settings under which a classifier is able to distinguish code developed by male or female coders. In our evaluation setting, we consider (i) *true positives* to be a pull request submitted by a female and that is classified as a female's submission, (ii) *false positives* to be a pull request submitted by a male and that is classified as a female's submission, (iii) *true negatives* to be a pull request submitted by a male that is classified as a male's submission, and (iv) *false negatives* to be a pull request submitted by a female and that is classified as a male's submission.

The specific classification performance metrics we consider are as follows:

Accuracy. Often the most simple metric that can be used to evaluate a classification model's performance, it describes the number of correctly predicted instances over the total number of instances. However, accuracy can fail to showcase the performance of classifiers that possess an internal threshold for tuning the trade-off between true positives and false positives. Moreover, it does not reflect the model's ability to distinguish between the correctly identified examples of different classes, especially in data imbalance settings. For this reason, we used other metrics (described below), which help in providing a more comprehensive evaluation of the model's performance across different classes.

Recall. Also referred to as *true positive rate* or *sensitivity*, recall is a fundamental performance metric that evaluates how well a classifier can correctly predict positive samples. It is defined as the ratio of the number of correctly identified instances to the total number of instances the classifier deems positive. In our case, we consider recall to be the ratio between the number of pull requests submitted by females that are indeed classified as females' submissions, and the total number of pull requests by females in the dataset (see Equation 3.1). A higher recall value indicates that the model is more sensitive in terms of predicting a larger portion of positive samples.

$$\frac{TP}{TP + FN} \tag{3.1}$$

Precision. This metric is defined as the ratio of the correctly positive predictions to all samples predicted as positive. In our case, it is the ratio between the number of pull requests submitted by females classified as being submitted by females', to the total number of pull requests classified as females' (see Equation 3.2). A higher precision value indicates that the model can identify true positive samples effectively (identify pull requests by females as females').

$$\frac{TP}{TP + FP} \tag{3.2}$$

Precision-recall curve. The use of precision and recall in tandem is particularly useful to assess prediction success in scenarios with highly imbalanced classes. More concretely, the *precision-recall curve* illustrates the trade-off between precision and recall at various internal thresholds of the classifier. High precision corresponds to a low false positive rate, while high recall corresponds to a low false negative rate. Achieving high scores for both metrics demonstrates that the classifier produces accurate results (high precision) while capturing a substantial portion of all positive outcomes (high recall). We use the precision-recall curve to visually inspect the performance of our classifier on our original dataset of pull requests which exhibits a majority of pull requests submitted by male developers (and a minory class composed of pull requests submitted by female developers).

F1 score. Finally, we also resort to the F1 score, a metric that combines precision and recall into a single value by calculating the harmonic mean of precision and recall (see Equation 3.3). It is especially useful when the classes are imbalanced because it gives equal weight to both metrics. The precision-recall curve and the F1 score are also closely related. By examining the precision-recall curve, we can identify the optimal internal classifier's threshold that maximizes the F1 score. This threshold represents the point on the curve where precision and recall are both high. Thus, the F1 score can be considered a summary measure derived from the precision-recall curve.

$$2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.3}$$

Chapter 4

Analysis and Results

In this chapter, we present the outcomes of our analysis on the pull request dataset. Our aim is to discern whether the application of machine learning techniques allows us to differentiate between pull requests submitted by female and male developers. Perhaps more interestingly, we wish to shed light on the main differences between the ways female and male developers write code, and which ultimately lead to machine learning classifiers being able to perform such distinction.

To achieve our goals, we initiate the process by conducting an in-depth analysis of our dataset, wherein we apply a set of simplifying assumptions to determine a) which attributes of pull requests should be considered in our study, and b) how to identify and exclude potentially irrelevant features. Throughout this chapter, we will build upon the baseline analysis, initially presented in Section 4.1, to explore the impact of additional feature filtering operations on our findings.

4.1 Analysis of the Pull Request Dataset

In this section, we discuss the results of our analysis on the pull request dataset obtained from GitHub. We start by analyzing the imbalanced dataset (Section 4.1.1), which we collected following the methodology explained in Section 3.1. Afterwards, we perform a separate analysis by applying data oversampling techniques to our initial dataset (see Section 4.1.2).

Baseline configuration. Towards analysing the two settings mentioned above, we consider a dataset of pull requests whose feature set is a combination of the three individual

Table 4.1: Number of attributes considered in each feature set (after filtering with $t \geq 0.9$).

Pull requests characteristics			**Code quality indicators**			**Keyword frequency**		
Total	Considered	Removed	Total	Considered	Removed	Total	Considered	Removed
17	17	0	94	42	51	35	35	0

feature sets detailed in Section 3.2.1, yielding a total of 145 features. We also assume that features with a percentage of missing values equal to or exceeding 90% are discarded from the analysis (i.e., $t \geq 0.9$, where t is the missing value ratio). After this step, we were left with 94 features to inform our analysis. Accordingly, we imputed the missing values for the remaining attributes, which survived this filtering step, as described in Section 3.2.2.

Table 4.1 reveals the number of attributes considered in each feature set composing our merged feature set, after applying our filtering operations. We can see from the table that the code quality indicators feature set (extracted via SonarQube) underwent substantial feature reduction, from a total of 94 features down to 42. In contrast, when examining the dataset of pull request characteristics utilized by Zhang et al.[71], the features of interest were entirely devoid of any missing values for the pull requests we considered. As a result, the application of filtering techniques to address missing data in this specific feature set was deemed unnecessary. For Python keywords, we considered the following: the absence of a keyword in a pull request source code resulted in its corresponding frequency being recorded as 0. In this sense, we end up with no records missing attributes related to the keyword-based feature set. From this analysis, it becomes evident that the primary focus of feature removal is centered around SonarQube attributes.

In Section 4.2, we will assess how the selection of different values of the filtering threshold t impact our findings. Later, in Section 4.3, we compare the results obtained by our merged feature set with each of the individual feature sets considered in Section 3.2.1.

4.1.1 Analysis on Imbalanced Training Data

Overall classification performance. After training our XGBoost classifier on the imbalanced pull request dataset, we gathered the following metrics of interest: precision, recall, and F1 score to evaluate the classifier's performance. The results, presented in Table 4.2, demonstrate how effectively the classifier identifies pull requests from female and male developers.

The model shows a moderate level of precision for pull requests by females, which means that when it predicts a pull request as females', it is correct about 44% of the time. Yet, it

Table 4.2: Results of the XGBoost classifier on the unbalanced dataset($t \geq 0.9$).

Class	**Precision**	**Recall**	**F1 score**
Female	0.44	0.38	0.41
Male	0.91	0.93	0.92

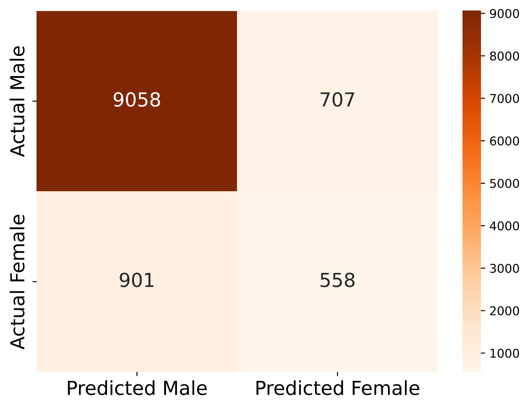


Figure 4.1: Confusion matrix of the classifier after imbalanced training ($t \geq 0.9$).

has a relatively low recall, indicating that the model misses 62% of the actual female pull requests, incorrectly predicting them as males'. Combining both precision and recall into one measure, the F1 score for the female class stands at 0.41.

On the other hand, we observed that the precision for the male class sits at 91%. This suggests that our XGBoost classifier attains a high degree of certainty in correctly identifying an element of the male class, given that a sample is deemed as being a pull request submitted by a male developer. A distinguishing factor, however, is the 93% recall obtained by the classifier when identifying pull requests submitted by males. In this case, we observe that substantial proportion of the actual male pull requests included in the dataset were captured by the model. The F1 score translates into 0.92, a large value given the high individual scores obtained in precision and recall.

Figure 4.1 shows the classifier's confusion matrix, providing valuable insights into the classifier's performance, by presenting a detailed breakdown of its predictions on the test set. We divided our dataset into 80% for training and 20% test set. In addition, we carefully maintained the proportional class distribution in the test set to ensure its representation of the original dataset.

The confusion matrix shows how identifying pull requests submitted by females is challenging for our model. The classifier struggled with correctly classifying female pull requests, as evidenced by the relatively low true positive count (TP=558). In addition, the relatively low precision value for the female class (0.44) indicates that only 44% of the pull requests predicted as female were actually female. On the other hand, the classifier shows impressive ability to correctly predict a substantial number of pull requests submitted by males (9058 out of 9765). The male class' high precision value (0.91) further corroborates this observation; 91% of the pull requests classified as male were indeed by male developers. Since the model exhibited a relatively low number of false negatives (707 out of 9765), where pull requests by males were incorrectly predicted as females', the model's proficiency in correctly classifying male pull requests is strengthened. Nonetheless, it exhibits a considerable number of misclassifications as it incorrectly identified pull requests made by females as belonging to the male class.

Trade-offs between precision and recall. So far, our results rely on a default internal threshold set in the XGBoost classifier. As such, studying the tradeoff between precision and recall is a critical aspect in evaluating the classifier's performance. To analyze the impact of varying this threshold, we plotted the precision-recall curve of the classifier in Figure 4.2. The figure illustrates that the classifier exhibits a precision of 0.60, indicating that when it predicts the class of a pull request as positive (submitted by a female developer), it is correct 60% of the time. Alternatively, the recall value of 0.25 signifies that the classifier can capture only 25% of the actual positive instances. As the recall increases beyond this point, precision tends to decrease. Consequently, we face a trade-off when setting the classifier's threshold. The area under the precision-recall curve is 0.39, which suggests that the classifier faces difficulties in achieving a balanced performance in terms of precision and recall.

When dealing with pull requests submitted by female developers, the selection of the classifier's threshold plays a pivotal role in balancing key aspects of performance. By setting a lower threshold, we can increase the number of correctly identified positive instances (higher recall) and capture most of the pull requests by females. However, this comes at the cost of reduced overall confidence in the classifier's predictions due to increased number of false positives.

Analysis of feature importance. Through the above experiments, we gained insights into the significance of various features in distinguishing between pull requests submitted by females and males. Leveraging the capabilities of the XGBoost classifier, we obtained a measure of each feature's impact for classification. Figure 4.3 provides a visual representation of the classifier's heavy reliance on these attributes when making predictions.

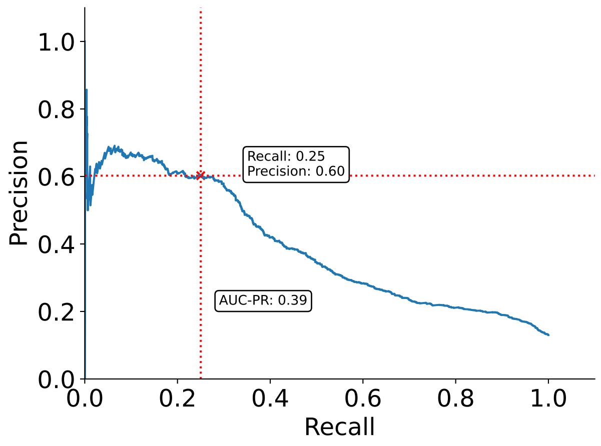


Figure 4.2: Precision-recall curve of the classifier after imbalanced training ($t \geq 0.9$).

The top ten most crucial features predominantly consist of attributes from the Zhang et al.[71] feature set (nine features), and one extracted from SonarQube. The results suggest that general pull request characteristics play a pivotal role in enabling the classifier to distinguish between pull requests submitted by female and male developers. To shed more light on the specific differences between the way male and female developers write code, Figure 4.4 presents the difference on per-class feature values for the three most important features identified by XGBoost. Namely, test inclusion (Figure 4.4(a)), the use of hash tags (Figure 4.4(b)), and whether the pull request was merged or not (Figure 4.4(c)). These figures suggest that there are some noticeable differences in the way female and male developers write their code. For instance, we can see that, proportionally, male developers tend to include a larger number of hash tags in their code, suggesting a more systematic utilization of comments compared to their female counterparts. Additionally, male developers tend to include more tests in their submitted code as compared to their female counterparts. This shows a difference in testing practices based on gender. This may be suggestive of diverse factors at play, such as variation in coding styles or approaches to ensuring code quality and reliability. Yet, it is important to note that this observation does not necessarily imply that one gender's coding style is better than the other's. Moreover, the discovery that whether a pull request is merged or not emerges as one of the most important features in our research on pull requests by female and male developers carries significant implications. It suggests that the outcome of whether a pull request gets merged has an influence on the classifier's ability to distinguish between coding contributions from female and male developers. This observation could reflect underlying gender-related differences in how pull

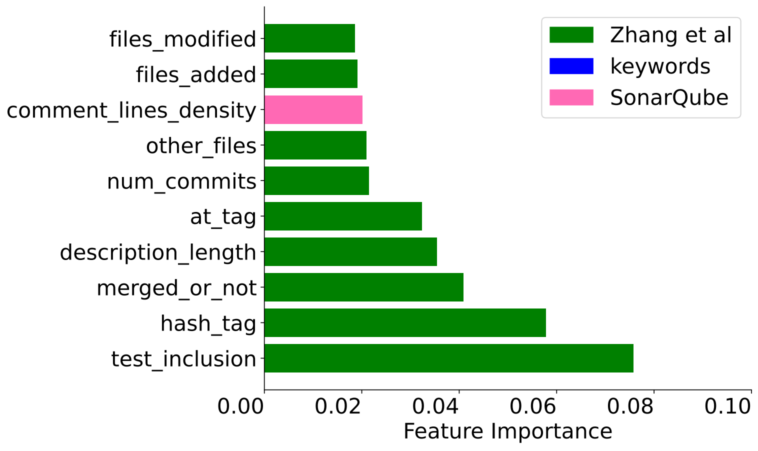
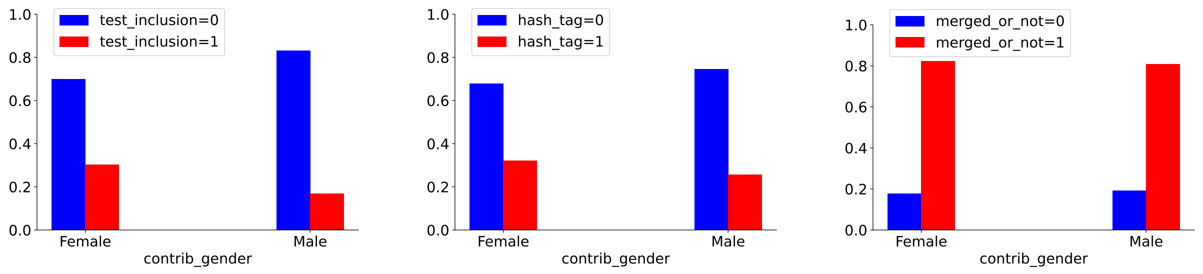


Figure 4.3: Top-10 most important features, as selected by the XGBoost classifier when applied to imbalanced data (after filtering with $t \geq 0.9$).



(a) test_inclusion (b) hash_tag (c) merged_or_not

Figure 4.4: Differences on per-class feature values' distributions for the top-3 most important features in the imbalanced dataset.

requests are evaluated, considered, or prioritized within the open-source community. The difference in the number of tests included could be influenced by multiple factors, including but not limited to project requirements and team dynamics. We note, however, that our limited data sample (i.e., four GitHub projects) precludes us from confirming that the above findings generally apply to the behavior of female and male developers across the whole OSS ecosystem. We discuss this limitation in Section 5, and suggest possible ways to enlarge the scope of our study in future work.

Table 4.3: Results of the XGBoost classifier on the balanced dataset.

Class	**Precision**	**Recall**	**F1 score**
Female	0.34	0.55	0.42
Male	0.93	0.84	0.88

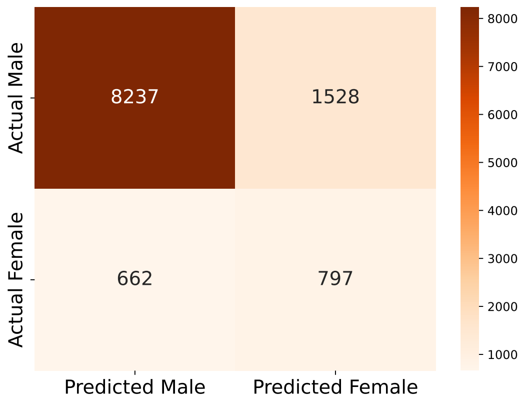


Figure 4.5: Confusion matrix of the classifier after balanced training set($t \geq 0.9$)

4.1.2 Analysis on Balanced Training Data

Overall classification performance. Upon training our XGBoost classifier on the balanced dataset of pull requests achieved through the implementation of Synthetic Minority Over-sampling Technique (SMOTE), we collected the same set of metrics used in Section 4.1 to assess the classifier's performance: precision, recall, and the F1 score. Our results are summarized in Table 4.3, and disclose the performance of the classifier when attempting to specifically identify the pull requests that were submitted by either females or males.

The classifier trained with the balanced dataset exhibits different set of strengths and weaknesses. Despite its low precision for females of 0.34, it compensates with a higher recall of 0.55. The previous demonstrates that the model with the balanced dataset successfully captures 55% of the actual female pull requests. The F1 Score of 0.42 implies a balanced performance between precision and recall for females. Overall, the model trained on a balanced dataset exhibits better recall for females, suggesting its advantage in correctly capturing a higher portion of actual female pull requests.

Given the emphasis on capturing more information about the female class, training the

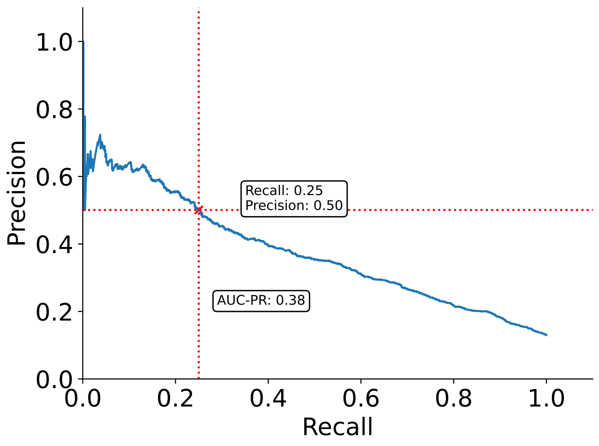


Figure 4.6: Precision-recall curve of the classifier after balanced training ($t \geq 0.9$).

classifier with a balanced dataset seems to be the more suitable choice. It has a better recall for females, allowing it to identify a greater number of female pull requests, which is significant when dealing with a minority class in an imbalanced dataset.

Figure 4.5 depicts the confusion matrix of the classifier, providing valuable insights into how well the classification model performed. For the female class, the model correctly predicted 797 pull requests made by females as females and misclassified 662 female pull requests as males. This suggest that the model was able to identify a significant number of actual female pull requests but also misclassified a considerable number of them as males.

On the other hand, for the male class, the model correctly predicted 8237 pull requests submitted by males. These findings illustrate the model's strong accuracy in correctly identifying male requests. Notably, the model exhibits better performance in handling male pull requests compared to female ones; the number of misclassifications is lower for males. This adds to our previous evidence that the classifier is limited in its ability to recognize certain patterns that may be specific to the female class.

Trade-offs between true positive and false positive rates. The tradeoff between precision and recall plays a critical role in explaining the classifier's performance under various circumstances. When giving priority to precision, the classifier becomes cautious in labeling positive instances, resulting in fewer false positives. However, it might miss some true positives (lower recall). Conversely, if recall takes precedence, the classifier becomes more lenient in predicting positive instances, which leads to higher recall but could also increase false positives (lower precision). Figure 4.6 demonstrates the trade-off

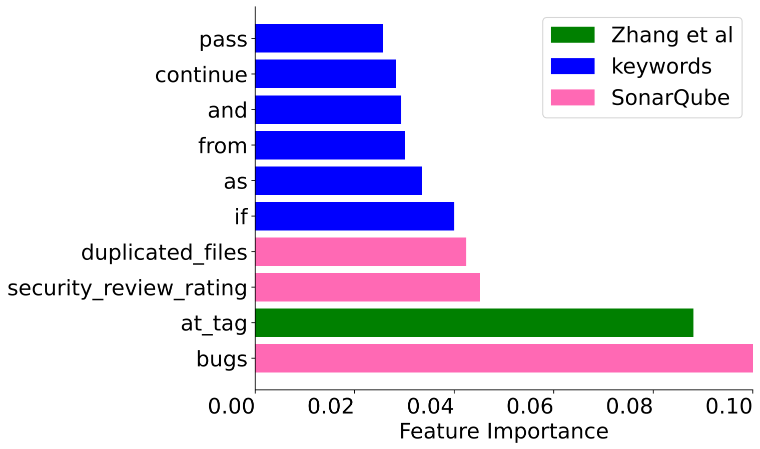
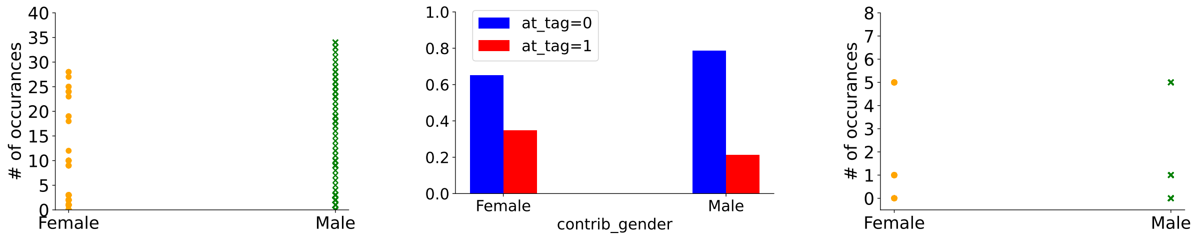


Figure 4.7: Top-10 most important features, as selected by the XGBoost classifier when applied to balanced data (after filtering with $t \geq 0.9$).



(a) bugs. (b) at_tag. (c) security_review_rating

Figure 4.8: Differences on per-class feature values' distributions for the top-3 most important features in the balanced dataset(after filtering with $t \geq 0.9$).

between the precision and recall obtained by the classifier, according to different thresholds. The figure shows when the classifier exhibits a precision value of 0.5, the recall stands at 0.25, suggesting that only 25% of actual positive instances were correctly identified by the classifier.

The precision-recall area under the curve provides a comprehensive evaluation of the classifier's performance at various thresholds (tradeoff points between precision and recall). An obtained area of 0.38 suggests that the classifier has moderate performance, signifying that the classifier possess some capacity to strike a balance between precision and recall.

Analysis of feature importance. Figure 4.7 highlights the top ten most important features that the classifier relies on to make predictions when trained over the balanced dataset. The top ten features include the count of some keywords (6 features), along with

3 features extracted from SonarQube, and 1 attribute from pull request-related feature set from Zhang et al. [71]. We can find in figures (Figure 4.8(a)), (Figure 4.8(b)), and (Figure 4.8(c)), that there are differences between how female and male developers code. We can see that male developers tend to have higher number of bugs in their code compared to female developers. This observation could be correlated to some factors, such as coding practices, experience level, or potential gender biases that influences code evaluation. Moreover, the higher prevalence of @ tags usage among male developers in Python compared to female ones suggests a potential gender-related differences in coding practices. This variation could be due to the difference in their level of experience, variations in confidence level when using complex features, and maybe the exposure to programming concepts in diverse learning environments.

4.2 Impacts of Feature Removal Threshold

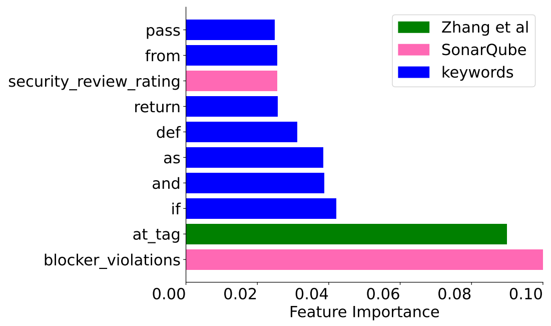
In this section, the primary focus is to discern whether such variations can yield significant differences in classification results. Also, we would like to obtain meaningful insights into the classifier's evaluation of feature importance while discerning pull requests submitted by male and female developers. In particular, our analysis involves evaluating the classifier's performance under different scenarios where specific features are removed based on the percentage of missing values across our dataset. We specifically explore the impact of removing features that have at least 20% or 75% of their values missing. Then, we compare these results with the previous findings obtained using a threshold of 90% or more missing values. We selected $t \geq 0.2$ and $t \geq 0.75$ after observing that removing features with a missing ratio of 75% left the remaining features with approximately 24% missing data. This suggests our interest in evaluating the classifier's performance when adopting a less stringent approach to eliminate attributes with lower proportions of missing values compared to the previous threshold Section 4.1.

For our analysis, we will focus on the balanced training dataset obtained via the SMOTE data oversampling technique. As seen Section 4.1.2, a model trained on balanced data provides a slightly higher effectiveness in correctly identifying the gender of pull requests submitted by females (the minority class). Further, as discussed in Section 4.1, we observed that changes in the feature removal threshold only affect the number of attributes considered from the code quality feature set.

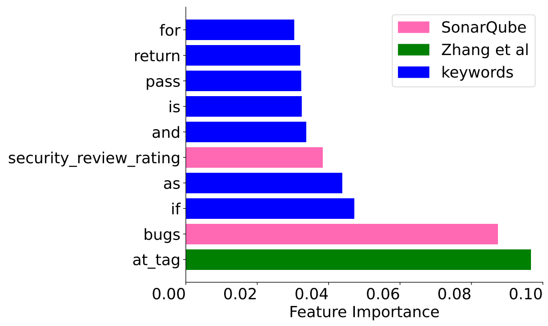
Overall classification performance. The classification metrics obtained by the classifier when we use the different feature removal thresholds are shown in Table 4.4. We can observe that the classification scores obtained when using different values of t are quite similar.

Table 4.4: Results of the XGBoost classifier on the balanced dataset at different thresholds.

Threshold	Class	Precision	Recall	F1 score
0.20	Female	0.36	0.50	0.42
	Male	0.92	0.87	0.89
0.75	Female	0.34	0.52	0.41
	Male	0.93	0.85	0.88
0.90	Female	0.34	0.55	0.42
	Male	0.93	0.84	0.88



(a) $t \geq$0.75



(b) $t \geq$0.20

Figure 4.9: Top-10 most important features, as selected by the XGBoost classifier when trained with balanced data (after feature removal).

Upon comparing the performance metrics of the different thresholds, we can see that results obtained from $t \geq 0.90$ exhibits certain strengths that make it a more favorable choice. While $t \geq 0.20$ and $t \geq 0.90$ share a similar F1 score of 0.42 for the females class, $t \geq 0.90$ stands out with a higher recall of 0.55, suggesting its ability to correctly identify a larger proportion of pull requests by females as true positives among all actual positive instances for females. This improved recall serves as evidence of the model proficiency in correctly identifying a larger number of instances from the female class. Furthermore, when it comes to the males class, $t \geq 0.90$ displays a precision value of 0.93, which means that a substantial majority of its predicted positive instances for males are accurate.

Analysis of feature importance. Figure 4.9 depicts the top ten most important features identified by the classifier when assuming a feature removal threshold of $t \geq 0.75$ (Figure 4.9(a)) and $t \geq 0.20$ (Figure 4.9(b)). We perform the following observations. First, we can observe that the usage of @ tags, the security review rating, and the frequency of some Python keywords, (such as if, and, as) remain in the top ten most important features, when considering the different thresholds ($t \geq 0.90$, $t \geq 0.75$, $t \geq 0.20$).

Second, we can observe that, in both cases, the top two most important features have a similar importance (close to 0.10), and which is close to double the importance attributed to the feature located in the top three. Interestingly, this trend is also observed when considering $t \geq 0.90$ (see Figure 4.7).

Lastly, we can see that the overall proportion of features drawn from each of the considered feature sets remains constant, despite the tuning of the feature removal threshold values. Specifically, we see that, for all cases, the top ten most important features consist of one feature from Zhang et al.'s dataset [71] (i.e., pull request characteristics), two code quality indicators extracted from SonarQube, and some attributes extracted tied to the frequency of Python keyword usage.

The fact that male developers use the Python keyword `if` more frequently may indicate that they rely more on branching in their code. On the other hand, female developers' lower usage of that keyword suggests that they might follow different control flow constructs.

4.3 Comparative Analysis of Feature Sets

In this section, we aim at understanding the individual contribution of each feature set to the ability of our classifier to distinguishing between pull requests submitted by male and female developers. The following paragraphs present the classification results and feature importance scores obtained when training our classifier with each of the feature sets we considered throughout this work. For each of the considered feature sets, we analyze the contribution of the most important features, as selected by the classifier, to gather additional insights on different aspects of how female and male developers write code.

Similarly to Section 4.2, we focus our analysis on the classifier trained with a balanced dataset obtained via SMOTE and perform our analysis when considering a feature removal threshold of $t \geq 0.90$ (found to be the best performing threshold we explored).

4.3.1 Feature Set 1 – Pull request characteristics

The pull request characteristics' feature set compromised a total of 17 attributes. Below, we describe the performance metrics obtained by the classifier when considering this feature set to infer whether a given pull request was submitted by a female or male developer.

Overall classification performance. Table 4.5 summarizes the classification results. Using the pull request characteristics feature set from [71], the model highlights notable disparities in its gender classification accuracy. It demonstrates a robust precision of 0.88 for pull requests by male developers, which signifies its ability to accurately identify a significant portion of male instances. However, its precision sharply declines to 0.19 for female instances, indicating a higher likelihood of misclassifying pull requests by females. This contrast is further underscored by the recall values, with males achieving a recall of 0.81, implying a successful capture of true male instances. For the female class, the model exhibits a lower recall of 0.3, which obviously indicates that the classifier is struggling in identifying actual female instances. Despite how impressive the F1 score of 0.84 for males is, which undoubtedly signifies a good compromise between precision and recall, the F1 score for females drops to 0.23, revealing an uneven tradeoff between precision and recall. Overall, the following model excels in identifying pull requests by males but faces challenges in accurately identifying female instances, leading to an overall performance disparity between the genders.

Analysis of feature importance. The feature that stands out as the most crucial in this model is the usage of the @ tag, ranking among the top 10 most significant features in all other models. The previous indicates that the model heavily relies on this particular feature to make gender-based predictions. This suggests that there might be differences in how female and male developers code in terms of structure. As a result, the classifier has learned to pick up on these patterns. Yet, it is absolutely essential to interpret such results cautiously to avoid drawing broad conclusions about gender differences in coding based solely on the model's performance. Further analysis and investigation are required to understand the underlying reasons for why the classifier emphasises on the @ tag.

Moreover, the utilization of the hash tag is a prevalent and significant feature shared between this model and the other model in Figure 4.3. The shared importance of this feature suggests its relevance across different models, potentially indicating that it is essential in distinguishing between how female and male developers code.

For this classifier, whether a pull request is merged or not is one of the top 3 most important features in distinguishing between female and male developers. This findings implies that there might be some possible differences in how female and male developers' pull requests are handled or evaluated in OSS community.

4.3.2 Feature Set 2 – Code Quality Indicators

This feature set includes a total of 42 attributes from our code quality indicators feature set, after filtering the initial feature set with a feature removal threshold of $t \geq 0.90$.

Table 4.5: Results of the classifier on the pull requests' characteristics feature set.

Class	Precision	Recall	F1 score
Female	0.19	0.30	0.23
Male	0.88	0.81	0.84

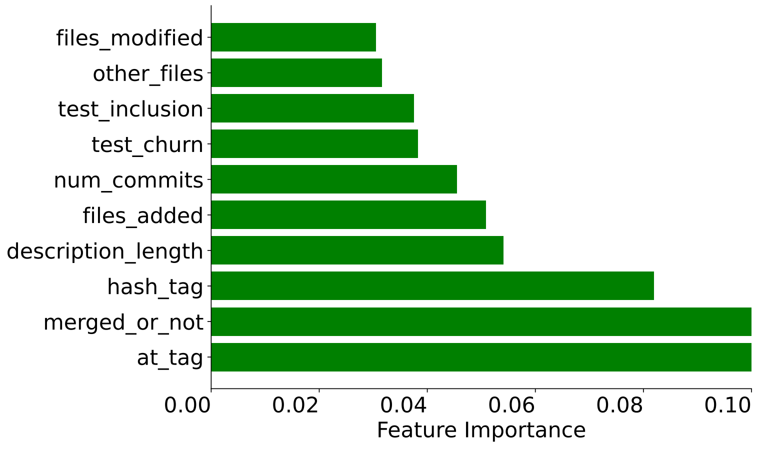


Figure 4.10: Top-10 most important features, as selected by the XGBoost classifier when applied to balanced data.

Overall classification performance. Table 4.6 summarizes the classification results. Regarding pull requests by females, only 15% of the pull requests predicted as female are actually female pull requests. Also, the model shows success in identifying 17% of all actual female pull requests present in the test set. The F1 score of 0.16 for the female class implies that the model is limited in classifying female pull requests.

For pull requests by males, 87% of pull requests predicted as male are, in fact, male pull requests. Similarly, the recall of 0.87 shows that the classifier accurately identify 87% of all the actual male pull requests present in the test set. The F1 score of 0.87 confirms the model's great performance in classifying male requests.

Analysis of feature importance. Figure 4.11 highlights the top ten most important features that the classifier relies on to predict the gender of the source code author, using the code quality indicators feature set. The following features have been extracted from SonarQube. The three most important features in this model are security review rating, the number of files, and file complexity. The security review rating serves as a measure

Table 4.6: Results of the classifier on the code quality indicators' feature set ($t \geq 0.90$).

Class	**Precision**	**Recall**	**F1 score**
Female	0.15	0.17	0.16
Male	0.87	0.87	0.87

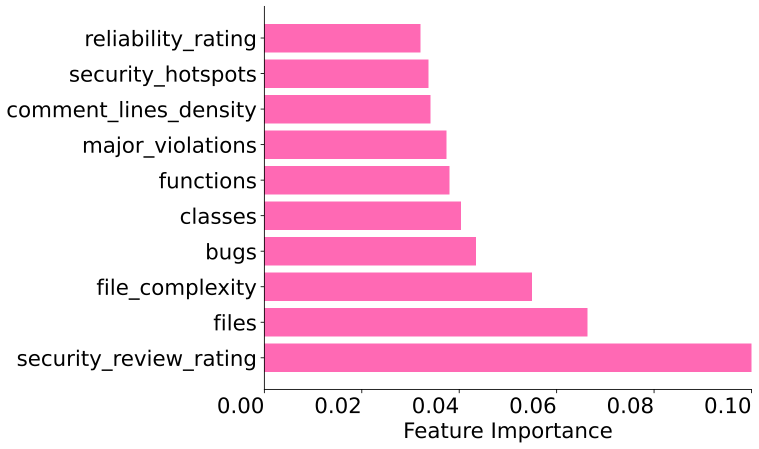


Figure 4.11: Top-10 most important features, as selected by the XGBoost classifier when applied to balanced SonarQube feature set (after filtering with $t \geq 0.90$).

of the code's security, and it suggests potential gender-related differences in the level of security scrutiny applied to code changes. The number of files provides insights into the breadth of a developer's contributions, which could potentially hold some gender-related differences too. File complexity is the average complexity of the files included, quantifies using cyclomatic complexity. File complexity feature could indicate gender-related patterns in how developers approach code organization.

4.3.3 Feature Set 3 – Keyword Frequency

In order to capture a broader range of characteristics from the source code of pull requests, we computed the frequency of 35 different *Python keywords* included in each pull request.

Overall classification performance. Let's break down the model performance for female pull requests. The precision of 0.12 shows that when the model predicts a pull request

Table 4.7: Results of the classifier on the keyword frequency feature set.

Class	Precision	Recall	F1 score
Female	0.12	0.51	0.19
Male	0.85	0.43	0.57

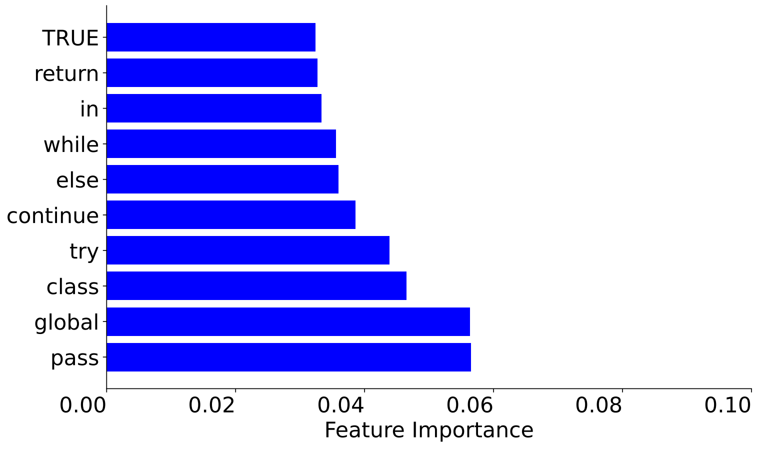


Figure 4.12: Top-10 most important features, as selected by the XGBoost classifier when applied to balanced data.

as a female's, it is correct only 12% of the time. The recall score tells us that the model manages to spot 51% of the actual female pull requests in the test set. The previous shows that the classifier struggles to accurately identify pull requests by females and might miss a substantial proportion of them.

Now let's take a look at how the model performs for pull requests by male developers. The precision value of 0.85 shows that the model is right 85% of the time when predicting a pull request as male's. The recall score of 0.43 means that the model accurately spots 43% of all the male pull request in the test set. Considering the previous, the model performance in classifying male pull requests is comparatively better than its performance in identifying female pull requests.

Analysis of feature importance. Figure 4.12 highlights the top ten most important features that the classifier relies on to predict the gender of the source code author, using the keywords frequency feature set. The top three important Python keywords: `pass`, `global`, and `class` provide some insights into how the classifier learned to identify source code by female and male developers. The presence of `pass` keyword could potentially indicate areas where the developer creates some stubs that would need further implementation. Developers use `global` to declare variables that are accessible throughout the entire code. Since the classifier relies on this keyword, among others, to distinguish between how female and male developers code, investigating how developers use it might reveal some differences in how different genders manage program-wide data. Lastly, the appearance of the Python keyword `class` could infer a difference in how female and male developers utilize object-oriented programming concepts.

In conclusion, our comparative analysis of three distinct sets of features for classifying pull requests originating from male and female developers offers intriguing glimpses into potential gender-related coding tendencies. The classifier consistently excels in accurately identifying pull requests authored by males, exhibiting high precision and recall rates. However, its performance exhibits variations when discerning pull requests from female authors, often displaying lower precision and recall scores. The examined features encompass a spectrum from pull request attributes to code quality indicators and keyword frequencies, revealing potential dissimilarities in coding styles, behaviors, and practices between genders. Noteworthy, certain features stand out as pivotal discriminators, such as the utilization of specific tags, security review evaluations, and the occurrence of keywords like `pass` and `class.`

Chapter 5

Discussion and Conclusions

5.1 Contributions

This thesis delved into the gender discrepancies that have been recognized for a considerable time as a challenge within the open-source software (OSS) community. Existing literature and documented findings suggest that the involvement of female developers in the OSS realm has consistently faced persistent marginalization and prejudice. This bias has often been presumed to stem from gender-related factors.

In this document, we presented a comprehensive investigation to shed the light on potential disparities in coding practices between female and male developers. Our objective was to comprehend whether substantive distinctions exist and to investigate any possible underlying factors contributing to the higher rate of rejection for pull requests from female contributors compared to their male counterparts. We curated a dataset that encompasses diverse dimensions of code style and quality. This careful compilation provided us with the means to undertake a more comprehensive investigation into the unique coding approaches utilized by female and male developers. We elaborate on potential approaches to expand this dataset and undertake a thorough analysis of coding behaviour across the vast expanse of the open-source software ecosystem.

As we conducted a more thorough exploration into the coding patterns exhibited by female and male developers, our investigation unveiled significant distinctions that may potentially contribute to the observed variations in how pull requests are accepted. Notably, we identified that source code by male developers displayed a higher incidence of bugs, alongside an elevated occurrence of blocker issue, as compared to code written by female developers. This intriguing insight challenges preconceived notions and disrupts the narrative that coding quality is inherently linked to gender. Similarly, the diverse employment of Python keywords emerged as a captivating facet, with females demonstrating a tendency to utilize `if` and @ tag keywords less frequently, thereby introducing an additional layer of intricacy to the multifaceted landscape of coding practices.

5.2 Recommendations for the OSS community

Given the results of our study, we draft a set of recommendations that may positively contribute to addressing gender disparities amongst the OSS community.

1. **Encouraging the development of gender-annotated datasets for comprehensive analysis of coding patterns.** We find that the gender-annotated dataset curated by Zhang et al. [71], along with features extracted from SonarQube, allowed us to perform a meaningful analysis of which stylistic and code quality features can help us identify the gender of pull request submitters. To further enhance this dataset's richness, we propose that after a decision has been made regarding the merging of a pull request, the submitter could be asked, on a voluntary basis, to provide their gender. Initiatives such as this one should be encouraged, as similar datasets can help researchers build a more complete understanding of gender-based coding patterns in the open-source community.

2. **Leveraging automated code analysis.** We should explore the integration of automated code analysis tools to impartially evaluate the technical merit of pull requests. This additional layer of evaluation will be uninfluenced by gender, so it can elevate the consistency of the review process.

3. **Fostering gender-neutral stylistic coding guidelines.** Our analysis suggested that code stylistic features may help one distinguish whether a given pull request has been submitted by a male or female developer. To prevent the use of similar analysis procedures to reinforce the prevalence of accepted contributions put forth by developers of a single gender, the OSS community could strive to enforce stylistic coding guidelines. These guidelines would aim at reducing disparities between the way individuals from different genders write code, thus contributing to the reduction of gender-related biases.

4. **Enhancing reviewer standards and guidelines.** We can create a comprehensive training initiatives aimed at cultivating reviewers' awareness of unconscious biases and gender-related assumptions.

5. **Collaborative coding hackathons.** We suggest organizing some collaborative hackathons and coding workshops to facilitate the convergence of developers from diverse gender backgrounds to work on projects collectively. Such events can help bridge the gap in coding styles while encourage knowledge sharing.

6. **Promoting inclusive project cultures through proposing outreach initiatives.** The OSS community could put forward a set of outreach initiatives to explicitly seek contributions from underrepresented groups, thus contributing to the equity, diversity, and inclusion of individuals across the vast spectrum of open-source software.

5.3 Limitations and Future Work

A key limitation of our study is that our analysis has been conducted over a limited set of four Python-based repositories on GitHub. While the over fifty thousand pull requests under analysis spanned the contributions of over forty thousand developers, our data sample might still be considered too narrow to encompass an analysis that is fully representative of the entire OSS ecosystem. For instance, our study overlooks different pieces of code that may a) have been written for other specific purposes; b) that are written in different programming languages; c) that focus repositories with a small and tightly knit contributor base, or; d) that include a potentially larger diversity of contributors. In light of the above, an interesting direction for future work would be that of extending our study in all the aforementioned dimensions, for instance, scraping pull requests across a larger span of repositories curated by Zhang et al. [71].

An additional constraint inherent in our study pertains to the finite array of analytical tools we employed to extract potential features from the code within pull requests. It is noteworthy to emphasize that our investigation did not encompass the utilization of a specific category of code stylometry tools, which are predicated upon the extraction and analysis of abstract syntax trees, as highlighted in the work by Caliskan-Islam et al.[12]. These tools have demonstrated notable efficacy within the domain of code authorship attribution, showcasing their ability to unravel the distinctive nuances embedded in coding styles and contributing to the identification of code creators. While we acknowledge the potential insights that could have arisen from integrating these tools, their omission underscores a focused exploration of other facets of the pull request analysis process in our study.

Moreover, the limitation concerning gender complexity within the study pertains to the binary categorization of gender into female and male might fail to adequately capture the intricate diversity and multifaceted nature of gender identities, which exist within the developer community. One of the contributing factors to this binary classification is the absence of a dataset that includes gender labels beyond the traditional binary classification. By exclusively relying on this binary framework, this study unintentionally overlooks the contributions and experiences of individuals, who identify as non-binary and gender-noncomforming. Neglecting this dimension not only constrains a comprehensive exploration of coding styles, but also sustains an incomplete understanding of the full spectrum of gender diversity in software development.

Finally, we observed that one interesting attribute extracted by the SonarQube analysis tool is related to the security review rating of a piece of code. While we did not delve in depth on the kind of security bugs present in the analyzed pull request code, it is possible that the kind and number of vulnerabilities introduced by male and female developers may differ, providing further information that may help a classifier distinguish code written by individuals of different gender. A potential direction for future work would be to collect a set of prominent static code analysis tools focused on finding security vulnerabilities, and leverage their reports to build an additional set of features that could also be used for enriching our analysis methodology.

References

[1] Andrea E Abele and Bogdan Wojciszke. Communal and agentic content in social cognition: A dual perspective model. In *Advances in experimental social psychology*, volume 50, pages 195–255. Elsevier, 2014.

[2] Edgar Acuna and Caroline Rodriguez. The treatment of missing values and its effect on classifier accuracy. In *Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification Societies (IFCS), Illinois Institute of Technology, Chicago, 15–18 July 2004*, pages 639–647. Springer, 2004.

[3] Faith Amuchi, Ameer Al-Nemrat, Mamoun Alazab, and Robert Layton. Identifying cyber predators through forensic authorship analysis of chat logs. In *2012 Third Cybercrime and Trustworthy Computing Workshop*, pages 28–37. IEEE, 2012.

[4] Shlomo Argamon, Moshe Koppel, Jonathan Fine, and Anat Rachel Shimoni. Gender, genre, and writing style in formal written texts. *Text & talk*, 23(3):321–346, 2003.

[5] Maya Bar-Hillel. The base-rate fallacy in probability judgments. *Acta Psychologica*, 44(3):211–233, 1980.

[6] Shaowen Bardzell. Feminist hci: taking stock and outlining an agenda for design. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1301–1310, 2010.

[7] Sandra L Bem. The measurement of psychological androgyny. *Journal of consulting and clinical psychology*, 42(2):155, 1974.

[8] Valeria Borsotti. Barriers to gender diversity in software development education: actionable insights from a danish case study. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, pages 146–152, 2018.

[9] Amiangshu Bosu and Kazi Zakia Sultana. Diversity and inclusion in open source software (oss) projects: Where do we stand? In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE, 2019.

[10] Steven Burrows and Seyed MM Tahaghoghi. Source code authorship attribution using n-grams. In *Proceedings of the twelth Australasian document computing symposium, Melbourne, Australia, RMIT University*, pages 32–39. Citeseer, 2007.

[11] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian*, 17(1):22, 2018.

[12] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 255–270, 2015.

[13] Edna Dias Canedo, Rodrigo Bonifácio, Márcio Vinicius Okimoto, Alexander Serebrenik, Gustavo Pinto, and Eduardo Monteiro. Work practices and perceptions from women core developers in oss communities. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2020.

[14] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[15] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[16] Kim B Clark. Product development performance: Strategy. *Organization, and Management in the World Auto Industry*, 1991.

[17] Jody Clarke-Midura, Frederick Poole, Katarina Pantic, Megan Hamilton, Chongning Sun, and Vicki Allan. How near peer mentoring affects middle school mentees. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 664–669, 2018.

[18] Matthew A Cronin and Laurie R Weingart. Representational gaps, information processing, and conflict in functionally diverse teams. *Academy of management review*, 32(3):761–773, 2007.

[19] Edwin Dauber, Rebekah Overdorf, and Rachel Greenstadt. Stylometric authorship attribution of collaborative documents. In *Cyber Security Cryptography and Machine Learning: First International Conference, CSCML 2017, Beer-Sheva, Israel, June 29-30, 2017, Proceedings 1*, pages 115–135. Springer, 2017.

[20] Helene De Ribaupierre, Kathryn Jones, Fernando Loizides, and Yulia Cherdantseva. Towards gender equality in software engineering: the nsa approach. In *Proceedings of the 1st International Workshop on Gender Equality in Software Engineering*, pages 10–13, 2018.

[21] Natália Pinheiro Ramos de Souza and Kiev Gama. Diversity and inclusion: Culture and perception in information technology companies. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 15(4):352–361, 2020.

[22] Christian Delcourt. Stylometry. *Revue belge de philologie et d'histoire*, 80(3):979–1002, 2002.

[23] AH Eagly. Sex differences in social behavior: a social role interpretation. psychology press, 1987.

[24] Joyce Ehrlinger, E Ashby Plant, Marissa K Hartwig, Jordan J Vossen, Corey J Columb, and Lauren E Brewer. Do gender differences in perceived prototypical computer scientists and engineers contribute to gender gaps in computer science and engineering? *Sex roles*, 78:40–51, 2018.

[25] Kathleen M Eisenhardt and Behnam N Tabrizi. Accelerating adaptive processes: Product innovation in the global computer industry. *Administrative science quarterly*, pages 84–110, 1995.

[26] Anirudh Ekambaranathan. Using stylometry to track cybercriminals in darknet forums. Master's thesis, University of Twente, 2018.

[27] Georgia Frantzeskou, Stephen MacDonell, Efstathios Stamatatos, and Stefanos Gritzalis. Examining the significance of high-level programming features in source code author classification. *Journal of Systems and Software*, 81(3):447–460, 2008.

[28] Lex Fridman, Steven Weber, Rachel Greenstadt, and Moshe Kam. Active authentication on mobile devices via stylometry, application usage, web browsing, and gps location. *IEEE Systems Journal*, 11(2):513–521, 2016.

[29] Rishab A Ghosh, Ruediger Glott, Bernhard Krieger, and Gregorio Robles. Free/libre and open source software: Survey and study. 2002.

[30] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[31] Stefan Gruner and Stuart Naven. Tool support for plagiarism detection in text documents. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 776–781, 2005.

[32] Jonathan Heawood. Pseudo-public political speech: Democratic implications of the cambridge analytica scandal. *Information polity*, 23(4):429–434, 2018.

[33] Amaç Herdağdelen and Marco Baroni. Stereotypical gender actions can be extracted from web text. *Journal of the American Society for Information Science and Technology*, 62(9):1741–1749, 2011.

[34] Cedric Herring. Does diversity pay?: Race, gender, and the business case for diversity. *American sociological review*, 74(2):208–224, 2009.

[35] David I Holmes. A stylometric analysis of mormon scripture and related texts. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 155(1):91–120, 1992.

[36] Sander Hoogendoorn, Hessel Oosterbeek, and Mirjam Van Praag. The impact of gender diversity on the performance of business teams: Evidence from a field experiment. *Management science*, 59(7):1514–1528, 2013.

[37] Sujin K Horwitz and Irwin B Horwitz. The effects of team diversity on team outcomes: A meta-analytic review of team demography. *Journal of management*, 33(6):987–1015, 2007.

[38] Farkhund Iqbal. *Messaging forensic framework for cybercrime investigation*. PhD thesis, Concordia University, 2011.

[39] Farkhund Iqbal, Hamad Binsalleeh, Benjamin CM Fung, and Mourad Debbabi. Mining writeprints from anonymous e-mails for forensic investigation. *digital investigation*, 7(1-2):56–64, 2010.

[40] Vaibhavi Kalgutkar, Ratinder Kaur, Hugo Gonzalez, Natalia Stakhanova, and Alina Matyukhina. Code authorship attribution: Methods and challenges. *ACM Computing Surveys (CSUR)*, 52(1):1–36, 2019.

[41] Robert T Keller. Cross-functional project groups in research and new product development: Diversity, communications, job stress, and outcomes. *Academy of management journal*, 44(3):547–555, 2001.

[42] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[43] Jonathan J. Koehler. The base rate fallacy reconsidered: Descriptive, normative, and methodological challenges. *Behavioral and Brain Sciences*, 19(1):1–17, 1996.

[44] Victor Kuechler, Claire Gilbertson, and Carlos Jensen. Gender differences in early free and open source software joining process. In *Open Source Systems: Long-Term Sustainability: 8th IFIP WG 2.13 International Conference, OSS 2012, Hammamet, Tunisia, September 10-13, 2012. Proceedings 8*, pages 78–93. Springer, 2012.

[45] Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[46] Kathleen J Lehman, Linda J Sax, and Hilary B Zimmerman. Women planning to major in computer science: Who are they and what makes them unique? *Computer Science Education*, 26(4):277–298, 2016.

[47] Ting-Peng Liang, James Jiang, Gary S Klein, and Julie Yu-Chih Liu. Software quality as influenced by informational diversity, task conflict, and learning in project teams. *IEEE Transactions on Engineering Management*, 57(3):477–487, 2009.

[48] Xiaoguang Lu, Hong Chen, and Anil K Jain. Multimodal facial gender and ethnicity identification. In *Advances in Biometrics: International Conference, ICB 2006, Hong Kong, China, January 5-7, 2006. Proceedings*, pages 554–561. Springer, 2005.

[49] Abby L Mello and Joan R Rentsch. Cognitive diversity in teams: A multidisciplinary review. *Small Group Research*, 46(6):623–658, 2015.

[50] C Chet Miller, Linda M Burke, and William H Glick. Cognitive diversity among upper-echelon executives: implications for strategic decision processes. *Strategic management journal*, 19(1):39–58, 1998.

[51] Dawn Nafus. 'patches don't have gender': What is not open in open source software. *New Media & Society*, 14(4):669–683, 2012.

[52] A Omar and BD Aldawsari. Towards a linguistic stylometric model for the authorship detection in cybercrime investigations. *International Journal of English Linguistics*, 9(5):182–192, 2019.

[53] Lisa Hope Pelled, Kathleen M Eisenhardt, and Katherine R Xin. Exploring the black box: An analysis of work group diversity, conflict and performance. *Administrative science quarterly*, 44(1):1–28, 1999.

[54] Whitney E Powell, D Scott Hunsinger, and B Dawn Medlin. Gender differences within the open source community: An exploratory study. *Journal of Information Technology*, 21(4):29–37, 2010.

[55] James Brian Quinn. Managing innovation: controlled chaos: harvard business review. *Harvard Business Review*, 2(4):485, 1987.

[56] Hoshiladevi Ramnial, Shireen Panchoo, and Sameerchand Pudaruth. Authorship attribution using stylometry and machine learning techniques. In *Intelligent Systems Technologies and Applications: Volume 1*, pages 113–125. Springer, 2016.

[57] Esther Ruiz Ben. Defining expertise in software development while doing gender. *Gender, Work & Organization*, 14(4):312–332, 2007.

[58] SonarSource. Sonarqube documentation, 2023.

[59] Janet T Spence. Gender-related traits and gender ideology: evidence for a multifactorial theory. *Journal of personality and social psychology*, 64(4):624, 1993.

[60] Benno Stein, Nedim Lipka, and Peter Prettenhofer. Intrinsic plagiarism analysis. *Language Resources and Evaluation*, 45:63–82, 2011.

[61] Ariel Stolerman, Rebekah Overdorf, Sadia Afroz, and Rachel Greenstadt. Breaking the closed-world assumption in stylometric authorship attribution. In *Advances in Digital Forensics X: 10th IFIP WG 11.9 International Conference, Vienna, Austria, January 8-10, 2014, Revised Selected Papers 10*, pages 185–205. Springer, 2014.

[62] Sayma Sultana. Identification and mitigation of gender biases to promote diversity and inclusion among open source communities. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.

[63] Sayma Sultana, Asif Kamal Turzo, and Amiangshu Bosu. Code reviews in open source projects: How do gender biases affect participation and outcomes? *arXiv preprint arXiv:2210.00139*, 2022.

[64] Eric Sundstrom, Kenneth P De Meuse, and David Futrell. Work teams: Applications and effectiveness. *American psychologist*, 45(2):120, 1990.

[65] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, 3:e111, 2017.

[66] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. Gender, representation and online participation: A quantitative study. *Interacting with Computers*, 26(5):488–511, 2014.

[67] Anna Vitores and Adriana Gil-Juárez. The trouble with 'women in computing': a critical examination of the deployment of research on the gender gap in computer science. *Journal of Gender Studies*, 25(6):666–680, 2016.

[68] Yi Wang and David Redmiles. Implicit gender biases in professional software development: An empirical study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 1–10. IEEE, 2019.

[69] Joan C Williams. Hacking tech's diversity problem. *Harvard Business Review*, 92(10):94–100, 2014.

[70] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*, pages 590–604. IEEE, 2014.

[71] Xunhui Zhang, Ayushi Rastogi, and Yue Yu. On the shoulders of giants: A new dataset for pull-based development research. In *Proceedings of the 17th international conference on mining software repositories*, pages 543–547, 2020.

[72] Yu Zhang, Rastogi. New pull request dataset (new_pullreq_msr2020), 2020. Last accessed on 07-25-2023.

[73] Zhang, Rastogi, Yu. Zhang et al. restricted dataset, 2020.

APPENDICES

Appendix A

Details on Feature Sets

A.1 Pull requests' characteristics

Listing A.1 provides a complete listing of the 17 features composing our pull requests' characteristics feature set, as curated from the overall set of attributes considered in the pull-based development study of Zhang et al. [71].

Listing A.1: Pull requests' characteristics features

```
1 - merged_or_not
2 - num_commits
3 - src_churn
4 - test_churn
5 - files_added
6 - files_deleted
7 - files_modified
8 - files_changed
9 - src_files
10 - doc_files
11 - other_files.
12 - churn_addition
13 - churn_deletion
14 - hash_tag
15 - at_tag
16 - test_inclusion
17 - description_length
```

A.2 Code quality indicators

Listing A.2 provides a complete listing of the 94 features composing our feature set based on code quality indicators, as extracted from the SonarQube static analysis tool.

Listing A.2: SonarQube features

```
1 - new_technical_debt
2 - blocker_violations
3 - bugs
4 - classes
5 - code_smells
6 - cognitive_complexity
7 - comment_lines
8 - comment_lines_density
9 - comment_lines_data
10 - class_complexity
11 - file_complexity.
12 - function_complexity
13 - complexity_in_classes
14 - complexity_in_functions
15 - branch_coverage
16 - new_branch_coverage
17 - conditions_to_cover
18 - new_conditions_to_cover
19 - confirmed_issues
20 - coverage
21 - new_coverage
22 - critical_violations
23 - complexity
24 - development_cost
25 - new_development_cost
26 - directories
27 - duplicated_blocks
28 - new_duplicated_blocks
29 - duplicated_files
30 - duplicated_lines
31 - duplicated_lines_density
32 - new_duplicated_lines_density
33 - new_duplicated_lines
34 - duplications_data
35 - effort_to_reach_maintainability_rating_a
36 - executable_lines_data
37 - false_positive_issues
38 - file_complexity_distribution
39 - files
40 - function_complexity_distribution
41 - functions
42 - generated_lines
43 - generated_ncloc
44 - info_violations
45 - violations
46 - line_coverage
47 - new_line_coverage
48 - lines
49 - ncloc
```

```
50 - ncloc_language_distribution
51 - lines_to_cover
52 - new_lines_to_cover
53 - sqale_rating
54 - new_maintainability_rating
55 - major_violations
56 - minor_violations
57 - ncloc_data
58 - new_blocker_violations
59 - new_bugs
60 - new_code_smells
61 - new_critical_violations
62 - new_info_violations
63 - new_violations
64 - new_lines
65 - new_major_violations
66 - new_minor_violations
67 - new_security_hotspots
68 - new_vulnerabilities
69 - open_issues
70 - projects
71 - public_api
72 - public_documented_api_density
73 - public_undocumented_api
74 - alert_status
75 - reliability_rating
76 - new_reliability_rating
77 - reliability_remediation_effort
78 - new_reliability_remediation_effort
79 - reopened_issues
80 - security_hotspots
81 - security_hotspots_reviewed
82 - new_security_hotspots_reviewed
83 - security_rating
84 - new_security_rating
85 - security_remediation_effort
86 - new_security_remediation_effort
87 - security_review_rating
88 - new_security_review_rating
89 - security_hotspots_reviewed_status
90 - new_security_hotspots_reviewed_status
91 - security_hotspots_to_review_status
92 - new_security_hotspots_to_review_status
93 - skipped_tests
94 - statements
```

A.3 Keyword frequencies

Listing A.3 provides a complete listing of the 35 features composing our feature set based on the usage frequency of the Python language keywords by each developed.

Listing A.3: Pythom Keywords features

```
1 - FALSE
2 - TRUE
3 - None
4 - and
5 - as
6 - assert
7 - async
8 - await
9 - break
10 - class
11 - continue
12 - def
13 - del
14 - elif
15 - else
16 - except
17 - finally
18 - for
19 - from
20 - global
21 - if
22 - import
23 - in
24 - is
25 - lambda
26 - nonlocal
27 - not
28 - or
29 - pass
30 - raise
31 - return
32 - try
33 - while
34 - with
35 - yield
```