UNIVERSITY OF SALERNO

Department of Computer Science

Master of Science in Computer Science

MASTER'S DEGREE THESIS

Software Vulnerability Prediction with Real-World Labelling

SUPERVISOR

Prof. **Filomena Ferrucci**

Doct. **Giulia Sellitto**

University of Salerno

CANDIDATE

Alexandra Sheykina

0522500387

Academic Year 2020-2021

Abstract

Software always grows both in size and complexity. Identifying security vulnerabilities in software code is very difficult. If vulnerable components of software systems could be detected before its distribution, many cybersecurity attacks could be avoided. So far, researchers have proposed several approaches to address this problem. Machine learning methods are extensively studied for software vulnerability prediction. Most researchers perform tests in a synthetic context, and therefore they are not always set up in a realistic way. In most experiments, the machine learning model is trained against perfect labelling information, regardless of the fact that vulnerabilities are discovered over time, and their presence in the code is not always known in reality. These considerations, first made by Jimenez et al.[24], motivate our empirical study. We want to analyze the performance of machine learning models for vulnerability prediction in the real context. The real context is intended as an experimental setting in which files are labeled as vulnerable if and only if the vulnerabilities present in them have been already discovered before training time. To run experiments in this setting, an ad-hoc dataset has to be built, including realistic labels for each specific training set. In the realistic setting we can understand if vulnerability prediction models can be applied during real software development. We use the PHP vulnerability dataset from Walden et al. [9] and the two types of learning based on supervised learning and unsupervised learning. We then compare the performance obtained from these models in the real context. Finally, we show that the realistic labeling approach has very low performance, highlighting that further research is required to make these approaches useful for practitioners.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In this chapter we provide an overview on our work. Initially, we present a brief overview of the vulnerability problem in software. Then, we explain the motivation and describe the empirical study design, along with a brief discussion on the obtained results. Finally, we present the organization of the thesis.

The problem of software vulnerabilities appeared at the same time as the software itself appeared, and it continues to be a major and costly problem. In 2020, the year of pandemic, lockdown and remote working, cybersecurity was put to the test, reaching the highest number of attacks ever recorded. The CLUSIT 2021 Report [1] (Figure 1.1) offers a detailed overview of the most significant security incidents occurring globally up to 2020.

Despite the relevance of the problem, the vast theoretical and practical experience accumulated, the considerable risks deriving from the use of vulnerabilities both for businesses and for society and for the state, the problem itself is still far from being completely solved.

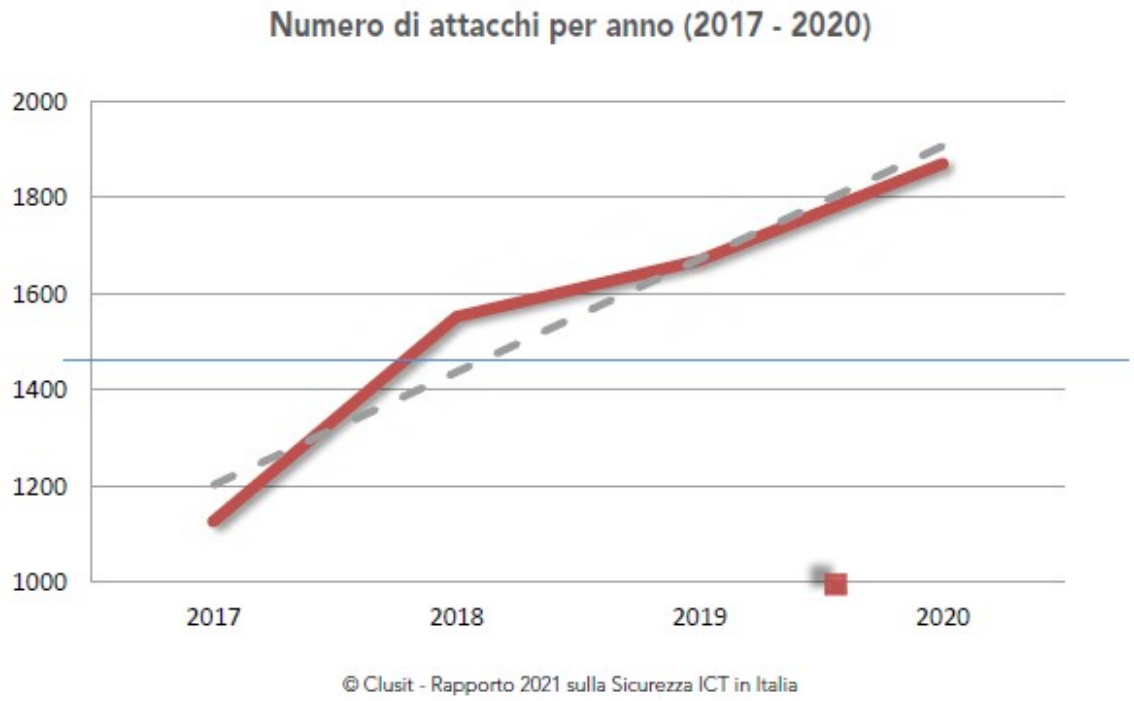


Figure 1.1: The CLUSIT 2021 Report on ICT security in Italy.

Much work has been done by scientists to address the problem of vulnerabilities in software by adopting different approaches and techniques. In particular, the use of machine learning models for vulnerability prediction would be very useful during software development. A developer could use such a model to predict whether any of the files he is working on are likely to have vulnerabilities. This way he could concentrate on testing these files and fixing the vulnerabilities present, to always release safe software.

Vulnerability prediction approaches involve training machine learning models on datasets that contain known vulnerabilities and using such models to classify new code as vulnerable or neutral.

Researchers are interested in whether vulnerability prediction models could actually be used in reality. To do this, they perform experiments to evaluate the performance of the models. However, as highlighted by Jimenez et al., [24] these experiments are performed in a research context, which is not always set up in a realistic way. In fact, in most of the experiments, all available data on vulnerabilities are considered, without taking into account the fact that these are discovered over time, and the presence of vulnerabilities in the code is not always known in reality. So there is a difference between the setting of the experiments carried out by researchers in the literature and the setting in which the models would be used in reality. This translates into a difference between the performances obtained by the models in the "ideal settings" of the experiments and the performances that the models would obtain in real contexts ("real setting") [24]. Our empirical study is interested in investigating the actual use of software vulnerability prediction models in reality. To do this, we perform experiments to evaluate the performance of the models. For our research we use the Walden et al.'s PHP vulnerability dataset from two popular open-source PHP applications, i.e., PHPMyAdmin and Moodle [13]. Our data is highly unbalanced. Imbalanced learning has drawn the attention of researchers in software vulnerability prediction. Imbalanced dataset can severely imprede the performance of machine learning algorithms and lead to conflicting and inconsistent results. To resolve the class imbalance problem, we applied the undersampling technique used by Walden et al. in [9] and Synthetic Minority Over-sampling TEchnique (SMOTE) employed by Jimenez et al. [24]. Undersampling technique consists in removing some samples of the majority class (neutral files) from the dataset, in order to match the number of samples in the minority class (vulnerable files). Oversampling technique consists in augmenting the number of samples in the minority class to match the size of the majority class.

We analyze the performance of vulnerability prediction models in the *real context* for *within-project* software vulnerability prediction, that is, we train and test our models on data related to the same software project. To perform our experiments we have implemented two

modes: `run_real` and `run_timespan`. In `run_real` mode we run our model to predict vulnerabilities in real context using Random Forest for supervised learning and KMeans clusters for unsupervised learning. Instead in the mode `run_timespan` we investigate how the effectiveness of machine learning models for predicting vulnerabilities changes in the real world context when using further data.

We shared our code publicly on github to give the possibility to reproduce the results of the experiments of the models for vulnerability prediction [38].

We have shown that vulnerability prediction models have really good performance when trained on data in the research context, i.e. they consider all available data on vulnerabilities. However, the performance is poor when considering data in the realistic context. Another finding that is in line with scientific studies is that the performance of a vulnerability prediction model is affected by data balancing approaches.

This thesis is structured as follows: Chapter 3 presents the related work. In Chapter 4, we explain our motivation, and Chapter 5 describes the metodology. The case study results and the threats to validity of our work are discussed in Chapter 6. We conclude the documentation and provide insights for future work in Chapter 7.

CHAPTER 2

Background

This chapter defines the terminology and concepts pertaining to the context of software vulnerability prediction using machine learning algorithms.

2.1 Cybersecurity after Covid-19

The Covid-19 pandemic has been an event of such great proportions that no aspect of our life has been spared. But given that the first security measure to counter the spread of the coronavirus is social distancing, one of the areas most affected has been the digital world. The main effects of Covid-19 on cyber security are an increase in phishing, malware and ransomware to organize scams related to the coronavirus, an increase in spam and fake news, an expansion of the security boundaries of the personal, corporate and public IT ecosystem. A hospital in the Czech Republic was forced to shut down all IT systems and cancel all scheduled operations for a ransomware attack [2]. CryCryptor ransomware masquerades as COVID-19 contact tracking app on Android device [3]. The University of California San Francisco (UCSF) was the target of ransomware attacks while working on the COVID-19 vaccine and was forced to pay $ 1.14 million to cybercriminals [4].

The boost of cyberthreats during the pandemic is confirmed by the 2021 CLUSIT Report, that shows a 12 % increase in the occurrence of cyber attacks globally. Of these attacks, 56 % were classified as having a "high" and / or "critical" level impact while 44 % were classified as "medium" severity. Notably, the last year has seen an increase in attacks that have actually been carried out through the exploitation of known vulnerabilities.

The Open Web Application a Security Project (OWASP) was created to help companies and organizations understand and improve the security of their applications and web services and thanks to the Top 10 OWASP ranking it was possible to focus attention on the most critical vulnerabilities that have caused the most damage to organizations worldwide in this decade:*Injection*, *Broken Authentication*, *Sensitive data exposure*, *XML External Entities (XXE)*, *Broken Access control*, *Security misconfigurations*, *Cross Site Scripting (XSS)*, *Insecure Deserialization*, *Using Components with known vulnerabilities*, *Insufficient logging and monitoring*.

Furthermore, according to the ISO / IEC 25010 [5] international standard on software quality, software security is one of the main quality characteristics of modern software products. In order to improve the security of the software produced, the software industries should ensure that their products are not bundled with critical vulnerabilities. Moreover,

tth European Union has released some specifications containing privacy information on the GDPR[6] (a European Union regulation on the processing of personal data and privacy, adopted on April 27, 2016), which software developers are also required to comply with.

2.2 Security Vulnerabilities

A security vulnerability is defined as *"a flaw in a software, firmware, hardware, or service component resulting from a weakness that can be exploited, causing a negative impact to the confidentiality, integrity, or availability of an impacted component or components."*[7]

A software vulnerability is one of the types of software **flaws** in the specification, development, or configuration of software that gives attackers the opportunity to bypass security restrictions on the target system, gain unauthorized access to confidential information, intentionally breach its integrity, and cause the failure of the system. With the growth of the software industry and the Internet, the number of attacks that exploit software vulnerabilities and the ease of implementation of these attacks are increasing. The CVE (Common Vulnerabilities and Exposures) and the NVD (National Vulnerability Database) provide comprehensive lists of all known vulnerabilities, so hackers can abuse this vulnerability to remotely trigger deadlocks, DoS, crashes, and even bypass security in the products with elevated privileges.

If these problems are not properly addressed, vulnerabilities can be exploited, often with disastrous consequences, as recent high-profile events such as the Heartbleed bug and the WannaCry network worm have shown.

There is no guarantee for the full release of a complex software product without vulnerabilities and those security flaws will inevitably be discovered after the software is released. At the same time, it is extremely important to improve software vulnerability discovery and analysis, measures for predicting and preventing vulnerabilities.

Software vendors can use software vulnerability prediction models as guidelines for predicting the severity and frequency of vulnerabilities and ensuring a certain level of software security.

2.3 Approaches to Vulnerability Prediction

In addition to traditional approaches such as static analysis, dynamic analysis and hybrid analysis, approaches using machine learning are popular in the analysis and detection of security vulnerabilities and are of growing interest [8]. Vulnerability prediction models built on machine learning methods use software attributes as input (for example, software metrics vs text-mining [9]) to distinguish between vulnerable and neutral components.

Advances in machine learning in recent years have made it possible to create a huge number of information security applications. Many large companies are investing billions of dollars in this area. For example, Google uses machine learning to analyze threats for Android mobile apps and to detect and remove malware from infected phones. In a similar way, Amazon acquired the harvest.ai startup and launched the Amazon Macie[10], a service that uses machine learning to detect, sort and classify malicious data stored in the cloud.

Machine learning methods for prediction can be divided into three approaches:

1. Supervised learning: The learning system builds a model based on labeled examples, where each example consists of a corresponding input and output value.

2. Unsupervised learning: in this case, there is no labeled training data available, the goal of the learning system is to build a model on the unlabeled data.

3. Reinforcement learning: building a learning system to achieve a specific goal, receive rewards and penalties through interaction with a dynamic environment.

CHAPTER 3

Related Work

In this chapter, we describe the related work in the topic of software vulnerability prediction using machine learning models.

Much work has been done by researchers to address the problem of vulnerability in software by adopting different approaches and techniques.

Neuhaus et al. [10] have focused on investigating the correlation between vulnerabilities and import declarations (imports or function calls) in the context of the Mozilla project.

Shin and Williams [31] investigated the use of execution complexity metrics along with code churn and developer metrics as indicators of software vulnerabilities. They performed two empirical case studies on open-source projects: the Red Hat Linux and the Mozilla Firefox web browser. The results indicate that 24 of the 28 metrics collected are discriminative of vulnerabilities for both projects.

Zimmerman et al. [32] carried out an empirical study to evaluate the code complexity, dependencies, efficacy of code churn and organizational measures to build a vulnerability prediction model in the context of Windows Vista. Their proposal obtained a low recall but good precision.

Walden et al. [9] performed experiments to compare the performance of predicting vulnerable software components based on software metrics versus text mining techniques. The authors first created a vulnerability dataset collected from three open source PHP web applications (PhpMyAdmin, Moodle, Drupa), containing 223 vulnerabilities. This dataset is offered to the research community.

Shepperd et al. [29] investigato sull'effetto dell'apprendimento sbilanciato e le sue interazioni con lo squilibrio dei dati, il tipo di classificatore, le metriche di input e il metodo di apprendimento sbilanciato. It concluded that unbalanced learning should only be considered for moderate or highly unbalanced data sets, and the indiscriminate application of imbalanced learning can be detrimental. In another study, Shepperd [34] shows that unsupervised models are comparablewith supervised models for both within-project and cross-project prediction. In general, unsupervised classifiers do not appear to perform any worse than supervised classifiers in their review. Among the different unsuoervised software vulnerability prediction learners, UnFuzzy andUnSC appear to have most potential.

Scandariato et al. [35] investigates the questions the questions of whether mobile applications developed for the Android platform are vulnerable or not, and how to predict which

classes of an Android application are vulnerable, by an alyzing the application of the Android Market and developing a vulnerability prediction model with high accuracy (over 80

Recently, Jimenez et al. [24] investigated how prediction models identify vulnerable components between software versions in the "real world". To establish realistic settings, prediction models are formed using the vulnerability information available at the time of release and are evaluated against the reported vulnerabilities for the entire period of time considered. Researchers showed significantly lower predictive effectiveness (mean MCC values of 0.08, 0.22 and 0.10 are achieved for Linux, OpenSSL andWiresark, respectively) when models are trained only on vulnerability tags that could realistically be available to the student at the time of model formation.

Vulnerabilities is the most specific case of defect. The defect prediction approaches studied in scientific literature are also applied to the prediction of software vulnerabilities[26].

The work of F.Zhang[18] compared the performance of unsupervised classifiers versus supervised classifiers using data from 26 projects from three publicly available datasets (i.e., AEEEM, NASA, and PROMISE) for cross-project setup and for within-project setting.

In Yang's[19] work supervised models are compared such as Linear Regression, Simple Logistic, Radial basis functions, Sequential Minimal Optimization, K-Nearest Neighbor, Propositional rule, Ripple down rules, Naive Bayes, J48, Logistic Model Tree, Random Forest, Bagging, Adaboost, Rotation Forest, Random Subspace and unsupervised especially K-Means. Experimental results, from open source software systems, show that many simple unsupervised models work better than supervised models in predicting defects. Liu et al. [20] performed an empirical study with the aim of investigating the predictive power of the unsupervised model of code abandonment in effort-conscious defect prediction.
A study by Meng Yan et al. [21] compares the effectiveness of unsupervised and supervised forecasting models for predicting defects at the file level. They conclude that unsupervised models do not perform statistically significantly better than supervised models in the within-project setting, and unsupervised models can function statistically significantly better than the supervised model in the cross-project setting.

The authors by Chen et al. [30] compared 48 different software vulnerability prediction

methods, of which 36 are supervised methods and 12 are unsupervised methods. For supervised methods, they consider 34 methods based on software metrics and two methods based on text extraction. Based on effort-conscious performance measures, they found that some recently proposed unsupervised and state-of-the-art methods may work better in both the scenario within-project and the scenario cross-projects.

Recent successes in natural language processing (NLP) have encouraged research into learning representation for source code, which relies on similar NLP methods for identifying vulnerable code [22]. Lately there are alternative approaches to predicting software vulnerability with modern methods of machine learning with methods of deep learning. *Deep learning methods could provide a new and potentially competitive alternative to existing techniques.*[23]

Matthieu Jimenez et al. [15] develop a prediction method using the machine translation framework encoder decoder that automatically learns the latent features of the code that are linked to the vulnerabilities. The authors validated their approach for three open source systems (Linux Kernel, OpenSSL and Wireshark) with historical vulnerabilities that were reported in the National Vulnerability Database (NVD). Their evaluation demonstrates good results in both recall and precision values.

Also Yanf et al[12] presented a study on applying deep-learning methods for just-in-time software defect prediction. Researchers proposed a Deeper approach that leverages deep learning techniques to predict defect-prone changes, n the context of six larg open source projects, i.e. Bugzilla, Columba, JDT, Platform, Mozilla and PostgreSQL.

CHAPTER 4

Motivation

In this chapter we present the motivations driving our empirical study.

Machine learning can be a determining factor in the planning and implementation of information systems capable of keeping up with cybercriminals. At the base of automatic learning there are a series of different algorithms that, starting from primitive notions, are able to make a specific decision rather than another or carry out actions learned over time, allowing the program to train on datasets, in order to be able to recognize, classify or predict something.

Vulnerability prediction approaches involve training models on datasets that contain known vulnerabilities and using a variety of machine learning techniques to classify code as vulnerable or neutral.

Machine learning models for vulnerability prediction would be very useful during software development. A developer could use such a model to predict whether any of the files he is working on are likely to have vulnerabilities. In this way he could concentrate on testing these files and fixing the vulnerabilities present, to always release safe software.

So the researchers are interested in whether vulnerability prediction models could actually be used in reality. To do this, they perform experiments to evaluate the performance of the models. However, these experiments are performed in a research context, which is not always set up in a realistic environment. In fact, in most of the experiments, all available data on vulnerabilities are considered, without taking into account the fact that these are discovered over time, and the presence of vulnerabilities in the code is not always known in reality. So there is a difference between the setting of the experiments carried out by researchers in the literature and the setting in which the models would be used in reality. This translates into a difference between the performances obtained by the models in the “ideal” settings of the experiments and the performances that the models would obtain in real contexts ("real" setting).

Recently, Jimenez et al.[24] claimed that *"the perfect labelling assumption that all vulnerabilities known from time t onwards are available at all times, even before t, is clearly unrealistic: a software engineer could only ever hope to train a predictive model on a partial ground truth that will include some degree of misclassification"*. They therefore investigated how prediction models would identify vulnerable components between software versions in the "real world". To establish

realistic settings, prediction models are formed using the vulnerability information available at the time of release and are evaluated against the reported vulnerabilities for the entire period of time considered. Researchers showed significantly lower predictive effectiveness (mean MCC values of 0.08, 0.22 and 0.10 are achieved for Linux, OpenSSL and Wiresark, respectively) when models are trained only on vulnerability tags that could realistically be available to the student at the time of model formation.

Since the presence of vulnerabilities in reality is very rare, in many cases it can happen that there are 0 instances of vulnerable files in the training set. Therefore a supervised learning model cannot be used.

In previous software vulnerability prediction studies, most of the proposed methods are based on supervised methods. But we cannot teach the supervised model to distinguish between vulnerable and neutral files, without giving it examples of vulnerable files. So we have to make the model learn all by itself, via an unsupervised learning algorithm. Unsupervised methods do not need any training data and can scale to large-scale projects. These characteristics can help to apply Software vulnerability prediction models to the industry. Perhaps the algorithm will be able to identify files in which there are vulnerabilities that we are not yet aware of.

Most empirical studies on the effectiveness of vulnerability prediction models have implicitly assumed that labeling information is available regardless of time constraints.

As unsupervised models identify vulnerable file without requiring the participation of labeled modules, it is meaningful to seek models that can achieve similar or better performance than supervised models for vulnerability prediction.

CHAPTER 5

Methodology

In this chapter, we present our empirical study. Fist, we introduce the goal, purpose, perspective and context of our work that we keep in mind during our investigation. After that, we discuss the settings of the proposed model and we describe the implementation of the experiments.

5.1 Research Goal

The main **goal** of this study is to analyze the performance of machine learning models in a real context, with the **purpose** of understanding if machine learning models are usable in a realistic software development context. The **perspective** is of both researchers and practitioners: the former are interested in whether vulnerability prediction models could actuallybe used in reality; the latter could use such a model to predict whether any of the files he isworking on are likely to have vulnerabilities. The **context** of the study consists of two datasets proposed by Walden et al.[9], i.e., PHPMyAdmin and Moodle.

Machine learning can be supervised or unsupervised, depending on how the dataset is constructed.

- **Supervised Learning:** Supervision means the presence of the solutions (labels) in the training dataset. A person (supervisor) gives the machine practical examples to the machine. Each example shows the input variables (x) and the correct result (y). The machine learns from the examples and builds a predictive model.

- **Unsupervised learning:** Conversely, unsupervised learning works by teaching the model to identify patterns on its own (therefore without a supervisor) from unlabeled data. This means that an input is provided, but not an output. One of the main unsupervised machine learning techniques is Clustering. The learning algorithm looks for regularities in the available data.

Based on the above goal, we defined our first research question:

> **RQ**$_1$:What is the power of supervised and unsupervised vulnerability prediction models in distinguishing between vulnerable and neutral files in a real-world environment using supervised and unsupervised learning models?

Since the presence of vulnerabilities in the real context is very rare, in many cases it can happen that there are no instances of vulnerable files in the training set. So we want to investigate what the percentage of vulnerable files must be out of the total number of files in a dataset to achieve satisfactory performance. This would allow developers to decide whether

machine learning models can be applied to the data sets in question.

> **RQ$_2$**: How effective are machine learning models to predict vulnerabilities in the real world context when using future data?

Imbalanced learning has drawn the attention of researchers in predicting software vulnerability. A dataset is imbalanced if the classes are not approximately equally represented. Imbalanced dataset can severely imprede the performance of machine learning algorithms and lead to conflicting and inconsistent results. Our data is highly unbalanced between the majority class (neutral files) and the minority class (vulnerable files). To resolve the class imbalance problem, we applied the subsampling technique used by Walden[9] and oversampling technique studied by Jimenez[24].

> **RQ$_3$**: Which data balancing technique leads to better performance within-project supervised and unsupervised learning?

5.2 Experiment Setup

5.2.1 Dataset

For our research we use the Walden et al.'s PHP vulnerability dataset which contains vulnerability data mined from two popular open-source PHP applications, i.e., PHPMyAdmin and Moodle[13].

Different vulnerability classes are represented in the dataset for each application:

- **Code Injection:** Attackers exploiting this vulnerability can modify arbitrary server-side variables, modify HTTP headers, or execute SQL, PHP or native code on the server
- **CSRF:** A type of attack on website visitors that exploits the shortcomings of the HTTP protocol, forcing them to perform some action on a vulnerable site on behalf of the victim.
- **XSS:** It allows you to enter or execute client-side code to collect, manipulate and redirect information, view and modify the data on the servers, alter the behavior of the software

- **Path Disclosure:** Path disclosure vulnerabilities allow an attacker to see the path to a root directory or file.
- **Authorization issues:** Once exploited, these vulnerabilities can lead to a compromise of system confidentiality, integrity or availability.
- **Other:** Miscellaneous vulnerabilities related to man-in-the-middle attacks, phishing or unspecified attack vectors.

Table 5.1 depicts the classes of vulnerabilities represented in the dataset for each application.

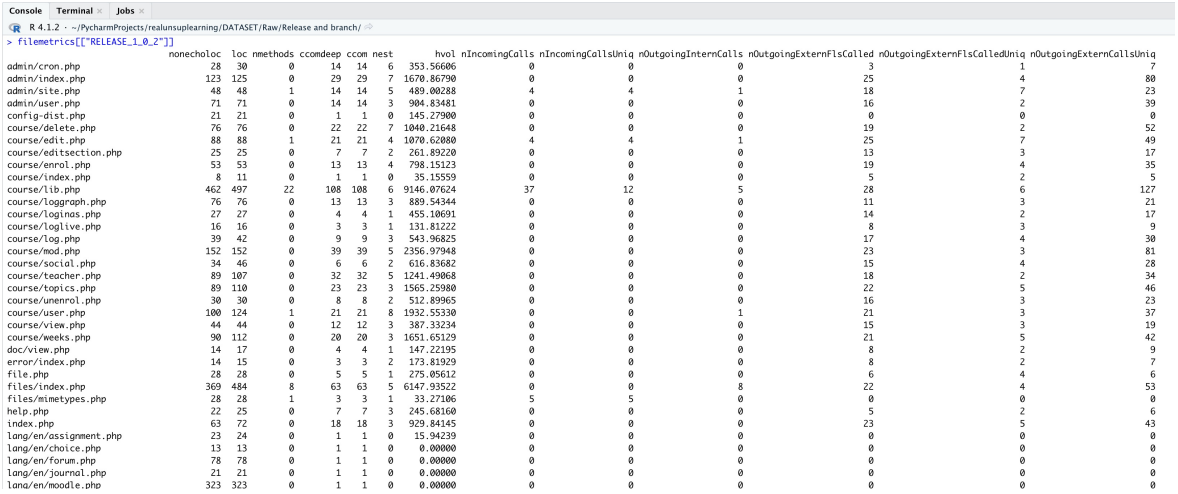
Table 5.1: Classes of vulnerabilities in each application.

	Moodle	PHPMyAdmin
Code Injection	7	10
CSRF	3	1
XSS	9	45
Path Disclosure	2	12
Authorization issues	28	6
Other	2	1

In each data set are files in *Rda* format that contain security vulnerability data and machine learning capabilities for a vulnerability prediction study. Files *filemetrics_phpmyadmin.Rda* e *filemetrics_moodle.Rda* contain a selection of computed, file-level source code metrics for each version of each application. Files *vuls_phpmyadmin.Rda* e *vuls_moodle.Rda* contain data frames describing each vulnerability analyzed for each application. Files *vulmovement_phpmyadmin.Rda* e *vulmovement_moodle.Rda* indicates the presence of vulnerabilities for each release. Files *mainbranch_phpmyadmin.Rda* e *mainbranch_moodle.Rda* contain metadata for each of the versions in particular: the version number of the version in question, the release date of the issue in question and he Git hash of the last commit that finalized the release in question.

The authors of the paper identified the following software metrics:

- *Lines of code:* The number of lines in the source file that contain PHP tokens. Blank lines and comments are not counted. Instead, when tokens span multiple lines they are



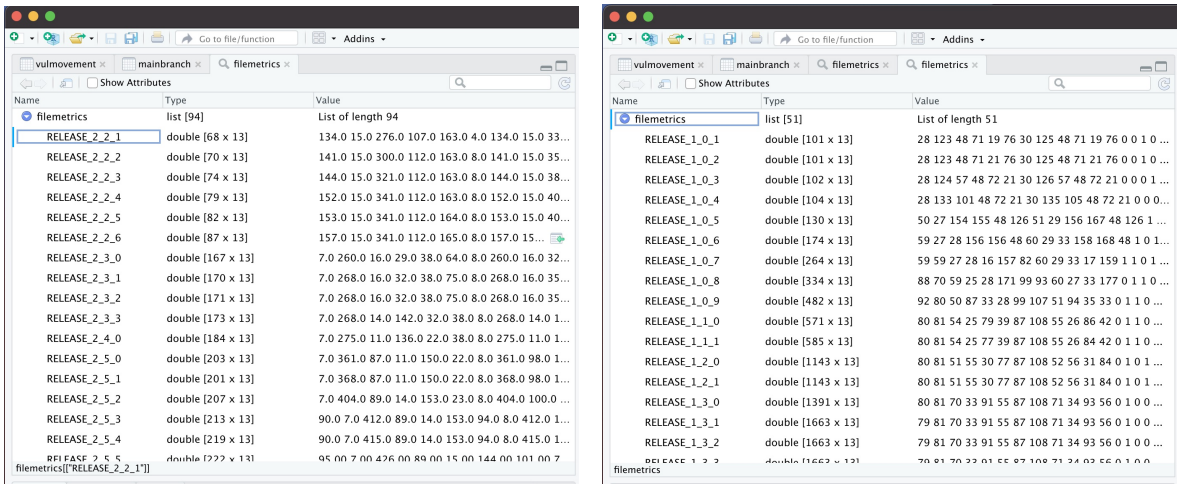
	nonecholoc	loc	nmethods	ccomdeep	ccom	nest	hvol	nIncomingCalls	nIncomingCallsUniq	nOutgoingInternCalls	nOutgoingExternFlsCalled	nOutgoingExternFlsCalledUniq	nOutgoingExternCallsUniq
admin/cron.php	28	30	0	14	14	6	353.56606	0	0	0	3	1	7
admin/index.php	123	125	0	29	29	7	1670.86790	0	0	0	25	4	80
admin/site.php	48	48	1	14	14	5	489.00288	4	4	1	18	7	23
admin/user.php	71	71	0	14	14	3	904.83481	0	0	0	16	2	39
config-dist.php	21	21	0	1	1	0	145.27900	0	0	0	0	0	0
course/delete.php	76	76	0	22	22	7	1040.21648	0	0	0	19	2	52
course/edit.php	88	88	1	21	21	4	1070.62080	4	4	1	25	7	49
course/editsection.php	25	25	0	7	7	2	261.89220	0	0	0	13	3	17
course/enrol.php	53	53	0	13	13	4	798.15123	0	0	0	19	4	35
course/index.php	8	11	0	1	1	0	35.15559	0	0	0	5	2	5
course/lib.php	462	497	22	108	108	6	9146.07624	37	12	5	28	6	127
course/loggraph.php	76	76	0	13	13	3	889.54344	0	0	0	11	3	21
course/loginas.php	27	27	0	4	4	1	455.10691	0	0	0	14	2	17
course/loglive.php	16	16	0	3	3	1	131.81222	0	0	0	8	3	9
course/log.php	39	42	0	9	9	3	543.96825	0	0	0	17	4	30
course/mod.php	152	152	0	39	39	5	2356.97948	0	0	0	23	3	81
course/social.php	34	46	0	6	6	2	616.83682	0	0	0	15	4	28
course/teacher.php	89	107	0	32	32	5	1241.49068	0	0	0	18	2	34
course/topics.php	89	110	0	23	23	3	1565.25980	0	0	0	22	5	46
course/unenrol.php	30	30	0	8	8	2	512.89965	0	0	0	16	3	23
course/user.php	100	124	1	21	21	8	1932.55330	0	0	1	21	3	37
course/view.php	44	44	0	12	12	3	387.33234	0	0	0	15	3	19
course/weeks.php	90	112	0	20	20	3	1651.65129	0	0	0	21	5	42
doc/view.php	14	17	0	4	4	1	147.22195	0	0	0	8	2	9
error/index.php	14	15	0	3	3	2	173.81929	0	0	0	8	2	7
file.php	28	28	0	5	5	1	275.05612	0	0	0	6	4	6
files/index.php	369	484	8	63	63	5	6147.93522	0	0	8	22	4	53
files/mimetypes.php	28	28	1	3	3	1	33.27106	5	5	0	0	0	0
help.php	22	25	0	7	7	3	245.68160	0	0	0	5	2	6
index.php	63	72	0	18	18	3	929.84145	0	0	0	23	5	43
lang/en/assignment.php	23	24	0	1	1	0	15.94239	0	0	0	0	0	0
lang/en/choice.php	13	13	0	1	1	0	0.00000	0	0	0	0	0	0
lang/en/forum.php	78	78	0	1	1	0	0.00000	0	0	0	0	0	0
lang/en/journal.php	21	21	0	1	1	0	0.00000	0	0	0	0	0	0
lang/en/moodle.php	323	323	0	1	1	0	0.00000	0	0	0	0	0	0

(a) PHPMyAdmin

Figure 5.1: Software metrics in the filemetrics Rda file, as visualized in the RStudio environment.

counted as one line.

- *Lines of code (non-HTML):* Same as lines of code, except HTML content embedded in PHP files. Content outside of php start/end tags is not taken into account.
- *Number of functions:* Number of function and method definitions in a file.
- *Cyclomatic complexity:* The size of the control block diagram after the linear chains of nodes is reduced to one. Calculated by adding one to the number of loop statements and solutions in the file.
- *Maximum nesting complexity:* Maximum depth of nesting of loops and control structures in the file.
- *Halstead's volume:* Volume estimate, which is calculated by the formula ((N1 + N2) log n1 + n2) using the number of unique operators (n1) and operands (n2) and the total number of operators (N1) and operands (N2) in the file, where operators are PHP method names and operators, and operands are parameter and variable names.
- *Total external calls:* The number of times a statement in a measured file calls a function or method defined in another file.



(a) PHPMyAdmin **(b)** Moodle

Figure 5.2: Structure of the filemetrics Rda file, as visualized in the RStudio environment.

- *Fan-in:* The number of files (excluding the measured file) that contain statements that call the function or method defined in the measured file.
- *Fan-out:* The number of files (excluding the measured file) that contain functions or methods that are called by statements in the measured file.
- *Internal functions or methods called:* The number of methods or functions found in the measured file that are called at least once by a statement in the same file.
- *External functions or methods called:* The number of methods or functions found in other files that are called at least once by a statement in the measured file. If a method or function call target is not defined, all possible call targets are considered to have been called.
- *External calls to functions or methods:* The number of files (excluding the measured file) that call a particular method or function defined in the measured file, summed over all methods and functions in the measured file.

Software metrics computed for all files of all releases of the considered application are available in a single `Rda` file.

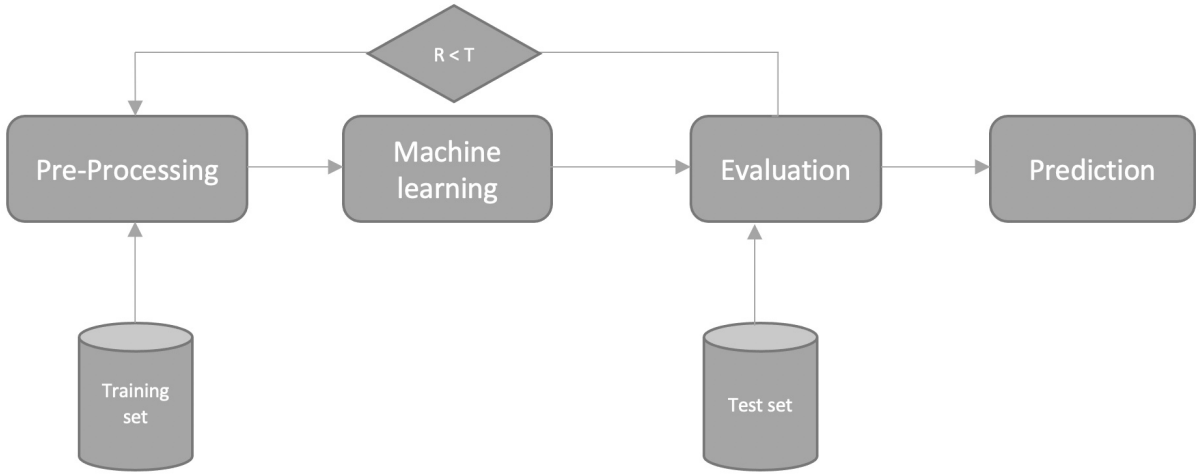


Figure 5.3: Phases of a Machine Learning pipeline.

5.2.2 Process

The stages of machine learning are as follows:

- **Pre-Processing:** It is the phase of analyzing the data in the learning dataset to optimize the performance of the algorithm.
- **Learning:** It is the phase in which the algorithm automatically learns from the data present in the learning dataset, called training set to elaborate a forecast model.
- **Evaluation:** It is the phase of evaluating the forecasting model created by the algorithm in terms of accuracy. To evaluate the predictive quality, a percentage T of correct answers is set as a reference parameter. The predictive model is tested with a test dataset, called a test set, other than the learning dataset, and is analyzed the results. If the percentage of correct answers R exceeds the threshold T, the predictive model is promoted. The answers from the machine are quite accurate. The margin of error of the answers is acceptable. If the percentage of correct answers R does not exceed the threshold T, the process resumes from the preprocessing phase because the quality of the prediction is not sufficiently effective.
- **Prediction:** It is the final application phase, the one in which the predictive model is

used with real data to solve practical problems, either by the machine itself or by the end users.

Our work is to analyze the difference in performance of vulnerability prediction models in the *real context* for *within-project* vulnerability prediction. Our study is based on software metrics techniques proposed by Walden for the PHPMyAdmin and Moodle datasets[9]. We ran the our model to predict vulnerabilities on real context using Random Forest for supervised learning and cluster KMeans for unsupervised learning. We conducted the experiments using SciKit Learn for Python [14]. Sklearn is a python language module with functions useful for machine learning.

5.2.3 Performance Measure

We base the performance evaluation on the confusion matrix. This is a well-known and well-established evaluation methodology, adopted by the absolute majority of previous studies[8][15][16]. The confusion matrix is constructed from the following elements:

- True negative (**TN**): The number of non-vulnerable items that the classifier predicts as non-vulnerable
- False positive (**FP**): The number of non-vulnerable items that the classifier predicts as vulnerable
- False negative (**FN**): The number of vulnerable items that the classifier predicts as non-vulnerable
- True positive (**TP**): The number of vulnerable items that the categorizer predicts as vulnerable

The prediction performance measurements are then computed using the confusion matrix elements, as follows:

- *Precision:* Represents the probability that the vulnerability prediction model declarations of vulnerable code are accurate. It is function of the True Positive (TP) rate and False

Positive (FP) rate of vulnerable source code files. Precision is calculated as seen:

$$Precision = \frac{\text{TP}}{\text{TP + FP}}$$

- *Accuracy* The relationship of the number of correct predictions to the total number of predictions

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

- *Recall:* Represents the probability that the VPM finds a source code file that contains at least one vulnerability. It is a function of the True Positive (TP) and True Negative (TN) rate of vulnerable source code files. Recall is calculated as seen:

$$Recall = \frac{TP}{TP + FN}$$

- *F1-Score:* Represents the geometric mean of precision and recall. Higher indicates better overall accuracy, assuming that precision and recall are weighted equally. F-Score is calculated as seen

$$F1 - score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}$$

- *Inspection Rate* Percentage of files that need to be reviewed so that true positive results can be found as determined by the model. This indicator can take values from 0 to 1 and is defined as follows:

$$InspectionRate = \frac{TP + FP}{TP + TN + FP + FN}$$

- *MCC* Matthews correlation coefficient (MCC) [26], a reliable indicator of the quality of predictive models [7, 36]. This metric takes into account true and false positives and negatives and is generally considered a balanced measure. MSS is defined as:

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

Thus, MCC returns a coefficient between 1 and -1, where MCC = 1 indicates an accurate prediction, MCC = -1 indicates a perfect inverse prediction (i.e., complete mismatch between prediction and observation), and MCC = 0 indicates that that the performance of the classifier is equivalent to guessing at random.

For example, one model may have high recall but low precision, indicating that the model has a high false positive rate. Another model may have high precision but low recall, indicating that the model rarely gives false positives but misses many vulnerabilities.

5.3 Data Preprocessing

The dataset provides the files in Rda format. Python can work safely with Rda files, using the pandas library and dataframe structures. However, Python cannot work with the filemetrics_phpmyadmin.Rda and filementrics_mooddle.Rda files which contain the metrics. This problem fixes an R script to turn these Rda format files into CSV files, which was done with RStudio.

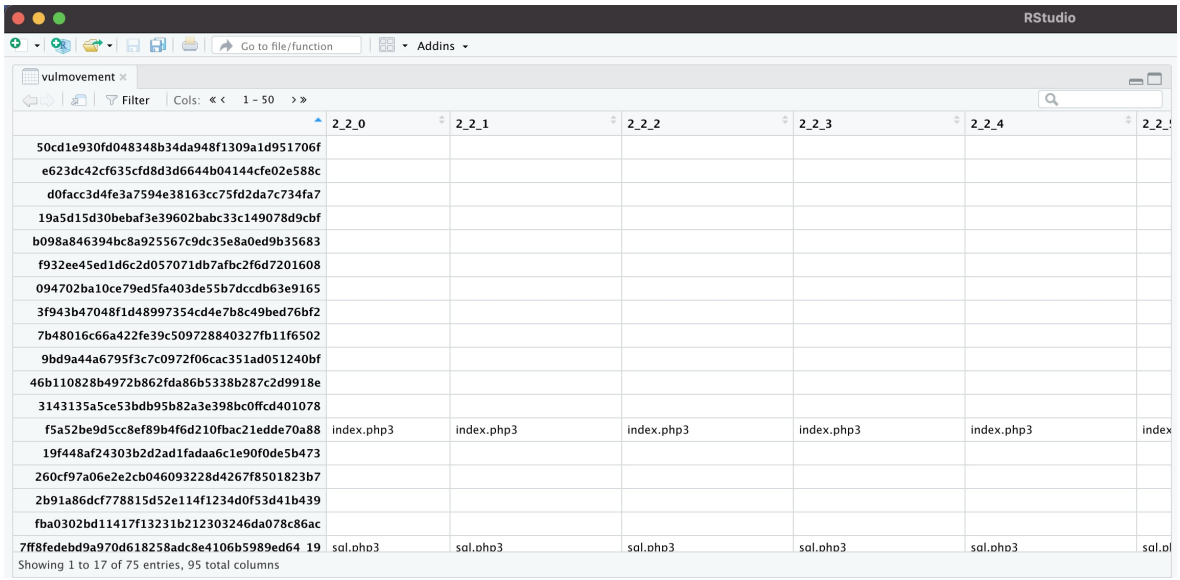
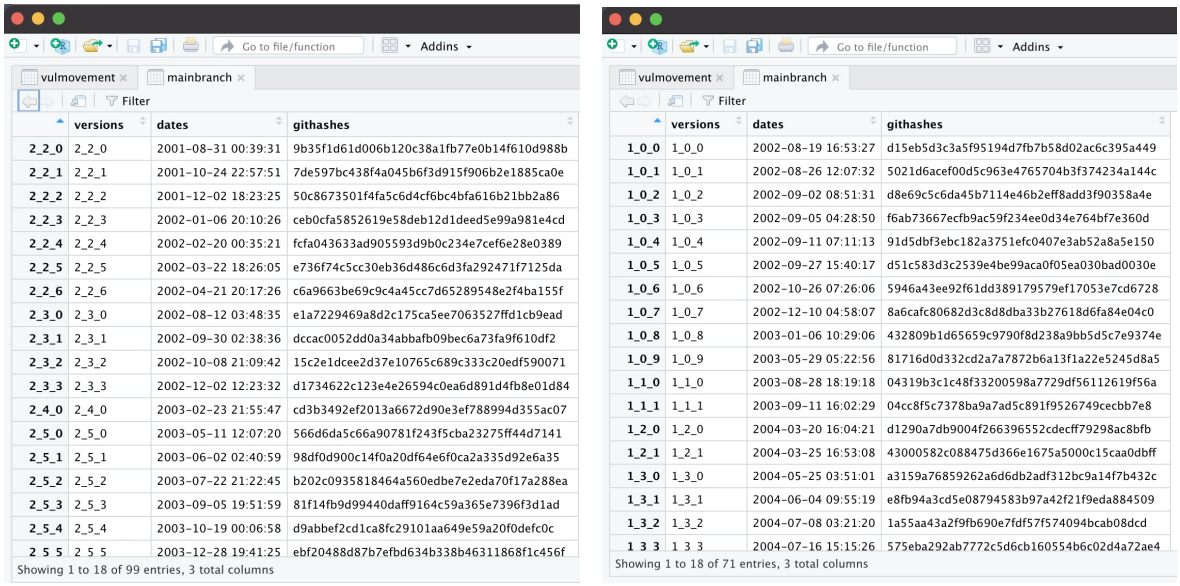


Figure 5.4: Structure of the vulmovement Rda file for PHPMyAdmin dataset, as visualized in the RStudio environment.

To carry out experimentswith the realistic setting, we must use a particular approach, based on the construction of an ad hoc dataset foreach experiment.

The procedure for building the dataset in the PHPMyAdmin context is described below. The procedure for constructing the dataset in the Moodle context is analogous. For each release R we get the release date D from the `mainbranch_phpmyadmin.rda` file for each

release X prior to R. After that we examine the `vulmovement_phpmyadmin.rda` file and select all the pairs (introhash, filename) of the release X that identify the files affected by vulnerabilities in that release. For each pair (introhash, filename) we retrieve information from the `vuls_phpmyadmin.rda` file and select the line relating to that introhash. We memorize the cveid indicated in that line. We need to make API calls to CIRCL[25] to get details on a specific vulnerability identified by the CVE. In these details, the P date of publication of the vulnerability will be provided, i.e. the date on which it was made known to the public. We will consider this as the vulnerability discovery date. If P is earlier than D, then that file must be labeled as vulnerable (in the implementation marked as `IsRealVulnerable`). Finally we keep track of the number of files tagged as vulnerable in the search context (in the implementation marked as `IsIdealVulnerable`) on the total number of files paying attention to duplicates.



vulmovement | mainbranch

	versions	dates	githashes
2_2_0	2_2_0	2001-08-31 00:39:31	9b35f1d61d006b120c38a1fb77e0b14f610d988b
2_2_1	2_2_1	2001-10-24 22:57:51	7de597bc438f4a045b6f3d915f906b2e1885ca0e
2_2_2	2_2_2	2001-12-02 18:23:25	50c8673501f4fa5c6d4cf6bc4bfa616b21bb2a86
2_2_3	2_2_3	2002-01-06 20:10:26	ceb0cfa5852619e58deb12d1deed5e99a981e4cd
2_2_4	2_2_4	2002-02-20 00:35:21	fcfa043633ad905593d9b0c234e7cef6e28e0389
2_2_5	2_2_5	2002-03-22 18:26:05	e736f74c5cc30eb36d486c6d3fa292471f7125da
2_2_6	2_2_6	2002-04-21 20:17:26	c6a9663be69c9c4a45cc7d65289548e2f4ba155f
2_3_0	2_3_0	2002-08-12 03:48:35	e1a7229469a8d2c175ca5ee7063527ffd1cb9ead
2_3_1	2_3_1	2002-09-30 02:38:36	dccac0052dd0a34abbafb09bec6a73fa9f610df2
2_3_2	2_3_2	2002-10-08 21:09:42	15c2e1dcee2d37e10765c689c333c20edf590071
2_3_3	2_3_3	2002-12-02 12:23:32	d1734622c123e4e26594c0ea6d891d4fb8e01d84
2_4_0	2_4_0	2003-02-23 21:55:47	cd3b3492ef2013a6672d90e3ef788994d355ac07
2_5_0	2_5_0	2003-05-11 12:07:20	566d6da5c66a90781f243f5cba23275ff44d7141
2_5_1	2_5_1	2003-06-02 02:40:59	98df0d900c14f0a20df64e6f0ca2a335d92e6a35
2_5_2	2_5_2	2003-07-22 21:22:45	b202c0935818464a560edbe7e2eda70f17a288ea
2_5_3	2_5_3	2003-09-05 19:51:59	81f14fb9d99440daff9164c59a365e7396f3d1ad
2_5_4	2_5_4	2003-10-19 00:06:58	d9abbef2cd1ca8fc29101aa649e59a20f0defc0c
2_5_5	2_5_5	2003-12-28 19:41:25	ebf20488d87b7efbd634b338b46311868f1c456f

Showing 1 to 18 of 99 entries, 3 total columns

(a) PHPMyAdmin

vulmovement | mainbranch

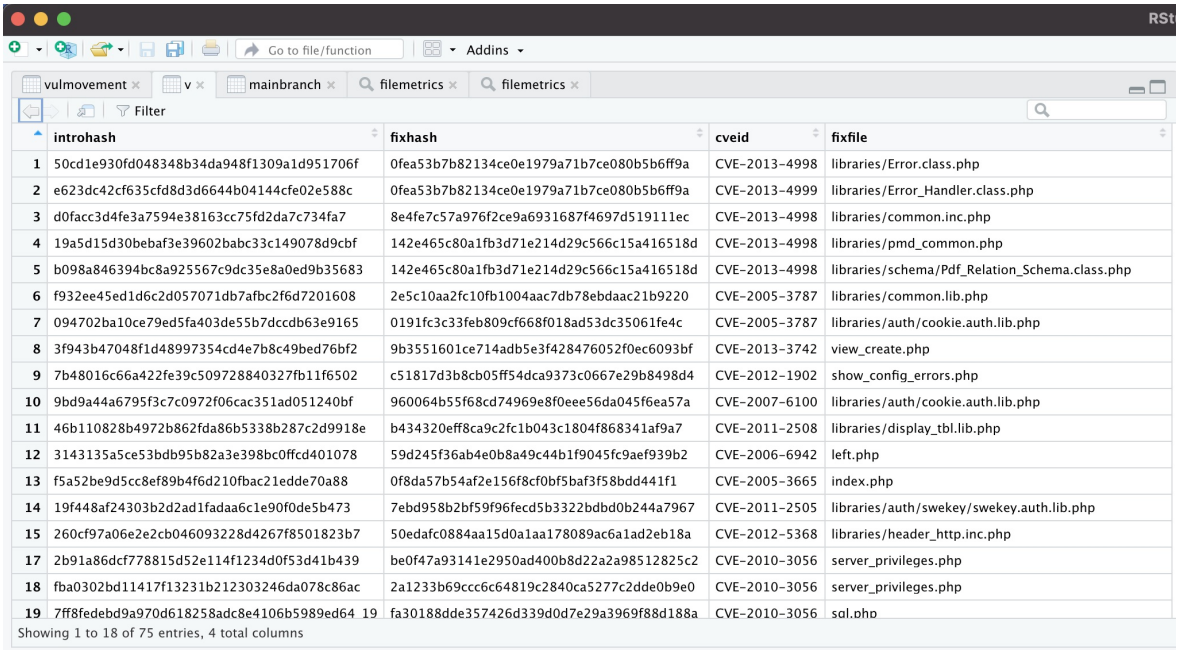
	versions	dates	githashes
1_0_0	1_0_0	2002-08-19 16:53:27	d15eb5d3c3a5f95194d7fb7b58d02ac6c395a449
1_0_1	1_0_1	2002-08-26 12:07:32	5021d6acef00d5c963e4765704b3f374234a144c
1_0_2	1_0_2	2002-09-02 08:51:31	d8e69c5c6da45b7114e46b2eff8add3f90358a4e
1_0_3	1_0_3	2002-09-05 04:28:50	f6ab73667ecfb9ac59f234ee0d34e764bf7e360d
1_0_4	1_0_4	2002-09-11 07:11:13	91d5dbf3ebc182a3751efc0407e3ab52a8a5e150
1_0_5	1_0_5	2002-09-27 15:40:17	d51c583d3c2539e4be99aca0f05ea030bad0030e
1_0_6	1_0_6	2002-10-26 07:26:06	5946a43ee92f61dd389179579ef17053e7cd6728
1_0_7	1_0_7	2002-12-10 04:58:07	8a6cafc80682d3c8d8dba33b27618d6fa84e04c0
1_0_8	1_0_8	2003-01-06 10:29:06	432809b1d65659c9790f8d238a9bb5d5c7e9374e
1_0_9	1_0_9	2003-05-29 05:22:56	81716d0d332cd2a7a7872b6a13f1a22e5245d8a5
1_1_0	1_1_0	2003-08-28 18:19:18	04319b3c1c48f33200598a7729df56112619f56a
1_1_1	1_1_1	2003-09-11 16:02:29	04cc8f5c7378ba9a7ad5c891f9526749cecbb7e8
1_2_0	1_2_0	2004-03-20 16:04:21	d1290a7db9004f266396552cdecff79298ac8bfb
1_2_1	1_2_1	2004-03-25 16:53:08	43000582c088475d366e1675a5000c15caa0dbff
1_3_0	1_3_0	2004-05-25 03:51:01	a3159a76859262a6d6db2adf312bc9a14f7b432c
1_3_1	1_3_1	2004-06-04 09:55:19	e8fb94a3cd5e08794583b97a42f21f9eda884509
1_3_2	1_3_2	2004-07-08 03:21:20	1a55aa43a2f9fb690e7fdf57f574094bcab08dcd
1_3_3	1_3_3	2004-07-16 15:15:26	575eba292ab7772c5d6cb160554b6c02d4a72ae4

Showing 1 to 18 of 71 entries, 3 total columns

(b) Moodle

Figure 5.5: Structure of the mainbranch Rda file, as visualized in the RStudio environment.

In this way, we get an overview of the situation: for each release to be considered as a test set, we know how many instances of vulnerable files there would be in the training set. In this way we can highlight the problem described above and provide a solid basis for the reasons for the thesis.



vulmovement | v | mainbranch | filemetrics | filemetrics

Filter

	introhash	fixhash	cveid	fixfile
1	50cd1e930fd048348b34da948f1309a1d951706f	0fea53b7b82134ce0e1979a71b7ce080b5b6ff9a	CVE-2013-4998	libraries/Error.class.php
2	e623dc42cf635cfd8d3d6644b04144cfe02e588c	0fea53b7b82134ce0e1979a71b7ce080b5b6ff9a	CVE-2013-4999	libraries/Error_Handler.class.php
3	d0facc3d4fe3a7594e38163cc75fd2da7c734fa7	8e4fe7c57a976f2ce9a6931687f4697d519111ec	CVE-2013-4998	libraries/common.inc.php
4	19a5d15d30bebaf3e39602babc33c149078d9cbf	142e465c80a1fb3d71e214d29c566c15a416518d	CVE-2013-4998	libraries/pmd_common.php
5	b098a846394bc8a925567c9dc35e8a0ed9b35683	142e465c80a1fb3d71e214d29c566c15a416518d	CVE-2013-4998	libraries/schema/Pdf_Relation_Schema.class.php
6	f932ee45ed1d6c2d057071db7afbc2f6d7201608	2e5c10aa2fc10fb1004aac7db78ebdaac21b9220	CVE-2005-3787	libraries/common.lib.php
7	094702ba10ce79ed5fa403de55b7dccdb63e9165	0191fc3c33feb809cf668f018ad53dc35061fe4c	CVE-2005-3787	libraries/auth/cookie.auth.lib.php
8	3f943b47048f1d48997354cd4e7b8c49bed76bf2	9b3551601ce714adb5e3f428476052f0ec6093bf	CVE-2013-3742	view_create.php
9	7b48016c66a422fe39c5097288403 27fb11f6502	c51817d3b8cb05ff54dca9373c0667e29b8498d4	CVE-2012-1902	show_config_errors.php
10	9bd9a44a6795f3c7c0972f06cac351ad051240bf	960064b55f68cd74969e8f0eee56da045f6ea57a	CVE-2007-6100	libraries/auth/cookie.auth.lib.php
11	46b110828b4972b862fda86b5338b287c2d9918e	b434320eff8ca9c2fc1b043c1804f868341af9a7	CVE-2011-2508	libraries/display_tbl.lib.php
12	3143135a5ce53bdb95b82a3e398bc0ffcd401078	59d245f36ab4e0b8a49c44b1f9045fc9aef939b2	CVE-2006-6942	left.php
13	f5a52be9d5cc8ef89b4f6d210fbac21edde70a88	0f8da57b54af2e156f8cf0bf5baf3f58bdd441f1	CVE-2005-3665	index.php
14	19f448af24303b2d2ad1fadaa6c1e90f0de5b473	7ebd958b2bf59f96fecd5b3322bdbd0b244a7967	CVE-2011-2505	libraries/auth/swekey/swekey.auth.lib.php
15	260cf97a06e2e2cb046093228d4267f8501823b7	50edafc0884aa15d0a1aa178089ac6a1ad2eb18a	CVE-2012-5368	libraries/header_http.inc.php
17	2b91a86dcf778815d52e114f1234d0f53d41b439	be0f47a93141e2950ad400b8d22a2a98512825c2	CVE-2010-3056	server_privileges.php
18	fba0302bd11417f13231b212303246da078c86ac	2a1233b69ccc6c64819c2840ca5277c2dde0b9e0	CVE-2010-3056	server_privileges.php
19	7ff8fedebd9a970d618258adc8e4106b5989ed64_19	fa30188dde357426d339d0d7e29a3969f88d188a	CVE-2010-3056	sql.php

Showing 1 to 18 of 75 entries, 4 total columns

Figure 5.6: Structure of the vuls Rda file for PHPMyAdmin dataset, as visualized in the RStudio environment.

Built dataset contains Ideal-World-Labeling, Real-World-Labeling and Total Number of Files for each version.

5.4 Experiments Implementation

To study the methods of machine learning for sotware vulnerability prediction we have implemented the code for executing supervised and unsupervised methods with different settings. The package includes three main modules:

- **Data Preprocessing module** has been described in the paragraph 5.3. The code that implements this module is found in the dataset.py script.
- **Selection of characteristics module**: The Python script `classes.py` contains the declarations of all the variables needed to conduct our experiments. Instead the experiments.py script contains the code to give the user the possibility to set the experiments to run. A brief summary of all the variables to be set are presented in the table 5.2.

Table 5.2: Summary of all the variables to be set

Variables	Instances
Dataset	PHPMyAdmin and Moodle
Run Mode	run_real and run_timespan
Modelling Approach	Software Metrics
Data Balancing Technique	none, undersampling and oversampling
Validation Method	real_labelling
Project Setting	Within-project
Learning Approach	Supervised (**Random Forest**) and Unsupervised(**KMeans**)

- **Grouping and labeling module** The Python script `implementation.py` contains functions that take features and labels as input and return data with the selected metrics according to different implementations. The two datasets are implemented in this file: training set, which contains the realistic data and test set, which contains the experimental data. Realistic data is historical data on vulnerabilities labeled as vulnerable and neutral based on information available at the time of release (i.e. vulnerabilities reported prior to the particular release). Experimental data are historical data labeled as vulnerable and neutral based on knowledge of the vulnerabilities reported over the entire time period considered.

  There are two run mode for this phase: In **`Run_real Mode`** for each R release, for which we researchers have the entire vulnerability history available, we consider all X_1 - X_n releases prior to R as training sets and train the learning model on all file metrics of X_1 - X_n releases to in order to allow the interrogation and analysis of the results of the experiments.

  The implementation of **`Run_timespan`** is similar to the previous implementation with the addition of the `timespan` variable. The `timespan` variable is the amount of time

between two moments, in our case it is 3 months. So for each version more training sets are considered (with different number of vulnerabilities discovered in these). These sets are defined precisely with the timespan variable, starting from the date of the first relsease and going forward in time until the date of the last relsease. In this way we can analyze how the model behaves with respect to the percentage of vulnerabilities present in the training set.

The same script contains the implementations related to the performance calculation for the vulnerability prediction models.

The performance indicators produced by each run of the experiment are saved in a `CSV` file together with all information related to the experiment setup, in order to allow the interrogation and analysis of the results of the experiments.

CHAPTER 6

Analysis of the results

In this chapter, we analyze the results of the experiments and discuss our answers to the research questions that we formulated in chapter 5.

6.1 Case Study Results

To evaluate our models for software vulnerability prediction, analyze performance, and answer research questions, we run a total of 4,009 experiments: 870 experiments for `run_real` mode and 3,139 experiments for `run_timespan` mode. Tables 6.1 and 6.2 summarize the experiments.

Table 6.1: Summary of the run_real experiments executions.

Dataset	Learning Approach	Data Balancing Technique	Number of Experiments
PHPMyAdmin	Supervised (*Random Forest*)	none	94
PHPMyAdmin	Supervised (*Random Forest*)	undersampling	94
PHPMyAdmin	Supervised (*Random Forest*)	oversampling	94
PHPMyAdmin	Unsupervised (*KMeans*)	none	94
PHPMyAdmin	Unsupervised (*KMeans*)	undersampling	94
PHPMyAdmin	Unsupervised (*KMeans*)	oversampling	94
Moodle	Supervised (*Random Forest*)	none	51
Moodle	Supervised (*Random Forest*)	undersampling	51
Moodle	Supervised (*Random Forest*)	oversampling	51
Moodle	Unsupervised (*KMeans*)	none	51
Moodle	Unsupervised (*KMeans*)	undersampling	51
Moodle	Unsupervised (*KMeans*)	oversampling	51
		Total	870

The choice of performances is based on the observation of the evaluation methodologies in the works examined. Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8 and 6.9, 6.10, 6.11, 6.12,

Table 6.2: Summary of the run_timespan experiments executions.

Dataset	Timestamp	Number of Experiments
PHPMyAdmin	$\Delta = 3$	1975
Moodle	$\Delta = 3$	1164
	Total	3139

6.13, 6.14, 6.15, 6.16 show the results of the performances obtained from the execution of the experiments. The figures show the performance results obtained from the execution the experiments for PHPMyAmind and Moodle dataset in `run_real` mode and PHPMyAmind and Moodle dataset in `run_timespan` mode respectively.

We can answer the research questions by analyzing the boxplot graphs.

RQ$_1$: What is the power of vulnerability prediction models in distinguishing between vulnerable and neutral files in a real-world environment using supervised and unsupervised model?

RQ$_3$: Which data balancing technique leads to better performance within-project supervised and unsupervised learning?

In the real world, unsupervised learning performs slightly better than supervised learning. This is explained because the presence of vulnerabilities is very rare in the real world, in many cases it can happen that there are 0 instances of vulnerable files in the training set. So we can't teach the supervised model to distinguish between vulnerable and neutral files, without giving it examples of vulnerable files.

We observe a relatively low performance for all approaches and datasets. Our precision values are 0.3 for PHPMyAdmin and 0.1 for Moodle and 0.2 of recall on average for all datasets. The literature [24][26] suggests that the reasonable values of precision and recall are $\geq$ 0.7. This indicates the high rate of false positives and low rate of correctly recog-

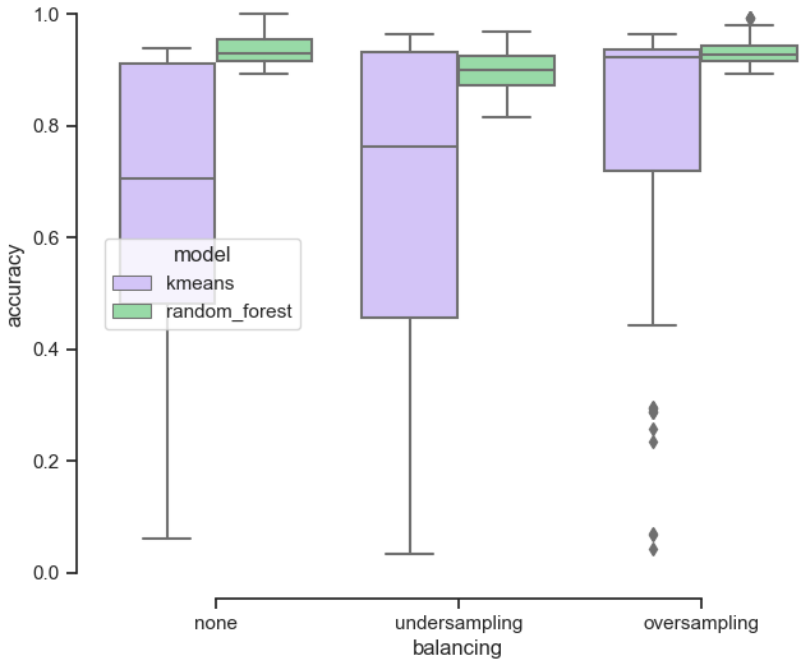


Figure 6.1: Accuracy on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

nized positives. High accuracy values for both PHPMyAdmin and Moodle indicate that the vulnerabilities share the same characteristics over time, as they are captured by the models.

After conducting the experiments and analyzing the results we can answer the research question $\mathbf{RQ}_1$. The performance of models with both supervised and unsupervised learning is poor. So these models still cannot be used in the real context for predicting vulnerabilities in software.

We applied classification algorithm Random Forest and clustering algorithm Kmeans in conjunction with two data balancing techniques on two public datasets, namely Moodle and PHPMyAdmin.

The figures 6.1, 6.2, 6.3, 6.4 and 6.5, 6.6, 6.7, 6.8 show that the performance of a vulnerability prediction model is affected by data balancing approaches. Specifically Random Forest for both the PhpMyAdmin dataset and Moodle performs better with undersampling data balancing tecnique. Instead KMeans in some ways for both the PhpMyAdmin dataset and

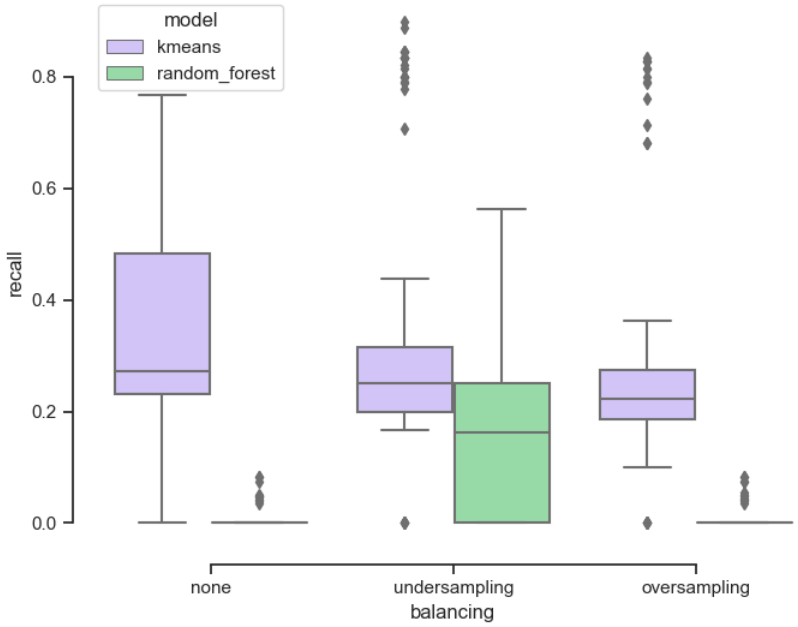


Figure 6.2: Recall on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

Moodle has better performance with the undersampling or oversampling data balancing technique. In general, responding to $\mathbf{RQ}_3$, we can say that the balancing techniques improve the performance obtained from the models for predicting the software vulnerability.

$\mathbf{RQ}_2$: How effective are machine learning models to predict vulnerabilities in the real world context when using future data?

To answer $\mathbf{RQ}_2$ we have chosen to analyze only the behavior of the model with Random Forest.The choice is related to the fact that KMeans algorithm in the training phase does not use information on the vulnerability. From the figures 6.9, 6.10, 6.11, 6.12 and 6.13, 6.14, 6.15, 6.16 we can see how the vulnerability prediction models behave based on the proportion of vulnerabilities present in the training set. In particular for the PHPMyAdmin database the performances begin to be reasonable with vulnerability = 6%, while for the Moodle dataset the performances begin to be reasonable with vulnerability = 1.9%

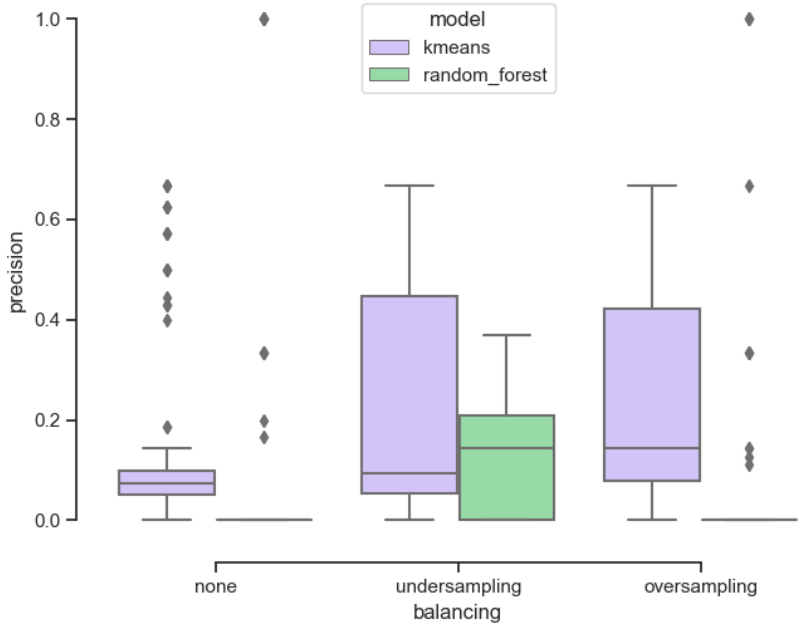


Figure 6.3: Precision on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

6.2 Threats to Validity

In this section, we identify factors that may threaten the validity of our results and present the actions we have taken to mitigate the risk. Wohlin et al.[36] suggested a validity analysis based on four types. These types are external validity, internal validity, construct validity, and conclusion validity.

External Validity: Threats to external validity are about whether the observed experimental results can be generalized to other project. To mitigate this threat, we chose datasets that have been extensively used in scientific literature for vulnerability prediction by many researchers. Our experiments were conducted using publicly available reference data from 2 projects, in particular PHPMyAdmin and Moodle proposed by Walden et al.[9]. This allows comparison with the results of other experiments. Perhaps unsupervised models actually identify vulnerabilities that are not yet known (vulnerabilities that will be discovered in the future), so we too are wrong in saying that their performance is low, because perhaps we do not even know the truth about the vulnerabilities present. But as these datasets have been widely used and validated, we don't believe this is a threat. An external validity threat is that

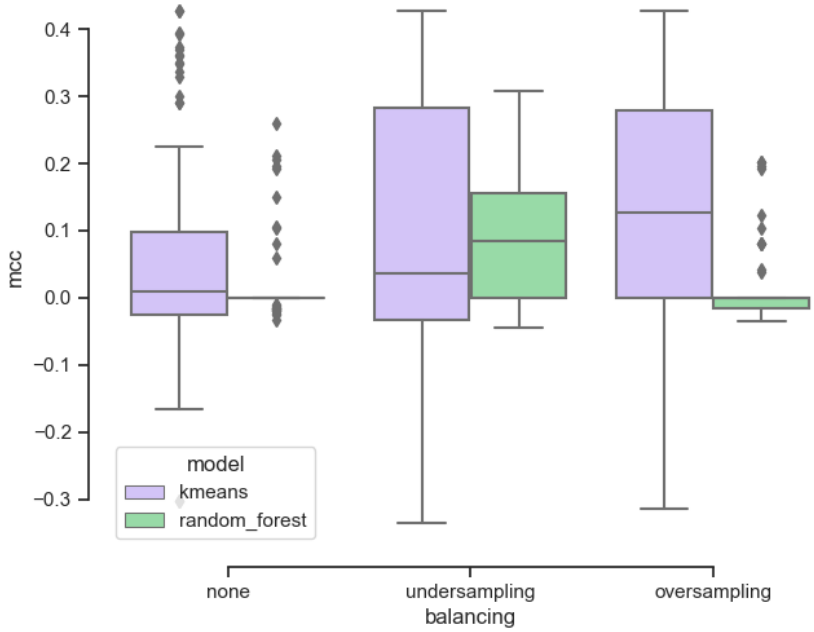


Figure 6.4: MCC on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

all of these projects were developed with PHP language and we do not consider the projects developed with other languages, such as C, C++, python, Java or to other types of software, such as desktop or mobile applications. Instead, for the creation of the ad hoc dataset we used the CVE site to obtain information on vulnerabilities. Hence, we do not claim that our results can be generalized to all systems, as the systems under study may not be representative of systems in general. To mitigate such threats, various data with different programming languages could be used for future experiments.

Internal Validity: Threats to internal validity are mainly concerned with the uncontrolled internal factors that might have infuence on the experimental results. For the implementation of our framework we used the third-party library implementation, including scikit-learn and R, to avoid potential errors in the implementation process, which is beneficial for mitigating the internal validity threat. In addition, a PhD student participated in this work in controlling the source code to minimize this threat. Furthermore, we have made available our experiment results and the source code of the framework [38]. This allows other researchers to replicate the results of the experiments. Even when you label files as vulnerable or neutral, some files are not reported as vulnerable in the dataset, either because they have not yet been discovered or because they have not been sampled. This introduces inaccuracies in the performance

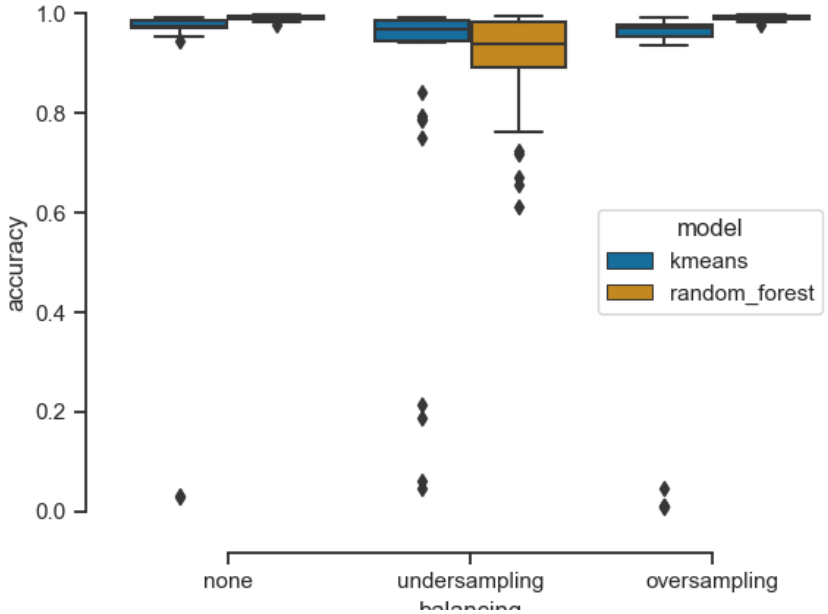


Figure 6.5: Accuracy on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

indicators.

Construct and Conclusion Validity: The construct validity relates to the suitability of our evaluation measures. Conclusion validity threats impact the ability to draw correct conclusions. Comparing the performance of learning systems are non-trivial tasks because there are many different performance metrics that reflect different aspects of performance. In this study we report widely used assessment parameters, namely Recall, Precition, Accuracy, F1 and MCC. These measures have been extensively studied previously by other researchers of the scientific literature.

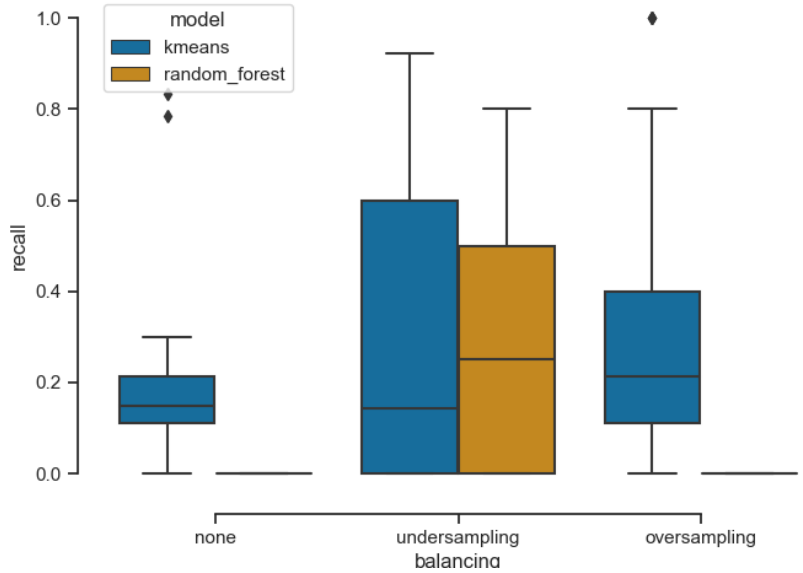


Figure 6.6: Recall on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

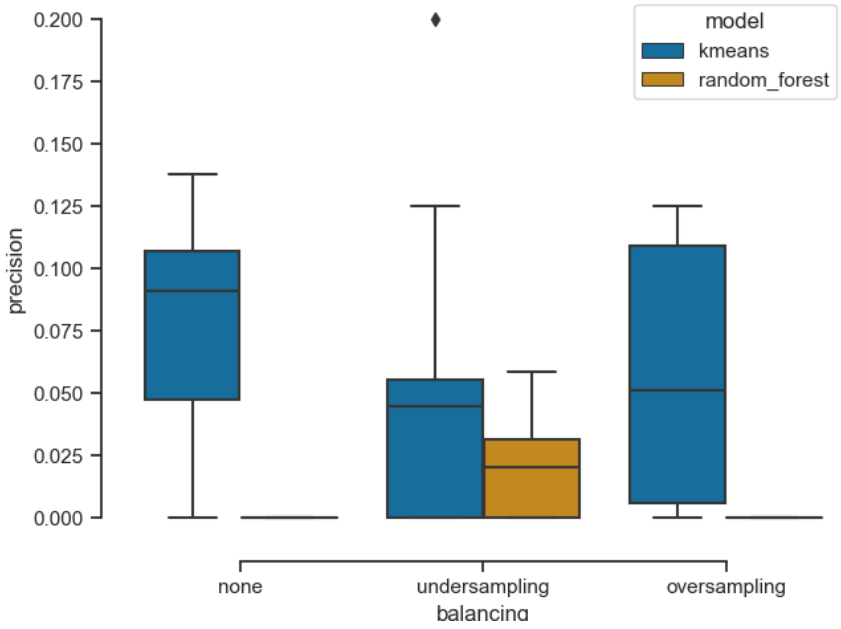


Figure 6.7: Precision on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

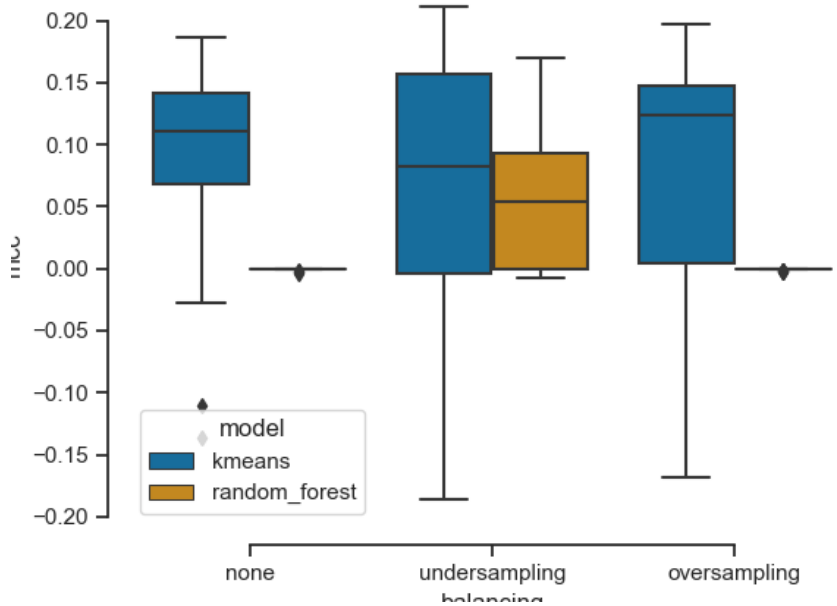


Figure 6.8: MCC on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when applying any data balancing.

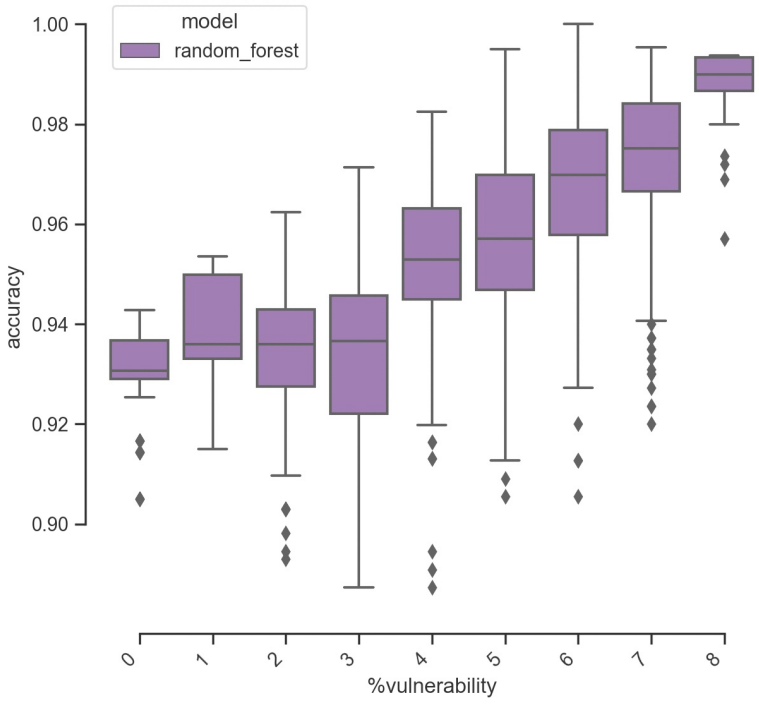


Figure 6.9: Accuracy on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

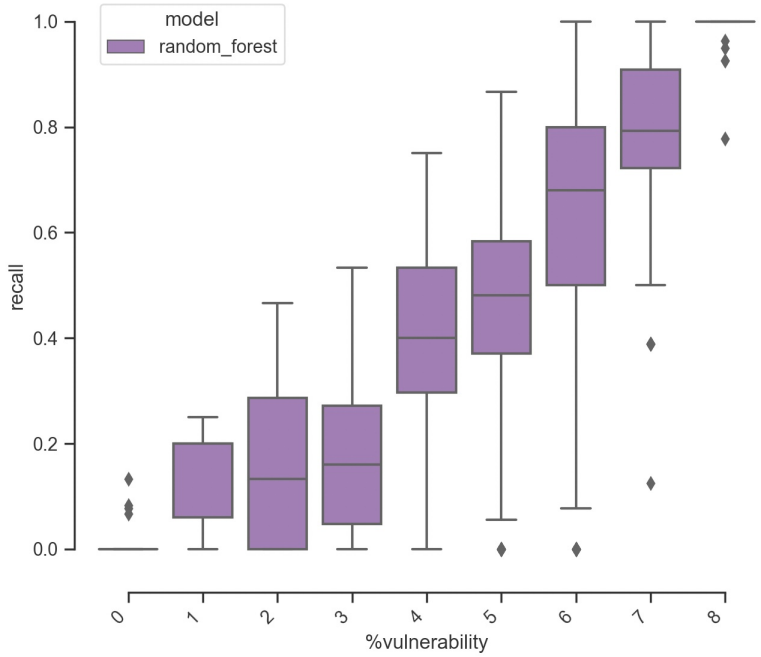


Figure 6.10: Recall on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

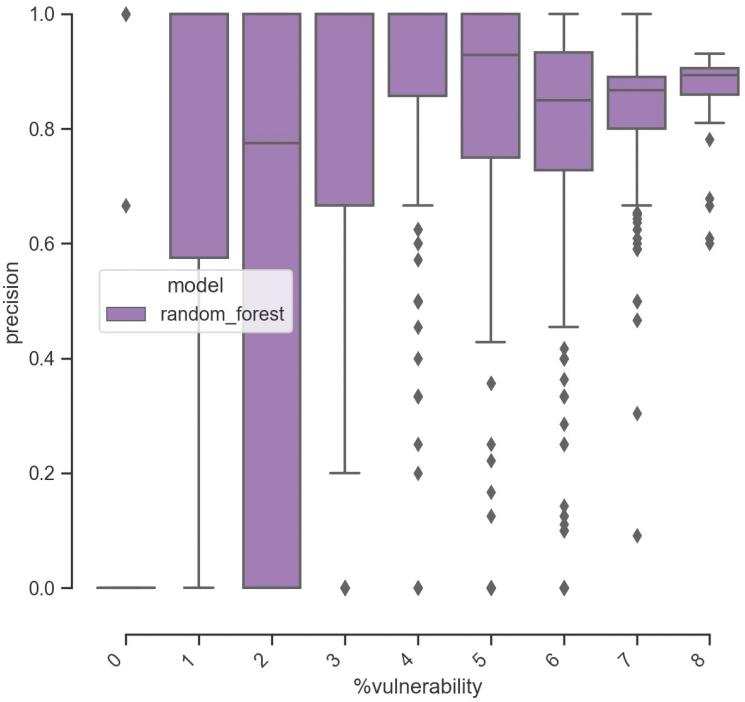


Figure 6.11: Precision on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

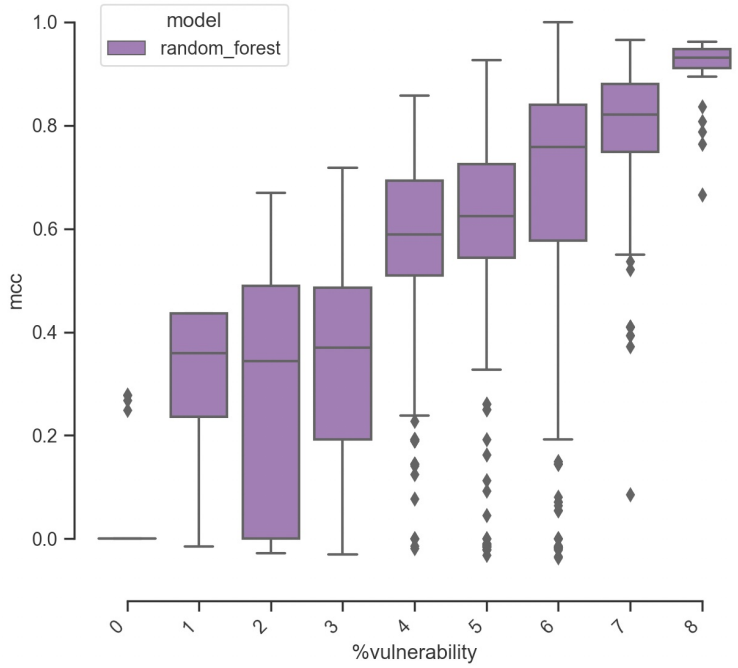


Figure 6.12: MCC on context PHPMyAdmin in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

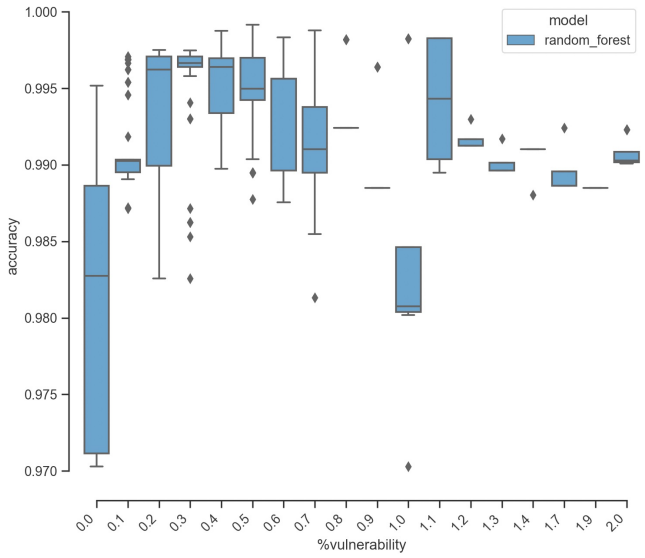


Figure 6.13: Accuracy on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

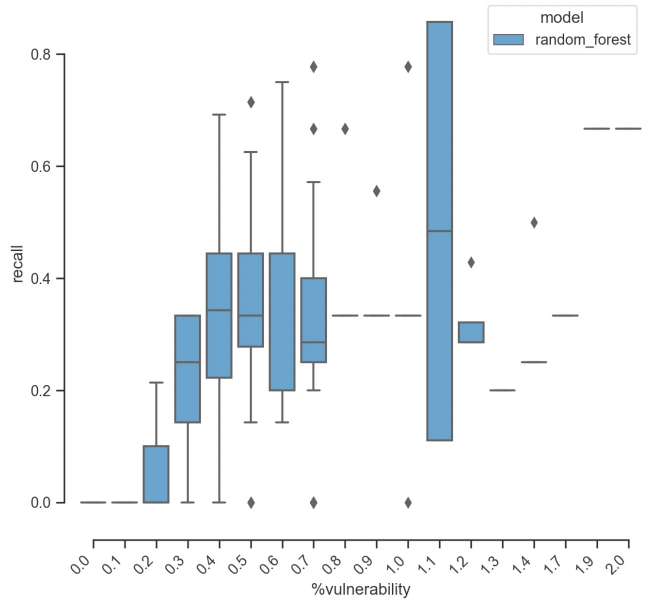


Figure 6.14: Recall on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

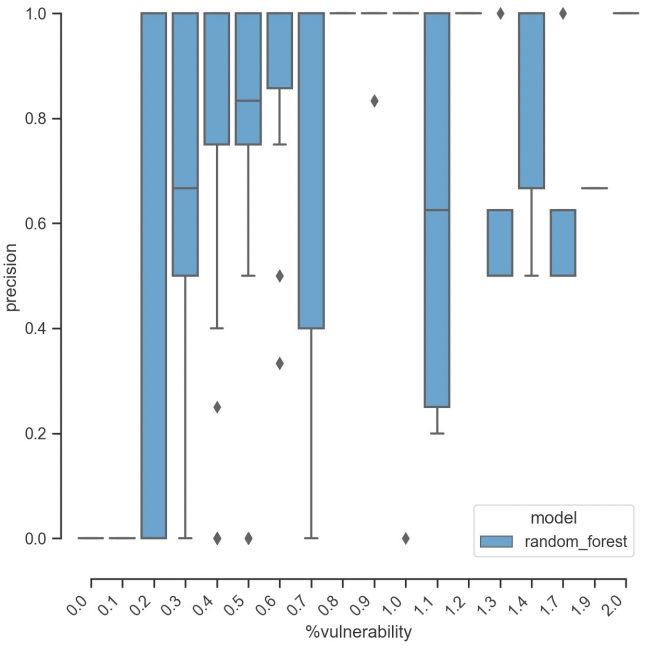


Figure 6.15: Precision on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

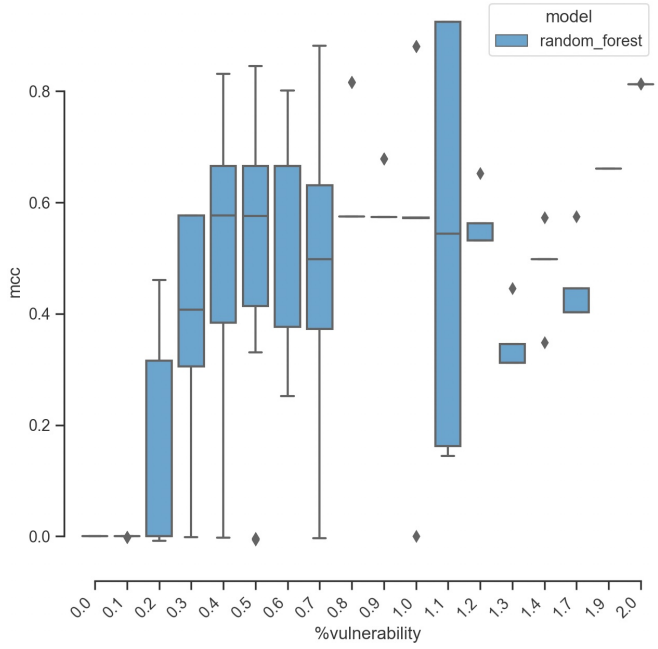


Figure 6.16: MCC on context Moodle in a "real world" setting - with data labelled information available at model-building time for Software Metrics when using future data.

CHAPTER 7

Conclusion

In this chapter, we draw up a summary of our work and propose our ideas for further research.

In this thesis work, we investigated the suitability of machine learning models for vulnerability prediction in a real software development setting.

To evaluate the models for software vulnerability prediction, analyze their performance, and answer our research questions, we run a total of 4,009 experiments: 870 experiments for `run_real` mode and 3,139 experiments for `run_timespan` mode.

For our research we used the Walden et al.'s PHP vulnerability dataset which contains vulnerability data mined from two popular open-source PHP applications, i.e., PHPMyAdmin and Moodle[13]. We studied the behavior of vulnerability prediction models within-project, applying balancing techniques such as undersampling and oversampling (SMOTE). For our experiments we have chosen two types of learning: supervised learning and unsupervised learning. Supervised learning was implemented with Random Forest classification algorithm, and unsupervised learning with KMeans clustering algorithm. In this study, we empirically investigate the effect of data balancing on the performance of software vulnerability prediction models. We applied classification algorithm Random Forest and clustering algorithm Kmeans in conjunction with two data balancing techniques on two public datasets.

We have seen that unsupervised models have a high false positive rate. Perhaps unsupervised models actually identify vulnerabilities that are not yet known, so we too are mistaken in saying that their performance is low, because perhaps we do not even know the truth about the vulnerabilities present. Still, it is worth noting that these datasets have been widely used and validated in previous research, making this conclusion difficult to hold.

Experimental results demonstrated that the performance of a vulnerability prediction model is affected by data balancing approaches. This confesses that highly unbalanced vulnerability prediction datasets should be balanced to accurately estimate vulnerable software components [16] [24] [28] [29].

Finally, we have shown that vulnerability prediction models have really good performance when trained on data in the research context, i.e. they consider all available data on vulnerabilities (recall of 0.89, 0.7 for PHPMyAdmin, Moodle, MCC of 0.97, 0.81 for PHPMyAdmin, Moodle). However, the performance is poor when considering data in the realistic context (recall of 0.2, 0.4 and MCC of 0.2, 0.1 for PHPMyAdmin, Moodle). These values are

much lower than the reasonable ones (of 0.7), suggested by the scientific literature. The results obtained are in line with previous works. So we can say that machine learning methods for software vulnerability prediction in the current state cannot be used in the real context.

We also investigated the effectiveness of machine learning models for predicting vulnerabilities in the real world context when using further data. We concluded that for the PHPMyAdmin database the performance starts to be reasonable with vulnerability ≥ 6 %, while for the Moodle dataset the performance starts to be reasonable with vulnerability $\geq$ 1.9%.

Finally, we have identified factors that may threaten the validity of our results and we presented the actions we have taken to mitigate the risk. To mitigate threats to external validity, we chose datasets that have been extensively used in scientific literature for vulnerability prediction by many researchers. Our experiments were conducted using publicly available reference data. This allows comparison with the results of other experiments. An external validity threat is that all of these projects were developed with PHP language and we do not consider the projects developed with other languages, such as C, C++, python, Java or to other types of software, such as desktop or mobile applications. Hence, we do not claim that our results can be generalized to all systems, as the systems under study may not be representative of systems in general. To mitigate such threats, various data with different programming languages could be used for future experiments. To mitigate threats to internal validity we implemented our framework using third-party library, including scikit-learn and R, to avoid potential errors in the implementation process, which is beneficial for mitigating the internal validity threat. Furthermore, we have made available our experiment results and the source code of our work [38]. This allows other researchers to replicate the results of the experiments.

Our study was based on within-project setting, it would be interesting to study the behavior of these models in cross-project setting. To address the threat to external validity, the current study should be extended to multiple datasets, programming languages, or other types of software, such as desktop or mobile applications. Another hurdle in studying software vulnerability prediction methods is the lack of standard benchmarking datasets. There

is an urgent need for a set to allow a correct evaluation and comparison of the effectiveness of the proposed approaches.

In this thesis work we have focused on the study of supervised and unsupervised learning per software vulnerability prediction. However, research works related to the use of deep neural networks approaches to study vulnerabilities in software are very promising. Future work could focus on meta-heuristics or genetic algorithms[39]. Another interesting approach to study is semantic code clone identification applications in vulnerability prediction[22] [27].

Bibliography

[1] Clusit. *Rapporto Clusit 2021 sulla sicurezza ICT in Italia*. URL: https://www.mmn.it/wp-content/uploads/2021/05/Rapporto-Clusit_03-2021-web.pdf.

[2] Cimpanu C. *Czech hospital hit by cyberattack while in the midst of a COVID-19 outbreak*. URL: https://www.zdnet.com/article/czech-hospital-hit-by-cyber-attack-while-in-the-midst-of-a-covid-19-outbreak/.

[3] Osborne C. *New ransomware masquerades as COVID-19 contact-tracing app on your Android device*. URL: https://www.zdnet.com/article/new-crycryptor-ransomware-masquerades-as-covid-19-contact-tracing-app-on-your-device/.

[4] Tidy J. *How hackers extorted $1.14m from University of California, San Francisco*. URL: https://www.bbc.com/news/technology-53214783.

[5] ISO/IEC: ISO/IEC 25010. *Systems and software engineering - Systems and soft- ware Quality Requirements and Evaluation (SQuaRE) - System and software qual- ity models. ISO/IEC (2011)*. URL: https://www.iso.org/standard/35733.html.

[6] GDPR. *Complete guide to GDPR compliance*. URL: https://gdpr.eu/.

[7] CVE. *Definition of vulnerability*. URL: https://www.cve.org/ResourcesSupport/Glossary#.

[8] Sayed Mohhamad Ghaffarian and Hamid Reza Shahriari. *Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey*. URL: https://dl.acm.org/doi/abs/10.1145/3092566.

[9] James Walden, Jeff Stuckman, and Riccardo Scandariato. *Predicting Vulnerable Components: Software Metrics vs Text Mining*. URL: https://ieeexplore.ieee.org/abstract/document/6982351.

[10] Stephan Neuhaus and Thomas Zimmermann. *Predicting Vulnerable Software Components*. URL: https://dl.acm.org/doi/pdf/10.1145/1315245.1315311.

[11] Chao Ni et al. *Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction*. URL: https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=6930&context=sis_research.

[12] Xinli Yang et al. *Deep Learning for Just-In-Time Defect Prediction*. URL: https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=4096&context=sis_research.

[13] Riccardo Scandariato, Jeffrey Stuckman, and James Walden. *Replication dataset*. URL: https://seam.cs.umd.edu/webvuldata/studydata.html.

[14] Scikit-learn. *Machine Learning in Python*. URL: https://scikit-learn.org/stable/.

[15] Matthieu Jimenez et al. *Learning from What We Know How to Perform Vulnerability Prediction using Noisy Historical Data*. URL: https://arxiv.org/pdf/2012.11701.pdf.

[16] Nitesh V. Chawla et al. *SMOTE: Synthetic Minority Over-sampling Technique*. URL: https://arxiv.org/pdf/1106.1813.pdf.

[17] Wei Fu and Tim Menzies. *Revisiting Unsupervised Learning for Defect Prediction*. URL: https://arxiv.org/pdf/1703.00132.pdf.

[18] Feng Zhang, Quan Zheng, and Ying Zou. *Cross-project Defect Prediction Using a Connectivity-based Unsupervised Classifier*. URL: https://www.researchgate.net/profile/Ahmed-E-Hassan-2/publication/303099415_Cross-project_defect_prediction_using_a_connectivity-based_unsupervised_classifier/links/5b7f26f6a6fdcc5f8b6371fa/Cross-project-defect-prediction-using-a-connectivity-based-unsupervised-classifier.pdf.

[19] Yibiao Yang, Yuming Zhou, and Jinping Liu. *Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better Than Supervised Models*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7886913.

[20] Jinping Liu et al. *Code churn: A neglected metric in effort-aware just-in-time defect prediction*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8169980.

[21] Meng Yan et al. *File-Level Defect Prediction: Unsupervised vs. Supervised Models*. URL: https://ieeexplore.ieee.org/document/8170121.

[22] Patrick Keller, Laura Plein, and Yves Le Traon. *What You See is What it Means! Semantic Representation Learning of Code based on Visualization and Transfer Learning*. URL: https://dl.acm.org/doi/pdf/10.1145/3485135.

[23] Akhan Akbulut Cagatay Catal. *Can we predict software vulnerability with deep neural network?* URL: https://www.researchgate.net/profile/Akhan-Akbulut/publication/322131540_Can_we_predict_software_vulnerability_with_deep_neural_network/links/5a467bb30f7e9ba868aa4b4a/Can-we-predict-software-vulnerability-with-deep-neural-network.pdf.

[24] Matthieu Jimenez et al. *The Importance of Accounting for Real-World Labelling When Predicting Software Vulnerabilities*. URL: https://dl.acm.org/doi/pdf/10.1145/3338906.3338941.

[25] CIRCL. *Cve-search Common Vulnerabilities and Exposures (CVE)*. URL: https://www.circl.lu/services/cve-search/.

[26] Yonghee Shin and Laurie Williams. *Can traditional fault prediction models be used for vulnerability prediction?* URL: https://link.springer.com/article/10.1007/s10664-011-9190-8.

[27] Guanjun Lin et al. *Software Vulnerability Detection Using Deep Neural Networks: A Survey*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9108283.

[28] Kaya Aydin et al. *The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models*. URL: https://www.researchgate.net/publication/332563143_The_impact_of_feature_types_classifiers_and_data_balancing_techniques_on_software_vulnerability_prediction_models.

[29] Martin Shepperd, Yuchen Guo, and Qinbao Song. *A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8359087.

[30] Xiang Chen et al. *Large-Scale Empirical Studies on Effort-Aware Security Vulnerability Prediction Methods*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8809906.

[31] Yonghee Shin, Andrew Meneely, and Laurie Williams. *Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5560680.

[32] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. *Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5477059.

[33] *Learning To Predict Vulnerabilities From Vulnerability-Fixes: A Machine Translation Approach*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5477059.

[34] Martin Shepperd, Ning Li, and Yachen Guo. *A Systematic Review of Unsupervised Learning Techniques for Software Defect Prediction*. URL: https://arxiv.org/pdf/1907.12027.pdf.

[35] Riccardo Scandariato and James Walde. *Predicting Vulnerable Classes in an Android Application*. URL: https://www.researchgate.net/publication/235760736_Predicting_Vulnerable_Classes_in_an_Android_Application.

[36] James Walden and Riccardo Scandariato. *Experimentation in software engineering*. URL: https://dl.acm.org/doi/book/10.5555/2349018.

[37] Tim Menzies and Martin Shepperd. *"Bad Smells" in Software Analytics Papers*. URL: https://arxiv.org/pdf/1803.05518.pdf.

[38] Alexandra Sheykina. *The Software Vulnerability Prediction based on supervised and unsupervised learning for Real-World Labelling*. URL: https://github.com/sashasheykina/realunsuperlearning.

[39] Canan Batur Şahin, Özlem Batur Dinler, and Laith Abualigah. *Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features*. URL: https://link.springer.com/article/10.1007/s10489-021-02324-3.

Acknowledgements

At the end of this thesis work I would like to thank all of the people who directly or indirectly contributed to the development of this work. A first acknowledgment is for my advisor, professor Filomena Ferrucci, who has supported me in any way and she has also played an important role in making me understand what research is, and why it can be loved. I have benefited greatly from your wealth of knowledge.

My heartfelt thanks to Phd student Giulia Sellitto for her assistance at every stage of the research project. Your encouraging words and thoughtful, detailed feedback have been very important to me.

I thank my family for all their help in every way and at all times, helping me as they could and putting up with me in moments of difficulty that unfortunately all students sooner or later encounter. Specially, I wish to show my appreciation to my husbend for his unwavering support and belief in me.