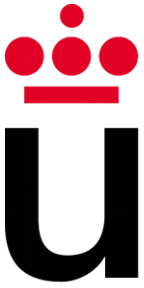

Universidad
Rey Juan Carlos

MÁSTER EN DATA SCIENCE

Curso Académico 2022/2023

Trabajo Fin de Máster

AUTOMATIC IDENTIFICATION OF BOT ACCOUNTS IN OPEN-SOURCE PROJECTS

Autor : Miguel Ángel Fernández Sánchez
Tutor : Dr. Felipe Ortega Soto

Trabajo Fin de Máster

Identificación Automática de Cuentas *bot* en Proyectos *Open-Source*.

Autor : Miguel Ángel Fernández Sánchez
Tutor : Dr. Felipe Ortega Soto

La defensa del presente Trabajo Fin de Máster se realizó el día XX de de 2023, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Madrid/Móstoles/Fuenlabrada, a de de 2023

A mi familia y amigos,
gracias por vuestro apoyo.

To my family and friends,
thank you for your support.

Acknowledgements

I want to thank my classmates from this Master's degree for their help and support during all the courses and through this final phase, especially Edgli, Enrique, and David. The COVID-19 pandemic arrived in our lives in the middle of our degree, so here goes a special recognition for them and the Master's professors for the extra aid and cooperation.

To my family and friends, who stood by my side all this time, bearing with me after endless promises of finishing this thesis once and for all. Your support kept me confident in the most challenging times. Specially, I want to thank my friend Quan, who also helped me solving technical issues and questions during the project.

To my tutor, Dr. Felipe Ortega, for accepting such a challenge and for the great help and guidance he provided me during this process with a lot of patience, great pieces of advice, and wise teachings. This encouraged me to keep improving and growing academically and personally throughout this project and beyond.

Last but not least, to Bitergia for supporting me in the course of this Master's degree; and also to Professor Tom Mens, Professor Alexandre Decan, and PhD student Mr. Mehdi Golzadeh from the University of Mons (Belgium); for guiding me during the early stages of this project with their generous ideas and knowledge.

Summary

People participating in software projects (in particular, in Free, Open-Source projects) rely on many tools and platforms to support their activity on many facets, such as code review or bug management. Within this scenario, automatic accounts (also known as *bot* accounts) are commonly used in software development to automate and ease repetitive or particular tasks.

Identifying these bot accounts and their activity in the projects is crucial for anyone willing to measure many aspects of the software project and the community of contributors behind it. *GrimoireLab* is a tool that provides metrics about the software development process, including a component to manage the contributors' identities, with an option to mark individual profiles as bots. Nonetheless, this labelling process is entirely manual.

In this MSc thesis, a *Python* tool to detect bots automatically based on their profiles' information and their activity in the project is developed. This tool can be integrated as a component inside the *GrimoireLab* toolchain. To this aim, we analysed the code changes from a set of software projects from the Wikimedia Foundation, produced between January 2008 and September 2021 using GrimoireLab, labelling manually the bot accounts generating activity with the purpose of creating an input dataset to train a binary classifier to detect whether a given profile is a bot or not.

After testing different classification models using the *Scikit-learn* module for *Python*, the model that performed best was a "Random Forest" classifier, where the most relevant features were a terms score calculated based on domain-related heuristics and statistical values obtained from the individuals' activity, such as number of changes in source code or number of words and files per code change submitted to the projects.

Resumen

Las personas que participan en proyectos de *software* (y en paricular en proyectos de *software* libre y código abierto), se apoyan en varias herramientas y plataformas para interactuar y tratar con diferentes aspectos de estos proyectos, tales como la revisión de código o la gestión de errores o *bugs*. En este contexto, las cuentas automáticas (también conocidas como cuentas *bot*) se usan frecuentemente en el desarrollo de software para automatizar y simplificar ciertas tareas repetitivas o específicas.

Para cualquier persona interesada en medir ciertos aspectos de un proyecto de software y de la comunidad de personas que lo sustenta, es crucial identificar estas cuentas *bot* y su actividad. *GrimoreLab* es una herramienta que proporciona métricas sobre el proceso de desarrollo de *software*, que incluye un componente para la gestión de los perfiles de contribuidores. Dicho componente cuenta con una opción para marcar aquellos perfiles que pertenezcan a una cuenta *bot*. Sin embargo, este proceso de etiquetado es enteramente manual.

En este Trabajo de Fin de Máster se propone una herramienta desarrollada en *Python* para detectar automáticamente cuentas bot, integrable como un componente dentro de *GrimoireLab*, utilizando como base la información de los perfiles de los diferentes individuos y de su actividad en el proyecto analizado. Para desarrollar esta herramienta se han analizado con *GrimoireLab* los cambios en el código de un conjunto de proyectos de *software* de la Fundación Wikimedia, producidos entre enero de 2008 y septiembre de 2021, etiquetando manualmente aquellas cuentas *bot* activas en ese periodo; con el propósito de crear un conjunto de datos (*dataset*) de entrada para entrenar un clasificador binario, que detecte si un determinado perfil pertenece a una cuenta *bot* o no.

Tras probar diferentes modelos de clasificación usando el módulo *Scikit-learn* para *Python*, el modelo que mejor resultados obtuvo fue un clasificador de tipo *Random Forest*. Entre sus caraterísticas más relevantes destaca el empleo de una puntuación numérica calculada en base a heurísticos de este dominio de aplicación junto con valores estadísticos obtenidos de la actividad de los individuos, tales como el número de cambios o el numero de palabras y ficheros de cada cambio producido en los proyectos analizados.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

People contributing to software projects (in particular, FLOSS projects) rely on several tools to support their activity on many aspects of the project, such as source code changes, project management and coordination, software bugs or issues [3]. Data generated by such interactions can be used to extract valuable information that project managers and leaders can use to make the right decisions for the future of the project (known as data-driven decisions). Some of the most common questions while analysing an open-source project are:

- How many contributors are participating?
- How many companies contribute to the project?
- How good are these participants at handling issues?

These data are also interesting for academic purposes, as researchers and practitioners may be interested in answering a set of questions about a given project [9].

1.1 Identity problems

From a project management perspective, a person (generally with a manager role) needs to know their community or project. In order to get valuable insights, that person may ask two main questions:

- How many unique contributors do the project have?
- How many different organisations are contributing to the project?

To answer these questions, we must manage contributor identities within the project.

After spotting the usage of a plethora of different tools within FLOSS projects, it is important to explain that, for interacting with each of these tools, each project contributor must be identified in some way. This could be done by creating an account or setting up a set of credentials, usually a combination of name and email. This means each contributor will

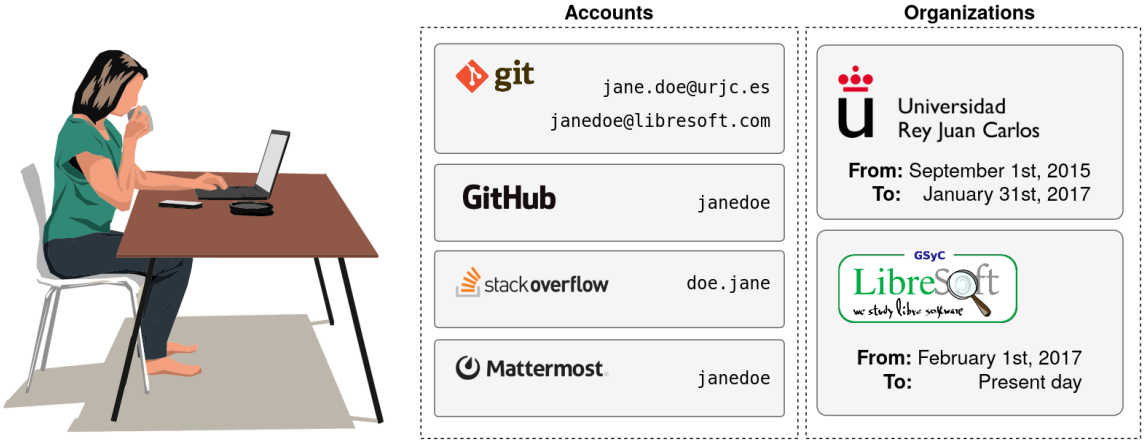

Figure 1.1: A project contributor can use many accounts across different tools and platforms, besides having a number of organizational affiliations.

end up with one or more different "accounts" or "identities" for the services the project is using.

In such a scenario, it could happen that some contributors use multiple accounts or credentials sets (from now on, we will refer to these as *identities*) for the same tool or service, for instance, to differentiate the contributions made through an organisational account from those made from a personal or academic account. We name an individual as the entity representing the many identities of a contributor, its profile, and enrollment information. This problem alone entails one of the hardest challenges: how to merge identities owned by the same individual.

This is where SortingHat, a component that is part of the GrimoireLab toolset (see 2.2.2), comes into play. This tool aims to ease the task of managing contributors' identities within a project or set of projects [14]. It will be described in detail in Section 2.2.3.

1.2 Automatic accounts: bots

It is essential to state that some interactions which occur within software development tools are not directly created by humans. Instead, they stem from an automated process set with a specific purpose and permission level to produce a specific output affecting the state of the project and its members.

This type of interaction is very common in open-source projects [5], including top-level projects and communities such as GitLab[1], Wikimedia Foundation[2] and OpenStack[3].

[1] https://gitlab.com/gitlab-org/gitlab
[2] https://wikimediafoundation.org/our-work/wikimedia-projects/
[3] https://www.openstack.org/software/

Some bots scan and re-post information, whereas others can also have a formal authority role associated with task evaluation. They can even play a management role combining evaluation and formal authority with interactive coordination, among other examples [10].

But, why is it important to identify bot accounts in open-source projects? A substantial reason is that their presence challenges any researcher or stakeholder interested in analysing the activity within a software project. Although these accounts are usually ignored in different studies, they may play an important role, as there are cases where they undertake a significant percentage of the total activity (e.g., projects where bots are responsible for accepting or rejecting 25% of all pull requests[4]) [8].

The number of bot accounts and their interactions depends on many factors, like:

- Type and purpose of the tool or service (issue management, messaging, bug tracker, etc.).
- Whether this is an option provided by default by the tool or it is an *ad-hoc* feature.
- The way these automated accounts (bots) are configured: triggered by events, periodic execution, etc.
- The amount of activity generated by humans or by other automatic accounts within the project.

SortingHat provides a way to mark unique identities as "bot" accounts by editing the identity's profile (configuring a Boolean field named `is_bot`). Currently, there is no automated way to identify which individuals from the whole data set could be marked as "bots", yet.

Up to now, this has been an entirely manual process that consumes substantial time from a person, who actively searches for suspicious identities of being bot accounts, looking at some key values such as username, email, or contribution type. This person must also double-check the original source of the data, looking for helpful extra information to verify the operator's guesses.

1.3 How this project was born

In this thesis, an approach to identify individuals from Automatic accounts (bots) is proposed, using machine learning techniques to build a classifier based on contributions produced by all identities from a given set of projects. As an additional goal, this classifier will be incorporated as a new feature in SortingHat, integrated with the original recommendation engine already implemented in this tool.

It is worth mentioning that this project was born within a strong research context. I had already started with this project when Prof. Tom Mens, Head of the Software Engineering

[4] A request for integrating changes into a repository.

Lab from the Faculty of Sciences at University of Mons (Belgium), contacted our Bitergia team to let us know about a research article regarding bot classification that they were developing at that time (September 2020). As soon I became aware of this project, I reached Prof. Tom Mens and his team to have a meeting to discuss the scope of their research and the possibility of starting a collaboration between Bitergia (the company I work for, at the time of writing this thesis) and the Software Engineering Lab from University of Mons.

From Bitergia's point of view, this was a long-desired topic to explore, as identifying bots is a crucial part of the identity management process that our company offers to customers. For the Software Engineering Lab researchers, it was helpful to promote their new tool for bot classification BoDeGHa[5] (previously BoDeGa) and their goal of having better ground-truth datasets for research purposes.

1.4 Project objectives

1.4.1 General objective

The main objective of this project is to assess if it is possible to develop an automated or semi-automated way to classify individuals from GrimoireLab's SortingHat into human users and bot accounts. This goal should be achieved through data obtained from each individual, using specific channels (data sources) to identify variables relevant to effectively undertaking this classification.

1.4.2 Specific objectives: Goals and Questions

The specific goals for this project have been defined following the "Goal, Question, Metric" approach. See Section 2.2.1 for more information.

Goal 1: Creating an automated process to discriminate between human users and bot accounts, integrated with the GrimoireLab toolset.

- **Q1.1.** How can bot accounts be separated from human users?
- **Q1.2.** Is the profile information from a given individual enough to classify it as human or bot?
- **Q1.3.** Are there differences between activity generated by humans and bots?
- **Q1.4.** How can this classifier be integrated into GrimoireLab's toolchain?

Goal 2: Finding which channels and footprints can be used to classify a user as human or bot.

- **Q2.1.** Are there any particular channels and footprints, as a combination of interactions, which can be used to classify a user as a human or bot?

[5] https://github.com/mehdigolzadeh/BoDeGHa

- **Q2.2.** Can the message content (commit messages, issue texts, etc.) be used to validate this classification?
  - **Q2.2.1.** Does a richer syntax give any hint about the nature of a user?
  - **Q2.2.2.** Can the entropy of a comment give a hint about the nature of a user?
- **Q2.3.** Do activity details (such as working hours or frequency of contributions) help with this classification?

Goal 3: Obtaining a curated dataset from real open-source communities with real examples of bot accounts.

- **Q3.1.** Which open-source communities should be analysed?
- **Q3.2.** Which data sources are we taking into account?
  - **Q3.2.1.** Which data should we consider form these sources?

1.5 Time planning

Considering natural time, I spent, roughly, 1 year and 7 months working mostly during weekends, as I conciliated it with my full-time job. Whilst the main conversations for starting this project began in September 2020, the first stage started on March 2021. The time I spent during the first stages of the project were quite uneven, but from April 2021 I was able to keep a more regular pace until its completion on January 2023. This is the estimation of when each task was carried out and how much time was spent on it:

- First design of the tool, data obtention and curation: March 2021-September 2021.
- Designing the tool, additional work on data curation: September 2021-October 2021.
- Building the input dataset and first experiments: October 2021-January 2022.
- Second round of experiments: April 2022-July 2022.
- Third round of experiments and writing the thesis: July 2022-January 2023.

1.6 Structure of the thesis

This thesis is outlined as follows:

- In this Chapter 1, "Introduction", the general context and motivation is described for the problem we aim to solve. Also, the objectives for the project were already detailed in subsection 1.4, "Project objectives".
- Next, (Chapter 2), "State of the Art", provides information about previous research work on this field and also a brief explanation of the technologies that were used during the process.

- The design process and architecture of the tool are detailed in Chapter 3, "Design and Implementation", including a breakdown of its components and a detailed analysis of the dataset obtained for the purpose of this project.
- In Chapter 4, "Experiments and validation", the classifiers performance and results are examined through different experiments, including the description of technical challenges encountered and how they were addressed.
- Wrapping up, Chapter 5, "Conclusions" evaluates whether the set objectives were met, and includes a discussion on the limitations of the tool, lessons learned and future work.
- Furthermore, the Appendix A, "Definitions", provides additional explanations for several key terms.

There is a web site dedicated to this final project[6], including this thesis. Complementary content, along with the code for this tool are available in a dedicated GitHub repository[7].

[6] https://mafesan.github.io/Memoria-TFM
[7] https://github.com/mafesan/2021-tfm-code

Chapter 2

State of the Art

As mentioned in the Introduction chapter, this project was born within a strong research context. When I contacted Prof. Tom Mens and his team about this Master's thesis, we first exchanged ideas regarding the scope of the project and how both initiatives could complement one another. Then, they shared with me the scientific paper they were developing, aimed at detecting bots in issues and PR comments from GitHub.

A discussion followed about which research lines could be addressed for this Master's thesis. The Software Engineering Lab from University of Mons did not implement the tool, nor the underlying classifier, to detect bots based on Git commit comments or any other Git-related information. As it would be relatively easy to extend their tool to also consider Git commit comments, it was likely the classifier features that they used to distinguish bots from humans in GitHub issue comments, and PR comments do not work that well on Git comments. Studying, testing, and extending this behaviour was one of the main ideas they proposed to me for this Master's thesis. Likewise, this study could be extended to look at other systems and data sources. Last, but not least, it was interesting for them to learn how this classification was going to be integrated with identity merging (mainly talking about GrimoireLab's SortingHat component).

After reviewing the paper from the Software Engineering Lab at University of Mons, by Mehdi Golzadeh et al., I discovered that this text pointed to other interesting articles about the same topic, which are relevant to this Master's thesis.

2.1 Research

In the following subsections, I summarise the two most relevant research articles on which this project is supported, and the technologies used for this project.

- The first one is the article by Researcher Mehdi Golzadeh, Prof. Tom Mens et al.: **"A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments"** [8], on detecting bots in issue and PR comments from OSS projects.

- The second one by Dey, B. Vacilescu et al.: **"Detecting and characterising bots that commit code"** [4], on detecting bots contributing code in OSS[1] projects.

2.1.1 A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments

The main goal of this paper is to propose an automated classification model to detect bots through comments submitted in GitHub issues and pull requests.

This article is divided into three large sections: First, they elaborate a ground-truth dataset of pull requests and issue comments from 5K GitHub accounts, from which 527 were identified as bots. Then, they propose a classification model that relies on comment-related features to classify accounts as either bot or human. Eventually, they propose an open-source tool based on the classification model to allow GitHub contributors to detect which accounts in their repositories correspond to bots.

Creating the ground-truth dataset

As the objective of this study is to focus on software development repositories, they need a way to identify which repositories from GitHub were created for such purpose. Hence, they relied on *libraries.io*, a monitoring service indexing information for several million packages distributed through several package registries, such as PyPI, npm, etc.

Their initial dump contained more than 3.3 million GitHub repositories, from which they randomly selected around 136K of them. From each of these repositories, they extracted the last 100 comments of the last 100 issues and pull requests during 4 days, in February 2020, using GitHub's API. They obtained over 10M comments from more than 837K contributors, and from more than 3.5M issues and pull requests.

As the initial dataset was too big, they applied some constraints. First, they excluded users who made less than 10 comments. This threshold comes from a previous study. Then, they extracted a subset of 5K commenters, selected both randomly and manually, adding 438 commenters who had been identified as bots in previous studies or contained a specific substring in their GitHub account name, such as "bot", "ci", "cla", "auto", "logic", "code", "io" and "assist".

Then, for the labelling process, they developed a web application where each commenter was presented to at least two of the four authors of the paper. Comments belonging to a certain user were displayed in batches of 20 comments (with the option of showing more, if needed). Then, the rater could select whether the commenter was a bot or a human being. All cases that were agreed upon were included in the ground-truth dataset.

Creating the classification model

These were the selected features to create the classification model:

[1] Open-Source Software.

1. Text distance between comments.

    - The main hypothesis is that bot commenters post more repetitive comments than humans do. This is why the metrics considered are text distance metrics, that are commonly used in natural language processing (NLP): the Jaccard and Levenshtein distances. The Jaccard distance A.2 aims to quantify the similarity of two texts based on their content, while the Levenshtein distance A.3 intends to capture the structural difference by counting single-character edits.

    - After a tokenization process, for each commenter the mean of the Jaccard and Levenshtein distances between all pairs of comments are computed. Results show that humans got higher median values for both distances than bots. Nonetheless, there is overlapping between the values from both classes, indicating that these mean distances are not enough to properly distinguish between the two classes.

    - Finally, a combination of both Jaccard and Levenshtein distances is used A.4.

2. Repetitive comment patterns.

    - Observations suggest that bots tend to have sets of similar comments, while most comments from humans are unique, except some of them that seem to follow a pattern (mostly, short answers such as "Thank you!", "`+1`" or "*LGTM*"[2].

    - To capture the comment patterns, they select *DBSCAN*, a density-based clustering algorithm. To capture both structural and content distance between comments, a combination of both Levenshtein and Jaccard distances is computed. For each commenter, *DBSCAN* is applied to its set of comments.

    - When the number of comment patterns (clusters) and the number of comments considered per commenter is represented, there is a clearer separation between humans and bots. The number of comment patterns for bots remained stable and low, regardless of the number of comments.

3. Inequality between comments in patterns

    - The inequality in the number of comments in each pattern is used as an additional feature to distinguish between bots and humans by using the *Gini* coefficient (a value of 0 expresses perfect equality, while a value of 1 expresses maximum inequality among values).

    - Humans show a lower inequality than bots with respect to the spread of comments within patterns, confirming that humans tend to have a lower inequality than bots, a consequence of many of their patterns containing a single comment.

4. Number of comments and empty comments.

[2]LGTM: Shorthand for "Looks good to me". "`+1`", as a common way to express agreement with something proposed in a previous comment or description.

- This feature makes it easier to distinguish between commenters having a similar number of patterns (the ones having more comments per pattern, will more likely be a bot).

- Regarding the number of empty comments, although the GitHub interface does not allow empty comments in a discussion, it does not prevent comments composed of whitespace characters. Data shows that these empty comments are mostly created by human commenters.

For selecting the classifier, they rely on a standard grid-search *10-fold* cross-validation process to compare five families of classifiers (random forest, k-nearest neighbours, decision trees, logistic regression, and support vector machines) over the training set (60% of the ground-truth dataset) using *scikit-learn* 2.2.4. In addition, the class "weight" parameter was set to address the class imbalance problem for each supported classifier.

The 10 subsets were created using a stratified shuffle split, to preserve the same proportion of bots and humans as in the complete training set.

The selected classifier is the "Random forest": using the *Gini* split criterion, they get 10 estimators (trees) and a maximum depth of 10 for these trees. Results are available in Table 2.1.

	Classified as bot	**Classified as human**	**P**	**R**	F_1
Bot	TP: 192	FN: 19	0.94	0.91	0.92
Human	FP: 13	TN: 1,776	0.99	0.99	0.99
Weighted avg			0.98	0.98	0.98

Table 2.1: Evaluation of the classification model using the test set.

BoDeGHa: an open-source tool to detect bots in GitHub repositories

The tool accepts as inputs the name of a GitHub repository and a GitHub API key. The output is computed in three steps:

1. Download all comments from that repository through GitHub's GraphQL API, which is transformed into a list of commenters and their corresponding comments.

2. Compute the features for the classification model: number of comments, empty comments, comment pattern, and inequality between the number of comments within patterns.

3. Apply the pre-trained model and outputs the prediction made by the model.

Conclusions

From the 15 bots classified as humans, most cases correspond to bots that use, convert, copy or translate text that humans initially produced. When looking at the 51 humans classified as bots, most have unfilled issue templates, use repetitive comments such as "Thank you" or "LGTM", or post empty comments. About 85% of the misclassified humans and about 75% of misclassified bots are initially difficult to classify by at least one of the raters, as "I don't know", "difficult", or "very difficult".

They also find several examples of commenters whose behaviour and comments correspond to those of both humans and bots, that is, mixed commenters using their GitHub accounts belonging to humans allowing automatic tools to make use of the account for specific tasks. These cases represent the 1.5% (78 commenters out of 5,082), and they exclude them from the ground-truth dataset, as they can not decide whether these commenters should be classified as bots or as humans. The mixed commenters are exposed to test how the model behaved with these cases, resulting in 29 being classified as bots (37.2%) and 49 as humans (62.8%).

Although other articles, such as the one from Dey et al. (explained in the following subsection), proposed approaches for identifying bot accounts based on their commit messages or their author information, such as checking the presence of the string "bot" in the account name or the committer name, it lead to numerous both false positives and false negatives.

2.1.2 Detecting and characterising bots that commit code

The main goal of this article is to find an automated way of identifying bots (and their contributions) that commit code in open-source projects and characterise them according to their activity.

To do so, they propose a systematic approach named **BIMAN** (*Bot Identification by commit Message, commit Association and author Name*) to detect bots considering different aspects of the commits made by an author:

1. Commit Message: Identify if commit messages are being generated from templates.
2. Commit Association: Predict if an author is a bot using a random forest model, using features related to the information from the commits as predictors.
3. Author Name: Match the author's name and email to common bot patterns.

This method is applied to the *World of Code* dataset [11], obtaining a subset of the data, gathering information about 461 bots detected by this approach and manually verifying as bots, each with more than 1,000 commits.

Their method to extract information about the authors consisted of these steps: First, obtaining a list of all authors from the *World of Code* dataset[3]. Second, identifying all commits

[3]The author's IDs were represented by a combination of name and email address.

from the authors. And third, extracting the list of files modified by a commit, the list of projects the commit is associated with, and the commit content for each commit for every author.

That being said, **BIMAN** (the proposed technique for detecting bots) comprises three methods, which are detailed in the following subsections. This dataset is also used to characterise the bots based on their patterns, such as the type of files modified and time distribution, to analyse their work and the programming languages they use.

Identifying bots by name (BIN)

After inspecting the dataset, regular expressions are used to identify if an author is a bot by checking if the author's name or the email contained the substring `bot`. These expressions have restrictions, like searching for the string preceded and followed by non-alpha characters (to avoid false positives, such as names like "Abbot") and excluding from this search the email domain.

The initial assumption is that bots are very active and produced a significantly greater number of commits than humans. However, the observations show that the number of commits between humans and bots is not quite different. Among the reasons behind this statement, one is that given an author ID consisting of a name-email combination, slight variations in this combination lead to consider some cases as different authors when they are not. Besides, bots might have been implemented as an experiment or as part of a course and never used afterwards. Another reason could be that some bots were designed for a project, but in the end, they were never fully adopted.

Detecting bots by commit messages (BIM)

The primary assumption is considering that bots use template messages as the starting point for the commit message. Thus, the goal is detecting if the commit message came from a template. Although humans can also generate commit messages with similar patterns, the hypothesis is that the variability of content within messages generated by bots is lower than the messages coming from humans.

The **BIM** approach uses a document template score algorithm, which compares document pairs and uses a similarity measure to group documents. A group represents documents suspicious of conforming to a similar base document. Each group has a single template document assigned to it, and this document is used for comparisons. A new group is created when the similarity of a document does not reach the threshold with any other template document for that group. After this, documents are compared, and a score is calculated based on the ratio of the number of template documents and the number of documents.

Detecting bots by files changed and projects associated with commits (BICA)

Twenty metrics are used as a starting point, using the files changed by each commit, the projects that the commit is associated with, and the timestamp and timezone of the commits.

The random forest model performs better than other approaches for predicting whether an author was a bot using the numerical features. Out of the 20 variables, only six features are retained as predictors (see Table 2.2).

Variable name	Variable description
Tot.FilesChanged	Number of files changed by author across commits
Uniq.File.Exten	Num. of unique file extensions in all the author's commits
Std.File.pCommit	Std. dev. of number of files per commit
Avg.File.pCommit	Mean number of files per commit
Tot.uniq.Projects	Num. of unique projects associated with commits
Median.Project.pCommit	Median num. of projects associated with commits

Table 2.2: Predictors used in the random forest model for BICA.

Ensemble model

The ensemble model is implemented as another random forest model and combine the outputs of the three methods explained so far (**BIN**, **BIM** and **BICA**) as predictors to make a final decision on whether an author is a bot or not.

Sinde the golden dataset is generated using the BIN method, the authors do not use it for training the ensemble model. Instead, they create a new training dataset partly consisting of 67 bots from which 57 author IDs are associated with eight bots and ten author IDs are linked to three other known bots that are not in the golden dataset. Furthermore, 67 human authors are included via random selection and manual validation.

The output from **BIN** is a binary value stating if the author ID matched the regular expressions detailed before; the output from **BIM** is a score, with higher values corresponding to a higher probability of the author being a bot; and the output from **BICA** is the probability for an author of being a bot.

BIMAN results

BIMAN identifies 58 (87%) out of 67 author IDs as bots, and 6 out of 9 other IDs could be identified as not actually being a bot via manual investigation, they were either spoofing the name or simply using the same name.

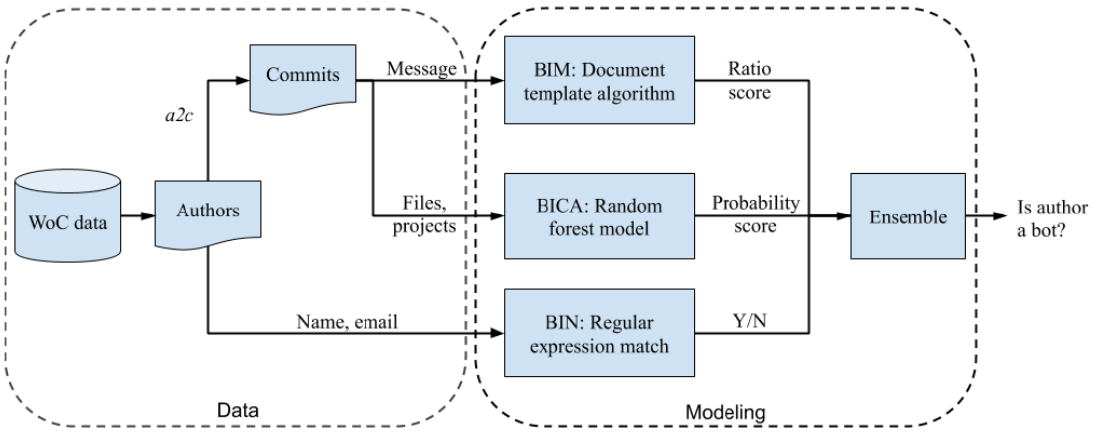

Figure 2.1: BIMAN workflow: Scores from each method are used by an ensemble model that classifies the given author as a bot or not a bot (taken from the original paper).

BIN peformance: during creation of the golden dataset, BIN obtained a precision close to 99%, which indicates that any author considered to be a bot using this method has a very high probability of being a bot. In general, humans do not try to disguise themselves as bots. The recall measure is not high, because BIN misses many cases where the bots do not explicitly have the substring "`bot`" in their name.

BIM performance: the document template score algorithm solely relies on the commit messages. The AUC-ROC value using the ratio values as predicted probabilities is 0.7. Some details about the classification output:

- **True Positive**: The cases where this model can correctly identify bots were cases where the bots actually used templates or repeated the same commit message.
- **False Negative**: The cases where this model cannot correctly identify bots were mostly cases where the bots reviewed code added by humans and created a commit message that added a few words with the commit message written by a human.
- **True Negative**: The human authors correctly identified have some variation in the text, with the usual descriptions of change.
- **False Positive**: Humans who are misclassified as bots usually had short commit messages that were not descriptive, and they reused the same commit message multiple times.

BICA performance: The golden dataset generated using the **BIN** method is used for training the model and testing its performance. 70% of the data, randomly selected, is used for training the model and the rest 30% is used for testing. This procedure is repeated 100 times with different random seeds. The model shows good performance, with an AUC-ROC value of 0.89.

Ensemble model performance: The dataset used for training and testing the performance

of this model has only 134 observations, because of reasons described in Section 2.1.2. 80% of the data are used for training, and 20% for testing. The process is repeated 100 times with different random seeds. The value of the AUC-ROC measure varies between 0.89 and 0.95, with a median of 0.90.

Conclusions

After studying the results, the authors conclude that a significant portion of authors can be identified as bots using the proposed method.

Among the limitations for this approach, they mention the lack of a golden dataset and the lack of a ground truth to validate this dataset against. Like in the previous article, another threat is that a number of developers use automated scripts to handle some of their works, which uses their Git credentials while making commits.

Moreover, they mention that **BIM**'s performance varies according to the language of the commit messages (e.g., Spanish and Chinese), and it does not support multilingual sets of commit messages.

They do not address the problem of multiple IDs belonging to the same author, so this was planned as future work to extend the **BIMAN** method.

2.2 Technologies

2.2.1 GQM approach

The "Goal Question Metric" (GQM) approach [1] is based upon the assumption that for measuring purposefully, first the goals must be specified for the project, then those goals must be traced to the data that are intended to define these goals operationally, and finally to provide a framework for interpreting the data with respect to the stated goals.

This approach helps to define the metrics that matter in each case, avoiding frequent bad practices. For example, people tend to use or define a set of metrics without having a clear idea about the specific goals they pursue. This usually leads to a "bottom-up" approach: besides having metrics misaligned with the project or business goals, the set of metrics may also be biased by the current technology applied to obtain these metrics.

The lack of a well-defined strategy also hinders practitioners from understanding which metrics are important and why. By using a "top-down" approach (first goals, then metrics), it becomes easier to materialise a targeted set of questions for the current situation and then check which metrics could be useful in the future or not, or how these metrics can help to reach the different goals by answering the questions that were raised.

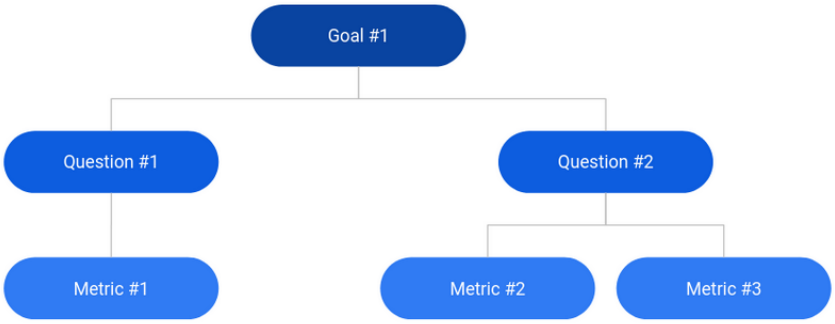

Figure 2.2: Example: Goal, Question, Metric approach hierarchy.

2.2.2 GrimoireLab

GrimoireLab [4] is a free, open-source toolset for software development analytics.

This toolset provides a whole platform that supports automatic and incremental data gathering from many tools (data sources or *backends*) related to open-source development (source code management, issue tracking systems, messaging tools, mailing lists, etc.).

Data obtained from these tools is stored in JSON documents following a uniform format, no matter the source. These JSON documents are stored in Elasticsearch and, then, undergo a data enrichment process which adds additional information such time calculations (delays, duration), contributors' affiliation, and more. Once the data have been augmented, they can be consumed by visualisation tools and also directly using the Elasticsearch API. GrimoireLab toolset comes with a tool named "Kibiter" which is a soft *fork* of Elastic's Kibana. A set of predefined dashboards and visualisations are included for each data source.

GrimoireLab is part of CHAOSS [5], a project sponsored by The Linux Foundation. It is mainly developed by the Spanish company Bitergia, and represents an evolution of the work done over more than 10 years in Bitergia and the LibreSoft research group at Rey Juan Carlos University.

2.2.3 SortingHat

SortingHat is the GrimoireLab component for identity management. It provides more than 20 commands to manipulate identities, including support for:

i) identity merging based on email addresses, usernames, and full names found on many tools used in software development;

ii) enrolling members to organisations for a given time span, marking identities as automatic accounts (bots);

[4] https://chaoss.github.io/grimoirelab/
[5] https://chaoss.community/about-chaoss/

iii) gender assessment, among other features [14];

This tool maintains a relational database with identities and related information extracted from different tools used in software development. An identity is a tuple composed of a name, email, username, and the source's name from where it was extracted. Tuples are converted to unique identifiers (i.e., uuid), which provide a quick mean to compare identities among each other. By default, SortingHat considers all identities as unique ones. Heuristics take care to automatically merge identities based on perfect matches on (i) uuids, (ii) name, (iii) email, or (iv) username.

In case of a positive match, an identity is randomly selected as the unique one, and the other identities are linked to it.

Currently, SortingHat is evolving into a service-based application implementing a *GraphQL API* in *Python*.

2.2.4 Python

Python [6] is an interpreted, object-oriented, high-level, open-source programming language for general-purpose programming created by Guido van Rossum in 1991 [17]. Nowadays, the most recent version is 3.10.0, from October 2021. Its design is focused on code readability and clear syntax, making it possible to program using fewer lines of code compared to other programming languages such as *C++* or *Ada*.

Python features a large standard library, which includes many tasks from text pattern matching to network scripting, in addition to a vast collection of third-party application libraries. Other remarkable features are portability, as *Python* interpreters are available for many operating systems; and the component integration, as *Python* scripts can easily communicate with other parts of an application or code, like *C++* libraries, *MySQL* databases, etc.

In this case, we focus on libraries oriented towards data science. Next, we describe several important libraries utilised in this project.

NumPy and Pandas

NumPy [7] is an open-source project aiming to enable numerical computing with Python. It was created in 2005, building on the early work of the Numeric and Numarray libraries.

NumPy has become the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

[6] https://www.python.org/
[7] https://numpy.org/

Pandas [8] is an open-source data analysis and manipulation tool, built on top of the Python programming language. This library was developed as an extension of NumPy, and it offers efficient data structures (such as *Dataframes*) and operations to handle with numeric tables and time series.

Matplotlib

Matplotlib [9] is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits.

Jupyter Notebooks

Jupyter Notebook[10] (formerly IPython Notebooks) is a web-based interactive computational environment for creating notebook documents.

A Jupyter Notebook document is a browser-based interactive, simple programming environment containing an ordered list of input/output cells which can contain code, text in *Markdown* format, mathematics, plots and rich media. Underneath the interface, a notebook is a JSON document, following a versioned schema, usually ending with the `.ipynb` extension.

Jupyter notebooks are built upon a number of popular open-source libraries, such as IPython [11].

Imbalanced-learn

Imbalanced-learn[12] (imported as Python module as `imblearn`) is an open-source, MIT-licensed library relying on scikit-learn (imported as sklearn) and provides tools when dealing with classification with imbalanced classes.

Scikit-learn

Scikit-learn[13] is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in *Cython* (a superset of the Python language that additionally supports calling *C* functions) to improve performance.

The classification models proposed and tested for this project are based on this library.

[8] https://pandas.pydata.org/
[9] https://matplotlib.org/
[10] https://jupyter-notebook.readthedocs.io/en/stable/
[11] https://ipython.readthedocs.io/en/stable/
[12] https://imbalanced-learn.org/stable/index.html
[13] https://scikit-learn.org

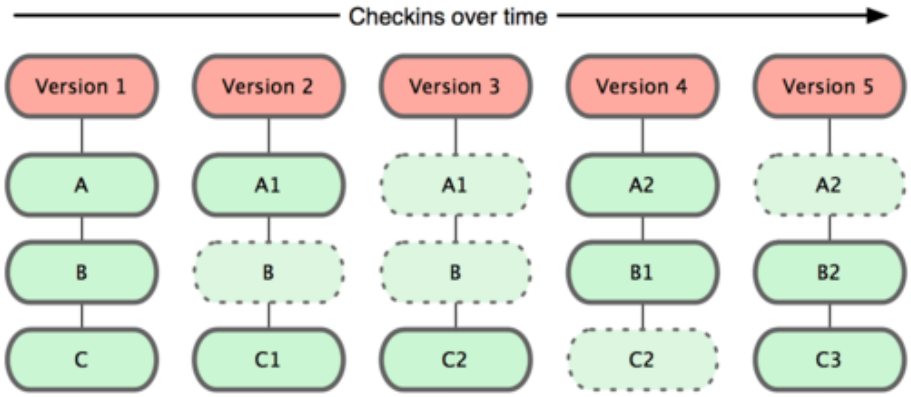

Figure 2.3: How *git* structures its information internally [16].

XGBoost

XGBoost[14] (from *eXtreme Gradient Boosting*) is an open-source, optimised distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

2.2.5 Git

Git is an open-source Version Control System (*VCS*), originally developed in 2005 by Linus Torvalds [15]. Like any other *VCS*, *git* is a system that records changes to a file or set of files over time so that you can recall specific versions later. According to the last surveys, it is by far the most used *VCS* in the world[15].

Git thinks of its data more like a series of snapshots of a miniature file system (See Figure 2.3). Every time you commit or save the state of your project, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, *git* does not store the file again, just a link to the previous identical file it has already stored. All this information is stored in a key-value system as *git* objects, with a unique identity for each of them.

[14] https://github.com/dmlc/xgboost
[15] https://insights.stackoverflow.com/survey/2021

Chapter 3

Design and implementation

The proposed tool for this project, *Revelio*, aims to provide a report with the results of an automatic classification to discriminate bot accounts from the rest, over the available information from individuals in a given GrimoireLab instance.

3.1 General architecture

Revelio is a tool which has been designed to be composed by different modules, to ease its adaptability to future updates and extensions.

The tool is composed of different modules (see Figure 3.1), and each module consists on one or more scripts.

As a general description, the tool requires a running GrimoireLab instance to execute. This GrimoireLab instance would contain data from many endpoints stored in an ElasticSearch instance, together with a relational database containing identity information.

With the GrimoireLab instance in place, *Revelio* accepts three main input parameters:

- The URL or the IP address of the ElasticSearch instance.
- The credentials to access ElasticSearch and SortingHat.
- The index name from ElasticSeach, containing the GrimoreLab-formatted data.

With these input parameters, the tool executes the following steps:

1. **Data extraction**: *Revelio* extracts the data per individual from the selected index querying the ElasticSearch instance.
2. **Data processing**: The extracted data is analysed and processed, creating the datasets for the classification phase.
3. **Classification**: In this phase, the classification models are defined and adjusted. The output of this chain is a report containing the results of the classification: An attribute

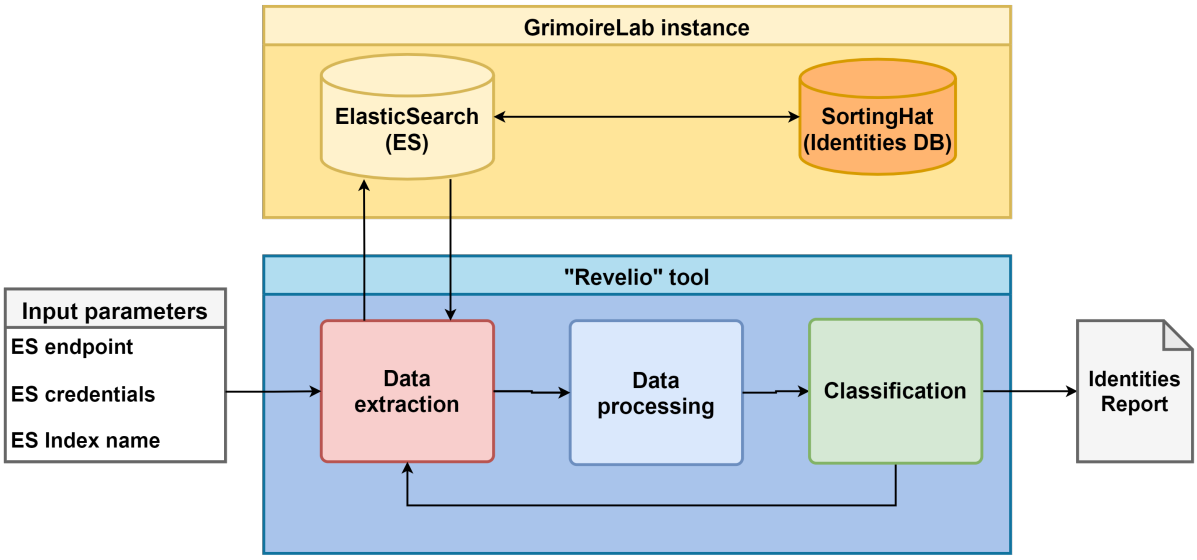

Figure 3.1: General architechture of the *Revelio* tool.

`is_detected_bot` for each individual, and another attribute for the accuracy of the result, `bot_acc`.

In the following sections, the structure and the implementation of the tool are explained. Each main phase from the general structure is more detailed in the diagram from Figure 3.2: keeping the colour code from Figure 3.1, this diagram shows the sub-modules from each main part of the tool and the execution flow detailed in the sections and subsections below.

3.2 Creating the initial dataset in GrimoireLab

3.2.1 Selecting the community to analyse

In order to create and test the classification models for this project, an initial dataset was needed. After some research, I chose the Wikimedia Foundation community as the target to analyse, as it is a well-known community with a lot of active projects, which many of them use automation tools.

Wikimedia Foundation has the "Wikimedia Tech community metrics dashboard"[1], which is a running GrimoireLab instance maintained by Bitergia dedicated to get activity metrics from the Wikimedia Tech community. In their public documentation[2] (see Figure 3.3) it is linked to the list of repositories they are tracking for their community metrics dashboard.

[1] https://wikimedia.biterg.io/
[2] https://www.mediawiki.org/wiki/Community_metrics

Figure 3.2: Process architecture.

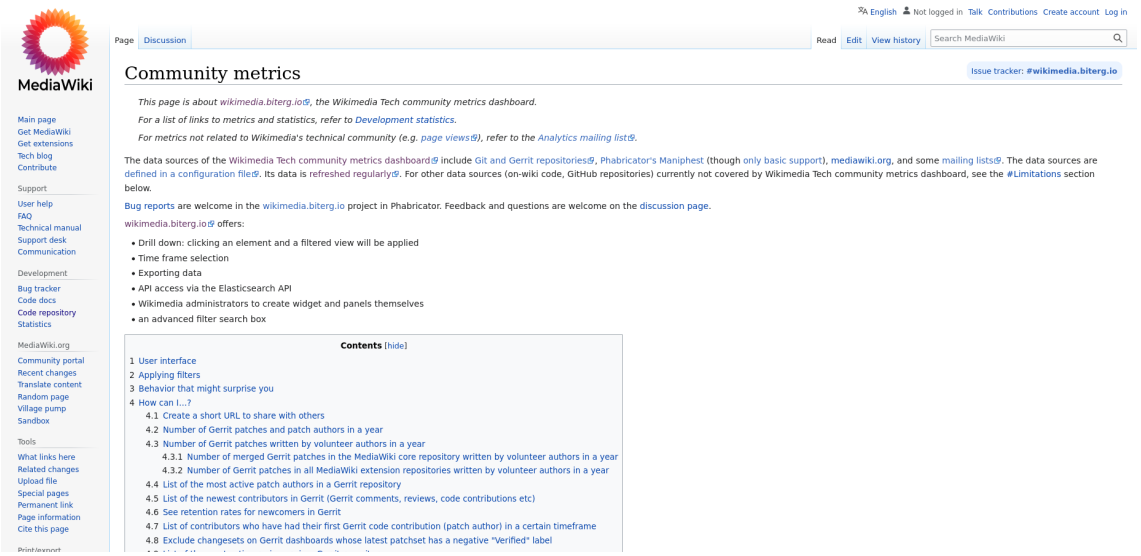

Figure 3.3: Community Metrics wiki page from Wikimedia Foundation.

I decided to set up a local GrimoireLab instance with a subset of the projects. As the main goal of this project is to detect automated accounts out of human accounts, I decided to apply a first filter to exclude most of Wikimedia's research projects. Research projects usually don't have the desired scale nor the level of activity required for the classification stage. Also, the results could not be as much generalised to other projects and communities.

As for the selected data sources to analyse, this project is focused on the data from Git. Some of the selected repositories were stored on GitHub, others were stored in a Wikimedia-managed Gerrit instance.

3.2.2 Setting-up the GrimoireLab instance

With this information, I deployed a local instance of GrimoireLab (Figure 3.4) by using a Docker-compose file available on GrimoireLab repository, configuring both the sources file, with the selected subset of repositories to analyse, and the setup configuration file, including:

- The name of the instance.
- The set-up parameters for the identities' management tool, *SortingHat*.
- The name of the ElasticSearch indexes and additional pre-computed studies over the data provided by GrimoireLab.

Once the platform is running, GrimoireLab instance performs many operations:

1. First, it downloads the raw data from the selected data sources. For this case, to get the data from Git, the repositories from the sources file are downloaded and the Git log file is parsed. This information is stored in JSON documents, one document per "Commit".

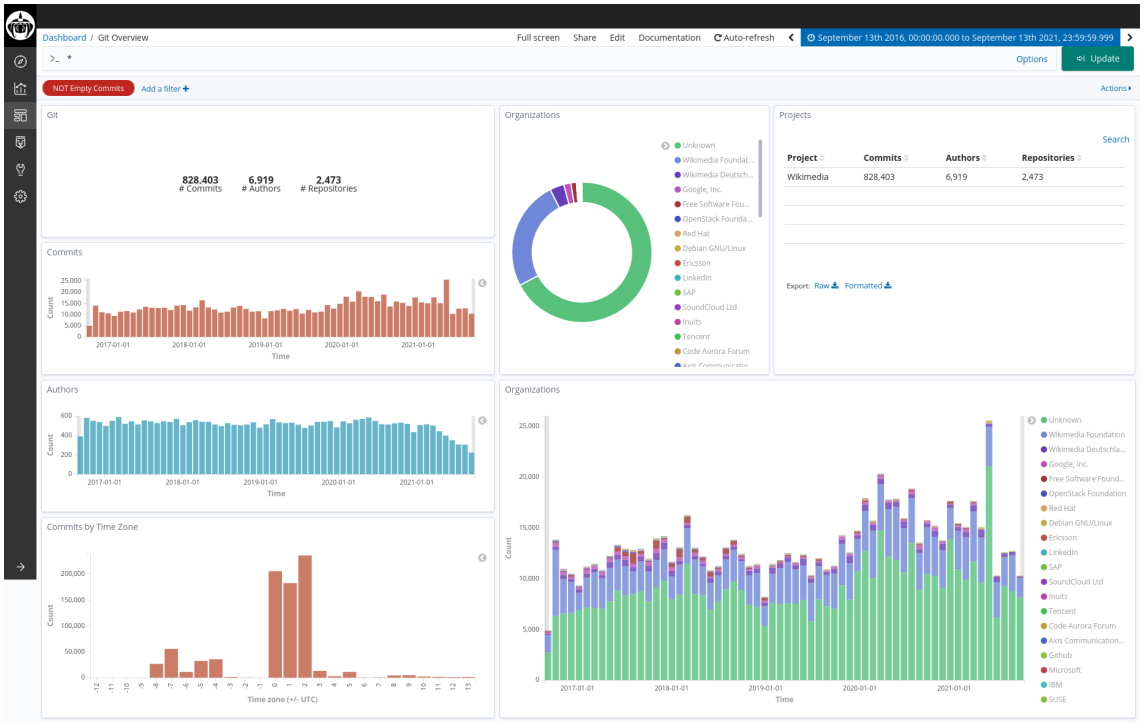

Figure 3.4: Git Overview dashboard from the local GrimoireLab instance.

2. Then, these documents are processed by another tool that adds additional information such as the number of files modified or the number of total lines added and removed on each commit, or pre-computed fields like time differences. This process is called *data enrichment*.

3. Besides, identities-related information is aggregated to this extended set of the data (enriched data). That is, each commit is authored by an individual, represented in the identities' database by a unique identifier, an associated profile, and a list of identities belonging to this individual (different email addresses, different GitHub accounts, etc.).

3.2.3 Curating identities information

The next step is to curate the identities' information. The GrimoireLab tool managing this process, SortingHat, offers mechanisms to automatically improve some key data.

Regarding account-merging, SortingHat provides many possibilities to merge profiles based on different fields from individuals (email, name and username) both individually or jointly. As we are not assuming any information, the safest approach was to merge profiles by their email. That is, if there are two or more individuals using the same email account, they are merged into the same profile.

Then, it comes the affiliation information. SortingHat features a way to automatically enrol individuals in an organisation based on the email domain from the profile. This is done by

using an organisation-domain map (e.g.: Domain `wikimedia.org` is linked to the organisation "`Wikimedia Foundation`").

Having said this, SortingHat does not have any automated way to detect which individuals are automatics accounts (*bots*), and this is where *Revelio* tool comes to play.

Two approaches are followed to identify which individuals were bots. The first step consisted of taking all the accounts which were already identified as "bots" by the Wikimedia tech community itself in a dedicated Affiliations dashboard[3], filtering in bot individuals (`author_bot:true`) from the `git` index.

Second, the rest of the individuals are manually reviewed to identify potential bots and then confirm they were looking at their activity. A total of 41 bot individuals were identified, out of 16, 284. With this information, a SortingHat-supported file is composed with information about each bot individual, for replication purposes.

3.3 Data extraction

Before data are extracted from ElasticSearch, it is important to understand how GrimoireLab platform is modelling such data. In this case, we are focusing on Git data.

GrimoireLab downloads each Git repository and parses the Git log (see Figure 3.5), storing the entire commits history. From this record per repository, GrimoireLab composes a JSON document per commit, with a set of fields reflecting the information associated with it, such as the unique identifier (*hash*), the number of modified lines, the number of modified files, the commit message, and more (see Listing 3.1). Later, there is another internal process where these data are augmented with extra fields, such as the identity information matched with data from the identity management system, together with some pre-processed fields that allow aggregating them in a simpler way in Kibana, the visualisation layer from the Elastic stack.

In summary, the `git` index stores one document per commit, with a set of fields whose meaning is described in the corresponding data schema[4].

3.3.1 Querying the data from ElasticSearch

A set of metrics were defined following the GQM approach [5] [1]. The next step is selecting the proper fields from the ElasticSearch index that will allow obtaining such metrics, listed in Table 3.1.

ElasticSearch is a search engine that behaves similarly to a NoSQL database. As the stored entities are single documents, the data need to be aggregated in some way. ElasticSearch

[3] https://wikimedia.biterg.io/app/kibana#/dashboard/Affiliations

[4] https://github.com/chaoss/grimoirelab-elk/blob/master/schema/git.csv

[5] These metrics can be found as additional content in the Jupyter Notebook dedicated to the Exploratory Data Analysis: https://mafesan.github.io/Memoria-TFM/exploratory-data-analysis.html.

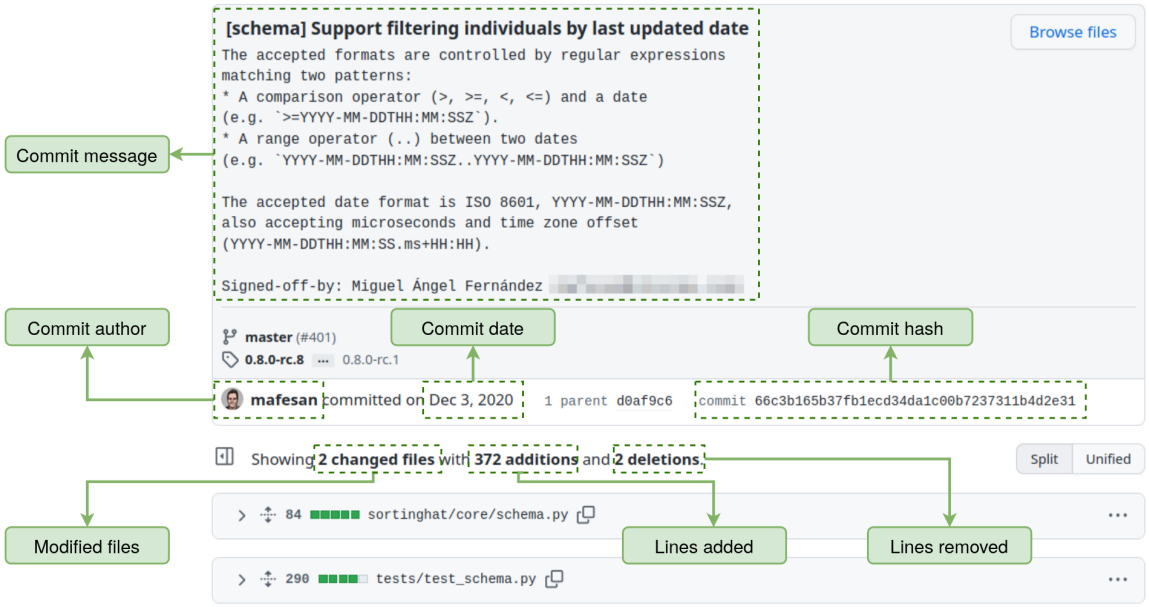

Figure 3.5: Graphic example of a Git log entry and some of the information we can extract from it.

has two basic types of aggregations: **metrics** and **buckets**. As their documentation explains:

- The **metrics** aggregation[6] computes metrics (such as the average, median, unique counts, etc.) based on values extracted in one way or another from aggregated documents. These values are typically extracted from documents fields.
- The **buckets** aggregation[7] consists on creating sets of documents (called buckets) taking a given criterion (depending on the aggregation type) which determines whether a document in the current context "falls" into it.

The piece of code in charge of retrieving these data is the script `ES-extract-datasets.py`.

A first process executes a *bucket* aggregation using the unique identifier for the contributor identities, `author_uuid`. This first query produces a list of contributors sending commits during a given period of time. Then, a second process executes a query for each author to retrieve the history of commits, asking for the fields that were defined in Table 3.1. Although results are paginated, this process is split by monthly date ranges from the main time period considered for the study, in this case from January 1st, 2008 to September 15th, 2021.

[6] https://www.elastic.co/guide/en/elasticsearch/reference/6.8/search-aggregations-metrics.html

[7] https://www.elastic.co/guide/en/elasticsearch/reference/6.8/search-aggregations-bucket.html

```
{
    "author_bot": false,
    "author_date": "2020-12-02T13:16:53",
    "author_date_hour": 13,
    "author_date_weekday": 3,
    "author_name": "Miguel Ángel Fernández",
    "author_org_name": "Bitergia",
    "author_user_name": "mafesan",
    "author_uuid": "226b402c4ab1a8a114ff9bd804f4e250c0aa05db",
    "commit_date": "2020-12-03T11:38:49",
    "commit_date_hour": 11,
    "commit_date_weekday": 4,
    "files": 2,
    "github_repo": "chaoss/grimoirelab-sortinghat",
    "grimoire_creation_date": "2020-12-02T13:16:53+01:00",
    "hash": "66c3b165b37fb1ecd34da1c00b7237311b4d2e31",
    "hash_short": "66c3b1",
    "lines_added": 372,
    "lines_changed": 374,
    "lines_removed": 2,
    "message": "[schema] Support filtering individuals by last updated date\n\nThe
↪  accepted formats are controlled by regular expressions\nmatching two patterns:\n* A
↪  comparison operator (>, >=, <, <=) and a date\n(e.g. `>=YYYY-MM-DDTHH:MM:SSZ`).\n* A
↪  range operator (..) between two dates\n(e.g.
↪  `YYYY-MM-DDTHH:MM:SSZ..YYYY-MM-DDTHH:MM:SSZ`)\n\nThe accepted date format is ISO
↪  8601, YYYY-MM-DDTHH:MM:SSZ,\nalso accepting microseconds and time zone
↪  offset\n(YYYY-MM-DDTHH:MM:SS.ms+HH:HH).\n\nSigned-off-by: Miguel Ángel Fernández
↪  <*****@*****>",
    "origin": "https://github.com/chaoss/grimoirelab-sortinghat",
    "project": "GrimoireLab",
    "repo_name": "https://github.com/chaoss/grimoirelab-sortinghat",
    "time_to_commit_hours": 1.63,
    "title": "[schema] Support filtering individuals by last updated date",
    "tz": 1,
    "url_id":
↪  "chaoss/grimoirelab-sortinghat/commit/66c3b165b37fb1ecd34da1c00b7237311b4d2e31",
    "utc_commit": "2020-12-03T10:38:49",
}
```


Listing 3.1: Example of the JSON file per commit produced by GrimoireLab (only the main fields are included).

The output is a set of JSON files, one for each unique contributor, containing the targeted variables for all commits 3.6.

3.3.2 Building the Contributors dataset

Once having the commit data for all the contributors, the next step is building the main dataset for this project. As the *Revelio* tool is meant to be integrated with SortingHat, the decision was to use the unique individuals as entry values for the classification model, which will predict whether a given contributor is a bot or not.

Our Users-Commits dataset, extracted from GrimoireLab data stored in ElasticSearch, is

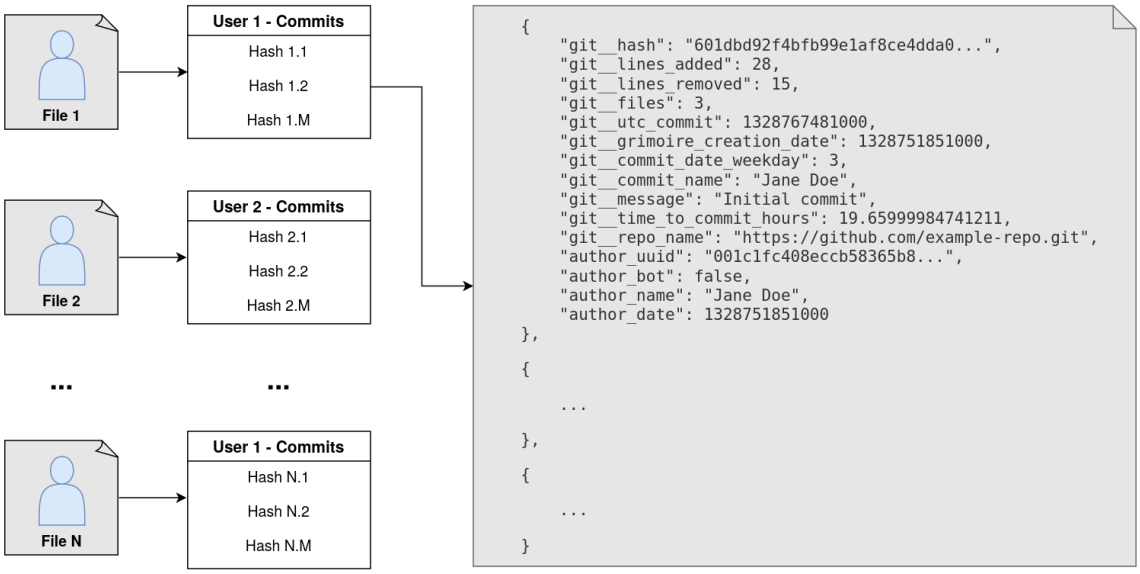

Figure 3.6: Graphic example of how data extracted from ElasticSearch look like: one file per author is produced, which contains a set of fields for all of its commits submitted within the time period for the analysis.

composed by single commits per author, and our new dataset needs to have one entry per contributor. Thus, we need to represent the information from the history of commits for each author in some way.

Following the metrics defined using the GQM approach, data are processed with a list of variables per contributor, as follows:

- The unique identifier of the author from SortingHat, used as the index field.
- The name of the author from SortingHat.
- The classification of the author as a bot or not, from SortingHat.
- The number of unique commits.
- The number of merge commits, which are the ones modifying no files.
- The number of commits submitted during weekends (either on Saturday or Sunday).
- The number of commits that have been signed off.
- The unique number of repositories an author contributed to.

Apart from these variables, another set is defined from statistical calculations:

- The ratio of merge commits, over the total number of commits.
- The ratio of commits submitted during weekends, over the total number of commits.
- The ratio of merge commits over the total number of commits.

- The ratio of signed commits over the total number of commits.
- The median and the interquartile range for:
  - the number of modified files per commit.
  - the number of added lines per commit.
  - the number of removed lines per commit.
  - the length of the commit message.
  - the number of words (including stop-words)[8] of the commit messages.

The output is a "Contributors" dataset, with one row per contributor, including the summarised information from its contributions in the period of time of the analysis.

3.4 Data processing

With the "Contributors" dataset composed, the next phase consists of exploring the data exhaustively, so it can be consumed by the different classification models we are testing for our problem.

3.4.1 Exploratory Data Analysis

For this last set of statistical calculations, a given amount of commits per author is required. In order to avoid considering data from pet projects and casual users (for example, some contributors only submit a small number of contributions as part of a learning course), an additional criterion is to ignore those authors having less than 10 commits for the selected time period.

After applying this first rule, the starting point is a dataset composed of $3,747$ rows, one for each contributor.

Looking at the statistical distribution of the variable we aim to classify, `author_bot` in Figure 3.7, it can be concluded that this is a highly unbalanced dataset: from these $3,747$ contributors, only 30 are marked as bots. This must be taken into account when creating the training, test, and validation datasets, and it also implies additional data processing in order to mitigate the effects of such an imbalance on the different classification models.

The remaining variables also have very uneven statistical distributions, except for the generated variables (median and interquartile range) for the length of the commit messages. This situation entails these variables should be transformed in order to approximate their distributions to a normalised version of themselves.

Furthermore, another issue to ponder is how these variables were generated as a summary of the activity of each contributor, which is independent, no relative information from other

[8]Words are counted as groups of characters split by white-space characters within a commit message. Stop-words are preserved, as they could be relevant when analysing patterns.

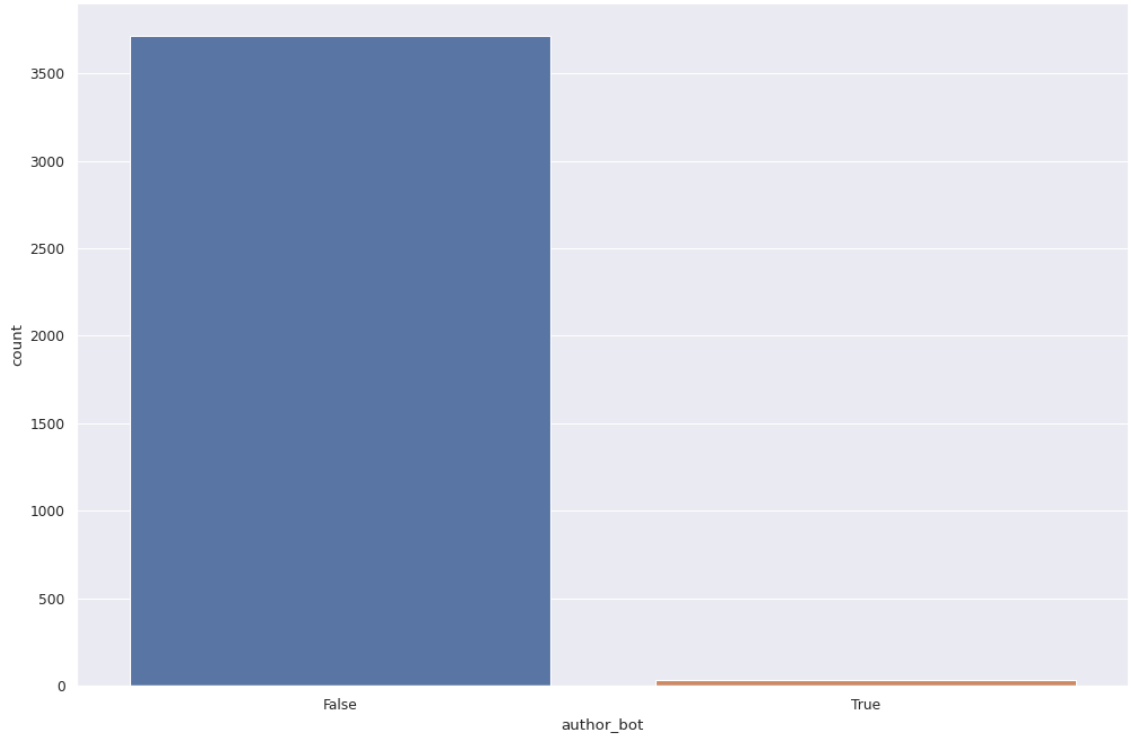

Figure 3.7: Proportion of contributors marked as a bot (False, on the left; True on the Right).

contributors is used to build them. Nonetheless, some of the numerical variables are not completely independent, as they were created to represent extra layers of information inferred from some of the other values, such as the variables counting the number of commits submitted during weekends, which is a subset of the information provided by the variable counting the total number of commits; or those whose value is a relative value computed as a ratio.

3.4.2 Building the training, test, and validation datasets

The main dataset needs to be split into subsets for the classification stage: one for training (60% of the samples), one for testing (25% of the samples), and another one for validation (15% of the samples), as it is represented in Figure 3.8.

These subsets need to be statistically similar, so the sample needs to be stratified in order to keep the human/bot proportion. The method `train_test_split` from Scikit-Learn's `model_selection` module was used to obtain these subsets, by using the option `stratify`.

3.4.3 Generation and selection of features

In this subsection, it is explained how the variables from the Users-Commits dataset have been transformed and selected for serving as the input for the classification models.

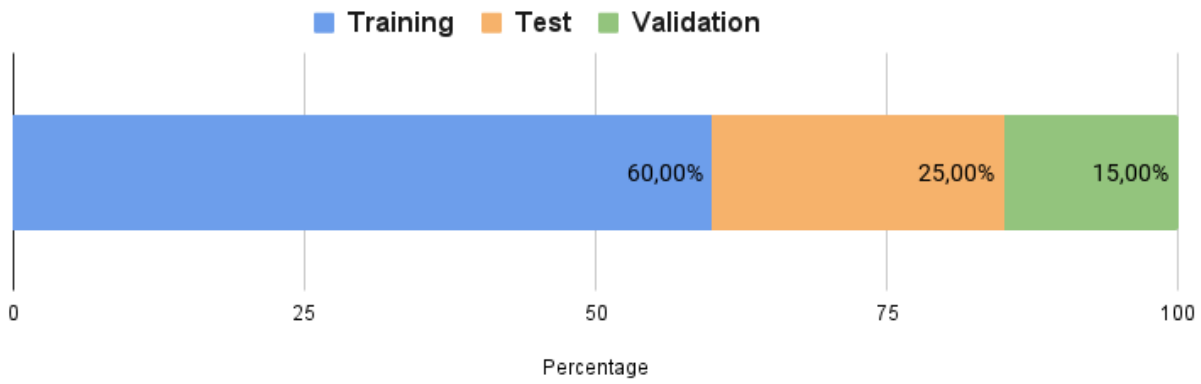

Figure 3.8: How the initial dataset was split (stratified) into Training, Test, and Validation sets, showing the percentage each of them represents out of the whole set.

Detection and processing of missing data

The data are provided by the GrimoireLab tool set, and the dataset is a collection of information from Git commits. The fields and the information per commit are expected to be standard.

As we are building in a later step a custom dataset summarising information per author, we are not having missing data except for author-related information, such as the author's name, username, or email (at least one of these three variables has to have a non-empty value).

Transformation of quantitative variables

The main variables from the dataset are generated in subsection 3.3.2, and most of them are quantitative variables, as they are representing a summary of the information obtained from the set of commits submitted by each unique author.

The first examination during the Exploratory Data Analysis phase (3.4.1) shows that most of these quantitative variables have remarkably uneven statistical distributions. There are huge differences in the ranges of values, and also these values are widespread. Under this situation, the approximation to take was to apply mathematical functions helping to scale these values. Although several transformations were tested, most of the variables have been transformed using a customised logarithmic function (see Section A.1) or the squared root function. The Table 3.2 summarises how the variables were transformed.

Transformation of qualitative variables

Although most of the variables from this dataset are quantitative, the statistical distribution of some of them points to a need to generate qualitative, binary variables which derive from them. Looking at the graphical representations, the referenced variables counting the number of commits under a given condition usually have some bias: observing the

statistical distribution, the general rule is finding a peak at 0 and then the rest of results are very spread over the histogram.

Following this reasoning, these variables were transformed into binary variables, with the intention of translating the major difference between these values in a conceptual way: For instance, looking at the number of *merge* commits, instead of counting how many of them there are, it could be enough to know just if a given author submitted merge commits or not.

On the other hand, I composed a list of heuristic terms (see Table 3.3) belonging to the application domain, using the terminology of tasks that bots are usually performing during the software development process. The first approach was to create a set of dummy variables with one column per term, where the value is 1 (**True**) when the term from that column is included in the author's name; and 0 (**False**) if it is not included. For instance, a user named `ghmerger` would have a 1 value in the 'merge' and 'merger' columns, and a 1 value in the rest of the columns.

Nonetheless, this approach was adding too much complexity to the system. After several experiments from chapter 4), the decision was to summarise the inclusion of these terms into a "terms score". The idea was to classify the heuristic terms in three different levels assigning weights to each of them, according to their relevance in the application domain. Doing so, an author name having one or more terms from the list would have a greater value, also taking into account their relevance by using these levels. Having a score of 0 means the `author_name` field doesn't include any of the relevant terms. The formula to compute this terms score is available at Section A.6, and the definition of these three levels is available at Table 3.4.

As a final note, our target variable `author_bot` is also a categorical, binary variable, as it is a property given by the GrimoireLab platform (the variable es 1 when a given author is marked as an automatic account in SortingHat, and 0 otherwise).

3.4.4 Correlation

To study the correlation between the variables in the dataset, in Figure 3.9 it is represented a correlation matrix with absolute values, taking into account the dataset with the transformed variables.

By representing the correlation matrix it was revealed there was a set of variables with high correlation values. The decision was to establish a hard threshold on correlation values greater than 0.75. In Figure 3.10, it is shown the pairs of variables which have correlation values above the threshold. After studying the relevance given the application domain; and also having into account that there were some variables that seemed redundant (the values from both the interquartile range and the median had a high correlation in most cases), the following variables were removed from the transformed dataset:

- `git__sqrt_ratio_merge_commits`

- `git__sqrt_ratio_weekend_commits`
- `git__sqrt_ratio_signed_commits`
- `git__log_iqr_lines_added`
- `git__log_iqr_lines_removed`
- `git__log_iqr_len_commit_message`
- `git__log_median_len_words_commit_message`

After these changes, the correlation values from the resulting dataset are detailed in Figure 3.11.

3.4.5 Imbalanced data

We already commented in the Exploratory Data Analysis Section (3.4.1) on the fact that one of the main challenges of this project is the imbalance in the target class we are aiming to detect. This context was taken into account when splitting the main dataset into the training, test, and validation tests, but it needs another processing stage before they feed the different classification models. Looking at the techniques that are commonly used to reduce the effect of imbalanced data, the one selected was **SMOTE** [2].

SMOTE consists of an algorithm generating new samples considering the k-nearest neighbours from each original sample from the **training set**. Each newly generated sample is interpolated between the original sample and one of the nearest neighbours; with a random component λ, which takes value in the range $[0, 1]$.

To apply SMOTE, we rely on the implementation under the `over_sampling` module from the Python library *imblearn* (2.2.4).

3.5 Classification model

The next step in the tool is running the classification model that allows to classify the contributors. The type of variables we have can be separated into two groups: those variables coming from the activity of each contributor and those variables we have generated from the contributors' names based on a set of heuristic terms.

The initial idea was to split the corresponding variables from these two groups into different subsets of the data, and each of the subsets would be evaluated by different classification models. Then, the output of both models can be directed to an ensemble model that would produce a final output.

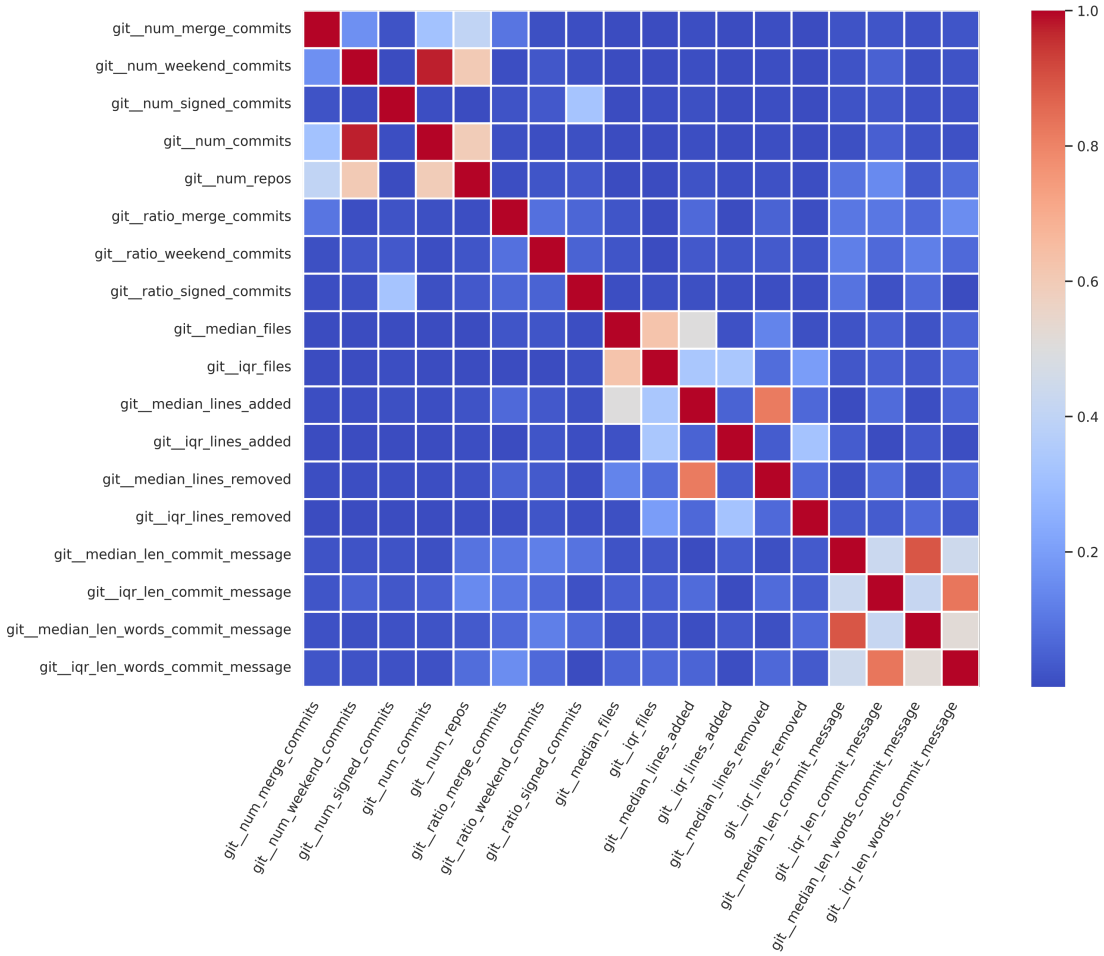

Figure 3.9: Correlation heat map of the initial variables from the training dataset.

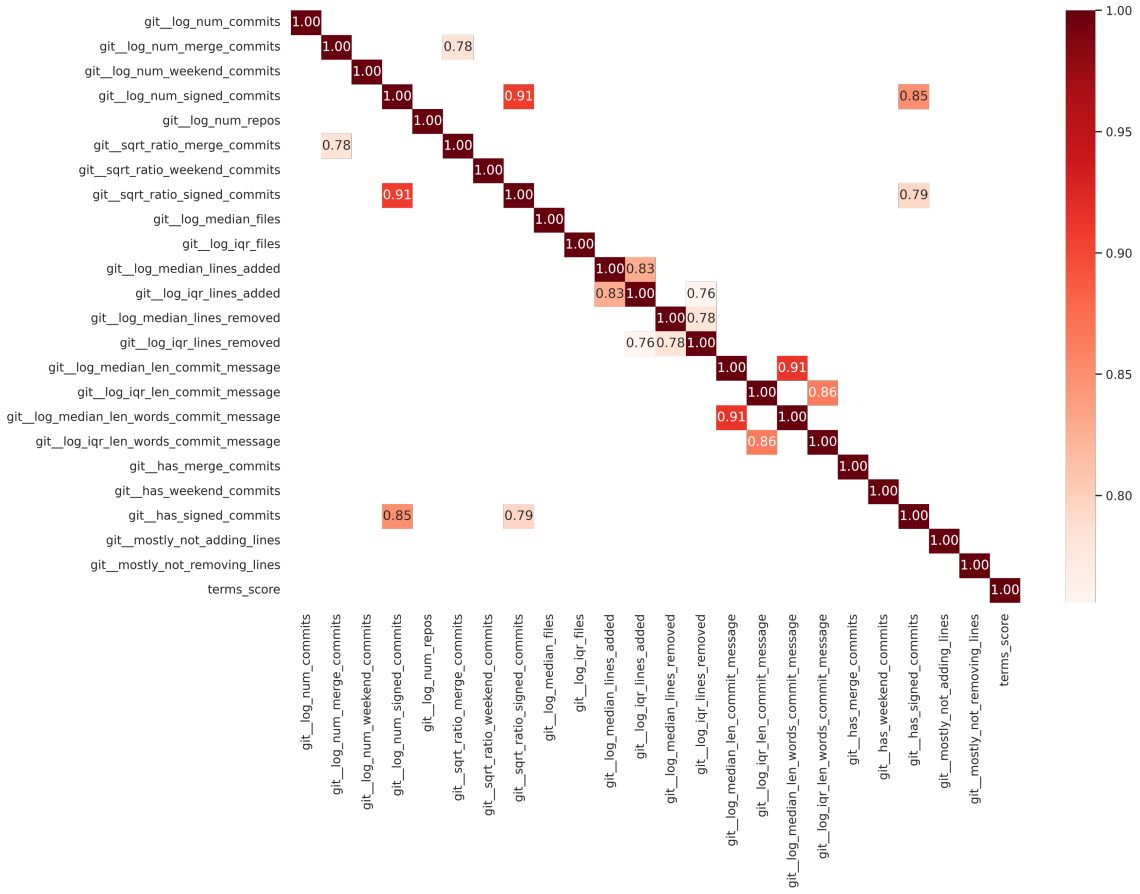

Figure 3.10: Correlation heat map showing the pairs of variables with an absolute correlation greater than 0.75 .

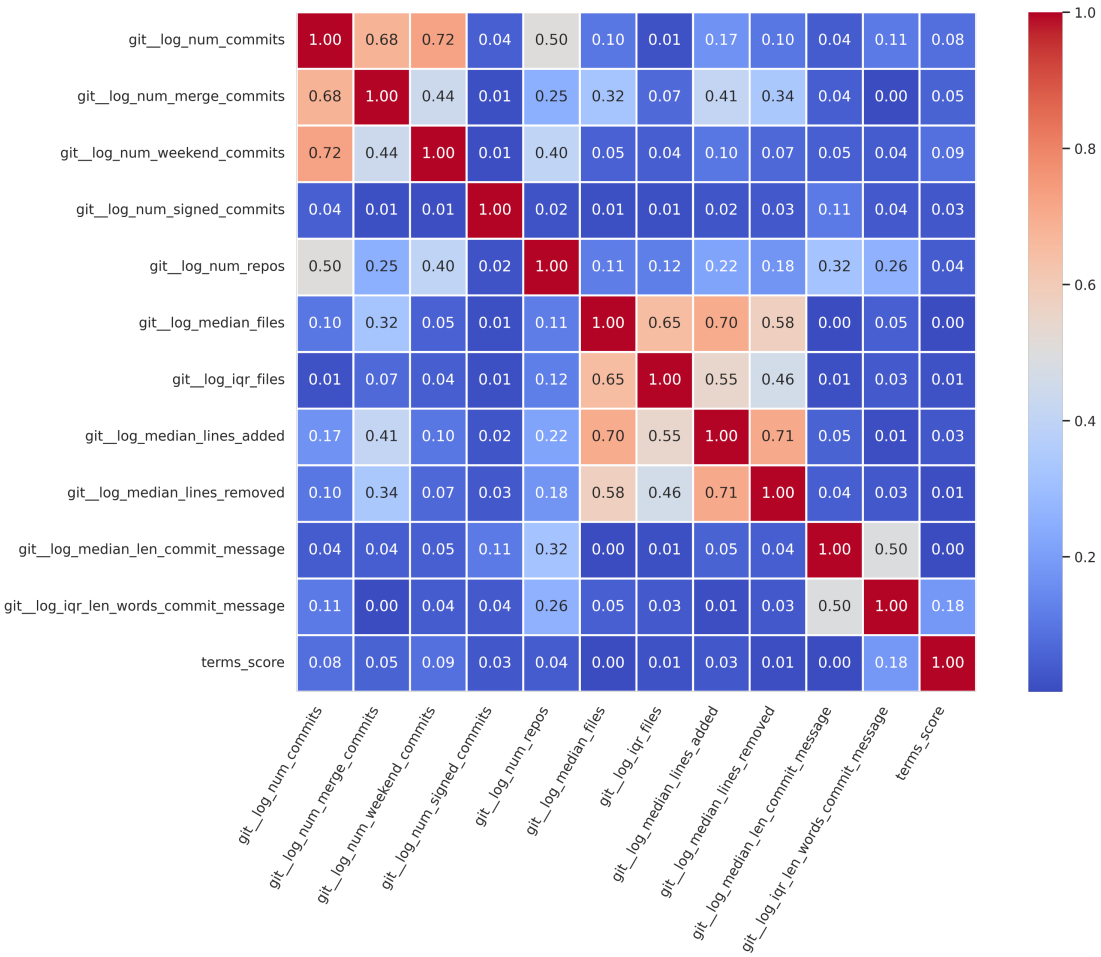

Figure 3.11: Correlation heat map of the transformed variables from the training dataset.

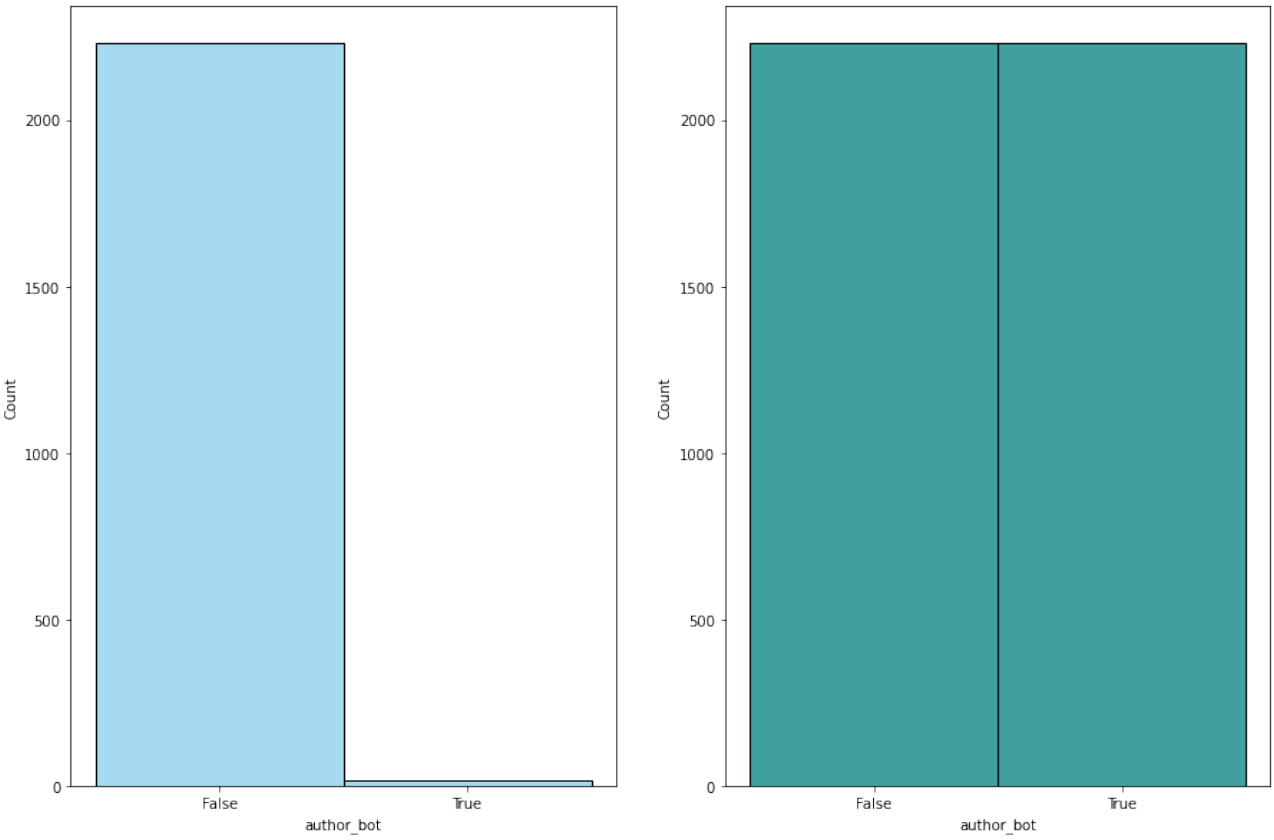

Figure 3.12: Effect of applying SMOTE over the training dataset on the target variable; before (left) and after (right)

3.5.1 Models definition

In this subsection, the classification models considered for this project are explained in more depth [9]. The decision was to select a set of models based on the ones used in the reference scientific papers described in chapter 2. The tests and results from applying these classification models to our dataset are presented in Section 4.2.

Naïve-Bayes

Naïve-Bayes supervised-learning algorithms are based on Bayes' theorem. They belong to the Probability-based learning family, and their approach is to use estimations of likelihoods to determine the most likely predictions that should be made and review them later, based on the available data and also extra evidence whenever it becomes available.

Naïve-Bayes classifiers are especially useful for problems with many input variables, categorical input variables with a vast number of possible values, and text classification. Among the advantages of using these classification models are their simplicity to apply (generally, no parameters to be adjusted) and their resistance to over-fitting.

The selected classifier was the **Gaussian Naïve-Bayes** algorithm implemented in Scikit-learn (`GaussianNB`[10], under `naive_bayes` module). Mathematically, the definition for this classification model can be expressed as follows:

Being $p_k(x)$ the posterior probability of an observation x to belong to the class k, which is defined as $p_k(x) = Pr(Y = k|X = x)$, in terms of $\pi_1, ..., \pi_k$ and $f_1(x), ..., f_k(x)$, where $\pi_1, ..., \pi_k$ are the prior probabilities and $f_1(x), ..., f_k(x)$ are the p-dimensional density functions for an observation in the k-th class for $k = 1, ..., K$; the Naïve-Bayes classifier makes a single assumption for estimating $f_1(x), ..., f_k(x)$ functions: Within the k-th class, the p-predictors are independent.

With this assumption, we can obtain an expression for the posterior probability,

$$P_r(Y = k|X = x) = \frac{\pi_k \times f_{k1}(x_1) \times f_{k2}(x_2) \times ... \times f_{kp}(x_p)}{\sum_{l=1}^{K} \pi_l \times f_{l1}(x_1) \times f_{l2}(x_2) \times ... \times f_{lp}(x_p)} \quad , \tag{3.1}$$

$$\text{for } k = 1, ...K.$$

Thus, to estimate the one-dimensional density function f_{kj} using training data $x_{1j}, ...x_{nj}$, if X_j is quantitative we can assume that, $X_j|Y = k \sim N(\mu_{jk}, \sigma^2_{jk})$. In other words, we assume that within each class, the jth predictor is drawn for a normal distribution.

Support Vector Classifier

The support vector classifier is based on the possibility of constructing a hyperplane that separates the hyperplane training observations perfectly according to their class labels.

[9] The mathematical definitions are extracted from the book "An Introduction to Statistical Learning" [7]

[10] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

Once this hyperplane exists, the ideal scenario is that a test observation is assigned to a class depending on which side of the hyperplane it is located.

Nonetheless, observations that belong to two classes are not necessarily separable by a hyperplane. In fact, even if a separating hyper- plane does exist, then there are instances in which a classifier based on a separating hyperplane might not be desirable. A classifier based on a separating hyperplane will necessarily perfectly classify all of the training observations; this can lead to sensitivity to individual observations and implies that it may have overfitted the training data.

That is, it could be worthwhile to misclassify a few training observations in order to do a better job discriminating the remaining observations. The support vector classifier, sometimes called a soft margin classifier, does exactly this. Rather than seeking the largest possible margin so that every observation is not only on the correct side of the hyperplane but also on the correct side of the margin, we instead allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.

Summarising, the support vector classifier classifies a test observation depending on which side of a hyperplane it lies. The hyperplane is chosen to correctly separate most of the training observations into the two classes, but may misclassify a few observations. Taking the mathematical definition of a p-dimensional hyperplane,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0 \quad , \tag{3.2}$$

the separating hyperplane is the solution to this optimization problem:

$$maximize\ M(\beta_0, \beta_1, ..., \beta_p, \epsilon_0, ...\epsilon_n, M) \quad , \tag{3.3}$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1 \quad , \tag{3.4}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geqslant M(1 - \epsilon_i) \quad , \tag{3.5}$$

$$\epsilon_i \geqslant 0, \quad \sum_{i=1}^{n} \epsilon_i \leqslant C \quad , \tag{3.6}$$

where C is a non-negative tuning parameter.

K-Nearest Neighbours

K-Nearest Neighbours is a similarity-based classification model whose main idea is to compute the classification from a simple majority vote of the nearest neighbours of each point: a query point is assigned the data class which has the most representatives within the nearest k (integer number) neighbours of the point.

Note that this algorithm uses the whole training dataset for making the predictions, and aside from other classification models, there are no specific assumptions that should be made concerning the data. One of the main setbacks is the fact that this algorithm is affected by noise, which implies this parameter k needs to be selected carefully, particularly when working with imbalanced datasets.

This model can be found as `KNeighboursClassifier`[11] in Scikit-learn, under `neighbours` module).

Mathematically, this classifier can be defined as follows: Given a positive integer K and a test observation x_0, the KNN classifier first identifies the neighbours K points in the training data that are closest to x_0, represented by N_0. It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:

$$P_r(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \quad . \tag{3.7}$$

Finally, KNN classifies the test observation x_0 to the class with the largest probability from equation 3.7.

Decision Tree / Random Forests

As explained in *Scikit-learn* documentation[12], the ***Decision Trees*** (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a set of if-else decision rules.

Providing a more academic definition, a classification tree predicts that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.

We use recursive binary splitting to grow a classification tree. Since we plan to assign an observation in a given region to the most commonly occurring class of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}) \quad . \tag{3.8}$$

[11] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[12] https://scikit-learn.org/stable/modules/tree.html

Here $\widehat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable. The Gini index is defined by

$$G = \sum_{k=1}^{K} \widehat{p}_{mk}(1 - \widehat{p}_{mk}) \quad , \tag{3.9}$$

a measure of total variance across the K classes. It is not hard to see that the Gini index takes on a small value if all of the $\widehat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node *purity*—a small value indicates that a node contains predominantly observations from a single class.

An alternative to the Gini index is entropy, given by

$$D = -\sum_{k=1}^{K} \widehat{p}_{mk} \log \widehat{p}_{mk} \quad . \tag{3.10}$$

Since $0 \leqslant \widehat{p}_{mk} \leqslant 1$, it follows that $0 \leqslant -\widehat{p}_{mk} \log \widehat{p}_{mk}$. One can show that the entropy will take on a value near zero if the $\widehat{p}_{mk}$'s are all near zero or near one. Therefore, like the Gini index, the entropy will take on a small value if the m-th node is pure. In fact, it turns out that the Gini index and the entropy are quite similar numerically.

When building a classification tree, either the Gini index or the entropy is typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than the classification error rate. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if the prediction accuracy of the final pruned tree is the goal.

Some of the main advantages of decision trees algorithm are:

- It is simple to understand and interpret. Trees can be visualised: if a given situation is observable, the explanation for the condition is easily explained by Boolean logic.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The most remarkable disadvantages are:

- DTs can create over-complex trees that do not generalise the data well (over-fitting).
- They can be unstable because small variations in the data might result in a completely different tree being generated.

- Decision-tree learners create biased trees if some classes dominate. In our case, this effect would be mitigated because we applied SMOTE to balance both classes.

Regarding the two first disadvantages, both can be addressed by using an ensemble model taking many decision trees. This is where the ***Random Forest*** (RF) classifier[13] comes into play: it builds a number of decision trees on bootstrapped training samples. When building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of $\sqrt{m}$ predictors is taken at each split, and typically we choose $m \approx p$—that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Then, the prediction of the ensemble is computed as the averaged prediction of these individual classifiers, improving the predictive accuracy and preventing over-fitting. The implementation of this classifier can be found as `RandomForestClassifier` (under the `ensemble` module) in Scikit-learn.

XGBoost Classifier

The XGBoost Classifier model belongs to the XGBoost library (see subsection 2.2.4). XGBoost is an ensemble model which uses decision trees as base learners. XGBoost uses CART trees (Classification and Regression trees), with scores on whether an observation belongs to a class or not. When this process reaches the max depth of the tree, the algorithm converts the scores into categories assigning a threshold value.

3.5.2 Evaluation metrics

We need to use a set of metrics that help us to evaluate the performance of the different classification models. The main method to compare the results from the different models is a confusion matrix, which displays the number of elements that have and have not been identified correctly. The structure of this matrix is exemplified in Table 3.6.

Looking at the possible values we can obtain, it is worth mentioning that not all the misclassified cases affect our use case in the same way: having "False" Negatives is worse than having "False" Positives. This means it is more important to classify as many bot accounts as possible (and not mistake any of them for a human) rather than classifying a human as a bot when it is not the case. In the first case, missing a bot account among the plethora of contributors in a community could mean that potentially this bot account remains hidden (and hardly going to be identified); while in the latter, this wrong recommendation could be just ignored.

This situation links directly to the definition of two basic metrics: *precision* and *recall*. As

[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

it is defined in Scikit-learn's documentation page[14], an intuitive definition of precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples.

[14] https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics

Precision and recall are defined mathematically as:

$$\text{Precision} = \frac{tp}{tp + fp} \tag{3.11}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{3.12}$$

Although it is common to use the *F1-score* as an evaluation metric for classification models, this score is considering that the recall and the precision are equally important. This is why the decision was to use a F_β score with $\beta = 2$, to penalise those classification models with a greater number of "False" Negatives.

Field name	Field description
author_bot	True if the given author is identified as a bot in SortingHat.
author_date	Author date (when the original author made the commit).
author_name	Author name from SortingHat.
author_uuid	Author UUID from SortingHat.
commit_date_weekday	Day of the week when the committer made the commit.
commit_name	Committer name
files	Number of files touched by this commit.
grimoire_creation_date	Commit date (when the original author made the commit).
hash	Commit hash.
lines_added	Number of lines added by this commit.
lines_removed	Number of lines removed by this commit.
message	Commit message as a single String.
time_to_commit_hours	Time in hours from author date to commit date.
repo_name	Repository the commit was submitted to.
utc_commit	Commit date in UTC.

Table 3.1: Selected fields from the git index produced by GrimoireLab.

Transformation	Feature
$Log_{10}(x)$	git__num_commits
	git__num_repos
$\sqrt{x}$	git__ratio_merge_commits
	git__ratio_weekend_commits
	git__ratio_signed_commits
$Log_{10}(1+x)$	git__num_merge_commits
	git__num_weekend_commits
	git__num_signed_commits
	git__median_files
	git__iqr_files
	git__median_lines_added
	git__iqr_lines_added
	git__median_lines_removed
	git__iqr_lines_removed
	git__median_len_commit_message
	git__iqr_len_commit_message
	git__median_len_words_commit_message
	git__iqr_len_words_commit_message

Table 3.2: Transformation applied to quantitative variables.

Common terms
auto, bot, build, cd, ci, code, commit, copy, dependency, fix, integration, issue, merge, patrol, pr, pull, release, request, review, sync, template, tool, and travis.

Table 3.3: Common terms used for the name, email, and/or username of automatic accounts.

Level	Weight	Heuristic terms
1	60	`bot`, `dependency`, `fix`, `integration` and `merge`
2	30	`auto`, `build`, `commit`, `copy`, `issue`, `release`, `request`, `review`, `sync`, `template`, `tool` and `travis`
3	10	`cd`, `ci`, `code`, `patrol`, `pr`, and `pull`

Table 3.4: Levels of heuristic terms and their assigned weights used for computing a term score.

Transformed feature	Meaning
`git__has_merge_commits`	1 if a given author has submitted at least one merge commit; 0 otherwise.
`git__has_weekend_commits`	1 if a given author has submitted at least one commit during Saturday or Sunday; 0 otherwise.
`git__has_signed_commits`	1 if a given author has submitted at least one signed commit; 0 otherwise.
`terms_score`	Integer value representing a score obtained from the author's name, computed as detailed in Section A.6.

Table 3.5: Transformation of qualitative variables.

		Predicted	
		Human	**Bot**
Real	**Human**	True Negative (TN)	False Positive (FP)
	Bot	False Negative (FN)	True Positive (TP)

Table 3.6: Example of confusion matrix to evaluate the classifiers' performance.

Chapter 4

Experiments and validation

In this chapter, the different experiments that were run over the different phases of the project are explained. Although the main experiments are those belonging to the testing and adjustment of the classification models, there were some efforts during the data processing that are worth mentioning.

4.1 Data processing: Analysing text

The following questions were proposed at the Specific objectives Section (1.4.2):

- **Q2.2.** Can the message content (commit messages, issue texts, etc.) be used to validate this classification?
  - **Q2.2.1.** Does a richer syntax give a hint about the nature of the user?
  - **Q2.2.2.** Can the entropy of a comment give a hint about the nature of the user?

Given that the initial dataset contains the history of commit messages for each user, the goal was to obtain a single, summary metric from these commit messages; so we can compare the results among the different users.

The main hypothesis from where this experiment starts is that it would be expected for bot users to have less dissimilarity among their commit messages than human users. Thus, in order to compare these comments from a given user, we decided to use a distance metric: the **Levenshtein distance** (A.3). Then, we would obtain distance matrices, so we could observe some properties, such as if the matrices are dense or sparse with respect to the average, for instance.

As this distance had to be computed per every single pair of messages, the result was a series of symmetric matrices, where the value from each position (i, j) was the Levenshtein distance from the message i to the message j; except from the main diagonal, whose values were 0 (there is no distance from one message to itself). This computation was largely costly, as the number of calculations per user is:

$$\text{NumCalculations} = \frac{(\text{NumMessages})^2}{2} \tag{4.1}$$

The decision to reduce the computation time was to limit the number of commit messages per user up to 1,500, selected randomly. This decision entails many problems, such as the temporal dependence of these commit messages. To avoid it, some main solutions were considered: one was to get the most recent commit messages, given the nature of the data, and the other one was to get a stratified sample by a given time period (months, years, etc.).

Aside from these complications, the next problem was to compare all these matrices by finding one metric capable of summarising the information from the matrix of distances. After exploring some options and reading some literature, an idea was to compute the *Mahalanobis distance* (A.5) between two matrices, taking into account that this distance is computed between two statistical distributions. Once computed, we would end up with a new feature characterising the set of values. When trying to compute the Mahalanobis distance between matrices, we encountered another problem. The algorithm computing this distance included, among its steps, computing the inverse matrix. It turned out some of the obtained matrices were singular (they had no inverse matrix). This last setback, together with the rest of the problems we encountered, was increasing too much the complexity of the experiment, so we decided not to continue further with it.

4.2 Choosing the classification model

In this phase of the experiment process, the goal is to train the classification models we defined (3.5.1) and test their performance against our dataset. Then, after having the results using the proposed evaluation metrics (3.5.2) we will choose the best classifier. This whole process is detailed in Figure 4.1, and it starts with the splitting of our initial dataset into three sets (Training, Test, and Validation), following the criteria we discussed at subsection 3.4.2 and then apply the pre-processing step: the transformation and selection of features (over the Training and Test datasets) and also applying SMOTE to mitigate the effect of the imbalance of the classes.

At this point, the decision was to show how the samples from the Training set were distributed using the t-SNE algorithm, a nonlinear dimensionality reduction technique [12]. This way we could convert our high-dimensional dataset into a two-dimensional one, preserving the distance between the samples in the new dimensional space. In Figures 4.2 and 4.3 we can observe the resized Training dataset before and applying SMOTE, respectively: In the first image (Figure 4.2), we can observe very few occurrences of positive bot accounts, and heavily mixed among the rest of the samples from the other class; while in the second image (Figure 4.3) we can observe a much clearer distinction between the two classes, after the synthetic samples generated by SMOTE. Apart from the interesting pat-

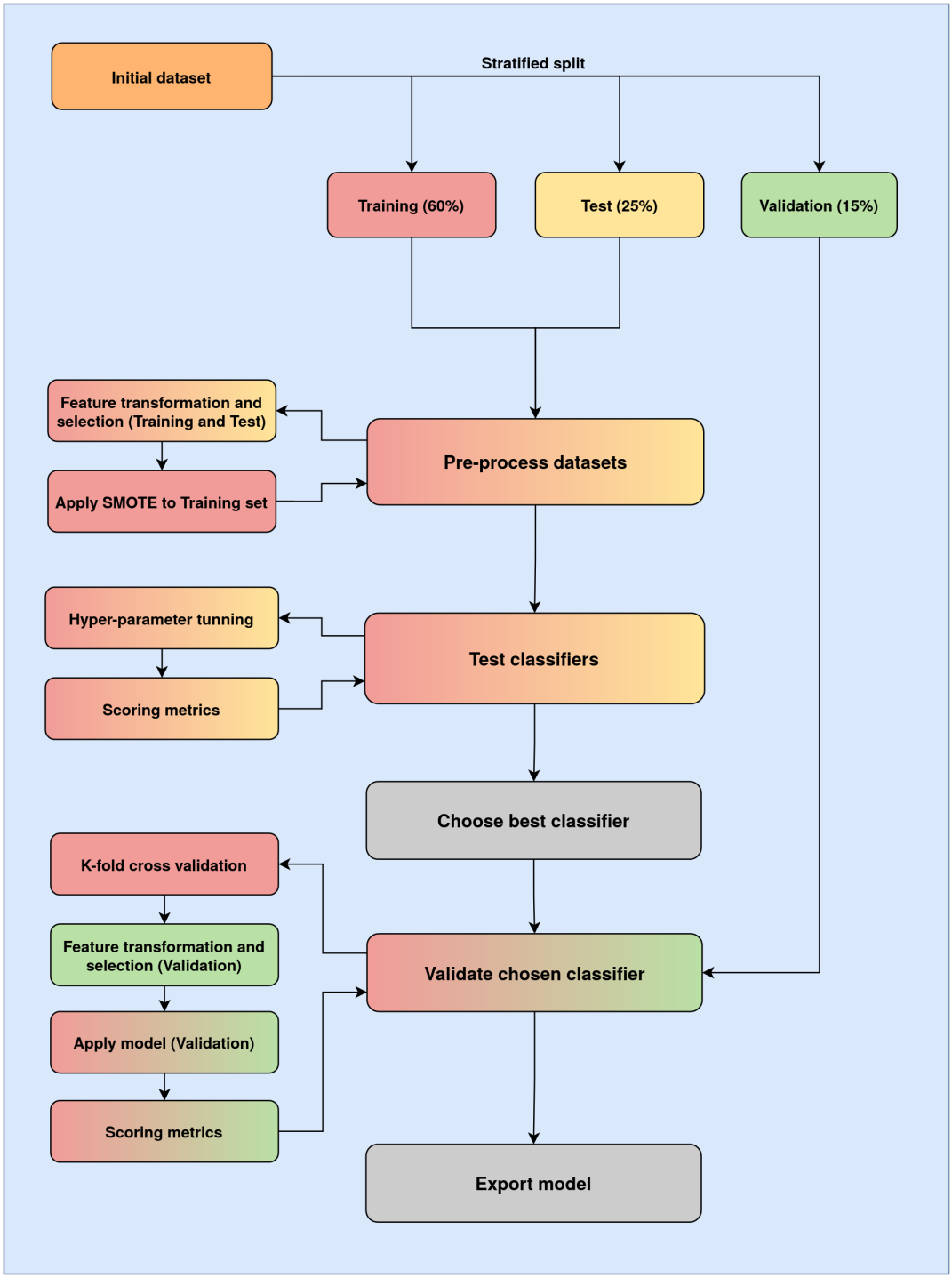

Figure 4.1: Description of the classification process. Background colours for each box explain which datasets (Training, Test and Validation) are involved in each step.

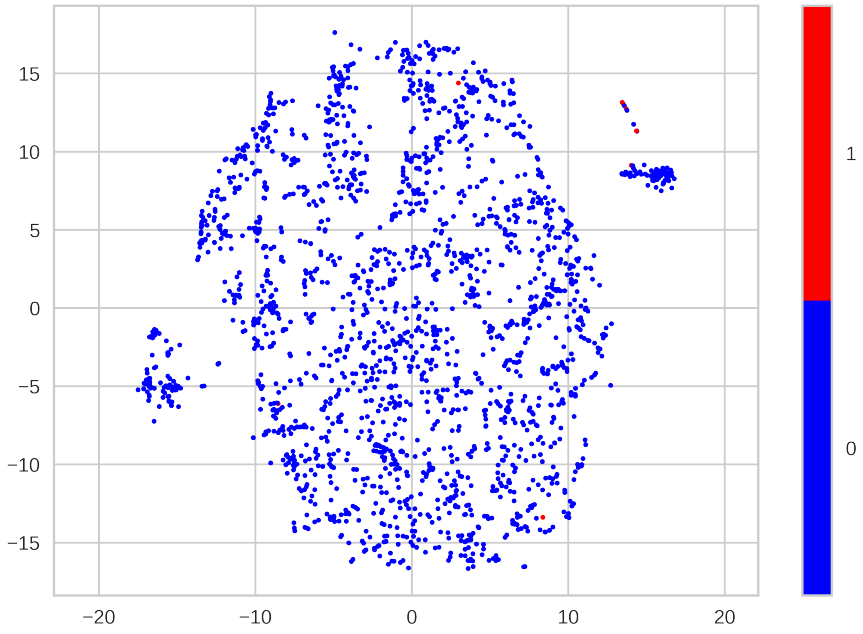

Figure 4.2: Visualising the Training dataset with t-SNE (Blue (0): Human, Red (1): Bot).

tern these samples are forming, we infer that there should be a classification model capable of separating both classes.

The proposed classifiers were trained and then tested, adjusting the specific hyper-parameters for each model until finding the best scoring (see subsection 3.5.2) for each of them. Furthermore, we applied PCA[1] to discover if there was a combination of features that would suit as input for the classification model, but we discarded it as the results indicated that one component accumulated most of the percentage of variance explained.

The results from the tested classifiers are summarised in Table 4.1. The classification model with the best results was the Random Forest Classifier, with $F_\beta = 0.811$ using the **Test dataset**. According to the corresponding confusion matrix (4.3), 6 out of 7 bot accounts were properly classified, and 826 human accounts out of 829. The Precision-Recall curves for the Test and Validation sets can be observed in Figures 4.4 and 4.5, respectively. These results were also tested using a *5-fold* cross validation.

The parameters that worked best for the Random Forest Classifier were:

- Number of estimators (Trees in the forest): 300.
- Split criterion: Gini impurity.
- Maximum depth: 4 levels.

[1]Principal Component Analysis

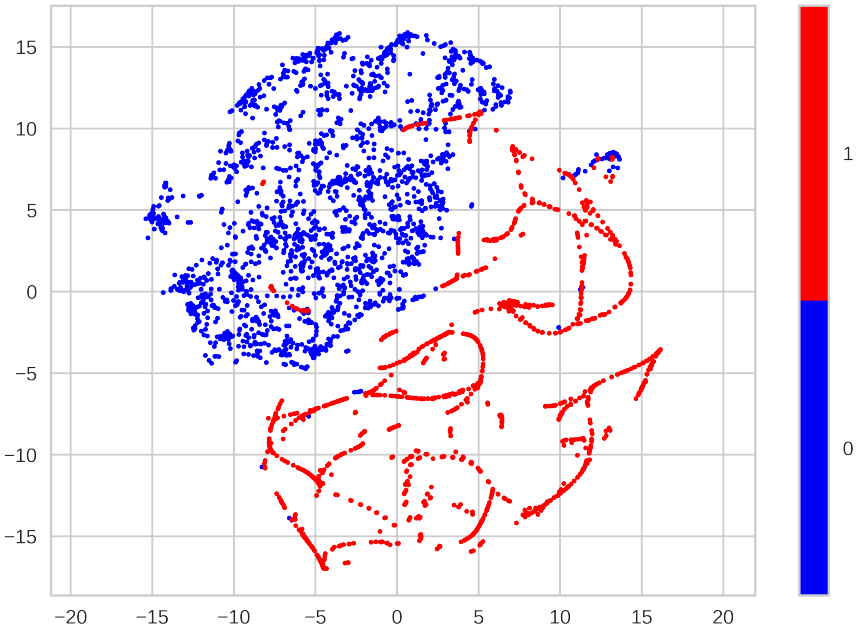

Figure 4.3: Visualising the Training dataset with t-SNE after applying SMOTE (Blue (0): Human, Red (1): Bot).

When trying these results with the **Validation dataset**, the obtained score was $F_\beta = 0.6$, obtaining the classification values displayed in Table 4.2 and in the corresponding confusion matrix (4.4), given that there were only 4 occurrences of bot accounts, three of them were classified correctly and only one was not. Regarding the human accounts, 493 out of 499 accounts were classified accurately.

Looking at feature importance values (see Figure 4.6) obtained from our chosen classifier, it is clear that the terms score variable we produced was the most relevant for deciding the classes, followed by the logarithmic transformation of the interquartile range of the number of words in the commit messages, with a relative importance of 60%. Then, the logarithmic transformation of the median number of files and the number of commits have a relative importance of around 20%, while the rest of the variables are barely significant for the classification.

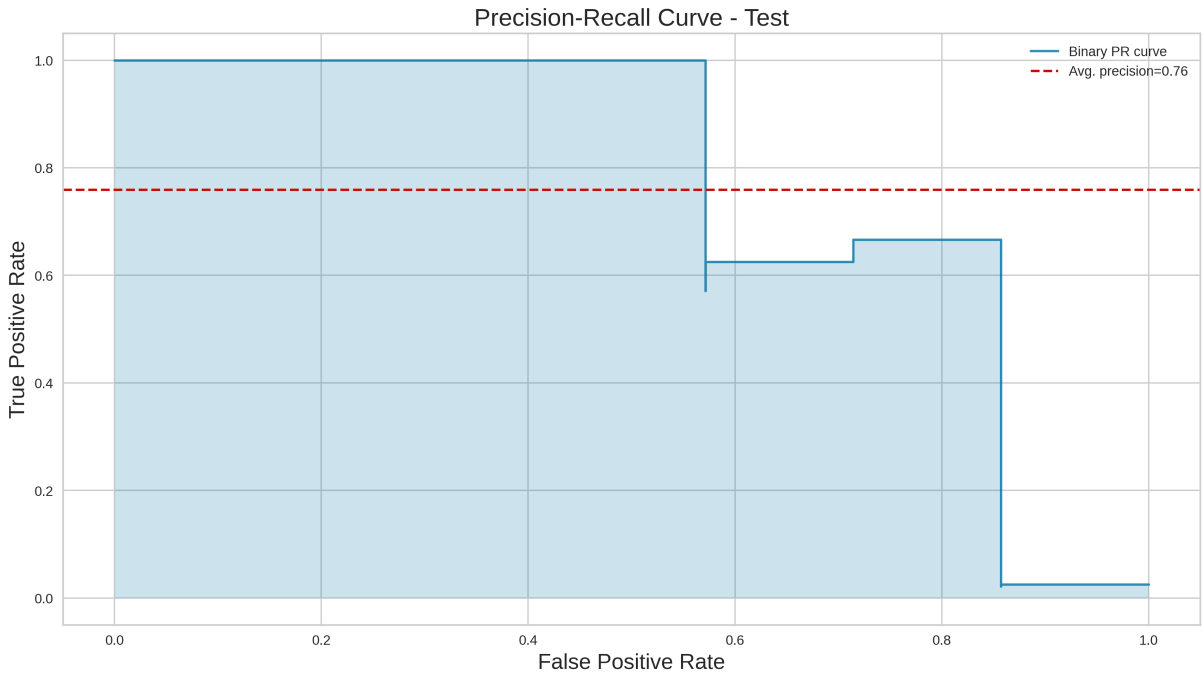

Figure 4.4: Precision-Recall Curve corresponding to the results with the test dataset.

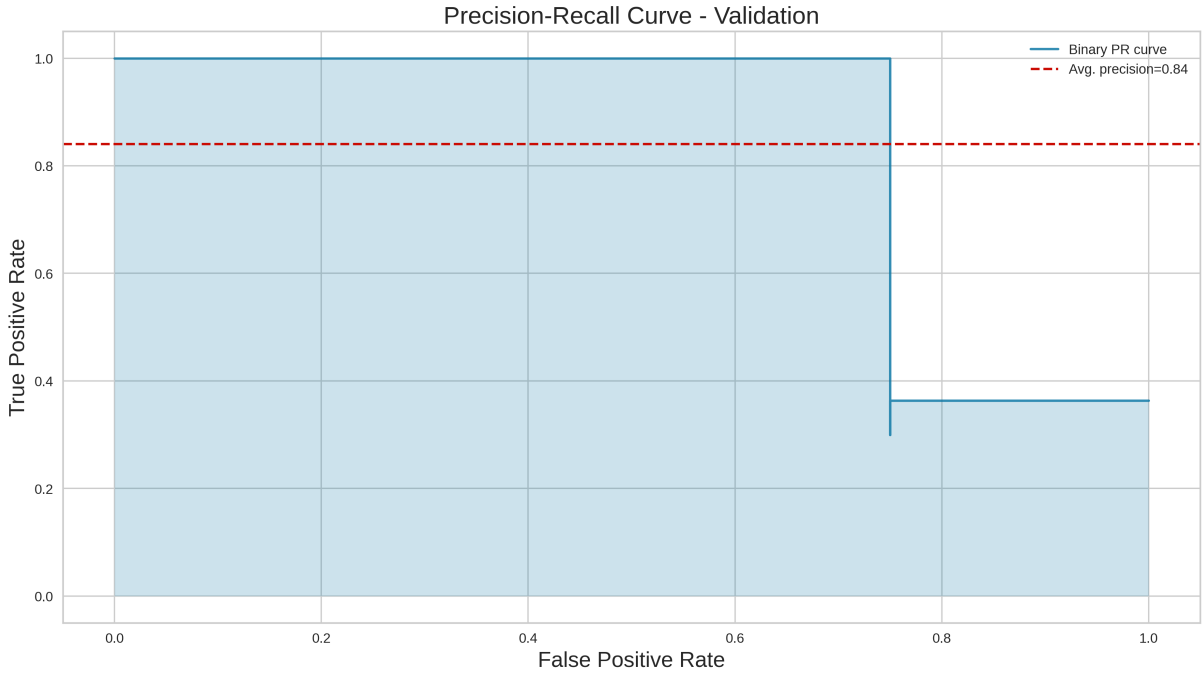

Figure 4.5: Precision-Recall Curve corresponding to the results with the validation dataset.

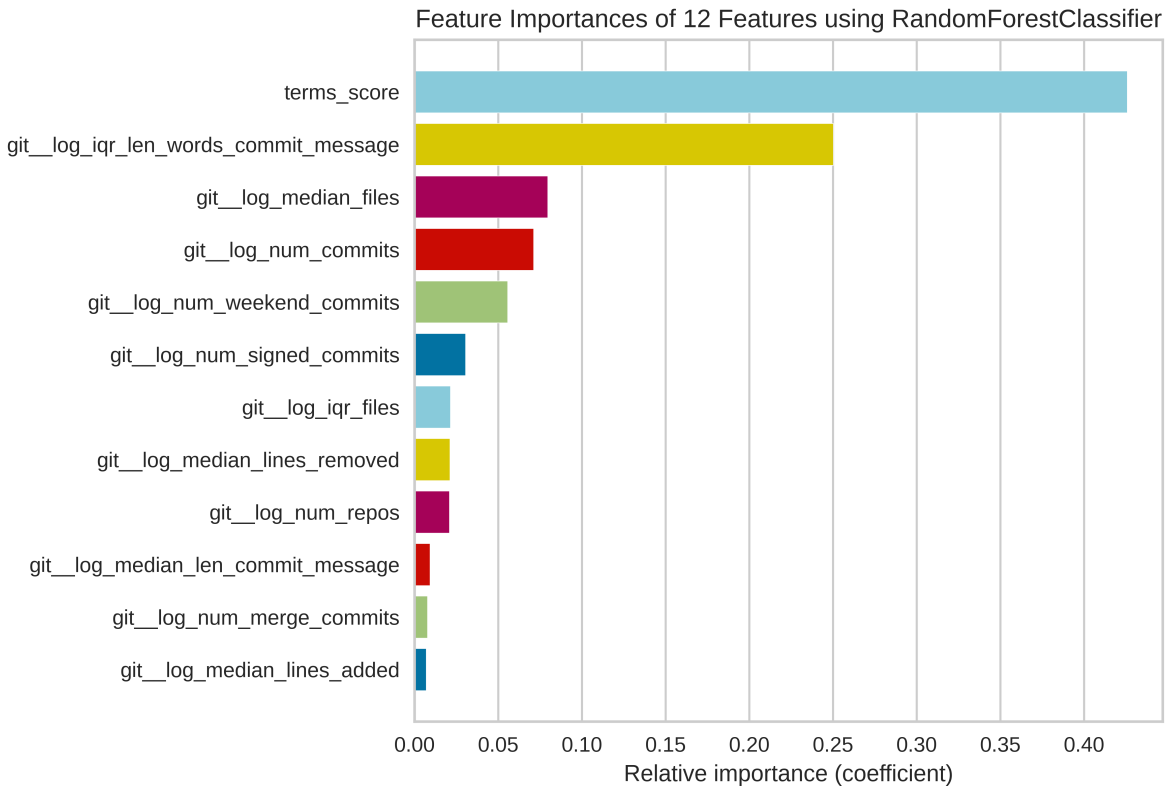

Figure 4.6: Feature importance for the Random Forest Classifier, displayed in descending order.

Model name	**Precision**	**Recall**	F_β **score**
Gaussian Naive-Bayes	0.136	0.857	0.417
Complement Naive-Bayes	0.167	0.857	0.469
LinearSVC	0.143	0.857	0.429
KNN	0.316	0.857	0.638
Decision Tree	0.385	0.714	0.61
Random Forest	0.667	0.857	0.811
XGBoost	0.444	0.571	0.541

Table 4.1: Results of the different classifiers showing the most relevant scores. The coloured row indicates the model with best overall results over the Test dataset.

Model name	**Precision**	**Recall**	F_β **score**
Random Forest	0.333	0.75	0.6

Table 4.2: Results of applying the chosen classifier to the Validation dataset.

		Predicted	
		Human	**Bot**
Real	**Human**	826	3
	Bot	1	6

Table 4.3: Confusion matrix of the results with the test dataset ($F_\beta = 0.811$). The green-coloured cells represent the cases where the predicted and the real value match; the red-coloured ones represent the cases where the predicted values did not match the real ones.

		Predicted	
		Human	**Bot**
Real	**Human**	493	6
	Bot	1	3

Table 4.4: Confusion matrix of the results with the validation dataset ($F_\beta = 0.6$). The green-coloured cells represent the cases where the predicted and the real value match; the red-coloured ones represent the cases where the predicted values did not match the real ones.

Chapter 5

Conclusions

In this chapter, we recap the outcomes, achievements, and limitations of this project, after the objectives described in Section 1.4.

This project has been a long and complicated run. Although the main idea for the project was clear, the nature of the target problem implied a huge variety of problems and challenges. Likewise, the project developments was affected by a global pandemic, which delayed and hinder its completion. Despite having faced many setbacks and limitations along the process, it is safe to say we are now able to provide answers to our initial objectives and also, enlighten the road for future work on this matter. Some of our hypotheses could not be bested, and some solutions did not work; but on the other hand, we found new research lines and solutions to explore in the future, gaining very valuable knowledge and experience along the journey.

5.1 Goal achievements

Regarding **Goal 1**, we can say we have achieved a well-defined process, including a classification model that works with the data obtained from the GrimoireLab toolset that allows us to discriminate between human users and bot accounts with reasonable accuracy. Due to the complexity of this project, the full automation and the integration was decided to be part of the future work (This links to the response to **Q1.4.**, available in subsection 5.4.2).

About the questions **Q1.1.** to **Q1.3.**: according to the results from the experiments, bot accounts can be separated from human users primarily by computing a score based on the terms included in their profile information and also computing a metric about the interquartile range of the number of words in the commit messages for a given individual, among other less-relevant features. Although the profile information from that individual is crucial for performing the classification, we can say it is not enough, and we need more features from the activity. The differences between this activity generated by humans and bots exist based on our observations, but we were not able to capture them in

highly-relevant features for our classification model. Furthermore, we performed an experiment where the features were divided into two different subsets: one subset included the variables obtained from the activity, and the other the ones from the profile information. Then, the output from both classifiers was submitted to a voting classifier, but the results were not promising.

Following with **Goal 2**, we can provide a partial answer to **Q2.1.**, as we only have obtained data from Git commits. The outcome could be complemented by analysing other channels and finding another set of footprints that can lead to better results, also categorised as future work to improve this project.

To answer **Q2.2.**, we have observed that indeed the message content (such as the commit messages) is relevant to perform this classification, but more experiments would need to be run exploring more what we started at the experiment detailed in Section 4.1 to give a more meaningful answer. Furthermore, to answer **Q2.3.**, we have observed that some of the activity details helped, such as the median number of files modified in each commit and the number of commits submitted during weekends, but their relevance was minor compared to the features detailed in the last section.

Last but not least, considering the **Goal 3**, we can safely say we have achieved a curated dataset from a real open-source community, in this case, Git commits from projects belonging to the Wikimedia Foundation (**Q3.1.**, **Q3.2.**). The curation of the identity information from the data we obtained was a hard, time-consuming task -even after having bot accounts identified by the community-, as all the unique identities were reviewed manually to get a dataset as accurate as possible. This process included cases that included mixed accounts, as they belonged to a human but also had submitted automated contributions using scripts or other processes. These cases, among others, would fit more in-depth research and another line of work for our classification tool.

5.2 Knowledge application

Throughout this master's degree, I have acquired knowledge of important concepts and tools through various courses and developed the ability to tackle new challenges and apply my knowledge in different scenarios.

Specifically, these were the most relevant courses for this project:

1. **Fundamentos de Análisis de Datos** (Fundamentals of Data Analysis): This introductory course provided me with the context of the cycle of Data Science and the process behind a complete analysis.

2. **Machine Learning I** and **II**: These courses were especially valuable because their contents focus on classification and prediction models and a variety of techniques used along the process.

3. **Programación Orientada a Ciencia de Datos** (Data Science-oriented Programming):

This course gave me the opportunity to gain more experience and knowledge to deal with the Data Science-related Python modules and tools.

4. **Text Mining**: In this course, I discovered a new field to explore, with concepts and techniques necessary for processing natural language.

5.3 Lessons learned

These are some of the learning outcomes I have reached thanks to this project:

1. Consolidate and amplify the knowledge I obtained during the Master's courses, after applying a real-life use case and facing a variety of problems along the process.
2. Foster my abilities in dealing with the programming side of data exploration and analysis, including new modules and techniques.
3. A valuable perspective about the research process, with its peaks and valleys, finding new lines of work and new opportunities, even when the initial results are not good.

5.4 Future work

The complexity of this project increased unexpectedly due to the nature of the problem we aimed to solve. This left several questions unanswered, pending work and many research lines to be explored in order to improve the classification results and applicability of this tool.

5.4.1 Improving and extending the classifier

Question: Does this classifier work with the data from other open-source communities and other projects using bot accounts?

- Once the classifier is trained for a given open-source project, is the result valid for other projects from the same community? And what about other communities?

To answer this question, we would need to test the current trained model with a different dataset. Also, this would help to answer if the obtained results are valid for other software projects and open-source communities or if there is a structured process that can be followed to adjust a set of parameters and improve the classifier's performance. For instance, one idea could be to include the heuristic terms in an external configuration file and categorise them in the three levels that were proposed. This list of terms could be adapted according to the project's needs and context.

Question: Are there other footprints that are helpful for this classification?

First, there could be a better way to build the initial dataset from the individual commits by finding other methods to summarise the information from the contributions of each individual. We also tried to separate the input features into two different categories: the

ones coming from the activity and the ones coming from the individual's profile. The idea was to use separate classification models and choose the best result using a Voting classifier, but the results were not good.

Some of the strongest lines of future work are:

- Keep studying the commit messages: We could not finish the experiment with the text distances. This and other related features and processes could help to improve this classifier.
- To augment this model by using the features from other data sources supported by GrimoireLab tools, such as GitHub Issues, Pull Requests and Comments.
- Explore "Concept Learning" [6] techniques to build the summarised information from all the contributions of a given individual and update the dataset incrementally (the more contributions an individual makes over time, the more information can be obtained from it).
- Mixed accounts (i.e., humans using automated tools) need to be studied more deeply. The classification would not be binary anymore but multi-class, leading to the possibility of characterising these automatic accounts using a smaller level of granularity.

5.4.2 Integration with SortingHat

This integration with GrimoireLab's identity management system was part of the initial design of the tool's architecture, but we decided to take this part out of the project's main scope.

In order to get *Revelio* working with SortingHat, the trained model would need to be exported, and the classification results should be turned into a formatted report, including the identities potentially identified as bots and an accuracy score. This output would be used for SortingHat's recommendation engine, which produces a list of recommendations, in this case, a list of individual profiles that might get labelled as bots: The recommendations would be returned in "Recommendation-like" objects, referencing the individual, if it was classified as a bot, and an accuracy value from 0 to 100 (see Listing 5.2). Also, unit and integration tests must be taken into account.

```
class BotRecommendation(EntityBase):
    individual = ForeignKey(Individual, on_delete=CASCADE)
    is_bot = BooleanField(default=False)
    accuracy = PositiveIntegerField()

class Meta:
    db_table = 'bot_recommendations'
    unique_together = ('individual',)

def __str__(self):
    return '%s - %s - %s' % (self.individual, self.is_bot, self.accuracy)
```

Listing 5.2: Proposed class for SortingHat's recommendation engine to include the results of the classification.

Appendix A

Definitions

A.1 Shifted logarithm

The decimal logarithm is shifted by 1 to cover variables containing zero values.

$$\mathrm{sLog}_{10}(x) = \mathrm{Log}_{10}(1 + x) \quad . \tag{A.1}$$

A.2 Jaccard distance

The Jaccard distance $J(c_1, c_2)$ measures the distance between two character sequences c_1 and c_2 by comparing the number of distinct common words in c_1 and c_2 with the total number of distinct words in c_1 and c_2.

$$J(c_1, c_2) = 1 - \frac{|words(c_1) \bigcap words(c_2)|}{|words(c_1) \bigcup words(c_2)|} \quad . \tag{A.2}$$

A.3 Levenshtein distance

The Levenshtein distance $Lev(c_1, c_2)$ measures the difference between two character sequences c_1 and c_2 by counting the minimum number of single-character edits (insertion, deletion, or substitution) required to convert c_1 into c_2. The normalised version is computed as:

$$L(c_1, c_2) = \frac{Lev(c_1, c_2)}{max(|c_1|, |c_2|)} \quad . \tag{A.3}$$

A.4 Combination of Jaccard and Levenshtein distances

The combination of the Levenshtein and Jaccard distances from Mehdi et al. article is defined as follows:

$$D(c_1, c_2) = \frac{L(c_1, c_2) + J(c_1, c_2)}{2} \quad , \tag{A.4}$$

where c_1 and c_2 are two character sequences.

A.5 Mahalanobis distance

The Mahalanobis distance is a measure of the distance between a point P and a distribution D [13].

Given a probability distribution Q on $\mathbb{R}^N$, with mean $\vec{\mu} = (\mu_1, \mu_2, \mu_3, \ldots, \mu_N)^T$, and positive-definite covariance matrix S, the Mahalanobis distance of a point $\vec{x} = (x_1, x_2, x_3, \ldots, x_N)^T$ from Q is:

$$d_M(\vec{x}, Q) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})} \quad . \tag{A.5}$$

A.6 Terms score

This terms score takes into account using different weighted heuristic terms, according to the knowledge domain. The terms are divided into 3 different levels: terms from level 1 have a weight of 60, terms from level 2 have a weight of 30, and terms from level 3 have a weight of 10.

The score is calculated as:

$$Ts(term) = 60Nl_1 + 30Nl_2 + 10Nl_3 \quad , \tag{A.6}$$

where:

- Nl_n is the number of heuristics from level n included in the term.

References

[1] Victor R. Basili, Gianluigi Caldiera, and Dieter H. Rombach. "The Goal Question Metric Approach." In: vol. I. John Wiley & Sons, 1994.

[2] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique." In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[3] Laura Dabbish et al. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository." In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. CSCW '12. Seattle, Washington, USA: Association for Computing Machinery, 2012, pp. 1277–1286. ISBN: 9781450310864. DOI: 10.1145/2145204.2145396.

[4] Tapajit Dey et al. "Detecting and Characterizing Bots That Commit Code." In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR '20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 209–219. ISBN: 9781450375177. DOI: 10.1145/3379597.3387478.

[5] Linda Erlenhov, Francisco Gomes de Oliveira Neto, and Philipp Leitner. "An Empirical Study of Bots in Software Development: Characteristics and Challenges from a Practitioner's Perspective." In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 445–455. ISBN: 9781450370431. DOI: 10.1145/3368089.3409680.

[6] Peter Flach. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. First. Cambridge, 2012, pp. 104–128.

[7] Trevor Hastie Gareth James Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning*. Second. Springer, 2021.

[8] Mehdi Golzadeh et al. "A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments." In: *Journal of Systems and Software* 175 (2021), p. 110911. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.110911.

[9] Hadi Hemmati et al. "The MSR Cookbook: Mining a decade of research." In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013, pp. 343–352. DOI: 10.1109/MSR.2013.6624048.

[10] Philipp Hukal et al. "Bots Coordinating Work in Open Source Software Projects." In: *Computer* 52.9 (2019), pp. 52–60. DOI: 10.1109/MC.2018.2885970.

[11] Yuxing Ma et al. “World of Code: An Infrastructure for Mining the Universe of Open Source VCS Data.” In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019, pp. 143–154. DOI: 10.1109/MSR.2019.00031.

[12] Laurens van der Maaten and Geoffrey Hinton. “Viualizing data using t-SNE.” In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605.

[13] Goeffrey J McLachlan. “Mahalanobis distance.” In: *Resonance* 4.6 (1999), pp. 20–26.

[14] David Moreno-Lumbreras et al. “SortingHat: Wizardry on Software Project Members.” In: May 2019. DOI: 10.1109/ICSE-Companion.2019.00036.

[15] Ravishankar Somasundaram. *Git: Version control for everyone*. Packt Publishing Ltd, 2013.

[16] Ben Straub and Scott Chacon. *Pro Git*. Second. Apress, 2014.

[17] Guido Van Rossum et al. “Python Programming Language.” In: *USENIX Annual Technical Conference*. Vol. 41. 2007, p. 36.