USING MACHINE LEARNING FOR DETECTING SECURITY VULNERABILITIES THROUGH BUG REPORT ANALYSIS
Caio Walter Dezidério Wanderley
1 2 9 0 UNIVERSIDADE D COIMBRA

UNIVERSIDADE D
COIMBRA

Caio Walter Dezidério Wanderley

USING MACHINE LEARNING FOR DETECTING SECURITY VULNERABILITIES THROUGH BUG REPORT ANALYSIS

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Nuno Laranjeiro and Prof. César Teixeira and presented to
Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Using machine learning for detecting security vulnerabilities through bug report analysis

Caio Walter Dezidério Wanderley

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Nuno Laranjeiro and Prof. César Teixeira and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering..

October 2021

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

A bug report is the description of an error in the program that was encountered by a developer. With the increasing amount of complexity in software systems, they are prone to have several bugs including those that could reveal sensitive information or allow for attackers to run malicious software. This is especially true for banks and large scale companies, in which a security bug that reveals the users' credentials or leaves their platforms vulnerable to malicious attacks, could affect human lives.

All modern large scale companies that have to create a software project have encountered bugs and have to fill in a bug report, and it is the job of a triagger to classify it according to its description. This task done by humans is very time-consuming and prone to a lot of error if the triagger does not know certain areas the report could be mentioned, this can lead to erroneous classification. So in recent decades, several studies are implementing text classification to classify these reports.

This thesis, it is first presented an analysis regarding the different approaches in the literature of the past decade for classifying Bug Reports. After the analysis of the literature, we experimented with different combinations of machine learning algorithms to determine the different impacts in the performance when dealing with vulnerabilities classification. We found that the results for the Area Under The Curve (AUC) being 81.21% $\pm$ 8.33 when using Title and Description of a bug report, 79.51% $\pm$ 7.96 when using the Title, and 78.04% $\pm$ 8.2 when using the Description.

Keywords

Vulnerability Classification, Security Issues, Security Bug classification, Security Bug Reports, Machine learning, Bag-of-words.

This page is intentionally left blank.

Resumo

Um relatório de bug é a descrição de um erro no programa que foi encontrado por um desenvolvedor. Com a crescente complexidade dos sistemas de software, eles estão propensos a ter vários bugs, incluindo aqueles que podem revelar informações confidenciais ou permitir que invasores executem software malicioso. Isso é especialmente verdadeiro para bancos e empresas de grande porte, em que um bug de segurança que revela as credenciais dos usuários ou deixar uma plataforma vulnerável para ataques maliciosos, podem afetar vidas.

Todas as empresas de grande porte modernas que têm que criar um projeto de software encontraram bugs e terão que preencher um relatório de bug, e é o trabalho de um triagger para classificá-lo de acordo com sua descrição. Essa tarefa feita por humanos é muito demorada e sujeita a muitos erros se o triagger não tiver o conhecimento de certas áreas que o relatório poderá estar mencionando, isso pode levar a uma classificação incorreta. Por causa disso, nas últimas décadas, vários estudos estão implementando a classificação de texto para classificar esses relatórios.

Nesta tese, é apresentada inicialmente uma análise a respeito das diferentes abordagens na literatura das última década para classificar relatórios de bugs. Após a análise da literatura, nós experimentamos diferentes combinações de algoritmo de machine learnig para determinar os diferentes impactos no desempenho ao lidar com a classificação de vulnerabilidades. Foi descoberto que os melhores resultado para o AUC sendo 81.21 % $\pm$ 8.33 ao usar o Título e a Descrição de um relatório de bug, 79.51 % $\pm$ 7.96 ao usar o Título e 78.04 % $\pm$ 8.2 ao usar a Descrição.

Palavras-Chave

Classificação de vulnerabilidades, Problemas de Segurança, Classificação de Bug de Segurança, Relatório de Bug de Seguranças, Algoritmos de Machine Learning, Bag of Words.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

AUC Area Under The Curve. iii, v, 19, 29, 36, 40

CVE Common Vulnerabilities and Exposures. 12, 13

KNN K-Nearest Neighbors. xi, 27, 32–34, 37, 38, 41

MRMR Minimum Redundancy Maximum Relevance. 25, 26, 37

NVD National Vulnerability Database. 12, 13

RFE Recursive Feature Elimination. 32

ROC Receiver Operating Characteristics. xi, 29, 30

SVM Suport Vector Machine. xi, 26, 28, 32–34, 37–39, 42

TF Term Frequency. 6, 24

TF-IDF Term Frequency–Inverse Document Frequency. 6, 22, 24, 25

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Software verification and validation are a set of important activities that are part of software projects, as they help build reliable software. These activities help keep track of bugs by analysis of reports. The reports containing information such as the type of defect, impact, and the steps to reproduce the bug are added to a given tracking system. The whole process can become very inefficient if bugs are not properly detected and classified [4], which is in itself a time-consuming task. This is especially true when dealing with security vulnerabilities, with the addition that if they aren't properly classified, they could have a greater impact on the system [5, 6].

The information present in a bug report is unstructured, because of this analyzing the text to understand its class is hard and time-consuming. This is where creating a text classification with machine learning becomes important, using text classifiers, we can gather the information available and structure it to allow a text classifier model to predict a document's class cost-effectively.

For that, many studies in the past decade were made to create a framework capable of automatically categorizing Bug Reports [7, 8], to reduce the amount of human effort, the time needed for the classification, and the number of bugs wrongfully categorized. And we also set our goal on creating a classifier, based on different techniques.

In this work, we take a focus on security problems. The concept we are taking for security is the set of controls that are put in place to provide confidentiality, integrity, and availability for all components of computer systems. These components include data, software, hardware, and firmware.

Our goal is to present an analysis regarding the different approaches in the literature of the past decade for classifying Bug Reports and experimental analysis of the different approaches when creating a model capable of classifying bug reports, with a focus on the classification of security bug reports. Particularly, we discuss techniques used for classification, preprocessing of the data-sets, the characteristics of the data-sets, how the set of features are handled, and the metrics for the evaluation of the models. With all that, we begging to experiment with the different combinations when making a model capable of classifying a bug report. As such, in this thesis, we implemented a model capable of executing the different combinations automatically and saved the results obtained so we could analyze and compare the impact each method from each different step has on the results.

This thesis contributes with:

- An analysis, in the form of a review, of the different approaches done in the past decade, and used in the other works for classifying security vulnerabilities;
- An analysis of the impact of different aspects of a text classification model on the performance.

This thesis is organized as follows. Chapter 2 provides some background on concepts regarding a machine learning algorithm and bug reports. In Chapter 3, we describe how the research for this thesis was performed and we discuss the answers to the research questions and identify gaps in the state of the art. Chapter 4 it is discussed how the experiments were made and how the different methods work. Chapter 5 presents the results obtained and the findings we had when analyzing the results. Chapter 6 concludes this thesis and presents the main threats to the validity of this work.

Chapter 2

Basic Concepts

In this section, we provide background concepts regarding bug reports and text classification with a machine learning algorithm.

2.1 Bug Reports

A Bug is a flaw or an error in an application that is affecting the normal flow of an application by mismatching expected behavior. Bug occurs when a mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester.

A bug report is a document that has information about what is wrong and needs to be fixed in software. The report lists what was viewed as wrong. But these documents are usually unstructured, difficulting their analysis by automatic systems.

A bug report can have several types, but for this thesis, we defined a binary classification problem defined by the classes:

- Security Bug: is a software bug that can be exploited to gain unauthorized access or privileges on a computer system, for example, gaining authentication of users and other entities, confidential data, getting authorization to of access rights and privileges, etc;
- Non-Security Bug: would be bugs that do not affect security, simple examples would be wrong visuals effects, or the link takes to wrong place.

2.1.1 Bug Life Cycle

The bug has different states in the Life Cycle. The Life cycle of the bug can be shown diagrammatically in Figure 2.1 to help understand the workflow of a bug life cycle.

- **New:** This is the first state of a bug in the bug life cycle. When any new bug is found, it falls in a *New* state, and validations and testing are performed on this bug in the later stages of the bug life cycle.
- **Assigned:** After the tester has posted the bug, the lead of the tester approves that the bug is genuine and he assigns the bug to a corresponding developer and the

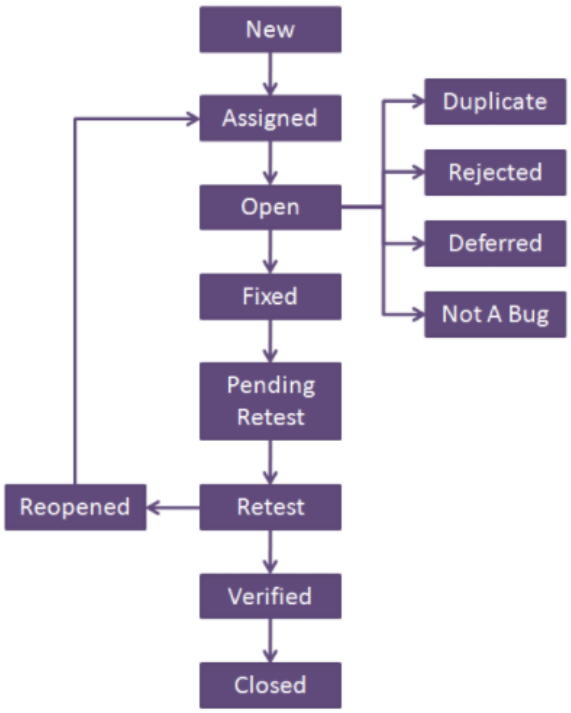

Figure 2.1: Bug life cycle. (Image taken from [1])

developer team and it is given the state *Assigned*.

- **Open:** At this state the developer has started analyzing and working on the bug fix.
  - **Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to *Duplicate*.
  - **Rejected:** If the developer feels that the bug is not an actual defect, he rejects the bug. Then the state of the bug is changed to *Rejected*.
  - **Deferred:** The bug, changed to *Deferred* state means the bug is expected to be fixed in the next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have a major effect on the software.
  - **Not a bug:** The state given as *Not a bug* if there is no change in the functionality of the application. For example: If a customer asks for some change in the look and feel of the application like the change of color of some text then it is not a bug but just some change in the look of the application.
- **Fixed:** When the developer makes necessary code changes and verifies the changes then it is given to the bug the status of *Fixed* and the bug is passed to the testing team.
- **Pending retest:** After fixing the bug the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is *Pending retest*.
- **Retest:** At this stage the tester does the retesting of the changed code which developer has given to him to check whether the bug got fixed or not.

- **Verified:** The tester tests the bug again after it got fixed by the developer. If the bug is not present in the software, he approves that the bug is fixed and changes the status to *Verified*.
- **Reopen:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to *Reopened*. The bug goes through the life cycle once again.
- **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to *Closed*. This state means that the bug is fixed, tested, and approved.

As observed in the steps above, a bug's life cycle is costly and time-consuming. The real problem comes when a security bug is incorrectly classified this makes it be longer time unsolved allowing individuals with ill intention to take advantage of them, that is why these bugs must be identified as soon as possible. For that reason, in recent decades, many studies have focused on the development of text classification approaches to automatize the identification of these bugs.

This life cycle of a bug begins when it is added to a tracking system, this system serves as a database that keeps track of reported software bugs in software development projects. Many bug tracking systems, most commonly those used by most open-source software projects, allow end-users to directly make their reports, the most commonly known would be Jira and Bugzilla.

2.2 Text Classification

In machine learning, a text classification is an approach that assigns to a text a set of predetermined categories. These models or test classifiers can be used to organize and categorize any kind of text or documents, like:

- Categorization of news articles into defined topics;
- Detection of spam and non-spam emails;
- Detection of urgency in Customer Support;
- Detection of fraud and online abuse.

The greater part of all available text information is composed of unstructured text. Because of this unstructured nature manually analyzing, understanding, organizing, and sorting through text data is hard, time-consuming, and much less accurate, leading to several errors in its interpretation.

This is where text classification with machine learning comes in. With the use of text classifiers, one can automatically find all manner of relevant patterns in text, from emails, legal documents, surveys, and more in a fast and cost-effective way, allowing to save time and automating the interpretation and classification of the texts.

Machine learning algorithms present advantages when compared to human text classification:

- **Scalability:** Machine learning can automatically analyze documents at a fraction of the cost, often in just a few minutes when compared to doing it manually, and text classification tools are scalable to any needs, large or small.

- **Real-time analysis:** There are critical situations that it is needed to identify as soon as possible and take immediate action, machine learning text classification can analyze documents in real time, so it enables the identification of critical information and allows to take action right away.
- **Consistent criteria:** Human triaggers make mistakes when classifying text data due to distractions, fatigue, boredom, and human subjectivity creates inconsistent criteria, machine learning, on the other hand, applies the same lens and criteria to all data and results.

Instead of relying on biased human behavior, machine learning learns to make classifications based on observations of previously used texts by using pre-labeled examples as training data. Machine learning algorithms learn the different associations between pieces of text and learn to relate a particular output with a particular input. The collection of these pre-labeled examples is also commonly known as a dataset.

The first step towards training a machine learning classifier is feature extraction, this method is used to transform each text into a numerical representation in the form of a vector. One of the most frequently used approaches is bag-of-words, where a vector represents the frequency of a word in a predefined dictionary of words, an example can be observed in Figure 2.2. Other forms include Term Frequency (TF) demonstrate the importance of the word to a document with an intuition that the more the term appears in a document means higher the importance is. And there is also Term Frequency–Inverse Document Frequency (TF-IDF) in which the importance of a word depends on the frequency it appears in a document, but it is offset by the number of documents that contain the word to adjust for the fact that some words appear more frequently in general.

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

Figure 2.2: Example of Bag of Words.(Image taken from [2])

But this approach causes a significant dimensionality problem, for the more documents the larger is the vocabulary, making the feature matrix be a huge sparse matrix, for that, the Bag-of-Words model is preceded by a preprocessing step (word cleaning, stop words removal, stemming/lemmatization) aimed to reduce the dimensionality problem. Another way that to reduce the dimensionality, and is usually used in conjunction with the preprocessing step, is Feature Selection. Feature Selection is a step that is carried after we already have a feature matrix this is done to drop some columns and reduce the matrix dimensionality by selecting a subset of the most statistically relevant feature variables (i.e., words of the vocabulary).

Then, the machine learning algorithm is fed with training data that is a matrix that consists of pairs of feature sets (vectors for each text example) and labels (e.g. security, non-security) to represent the class of the text and to produce a classification model. Once

trained with enough samples, the model can be used to make automatic classifications. The same feature extractor is used to transform the unseen text to feature sets, which then can be fed into the classification model to get predictions on labels (Figure 2.3 for reference).

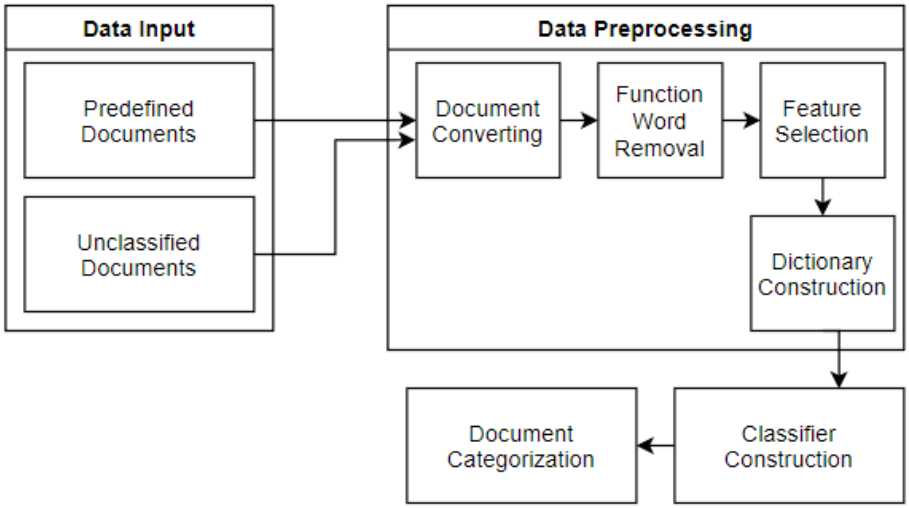

Figure 2.3: Example of usual Classification Model.

Chapter 3

State of the Art

In this chapter, we describe the studies that will help us with the analyzes of the state-of-the-art on software Security Vulnerability classification.

3.1 Review Methodology

Well-established guidelines for conducting this review [[9], [10]] were followed , and comprised the following steps:

- **Analysis of related surveys**: Identify and analyze secondary studies (i.e., surveys or other reviews) that cover the topic of vulnerability report classification, even if partially. The related surveys are analyzed so that gaps and limitations are identified and covered in our discussion.
- **Definition of research questions**: After the gaps are identified, we define the research questions that our thesis will focus on.
- **Identification of studies**: Identify relevant primary studies, that contribute to our analyzes. We then identify a query to be used to perform the search and then the snowballing process [11] is used to complement the identification of studies.
- **Study selection**: With this step, we refer to the application of criteria for the inclusion and exclusion of the primary studies found, as well as for quality assessment. The objective is to select the reviews that meet the goal and scope of our thesis.
- **Data extraction and synthesis**: We extract the important data, according to the research questions, from the primary studies selected. The information gathered is then synthesized to help us answer the research questions.

Each of these steps is described in further detail in the following sections.

3.1.1 Analysis of related surveys

In the following paragraphs, we present the related secondary studies (e.g., surveys, other reviews):

- S. M. Ghaffarian and H. R. Shahriari, *Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey*, ACM Comput. Surv., vol. 50, no. 4, p. 56:1-56:36, Aug. 2017, doi: 10.1145/3092566.
- D.-G. Lee and Y.-S. Seo, *Systematic Review of Bug Report Processing Techniques to Improve Software Management Performance*, Journal of Information Processing Systems, vol. 15, no. 4, pp. 967–985, Aug. 2019, doi: 10.3745/JIPS.04.0130.
- J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei, *A survey on bug-report analysis*, Sci. China Inf. Sci., vol. 58, no. 2, pp. 021101–021101, Jan. 2015, doi: 10.1007/s11432-014-5241-2.

S. M. Ghaffarian and H. R. Shahriari [12] provide an extensive review of many different works in the field of software vulnerability analysis and discovery, using machine learning and data mining techniques. They reviewed different types of works in the field, discussed the advantages and disadvantages, and pointed out the challenges and some unknown areas in such a field. At the end of the survey, they concluded that the approaches they studied were still in an immature state of research.

The study conducted by Dong-Gun Lee and Yeong-Seok Seo [13] focuses on investigations to improve the accuracy of existing techniques for bug report processing. They study the bug report processing technology based on the following three factors: the techniques used, the experimental goal, and the comparison with the two fields (duplicate bug report identification and classification accuracy improvement) to improve software maintenance performance. In this study, the authors found problems with the technique of identifying similarities between bug reports.

Zhang et al. [14] present a detailed survey of the existing work of bug report analysis, such as bug report optimization, triaging, and bug repair. In the end, the author concludes that many problems are still unresolved or even researched, and many researchers are committed to the automation of handling bug reports (e.g., bug-report assignment, duplicate detection, bug localization), but none achieved a satisfactory accuracy (e.g., more than 95%). That is, it is difficult to apply existing automatic methods in practice. Furthermore, the bug tracking system still can not provide any support for automatically processing bug reports.

3.1.2 Research questions

Based on the analysis of existing secondary studies, we defined the following set of five research questions that guided the guided objectives defined for this thesis:

- **RQ-1**: What are the objective of the works?
- **RQ-2**: What are the characteristics of the data-sets used?
- **RQ-3**: What are the most used Machine Learning approaches?
- **RQ-4**: How are the data-sets partitioned?
- **RQ-5**: What are the metrics used for the evaluation of the approach?

With the research questions, we intend to analyze the different types of approaches identified in the literature, by breaking down each approach into four parts.

3.1.3 Identification of studies

Six well-known online libraries were used to search for primary studies, which are common presence in related studies. The data sources are: *Google Scholar* [15], *DBLP* [16], *IEEE Xplore* [17], *ACM Digital Library* [18], *Scopus* [19], *Springer Link*, [20]. To perform the search, the following query string was used, built based on preliminary observations using the respective search engines of the online libraries:

("vulnerability" OR "security defect" OR "security bug" OR "security issue") AND ("bug report" OR "issue report" OR "defect report" OR "vulnerability report") AND (classification OR detection OR identification)

As a complement to the online search and to enrich the set of selected studies, we also used snowballing [11].

3.1.4 Study selection and quality assessment

We filtered the identified studies mostly by applying a set of inclusion and exclusion criteria, which also reflects a conservative quality assessment criterion of keeping only peer-reviewed (or otherwise cited) studies, to avoid erroneously excluding papers from the analyzes. The criteria are as follows:

- Inclusion Criteria:
  - A study whose focus is set on the automatic classification of security vulnerability reports.
  - The study should be sufficiently complete to allow answering the identified set of research questions.
- Exclusion Criteria:
  - A study whose focus is not set on automatic classification of security vulnerability reports.
  - Non-peer reviewed, or otherwise uncited, research should be excluded from the analyzes.

To apply these criteria, we went through three pruning phases, each of which respectively targeted the analysis of the title, abstract, and the full text.

3.1.5 Data extraction and synthesis

This step consists of analyzing all primary studies and gathering information regarding their approach. As the authors use very diverse terms, this was carried out iteratively, starting first with the identification of the specific terms used by the authors, that allow responding to the research questions, which were then individually discussed and iteratively generalized to more widely accepted consensual terms. This process involved discussion and agreement between and the advisers.

3.1.6 Outcome of the study identification and selection

Table 3.1 sets the outcome of the selection and identification process in perspective with what is analyzed in related surveys. On the left-hand side, the table identifies the related surveys, the respective periods analyzed in each survey, the total number of papers analyzed, and how many of those relate to security bugs. Since having the idea of focusing on classifying bug reports between security or non-security, the surveys did not show any results, only one of them as we can see presented only two studies relate to our thesis.

Survey	Period	Total Papers	Security bugs
S. M. Ghaffarian and H. R. Shahriari	2001 - 2016	39	2
Dong-Gun Lee and Yeong-Seok Seo	2006 - 2018	29	-
Zhang et al.	2003 - 2014	107	-
This work	2009 - 2021	42	42

Table 3.1: Number of studies analyzed in the related surveys

3.2 Analysis

In this section, are discussed the main findings identified during the analysis of the state-of-the-art, in perspective with the research questions. After answering the research questions it is presented a quick summary of our finds. We then highlight the gaps present in the state-of-the-art.

3.2.1 Research questions discussion

We begin by discussing the research question **RQ-1 What are the objective of the works?**. There are two types of studies that we have identified, those that focus on identifying security bugs through its description, just like our own focus, and those that want to classify already identified security bugs.

Objective	Total
Identify	29
Classify	15

Table 3.2: Counts of studies and each objective.

The difference between these two approaches is within their objective the studies that desire to identify bugs have the objective to see if a bug description relates to a security issue by using information given by the one that identified it like the title and description of the bug. The other objective that this thesis does not consider is classifying the security bug by its type or severity using pieces of information to calculate the severity of a bug or by the description and comments to give a type of security error that could be.

We continue discussing the research question **RQ-2 What are the characteristics of the data-sets used?** for which, we found a plethora of sources for the data-sets. When grouping the datasets used in the different approaches, the origin of most bug reports datasets was identified as shown in Table 3.3. Figure 3.1 depicts a better visualization of the distribution of the number of studies that used each tracking system.

Dataset	NVD	Shuai et al.[4], G. Spanos and L. Angelis [21], O. Jormakka [22], Huang et al. [23], S. E. Sahin and A. Tosun [24], Le et al. [25], Kudjo et al. [26], A. Duppils and M. Tullberg [27], T. M. Adhikari and Y. Wu [28], Spanos et al. [29], Chen et al. [30]
	CVE	M. M. Rahman and S. Yeasmin [31], Wijayasekara et al. [32], S. Mostafa and X. Wang [6], Mostafa et al. [7], Kudjo et al. [26], S. S. Alqahtani [33], Palacio et al. [34], Wijayasekara [5], Chen et al. [30]
	Monorail	O. Jormakka [22], Shu et al. [35], Peters et al. [36], Shu et al. [37], Wu et al. [38], Jiang et al. [8], Aljedaani et al. [39], Wu et al. [40]
	Bugzilla	Wijayasekara [5], Behl et al. [41], Wijayasekara et al. [32],S. Mostafa and X. Wang [6], M. Davari [42],Y. Zhou and A. Sharma [43], Zou et al. [44], Mostafa et al. [7], Catolino et al. [45], Z. S. M. Alharthi and R. Rastogi [46], Bhuiyan et al. [47], Bulut et al. [48]
	GitHub's Issue tracking	Y. Zhou and A. Sharma [43], L. Wan [49], Palacio et al. [34], A. Duppils and M. Tullberg [27]
	Jira	Y. Zhou and A. Sharma [43], D. C. Das and Md. R. Rahman [50], Pandey et al. [51], Catolino et al. [45], O. Jormakka [22], Shu et al. [35], Peters et al. [36], Xia et al. [37], Jiang et al. [8], Wu et al. [38], Pandey et al. [51], Wu et al. [40]
	Launchpad	Wu et al. [38], Wu et al. [40]
	Miscellaneous	Zhang et al. [52]

Table 3.3: Data-set studies distribution.

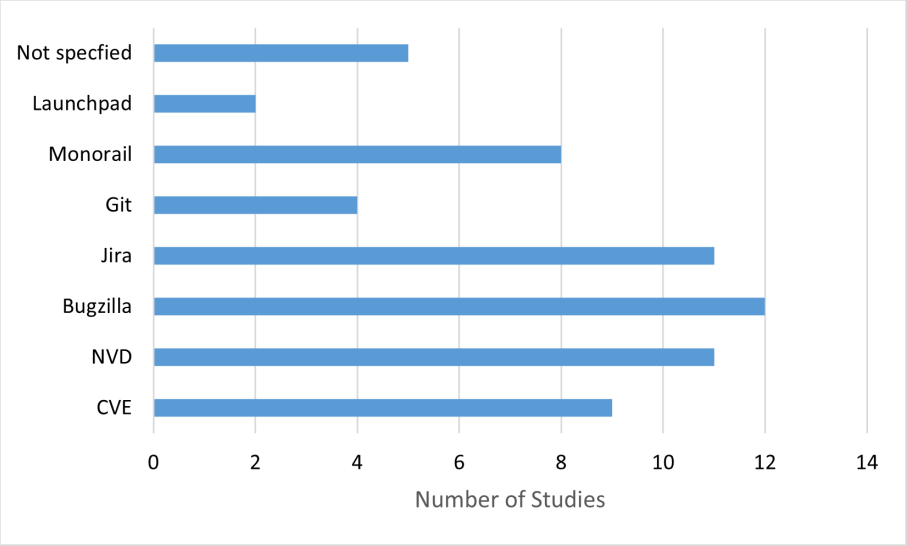

Figure 3.1: Distribution of data-set origin.

As we can see in Figure 3.1 that represents the distribution of the Table 3.3, most of the bug reports were removed from two specific tracking systems. Jira contains the reports from Apache projects. And Bugzilla where it contains the bug reports from their other applications (e.g., Mozilla, Firefox, Thunderbird). Because of having bug reports of so big applications, one can easily find a big quantity of Bug Reports separated per each of their respective projects. Following them some works decide to collect information from tracking systems with a focus on security description (i.e., National Vulnerability Database (NVD) and Common Vulnerabilities and Exposures (CVE)) they, different from the Bugzilla and Jira, don't have bug reports, instead, they have the description of several most common types of vulnerabilities. And within the analysis we noticed that 1/2 of the works that use either CVE or NVD would make their data-set with information from bug reports removed from tracking systems and the description of the security vulnerabilities to create a more balanced data-set, the other half was experimenting with training a classifier using only the descriptions available in NVD or CVE.

About the dataset, from each tracking system, with exception of the NVD and CVE, the

Systems	Operation System	Linux	Wijayasekara et al. [32], L. Wan [49], Wijayasekara [5], Chen et al. [30]
		Enterprise Linux	Chen et al. [30]
		Mac OS	Kudjo et al. [26]
		Chrome OS	Kudjo et al. [26]
		Win 10	Kudjo et al. [26], Chen et al. [30]
		Win 7	Kudjo et al. [26]
		Cisco Software	Gegick et al. [53]
	Browser	Firefox	M. Davari [42], Kudjo et al. [26], Z. S. M. Alharthi and R. Rastogi [46], Zou et al. [44]
		Chrome	Kudjo et al. [26], Chen et al. [30]
		Edge	Kudjo et al. [26]
		Safari	Chen et al. [30]
		Explorer	Kudjo et al. [26]
		Chromium	O. Jormakka [22], Shu et al. [35], Peters et al. [36], Shu et al. [37], Wu et al. [38], Jiang et al. [8], Aljedaani et al. [39], Wu et al. [40]
		Seamonkey	Zou et al. [44], M. Davari [42], Kudjo et al. [26]
	Data Management	MySQL	Wijayasekara [5]
		Derby	D. C. Das and Md. R. Rahman [50], O. Jormakka [22], Shu et al. [35], Peters et al. [36], Shu et al. [37], Wu et al. [38], Jiang et al. [8], Wu et al. [40]
		Hbase	S. S. Alqahtani [33]
		Hive	S. S. Alqahtani [33]
		Jackrabbit	Pandey et al. [51]
		Sling	S. S. Alqahtani [33]
		Spark	S. S. Alqahtani [33]
		Hadoop	S. S. Alqahtani [33]
		Lucene	Pandey et al. [51]
	Message Oriented Middleware	Camel	D. C. Das and Md. R. Rahman [50], O. Jormakka [22], Shu et al. [35], Peters et al. [36], Shu et al. [37], Jiang et al. [8], Wu et al. [40]
	Open Source Solutions	RedHat	Mostafa et al. [7]
		Mozilla	J. P. Tyo [54], Mostafa et al. [7], Catolino et al. [45],Z. S. M. Alharthi and R. Rastogi [46], Bhuiyan et al. [47]
		Core (Mozilla)	M. Davari [42]
		Apache	S. Mostafa and X. Wang [6], Xia et al. [37], Catolino et al. [45]
	File Editor/Reader	Microsoft Office	Chen et al. [30]
		Foxit	Chen et al. [30]
		PDFbox	S. S. Alqahtani [33]
		Tika	S. S. Alqahtani [33]
		Ffmpeg	L. Wan [49]
		QuickTime	Chen et al. [30]
		Adobe Flash Player	Chen et al. [30]
	Network	HTTPClient	Pandey et al. [51]
		Wireshark	L. Wan [49], Bulut et al. [48]
	E-mail	Thunderbird	M. Davari [42], Zou et al. [44]
	Software Development	Eclipse	Catolino et al. [45], Z. S. M. Alharthi and R. Rastogi [46]
		Openstak	Wu et al. [38], Wu et al. [40]
		Wicket	D. C. Das and Md. R. Rahman [50], O. Jormakka [22], Shu et al. [35], Peters et al. [36], Shu et al. [37], Jiang et al. [8], Wu et al. [40]
		Qemu	L. Wan [49]
	Management	Ambari	D. C. Das and Md. R. Rahman [50], O. Jormakka [22], Shu et al. [35], Peters et al. [36], Shu et al. [37], Jiang et al. [8], Wu et al. [40]
		Oracle Business Suite	Chen et al. [30]
		Stanbols	S. S. Alqahtani [33]
	Framework	Felix	S. S. Alqahtani [33]
		Karaf	S. S. Alqahtani [33]
	No Group	Ground/Flight Missions	J. P. Tyo [54], J. Tyo [55]
		CFX	S. S. Alqahtani [33]
		Miscellaneous	M. M. Rahman and S. Yeasmin [31], Palacio et al. [34], Y. Zhou and A. Sharma [43], Zhang et al. [52]
		Not Specified	Shuai et al.[4], G. Spanos and L. Angelis [21], Huang et al. [23], S. E. Sahin and A. Tosun [24], Le et al. [25], A. Duppils and M. Tullberg [27], T. M. Adhikari and Y. Wu [28], Behl et al. [41], Spanos et al. [29]

Table 3.4: Systems Used.

bugs were recovered from specific applications. The ones that were retrieved from Jira are Apache projects, namely: Lucene, Jackrabbit (JCR), HTTPClient, Wicket, Ambari, Camel, Derby. Bugzilla since it is a tracking system from the Mozilla Foundation, like Firefox or Thunderbird. From them, it was usually taken the entirety of bug reports. Differently, NVD and CVE, they contain the description of specific vulnerabilities, each containing an identification number, a description, and at least one public reference. Since they have a great number of entries, around 140000, the studies contained with getting the most recent entries, at the time.

As discussed above most studies preferred to take bug reports of real projects with its real proportions between Non-Security Bug Report and Security Bug Report and some only taking Security Bug Report, both of these approaches could be problematic faced with a situation where the classifier can't make a decisive decision in what to classify a report,

making the classifier bias and it will end up classifying the report concerning its bigger class, so a more correct approach to this would be taking a mixed data-set intending to create a more balanced data to prevent class bias to guarantee a more accurate classification.

Data-set Size	[1.000, 10.000]	Behl et al. [41], J. P. Tyo [54], J. Tyo [55], Catolino et al. [45], Pandey et al. [51], Wijayasekara et al. [32]
	]10.000, 20.000]	Kudjo et al. [26], O. Jormakka [22], Aljedaani et al. [39]
	]20.000, 30.000]	Zou et al. [44], D. C. Das and Md. R. Rahman [50], Chen et al. [30], Rahman and S. Yeasmin [31]
	]30.000, 40.000]	Zhang et al. [52], Y. Zhou and A. Sharma [43]
	]40.000, 50.000]	T. M. Adhikari and Y. Wu [28],Shu et al. [35], Peters et al. [36], Shu et al. [37], Wu et al. [38]
	]50.000, 60.000]	Shuai et al.[4], Huang et al. [23]
	]60.000, 70.000]	Mostafa et al. [7], L. Wan [49]
	]70.000, 80.000]	S. E. Sahin and A. Tosun [24], Spanos et al. [29]
	]80.000, 90.000]	G. Spanos and L. Angelis [21]
	]90.000, 1.000.000[	Wijayasekara [5], M. Davari [42], Le et al. [25], Pereira et al. [56], S. S. Alqahtani [33], Palacio et al. [34], A. Duppils and M. Tullberg [27], Jiang et al. [8], Bhuiyan et al. [47], Wu et al. [40]

Table 3.5: Dataset size studies distribution.

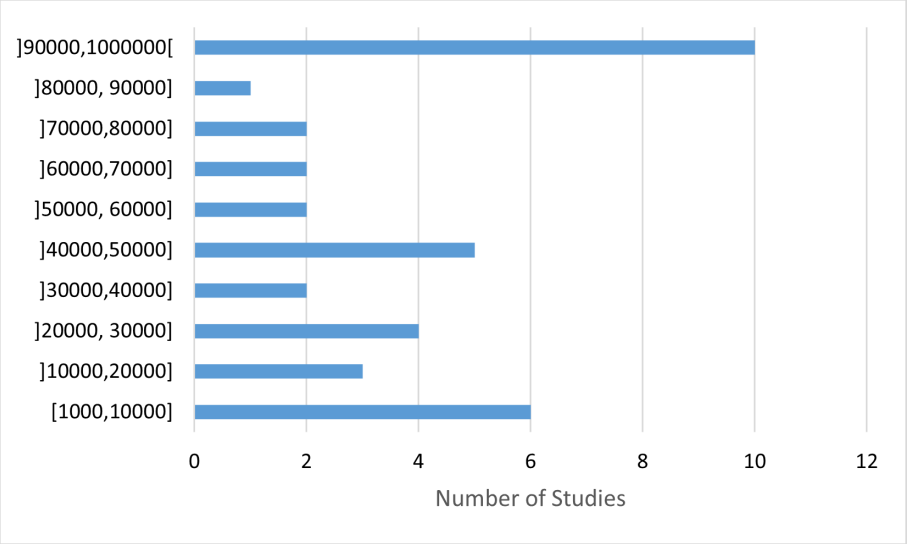

Figure 3.2: Distribution of data-set sizes.

As we can see in the Figure 3.2 that represents the distribution of the Table 3.5, the preferences to the size of the data-set are quite the opposite to each other, it either goes to an interval between [1000, 10000] or an interval between]90000, $+\infty$[with a few others going in the middle]40000, 50000] with the median in located in this interval with around 45000 bug reports. Most of those in the]90000, $+\infty$[interval do not go too far from the 100000 marks. Only one of them [56] goes way beyond that mark, by having the high value of Bug Reports, more than 1000000, but because in this study the proposed method was to create a classifier that could classify the report using only the Title and noise data, in there case, a Bug Report is considered noise when parts of it would not be available to not reveal sensitive information (i.e., password, personally identifying information, ...).

This characteristic of the data-set is quite experimental since there a no decisive information that says what is the perfect good size for training data, but what we can understand is that more than half the works prefer the size of its training data to be smaller than 50000.

In this part, related to the third question, **RQ-3 What are the most used Machine Learning approaches?** In this section, we will analyze the methods used in the different

stages of the approaches made, first, we will discuss the pre-processing techniques, following it we have Feature Extraction, then Feature Reduction, and at last the Classifiers. At first, we have the techniques used for pre-processing. The choice of techniques used does not differ much from study to study. The part of pre-processing always begins with the tokenization of the report description followed by these techniques.

Pre-Processing	Stop Words Removal	Gegick et al. [53], Wijayasekara [5], Shuai et al.[4], Behl et al. [41], Wijayasekara et al. [32], J. P. Tyo [54], Spanos et al. [29], Pandey et al. [51], Zou et al. [44], D. C. Das and Md. R. Rahman [50], G. Spanos and L. Angelis [21], O. Jormakka [22], S. E. Sahin and A. Tosun [24], Le et al. [25], Pereira et al. [56], Peters et al. [36], Catolino et al. [45], Kudjo et al. [26], S. S. Alqahtani [33], Zhang et al. [52], Wu et al. [38], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Z. S. M. Alharthi and R. Rastogi [46], Aljedaani et al. [39]
	Lemmatization	O. Jormakka [22], Huang et al. [23], T. M. Adhikari and Y. Wu [28], Aljedaani et al. [39]
	Stemming	D. C. Das and Md. R. Rahman [50], J. Tyo [55], G. Spanos and L. Angelis [21], Mostafa et al. [7], Le et al. [25], Catolino et al. [45], Kudjo et al. [26], Palacio et al. [34], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Chen et al. [30], J. P. Tyo [54], Wijayasekara et al. [32], Spanos et al. [29], Huang et al. [23]
	Not Specified	M. Davari [42], Y. Zhou and A. Sharma [43], L. Wan [49], Shu et al. [35], Bulut et al. [48], Shu et al. [37], Bhuiyan et al. [47], Wu et al. [40], S. Mostafa and X. Wang [6], M. M. Rahman and S. Yeasmin [31], A. Duppils and M. Tullberg [27]

Table 3.6: Pre-processing studies distribution.

Stemming or Lemmatization, the idea behind this is that since they tokenized the text in the bug reports, using stemming they would find all the synonymous and use one word to represent all of them, instead of having several representing the same thing. The other method, lemmatization, is like the one mentioned before, but instead of identifying the synonymous it puts the words in the base form and removes the duplicates.

Removal of *stop words*, which consists of removing commonly used words (such as "the", "a", "an", "in") most of them did this because they would not want these words to take up valuable processing time since these words would not bring anything of value to help in the identification of the class in which the bug report should be in.

We are discussing these techniques even though they are common in pre-processing because upon discussing with the supervisors, it was informed to them that most works were referring to these techniques to be the ones for Feature Selection and Dimension reduction, we further discussions with it became clear that this is a bad approach for these problems and this was then identified as gaps in the works.

This second part of the question aims at learning the different types of algorithms of feature extraction. Figure 3.3 shows the different techniques used in the studies.

As we can see in Table 3.7 represented by Figure 3.3, TF-IDF is the preferred algorithm between those available. TF-IDF is most used because it does not look only at how many times a term appears in a document it also calculates the importance the mentioned term has in the document. Following the TF-IDF we have TF and N-Gram. TF is a simpler version that only calculates the frequency a term appears locally in the document, this method is more useful when classifying a single document without other references. And the N-Gram is a technique more used to calculate the probability of a word appearing in the context taking into account the N previous words, this is not much what we want, an approach that gives more information of word importance is preferred.

Considering an approach where we have a balanced data-set between Security Bug Report and Non-Security Bug Report using TF-IDF could be the best option comparing to the other simply for seeing word importance on the overall set of reports used instead of looking locally on the report description, but for that, it needs a well-balanced data between

Feature Extraction	TF - IDF	Wijayasekara [5], Shuai et al.[4], Behl et al. [41], S. Mostafa and X. Wang [6], J. P. Tyo [54], D. C. Das and Md. R. Rahman [50], J. Tyo [55], Huang et al. [23], Mostafa et al. [7], S. E. Sahin and A. Tosun [24], Peters et al. [36], Le et al. [25], Pereira et al. [56], Catolino et al. [45], Bulut et al.[48], Zhang et al. [52], A. Duppils and M. Tullberg [27], T. M. Adhikari and Y. Wu [28],Jiang et al. [8], Chen et al. [30]
	TF - IGM	Kudjo et al. [26], Chen et al. [30]
	TF	Gegick et al. [53], Wijayasekara et al. [32], J. P. Tyo [54], Spanos et al. [29], Pandey et al. [51], J. Tyo [55], G. Spanos and L. Angelis [21], Le et al. [25], Zhang et al. [52], S. E. Sahin and A. Tosun [24]
	N-Gram	O. Jormakka [22], Zhang et al. [52], Z. S. M. Alharthi and R. Rastogi [46],Y. Zhou and A. Sharma [43], Zou et al. [44], L. Wan [49], Palacio et al. [34], Bhuiyan et al. [47], S. E. Sahin and A. Tosun [24]
	Manually	M. M. Rahman and S. Yeasmin [31]
	Not Specified	M. Davari [42], O. Jormakka [22],Shu et al. [35], S. S. Alqahtani [33], Wu et al. [38], Z. S. M. Alharthi and R. Rastogi [46], Shu et al. [37], Wu et al. [40]

Table 3.7: Feature Extraction studies distribution.

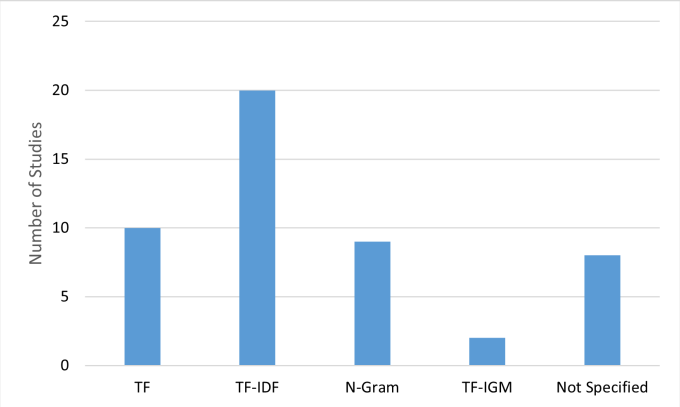

Figure 3.3: Distribution of Feature Extraction used.

Security Bug Report and Non-Security Bug Report, if we were to have more cases of Non-Security Bug Report this could lead to problems where terms of importance in a security context could be considered unimportant leading to that term have a lower value of importance. So when using the TF-IDF to extract features of labeled data sets it seems like a good idea to have a well-balanced data set.

In this following part of the question, what we could gather by analyzing the studies is that most of the works do not consider using these methods, as mentioned before, and consider the prepossessing techniques to be enough for them. But not all of them went in that direction few of them did use specific techniques for the Feature Reduction instead of keeping the results of simple techniques such as steaming and stop words removal.

Feature Selection	Information Gain	Huang et al. [23], Chen et al. [30]
	Log Odd Ratio	D. C. Das and Md. R. Rahman [50]
	Not Specified	Gegick et al. [53] , Wijayasekara [5], M. M. Rahman and S. Yeasmin [31], Shuai et al.[4], Behl et al. [41], Wijayasekara et al. [32], S. Mostafa and X. Wang [6], J. P. Tyo [54], M. Davari [42], Spanos et al. [29],Y. Zhou and A. Sharma [43], Pandey et al. [51], Zou et al. [44], J. Tyo [55], G. Spanos and L. Angelis [21], O. Jormakka [22], L. Wan [49],Mostafa et al. [7], S. E. Sahin and A. Tosun [24], Le et al. [25],Shu et al. [35], Pereira et al. [56], Peters et al. [36], Catolino et al. [45], S. S. Alqahtani [33], Palacio et al. [34], Bulut et al. [48], Shu et al. [37], Zhang et al. [52], A. Duppils and M. Tullberg [27] , Wu et al. [38] , T. M. Adhikari and Y. Wu [28] , Jiang et al. [8] , Z. S. M. Alharthi and R. Rastogi [46] , Bhuiyan et al. [47], Kudjo et al. [26]

Table 3.8: Feature Selection studies distribution.

As mentioned before, when we referred to the prepossessing those were the only ones

Dimension Reduction	Singular Value Decomposition	Gegick et al. [53]
	Linear Discriminant Analysis (LDA)	Pandey et al. [51], Zhang et al. [52]
	Not Specified	Wijayasekara [5], M. M. Rahman and S. Yeasmin [31], Shuai et al.[4], Behl et al. [41], Wijayasekara et al. [32],S. Mostafa and X. Wang [6], J. P. Tyo [54], M. Davari [42], Spanos et al. [29],Y. Zhou and A. Sharma [43], Zou et al. [44], D. C. Das and Md. R. Rahman [50], J. Tyo [55], G. Spanos and L. Angelis [21], O. Jormakka [22], L. Wan [49], Huang et al. [23], Mostafa et al. [7], S. E. Sahin and A. Tosun [24], Le et al. [25],Shu et al. [35], Pereira et al. [56], Peters et al. [36], Catolino et al. [45], S. S. Alqahtani [33], Palacio et al. [34], Bulut et al. [48], H. Altunel, and A. Tosun, Shu et al. [37], A. Duppils and M. Tullberg [27] , Wu et al. [38] , T. M. Adhikari and Y. Wu [28] , Jiang et al. [8] , Z. S. M. Alharthi and R. Rastogi [46] ,Chen et al. [30] , Bhuiyan et al. [46], Kudjo et al. [26]

Table 3.9: Dimension Reduction studies distribution.

used to reduce the features, and very few studies made use of formal techniques. From each approach (i.e., Feature Selection, Dimension Reduction) the maximum number of references by technique were 3 as can be observed on Tables 3.8 and 3.9, those techniques were Information Gain and LDA.

It is a mistake not using techniques to properly reduce the features can deteriorate the performance of the tool, having a large feature set could make the model related to the training data and not generalize it well. If that happens the model would work too well on the training data-set, but it would fail on future unseen data and make its prediction unreliable.

At last, the techniques used for vulnerability report classification. We found some diversity in the techniques used with a focus on known techniques.

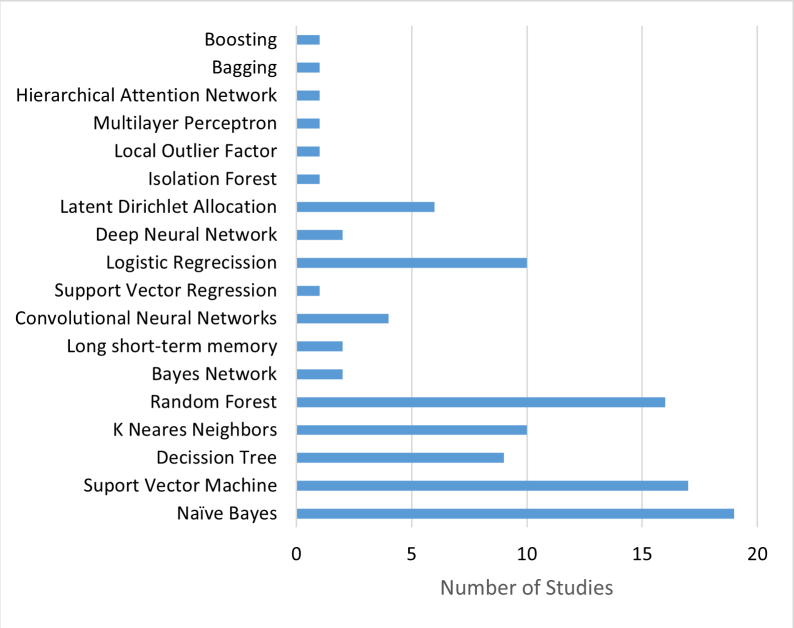

Figure 3.4: Distribution of Classification methods used.

Figure 3.4 shows that the studies tend to be a bit experimental, but still have a preference for well-known approaches, seeing that they are being the most used ones. The technique that took a lead was Naïve Bayes with slightly more articles preferring this method, followed by Support Vector Machine, and Random Forest. These techniques are of most common use for their efficiency on problems with high dimensionality such as text classi-

Classifier	Naive Bayes	M. M. Rahman and S. Yeasmin [31], Behl et al. [41], D. Wijayasekara [32], S. Mostafa and X. Wang [6], J. P. Tyo [54], M. Davari [42], Pandey et al. [51], D. C. Das and Md. R. Rahman [50], J. Tyo [55], Le et al. [25],Shu et al. [35], Pereira et al. [56], Peters et al. [36], T. M. Adhikari and Y. Wu [28], Z. S. M. Alharthi and R. Rastogi [46], Wu et al. [40], Wijayasekara [5], Catolino et al. [45], Shu et al. [37]
	Suport Vector Machine	S. Mostafa and X. Wang [6], J. P. Tyo [54], Spanos et al. [29],Y. Zhou and A. Sharma [43], Pandey et al. [51], Zou et al. [44], J. Tyo [55], Mostafa et al. [7], Le et al. [25], Bulut et al. [48], Zhang et al. [52], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Bhuiyan et al. [47], Shuai et al. [4], O. Jormakka [22], Catolino et al. [45]
	Decision Tree	Gegick et al. [53], Wijayasekara et al. [32], M. Davari [42], Spanos et al. [29], G. Spanos and L. Angelis [21], S. E. Sahin and A. Tosun [24], Kudjo et al. [26], Chen et al. [30], Zhang et al. [52]
	K-Nearest Neigbor	P. Tyo [54],Y. Zhou and A. Sharma [43], Pandey et al. [51], J. Tyo [55],Shu et al. [35], Peters et al. [36], Kudjo et al. [26], Jiang et al. [8], Chen et al. [30], Bhuiyan et al. [47]
	Random Forest	J. P. Tyo [54], M. Davari [42],Y. Zhou and A. Sharma [43], Pandey et al. [51], J. Tyo [55], G. Spanos and L. Angelis [21], Le et al. [25],Shu et al. [35], Peters et al. [36], Kudjo et al. [26], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Chen et al. [30], Bhuiyan et al. [47], Catolino et al. [45], Shu et al. [37]
	Bayes Network	M. M. Rahman and S. Yeasmin [31], J. P. Tyo [54]
	Long Short Term Memory	L. Wan [49], S. E. Sahin and A. Tosun [24]
	Convolutional Neural Networks	L. Wan [49], S. E. Sahin and A. Tosun [24], Palacio et al. [34], A. Duppils and M. Tullberg [27]
	Logistic Regression	Y. Zhou and A. Sharma [43], Pereira et al. [56], Peters et al. [36], Zhang et al. [52], Wu et al. [38], Jiang et al. [8], Bhuiyan et al. [47], Catolino et al. [45], M. Davari [42], Shu et al. [37]
	Support Vector Regression	Bulut et al. [48]
	Deep Neural Network	Huang et al. [23], Bhuiyan et al. [47]
	Latent Dirichlet Allocation (LDA)	Shuai et al. [4], Pandey et al. [51], S. S. Alqahtani [33], Zhang et al. [52], Aljedaani et al. [39], Catolino et al. [45]
	Isolation Forest	O. Jormakka [22]
	Local Outlier Factor	O. Jormakka [22]
	Multilayer Perceptron	Shu et al. [37]
	Hierarchical Attention Network	A. Duppils and M. Tullberg [27]
	Bagging	Zhang et al. [52]
	Boosting	Zhang et al. [52]

Table 3.10: Classifier studies distribution.

fication. We were also able to identify that, for the last two years, that Neural Network approaches are being implemented.

In this section, we will discuss the fourth question **RQ-4 How are the data-sets partitioned?** and see how the data-set was partitioned.

As we can identify in Table 3.12 it is mostly used K-fold, since it is one of the most common techniques used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform. In practice, this is the case of the studies searched.

Makes sense for K-fold to be a more used method the way it implements the separation of training and test sets ensures that all folds are allowed to be used as a test set 1 time and used to train the model K-1 times, this sees if our model has properly generalized our data. In Hold-out we can not compare the results to models that have different training data we can not ensure that our model has the best performance for unseen data.

In this last research question **RQ-5 What are the metrics used for the evaluation of the approach?** aims at characterizing the type o metrics utilized by the studies to validate the results. Figure 3.5 shows a prevalence on four metrics (i.e., F-measure, Precision, Accuracy, Recall) with a few others (i.e., AUC, Matthews Correlation Coefficient, G-Score) used by some studies.

Partition	K-fold Cross Validation	Wijayasekara et al. [32], S. Mostafa and X. Wang [6], J. P. Tyo [54], M. Davari [42], Spanos et al. [29], Y. Zhou and A. Sharma [43], Pandey et al. [51], Zou et al. [44], D. C. Das and Md. R. Rahman [50], J. Tyo [55], G. Spanos and L. Angelis [21], L. Wan [49], Mostafa et al. [7], Le et al. [25], Shu et al. [35], Catolino et al. [45], Kudjo et al. [26], Zhang et al. [52], Jiang et al. [8], Z. S. M. Alharthi and R. Rastogi [46], Chen et al. [30], Bhuiyan et al. [47], Wu et al. [40], Shu et al. [37], O. Jormakka [22], Bulut et al. [48], T. M. Adhikari and Y. Wu [28]
	Hold-oud	Gegick et al. [53], J. P. Tyo [54], S. E. Sahin and A. Tosun [24], Pereira et al. [56], Palacio et al. [34], A. Duppils and M. Tullberg [27], Huang et al. [23], Shuai et al.[4]
	Not Specified	Wijayasekara [5], M. M. Rahman and S. Yeasmin [31], Behl et al. [41], S. S. Alqahtani [33], Wu et al. [38], Aljedaani et al. [39], Peters et al. [36]

Table 3.11: Data-set Partition studies distribution.

Algorithm	Total
K-fold	28
Hold-out	8
Not Specified	7

Table 3.12: Data-set Partition techniques used.

As we could gather from our analysis F-Measure, also called F-Score, is a model that is used for evaluation of binary classification, and it is the model with the highest use among the other methods, followed by Precision and Recall, which was not to our surprise since to calculate F-Measure it is needed to calculate both the Precision and Recall of the model in question. But since 2018 after the work of D. C. Das and Md. R. Rahman [50] and the increasing number of studies done, Area Under The Curve (AUC) has started to gain some attention since it is also a technique most commonly used to evaluate the performance of a binary classifier.

3.2.2 Summary and Gaps of Related work

As we could understand from answering our questions, is that some aspects of creating a Security Bug Report classification tool are experimental in some of its aspects, like what is the composition of the data-set as well the amount of reports needed with most studies preferring data-sets that come from applications with a size less than 50000, as which classifier is best for the approaches. Concerning the classifiers, where the three most used are known to have good performance with high dimensional data but recently has begun appear some studies with attempts to make unsupervised approaches. But other aspects are not that much experimental, for extraction of features TF-IDF is mostly chosen for its capacity to determine the importance of terms in the global of its data-set and not locally (inside a Bug Report), the metrics and partition as well, both have its set of most used techniques.

As identified and mentioned in the **RQ-3** there is a gap associated with the processing of the features, most did not make use of formal techniques, which may imply several challenges, namely the complexity or time to process or even the result and reliability of the classifier. This problem could come from the unnecessary large dimension of features, that may come with a badly generalized set of features making the model present itself unreliable when classifying future unseen reports, since having such a large dimension means that the classifier could be too close to the information presented on the Bug Report that it is not capable to generalize the information making it perform well on the same data-set, but not well enough on others.

Metrics	F-measure	S. Mostafa and X. Wang [6], M. Davari [42], Spanos et al. [29], Pandey et al. [51], Zou et al. [44], J. Tyo [55], G. Spanos and L. Angelis [21], L. Wan [49], Huang et al. [23], Mostafa et al. [7], S. E. Sahin and A. Tosun [24], Le et al. [25], Catolino et al. [45], Kudjo et al. [26], S. S. Alqahtani [33], Bulut et al. [48], Shu et al. [37], A. Duppils and M. Tullberg [27], Wu et al. [38], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Z. S. M. Alharthi and R. Rastogi [46], Chen et al. [30], Bhuiyan et al. [47], Wu et al. [40]
	Precision	Shuai et al.[4], Behl et al. [41], M. Davari [42], Spanos et al. [29],Y. Zhou and A. Sharma [43], Zou et al. [44], J. Tyo [55], G. Spanos and L. Angelis [21], L. Wan [49],Huang et al. [23], Mostafa et al. [7], S. E. Sahin and A. Tosun [24], Kudjo et al. [26], S. S. Alqahtani [33], Bulut et al. [48], A. Duppils and M. Tullberg [27], Wu et al. [38], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Z. S. M. Alharthi and R. Rastogi [46], Chen et al. [30], Bhuiyan et al. [47], Wu et al. [40]
	Accuracy	Gegick et al. [53], M. M. Rahman and S. Yeasmin [31] M. Davari [42], Spanos et al. [29], Pandey et al. [51], Zou et al. [44], J. Tyo [55], G. Spanos and L. Angelis [21], Huang et al. [23], Le et al. [25], Pereira et al. [56], Kudjo et al. [26], Palacio et al. [34], Wu et al. [38], Z. S. M. Alharthi and R. Rastogi [46]
	Recall	M. Davari [42], Spanos et al. [29], Y. Zhou and A. Sharma [43], Zou et al. [44], J. Tyo [55], G. Spanos and L. Angelis [21], L. Wan [49], Huang et al. [23], Mostafa et al. [7], S. E. Sahin and A. Tosun [24], Shu et al. [35], Kudjo et al. [26], S. S. Alqahtani [33], Bulut et al. [48], Shu et al. [37], A. Duppils and M. Tullberg [27], Wu et al. [38], T. M. Adhikari and Y. Wu [28], Jiang et al. [8], Z. S. M. Alharthi and R. Rastogi [46], Chen et al. [30], Wu et al. [40]
	Area Under the Curve	D. C. Das and Md. R. Rahman [50], O. Jormakka [22], Pereira et al. [56], Catolino et al. [45], Kudjo et al. [26], Palacio et al. [34], Bulut et al. [48], Zhang et al. [52], Chen et al. [30], Wu et al. [40]
	Matthews Correlation Coefficient	Kudjo et al. [26], Chen et al. [30]
	G-Score	J. P. Tyo [54], J. Tyo [55], Peters et al. [36], Jiang et al. [8]
	Bayesian Detection Rate	Wijayasekara [5]
	Probability of false alarm	Wijayasekara [5], J. Tyo [55], Shu et al. [35], Shu et al. [37], Jiang et al. [8]

Table 3.13: Metrics studies distribution.

Another aspect that could be seen as a gap, is that most studies presented an unbalanced distribution between Security Bug Report and Non-Security Bug Report, as mentioned before when answering the research question **RQ-2** this is problematic since it could lead to creating a bias classifier, making it not have reliable results. Having a bias classifier means that when faced with data where it does not have enough information to give a classification they will end up classifying the data with the same class as the class with a higher quantity of data.

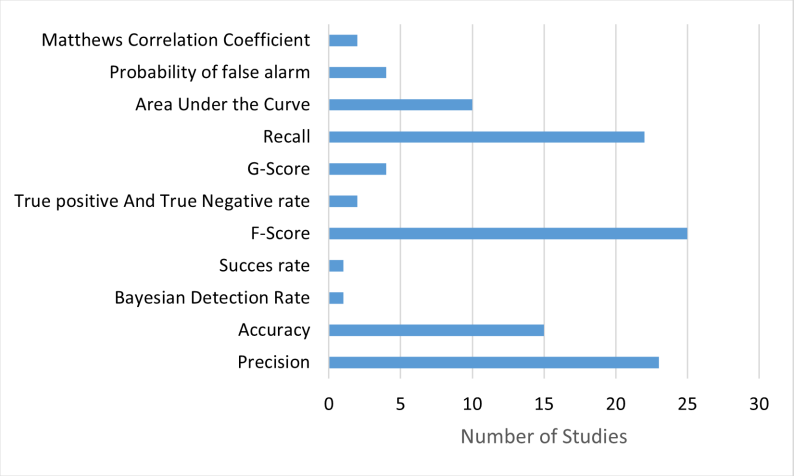

Figure 3.5: Distribution of Metrics for evaluation used.

Chapter 4

Data and Methods

In this section, we will discuss the approach used on the experiments made in this thesis, which encompasses the following steps:

1. **Collection and preprocessing of the dataset:** where it is explained where the datasets were taken from as well as the techniques used for the reduction of the vocabularies;
2. **Feature Extraction:** focus on the explanation of the TF-IDF;
3. **Feature Selection:** it is explained the functionality of the four feature selection algorithms used;
4. **Configuration and Training of the Classifiers:** in here it is presented with the explanations on how each classifier implemented;
5. **Classifiers Performance Assessment:** for last we have detailed all the metrics used in the process of evaluating the performance of the experiments.

The experiments were done in a machine with the following configuration:

- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.81
- Installed RAM: 16GB
- Operation System: Windows 10
- System Type: 64-bit operating system

4.1 Colletion and preprocessing of the dataset

The creation and preprocessing of the dataset used in this work involved the following steps:

i Selection of a set of projects;

ii Retrieval of a set of bug descriptions related to security and non-security the projects bug tracking platforms;

iii Tokenization and filtering of the descriptions

We started by selecting seven different projects from three different tracking systems. From Launchpad bugs from Openstack and Ubuntu; from Jira we picked bugs Kafka and Hadoop; and bugs from MySql, Firefox, and Eclipse were taken from Bugzilla. In total 277093 closed and resolved bug reports were collected with 8689 of them being related to security, in Table 4.1 it can be observed in more detail the composition of the dataset. During this process, some reports were verified to ensure the reports were not misclassified before our experiments so it would not affect the performance of the experiments.

Project	Security	Non-Security	Tracking System	Language
Kafka	42	2046	Jira	Java
Hadoop	546	13938	Jira	Java
Openstack	357	90728	Launchpad	Python
Ubuntu	3157	87703	Launchpad	C
Firefox	4229	32335	Bugzilla	C++ and JavaScript
MySql	171	28477	Bugzilla	SQL
Eclipse	187	13177	Bugzilla	Java

Table 4.1: Number of bugs collected from each project.

The information gathered from the bug reports used for the experiments were the title 4.1, and description 4.2 of the bug. After collecting the bugs we start to preprocess the information collected to execute this task the Matlab's [57] toolbox **Text Analytics Toolbox** [58].

OpenStack API Requiring X-Auth-User on all requests

Figure 4.1: Example of a bug title

We first tokenize all the document using tokenizedDocument function, after tokenizing the document we intent remove unnecessary information and for that, we use the functions:

- addPartOfSpeechDetails: detects parts of speech in documents and updates the token details. The function, by default, retokenizes the text for part-of-speech tagging. For example, the function splits the word "you're" into the tokens "you" and "'re";
- removeStopWords: removes the stop words from the tokenizedDocument array documents;
- normalizeWords: reduces the words in documents to a root form;
- erasePunctuation: erases punctuation and symbols from documents.

After preprocessing our data, the data is separated into training and testing sets using 10 Fold cross-validation. We decided to do undersampling for training while for the test we keep all the data. The undersampling was implemented by randomly selecting reports from the majority class (Non-security) to have at the end the same number of reports observed in the minority class (Security).

```
After authenticating with the OpenStack API, subsequent requests to nova should .
only require the X-Auth-Token for authentication. However, it also requires the '
X-Auth-User header to be present; it doesn't matter what the user value is.

Here is the stack trace for a request with only X-Auth-Token:

Traceback (most recent call last):
  File "/usr/local/lib/python2.6/dist-packages/eventlet/wsgi.py", line 336, in
handle_one_response
    result = self.application(self.environ, start_response)
  File "/usr/local/lib/python2.6/dist-packages/webob/dec.py", line 159, in
__call__
    return resp(environ, start_response)
  File "/usr/local/lib/python2.6/dist-packages/routes/middleware.py", line 131,
in __call__
    response = self.app(environ, start_response)
  File "/usr/local/lib/python2.6/dist-packages/webob/dec.py", line 159, in
__call__
    return resp(environ, start_response)
  File "/usr/local/lib/python2.6/dist-packages/webob/dec.py", line 159, in
__call__
    return resp(environ, start_response)
  File "/usr/local/lib/python2.6/dist-packages/webob/dec.py", line 159, in
__call__
    return resp(environ, start_response)
  File "/usr/local/lib/python2.6/dist-packages/webob/dec.py", line 147, in
__call__
    resp = self.call_func(req, *args, **self.kwargs)
  File "/usr/local/lib/python2.6/dist-packages/webob/dec.py", line 208, in
call_func
    return self.func(req, *args, **kwargs)
  File "/root/novascript/nova/nova/api/openstack/__init__.py", line 109, in
__call__
    username = req.headers['X-Auth-User']
  File "/usr/local/lib/python2.6/dist-packages/webob/datastruct.py", line 40, in
__getitem__
    return self.environ[self._trans_name(item)]
KeyError: 'HTTP_X_AUTH_USER'
```

Figure 4.2: Example of a bug description.

4.2 Feature Extraction

The goal of this step is to extract representative values from the reports' texts for informative words. The Feature Extraction used was TF-IDF even though there are other methods, namely: Bag-of-Words or Term Frequency.

The reason Bag-of-Words was not used is that because it only counts the number of times a word appears in a document. Doing this does not distinguish properly represent the importance a word has on a document or in the entire dataset.

The TF technique was not used because it counts the appearances of the word in the document given a higher value to the words that appear often in a document. This is problematic since the words that would have the highest value would be words common to appear in our context, making the classifiers most likely show poor performance.

The TF-IDF evaluates the relevance of the word in the collection of documents allowing us to understand its importance for a certain document, and to calculate the TF-IDF for a word in a document it is needed to multiply two different metrics. The TF part counts the appearances the world has in a document and the IDF is the ratio between the number of documents where a certain term t appears and the total number of documents. Using this method ensures that common words with higher counts in the entire dataset has a smaller value of importance for the classification problem.

The IDF value for a certain term t in the dataset D is calculate by the given Equation 4.1:

$$IDF(t, D) = \log\left(\frac{O}{df_t}\right). \tag{4.1}$$

where O represents the number of documents in the data set D, and the denominator is

the number of documents where the given term t appears, if the mentioned denominator is 0, it is added 1 so that the division becomes possible. When the values for the TF and IDF are multiplied we obtain the value TF-IDF, this allows us to know the importance a word has to each document. A word has more importance when it appears several times in a document and a few times in the dataset.

The TF represents the times a term t is in the a document d, in the end the reports are represented, given us the following Equation 4.2

$$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D) \tag{4.2}$$

4.3 Feature Selection

Four feature selections were selected in this study and they were: Chi-Square, Minimum Redundancy Maximum Relevance (MRMR) [59], F-Test, and SVM-RFE. The most common method used when dealing with text data is the Chi-Square or $\mathcal{X}^2$. This is used to test the independence of two events, more specifically in our case, test whether the occurrences of a specific term and a specific class are independent.

$\mathcal{X}^2$ dependence test is a hypotheses Test, so the two hypotheses are as follows:

- Null Hypothesis H_0: Assumes there is no dependence between the term and the class variables;
- Alternative Hypothesis H_1: Assumes there is dependence between the two variables.

With it, we compare the results below:

- Expected Result - We calculate the result following the H_0 assumption;
- Observed Result - This is what the testing(training) data presents

Then each term is ranked by the value returned by the following Equation 4.3:

$$\mathcal{X}^2(D, t, c) = \sum_{e_t \in 1,0} \sum_{e_c \in 1,0} \frac{(O_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \tag{4.3}$$

Where O is the observed frequency, in other words, the actual observed data, for example, you roll a dice ten times and then count how many times each number is rolled and E the expected frequency, meaning the count is calculated using probability theory, for instance, before you roll a six-sided dice you calculate the probability of any one number being rolled as $1/6$, e_t takes the value 1 if the document contains the term t and 0 otherwise and e_c takes the value 1 if the document is in class c and 0 otherwise. The high scores a term receives indicate that the term and the class of the document should be dependent.

A similar method to Chi-Square is using univariate Feature Ranking Using F-Tests examines the importance of each feature individually using an F-test. Each F-test tests the hypothesis:

- Null Hypothesis H_0: The class values grouped by feature values are drawn from populations with the same mean;

- Alternative Hypothesis H_1: the population means are not all the same.

A small p-value of the test statistic indicates that the corresponding predictor is important.

Next, we have MRMR [59]. MRMR aims to select, from among the features, the ones that show a low correlation among each other but still have a high correlation with the classification labels, meaning if a feature A and a feature B are relevant, but they both bring the same information, MRMR will select only one of them and discard the other. MRMR works iteratively, it identifies the best feature and adds to a set of selected features. A features importance can be obtained by the following Equation 4.4:

$$f(X_i) = I(Y, X_i) - \frac{1}{|S|} \sum_{X_s \in S} I(X_s, X_i) \tag{4.4}$$

where X_i is the feature to determine the importance with $i \in \{1, 2, \ldots, n\}$ when n is the total number of features and $X_i \notin S$, Y is the class labels, S is the set of feature that were already selected, $|S|$ is the total number of features already selected, X_s is one of the features belonging to the feature set S, and $I(\cdot, \cdot)$ is the function that calculates the mutual information.

And for last we have SVM-RFE which is a wrapper feature selection method. SVM-RFE is an application of RFE using the weight magnitude of a linear SVM classifier as ranking criterion. We present in Algorithm 1 an outline of the algorithm, made by Guyon et al. in [60]:

Algorithm 1 SVM-RFE:

Input:
$X_0 = [x_1, x_2, \cdots, x_k, \cdots, x_l]^T$ ▷ Training examples, where each x is a vector with each example information
$y = [y_1, y_2, \cdots, y_k, \cdots, y_l]^T$ ▷ Class labels
Initialize:
$s = [1, 2, \cdots, n]$ ▷ Subset of surviving features
r = [] ▷ Feature ranked list
Repeat until s =[]
$X = X_0(:, s)$ ▷ Restrict training examples to good feature indices
$\alpha = SVM - train(X, y)$ ▷ Train the classifier
$w = \sum_k \alpha_k y_k x_k$ ▷ Compute the weight vector of dimension length(s)
$c_i = (w_i)^2$, for all i ▷ Compute the ranking criteria
$f = \arg\min(c)$ ▷ Find the feature with smallest ranking criterion
r = [s(f), r] ▷ Update feature ranked list
s = s(1:f - 1, f + 1:length(s)) ▷ Eliminate the feature with smallest ranking criterion
Output:
r ▷ Feature ranked list

4.4 Classifiers

In this section, we will discuss and explain the functionality of the five classifiers used in this study: k-Nearest Neighbor, Naïve Bayes, SVM, Linear classifier, and Discriminant classifier. For each classifier three types of experiment were made, one in which the training

data is developed based on the title and description, another where training was just based on the title, and one that is based on the description.

At first we describe a generic Linear classifier, defined as:

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_j\right) \tag{4.5}$$

where $\vec{w}$ is the vector of weights, $\vec{x}$ is the features vector and $f(\cdot)$ is the function given by:

$$f(w \cdot x) = \begin{cases} 1 & \text{if w} \cdot \text{x} > T \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where T is a threshold to determine in which value to separate the classes.

Following Linear classifier we have Discriminant Analysis classifier, this classifier uses three values to classify its observation: posterior probability, prior probability, and cost.

$$\hat{y} = \underset{y=1,\ldots,K}{\arg\min} \sum_{k=1}^{K} \hat{P}(k|x) C(y|k) \tag{4.7}$$

where $\hat{y}$ is the predicted class, K is the number of classes, $\hat{P}(k|x)$ is the posterior probability of class k for observation x and $C(y|k)$ is the cost of classifying an observation as y when its true class is k. In [61] Matlab explains in more detail and explanation this classifier.

KNN labels a new pattern by finding its k closest neighbors and by making its label the one with the highest count among its neighbors, for that it is better to have an odd count of neighbors to ensure there is a class with a higher count then the other. The best parameter k is obtained with experimentation. In in this work eight different values were considered: *5, 7, 9, 11, 13, 15, 17, and 19.* And for the calculation of the distance, we use the default Euclidian distance.

Now we will discuss the Naïve Bayes. As the name implies it is based on the Bayes rule, given by the Equation 4.8 bellow:

$$\mathcal{P}(\omega|\mathbf{x}) = \frac{\mathcal{P}(\omega_i)\mathcal{P}(\mathbf{x}|\omega_i)}{\mathcal{P}(\mathbf{x})} \tag{4.8}$$

In the Equation 4.8 the $\mathbf{x}$ represents the feature vector of the report to be classified, ω_i represents the different classes, in our case SBR and NSBR, $\mathcal{P}(\mathbf{x}|\omega_i)$ is the likelihood for class ω and describes the probability of $\mathbf{x}$ knowing ω_i, $\mathcal{P}(\omega_i)$ is the prior probability and is computed by assessing the relative number of patterns belonging to ω_i in the training data. $\mathcal{P}(\mathbf{x})$ is the probability of $\mathbf{x}$ independent of its class. In the end the class of $\mathbf{x}$ is given by the biggest $\mathcal{P}(\omega_i)$. Then in the training of the classifier its objective is to determine $\mathcal{P}(\omega_i)$ and $\mathcal{P}(\mathbf{x}|\omega_i)$ for every class, where $\mathcal{P}(\mathbf{x}|\omega_i)$ can be given simply by:

$$\mathcal{P}(\mathbf{x}|\omega_i) \approx \prod_{k=1}^{n} \mathcal{P}(\mathbf{x}_k|\omega_i) \tag{4.9}$$

where $\mathbf{x}_k$ is a given value of the kth feature and n is the total number of features. In relation to the distribution type assumed for each $\mathcal{P}(\mathbf{x}_k|\omega_i)$, in this paper we considered the Multivariate multinomial distribution.

The SVM objective is to find a hyperplane in an N-dimensional space that can classify the data points correctly, where n is the number of features. This hyperplane however can not be any possible one, it has to be the one that maximizes the margin that separates two different classes. This hyperplane is the boundary that classify the data points, the class of the data points is attributed depending on which side of the hyperplane. If we consider a training data pair $\{x_i, y_i\}$ that is non-linearly separable where $i = 1, \cdots, N$, the class labels are defined by $y_i \in [-1, 1]$ and the corresponding feature vectors can be be represented as $x_i \in \mathcal{R}^d$, where the Equation 4.10 defines the hyperplane:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 - \xi_i; y_i \in [-1, 1] \tag{4.10}$$

where the vector normal to the hyperplane is represented by $\mathbf{w} = [w_1, \ldots, w_d]^T$ and and the degree of misclassifications is quantified by ξ. To obtain $\mathbf{w}$ and b it is needed to minimize the criterion in Equation 4.11.

$$\Psi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i, \text{ is subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 - \xi_i; i = 1, \ldots, N \tag{4.11}$$

where the influence of ξ on Ψ is defined by C, this influence of C is also tested in our experiments.

But usually, the classification problem has fairly complex data that is not linearly separable. For that SVM enable a data transformation to mirror the data into a new high-dimensional space in a way that the problem becomes linearly separable.

$$d(\mathbf{x}) = w_1 f_1(\mathbf{x}) + \ldots + w_k f_k(\mathbf{x}) + b \tag{4.12}$$

where the non-linear transformation from the original plane to the new plane of dimension k is represented by $\mathbf{f} = [f_1, \ldots, f_k]^T$. And by considering the dual optimization we can demonstrate that the decision function is as follows:

$$d(\mathbf{x}) = \sum_{SVs}\alpha_i y_i \mathbf{f}^T(\mathbf{x}_i)\mathbf{f}(\mathbf{x}) = \sum_{SVs}\alpha_i y_i(\mathbf{f}^T(\mathbf{x}_i)\cdot\mathbf{f}(\mathbf{x})) = \sum_{SVs}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}). \tag{4.13}$$

Where the Lagrange multipliers are the α_i. $K(\mathbf{x}_i, \mathbf{x}) = \mathbf{f}(\mathbf{x}_i)$ is inner-product kernel, where the most popular and used one is the Gaussian RBF:

$$K(\mathbf{x}_i, \mathbf{x}) = e^{\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}} = e^{-\gamma\|\mathbf{x}-\mathbf{x}_i\|^2}. \tag{4.14}$$

$\gamma = \frac{1}{2\sigma^2}$ is the parameter that controls the aperture of the kernel function, and go trough a tuning procedure in conjunction with the parameter C.

4.5 Classifiers Performance Assessment

To validate the classifiers, it was used K-Fold Cross Validation was applied with the most common number of folds, i.e, K = 10. Supposing that the dataset is composed of $\mathbf{X}$ reports, the algorithm will split the dataset in K folds, after splitting the data the classifiers are trained K times. In each iteration, the training set consists of K-1 parts, this way each part is used once as the test set and K-1 times as the training set. After the test is complete the following measurements were taken:

- True positive (TP): number of reports correctly classified as security vulnerability;
- False positive (FP): number of reports of the non-vulnerability class wrongfully classified as containing a vulnerability issue;
- True negative (TN): number of reports correctly classified as non-vulnerability;
- False positive (FN): number of reports of the security vulnerability class wrongfully classified as not containing a vulnerability issue;

We evaluate the algorithms with 5 metrics: accuracy, recall, precision, F-Measure

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.15}$$

$$recall = \frac{TP}{TP + FN} \tag{4.16}$$

$$precision = \frac{TP}{TP + FP} \tag{4.17}$$

$$F - Measure = \frac{(2 \times precision \times recall)}{precision + recall}, \tag{4.18}$$

and at last, the AUC, the AUC represents the performance measurement of the classifier at a different threshold. By calculating ROC we get a probability curve between the TPR and FPR, helping us visualize how much the model can distinguish two classes (see Figure 4.3).

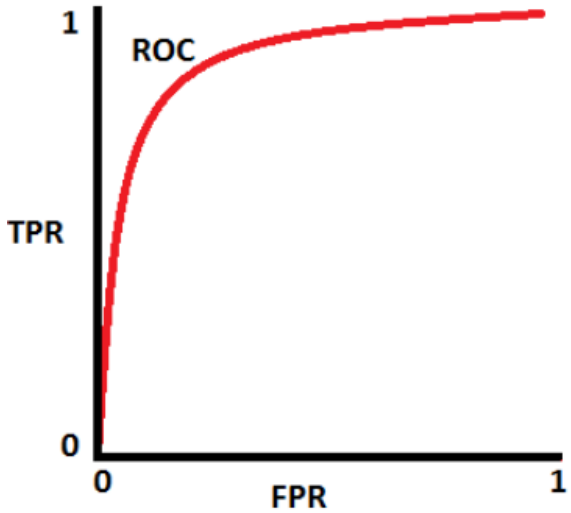

Figure 4.3: Example of a ROC curve.(Image taken from [3])

A classifier with good performance can have the curve closer to the value 1 on the TPR axis and closer to 0 on the FPR axis (Figure 4.4).

But having graphs is complicated when it is desired to make comparisons with several other experiments, so AUC is used as a value that represents the graphs, it is the value of the area below the curve. An AUC value close to 1 would mean the closer the curve is to the optimal curve. .

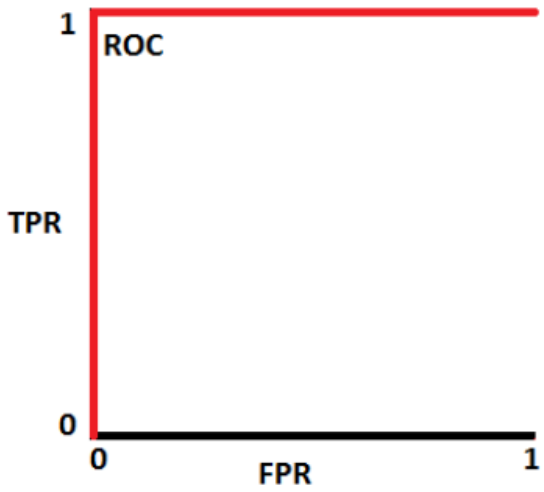

Figure 4.4: Example of a optimal ROC curve.(Image taken from [3])

Chapter 5

Results and Discussion

This section discusses the results obtained from the different experiments. The experiments were made using undersampling to have the same quantity of reports from both classes on the training data, while the test data remained with the same proportions as it is found in a real world scenario. With it, three experiments were made one using the title and description of the reports, another using only the title, and another using only the description. The values presented in this section refer to the average of the Area Under the Curve results obtained after running all experiments with K-Fold Cross-Validation. In all experiments the test of Kolmogorov Smirnov was used to test normality and when then the Kruskal-Wallis test was used when data normality was rejected.

This section is also divided into three parts, the first part is an analysis of the results when using all the projects. In the second part, we have another analysis of results but based on experimentation using one project. And for last a compilation of the main findings.

The stars in the graphs represent that there is a significant differences between pairs. Stars are drawn according to:

- * represents a p-value≤ 0.05
- ** represents a p-value$\leq 1 \times 10^{-2}$
- *** represents a p-value$\leq 1 \times 10^{-3}$

5.1 Analysis of results using all projects

To verify the influence of the different aspects of our experiments a statistical analysis was implemented to verify if there is or not a statistical difference between experiments. So in the first place, an analysis of the data distribution was made to verify data normality, and for that Kolmogorov Smirnov test was used. In the first case, we will analyze the impact of information used, i.e., only Title, only Description, or Title and Description together.

The Kolmogorov Smirnov test was verified that, with a confidence level of 95%, indicated that data is not normal and consequently, the non-parametric Kruskal-Wallis test was considered. After executing the Kruskal-Wallis test, the observation was that all three methods present performance values that are statistically different.

In Figure 5.1, we can see that using only the Title leads to slightly worst results than using Title and Description, and using only the Description is the one with the worst values.

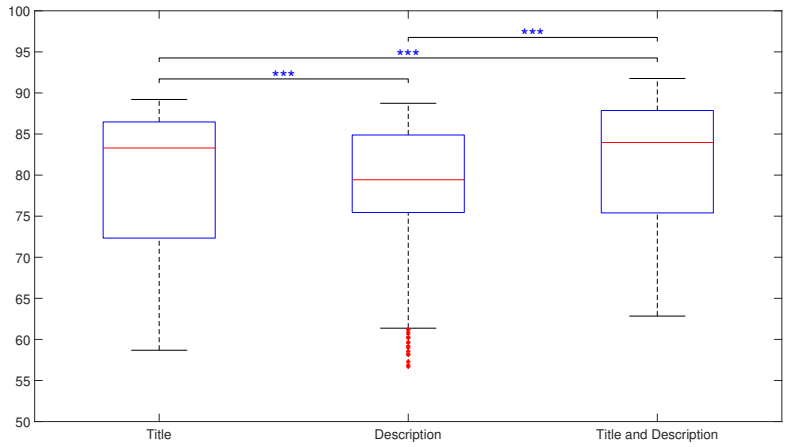

Figure 5.1: Statistical comparison of the data used on the model.

Now to compare the different feature selections used in this study, we did the same test of normality of Kolmogrov Smirnof. The hypothesis of the data following a normal distribution was rejected which makes use the same non-parametric test.

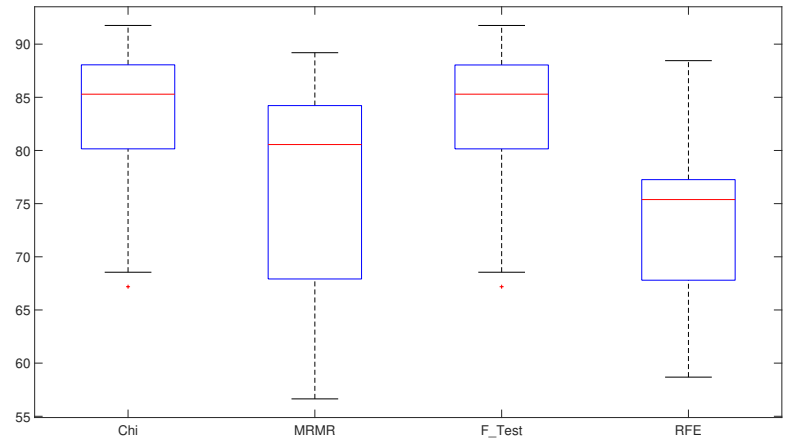

Figure 5.2: Statistical comparison of the different Feature Selections used.

After implementing the Kruskal-Wallis test, we can observe in Figure 5.2 that Chi-Square and F-test present similar results and above the other two feature selections, with Recursive Feature Elimination (RFE) presenting the worst results overall. F-test and Chi-Square presented the best results.

Before comparing the performance obtained with the different classifiers, an analysis of which k in KNN, and which C in SVM presents best results was implemented. After that, the best k and C values will be fixed and used in the remaining analysis.

Again with KNN we did the same test of normality of Kolmogorov Smirnov, and again the data normality was rejected and with Kruskal-Wallis test, it was observed that using a k equal to 15 is the one that has a difference with some of the other k's as seen in Figure 5.3.

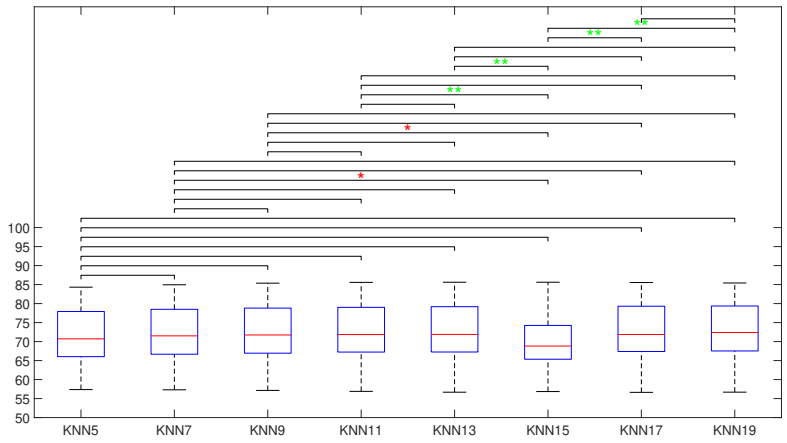

Figure 5.3: Statistical comparison of the KNN with different number of neighbors.

As we can observe in Figure 5.3 the mean results of k equal to 15 are the lowest of them. So we used the k equal to 19 to make the comparison between models, although in this case all others could be used in place of 19, the only exception being 15.

Again with SVM with the different parameter C we did the same test of normality and also an individual test with the graphs for each, in the test used when the normality is rejected it was observed that using a C equal to 2^{-14} is the one that has a difference with the other parameter C as seen in 5.4, with the only exception being the parameter C equal to 2^{-12}.

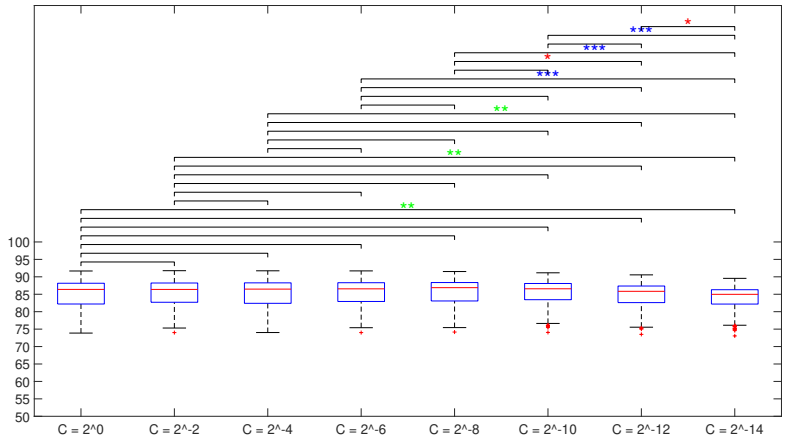

Figure 5.4: Statistical comparison of the SVM with different values for parameter C.

As we can observe in 5.4 the mean results of C equal to 2^{-14} are the lowest them, with C equal to 2^{-12} the second-worst, and what can also be observed is that there is not a big difference in means on the rest of experiments, so we used the C equal to 2^{-8} but any of the others could also be used for the comparisons that follow.

Now it is time for the model comparison, same as the other tests we begging with testing the normality of the data and once again it is rejected with a 5% significance level, then

Kruskal-Wallis test is implemented that gives the results seen in Figure 5.5

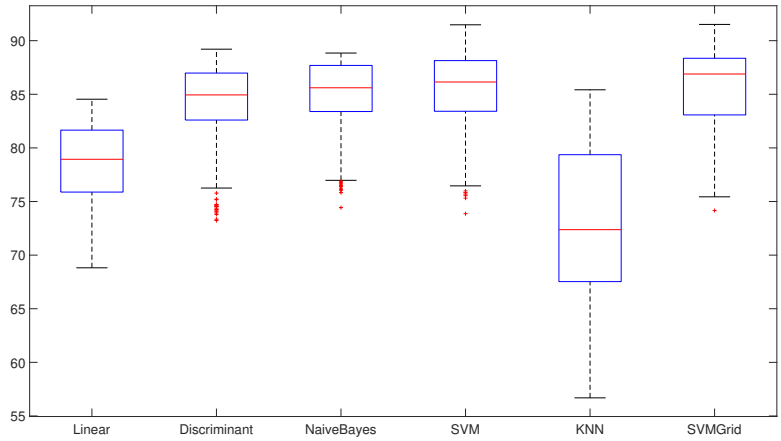

Figure 5.5: Statistical comparison of the different Classifiers.

As we can observe both SVM with the default parameter C and the SVM with our selected parameter C (SVMGrid) present no significant difference and have the highest mean followed by Naive Bayes that has no difference with SVM, but does have with the SVM with our custom parameter C. KNN on the other hand presents the worst results, this may be because it has to look to the k closest points to determine the class, which can lead to some miss classification if an appropriate k is not found.

For last we have the comparison of the different range of features used. In this part of the experiments we wanted to see the influence in the use of a different range in features had in the classification, our range varied from 120 features to 300 with jumps of 20. Just like the others the test of the null hypothesis of the data following a normal distribution was rejected, with that, a Kruskal-Wallis test is implemented as can be observed in Figure 5.6

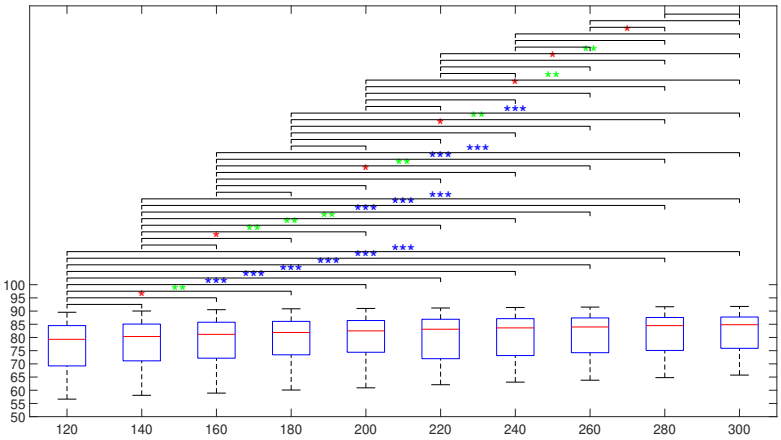

Figure 5.6: Statistical comparison of the different quantities of features used.

Interestingly the means do not have a big difference for more than 240 features, and it appears to stagnate for more features than this number.

To simplify the analysis, and give an overview of the obtained results, we selected the value of 300 features, but as discussed in the previous section 240 and 280 do not have a big difference from 300 so they could also be used for this presentation.

Feature Selection	Algorithm	Metrics				
		Precision (%)	*Recall (%)*	*F-Measure (%)*	*Accuracy (%)*	*AUC (%)*
Chi-Square	Linear	11.40	65.64	19.32	82.39	82.61
	Discriminant	15.72	73.18	25.82	86.63	89.21
	KNN	6.69	87.75	12.43	61.24	76.75
	SVM	16.87	69.61	27.14	88.27	88.38
	Naïve	10.31	81.28	18.29	77.23	88.55
	SVM 2^-8	17.04	70.66	27.45	88.28	88.81
MRMR	Linear	11.46	61.81	19.21	83.19	81.06
	Discriminant	13.54	71.58	22.14	82.53	87.06
	KNN	5.67	89.77	10.66	52.50	72.02
	SVM	16.82	64.06	26.58	88.86	86.02
	Naïve	11.89	75.02	19.96	78.97	86.86
	SVM 2^-8	18.37	65.96	28.70	89.69	87.37
F-Test	Linear	11.89	67.20	20.14	83.01	83.00
	Discriminant	15.72	73.19	25.82	86.62	89.21
	KNN	6.69	87.74	12.43	61.23	76.75
	SVM	16.91	69.43	27.18	88.32	88.38
	Naïve	10.31	81.29	18.29	77.22	88.55
	SVM 2^-8	17.03	70.66	27.44	88.28	88.81
SVM-RFE	Linear	11.79	67.19	20.00	82.91	82.70
	Discriminant	15.54	72.15	25.57	86.83	88.45
	KNN	6.22	84.54	11.58	59.47	74.82
	SVM	16.58	70.67	26.85	87.92	87.69
	Naïve	14.10	71.81	23.53	85.17	87.94
	SVM 2^-8	18.15	68.07	28.65	89.36	88.03

Figure 5.7: Results from experiment using only the Titles.

Feature Selection	Algorithm	Metrics				
		Precision (%)	*Recall (%)*	*F-Measure (%)*	*Accuracy (%)*	*AUC (%)*
Chi-Square	Linear	11.01	63.76	18.72	82.39	81.16
	Discriminant	12.01	70.22	20.51	82.92	85.08
	KNN	8.71	76.92	15.64	73.96	81.11
	SVM	20.86	66.14	31.70	91.07	88.53
	Naïve	7.49	85.01	13.76	66.51	85.89
	SVM 2^-8	20.26	66.32	31.02	90.75	88.64
MRMR	Linear	14.97	60.64	23.66	86.49	79.74
	Discriminant	14.86	62.59	23.57	86.39	82.71
	KNN	5.03	90.04	9.52	45.70	65.74
	SVM	22.59	58.33	32.47	92.32	84.84
	Naïve	15.65	63.69	25.01	87.80	84.93
	SVM 2^-8	23.91	57.63	33.61	92.75	85.11
F-Test	Linear	9.89	68.11	17.17	78.55	81.36
	Discriminant	12.01	70.22	20.51	82.92	85.08
	KNN	8.71	76.92	15.64	73.96	81.11
	SVM	20.94	66.16	31.81	91.10	88.56
	Naïve	7.49	85.01	13.76	66.51	85.89
	SVM 2^-8	20.26	66.33	31.03	90.75	88.64
SVM-RFE	Linear	4.81	82.09	9.08	48.24	71.44
	Discriminant	5.03	85.49	9.50	48.93	74.57
	KNN	4.81	85.15	9.11	46.59	68.00
	SVM	6.08	82.01	11.03	55.29	77.20
	Naïve	5.60	83.45	10.50	55.35	77.31
	SVM 2^-8	6.81	79.17	12.03	60.13	77.51

Figure 5.8: Results from experiment using only the Description.

As we can observe from the Tables 5.7, 5.8 and 5.9 that the results from measurements that do not take into account the difference in data from the different classes have quite mixed results, for example, accuracy represents the value of correctly classified documents and looking at the results and not the data we could say that it has a great performance, but the problem is that from $\approx$ 27.000 documents that were part of the testing set only around 800 were considered the positive class, being a security bug report, so the high value in the accuracy could be the correctly classified from the negative class, non-security bug reports, that offsets the ones from the positive.

Feature Selection	Algorithm	Metrics				
		Precision (%)	*Recall (%)*	*F-Measure (%)*	*Accuracy (%)*	*AUC (%)*
Chi-Square	Linear	8.86	79.89	15.88	72.76	84.17
	Discriminant	13.61	76.77	23.06	83.81	88.17
	KNN	11.42	75.02	19.82	80.95	85.42
	SVM	25.04	70.95	37.00	92.42	91.48
	Naïve	9.12	86.45	16.49	72.43	88.85
	SVM 2^-8	24.18	71.77	36.15	92.05	91.52
MRMR	Linear	16.79	63.12	25.86	87.54	81.31
	Discriminant	22.30	62.16	32.57	91.78	85.56
	KNN	5.08	92.18	9.62	44.86	67.58
	SVM	28.49	61.42	38.85	93.91	86.60
	Naïve	19.19	66.02	29.65	90.12	87.18
	SVM 2^-8	29.43	62.57	39.95	94.09	87.67
F-Test	Linear	8.86	79.89	15.88	72.76	84.17
	Discriminant	13.61	76.77	23.06	83.81	88.17
	KNN	11.42	75.02	19.82	80.95	85.42
	SVM	25.21	70.76	37.17	92.50	91.47
	Naïve	9.12	86.45	16.49	72.43	88.85
	SVM 2^-8	24.20	71.76	36.18	92.06	91.52
SVM-RFE	Linear	6.12	86.51	11.42	57.67	75.37
	Discriminant	5.03	89.42	9.52	46.59	76.26
	KNN	4.72	91.62	8.97	41.52	69.92
	SVM	6.29	88.69	11.75	58.15	77.95
	Naïve	6.57	86.59	12.21	60.82	78.60
	SVM 2^-8	5.97	89.64	11.20	55.39	78.13

Figure 5.9: Results from experiment using Title and Description.

The same can be observed with precision, the great difference in the number of documents from each class especially having more documents from the negative one leads to a higher quantity of false positives leading to very low precision. Recall although not affected by the different quantity of reports of each class, in our case, is not a very good metric exactly for that even though it presents good values.

We gave also importance to another metric the AUC, which is also an appropriate measure for unbalanced data-sets. The higher values of AUC the better the classifier are in distinguishing the two classes.

5.2 Analysis of results using one project

A similar analysis as presented in the previous section was done, but this time only one project is used. Firefox was the one decided to be used for having the highs number of security bug reports compared to the other projects. The goal of the second experimentation is to see if the use of multiple projects would have an impact on its performance compared to using only one project.

As we can observe in Figure 5.10 for the different data used in the model, we see a significant difference compared with the previous experiment. The title is now the one with a better median instead of the use of Title and Description that has a lower median than the experiments with all projects, and the same can be observed on Description. This difference observed can probably be since we have only descriptions of the problem of Firefox and descriptions that were identified as of a security problem on other projects are not present and perhaps not pushing those terms to have a better relevance when identified by a feature selection. On the other hand, Title has a better performance for it having terms only related with their project which could make their specific terms more relevant overhaul when helping identify their documents class.

This pattern can be observed on the other graphs as well, but we will discuss them one by one. At first, we have the feature selection on Figure 5.11, this one does not have more to

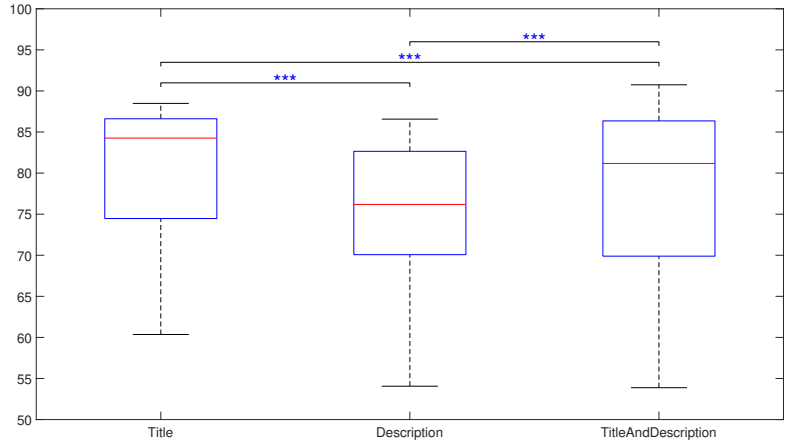

Figure 5.10: Statistical comparison of the data used on the model.

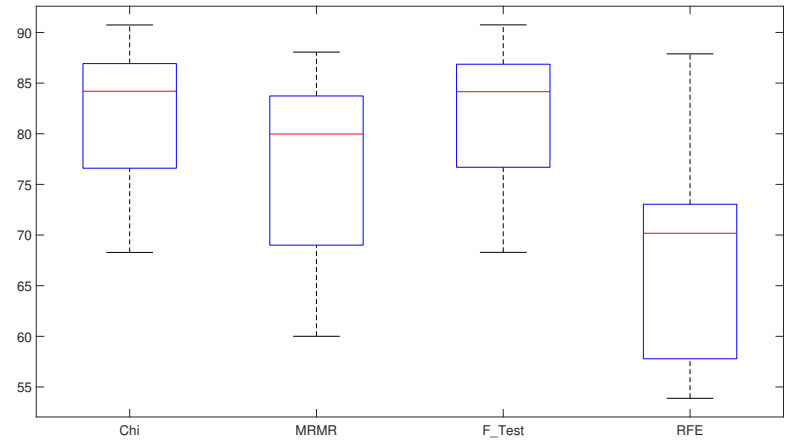

Figure 5.11: Statistical comparison of the different Feature Selections used.

add, its differences with the experiment with multiple projects is the same as the previous example, the difference is slightly lower medians and lower low values, with MRMR being the only exception having a higher lower value than with all projects, this could be because of how MRMR selects its features, as explained in Chapter 4m since all terms are related to a single project it is easier to identify terms that are important since terms only present in this project can be selected.

Now on KNN, Figure 5.12, the situation is a little different. The median value saw an improvement instead of being worse, but the highest and lowest values saw a decrease in value. This happens due to the presence of more particular terms is why we can see this slight improvement in the median, these terms only present in the Firefox project probably help in separate better between the two classes which helps the KNN to determine a reports class.

With the different values of C when using SVM a similar decrease in performance is seen like with using different data to do the experiment or feature selection, Figures 5.10 and 5.11, and also a worst statistical differentiation between them similar to what we saw in

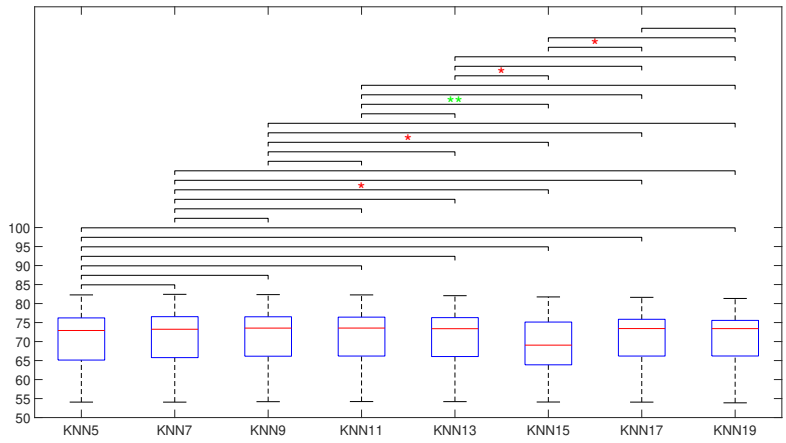

Figure 5.12: Statistical comparison of the KNN with different number of neighbors.

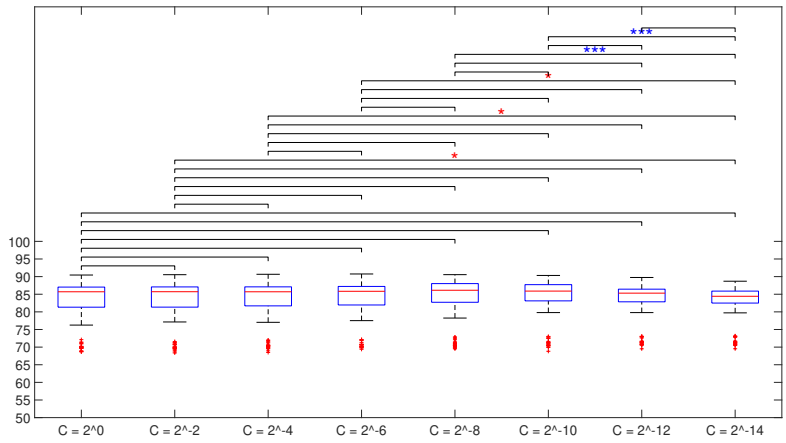

Figure 5.13: Statistical comparison of the SVM with different values for parameter C.

KNN.

Now for last, we will discuss the different classifiers and numbers of features used, Figure 5.14 and 5.15 respectively. Here the difference in performance is slightly worse as well when compared to when we used several projects. In the case of the classifiers, this could be for the nonexistence of several terms that are present in the other projects that better help in this problem in classification and its several aspects. KNN is the only exception, is the only classifier that had an improvement in the use of only one project, but although with this improvement it is not enough to make it the better option. In the case of the features, there is a slight reduction in performance, but we can still see the indifference when using more than 240 features.

5.3 Main findings

In this part, we will discuss the main findings when we look at the results that we obtained.

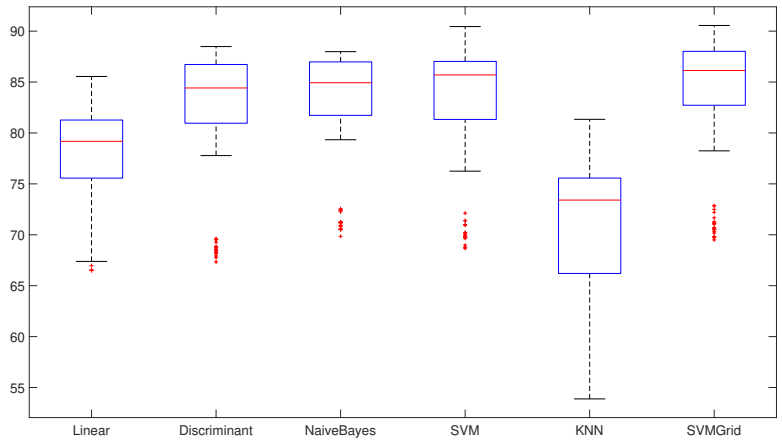

Figure 5.14: Statistical comparison of the different Classifiers.

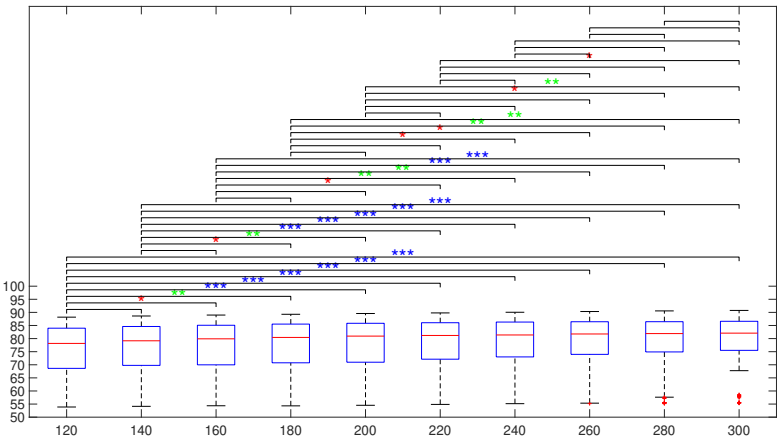

Figure 5.15: Statistical comparison of the different quantities of features used.

- As discussed previously in this chapter, using only the Title as input data to train the model, produces results worst than using as input data the Title with Description, But since the difference, although not negligible, are minimal we can search for other aspects than simply the performance in results to see a benefit in using only the Title as input data, for example when it is observed the time it takes to train the models we could observe differences up to 3 days for the completion of the experiments with using only the Title being the one that takes less time;

- A linear or probabilistic approaches like SVM and Naive Bayes are two of the best approaches with the Discriminant classifier being a little behind them. Another classifier that sees a good result would be SVM when we execute a search for parameter C, but the search for a parameter C has a negligible improvement in results compared to the time it takes to conclude the experiments are. The lack of improvement makes it not worth it when having large datasets since the normal SVM already obtains great results;

- When using a single project to train the model, instead of using several projects, we

can observe that it produces a minor difference between them. The use of one project has a decrease in performance of the results, but all the steps for when training a model took significantly less time than when using several projects, obviously because of the low quantity of data to process;

- The more features (words) that are used the better the results obtained, as can be observed in the two Figures 5.2 and 5.11 the higher the count of features used the better they get, but as can also be observed that for more than 240 features no significant improvements in performance was observed;
- Feature selection with filter is better than making use of SVM-RFE. But we can not guaranty that they are better than a wrapper type since we do not have more examples to compare.

Concerning how our work compares to the other works done previously, in terms of performance. We had mixed conclusions, the precision and recall on those studies had better results, but as far as we can tell we are the firsts to do undersampling only on the training data and not on testing data. But the interesting aspect is that we do have higher values in the AUC when compared to some studies [22, 26, 30, 34, 48] that made use of it with a minimum improvement in the performance of around 3% [30], when comparing highest value obtained.

Chapter 6

Conclusion

Software verification and validation include numerous important activities that can be used to build more reliable software. Whenever a bug is registered, the proper resources need to be allocated. The primary matter is to decide if the report is indeed a bug. Then a priority is given, meanwhile, information such as the nature of the error, impact, how to reproduce it, etc. have been added to the given tracking system, which allows the developer to correct them. If it's not detected and classified correctly, then the entire process will become very inefficient, which in itself is a time-consuming task. Companies like Banks, Google, Facebook, etc. could suffer if a bug is incorrectly classified as a non-security bug, the company could suffer from hackers attacks or reveal user-sensitive information.

In this thesis, we present a review of the state-of-the-art concerning Security Bug Report classification and the results and findings of our experimentation. We analyzed a final set of 42 papers. The focus was to answer five main questions related to i) the objective of the works ii) the characteristics of the datasets; iii) the most used pipeline; iv) the partition of the dataset; v) the metrics used for the evaluation of the approach. The discussion identified a clear gap related to how the features are processed.

Based on the analyzes of the state-of-the-art, we designed an approach for classifying Security Bug Report. As our main findings, we found out that the use of several projects have a performance similar to the use of a single project as well as finding out that some of the approaches used do indeed affect the performance of the model for example KNN presenting a visible performance problem when compared to the other classifiers.

One of the main difficulties found during the identification of works is related to the use of different terms to refer to classifying Security Bug Report across authors. This may have led us to wrongly exclude certain papers as well as to include less relevant papers. Since the process relies heavily on the richness of the data available in the online databases, the final set of papers may be incomplete, in case research is neither properly indexed nor present by the online databases. To minimize the effects of this threat, we make use of a total of six online databases, which are well-known and frequently used in this type of study. Also, the search string used to search the online databases may be weak leading to either too many results (including irrelevant ones) or too little. To resolve this issue, we included different words based on common terms used by authors, which we found in a preliminary analysis of the state-of-the-art.

The other difficulties came when executing the experiment those experiments took much more time to execute than expected, due to the amount of data collected that we had to make use of undersampling the data used for training although it had a much better time

than using all the information collected it still took longer than expected to complete.

And as for some ideas on what to the in future experiments, that we were unable to execute in this thesis. For starters what could be done in future work could be of making use of a better method of fine-tuning the SVM with a better grid search approach not only to the parameter C, but also to other aspects of this classifier, to do a more refined search for configuration on different value like the parameter C, the Epsilon, KernalScale. The same can be said for Naive Bayes, although not having as many values to test with as SVM it does have some, like the Distribution of the data at the start we had it set for a normal distribution, but after testing with Multivariate multinomial distribution we obtained a better set of results and used it for the rest of experiments.

References

[1] TRY QA. http://tryqa.com/what-is-a-defect-life-cycle, Last accessed September 2021.

[2] O'Reilly. https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/ch04.html, Last accessed September 2021.

[3] Towards Data Science. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5, Last accessed June 2021.

[4] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang. Automatic classification for vulnerability based on machine learning. pages 312–318, 2013.

[5] D. Wijayasekara, M. Manic, J. L. Wright, and M. McQueen. Mining bug databases for unidentified software vulnerabilities. pages 89–96, 2012. ISSN: 2158-2254.

[6] Shaikh Mostafa and Xiaoyin Wang. Automatic identification of security bug reports via semi-supervised learning and CVE mining. 2016.

[7] S. Mostafa, B. Findley, N. Meng, and X. Wang. SAIS: Self-adaptive identification of security bug reports. pages 1–1, 2019.

[8] Yuan Jiang, Pengcheng Lu, Xiaohong Su, and Tiantian Wang. LTRWES: A new framework for security bug report detection. 2020.

[9] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 2007.

[10] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele Univ.*, 33, 2004.

[11] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. Association for Computing Machinery, 2014.

[12] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Computing Surveys*, 50:56:1–56:36, August 2017.

[13] Dong-Gun Lee and Yeong-Seok Seo. Systematic Review of Bug Report Processing Techniques to Improve Software Management Performance. *Journal of Information Processing Systems*, 2019.

[14] Jie Zhang, XiaoYin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. A survey on bug-report analysis. *SCIENCE CHINA Information Sciences*, 2015.

[15] Google. Google Scholar, Last accessed 03-11-2020. https://scholar.google.com/.

[16] DBLP Team. DBLP, Last accessed 03-11-2020. https://dblp.uni-trier.de/.

[17] IEEE. IEEE Xplore, Last accessed 03-11-2020. https://ieeexplore.ieee.org/Xplore/home.jsp.

[18] Association for Computing Machinery. ACM Digital Library, Last accessed 03-11-2020. https://dl.acm.org/.

[19] Elsevier BV. Scopus, Last accessed 03-11-2020. https://www.scopus.com/home.uri.

[20] Springer Nature Switzerland AG. Springer Link, Last accessed 03-11-2020. https://link.springer.com/.

[21] Georgios Spanos and Lefteris Angelis. A multi-target approach to estimate software vulnerability characteristics and severity scores. pages 152–166, 2018.

[22] Ossi Jormakka. Approaches and challenges of automatic vulnerability classification using natural language processing and machine learning techniques. 2019.

[23] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao. Automatic classification method for software vulnerability based on deep neural network. 2019.

[24] Sefa Eren Sahin and Ayse Tosun. A conceptual replication on predicting the severity of software vulnerabilities. In *Proceedings of the Evaluation and Assessment on Software Engineering*. Association for Computing Machinery, 2019.

[25] T. H. M. Le, B. Sabir, and M. A. Babar. Automated software vulnerability assessment with concept drift. pages 371–382, 2019. ISSN: 2574-3864.

[26] P. K. Kudjo, J. Chen, M. Zhou, S. Mensah, and R. Huang. Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment. 2019.

[27] Anton Duppils and Magnus Tullberg. Semi-supervised text classication: Automated weak vulnerability detection. 2020.

[28] T. M. Adhikari and Y. Wu. Classifying software vulnerabilities by using the bugs framework. 2020.

[29] Georgios Spanos, Lefteris Angelis, and Dimitrios Toloudis. Assessment of vulnerability severity using text mining. ACM Press, 2017.

[30] Jinfu Chen, Patrick Kwaku Kudjo, Solomon Mensah, Selasie Aformaley Brown, and George Akorfu. An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection. 2020.

[31] Mohammad Masudur Rahman and Shamima Yeasmin. Adaptive bug classification for CVE list using bayesian probabilistic approach. 2013.

[32] Vulnerability identification and classification via text mining bug databases. pages 3612–3618, 2014.

[33] Sultan S. Alqahtani. Automated extraction of security concerns from bug reports. pages 1–3, 2019. ISSN: 2643-4202.

[34] D. N. Palacio, D. McCrystal, K. Moran, C. Bernal-Cárdenas, D. Poshyvanyk, and C. Shenefiel. Learning to identify security-related issues using convolutional neural networks. pages 140–144, 2019. ISSN: 2576-3148.

[35] Rui Shu, Tianpei Xia, Laurie Williams, and Tim Menzies. Better security bug report classification via hyperparameter optimization. 2019.

[36] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. Text filtering and ranking for security bug report prediction. (6), 2019.

[37] Rui Shu, Tianpei Xia, Jianfeng Chen, Laurie Williams, and Tim Menzies. Improved recognition of security bugs via dual hyperparameter optimization. 2019.

[38] Xiaoxue Wu, Wei Zheng, Xiang Chen, Fang Wang, and Dejun Mu. CVE-assisted large-scale security bug report dataset construction method. 2020.

[39] Wajdi Aljedaani, Yasir Javed, and Mamdouh Alenezi. LDA Categorization of Security Bug Reports in Chromium Projects. In *Proceedings of the 2020 European Symposium on Software Engineering*, pages 154–161. Association for Computing Machinery, 2020.

[40] Xiaoxue Wu, Wei Zheng, Xiang Chen, Yu Zhao, Tingting Yu, and Dejun Mu. Improving high-impact bug report prediction with combination of interactive machine learning and active learning. *Information and Software Technology*, 2021.

[41] Diksha Behl, Sahil Handa, and Anuja Arora. A bug mining tool to identify and analyze security bugs using naive bayes and TF-IDF. pages 294–299, 2014.

[42] Maryam Davari. Classifying and predicting software security vulnerabilities based on reproducibility. 2016.

[43] Yaqin Zhou and Asankhaya Sharma. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, pages 914–919. ACM Press, 2017.

[44] Deqing Zou, Zhijun Deng, Zhen Li, and Hai Jin. Automatically identifying security bug reports via multitype features analysis. In Willy Susilo and Guomin Yang, editors, *Information Security and Privacy*, Lecture Notes in Computer Science. Springer International Publishing, 2018.

[45] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. Not all bugs are the same: Understanding, characterizing, and classifying bug types. 2019.

[46] Zaher Shuraym M. Alharthi and Ravi Rastogi. An efficient classification of secure and non-secure bug report material using machine learning method for cyber security. page S2214785320361836, 2020.

[47] Farzana Ahamed Bhuiyan, Raunak Shakya, and Akond Rahman. Can we use software bug reports to identify vulnerability discovery strategies? In *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*. Association for Computing Machinery, 2020.

[48] F. G. Bulut, H. Altunel, and A. Tosun. Predicting software vulnerabilities using topic modeling with issues. pages 739–744, 2019.

[49] Liuyang Wan. Automated vulnerability detection system based on commit messages, 2019.

[50] Dipok Chandra Das and Md. Rayhanur Rahman. Security and performance bug reports identification with class-imbalance sampling and feature selection. pages 316–321, 2018.

[51] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. Automated classification of software issue reports using machine learning techniques: an empirical study. 2017.

[52] X. Zhang, H. Xie, H. Yang, H. Shao, and M. Zhu. A general framework to understand vulnerabilities in information systems. 2020.

[53] Michael Gegick, Pete Rotella, and Tao Xie. Identifying security fault reports via text mining. 2009.

[54] Jacob P. Tyo. Empirical analysis and automated classification of security bug reports. 2016.

[55] Identification of security related bug reports via text mining using supervised and unsupervised classification. pages 344–355, 2018.

[56] Mayana Pereira, Alok Kumar, and Scott Cristiansen. Identifying security bug reports based solely on report titles and noisy data. pages 39–44, 2019.

[57] MathWorks. Matlab, Last accessed June 2021. `https://www.mathworks.com/products/matlab.html`.

[58] MathWorks. Text Analytics Toolbox, Last accessed June 2021. `https://www.mathworks.com/products/text-analytics.html`.

[59] Zhenyu Zhao, Radhika Anand, and Mallory Wang. Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform.

[60] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines.

[61] MathWorks. Discriminant Analysis, Last accessed June 2021. `https://www.mathworks.com/help/stats/prediction-using-discriminant-analysis-models.html`.