
# Data mining software management

**Maria Vasileiou**

SID: 3308210048

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

*Master of Science (MSc) in Data Science*

JANUARY 2023

THESSALONIKI – GREECE


# Data mining software management

**Maria Vasileiou**

SID: 3308210048

| | |
|---|---|
| Supervisor: | Prof. Christos Tjortjis |
| Supervising Committee | Dr Berberidis |
| Members: | Dr Koukaras |

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

*Master of Science (MSc) in Data Science*

JANUARY 2023

THESSALONIKI – GREECE

# Acknowledgements

I would like to express my deepest gratitude to Professor Christos Tjortjis for his support and guidance as well as for the time he devoted to me for the preparation of this dissertation. Finally, I would like to thank my family for their support and the trust they showed me as well as my friends for their support during my studies.

# Abstract

With the continuous technological evolution, the amount of software that is implemented is constantly increasing. Also, due to the fact that electronic devices are nowadays a significant part of people's lives, there is a need for the software to become increasingly better. As the demand grows so does the need to produce new software and improve the existing one. In order to achieve the upgrade of the existing software as quickly as possible while remaining on budget, in this dissertation, a number of data mining techniques were used. Many techniques have been used in previous research for software defect detection. In this dissertation, some of those techniques were applied in data extracted from the source code of notepad++ to find bugs and defects. Finally, the results of these techniques will be validated using the actual changes that have been made in the next release of the chosen application and the purpose is to examine and compare the results of the algorithms that were used.

**Keywords**: Data mining, software defect detection, software management, machine learning

Maria Vasileiou

07/01/2023

# Contents

# 1 Introduction

As the need for software-based solutions increases so does the need to create maintainable and extendable code. Regardless of the size of a project, software maintenance can be a very difficult goal to achieve, and it requires a lot of time, effort and human resources [1]. Due to this, the cost of software maintenance is also significantly high so it is crucial to find suitable methods in order to detect software defects in an early stage.

Predicting software components that are prone to defects is a key objective in software engineering. This would make it possible to allocate testing resources effectively and make better-informed choices regarding the caliber of releases [2]. Consequently, there are numerous studies on software quality and software defect prediction [17],[18],[19].

Metrics can be used to analyze code quality and maintainability and to assess the characteristics of software. Then data mining can be used to extract information and find hidden patterns in data, enabling the analysis of software metrics for maintenance reasons. Data mining is seen as a good solution for huge, unfamiliar software systems since it can handle massive volumes of data without any prior subject expertise [3].

In this dissertation, research has been conducted in order to find the defective files using a dataset produced from the source code of notepad++. This dataset has been produced by extracting some metrics from the source files of twenty-six releases of the open-source program notepad++. After the files of each of those releases had been classified to defective or non-defective, various data mining techniques were applied to this data in order to classify either a subset of these files or the files of the last release of notepad++, which at the time that this dissertation is conducted is the 8.4.5 release. During this research eight different experiments have been conducted, each with a different combination of training and test dataset.

The purpose of this dissertation is to compare various data mining techniques by applying to a portion of the whole dataset and to the data of the newest release of notepad++ and evaluate those results using the data of the next release of the application.

The results of this dissertation shows that the combination of the defective and non-defective samples in the training set can significantly affect the produced metrics of all the

models. Also, it is observed that in the case that an extremely highly uneven dataset is used as training set and the dataset of the last release of the application as test set, the ML algorithm that was affected the most was Support Vector Machine (SVM). Finally, when the dataset consists of the same number of defective and non-defective samples and 20% of that is used as test data the models give the best results in terms of precision, recall and f1-score.

The remaining of this dissertation is organized as follows: Chapter 2 introduces key concepts on software defect detection and data mining and briefly describes related work that have been done in the past. Chapter 3 discusses the dataset that is used during this dissertation and the methodology that were used. Chapter 4 presents experimental results. Finally, Chapter 5 discusses these results and evaluates threats to validity and Chapter 6 concludes this dissertation with the conclusions that can be extracted from the results and with directions for further work.

# 2 Background

In this chapter related work that was conducted in the past as well as some important definitions used in this dissertation are presented.

## 2.1 Literature review

### 2.1.1 Software defect detection

The main elements influencing software quality are software defects. Several things can lead to software defects. The main determinants of the factors are the features of the software itself and the technical implementation, such as the scale and complexity of the software, the developers' comprehension of the client's requirements, the grammar and algorithms employed during project development, and the degree of teamwork. Software defects are mostly caused by coding mistakes [5].

Software defect prediction is an active research topic in the field of computer science. Predictions might be dynamic or static when it comes to software defects. The majority of dynamic software defect prediction relies on empirical or statistical methods to estimate the allocation of software defects throughout the course of the software's life cycle. On the other hand, static software defect prediction develops a model for forecasting the number and the allocation of defects in undisclosed modules based on software defect metrics. Despite the fact that several software defect prediction approaches have been suggested, the technology is still regarded as imprecise [5].

As it is referred above static software defect detection is depending on software metrics. Software metrics are indices and parameters that characterize the properties of software products and offer a measure of software quality. The McCabe, Halstead, Childamber Kemerer, and complexity metrics are current software metrics, which are widely used in software defect prediction [5].

The subset of software metrics that is called software quality metrics is concerned with the project, process, and product quality. Product metrics are used to define a product's attributes, such as its size, complexity, design characteristics, efficiency, and degree of quality. To improve software development and maintenance, process metrics can be used,

such as the effectiveness of defect removal during development, the pattern of testing fault arrival, and consequently the reaction time of the fix process. The number of software engineers, the pattern of staffing over the course of the software life cycle, cost, timeline, and efficiency are only a few examples of the project features and execution that are described by project metrics [6].

The objectives of existing metrics-based software defect prediction solutions may be essentially separated into two categories: ranking and categorization. The former seeks to forecast the quantity of problems in software modules, while the latter seeks to forecast if the module includes defects at all. Both of the preceding methodologies, which have been investigated for decades, assist developers in efficiently deploying resources [5].

Akiyama developed the Akiyama model, which is a relationship between the number of defects and lines of code (LOC). Following that, Arthur, Ottensteln, and Lipow hypothesized a relationship between defect count and complexity measures. These models just have one variable, such as LOC. With the increasing diversity of metrics, several regression approaches, such as multiple linear regression, negative binomial regression (NBR), SVMs, and random forest algorithms, are employed to describe the relationship between metrics and the number of errors. Several classification techniques have been employed concurrently to build software defect prediction models for completing a classification task [5].

### 2.1.2 Data Mining

The process of extracting implicit, previously undiscovered, and possibly beneficial information from data is known as data mining. It involves looking for patterns in vast quantities of data using methods like classification, association rule mining, and clustering [1]. Various of those techniques as well as some combinations of them have been used for software defect detection. In this dissertation, only classification methods have been used.

### 2.1.3 Data Mining Algorithms Selection

As it is mentioned above from all the different data mining techniques that can be used for software defect detection in this dissertation only classification techniques have been used. More specifically for the experiments that have been conducted during this dissertation five Machine Learning (ML) models have been used. Those models are SVMs, Naïve Bayes, Logistic regression, Random Forest and k-Nearest Neighbors (kNN).

#### 2.1.3.1 Support vector machine description

SVMs are a group of associated supervised learning techniques applied to regression and classification problems. SVMs are systems that use the hypothesis space of a linear function in a high-dimensional feature space and are trained using an optimization theory-based learning algorithm that incorporates a learning bias [7].

#### 2.1.3.2 Naïve Bayes description

One of the best and most productive classification methods is naive Bayes. A learner attempts to build a classifier from a set of training examples with class labels in classification learning tasks [8].

The simplest type of Bayesian network is a naive bayes network, in which all characteristics are independent of one another regardless of the value of the class variable. This is known as conditional independence. It is clear that most real-world applications rarely satisfy the conditional independence assumption. Extending the structure of naive Bayes to explicitly indicate attribute dependencies is an easy way to get around this disadvantage [9].

#### 2.1.3.3 Logistic regression description

Logistic regression models are statistical models that describe the relationship between a qualitative dependent variable (that is, one that can only take particular discrete values) and an independent variable.

To investigate the influence of predictor variables on categorical outcomes, logistic regression models are used. When the outcome is typically binary, such as the existence or absence of an illness, binary logistic model is used. A logistic regression model is referred to as simple logistic regression when it has just one predictor variable. The model is known as a multiple or multivariable logistic regression when there are numerous predictors, including categorical and continuous variables as predictors [10].

#### 2.1.3.4 Random Forest description

Random forest is a supervised learning approach that follows the straightforward yet powerful "divide and conquer" principle: sample subsets of the data, generate a random tree predictor on each tiny piece, then combine these predictors [11].

The fact that forests may be used to solve a variety of prediction issues and only require a small number of tuning parameters has tremendously boosted their popularity. The approach is well known for its accuracy, minimal sample sizes, and high-dimensional

feature spaces in addition to being straightforward to apply. It also has the ability to handle big real-world systems because it is quickly parallelizable [11].

#### 2.1.3.5 k-Nearest Neighbors (kNN) description

In classification issues, kNN has been widely employed. KNN is built on a distance function, which calculates how different or similar two instances are. The distance function is frequently the typical Euclidean distance d(x, y) between two instances, x and y [12].

KNN determines the most prevalent class of an instance x's k nearest neighbors for an instance x. Also, KNN is a case of lazy learning. Lazy learning merely saves training data at the time of training and postpones learning until the time of categorization. While eager learning creates an explicit model during training [12].

## 2.2 Related work

### 2.2.1 Using Classifiers for Software Defect Detection

One of the related studies that have been conducted in the past is the work by Perreault et al. In this study, the performance of five classifiers is assessed in the context of software defect detection. Those classifiers are naive Bayes, neural networks, support vector machines (SVM), logistic regression, and k-nearest neighbor (kNN). For the assertion of each of those classifiers five datasets from the NASA metrics data program repository were used. Regarding the evaluation of those models, accuracy and F1-score were used to assess performance, while an ANOVA was used to assess significance [4].

The objective of this study was to ascertain whether or not the classifier that is used matters while attempting to forecast and detect software defects. This is tested by putting a number of classifier models into practice and evaluate how well they perform on datasets for defect detection [4].

The datasets targeted for defect detection in this work are those where each datapoint corresponds to a snapshot of the underlying source code. Each datapoint's properties are designed to give a clue as to whether or not the code is flawed [4].

In this study, two simultaneous sets of experiments were conducted using several metrics in order to prevent biasing one particular metric. The null hypothesis for the accuracy experiments asserts that there is no change in classification accuracy depending on the classifier selected. Similar to this, the null hypothesis for the F1 experiments is that there is no variation in the F1 measure between classifiers. These hypotheses are intended to

provide a response to the query of which classifier, if any, should be used for defect prediction in software systems [4].

By analyzing the results, it is observed that in every instance, there is little to no difference between using accuracy and F1 as a response variable. Despite how intriguing this material is, no conclusions can be made from it that are not reliant on intuition. Also, results reveal that all models, albeit being unsophisticated, can reliably identify software flaws using static program properties. Finally, it becomes apparent that for some datasets, the remaining methods outperformed Bayes and SVMs [4].

### 2.2.2 Data mining techniques for software defect detection

Different data mining techniques have been used over the years in order to detect software defects. One of those is the self-organizing data mining method (SODM). The fundamental component of this method is the group method of data handling (GMDH), which divides the data into training and test sets. The exterior criterion is used to choose the interior candidate model in the training set, and the interior criterion is used to estimate the parameters. Until the external criterion value is no longer improved, this process is repeated. This termination rule provides plausible predictions at a specific degree of noise and ensures the accuracy of data fitting, resulting in a complexity model with ideal balance [5].

Another widely used method is regression. Regression is a statistical method for assessing the connection between several variables. It examines the connection between independent or predictor variables and the dependent or response variable. A mathematical equation that predicts the response variable as a linear function of the predictor variable represents the connection [6].

In addition, association rule mining is an approach for locating intriguing connections between variables in huge databases. Finding associations or connections between groups of elements or objects in a database is the objective. Finding rules that can anticipate the occurrence of an item based on the occurrence of other things is basically its main purpose [6].

Clustering is the process of organizing a group of objects into groups or clusters whose members share some characteristics. It involves clustering a collection of things so that they are similar to one another and different from those in other clusters [6].

Classification is another technique that has been used which entails foreseeing a specific consequence of a given input. The input data for a classification approach, sometimes referred to as a training set, contains all the objects that have previously been assigned class labels. The goal of a classification algorithm is to study the training data set, learn from it, and create a model. The classification of test data for which the class labels are unknown is then performed using this model. Some widely known classification techniques are: Neural Networks, Decision Trees, Naive Bayes, SVMs, Case Based Reasoning.

# 3 Data and methodology

## 3.1 Data Preparation

The first step of this dissertation was the creation of the dataset that were used later on in the experiments that were conducted. The program that was examined was notepad++ and in order to examine this program and find which of its files will be defective in the next release, a dataset with the metrics of each of the examined files of this program needed to be created. For this purpose, the LocMetrics program was used. This is a software tool that was used in order to extract metrics from some of the releases of notepad++. LocMetrics extracts metrics on file, folder and function level but in this dissertation, the reports on a file level will be used.

Using this program, twenty-six reports have been generated one for each of the 8.0, 8.1, 8.1.1, 8.1.2, 8.1.3, 8.1.4, 8.1.5, 8.1.6, 8.1.7, 8.1.8, 8.1.9, 8.1.9.1, 8.1.9.2, 8.1.9.3, 8.2, 8.2.1, 8.3, 8.3.1, 8.3.2, 8.3.3, 8.4, 8.4.1, 8.4.2, 8.4.3, 8.4.4 and 8.4.5 of the notepad++ releases.

The reports that have been generated are referring to all the files of the source code of each release that have either of the following types: *.cpp, *.h, *.hpp. The columns of the dataset that was created and their meaning are shown in Table 1 below:

| | |
|---|---|
| LOC | Lines of Code |
| SLOC-P | Source Lines of Code-Physical: Physically executable source lines of code |
| SLOC-L | Source Lines of Code-Logical: Logically executable source lines of code |
| MVG | McCabe's VG Cyclomatic Complexity: A measure of the decision complexity of the functions which make up the program. The number of linearly independent paths across a directed acyclic graph, which depicts the |

| | |
|---|---|
| | control flow of a subprogram, is the precise definition of this metric. The analyzer counts this by recording the number of distinct decision outcomes contained within each function, which yields a good approximation to the formally defined version of the measure. |
| BLOC | Blank Lines of Code |
| C&SLOC | Code and Comment lines of code |
| CLOC | Comment only lines of code |
| CWORD | Commentary Words |
| HCLOC | Header Comment Lines of code |
| HCWORD | Header Commentary Words |

Table 1. The metrics of the notepad++ generated dataset.

In addition to all of the above metrics in each report on a file level that have been generated two additional columns have been added. The first column that was added includes the release in which each file and its corresponding metrics belong, and the second column consists of true of false depending on if the corresponding file is considered defective or not. As the code of the notepad++ is available online it is more optimal to compare the branch of each release with its next and find the files that have been changed and marked them as defective while all the others will be marked as non-defective. More specifically in this dataset a file is consider as defective if it has been edited in any way during a commit that declares that is fixing something. If a file is not edited in a commit with the word “fix” in its title or description is marked as non-defective.

For the creation of the dataset that was used in this dissertation all the LocMetrics reports on file level that have been generated for all of the above releases except from release 8.4.5 were added in a common csv file. The data for the 8.4.5 release were excluded in order to be used as test data in some of the experiments that were conducted during this dissertation.

The dataset that is created has thirteen columns-attributes and 14038 samples, 13752 of which are non-defective and 286 defective.

In order to use those data as input in various ML algorithms some preprocessing needed to be performed. The first step was to observe if there are any null values in the dataset.

After the usage of the appropriate python code, it is observed that this dataset does not contain any null values. The next step was to remove the "File" column that contains the file names and the "RELEASE" column that contains the release in which each file belongs, as those are both objects and cannot be used to train the model. The final step was to transform the values of "DEFECTS" column to integers, so that the true becomes one (1) and the false zero (0). Then the transformed values of the "DEFECTS" column were removed from the dataset and added to a new variable as this column contains the results.

## 3.2 Methodology

In order to perform software defect detection on the data described in the previous section various ML models have been used.

The ML models that were applied were Support Vector Machines (SVMs), Naïve Bayes, Logistic regression, Random Forest and k-Nearest Neighbors (kNN). Those algorithms have been used to conduct eight different experiments. These experiments differ from each other in terms of the training set and the test set of the ML algorithms that are used in them, as it is shown in figure 3.1.

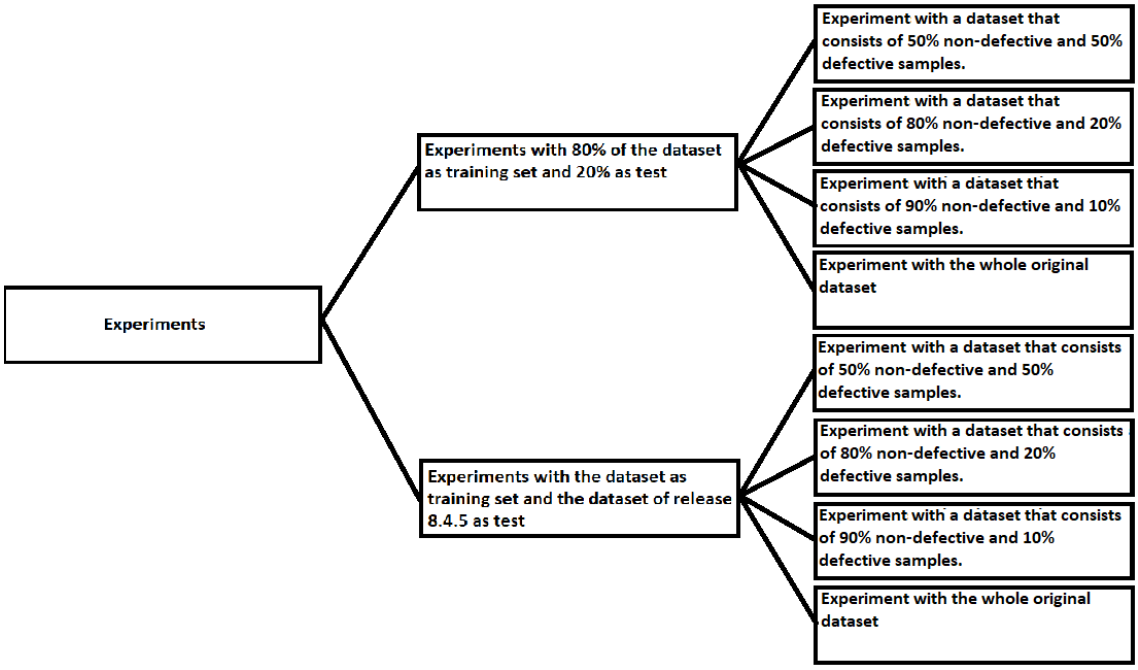


Figure 3.1. Methodology as a diagram.

In the first experiment, the dataset that was used for each of those models were only part of the original dataset. More specifically, the dataset used in this experiment consists of

50% non-defective samples and 50% defective. This subset of the original dataset is chosen in order to train and test the models using a balanced dataset.

In order to create this dataset, the number of the defective samples of the original dataset needed to be found. As the defective samples were found to be 286, this number was used in order to randomly extract an equal number of samples from the 13752 samples of the subset of the non-defective samples. Then by combining those two subsets the new dataset that were used in this experiment were created.

The need to create this balanced dataset with only part of the original dataset has been created due to the fact that the original dataset is highly imbalanced. The original dataset contains 14038 samples, 13752 of which have been declared as non-defective and only 286 of them as defective. This imbalance between the samples of these two categories was expected as it is normal for a project to have more non-defective files than defective ones. In the first experiment that was conducted 20% of the dataset that was used was randomly selected in order to be used as test dataset. Then all the ML algorithms that is mentioned above have been used with the 80% of the new dataset as training set and with 20% of it as test dataset.

During the second experiment the same algorithms were used but this time the dataset that was used consisted of 90% non-defective samples and 10% defective. This new dataset was acquired in a similar way with the dataset of the first experiment. More specifically, given that the 10% of the new dataset needed to be the 286 defective samples, the size of the new dataset was calculated to be 2860. Then a subset of 2860*0.9=2574 samples was randomly extracted from the subset of the non-defective samples. The defective and non-defective samples that were extracted were added in the same dataset which is the dataset that was used in this experiment. Finally, 80% of this dataset were used to train the chosen models and 20% of this were used to test them.

The third experiment is similar to the second but in this experiment the dataset that was used included 80% of non-defective samples and 20% of defective ones. In order to create this dataset, the same methodology as in the second experiment were used. The difference is that in this case given that the 286 defective samples should be the 20% of the new dataset, the size of the new dataset was calculated to be 1430. As a result, 1430*0.8=1144 of the non-defective samples were extracted to be used in the new dataset. After the defective and non-defective subsets are added to the new dataset, 80% of this dataset is used to train the models and 20% of it is used to test them.

The fourth experiment were the one where the whole dataset was used. In this experiment the initial imbalanced dataset was used without any alteration and more precisely 80% of it was used as training dataset and 20% of it as test dataset for all models as in the previous experiments

The fifth experiment differed from the previous ones as in this the whole dataset was used as training data and the data that were extracted from the release 8.4.5 -final release from which data could be extracted was used as test data.

The other three experiments that were conducted were similar to the fifth as they also had the data that were extracted from the release 8.4.5 as test data but the training dataset differed in each one of them.

More specifically, in the sixth experiment the training dataset is the one that consists of 90% non-defective and 10% defective samples, in the seventh experiment the training dataset is the one that consists of 80% non-defective and 20% defective samples and finally, in the eighth experiment the training dataset is the one that consists of 50% non-defective and 50% defective samples.

## 3.3 Evaluation metrics

Each of the models that were used, produce some metrics that will be used afterwards for their comparison. Those metrics are accuracy, precision, recall and f1-score.

Accuracy is defined as the percentage of the correctly classified units and is calculated by dividing the summary of true positives (TP) and true negatives (TN), with the summary of all samples [2],[13]. In this dissertation, true positives are the samples that were defective and were predicted as defective and true negatives are the samples that were non-defective and were predicted as non-defective.

Precision, or the caliber of a successful prediction made by the model, is another measure of the model's performance. Precision is calculated by dividing the proportion of true positives by the total number of positive predictions [14].

The recall is determined as the proportion of positive samples that were correctly identified as positive to all positive samples. Recall measures the model's ability to recognize positive samples. The recall increases as more positive samples are found [15].

The F1 score is defined as the harmonic mean of recall and precision. It can receive a minimum score of 0 and a maximum score of 1 (perfect recall and precision). Overall, it is an indicator of how accurate and reliable the model is [16].

A Confusion Matrix displays all of those metrics. An evaluation tool for classification models is a condensed table known as a confusion matrix, sometimes known as an error matrix. The number of accurate and incorrect predictions for each class is expressed using count values [16]. An example of a confusion matrix is shown in figure 3.2.

| | Predicted defective | Predicted defect free |
|---|---|---|
| Observed defective | True Positive (TP) | False Negative (FN) |
| Observed defect free | False Positive (FP) | True Negative (TN) |

Figure 3.2. Example of a confusion matrix [2]

# 4 Results

## 4.1 Results from the experiments' perspective

**Experiment 1**

As it is mentioned above in every experiment five ML methods have been applied. From the results of the first experiment where only 286 samples for each category (defective and non-defective) were used, the metrics that were described in the previous chapter for each model are generated and are shown in Table 2 and in Figure 4.1.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 80 | 86,207 | 76,923 | 81,301 |
| Logistic Regression | 80,87 | 90,566 | 73,846 | 81,356 |
| Naive Bayes | 68,696 | 87,179 | 52,308 | 65,385 |
| Random Forest | 79,13 | 83,607 | 78,462 | 80,952 |
| KNeighbors | 82,609 | 80,822 | 90,769 | 85,507 |

Table 2. Metrics of all models when applied to a dataset with 50% non-defective and 50% defective samples.

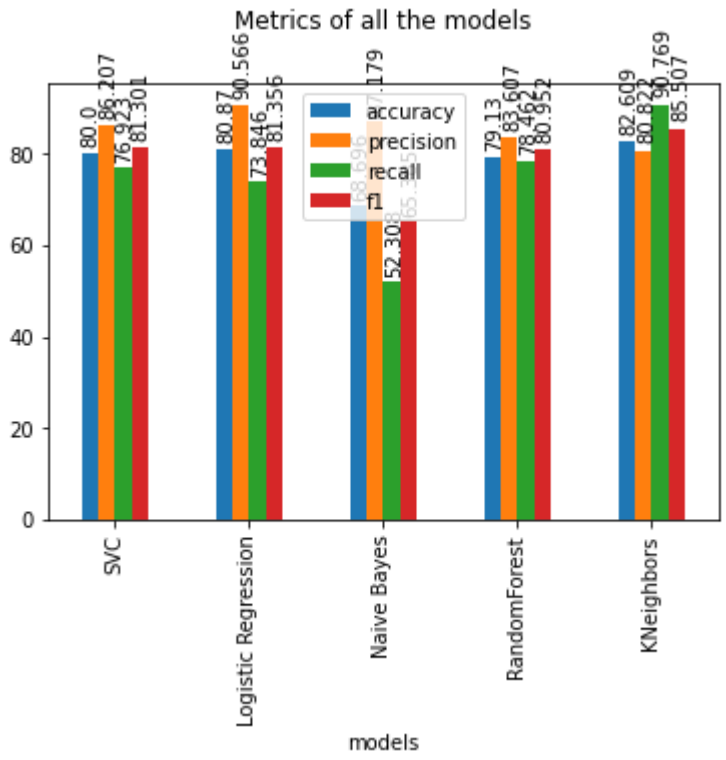


Figure 4.1. Metrics of all models when applied to a dataset with 50% non-defective and 50% defective samples.

From the above results it is apparent that the ML algorithm with the best accuracy is k-nearest neighbor, while the algorithm with the worst accuracy is Naïve Bayes. The logistic regression algorithm is also the one with the best precision while k-nearest neighbor has the worst one. However, k-nearest neighbor has also the best recall and f1 score where Naïve Bayes has the least optimal results for both of those metrics.

## Experiment 2

In the second experiment in which 90% of the dataset that were used was non-defective and only 10% defective the results seem to differ. As it is shown in Table 3 and Figure 4.2 in this experiment the best results seem to be those of logistic regression regarding accuracy and precision and those of random forest regarding recall and f1 score. The less accurate results seem to be those of k-nearest neighbor as it has the smallest accuracy and precision and those of support vector machine as it has the smallest recall and f1 score.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 91,259 | 68,75 | 19,643 | 30,556 |
| Logistic Regression | 91,958 | 72,727 | 28,571 | 41,026 |

| | | | | |
|---|---|---|---|---|
| Naive Bayes | 90,734 | 54,545 | 32,143 | 40,449 |
| Random Forest | 90,734 | 53,488 | 41,071 | 46,465 |
| KNeighbors | 90,909 | 56,25 | 32,143 | 40,909 |

Table 3. Metrics of all models when applied to a dataset with 90% non-defective and 10% defective samples.

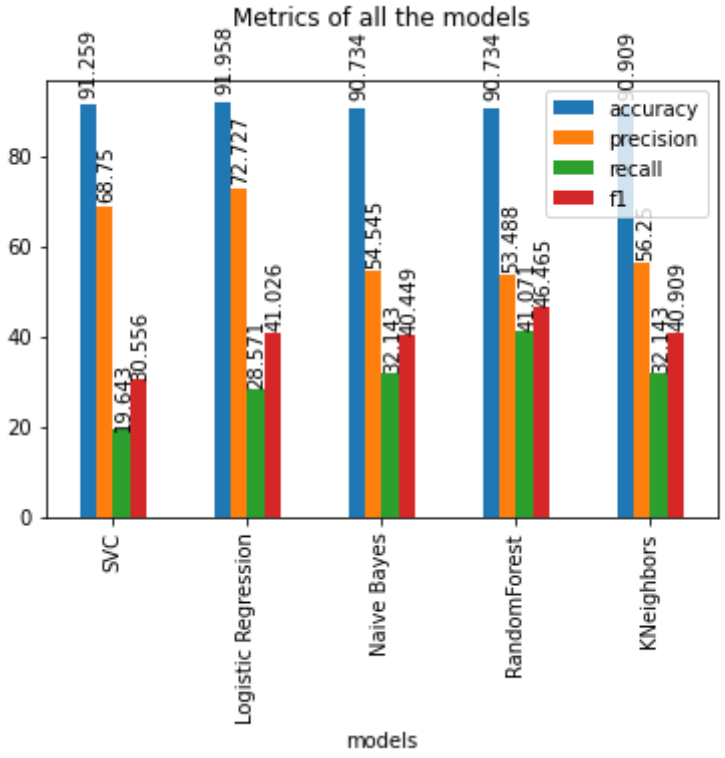


Figure 4.2. Metrics of all models when applied to a dataset with 90% non-defective and 10% defective samples.

## Experiment 3

The results of the third experiment where the 80% of the samples were non-defective and only 20% defective are shown in Table 4 and Figure 4.3. In this experiment Naïve Bayes has given the best accuracy while support vector machine has given the best precision. The most optimal results regarding the recall and the f1-score were those of random forest, while logistic regression had the less optimal results in all metrics.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 86,014 | 83,333 | 35,714 | 50 |

| | | | | |
|---|---|---|---|---|
| Logistic Regression | 83,217 | 63,333 | 33,929 | 44,186 |
| Naive Bayes | 86,364 | 81,481 | 39,286 | 53,012 |
| Random Forest | 85,664 | 67,442 | 51,786 | 58,586 |
| KNeighbors | 84,965 | 65,854 | 48,214 | 55,67 |

Table 4. Metrics of all models when applied to a dataset with 80% non-defective and 20% defective samples.

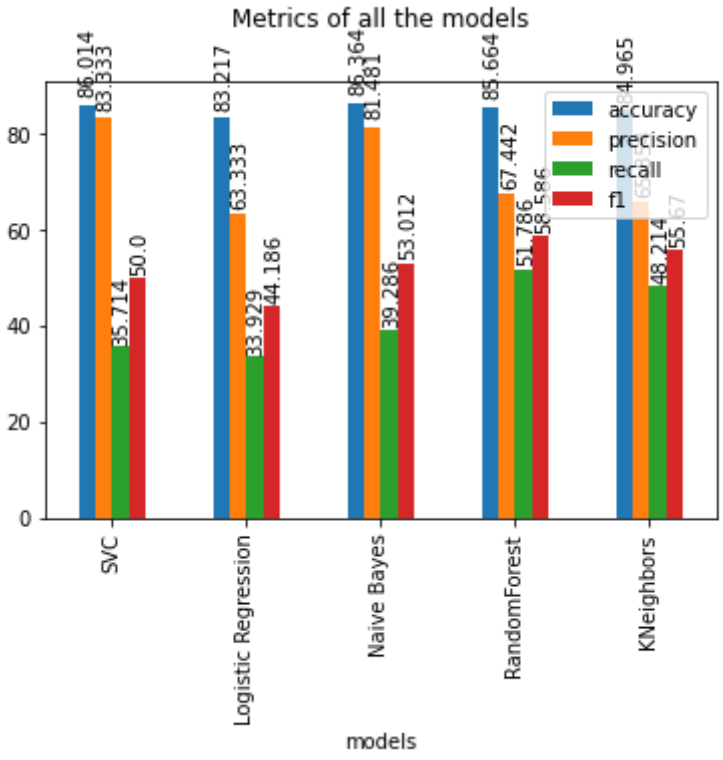


Figure 4.3. Metrics of all models when applied to a dataset with 80% non-defective and 20% defective samples.

## Experiment 4

The fourth experiment that was conducted -the one where the whole dataset was used- resulted in the metrics that are shown in Table 5 and Figure 4.4.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 97,899 | 50 | 3,39 | 6,349 |
| Logistic Regression | 97,187 | 16,667 | 8,475 | 11,236 |
| Naive Bayes | 96,296 | 21,519 | 28,814 | 24,638 |

| | | | | |
|---|---|---|---|---|
| Random Forest | 97,685 | 37,5 | 15,254 | 21,687 |
| KNeighbors | 97,792 | 38,462 | 8,475 | 13,889 |

Table 5. Metrics of all models when applied to the whole dataset.

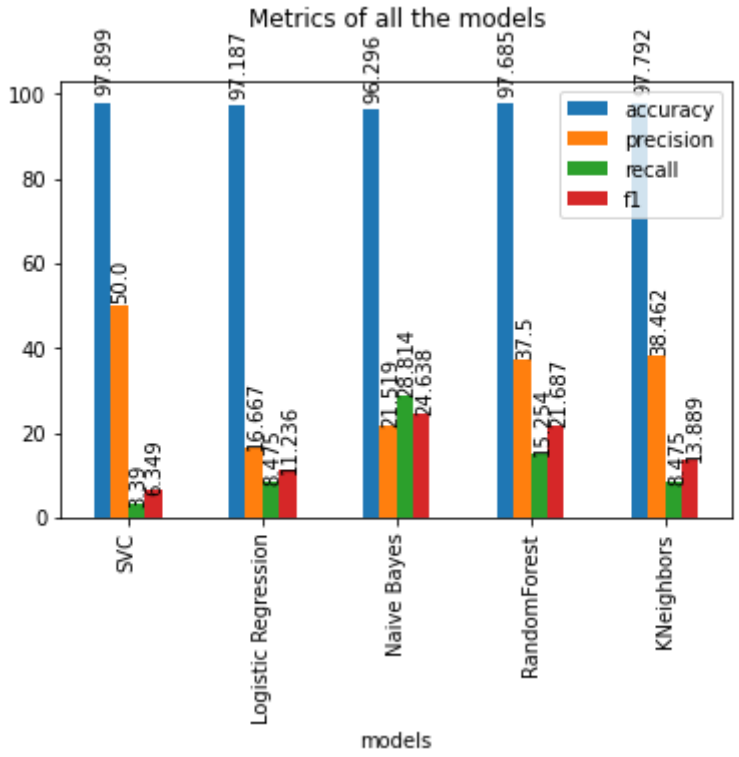


Figure 4.4. Metrics of all models when applied to the whole dataset.

In this figure is apparent that the model with the bigger accuracy and precision was the support vector machine. On the other hand, support vector machine had the smallest recall and f1 score while the naïve bayes was the model with the bigger ones.

## Experiment 5

The fifth experiment that was conducted was the one where the whole dataset was used as training data while the data from the last release 8.4.5 was used as the test data. The results of this experiment are shown in Table 6 and Figure 4.5. From the figure it is apparent that the experiment gave the less optimal results regarding the other experiments. The accuracy is high for all algorithms, but the other metrics seem to be small. The algorithm with the less optimal results in this case is support vector machine as aside from the

accuracy all the other metrics are zero. Random forest has the best accuracy and precision and naïve bayes has the best recall and f1 score.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 97,007 | 0 | 0 | 0 |
| Logistic Regression | 96,655 | 20 | 6,25 | 9,524 |
| Naive Bayes | 95,423 | 25 | 31,25 | 27,778 |
| Random Forest | 97,183 | 50 | 18,75 | 27,273 |
| KNeighbors | 97,007 | 40 | 12,5 | 19,048 |

Table 6. Metrics of all models when applied to a dataset with the last release's data as test data.

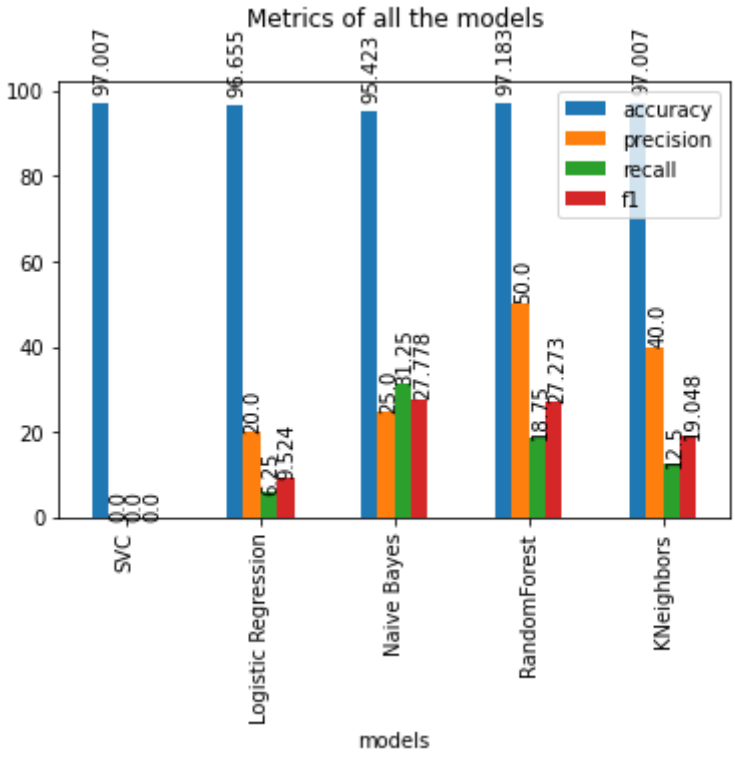


Figure 4.5. Metrics of all models when applied to a dataset with the last release's data as test data.

**Experiment 6**

The results of the sixth experiment where the test dataset were the data from the 8.4.5 release and as training data were a dataset with 90% non-defective and 10% defective samples are shown in the Table 7 and the Figure 4.6 below.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 96,831 | 33,333 | 12,5 | 18,182 |
| Logistic Regression | 95,951 | 29,412 | 31,25 | 30,303 |
| Naive Bayes | 94,718 | 20,833 | 31,25 | 25 |
| Random Forest | 96,479 | 40 | 50 | 44,444 |
| KNeighbors | 95,775 | 30 | 37,5 | 33,333 |

Table 7. Metrics of all models when applied to a dataset with the last release's data as test data and trained with 90 and 10 percent of non-defective – defective samples, respectively.

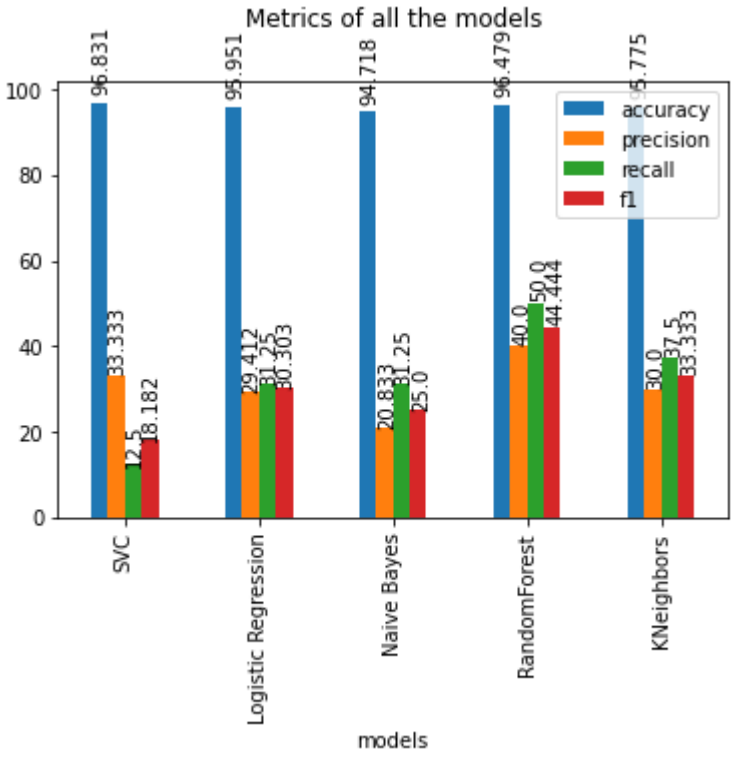


Figure 4.6. Metrics of all models when applied to a dataset with the last release's data as test data and trained with 90 and 10 percent of non-defective – defective samples, respectively.

From the above results it is obvious that support vector machine is the algorithm with the best accuracy when random forest is the algorithm with the best precision, recall and f1-

score. The least accurate results belong to naïve bayes in terms of accuracy and precision and to the support vector machine algorithm in terms of recall and f1-score.

**Experiment 7**

In the seventh experiment where the test data were the last release's data and the training data included 80% of non-defective and 20% of defective samples, support vector machine has the best accuracy while random forest has the best precision and f1-score. The best recall is the one of kNN. The results for this experiment are shown in Table 8 and Figure 4.7 below. The least accurate precision, recall and f1-score belong to logistic regression.

| | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 94,542 | 20 | 31,25 | 24,39 |
| Logistic Regression | 94,014 | 17,857 | 31,25 | 22,727 |
| Naive Bayes | 94,014 | 17,857 | 31,25 | 22,727 |
| Random Forest | 92,782 | 22,222 | 62,5 | 32,787 |
| KNeighbors | 90,845 | 18,966 | 68,75 | 29,73 |

Table 8. Metrics of all models when applied to a dataset with the last release's data as test data and trained with 80 and 20 percent of non-defective – defective samples.

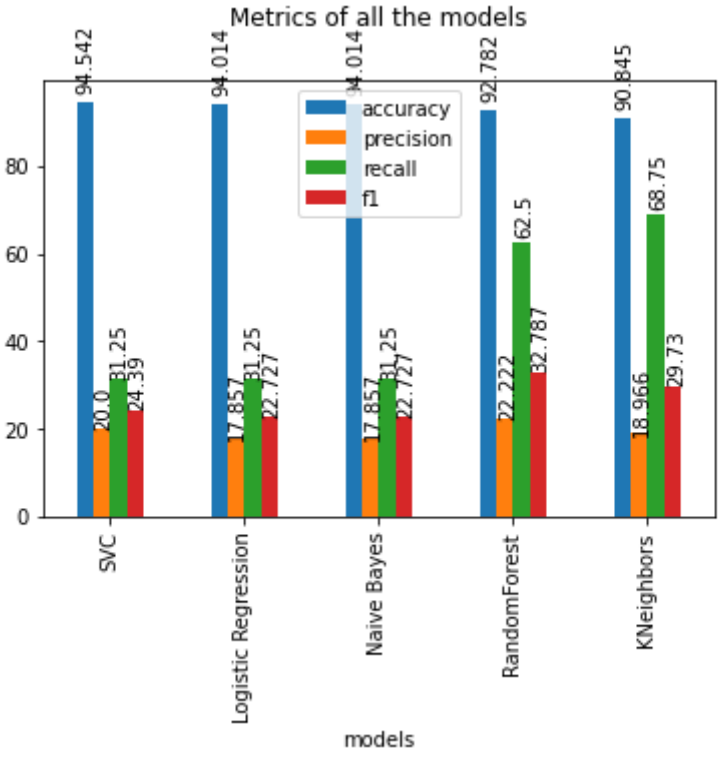

Figure 4.7. Metrics of all models when applied to a dataset with the last release's data as test data and trained with 80 and 20 percent of non-defective – defective samples.

## Experiment 8

In the eighth experiment where the test data were the last release's data and the training data included 50% of non-defective and 50% of defective samples, naïve bayes had the best accuracy, precision and f1-score when kNN had the best recall. Also, kNN algorithm had the least accurate results in terms of accuracy, precision and f1-score. The results of this experiment are shown in Table 9 and Figure 4.8 below.

| models | accuracy | precision | recall | f1-score |
|---|---|---|---|---|
| SVM | 86,444 | 10,39 | 50 | 17,204 |
| Logistic Regression | 86,092 | 12,048 | 62,5 | 20,202 |
| Naive Bayes | 92,077 | 16,279 | 43,75 | 23,729 |
| Random Forest | 82,394 | 10,377 | 68,75 | 18,033 |
| KNeighbors | 76,937 | 9,22 | 81,25 | 16,561 |

Table 9. Metrics of all models when applied to a dataset with the last release's data as test data and trained with 50 and 50 percent of non-defective – defective samples.

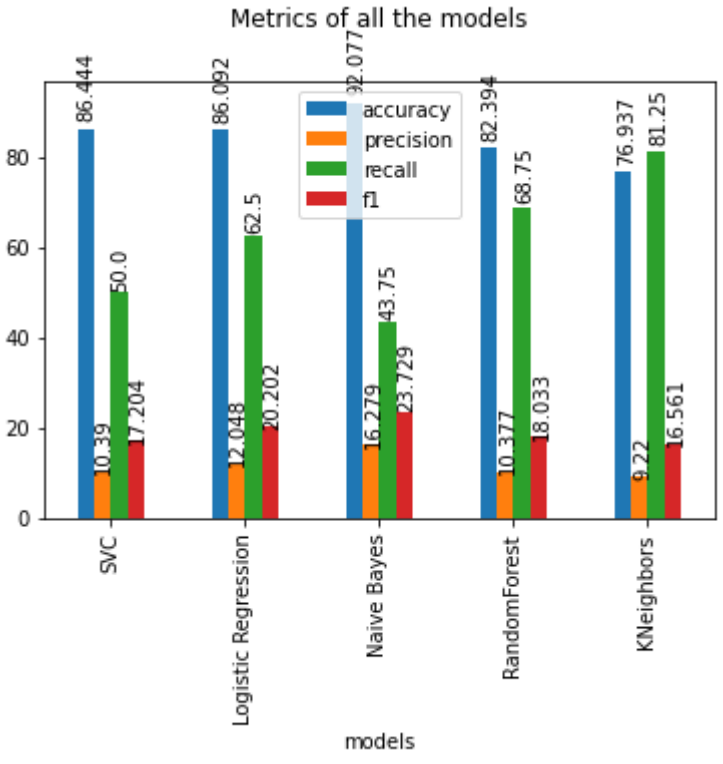

Figure 4.8. Metrics of all models when applied to a dataset with the last release's data as test data and trained with 50 and 50 percent of non-defective – defective samples.

## 4.2 Results from the models' perspective

In the previous section the results of all models are displayed grouped by the experiment in which they were created. As it is also interesting to see those results from the models' perspective, in this section the results of all models in every experiment will be shown grouped by model.

**Support vector machine**

The first algorithm that was applied in every experiment that was conducted was support vector machine (SVM). The results of this algorithm for each of the experiments in which it was used differ significantly and the evaluation metrics (accuracy, precision, recall and f1-score) of this model are shown in the Table 10 below:

| Experiments | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Experiment 1 | 80 | 86,207 | 76,923 | 81,301 |
| Experiment 2 | 91,259 | 68,75 | 19,643 | 30,556 |
| Experiment 3 | 86,014 | 83,333 | 35,714 | 50 |
| Experiment 4 | 97,899 | 50 | 3,39 | 6,349 |
| Experiment 5 | 97,007 | 0 | 0 | 0 |
| Experiment 6 | 96,831 | 33,333 | 12,5 | 18,182 |
| Experiment 7 | 94,542 | 20 | 31,25 | 24,39 |
| Experiment 8 | 86,444 | 10,39 | 50 | 17,204 |

Table 10. Metrics of Support vector machine model for all experiments

From the table 10 above it is observed that the best result of the support vector machine algorithm regarding precision, recall and f1-score was achieved in the first experiment where the data that were used consisted of 50% defective samples and 50% non-defective samples and 20% of those samples were used as test data. However, the best results

regarding the accuracy of the model were achieved during the fourth experiment where the whole original dataset was used for the training and the testing of the models.

Regarding the experiments where the dataset of release 8.4.5 was used, the fifth experiment where the whole dataset where used had the best accuracy, the sixth experiment where the dataset consists of 90% non-defective and 10% defective samples had the best precision and f1 score and the eighth experiment where the dataset consists of 50% non-defective and 50% defective samples had the best recall.

**Logistic Regression**

The second algorithm that was applied in every experiment that was conducted was logistic regression. The results of this algorithm for each of the experiments regarding the evaluation metrics (accuracy, precision, recall and f1-score) of this model are shown in the Table 11 below:

| Experiments | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Experiment 1 | 80,87 | 90,566 | 73,846 | 81,356 |
| Experiment 2 | 91,958 | 72,727 | 28,571 | 41,026 |
| Experiment 3 | 83,217 | 63,333 | 33,929 | 44,186 |
| Experiment 4 | 97,187 | 16,667 | 8,475 | 11,236 |
| Experiment 5 | 96,655 | 20 | 6,25 | 9,524 |
| Experiment 6 | 95,951 | 29,412 | 31,25 | 30,303 |
| Experiment 7 | 94,014 | 17,857 | 31,25 | 22,727 |
| Experiment 8 | 86,092 | 12,048 | 62,5 | 20,202 |

Table 11. Metrics of Logistic Regression model for all experiments

From the table 11 above it is observed that the best result of the logistic regression algorithm regarding precision, recall and f1-score was achieved in the first experiment where the data that were used consisted of 50% defective samples and 50% non-defective samples and 20% of those samples were used as test data. However, the best results regarding the accuracy of the model were achieved during the fourth experiment.

Also, similarly with the SVM among the experiments that use the dataset of release 8.4.5 as test data, experiment 5 had the best accuracy, experiment 6 had the best precision and f1-score and experiment 8 had the best recall.

**Naive Bayes**

Every experiment that was conducted used naive bayes as the third algorithm. The evaluation metrics (accuracy, precision, recall, and f1-score) for this model are presented in Table 12 below. It is observed that the outcomes of this approach for each experiment in which it was applied vary greatly.

| Experiments | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Experiment 1 | 68,696 | 87,179 | 52,308 | 65,385 |
| Experiment 2 | 90,734 | 54,545 | 32,143 | 40,449 |
| Experiment 3 | 86,364 | 81,481 | 39,286 | 53,012 |
| Experiment 4 | 96,296 | 21,519 | 28,814 | 24,638 |
| Experiment 5 | 95,423 | 25 | 31,25 | 27,778 |
| Experiment 6 | 94,718 | 20,833 | 31,25 | 25 |
| Experiment 7 | 94,014 | 17,857 | 31,25 | 22,727 |
| Experiment 8 | 92,077 | 16,279 | 43,75 | 23,729 |

Table 12. Metrics of Naïve Bayes model for all experiments

The first experiment, in which the data utilized consisted of 50% defective samples and 50% non-defective samples and 20% of those samples were used as test data, produced the best results for the naive bayes algorithm in terms of precision, recall, and f1-score. However, the fourth experiment where the whole original dataset was used produced the best outcome in terms of the model's accuracy.

Regarding the experiments 5-8 where the data of the last release where used as test data, experiment 5 had the best accuracy, precision and f1-score, while experiment 8 had the best recall.

**Random Forest**

The fourth method, random forest, was also employed in all experiments. Table 13 below lists the model's evaluation metrics (accuracy, precision, recall, and f1-score). For each

experiment in which this method was used, it has been noticed that the results varied substantially.

| Experiments | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Experiment 1 | 79,13 | 83,607 | 78,462 | 80,952 |
| Experiment 2 | 90,734 | 53,488 | 41,071 | 46,465 |
| Experiment 3 | 85,664 | 67,442 | 51,786 | 58,586 |
| Experiment 4 | 97,685 | 37,5 | 15,254 | 21,687 |
| Experiment 5 | 97,183 | 50 | 18,75 | 27,273 |
| Experiment 6 | 96,479 | 40 | 50 | 44,444 |
| Experiment 7 | 92,782 | 22,222 | 62,5 | 32,787 |
| Experiment 8 | 82,394 | 10,377 | 68,75 | 18,033 |

Table 13. Metrics of Random Forest model for all experiments

From the results of the above table, it is apparent that random forest such as the previous models has a better accuracy score in the fourth experiment and a better score regarding the precision, the recall and the f1-score in the first experiment. The first experiment is the one where the data utilized consisted of 50% defective samples and 50% non-defective samples and 20% of those samples were used as test data. The fourth experiment was the experiment where the whole dataset was used and 80% of this dataset were used to train the models and 20% of it were used in order to test them.

From the results of the experiments where the test dataset is the data of the last release, it is observed that experiment 5 had the best accuracy and precision, experiment 6 had the best f1-score and experiment 8 had the best recall.

**k-nearest neighbors**

In all experiments, the fifth and final technique that were used is kNN. Metrics for evaluating the model are listed in Table 14 below. It has been noted that the outcomes for every experiment in which this method was applied differed greatly.

| Experiments | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Experiment 1 | 82,609 | 80,822 | 90,769 | 85,507 |
| Experiment 2 | 90,909 | 56,25 | 32,143 | 40,909 |

| | | | | |
|---|---|---|---|---|
| Experiment 3 | 84,965 | 65,854 | 48,214 | 55,67 |
| Experiment 4 | 97,792 | 38,462 | 8,475 | 13,889 |
| Experiment 5 | 97,007 | 40 | 12,5 | 19,048 |
| Experiment 6 | 95,775 | 30 | 37,5 | 33,333 |
| Experiment 7 | 90,845 | 18,966 | 68,75 | 29,73 |
| Experiment 8 | 76,937 | 9,22 | 81,25 | 16,561 |

Table 14 Metrics of kNN model for all experiments

From the findings in the table above, it is clear that kNN, like the previous models, performed better in terms of accuracy in the fourth experiment and precision, recall, and f1-score in the first experiment.

Among the experiments where the data of release 8.4.5 were used as test data, experiment 5 had the best accuracy and precision, experiment 6 had the best f1-score and experiment 8 had the best recall.

# 5 Discussion

The results of the experiments that were conducted shows that there is not one ML algorithm that has given the best result in all four metrics in any of the conducted experiments. However, there were experiments where there was one algorithm that had three out of the four metrics better that the other models. Those models are kNN in terms of accuracy, recall and f1-score in Experiment 1, Random Forest in terms of precision, recall and f1-score in Experiment 6 and Naïve Bayes in terms of accuracy, precision and f1-score in Experiment 8. It is apparent that even in the experiments that one algorithm has the best results in the majority of the metrics it is not the same algorithm in more than one experiment.

Moreover, the results of this dissertation show that the SVM model is the one that is affected the most by the major imbalance of the non-defective and the defective samples in the dataset as it seems from the results of SVM in the experiment that the whole dataset was used. In this experiment the results of SVM were significantly worse that those of the other models.

Similarly with the results of the work by Perreault *et al* in this dissertation there is not one classifier that gives the best results in all the experiments but also the is not one classifier that gives the worst.

The current dissertation has many similarities with the work by Perreault *et al*, but it also has some differences. The most significant difference is regarding the datasets used. The datasets used in the study by Perreault *et al.* were from the NASA metrics data program repository while the dataset in this dissertation have been extracted from the source code of an existing program and instead of comparing the results of different datasets, in this dissertation variations of the same dataset were used [4].

## 5.1 Threats to validity

One threat to validity is the metrics that were used to evaluate the results of the models in each experiment. The metrics that were used in this dissertation are accuracy, precision, recall and $F_1$-score. Those metrics are commonly used to evaluate models that were used

in software defect detection. However, there are also other metrics that could have been used in the same case, some of them are $F_2$-score and $F_{0.5}$-score.

Another threat to validity is the admission that all the files that are part of the commit that has in the title the word “fix” are defective. Although this is a valid admission it is possible that not all of the files that are involved in these commits are defective. However, even in this case as the purpose is to find the defective files in the application it is better to have classify a file as defective even if it is not than classified it as non-defective in case it is defective.

# 6 Conclusions and further work

## 6.1 Conclusions

In this dissertation various ML techniques have been applied to a dataset extracted from the source code of notepad++. This dataset was used in eight experiments. The ML algorithms used in all experiments were SVM, Naive Bayes, Random Forest, kNN and logistic regression. Those experiments differ regarding the consistency of defective and non-defective samples in the training dataset as well as regarding the data that were used as a test dataset. For the evaluation of the ML models in each of those experiments four metrics were used. Those metrics were accuracy, precision, recall and f1-score. From the results of those experiments, it is apparent that none of those algorithms has given the best results regarding all the metrics for all experiments, but in each experiment, there is one that is better regarding one or more metrics. What can be concluded from all of those experiments is that the consistency of the training data in defective and non-defective results has a major impact on the results of all models. Also, the best results for all metrics seem to be in the experiment where the dataset that was used consisted of 50% defective and 50% non-defective results and where 80% of this dataset was used as training data and 20% of it as test data.

Moreover, regarding the results from the models' perspective, it can be concluding that all the ML models that were used in this dissertation have their best accuracy score in the fourth experiment. This is the experiment where the whole dataset was used and 80% of that was used for the training and 20% for the testing. In addition, all the models had their best precision, recall and f1-score in the first experiment. The first experiment is the one where the dataset used consisted of equal number of defective and non-defective samples when 20% of them were used for testing the models.

More generally, it is observed that all the models have better results in the experiments where 80% of the dataset is used as training data and 20% of the dataset as test data. Also, in those experiments the accuracy for all the models is better in the Experiment 4 where the whole dataset was used, while precision, recall and f1-score are better in Experiment

1 where the dataset that was used consisted of 50% non-defective and 50% defective samples.

## 6.2 Further work

There are several ways that this dissertation can be expanded in the future. Firstly, it would be interesting to see the results of the same models with other hyperparameters in order to examine if a better result can be achieved. Also new models can be used in the same variations of the dataset and the produced results can be compared to the existing ones.

Moreover, other experiments can be added where the dataset that will be used will consist of other combinations of defective and non-defective samples of the existing dataset. These experiments can be conducted with the same models that were used in this dissertation or in all these experiments more models can be added. For example, two more models that could be added are Neural Network and Decision tree algorithms.

Furthermore, the dataset used can be expanded by adding data from more versions of notepad++ or data from another program can be used with the same techniques in order to see how the change of the dataset impacts the results of the models used in this dissertation.

Finally, using the results of this dissertation another research can be conducted one where the input features that are more likely to determine the result can be found.

# Bibliography


[1] Antonellis, P., Antoniou, D., Kanellopoulos, Y., Makris, C., Theodoridis, E., Tjortjis, C., & Tsirakis, N. (2009). Clustering for Monitoring Software Systems Maintainability Evolution. *Electronic Notes in Theoretical Computer Science*, *233*, 43-57.

[2] Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, *40*(6), 603-616.

[3] Arshad, S., & Tjortjis, C. (2016, May). Clustering software metric values extracted from c# code for maintainability assessment. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence* (pp. 1-4).

[4] Perreault, L., Berardinelli, S., Izurieta, C., & Sheppard, J. (2017, October). Using classifiers for software defect detection. In *26th International Conference on Software Engineering and Data Engineering* (pp. 2-4).

[5] Ren, J. H., & Liu, F. (2019). Predicting Software Defects Using Self-Organizing Data Mining. IEEE Access, 7, 122796-122810.

[6] Prasad, M. C., Florence, L., & Arya, A. (2015). A study on software metrics based software defect prediction using data mining and machine learning techniques. International Journal of Database Theory and Application, 8(3), 179-190.

[7] Jakkula, V. (2006). Tutorial on support vector machine (svm). *School of EECS, Washington State University*, *37*(2.5), 3.

[8] Zhang, H., & Su, J. (2004, September). Naive bayesian classifiers for ranking. In *European conference on machine learning* (pp. 501-512). Springer, Berlin, Heidelberg.

[9] Zhang, H. (2004). The optimality of naive Bayes. *Aa*, *1*(2), 3.

[10] Nick, T. G., & Campbell, K. M. (2007). Logistic regression. *Topics in biostatistics*, 273-301.

[11] Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, *25*(2), 197-227.

[12] Jiang, L., Cai, Z., Wang, D., & Jiang, S. (2007, August). Survey of improving k-nearest-neighbor for classification. In *Fourth international conference on fuzzy systems and knowledge discovery (FSKD 2007)* (Vol. 1, pp. 679-683). IEEE.

[13] Space, T. (2006). J. Han and M. Kamber, Data Mining: Concepts and Techniques. *Morgan Kaufmann*, *9*, 918-929.

[14] «Precision» [Online]. Available: https://c3.ai/glossary/machine-learning/precision/

[15] «Precision and Recall in Machine Learning» [Online]. Available: https://www.javatpoint.com/precision-and-recall-in-machine-learning

[16] «Understanding the Confusion Matrix and How to Implement it in Python,» [Online]. Available: https://towardsdatascience.com/understanding-the-confusion-matrix-and-how-to-implement-it-in-python-319202e0fe4d

[17] Tjortjis, C. (2020). Mining Association Rules from Code (MARC) to support legacy software management. Software Quality Journal, 28(2), 633-662.

[18] Papas, D., & Tjortjis, C. (2014, May). Combining clustering and classification for software quality evaluation. In Hellenic Conference on Artificial Intelligence (pp. 273-286). Springer, Cham.

[19] Kanellopoulos, Y., Antonellis, P., Tjortjis, C., Makris, C., & Tsirakis, N. (2011). k-Attractors: a partitional clustering algorithm for numeric data analysis. Applied Artificial Intelligence, 25(2), 97-115.