Alma Mater Studiorum · University of Bologna

Department of Statistical Sciences
Master's degree in Statistical Sciences (Data Science)

Spectral and neural gas clustering techniques for software defect prediction: performance evaluation with different feature selection methods

Master's Thesis

Student:
Gianluca Bertaccini

Supervisor:
Elisabetta Ronchieri

Co-supervisor:
Marco Canaparo

March session
Academic Year 2020-2021

I would like to thank my family and friends for their support.
I would also like to thank my supervisor, Elisabetta Ronchieri, and co-supervisor, Marco Canaparo, for their continuous guidance and advice: their constant presence, eagerness to share their knowledge and to include me in their process of doing research, have all taught me a lot and have been vital for the writing of this dissertation.
This thesis job was supported and carried on with INFN - CNAF, the National Center for informatics and telematics of the Italian National Institute for Nuclear physics.

Abstract

Developing and revising software requires time and resources. Different machine learning models are able to classify in advance instances of code as defective or clean, by analyzing datasets containing software metrics (computed directly from the source code). Correct identification helps developers focus their efforts on fixing those defective instances and save the amount of time and resources spent on revising software. Furthermore, there is a greater need for unsupervised prediction models rather than supervised ones, because many software defects datasets are incomplete or unlabeled. Current research is still trying to explore unsupervised methods and trying to optimize their performance.

In this thesis, different feature selection and feature construction approaches are applied to software defect prediction models, in order to see which ones influence positively the performance of these models. The models compared are either supervised classifiers (i.e. Naive Bayes and Random Forest) or unsupervised methods (i.e. Neural Gas and Spectral Clustering): both types are used for the same purpose, which is predicting if an instance of code is defective. The performances of the models are studied and compared by taking into consideration the different software metrics selected by the feature selection approaches, and by using nonparametric tests. According to the results, Spectral Clustering on the complete software defects datasets performs slightly better than Naive Bayes and Random Forest, where features have been selected through stepwise selection methods. The Spectral Clustering reaches an average area under the curve (receiver operating characteristic curve) of 0.66, while the Naive Bayes reaches 0.65. In conclusion, the Spectral Clustering performs similarly to the supervised method.

Contents

Introduction

In software development it is of paramount importance to identify which instances of code are defective in order to allow developers to focus their efforts on revising this code. Many approaches have been tried to correctly identify defective instances [10]. Many of these approaches rely on the analysis of different software metrics [22], which are computed directly from the source code of a software project [11]. Defective instances identification conducted through supervised machine learning methods is the best approach for now [26]. However, many software metrics datasets are better suited for unsupervised methods [12], because they often lack (completely or in part) historical labeled data to correctly train the supervised machine learning algorithms. For this reason, an important problem in software defect prediction consists in finding unsupervised methods that perform as well [43], or even better, as the best supervised models.

Software metrics often are highly correlated between each other [27], and that can influence negatively the performance of the defect prediction models. Therefore, applying the correct feature selection approaches can reduce collinearity: different methods of feature selection and feature construction are applied and compared, when running the prediction algorithms.

This thesis aims at evaluating the performance of two unsupervised methods (i.e. Neural Gas [20] and Spectral Clustering [43]) and two supervised methods (i.e. Naive Bayes [35] and Random Forest [13]) in their ability to correctly identify defective instances of code. Furthermore, it compares the application of these methods with different types of feature selection [23], in order to see which feature selection approach is better to achieve high performance of the algorithms. In conclusion the Spectral Clustering on the complete datasets performs slightly better than Naive Bayes and Random Forest (where the features have been chosen through stepwise selection methods), while Neural Gas Clustering performs significantly worse than the others. Additionally, feature selection approaches do not influence significantly the performance of the machine learning methods, except for principal components analysis which gives the worst results.

Chapter 1, Related Works, discusses the current research on software defect prediction, from the supervised methods to the unsupervised ones, and explains what are the most used methods of comparison of different classifiers. It relates the available research up to now with the development of this thesis, and shows how focusing the research on feature selection methods and unsupervised methods is important.

Chapter 2, Methodology, describes the steps followed to deal with the problem of the thesis. The adopted approach is characterized by four steps detailed from Chapter 3 to Chapter 6. This chapter shows a reference framework that serves as a baseline for the whole research, and it is only a brief summary containing the essential elements that will be explained in more detail in the later chapters.

Chapter 3, Data, talks about the nature of the data and the way different metrics have been computed. The data used by the models is composed of different software metrics and computed directly from the source code of open source software projects. The Chapter also includes an exploratory analysis of the available datasets, which highlights the fact that the data is highly skewed and therefore, in certain situations, will require nonparametric tests.

Chapter 4, Feature Selection, details the importance of variable selection (in this case selection of software metrics to predict defective classes) and compares different feature selection methods (two stepwise selection methods [23] and LASSO - least absolute shrinkage and selection operator [40]) and a feature construction method (PCA - principal components analysis [25]). A detailed explanation of these methods and the reasoning behind their use for the purpose of this thesis are provided.

Chapter 5, Supervised and Unsupervised Methods, explores the application of different models for software defect prediction: Naive Bayes [35], Random Forest [13], Neural Gas Clustering [20], Spectral

Clustering [43]. The last two methods are compared to the first two, which are some of the most common and better performing methods available. Therefore, Naive Bayes and Random Forest serve as a baseline for comparison with the other methods. Every model is applied using the different feature selection methods described above.

Chapter 6, Comparison of Classifiers, contains an evaluation of the performance of every model with different selected features, through the most common evaluation metrics in the field of software defect prediction: accuracy, F-score, area under the curve (AUC) of receiver operating characteristic (ROC), specificity, precision, recall. The difference between these metrics is compared by using a nonparametric test, the Friedman test, and in case of rejection of the null hypothesis, the Nemenyi test [31] for multiple comparisons.

Chapter 7, Conclusion, explores the results obtained through this research. In particular, it focuses on the performance of the unsupervised methods with respect to the different feature selection methods, and with respect to the baseline supervised methods considered.

Chapter 1

Related Works

This section presents an overview of the previous and recent literature on software defect prediction focusing on two main issues: the collinearity of software metrics and the performance evaluation of supervised and unsupervised ML (machine learning) algorithms. In the end, some comments on the analysed studies are provided.

1.1 Dealing with collinearity between software metrics:
How current researchers are dealing with high correlation between software metrics. Overview of the main approaches to solve this problem.

1.2 Studying the performance of supervised and unsupervised algorithms:
How different software defect prediction models perform according to different evaluation measures. Shift from supervised to unsupervised methods and reasons why it is happening in the current research.

1.3 Comments on related work:
Why the papers selected are influential for the field of software defect prediction in general, and for the purpose of this thesis. Main conclusions drawn by reading these papers.

1.4 New approaches introduced
Differences between the new methods introduced in the present thesis and the ones used in the current literature. How the present thesis contributes to research in software defect prediction.

1.1 Dealing with collinearity between software metrics

Many machine learning (ML) methods for classification assume independence of the predictors. Therefore, classifiers applied to datasets where the independent variables are highly correlated can give wrong results. Many researchers have tried to deal with this problem either by transformation of the variables or by feature selection and construction (principal components analysis - PCA - for example), in order to reduce correlation.

Al Mamun et al. [27] identify software metrics by their type of measurement: cumulatively, organic, density and average. By studying the Kendall's tau-B correlations (since the software defect data is highly skewed, a Pearson correlation would not give plausible values) and by comparing values using different nonparametric tests (mainly Mann-Whitney U test, Wilcoxon signed rank test and paired-samples sign test), the research shows how variables measured cumulatively are highly correlated between themselves (average of 0.79). Instead the variables measured by density, average and organically have a lower correlation between themselves. The paper makes two important observations: cumulative measurements are a high source of collinearity between variables, and correlations between variables that come from different measurement type are very low. Therefore the researchers propose that transforming a feature from its original measurement type to another can produce a new non-collinear feature, that can help making good predictive modeling.

Malhotra et al. [26] use univariate Logistic Regression to study the relation between many independent variables (from CK, MOOD and QMOOD software metrics suites) and the response variable (label representing the presence or absence of defects with 1 or 0). The CK metrics are the ones defined by Chidamber and Kemerer [15], while the MOOD metrics are the "Metrics for Object Oriented Design". This way, the researchers are able to keep only the variables for which the coefficient is statistically significant. Then they apply the forward stepwise selection method on the significant variables, to select only the important ones before applying the ML classifiers.

D'Ambros et al. [16] perform principal component analysis in order to avoid the problem of collinearity between independent variables. Before predicting the response variable (containing the number of defects) using generalized linear models, the researchers apply PCA and select the components that explain at least 95% of the variability. By doing this, they're not actually fitting models using the original software metrics, but they're using principal components (PC) which are independent.

Esteves et al. [18] show that on the Jureczko datasets for defect prediction, the best number of features is never the full pool of independent variables, because the best performing models are composed by up to 3 features. Their study is done by computing SHAP (SHapley Additive exPlanations) values for each feature. SHAP represents the impact of a feature on the prediction model output.

1.2 Studying the performance of supervised and unsupervised algorithms

Researchers have studied the performance of many classifiers, both supervised and unsupervised, in order to predict the presence or absence of defects in software metrics datasets.

Esteves et al. [18] work on the Jureczko datasets, consisting mainly of CK and other object-oriented software metrics. The researchers study the most common supervised classifiers such as Logistic Regression, Naive Bayes, K-Nearest Neighbours, Neural Network, Support Vector Machine, Decision Trees and Random Forest, and compare them to a tree boosting algorithm (XGBoost) using the AUC (area under the curve) and F-score values. They also build an optimized version of XGBoost, called US-XGBoost, and show how this optimized algorithm performs better than the other supervised methods.

Malhotra et al. [26] compare statistical and supervised machine learning methods using accuracy, specificity, sensitivity (recall) and AUC values. These methods are applied to software metrics that come from Java open source projects. The methods are multivariate Logistic Regression, Random Forest, AdaBoost, Bagging, Multilayer Perceptron, Support Vector Machine and Genetic Programming. By comparing the results, the researchers are able to find that Random Forest and Bagging perform better than the other methods, because they reach AUC values of 0.875 and 0.876 respectively.

Yan et al. [41] investigate the performance of fault prediction models at the file level. The data is taken from 10 projects of the PROMISE data repository, and the metrics considered are CK and other object-oriented metrics, divided into five categories: complexity, coupling, cohesion, abstraction and encapsulation. Many supervised models are applied, from the most common ones such as Random Forest, AdaBoost, Bagging, Naive Bayes, Logistic Regression, to others less common in software defect prediction (Radial Basis Function Network, Rotation Forest, Random Subspace and Effort-Aware Linear Regression, known as EALR). The results of these supervised models are compared to some unsupervised ranking models (that rank files as more or less defect prone), and the results are used both in within-project and cross-project settings. According to the results (given by values of ACC and P(opt), optimized performance indicator - see [29]), in the within-project setting the EALR, which is a supervised method, performs better than everything else. Therefore ranking unsupervised models considered in the paper do not perform better than the best supervised models in the within-project case. Instead, it is shown that in a cross-project setting, ranking unsupervised models can perform better than the best supervised models such as EALR.

Yan et al. [42] show how the introduction of unsupervised models in software defect prediction could be really important, because it happens often that the available data is not labeled and so being able to rank or classify classes as defective-prone without the label would be a big step forward in the research. In this paper, they build simple unsupervised models on software change metrics, and compare them to supervised models using ACC (a measure similar to recall) and P(opt) as evaluation measures. The supervised models considered are divided into six categories: "function" (regression models and neural networks), "lazy" (models based on lazy learning), "bayes", "rule" (rule based models), "tree" (models based on decision trees) and "ensemble" (models built with multiple base learners). According to the results, it is shown that sometimes simple unsupervised models (that rely only on a few selected independent variables) can perform better than most supervised models.

Boucher et al. [12] introduce the HySOM (Hybrid Self Organizing Map) algorithm (unsupervised) and compare it to Naive Bayes Network, Artificial Neural Network and Random Forest. The algorithm is tested on 12 public datasets containing software metrics. The HySOM is a threshold based approach: for every metric, a threshold value has been computed either using ROC curves or Alves rankings. ROC curves thresholds are found by maximizing the ROC (receiver operating characteristic) curve where 1-specificity and sensitivity are the highest. Alves rankings is better to compute thresholds when label data is not available, otherwise ROC curves are better. When three or more of the six metrics exceed the threshold values, then the class is considered fault-prone. The evaluation metrics considered are error rate, false positive rate, false negative rate and g-mean (which is given by $\sqrt{TPR \cdot TNR}$ (where TPR is true positive rate and TNR is true negative rate). The SOM (self organizing map) is a clustering algorithm that performs better than K-Means. By turning it into HySOM, it also makes use of the thresholds and performs better than the original. In the paper it is shown that the HySOM performs better than three common supervised algorithms for defect prediction: Naive Bayes (NB), Artificial Neural Network (ANN) and Random Forest (RF).

Nam et al. [30] propose the use of CLA ("Clustering and LAbeling") and CLAMI ("Clustering, LAbeling, Metric selection, Istance selection"), which are two unsupervised approaches for defect prediction on unlabeled datasets. The researchers study the performance of both methods by computing F-measure and AUC (area under the curve). The approaches follow these steps: clustering of the istances (which can be performed by K-Means or by other clustering algorithms), labeling of the instances in the clusters (according to the magnitude of their metrics), metric selection (selection of only the necessary variables) and instance selection (selection of observations for training models). CLA only has the first two steps of clustering and labeling, while CLAMI follows all the steps. These approaches are proposed because in software defect prediction there are many situations where historical data is missing, and so it is necessary to find a way to label classes (instances) as defect prone or not. The CLA and CLAMI approaches are compared to a supervised learning (SL) method (in this case Logistic Regression), a threshold based approach (THD) and an expert based approach (EXP). Their values of F-measure and AUC are compared using the Friedman test. Overall, CLA and CLAMI perform similar to SL and EXP, but outperform the threshold based approach (THD). The researchers also compare CLA and CLAMI with common supervised algorithms for defect prediction: Bayes Network (BN), J48 Decision Tree, Logistic Model Tree, Logistic Regression, Naive Bayes, Random Forest and Support Vector Machine. They use Friedman test and then Nemenyi test for multiple comparisons. It is shown that CLA and CLAMI perform as well as most of these supervised algorithms.

Yan et al. [42] apply a self-learning method, CLAMI, to predict change-prone classes. It also extends CLAMI by introducing CLAMI+, which performs better on average. The introduction of this method advances the research of software defect prediction when historical data is limited or when the datasets are unlabeled. Different object-oriented metrics have been considered along with some others, and supervised learning approaches (Bayesian Networks, Neural Networks, Multivariate Regression and Ensemble methods) are compared to CLAMI and CLAMI+. The datasets considered are 14, coming from different open source projects. In CLAMI+, the clustering phase is extended by transforming the 1 or 0 result (which indicates if an observation has a violation or not) to a continuous value from 0 to 1 (which represents the violation degree for every observation). Since the violation degree is used in CLAMI+, also the next phase (labeling) is going to give different results than CLAMI. Therefore, also the phase of metric selection will be different (metrics are selected according to their MVS, metric violation score). The researchers find that a self-learning method has an advantage over typical prediction methods because it does not need historical labeled dataset. Both CLAMI and CLAMI+ outperform on average the supervised methods considered, according to CCR (correct classification rate) and AUC (area under the curve). They also show that, according to the same results, CLAMI+ has a slightly better performance than CLAMI in all of the datasets considered except for one.

Zhang et al. [43] examine two types of unsupervised classifiers, distance-based and connectivity-based. For connectivity-based classifier, Spectral Clustering is proposed and it is compared to the other unsupervised and unsupervised methods. The data comes from 26 projects of publicly available datasets. In general, it is shown that unsupervised classifiers (especially distance-based ones, such as K-Means) perform worse than supervised classifiers. Spectral Clustering makes an exception, because it performs as well as supervised classifiers. Especially in a cross-project setting, spectral clustering is one of the best classifiers (achieving 0.71 AUC). Instead in a within-project setting, Random Forest is shown to be the best classifier (according to AUC values), while Spectral Clustering is slightly worse along with other common supervised methods (Logistic Regression, Logistic Model Tree, Naive Bayes). The comparison between methods is made with the AUC values and using the Scott-Knot test, which ranks the evaluated classifiers into different tiers.

1.3 Comments on related work

As derived from previous literature, it is important to correctly choose the metrics to apply on a classifier. Most publicly available datasets on software defects contain CK and other object-oriented metrics, often divided into categories such as complexity, coupling, cohesion and abstraction. Many approaches have been tried to select or build the correct features in order to have the best performing models, from variable transformation to stepwise selection or principal components analysis. It is essential to reduce correlation between features, otherwise the prediction models will not give good results (since most of them assume the independence of features between each other).

Besides the problem of feature selection, researchers have tried to find the best models to predict defective classes (or files), both supervised and unsupervised. In general, there has been a shift from supervised models to unsupervised ones, because software metrics datasets are often unlabeled and lack historical data. A good unsupervised model (that does not need labels) will be more useful than a good supervised one, because it allows researchers to classify new software metrics datasets without the need of historical data. That is why many unsupervised models have been studied lately, such as HySOM, CLAMI+ or Spectral Clustering. These are the kinds of models worth exploring in software defect prediction.

1.4 New approaches introduced

In the present thesis, the problem of collinearity between software metrics is studied by trying out different feature selection methods on the same datasets: backward and forward stepwise regression, LASSO (least absolute shrinkage) and PCA (principal components analysis). In particular, the introduction of LASSO for feature selection is a new approach that has to be studied and could help select the right features for better performing defect prediction models (both supervised and unsupervised).

In the current thesis the shift from supervised to unsupervised models is highlighted, and expanded to include a comparison of two unsupervised methods newly introduced in software defect prediction (Neural Gas Clustering and Spectral Clustering), with some of the most common supervised methods (Naive Bayes and Random Forest). Furthermore, the comparison is made between the performance of these methods and their feature selection methods, which allows to see which feature selection is better and which metrics are the most important for defect prediction.

Thus, this thesis extends the current research in software defect prediction by studying in depth the relationship between new unsupervised methods and feature selection methods, to see how different metrics selected influence the performance of prediction and classification models.

Chapter 2

Methodology

This Chapter shows the steps that are followed in the thesis to deal with the problem of software defect prediction. Each step is described in the following Chapters and developed through the usage of the R language. The Eclipse datasets are available at [1] under the heading "CK and other 11 object oriented metrics for the last version of the system plus categorized (with severity and priority) post-release defects", and referenced in [16]. The NASA datasets are available at [2] and referenced in [37]. The R code to reproduce the results is available at [3] and the datasets modified for the purpose of this thesis are available at [4] and [5]. For more information about the R functions specifically developed for this thesis, see the Appendix, Section 8.7.

2.1 Methodology description

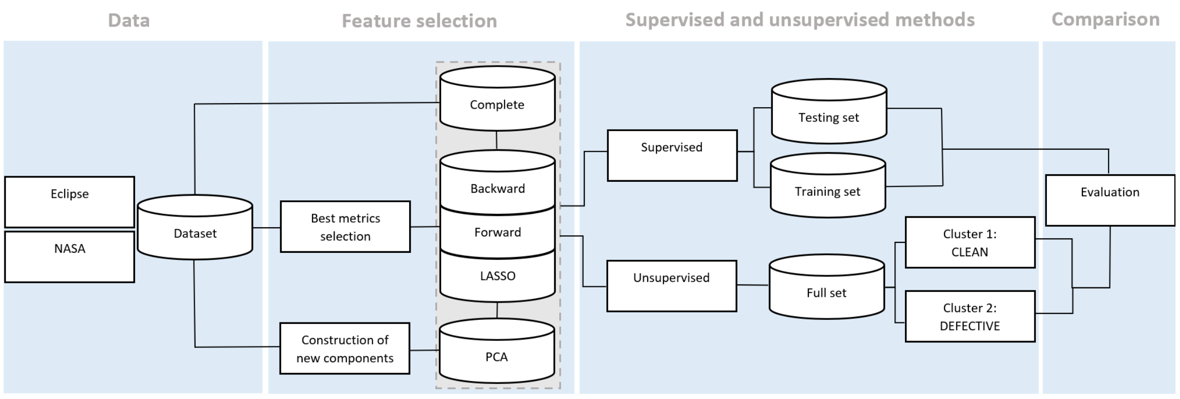


Figure 2.1: Methodology used in this thesis

Figure 2.1 summarizes the steps followed in this thesis. Each step is enveloped in a block from left to right.

Data, the first step, includes a qualitative and quantitative exploratory analysis of the available datasets, studying the distributions of the data, and the number of defective classes with respect to the number of non-defective classes. The Shapiro-Wilk test has been used to test the normality of data that shows not to follow a Gaussian distribution. Therefore, Spearman or Kendall correlations have been used to compute correlations between features. The datasets are 17 in total, 5 coming from the Eclipse project (or related projects) [16] and 12 coming from NASA [37] software defects repository. The related projects and software metrics are described in Chapter 3. The analysis of the metrics and their distributions, along with their correlations, shows how the datasets can benefit from a reduction in features and therefore why the next step, feature selection, is so important.

The feature selection step contains all the methods used and compared in this project. Two methods are stepwise variable selection: the backward and forward directions of the algorithms are considered, in order to have an exhaustive overview of the performance of stepwise methods. Then, LASSO [40] (least absolute shrinkage and selection operator) with respect to logistic regression is introduced. The original version of LASSO used with linear regression models is not considered, since the response variable in the available datasets is always binary (1 or 0, depending if the observed class is defective or not). This type of variable selection has not been used before in software defect prediction, and is introduced and studied because it can be an improvement over the current methods. Finally, a feature construction method is presented, PCA [25] (principal components analysis). By far the most used method for building components and reducing collinearity used in this field, it is compared to the first three. Overall, 17 datasets will be produced for each of the four feature selection methods. Considering also the complete datasets (which included in the comparison of the research), after this step the datasets available are 85 in total.

The next step contains a detailed explanation of the supervised methods and unsupervised methods used in this research. Naive Bayes [35] and Random Forest [23] are two common supervised classification methods that have good performance in software defect prediction. Neural Gas [20] and Spectral clustering [28] are two unsupervised methods (spectral clustering is the newest introduced in this field) that are used to classify classes are defective or not without the need of a response variable (a label indicating 1 or 0). That is why, in this project, these algorithms divide the observations only in two clusters, one containing the defective classes and the other one containing the faultless ones. In total, these 4 methods (two supervised and two unsupervised) are applied to classify the 85 datasets available, allowing to compare the classification methods and the feature selection methods too.

In the last step, the comparison of classifiers is made through the evaluation of different methods for each dataset and each method. For every model applied, a confusion matrix is generated and from that all the measures are computed [32]. Accuracy, F-score and AUC (area under the curve of a ROC curve) are the most common evaluation measures used in software defect prediction. Sensitivity, specificity and recall are also included to allow for a more accurate comparison. The Tables containing the results are compared using the Friedman test [19] (a nonparametric test is chosen to perform the comparison because the results given by the evaluation measures are not normally distributed). If the null hypothesis is not rejected, then the compared methods are not different from each other; if the null hypothesis is rejected, the Nemenyi test for multiple comparisons [31] is applied, in order to achieve a deeper understanding of the results and see which method performs better and which method performs worse.

Finally, all the results are interpreted in the conclusion by analyzing the comparisons between classifiers and in general by answering three main research questions (RQ):

RQ1. Do current unsupervised methods perform better or worse than the supervised ones?

RQ2. Which feature selection method is better for supervised methods?

RQ3. Which feature selection method is better for unsupervised methods?

Chapter 3

Data

This Chapter provides information about data (the first block in Figure 3.1) that contains software metrics and derives from different projects, which belong to two main groups: Eclipse [1] and NASA [2].

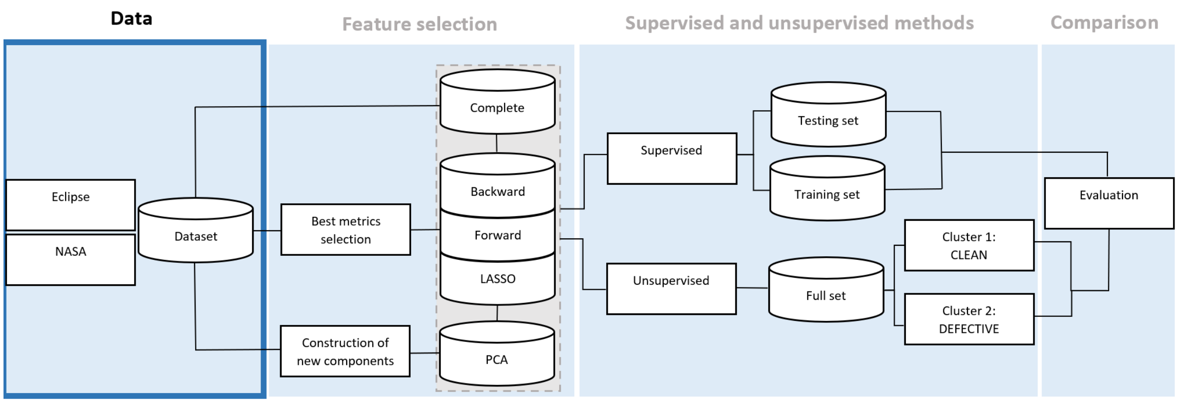


Figure 3.1: Methodology: the block of data

3.1 Eclipse datasets

The first five datasets contain software metrics computed from the Eclipse software and other related projects (which are Eclipse plugins or projects connected to Eclipse). These projects are:

- **Eclipse JDT Core:** one of the most popular IDE (integrated development environment) for coding in Java and other programming languages. It is written mostly in Java.
- **Equinox Framework:** a Java framework for developing and deploying modular software programs and libraries. It is part of the Eclipse project.
- **Apache Lucene:** a Java library that provides users with indexing and search features.
- **Mylyn:** a framework for Eclipse providing task management tools for developers.
- **Eclipse PDE UI:** plug-in development environment (PDE) that provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments and features.

All the variables, which are different types of software metrics, contained in the datasets have been computed through three tools [16]: inFusion (available at [6]), Moose (available at [7]) and Churrasco (available at [8]). The Eclipe datasets used in this thesis are available at [4].

3.1.1 Overview of the datasets

The Table 3.1 shows the number of observations of each dataset (each observation is a class) and the number the defective ones. The proportion of defective observations is shown as a percentage: the highest percentage of defective classes is reached in Equinox dataset (39.81%), while the lowest is in Lucene (9.26%).

Project	Number of classes	Number of defective classes	Percentage of defective classes
Eclipse	997	206	20.66%
Equinox	**324**	**129**	**39.81%**
Lucene	**691**	**64**	**9.26%**
Mylyn	1862	245	13.16%
PDE	1497	209	13.96%

Table 3.1: Number of defective observations - Eclipse datasets

Usually the number of defective classes is always lower than half of the total observations. The dataset with most defective classes is Equinox, while Lucene has the lowest number. In general the higher the values of the software metrics (explored in Section 3.1.3), the higher the probability that a class is defective. Since the number of faulty observations is always lower than the total number of observations, it means that the data will probably be skewed rather than normal. This evidence is explored further in the following Sections.

3.1.2 Metrics description

The Eclipse datasets are composed of 17 variables (i.e. software metrics) and a response variable, called "Defective", which contains the value 1 if the class is defective and 0 otherwise.

The software metrics included are either CK metrics (belonging to the Chidamber & Kemerer suite [15]) or other object-oriented metrics. All the variables, including the response, are numeric.

The 6 CK metrics are:

- **cbo:** coupling between objects, it is the count of the other classes that are connected to the current one.
- **dit:** depth of inheritance tree, it measures the length between the root node of the inheritance tree and the node of the current class. In general, the higher this value the more complex the code is (because it relies on inherited classes), and so the more fault-prone the class becomes.
- **lcom:** lack of cohesion methods, measures the number of methods pairs inside a class whose similarity is zero. High values of lcom means that the methods inside the class are very dissimilar to each other, and this adds a level of complexity that makes the class more fault-prone.
- **noc:** number of children, it is the number of subclasses immediately subordinate to the current class. A high value of noc means that the class is very important in the inheritance tree, because many other classes depend on it.
- **rfc:** response for a class, counts the number of methods that can be executed when an object of the current class is called. If rfc is high, it means that many methods can be executed when calling an object of this class, therefore it makes the class more likely to be defective.
- **wmc:** weighted methods per class, sum of the complexities of the methods inside a class. Complexities can be of different types (such as McCabe's Cyclomatic Complexity detailed in Section 3.2.2), but often they are set as 1 for each method: in this case, the value of wmc is just the number of methods in a class. In general the higher the value, the more fault-prone the class is.

The other 11 object-oriented metrics are:

- **fanIn:** number of classes that reference the current class.

- **fanOut:** number of classes that are referenced by the current class.
- **noa:** number of attributes in the class.
- **noai:** number of attributes inherited by the class.
- **loc:** number of lines of code of the current class.
- **nom:** number of methods in the class.
- **nomi:** number of methods inherited.
- **nopra:** number of private attributes. Private attributes are only accessible through the class in which they belong to.
- **noprm:** number of private methods. Private methods are only accessible through the class in which they belong to.
- **nopa:** number of public attributes. Public attributes can be accessed through classes other than the current one.
- **nopm:** number of public methods. Public methods can be accessed through classes other than the current one.

In general, the CK and the other object-oriented metrics are used to predict software defects. The higher the values of these variables, the higher is the complexity and importance of their class, which may make it fault-prone.

3.1.3 Variable distributions and normality tests

To test the distribution of the data, the Shapiro-Wilk test [36] is applied. It is important to know if the variables in the datasets are normally distributed.

The Shapiro-Wilk test works on univariate continuous data. Therefore, in order to test for the normality of the variables in the datasets, the test has to be applied to all the variables one at a time. It is computed according to the following equantion:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{3.1}$$

Where W is the test statistic, n is the number of observations, $(a_1, ..., a_n)$ is a vector of coefficients, $x_{(i)}$ is the i-th order statistic (which is the i-th smallest number inside the sample), x_i is the value of the i-th observation and $\overline{x}$ is the mean of the sample. The vector of coefficients is computed as:

$$(a_1, ..., a_n) = \frac{m^T V^{-1}}{C} \tag{3.2}$$

Where m is a vector $(m_1, ..., m_n)^T$ composed of the values of the expectations taken from a sample of the standard normal distribution. V is the covariance matrix of the expectations of m, and C is the norm of the two elements in the numerator:

$$C = ||V^{-1}m|| \tag{3.3}$$

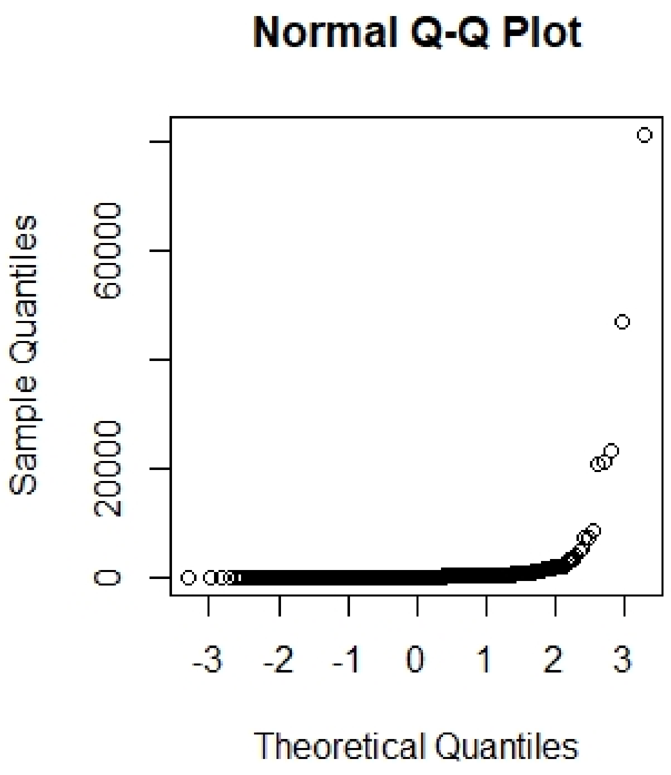
The reasoning behind the Shapiro-Wilk test is simple. Both the numerator and the denominator represent two attempts at estimating the population variance σ^2. The numerator is the slope of the observed data with respect to the expected normal values (taken directly from samples of a normal distribution). This slope can be visualized with the *qqplot* or *qqnorm* functions in R, as seen at the end of the Section. If the data is normally distributed, $\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2$ should represent correctly the population variance. The denominator is just the sum of squares, so it is a good estimate of σ^2.

In general, if the denominator is equal to the numerator, it means that the available data follows a normal distribution and $W = 1$ (or close to 1). The more the observed data deviates from the normal distribution, the farther the result of the test statistic is from 1 (because it becomes lower), which means the null hypothesis (that the data is normally distributed) is to be rejected.

Table 3.2 contains the values of the Shapiro-Wilk test statistics for every variable of every Eclipse dataset. All the values of W deviate greatly from 1, and so the corresponding p-values are all minor than the threshold of $\alpha = 0.01$. The metrics lcom and nopa are the ones deviating the most from a normal distribution, but all the others are not normal either. The ones with higher Shapiro-Wilk values, such as dit or fanOut, are still very different from a normal distribution. This is to be expected since, as seen in Section 3.1.1, the number of defective classes is always lower than half of the total number of classes. The Shapiro-Wilk test is further evidence that the data is heavily skewed. As an example, let us visualize the quantile plots of lcom and fanOut (taken from the Eclipse dataset) against the quantiles of a normal distribution, Figure 3.2. As a further example, histograms of the cbo variable are available in the Appendix, Section 8.1.

Variables	W_Eclipse	W_Equinox	W_Lucene	W_Mylyn	W_PDE
cbo	0.56	0.78	0.43	0.42	0.46
dit	**0.86**	0.51	**0.79**	0.65	**0.80**
fanIn	0.34	0.62	0.31	0.24	0.20
fanOut	0.68	0.71	**0.79**	0.70	**0.78**
lcom	**0.07**	**0.37**	**0.13**	**0.21**	**0.37**
noc	0.36	0.27	0.25	0.18	0.24
noa	0.26	0.48	0.61	0.43	0.03
noai	0.68	0.23	0.30	0.31	0.26
loc	0.36	0.53	0.40	0.25	0.66
nom	0.40	0.70	0.58	0.63	0.77
nomi	0.78	0.58	0.79	0.69	0.49
nopra	0.51	0.51	0.56	0.48	0.68
noprm	0.24	0.39	0.33	0.46	0.53
nopa	**0.14**	**0.13**	**0.47**	**0.11**	**0.01**
nopm	0.29	0.68	0.70	0.62	0.74
rfc	0.36	0.50	0.49	0.43	0.65
wmc	0.39	0.54	0.42	0.44	0.65

Table 3.2: Shapiro-Wilk test for the 5 Eclipse datasets



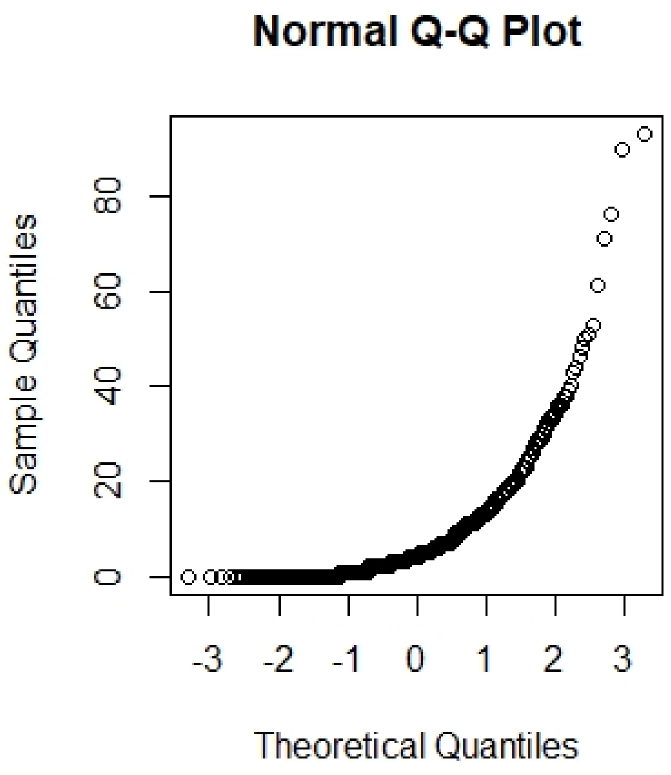


Figure 3.2: Quantile plots of lcom and fanOut variables - Eclipse dataset

3.1.4 Correlations

Since the data is not normally distributed, in order to study the correlations between variables it is better to compute the Spearman correlations [21] (because they are nonparametric measure, which do not assume normal distribution of the variables), rather than the Pearson [21].

The Spearman correlation measures rank correlation between two variables (which is the statistical dependence between the rankings of these variables). Its value varies from -1 to 1. It is defined as:

$$r_s = \frac{Cov((R(X), R(Y))}{\sigma_{R(X)} \cdot \sigma_{R(Y)}} \tag{3.4}$$

Where X and Y are the variables to be compared, and $R(X)$ and $R(Y)$ are the rankings of their values (called rank variables). The $\sigma_{R(X)}$ and $\sigma_{R(Y)}$ are the standard deviations of the rank variables. The numerator is the covariance between the two rank variables, while the denominator is their product. Basically, the Spearman correlation is the Pearson correlation of the ranks of the observations, rather than of the values of the observations themselves.

A Spearman correlation of 1 indicates that if the variable X increases, then the variable Y tends to increase as well. Viceversa, if the correlation is -1, when X then Y tends to decrease. When the correlation is 0 (or close to 0), Y does not have a tendency to increase or decrease if X is increasing.

In general, for the purpose of software defect prediction it is important to have low correlation between software metrics. That is why reducing the number of variables is often useful to build better classification models. Some variables are highly correlated between each other and that can influence negatively the performance of a model. For example, in Table 3.3 the Spearman correlations of the Eclipse dataset software metrics are shown. It is clear that some metrics are highly correlated, in particular: wmc with rfc, loc, nom and lcom; nom with lcom and loc; loc with rfc and lcom.

	cbo	dit	fanIn	fanOut	lcom	noc	noa	noai	loc	nom	nomi	nopra	noprm	nopa	nopm	rfc	wmc
cbo	1.00	-0.04	0.59	**0.88**	0.66	0.25	0.40	0.10	0.79	0.66	0.04	0.12	0.40	0.27	0.49	0.81	0.78
dit	-0.04	1.00	-0.14	0.06	-0.09	0.01	-0.28	0.75	-0.09	-0.09	**0.86**	-0.15	-0.29	-0.12	-0.04	-0.03	-0.08
fanIn	0.59	-0.14	1.00	0.25	0.40	0.38	0.27	-0.07	0.31	0.41	-0.08	0.07	0.15	0.16	0.38	0.31	0.33
fanOut	0.88	0.06	0.25	1.00	0.62	0.12	0.38	0.18	0.83	0.62	0.13	0.13	0.43	0.26	0.42	0.87	0.81
lcom	0.66	-0.09	0.40	0.62	1.00	0.17	0.54	0.08	0.82	**1.00**	0.07	0.29	0.38	0.15	0.73	0.85	0.83
noc	0.25	0.01	0.38	0.12	0.17	1.00	0.02	0.06	0.11	0.17	0.00	-0.23	-0.06	0.06	0.16	0.11	0.12
noa	0.40	-0.28	0.27	0.38	0.54	0.02	1.00	-0.14	0.57	0.54	-0.22	0.46	0.31	0.46	0.41	0.52	0.56
noai	0.10	0.75	-0.07	0.18	0.08	0.06	-0.14	1.00	0.10	0.08	0.73	-0.15	-0.18	0.05	0.09	0.15	0.11
loc	0.79	-0.09	0.31	0.83	0.82	0.11	0.57	0.10	1.00	0.82	0.02	0.21	0.50	0.33	0.59	**0.96**	**0.98**
nom	0.66	-0.09	0.41	0.62	1.00	0.17	0.54	0.08	0.82	1.00	0.07	0.29	0.38	0.15	0.73	0.85	0.83
nomi	0.04	0.86	-0.08	0.13	0.07	0.00	-0.22	0.73	0.02	0.07	1.00	-0.08	-0.26	-0.15	0.05	0.09	0.02
nopra	0.12	-0.15	0.07	0.13	0.29	-0.23	0.46	-0.15	0.21	0.29	-0.08	1.00	0.18	-0.21	0.13	0.24	0.20
noprm	0.40	-0.29	0.15	0.43	0.38	-0.06	0.31	-0.18	0.50	0.38	-0.26	0.18	1.00	0.23	0.20	0.46	0.50
nopa	0.27	-0.12	0.16	0.26	0.15	0.06	0.46	0.05	0.33	0.15	-0.15	-0.21	0.23	1.00	0.23	0.25	0.34
nopm	0.49	-0.04	0.38	0.42	0.73	0.16	0.41	0.09	0.59	0.73	0.05	0.13	0.20	0.23	1.00	0.58	0.61
rfc	0.81	-0.03	0.31	0.87	0.85	0.11	0.52	0.15	0.96	0.85	0.09	0.24	0.46	0.25	0.58	1.00	**0.95**
wmc	0.78	-0.08	0.33	0.81	0.83	0.12	0.56	0.11	0.98	0.83	0.02	0.20	0.50	0.34	0.61	0.95	1.00

Table 3.3: Spearman correlations of Eclipse dataset

3.2 NASA datasets

From the NASA Promise repository 12 datasets have been extracted [37]. They contain software metrics computed from different projects, and a binary column labeling each class as defective or not. The included projects are:

- **CM1:** software developed in C for a NASA spacecraft instrument.
- **JM1:** software that computes simulations in order to generate real-time predictions about ground systems. Developed in C.
- **KC1, KC3:** software for storage management developed in C++. It receives and processes ground data. It is split into two projects, where KC1 is the main one and the other is a relative component.
- **MC1, MC2:** no information available on these projects.

- **MW1:** no information available on this project.
- **PC1, PC2, PC3, PC4, PC5:** flight software developed for a satellite orbiting around earth. PC1 is the main project, while the others are minor projects connected to the main one. Written in C.

These projects are taken from the Promise repository, and the process of transformation and cleaning of the data is reported in [37]. See [5] for the transformed datasets used in this thesis.

3.2.1 Overview of the datasets

The following Table 3.4 shows the number of observations for every dataset (which are modules if the project is written in C, and functions if the project is written in C++). The number of defective instances is shown, together with the percentage with the respect to the total instances of the project. The datasets contain slightly different software metrics, some of which are overlapping and some of which are not. To solve this problem, the 12 datasets are splitted into 5 groups according to their variables: datasets belonging to the same group have the same exact variables.

Group	Project	N of variables	N of obs.	N of defective obs.	Percentage of defective obs.
1	CM1	37	327	42	13%
1	MW2	37	250	25	10%
1	PC1	37	679	55	8%
1	PC3	37	1053	130	12%
1	PC4	37	1270	176	14%
2	JM1	21	7720	1612	21%
2	KC1	21	1162	294	25%
3	KC3	39	194	36	19%
3	**MC2**	39	**124**	**44**	**35%**
4	MC1	38	1952	36	2%
4	PC5	38	1694	458	27%
5	**PC2**	36	**722**	**16**	**2%**

Table 3.4: Number of defective observations - NASA datasets

As with the Eclipse datasets, the frequency of defective data is lower than half of the total observations. In the NASA datasets, this frequency is even lower: projects like MC1 and PC2 only have 2% of observations labeled as defective, while the maximum percentage of defective observations is registered for project MC2 as 35%. This means that the data will be even more skewed in the NASA datasets, as explored in depth in Section 3.2.3.

3.2.2 Metrics description

The 12 NASA datasets contain different types of software metrics: lines of code metrics, McCabe and other complexity metrics, Halstead metrics and others, such as density metrics. All of them have a label column, called "Defective", which is 1 if the observation is faulty and 0 otherwise. Below, a detailed explanation of all the variables present.

- **LOC_BLANK:** number of blank lines of code.
- **BRANCH_COUNT:** number of logical branches.
- **CALL_PAIRS:** executable calls between modules.
- **LOC_CODE_AND_COMMENT:** sum of lines of code and lines of comment.
- **LOC_COMMENTS:** number of lines of comment.
- **CONDITION_COUNT:** number of conditions included in the module.

- **CYCLOMATIC_COMPLEXITY:** McCabe's cyclomatic complexity (maximum number of linearly independent paths through a program's source code, computed from the control-flow graph). It's the number of conditional branches in the control-flow graph. Can be described by the equation $M = E - N + 2P$, where E is the number of edges, N the number of nodes and P the number of connected components). Also called $v(G)$.
- **CYCLOMATIC_DENSITY:** cyclomatic complexity divided by the size of the system (in source statements).
- **DECISION_COUNT:** number of decisions.
- **DECISION_DENSITY:** number of decisions divided by the size of the system.
- **DESIGN_COMPLEXITY:** number of paths that call something.
- **DESIGN_DENSITY:** design complexity divided by cyclomatic complexity (total paths).
- **EDGE_COUNT:** number of edges of the control-flow graph.
- **ESSENTIAL_COMPLEXITY:** the sum of non reducible nodes plus one, plus one for each entry point after the first, plus one for each termination point after the first. Can be described by the equation $ESSCOM = (NONREDUC + 1) + (ENTRYPT - 1) + (TERMPT - 1)$. It's basically the complexity of the reduced flow graph (from which the structured constructs have been removed). Also called $ev(G)$.
- **ESSENTIAL_DENSITY:** it is the essential complexity with respect to the cyclomatic complexity. Described as $(ev(G) - 1)/(v(G) - 1)$.
- **LOC_EXECUTABLE:** lines of executable code.
- **PARAMETER_COUNT:** number of parameters.
- **HALSTEAD_CONTENT:** a complexity measure that is computed in the same way regardless of the language used. Can be described by $L \cdot V$, where L is the level and V is the volume of the module.
- **HALSTEAD_DIFFICULTY:** error proneness of the program, computed as $D = (UOP/2) \cdot (OD/UOD)$, where the elements of the equation are described further in this list.
- **HALSTEAD_EFFORT:** it represents the mental effort required to develop or maintain a class. Described by $E = D \cdot V$, where D is the difficulty and V the volume of the module.
- **HALSTEAD_ERROR_EST:** estimated Halstead error.
- **HALSTEAD_LENGTH:** total of all the lengths in the modules.
- **HALSTEAD_LEVEL:** described by $L = 1/D$, where D is the difficulty of the module.
- **HALSTEAD_PROG_TIME:** time (in seconds) to implement a program. Described by $T = E/18$, where E is the Halstead effort. The number 18 comes from empirical experiments.
- **HALSTEAD_VOLUME:** total of all the volumes in the methods (the volume V is the information content of a program, measured in mathematical bits. $V = length(program) \cdot \log_2(UOD + UOP)$). The length of a program is computed as $length(program) = OP + OD$.
- **MAINTENANCE_SEVERITY:** essential complexity divided by cyclomatic complexity. Described by $ev(G)/v(G)$.
- **MODIFIED_CONDITION_COUNT:** count for MC/DC (Modified Condition/Decision Coverage).
- **MULTIPLE_CONDITION_COUNT:** Count for MCC (Multiple Condition Coverage).
- **NODE_COUNT:** number of nodes in the control-flow graph.

- **NORMALIZED_CYCLOMATIC_COMPLEXITY:** normalized McCabe's cyclomatic complexity
- **NUM_OPERANDS:** number of operands OD.
- **NUM_OPERATORS:** number of operators OP.
- **NUM_UNIQUE_OPERANDS:** number of unique operands UOD.
- **NUM_UNIQUE_OPERATORS:** number of unique operators UOP.
- **NUMBER_OF_LINES:** number of lines.
- **PERCENT_COMMENTS:** percentage of comments in the total lines of code.
- **LOC_TOTAL:** total lines of code.
- **GLOBAL_DATA_COMPLEXITY:** complexity of the global data reduced flow graph (count of paths through global data).
- **GLOBAL_DATA_DENSITY:** global data complexity divided by cyclomatic complexity $v(G)$.
- **PATHOLOGICAL_COMPLEXITY:** measures the degree to which a module contains extremely unstructured constructs. Also called $pv(G)$.

3.2.3 Variable distributions and normality tests

As in the Eclipse datasets, in the NASA datasets all the variables are numeric. It is necessary to study their distribution, and see if they deviate from normality. To do that, the Shapiro-Wilk test is applied on every variable of every dataset.

The results of the Shapiro-Wilk test statistics of the first group of projects (CM1, MW2, PC1, PC3, PC4) are shown in the Table 3.5. The Shapiro-Wilk test statistics for the other projects are present in the Appendix, Section 8.2.

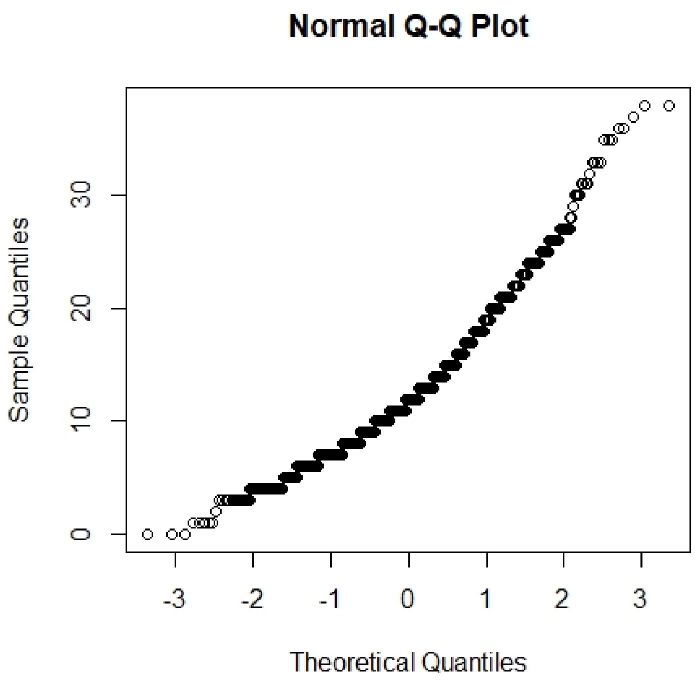
While in the Eclipse datasets all the variables are not normally distributed, by looking at the Table 3.5 it is possible to see that some Shapiro-Wilk test statistics are very close to 1 (because equal or higher than 0.90). This means that some variables, for example CYCLOMATIC_DENSITY, MAINTENANCE_SEVERITY, NUM_UNIQUE_OPERATORS or NORMALIZED_CYCLOMATIC_COMPLEXITY are actually normally distributed. The p-values obtained by the test statistics still are below the threshold of $\alpha = 0.01$, because the Shapiro-Wilk test is extremely sensitive to even slight deviations from normality. To verify the normality of these variables it is sufficient to look at their quantiles plot fitted against a normal quantiles plot.

Figure 3.3 shows the quantile plots of the variables NUM_UNIQUE_OPERATORS from the project PC4 and PERCENT_COMMENTS from the project CM1. These are clearly closer to a normal distribution.

Figure 3.4 shows the quantile plots of the variable HALSTEAD_PROG_TIME and of the variable LOC_CODE_AND_COMMENT taken from project CM1. Most of the variables are like this: they have a Shapiro-Wilk test statistic much lower than 1 and deviate strongly from the normal distribution. As a further example, histograms of the DECISION_COUNT variable are available in the Appendix, Section 8.1.

Variables	W_CM1	W_MW2	W_PC1	W_PC3	W_PC4
LOC_BLANK	0.64	0.84	0.53	0.75	0.70
BRANCH_COUNT	0.56	0.77	0.51	0.31	0.63
CALL_PAIRS	0.80	0.86	0.75	0.78	0.74
LOC_CODE_AND_COMMENT	0.56	0.13	0.36	0.50	0.46
LOC_COMMENTS	0.54	0.78	0.51	0.61	0.60
CONDITION_COUNT	0.58	0.74	0.51	0.28	0.63
CYCLOMATIC_COMPLEXITY	0.53	0.75	0.50	0.32	0.63
CYCLOMATIC_DENSITY	**0.91**	**0.93**	**0.95**	**0.95**	0.88
DECISION_COUNT	0.59	0.73	0.49	0.26	0.64
DECISION_DENSITY	0.42	0.28	0.47	0.55	0.75
DESIGN_COMPLEXITY	0.57	0.70	0.37	0.51	0.51
DESIGN_DENSITY	0.89	0.78	0.91	0.92	0.85
EDGE_COUNT	0.61	0.80	0.49	0.32	0.62
ESSENTIAL_COMPLEXITY	0.55	0.53	0.36	0.53	0.44
ESSENTIAL_DENSITY	0.67	0.27	0.73	0.69	0.56
LOC_EXECUTABLE	0.59	0.86	0.51	0.38	0.70
PARAMETER_COUNT	0.75	0.78	0.76	0.87	0.75
HALSTEAD_CONTENT	0.77	0.92	0.76	0.26	0.13
HALSTEAD_DIFFICULTY	0.78	0.87	0.65	0.61	0.79
HALSTEAD_EFFORT	0.27	0.56	0.15	0.06	0.31
HALSTEAD_ERROR_EST	0.57	0.81	0.43	0.17	0.36
HALSTEAD_LENGTH	0.64	0.85	0.53	0.25	0.48
HALSTEAD_LEVEL	0.78	0.90	0.79	0.83	0.73
HALSTEAD_PROG_TIME	0.27	0.56	0.15	0.06	0.31
HALSTEAD_VOLUME	0.57	0.81	0.43	0.17	0.36
MAINTENANCE_SEVERITY	0.91	0.92	0.92	0.92	0.86
MODIFIED_CONDITION_COUNT	0.58	0.72	0.52	0.30	0.60
MULTIPLE_CONDITION_COUNT	0.58	0.72	0.51	0.28	0.60
NODE_COUNT	0.63	0.81	0.49	0.32	0.60
NORMALIZED_CYLOMATIC_COMPLEXITY	0.86	**0.92**	**0.94**	**0.93**	0.85
NUM_OPERANDS	0.65	0.85	0.55	0.26	0.43
NUM_OPERATORS	0.64	0.84	0.52	0.24	0.50
NUM_UNIQUE_OPERANDS	0.69	0.91	0.53	0.38	0.35
NUM_UNIQUE_OPERATORS	0.86	**0.90**	0.82	**0.90**	**0.95**
NUMBER_OF_LINES	0.65	0.86	0.58	0.55	0.65
PERCENT_COMMENTS	**0.95**	**0.91**	0.82	0.83	0.85
LOC_TOTAL	0.59	0.86	0.52	0.40	0.70

Table 3.5: Shapiro-Wilk test for the first 5 NASA datasets



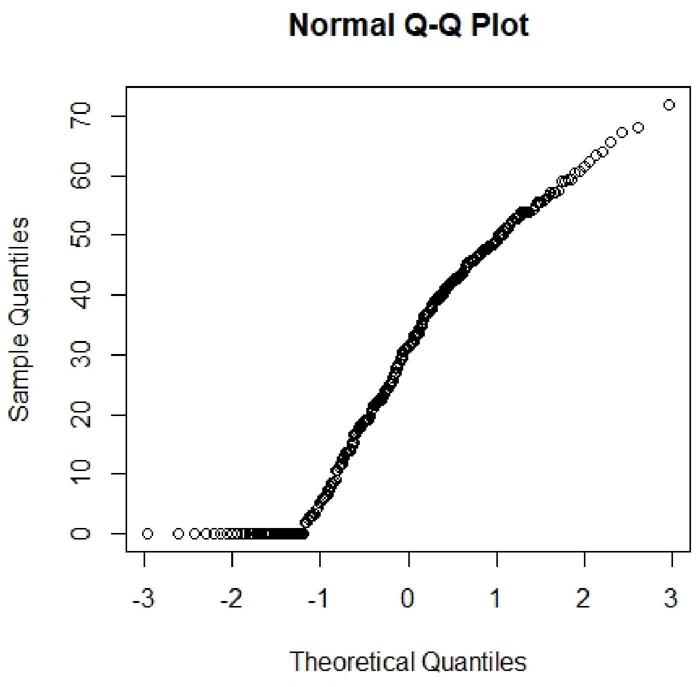
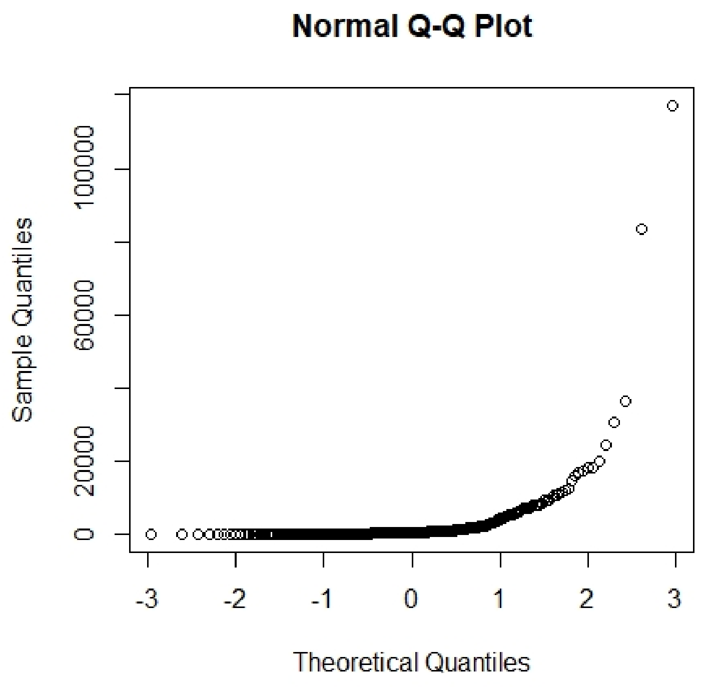


Figure 3.3: Quantile plots of NUM_UNIQUE_OPERATORS (PC4) and PERCENT_COMMENTS (CM1)



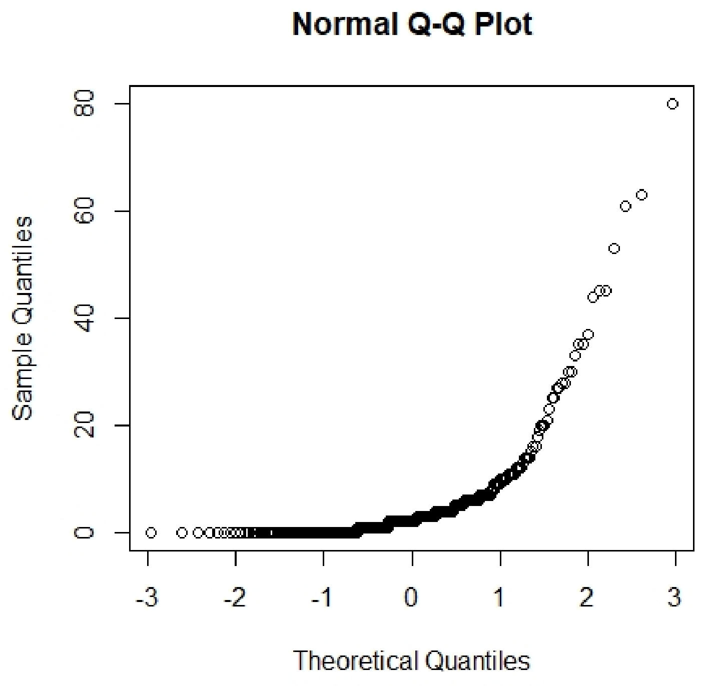


Figure 3.4: Quantile plots of HALSTEAD_PROG_TIME and LOC_CODE_AND_COMMENT (CM1)

3.2.4 Correlations

Since the majority of the data is not normally distributed, as seen in the Section 3.2.3 above, the study of the Spearman correlations between software metrics is preferred over the study of the Pearson correlations.

In Table 3.6 the Spearman correlations of the Halstead software metrics for project CM1 are shown. All of them are highly correlated, and this is the reason a feature selection method is necessary to select the ones useful for defect prediction.

Furthermore, in Table 3.8 the Spearman correlations of some complexity and density software metrics for project CM1 are shown. The acronyms for the metrics are stated in Table 3.7. Some of these metrics have very high correlation, such as EDGE_COUNT and DECISION_COUNT, CYCLOMATIC_COMPLEXITY and DECISION_COUNT, ESSENTIAL_DENSITY and ESSENTIAL_COMPLEXITY. As shown in these correlation plots, it is important to reduce the number of variables in order to reduce the correlations: this is explored in depth in Chapter 4.

	CONTENT	DIFFICULTY	EFFORT	ERROR_EST	LENGTH	LEVEL	PROG_TIME	VOLUME
CONTENT	1.00	0.51	0.78	**0.89**	**0.88**	-0.51	0.78	**0.89**
DIFFICULTY	0.51	1.00	**0.93**	0.84	0.84	-1.00	**0.93**	0.84
EFFORT	0.78	0.93	1.00	**0.98**	**0.98**	-0.92	**1.00**	**0.98**
ERROR_EST	0.89	0.84	0.98	1.00	**1.00**	-0.83	**0.98**	1.00
LENGTH	0.88	0.84	0.98	1.00	1.00	-0.84	**0.98**	**1.00**
LEVEL	-0.51	-1.00	-0.92	-0.83	-0.84	1.00	-0.92	-0.83
PROG_TIME	0.78	0.93	1.00	0.98	0.98	-0.92	1.00	**0.98**
VOLUME	0.89	0.84	0.98	1.00	1.00	-0.83	0.98	1.00

Table 3.6: Spearman correlations for Halstead metrics of project CM1

Variable Name	Abbreviation
CYCLOMATIC_COMPLEXITY	CC
CYCLOMATIC_DENSITY	CD
DECISION_COUNT	DC
DECISION_DENSITY	DD
DESIGN_COMPLEXITY	DESIGN_C
DESIGN_DENSITY	DESIGN_D
EDGE_COUNT	EC
ESSENTIAL_COMPLEXITY	ESS_C
ESSENTIAL_DENSITY	ESS_D

Table 3.7: Acronyms for Table 3.8

	CC	CD	DC	DD	DESIGN_C	DESIGN_D	EC	ESS_C	ESS_D
CC	1.00	0.18	**0.93**	0.50	0.78	-0.15	**0.95**	0.64	0.51
CD	0.18	1.00	0.08	0.27	-0.04	-0.31	-0.00	0.23	0.24
DC	0.93	0.08	1.00	0.32	0.75	-0.09	0.92	0.56	0.42
DD	0.50	0.27	0.32	1.00	0.28	-0.32	0.39	0.71	0.68
DESIGN_C	0.78	-0.04	0.75	0.28	1.00	0.42	0.82	0.45	0.31
DESIGN_D	-0.15	-0.31	-0.09	-0.32	0.42	1.00	-0.00	-0.26	-0.24
EC	0.95	-0.00	0.92	0.39	0.82	-0.00	1.00	0.56	0.41
ESS_C	0.64	0.23	0.56	0.71	0.45	-0.26	0.56	1.00	**0.94**
ESS_D	0.51	0.24	0.42	0.68	0.31	-0.24	0.41	0.94	1.00

Table 3.8: Sperman correlations for complexity and density metrics of project CM1

Chapter 4

Feature selection

To reduce collinearity between software metrics, it is necessary to select only the important variables or construct new features that are uncorrelated. These activities are part of the second block of Figure 4.1. Four different methods (three methods of feature selection and one of feature construction) are studied and applied to each of the 17 datasets available. In the following sections, each method is explained and in the last section the results of the methods are shown. A Table showing the number of variables selected from the different methods is produced.

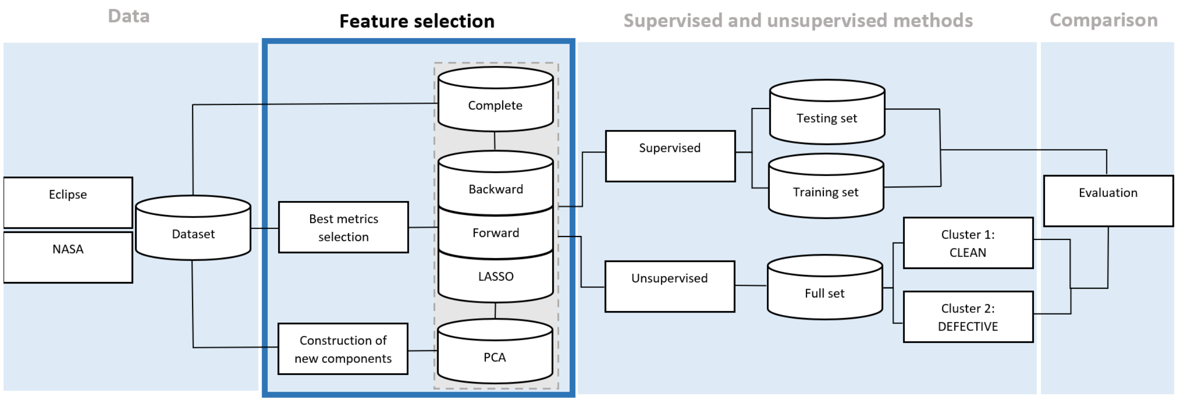


Figure 4.1: Methodology: the block of feature selection

4.1 Backward stepwise selection

Backward selection, or backward elimination, is an algorithm to reduce the number of variables in a model. The first step of the algorithm consists in fitting a regression model containing all the p independent variables of the model and the response variable. In this thesis, each dataset has a binary response variable called *Defective* (which gives 1 if the observation is defective and 0 otherwise), which is a common occurrence in software defect prediction datasets. Instead, the independent variables are all numeric. Therefore, the regression model fitted at the first step will be a logistic regression.

Given a model with p independent variables described as $(x_{1i}, ..., x_{pi})$ (where $i = 1, ..., n$ and n is the number of observations) and a binary response variable described as y_i, the model fitted at the first step is of this type:

$$\ln(\frac{y_i}{1-y_i}) = \beta_0 + \beta_1 x_{1i} + ... + \beta_p x_{pi} \tag{4.1}$$

Or, written in the equivalent form:

$$y_i = \frac{1}{1+e^{-(\beta_0+\beta_1 x_{1i}+...+\beta_p x_{pi})}} \tag{4.2}$$

Where β_0 is the intercept and $\beta_1, ..., \beta_p$ are the coefficients of every independent variable.

Once the full logistic regression has been computed, then the second step consists in fitting p logistic regressions, each one containing a different combination of $p-1$ variables. Basically, all the models containing one variable less than the full model are computed. Then, the Akaike Information Criterion (AIC) [9] is computed for each model of the second step. In the case of this thesis the AIC has been chosen to compare the models, but other methods of measurement can be used, such as Bayesian Information Criterion (BIC) [39]. The AIC is defined as:

$$AIC = 2k - 2\ln(\hat{L}) \tag{4.3}$$

Where k is the number of parameters in the model being studied. So, in case of any model of the second step, k is the dimension of the vector $(\beta_0, \beta_1, ..., \beta_{p-1})$. $\hat{L}$ is the maximum obtained from the likelihood function. At the end of the second step, each of the p logistic regressions is compared and the one with lowest AIC is chosen as the best.

In the following step of the algorithm, $p-1$ logistic regressions are fitted, each containing a different combination of $p-2$ variables. Basically, from the best model selected at the second step, one variable at a time is removed and new logistic regressions are fitted (containing one less variable). These models are evaluated again through AIC, and the one with the lowest AIC value is selected as the best.

The following steps follow the same procedure until a stopping criteria is reached. In this case, since the AIC is being used to evaluate the models, when removing a variable does not improve significantly the fit of the model the algorithm stops. In other words, when the AIC is not reduced significantly by removing variables, the procedure is stopped and the best model is selected from the ones in the current step. In R, this is done automatically within the *step* function.

4.2 Forward stepwise selection

Forward selection functions in a similar way to backward selection. However, the first step does not consist in fitting a full regression model, but in fitting a regression model containing only the intercept, response variable and no independent variables. Since the response variable is binary, the first step consists in fitting this simple logistic regression:

$$\ln(\frac{y_i}{1-y_i}) = \beta_0 \tag{4.4}$$

Or equivalently:

$$y_i = \frac{1}{1+e^{-\beta_0}} \tag{4.5}$$

In the second step of the algorithm p new models are fitted, each one containing a single different variable. Then these models are evaluated through AIC (or other measures), and the best one is selected according to the lowest AIC value. For example, if the first independent variable is chosen, the fitted model will be:

$$\ln(\frac{y_i}{1-y_i}) = \beta_0 + \beta_1 x_{1i} \tag{4.6}$$

Or, written in a different way:

$$y_i = \frac{1}{1+e^{-(\beta_0+\beta_1 x_{1i})}} \tag{4.7}$$

In the following step, starting from the best model selected earlier, new models are fitted by adding to each one a single new variable. Again, the best one is selected through AIC.

The next steps follow the same procedure until, as it happens in backward elimination, a stopping criteria is reached. When adding a new variable does not improve significantly the fit of the model, the algorithm stops and the best logistic regression is chosen between the ones in the current step.

4.3 Least absolute shrinkage and selection operator

Least absolute shrinkage and selection operator, also known as LASSO [40], is a penalized regression method that features integrated variable selection and regularization (shrinks the parameters of the regression associated to each variable, in order to reduce bias). For the purpose of this thesis, the

LASSO is applied to achieve variable selection: this is obtained because the process of regularization (or shrinkage) reduces the values of the parameters β_j (for $j = 1, ..., p$ and p is the number of variables) until some of these parameters become equal to 0. The variables for which the parameter is equal to 0 are removed from the model.

With LASSO, the regression coefficients are computed by maximizing the log-likelihood function, with the addition of an important constraint:

$$\sum_{j=1}^{p} |\beta_j| < c \tag{4.8}$$

Where $|\beta_j|$ is the absolute value of the coefficient associated to each variable x_j, and c is a positive tuning constant. The regularization process happens when c is set as:

$$c < \sum_{j=1}^{p} |\beta_j^{\hat{LS}}| \tag{4.9}$$

Where $|\beta_j^{\hat{LS}}|$ is the absolute value of the coefficient of each variable x_j, estimated by the least squares method. Setting c as small forces the parameters obtained by LASSO to shrink and sometimes become equal to 0 (which is how the variable selection happens).

Often LASSO is applied with linear regression, but the datasets in this project have a binary response variable and so the LASSO is applied with logistic regression (available in R, *glmnet* package). The probability of a class (or module) being defective is described as:

$$P(y_i = 1|\bar{x_i}) = \frac{e^{(\beta_0+\beta_1 x_{1i}+...+\beta_p x_{pi})}}{1 + e^{(\beta_0+\beta_1 x_{1i}+...+\beta_p x_{pi})}} \tag{4.10}$$

Where y_i is the response variable, $\bar{x_i} = (x_{i1}, ..., x_{ip})$ is the vector of predictors and $(\beta_0, ..., \beta_p)$ is the vector of parameters associated to each predictors (the parameters estimated through LASSO).

LASSO minimizes the negative of the log-likelihood in order to estimate the coefficients (which are subject to a constraint):

$$min \sum_{i=1}^{n} \left[-y_i(\beta_0 + ... + \beta_p x_{pi}) + \ln\left(1 + e^{(\beta_0+...+\beta_p x_{pi})}\right) \right], \quad \sum_{j=1}^{p} |\beta_j| < c \tag{4.11}$$

Since LASSO is not equivariant with respect to the scales of the data, each dataset will be standardized before applying this method. The constraint, in fact, requires data to be of comparable size otherwise the results will not be computed appropriately.

4.4 Principal components analysis

Principal components analysis (PCA) is a procedure that computes principal components (PCs) starting from the original p variables of a dataset [25]. It builds new features that are linear combinations of the original variables, and these new features are called PCs. Given the data available (p variables with n observations), PCA starts by finding the first principal component (PC_1). Given the standardized variables $X_1, ..., X_p$, PCA finds the linear combination:

$$PC_1 = w_{11}X_1 + w_{21}X_2 + ... + w_{p1}X_p \tag{4.12}$$

such that the variance $Var(PC_1)$ is maximized, and the vector of weights (also called loadings) $\bar{w_1} = w_{11} + ... + w_{p1}$ is subject to this constraint:

$$||\bar{w_1}|| = 1 \tag{4.13}$$

Therefore, PC_1 is a linear combination that assigns certain loadings to each variable of the dataset. The value of $Var(PC_1)$, divided by the sum of all the variances, expresses the percentage of variability explained by PC_1 with respect to the whole data.

After PC_1 has been computed, the next component Pc_2 is given by:

$$PC_2 = w_{12}X_1 + w_{22}X_2 + ... + w_{p2}X_p \tag{4.14}$$

such that it is orthogonal to PC_1, its variance $Var(PC_2)$ is maximized and the vector of loadings $\bar{w_2} = w_{12} + ... + w_{p2}$ is subject to this constraint:

$$||\bar{w_2}|| = 1 \tag{4.15}$$

The third component (PC_3) is a linear combination obtained in the same way as the others and with the same constraints, but also has to be orthogonal to all the previous components (in this case, PC_3 is orthogonal to both PC_1 and PC_2).

By following these transformation procedures, p PCs are computed (one for each original variable in the dataset). However, not all of them will be selected for the purpose of this thesis, but only the PCs explaining 95% of the variability will be chosen in order to reduce the number of variables. Furthermore, applying PCA to build new features is a good solution to deal with collinearity between variables, because the PCs generated are highly uncorrelated.

4.5 Selected features

The four methods of feature selection described in the previous Sections are applied to the 17 datasets available (5 Eclipse datasets and 12 NASA datasets). To reproduce the selections in R, see [3]. The Table 4.1 shows the number of variables selected by each feature selection method. In the variables count the binary response variable *Defective* is included. Furthermore, the column *Complete* shows the original number of variables in the complete datasets.

Dataset	Complete	Backward	Forward	LASSO	PCA
Eclipse	18	10	7	5	11
Equinox	18	7	6	9	10
Lucene	18	9	5	15	11
Mylyn	18	9	7	11	11
PDE	18	**13**	7	**14**	10
CM1	38	**13**	10	9	10
MW2	38	13	7	8	11
PC1	38	**19**	14	**18**	11
PC3	38	**23**	10	**26**	11
PC4	38	**21**	19	**29**	14
JM1	22	12	14	**21**	9
KC1	22	**13**	12	**16**	8
KC3	40	**14**	7	**12**	9
MC2	40	**20**	5	8	10
MC1	39	**16**	9	**15**	15
PC5	39	**22**	15	**27**	14
PC2	37	**11**	6	**8**	8

Table 4.1: Number of features selected by different methods

The number of variables selected (or constructed, in case of the PCA) varies greatly between different methods. Backward and LASSO selection methods tend to select more variables than the other methods, especially in PDE and in most NASA datasets. The subset of features selected will have an influence on the performance of supervised and unsupervised methods for classification and clustering, as will be explored in depth in the following Chapters.

Chapter 5

Supervised and unsupervised methods

In this Chapter, two supervised methods of classification and two unsupervised methods of clustering (the third block in Figure 5.1) are studied on the available 17 datasets selected through different feature selection methods. The two supervised methods are Naive Bayes and Random Forest, two of the most common and best performing methods in software defect prediction. The results obtained from these methods are used as a baseline for a comparison with the two unsupervised clustering methods, Neural Gas and Spectral Clustering. These unsupervised methods are relatively new to the field of software defect prediction. Their performance results (see Section 1.2) are described in depth and a comparison with the four feature selections of Chapter 4 is presented.

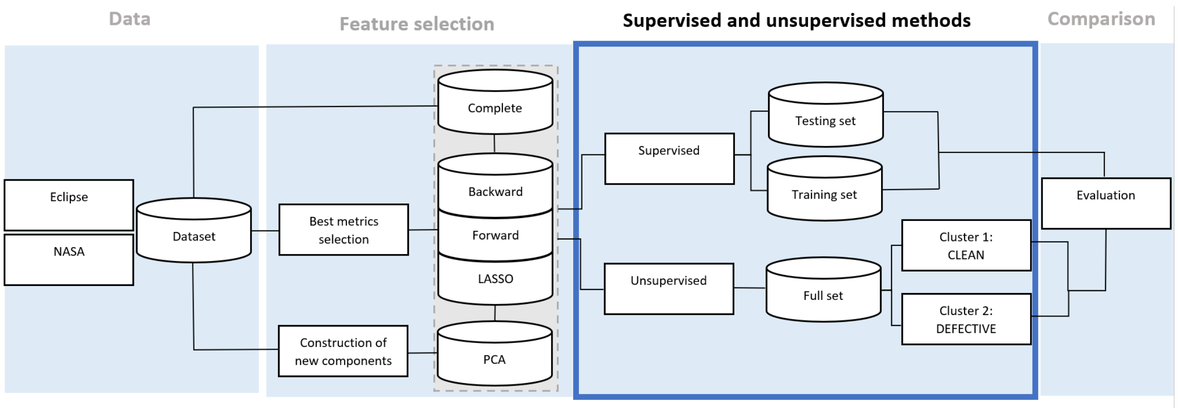


Figure 5.1: Methodology: the block of supervised and unsupervised methods

5.1 Naive Bayes

Naive Bayes classifier [23] is a method based on the application of Bayes' theorem [35]. It assumes independence between features. In the software defect datasets of this thesis, the response variable is a binary label that gives 1 if the observation is defective, and 0 otherwise. Therefore, given the defective label Y and a certain feature X, the Bayes' theorem is described as:

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)} \tag{5.1}$$

Where $P(Y|X)$ is the posterior probability of Y being labeled as defective or not given the predictor, $P(X|Y)$ is the likelihood of a certain predictor X given the label Y, $P(Y)$ is the unconditional (prior) probability of the class being defective or not, and $P(X)$ (the prior probability of the predictor) is:

$$P(X) = P(Y = 1|X) \cdot P(Y = 1) + P(Y = 0|X) \cdot P(Y = 0) \tag{5.2}$$

Starting from this theorem, the Naive Bayes classifier assigns the probability of being defective or not to each observation. In practice, given a set of predictors $(x_1, ..., x_p)$, the classifier predicts the posterior:

$$P(y_k|x_1, ..., x_p) = \frac{p(y_k) \cdot p(x_1, ..., x_p|y_k)}{p(x_1, ..., x_p)} \tag{5.3}$$

Where $k = 0, 1$ and p is the number of independent variables in a dataset of n observations. Since the data is available for every observation, $P(x_1, ..., x_p)$ can be considered as constant. Furthermore, the numerator can be written as the joint probability:

$$P(y_k) \cdot P(x_1, ..., x_p|y_k) = P(y_k, x_1, ..., x_p) \tag{5.4}$$

Since the features are assumed to be independent, then this proportionality is true:

$$P(y_k|x_1, ..., x_p) \propto P(y_k, x_1, ..., x_p) \propto P(y_k) \cdot \prod_{i=1}^{p} P(x_i|y_k) \tag{5.5}$$

The numerator of Bayes' theorem is excluded from the proportionality because it is constant. Finally, the Naive Bayes classifier can be described as:

$$\hat{y} = \underset{k \in 0,1}{\text{argmax}} P(y_k) \cdot \prod_{i=1}^{p} P(x_i|y_k) \tag{5.6}$$

The classifier has been trained on each dataset on 70% of the data, and tested on the remaining 30%. To improve its performance, since the data is skewed and not normally distributed for most variables, kernel density estimation [24] has been applied (this means that the algorithm did not assume normal distribution of the data, and instead computed an estimate of the data distribution before applying the actual classifier).

5.2 Random Forest

Random Forest is an ensemble method [23] based on the construction of multiple decision trees (DTs). In software defect prediction, multiple DTs are computed for the same observation, and then the observation is labeled as defective or not according to the results reached by the majority of the DTs. Each DT [33] starts from a root node (from where two possible decisions can be made, which lead to two new nodes). After the first decision is made, the DT goes further through many other internal nodes (each one leading to two new nodes), where new decisions are made according to different variables. When the DT reaches the end, it reaches one of these two possible leaf nodes: one containing the value 1 [34], labeling the instance as defective; one containing the value 0, labeling the instance as not defective.

Since all the variables of these software defect datasets are numeric, each node of the tree contains a threshold of a certain variable, from which the decision of where to go next in the tree is made. For example, considering the variable *cbo* a threshold can be set up at 50, which means that if the observation has *cbo* value lower than 50, the observation will proceed in the next node to the left, otherwise it will proceed in the next node to the right. Each node contains a new variable and a certain threshold, from which the decision of where to go next is made. The DT goes further until a leaf node is reached, where there are no variables and no thresholds, but only a label containing either 1 or 0.

The variable and the threshold of each node are selected according to impurity measures, in the case of this thesis the Gini impurity (G). At the top of the tree (root node) there is the variable and threshold with the lowest Gini impurity. At each new step of the DT, new Gini impurities are computed for each node and the node with lowest value of impurity is chosen. For example, given a variable X and a threshold x_T at the root node, these are the Gini impurities of the next node to the left and of the next node to the right:

$$G_{left} = 1 - P(Y = 1|X < x_T)^2 - P(Y = 0|X < x_T)^2 \tag{5.7}$$

$$G_{right} = 1 - P(Y = 1|X \geq x_T)^2 - P(Y = 0|X \geq x_T)^2 \tag{5.8}$$

Where Y is the defective label, $P(Y = 1|X \geq x_T)$ is the probability of the observation being defective given that its variable X is lower than the threshold x_T.

For each node, a variable X and a threshold x_T are selected. The variable X (taken from one of the p variables of the dataset) is chosen by computing the weighted average of the Gini impurities of the nodes directly below it (only one step down the DT). The best threshold x_T is selected by ordering all the values of the variable from lowest to highest: $x_{(1)}, ..., x_{(n)}$. Then, by computing the averages between each value:

$$\left(\frac{x_{(1)} + x_{(2)}}{2}\right), \left(\frac{x_{(2)} + x_{(3)}}{2}\right), ..., \left(\frac{x_{(n-1)} + x_{(n)}}{2}\right) \tag{5.9}$$

Where n is the total number of observations. Each of these averages is taken as a threshold for the variable X, and finally the threshold selected x_T is the one for which the next nodes have lowest weighted G.
At each new step of the DT, a node containing a different variable and a threshold is computed according to the Gini impurities. The algorithm stops when it reaches one of the the leaf nodes, containing either 1 or 0.

The Random Forest [13] works by creating a bootstrapped dataset, containing only 70% random observations (which can be repeated). Then, it creates a DT by selecting a random subset of independent variables (instead of using all the predictors available, it uses only some of them by picking them at random). At each step of the DT, a random subset of independent variables is selected from the variables used in the previous step.

After the first DT has been made following this method, Random Forest creates a new bootstrapped dataset (containing again 70% of the observations picked at random). From that, the algorithms builds a new DT, and so on. In this project, each application of Random Forest builds 500 trees: these trees are then used for every observation of the testing data (remaining 30% of the data). Every single observation is run through the 500 DTs computed earlier, and if the majority of the DTs has labeled it 1, then the observation is considered defective; otherwise, if the majority of the DTs has labeled the observation with 0, the observation is not defective.

5.3 Neural Gas Clustering

Neural Gas [20] is an artificial neural network (ANN), used in this thesis for the purpose of clustering [14] as a better alternative to other common clustering methods such as K-Means. Given a set of observations $\bar{x}_i$ (where $i = 1, ..., n$ and n is the total number of observations in the dataset) and two centers $\bar{w}_1$ and $\bar{w}_2$, the algorithm starts by choosing a random observation vector $\bar{x}_i = (x_{i1}, ..., x_{ip})$, where p is the total number of variables. Normally, a fixed number of centers has to be selected before running the algorithm: here, since the required clusters are only two (one with defective observations and one with non defective observations), only two centers are included.

After a first data point has been selected at random, the indexes of the two centers are computed: i_0 is the index of the closest center, while i_1 is the index of the center further away from the data point. Each selection of a random data point corresponds to the t-th step of the algorithm. At the each step, both centers $\bar{w}_1$ and $\bar{w}_2$ are updated: they get closer to the chosen observation, but the length they move depends on the previous distance between the observation and the center. If the center is already close to the observation, then it will be moved a lot closer; if the center is far from the observation, then it will be moved towards the observation but by a smaller distance. The update of the centers can be described as:

$$\bar{w}_{new} = \bar{w}_{old} + h(t)(\bar{x}_i - \bar{w}_{old}) \tag{5.10}$$

Where $\bar{w}_{new}$ is the updated center, $\bar{w}_{old}$ is the old center, $h(t)$ is the *increase* factor that is very high if the observation is close to the center but very low if the observation is far from the center. Instead $\bar{x}_i$ is the i-th observation, the one that has been picked at the t-th step. Considering there are only 2 centers in the application of Neural Gas for this thesis, the update process of each center is in more detail:

$$\bar{w}_k^{t+1} = \bar{w}_k^t + h(t)(\bar{x}_i - \bar{w}_k^t) \tag{5.11}$$

Where $k = 1, 2$. The *increase* factor $h(t)$ can be expanded, to see how it works in depth:

$$h(t) = \varepsilon \cdot \exp\left(\frac{-k}{\lambda}\right) \tag{5.12}$$

Where ε is the adaptation step size: its value is reduced at each step of the algorithm, until it goes to 0. Instead, λ is the neighbourhood range. The λ is reduced as well at each step of the algorithm. The presence of ε and λ is what makes the centers that are already close to an observation move more, while it makes the centers that are already further away move less. However, regardless of the strength of $h(t)$, the centers move in any case towards the observation considered at the t-th step.

5.4 Spectral Clustering

Spectral Clustering [28] is an unsupervised method for clustering that is based on the eigenvalues of a similarity matrix, which is obtained from the data. While many other clustering methods are distance-based, such as K-Means, this is a connectivity based classifier [43].

For the purpose of this method, each observation will be considered a *node*. Instead, each *edge* is the connection between two nodes: the weight of this connection is based on the similarity of values between two observations, described as:

$$w_{ij} = \bar{x}_i \cdot \bar{x}_j = \sum_{k=1}^{p} a_{ik} a_{jk} \tag{5.13}$$

Where $i = 1, ..., n$ and $j = 1, ..., n$ (n number of observations). Each observation is a vector of values $\bar{x}_i = (a_{i1}, ..., a_{ip})$, where p is the number of features. Thus, starting from the original matrix of data A, the normalized matrix of data A_{norm} is computed. Then from the normalized matrix A_{norm}, the matrix of similarities W is obtained (also called weighted adjacency matrix). This matrix contains all the values computed of w_{ij}: since the objective of the first step of the algorithm is to obtain a symmetric Laplacian matrix by transforming W, then all the values of $w_{ij} < 0$ are set equal to 0.

Given D, which is the diagonal matrix containing the number of edges of every node, one can say that:

$$D^{-\frac{1}{2}} = Diag\left(d_1^{-\frac{1}{2}}, ..., d_n^{-\frac{1}{2}}\right) \tag{5.14}$$

Where $d_i^{-\frac{1}{2}} = \left(\sum_{j=1}^{n} w_{ij}\right)^{-\frac{1}{2}}$. Finally, the Laplacian matrix is $L = D - W$ and so the symmatric Laplacian matrix can be obtained by:

$$L_{sym} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{5.15}$$

Where I is the $n \times n$ identity matrix. This is the end of the first step.

The second step consists in the eigendecomposition of the symmetric Laplacian matrix:

$$L_{sym} - \lambda I = 0 \tag{5.16}$$

This is done to obtain the eigenvectors:

$$\bar{v}_0 = (\lambda_{01}, ..., \lambda_{0n})\bar{v}_1 = (\lambda_{11}, ..., \lambda_{1n})...\bar{v}_k = (\lambda_{k1}, ..., \lambda_{kn}) \tag{5.17}$$

Where each λ_{ij} is an eigenvalue. These eigenvectors are ordered increasingly, and $\bar{v}_1$ is selected, because it is the second smallest eigenvector. This choice is done according to the normalized cut developed by [38]. The normalized cut is the cost of cutting two partitions in a graph: what spectral clustering is doing, is trying to minimize this cost.

The last step consists in creating the two clusters, one with the defective observations and one with non-defective observations. The threshold 0 is defined, and each eigenvalue contained in $\bar{v}_1 = (\lambda_{11}, ..., \lambda_{1n})$ is put in cluster C_{pos} if $\lambda_{1i} > 0$, or put in cluster C_{neg} otherwise (where $i = 1, ..., n$.

To distinguish which cluster between C_{pos} and C_{neg} is actually the one containing defective instances, it is sufficient to look at the values of the observations inside and see which one has the highest values. Most often in software defect prediction, high values for the software metrics mean that the class or module (instance) is defect-prone. In the datasets of this thesis in particular, this fact is true (as explored in Chapter 4), and so this heuristic for defining C_{pos} and C_{neg} is strong.

5.5 Performance of methods

To compare the performance [32] of the algorithms, a confusion matrix for every application of each algorithm is computed (these applications are on 17 complete datasets, 17 selected by backward elimination, 17 selected by forward selection, 17 selected by LASSO and 17 selected by PCA). A confusion matrix, Table 5.1, is a frequency matrix that compares the true classification of the data (the observations labeled as defective and as non-defective) against the classification of the data made by each machine learning classifier (or clustering method).

		Reference	
		0	1
Predicted	0	TP	FP
	1	FN	TN

Table 5.1: Confusion matrix

True positive (TP) represents the number of instances correctly identified as positive, which means non-defective. False positive (FP) represents the number of instances identified as positive (non-defective), but that are actually negative (defective). False negative (FN) represents the number of instances identified as defective but that are actually non-defective. True negative (TN) represents the number of defective instances correctly identified as such.

From these values, 6 different evaluation measures are computed:

- **Accuracy:** the number of correct predictions (which are the negatives and positives correctly identified) with respect to the total population. Defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.18}$$

- **F-score:** a different measure of the accuracy of an algorithm. In this thesis, the $F_1 - score$ is computed, which corresponds to the harmonic mean of recall and precision. Defined as:

$$F_1 - score = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{5.19}$$

- **AUC:** the area under the curve, also known as AUC, is the area below the ROC (receiver operating characteristic curve). This curve is made by plotting the recall (or TPR, true positive rate) against the FPR (false positive rate), which can be computed as $1 - specificity$. This is one of the most common evaluation methods used in machine learning and especially for the comparison of software defect prediction models.

- **Recall:** also known as sensitivity or TPR, is the proportion of relevant instances classified correctly. It is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{5.20}$$

- **Precision:** the proportion of instances that are relevant between the ones that have been considered positive. It is computed as:

$$Precision = \frac{TP}{TP + FP} \tag{5.21}$$

- **Specificity:** also known as TNR (true negative rate). It is the probability of a non-defective class being actually classified as non-defective. Defined as:

$$Specificity = \frac{TN}{TN + FP} \tag{5.22}$$

All of these measures go from 0 to 1. The closer they are to 1, the better the prediction model is, which means the classifier is correct most of the time (or all the time if these evaluation measures have value 1). An in depth comparison of the results is made in Chapter 6.

Tables 5.2, 5.3, 5.4 and 5.5 show the accuracy results of each algorithm applied to all the datasets (made with different selection methods).

By the values shown in the accuracy Tables, the supervised algorithms perform better than the unsupervised ones. But also Neural Gas Clustering has a decent performance. The situation is not clear-cut and there is no algorithm that performs far better than the others (except for Spectral Clustering, which seems to be the worst).

However, by looking at the AUC Tables (5.6, 5.7, 5.8 and 5.9), Naive Bayes with backward elimination reaches an average of 0.65 AUC and Random Forest with forward selection reaches an average of 0.61; Neural Gas with forward achieves 0.61 and Spectral Clustering with LASSO achieves 0.67. This is slightly different than what can be seen by the accuracy Tables only.

See the Appendix, Section 8.3, for the F1-scores. For the recall values, see Section 8.4. The precision values are in Section 8.5. The specificity values are in Section 8.6. Since the results are not so clear-cut, it is necessary to explore them in depth considering all the evaluation measures when necessary (instead of considering only accuracy or AUC, for example), and using nonparametric methods. This is done in Chapter 6, Comparison of Classifiers, and a further interpretation is available in Chapter 7, Conclusion. All the Tables of the other evaluation measures, and the relative code to produce them, are available at [3].

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.81	0.78	0.85	0.79	0.81
Equinox	0.66	0.66	0.73	0.78	0.72
Lucene	0.89	0.84	0.86	0.87	0.84
Mylyn	0.86	0.87	0.87	0.85	0.82
PDE	0.79	0.73	0.78	0.78	0.79
CM1	0.74	0.80	0.76	0.75	0.74
MW2	0.84	0.86	0.75	0.91	0.84
PC1	0.84	0.80	0.82	0.87	0.90
PC3	0.75	0.77	0.84	0.77	0.84
PC4	0.78	0.82	0.78	0.76	0.84
JM1	0.72	0.71	0.74	0.72	0.74
KC1	0.67	0.64	0.67	0.70	0.74
KC3	0.66	0.67	0.76	0.72	0.78
MC2	0.65	0.78	0.84	0.61	0.73
MC1	0.94	0.94	0.96	0.98	0.93
PC5	0.73	0.72	0.70	0.68	0.71
PC2	0.85	0.87	0.96	0.96	0.93

Table 5.2: Accuracy - Naive Bayes

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.83	0.83	0.86	0.87	0.85
Equinox	0.75	0.70	0.65	0.77	0.73
Lucene	0.91	0.93	0.91	0.92	0.89
Mylyn	0.88	0.86	0.87	0.88	0.86
PDE	0.88	0.88	0.86	0.84	0.83
CM1	0.86	0.84	0.82	0.87	0.89
MW2	0.86	0.96	0.94	0.85	0.88
PC1	0.92	0.93	0.92	0.92	0.93
PC3	0.87	0.88	0.87	0.86	0.89
PC4	0.88	0.89	0.91	0.90	0.91
JM1	0.79	0.79	0.79	0.80	0.80
KC1	0.79	0.76	0.79	0.77	0.76
KC3	0.79	0.83	0.88	0.83	0.84
MC2	0.63	0.71	0.68	0.68	0.58
MC1	0.98	0.97	0.98	0.98	0.97
PC5	0.78	0.78	0.78	0.77	0.79
PC2	0.98	0.97	0.97	0.98	0.99

Table 5.3: Accuracy - Random Forest

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.82	0.64	0.82	0.84	0.69
Equinox	0.67	0.68	0.69	0.70	0.68
Lucene	0.88	0.89	0.88	0.87	0.77
Mylyn	0.85	0.85	0.86	0.86	0.81
PDE	0.82	0.83	0.83	0.82	0.70
CM1	0.82	0.83	0.59	0.80	0.66
MW2	0.78	0.84	0.79	0.78	0.77
PC1	0.89	0.88	0.80	0.79	0.62
PC3	0.85	0.87	0.64	0.85	0.64
PC4	0.79	0.80	0.78	0.76	0.41
JM1	0.79	0.79	0.79	0.79	0.79
KC1	0.75	0.76	0.75	0.76	0.73
KC3	0.78	0.75	0.70	0.70	0.44
MC2	0.69	0.69	0.72	0.72	0.56
MC1	0.89	0.96	0.91	0.83	0.59
PC5	0.73	0.73	0.67	0.74	0.67
PC2	0.97	0.97	0.92	0.74	0.86

Table 5.4: Accuracy - Neural Gas Clustering

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.78	0.56	0.73	0.75	0.36
Equinox	0.71	0.73	0.72	0.72	0.70
Lucene	0.69	0.68	0.64	0.70	0.43
Mylyn	0.71	0.65	0.67	0.66	0.55
PDE	0.70	0.70	0.73	0.70	0.42
CM1	0.74	0.73	0.61	0.69	0.34
MW2	0.70	0.71	0.66	0.65	0.80
PC1	0.72	0.71	0.70	0.70	0.43
PC3	0.72	0.65	0.61	0.69	0.37
PC4	0.68	0.67	0.67	0.66	0.50
JM1	0.70	0.69	0.69	0.70	0.48
KC1	0.67	0.68	0.67	0.67	0.60
KC3	0.68	0.66	0.61	0.66	0.45
MC2	0.72	0.70	0.69	0.68	0.63
MC1	0.68	0.72	0.58	0.65	0.66
PC5	0.73	0.73	0.59	0.70	0.35
PC2	0.67	0.69	0.59	0.41	0.86

Table 5.5: Accuracy - Spectral Clustering

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.70	**0.67**	0.74	0.67	0.71
Equinox	0.65	**0.65**	0.72	0.77	0.70
Lucene	0.64	**0.63**	0.56	0.60	0.56
Mylyn	0.65	**0.62**	0.57	0.58	0.65
PDE	0.64	**0.59**	0.66	0.65	0.58
CM1	0.69	**0.66**	0.61	0.61	0.46
MW2	0.72	**0.81**	0.62	0.82	0.59
PC1	0.61	**0.70**	0.70	0.64	0.69
PC3	0.64	**0.61**	0.67	0.70	0.66
PC4	0.70	**0.78**	0.68	0.72	0.67
JM1	0.60	**0.62**	0.61	0.63	0.62
KC1	0.64	**0.59**	0.60	0.62	0.63
KC3	0.56	**0.52**	0.53	0.46	0.60
MC2	0.63	**0.72**	0.78	0.58	0.70
MC1	0.52	**0.47**	0.59	0.58	0.48
PC5	0.69	**0.68**	0.65	0.63	0.65
PC2	0.63	**0.77**	0.50	0.66	0.57

Table 5.6: AUC - Naive Bayes

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.67	0.72	**0.70**	0.67	0.72
Equinox	0.73	0.70	**0.65**	0.77	0.68
Lucene	0.52	0.56	**0.57**	0.60	0.52
Mylyn	0.57	0.58	**0.59**	0.58	0.59
PDE	0.57	0.55	**0.54**	0.65	0.53
CM1	0.50	0.49	**0.50**	0.61	0.49
MW2	0.54	0.49	**0.69**	0.82	0.54
PC1	0.61	0.59	**0.53**	0.64	0.60
PC3	0.53	0.53	**0.54**	0.70	0.55
PC4	0.65	0.66	**0.75**	0.72	0.69
JM1	0.58	0.57	**0.57**	0.63	0.56
KC1	0.64	0.62	**0.63**	0.62	0.60
KC3	0.54	0.63	**0.78**	0.46	0.62
MC2	0.57	0.61	**0.66**	0.58	0.50
MC1	0.50	0.50	**0.55**	0.56	0.50
PC5	0.65	0.66	**0.67**	0.63	0.67
PC2	0.50	0.50	**0.50**	0.66	0.50

Table 5.7: AUC - Random Forest

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.57	0.52	**0.58**	0.65	0.51
Equinox	0.59	0.59	**0.62**	0.63	0.62
Lucene	0.54	0.55	**0.58**	0.54	0.62
Mylyn	0.56	0.55	**0.59**	0.57	0.49
PDE	0.59	0.59	**0.60**	0.59	0.47
CM1	0.60	0.59	**0.69**	0.61	0.52
MW2	0.72	0.71	**0.72**	0.74	0.70
PC1	0.60	0.66	**0.65**	0.66	0.47
PC3	0.51	0.49	**0.50**	0.55	0.50
PC4	0.55	0.56	**0.58**	0.58	0.71
JM1	0.53	0.55	**0.55**	0.53	0.54
KC1	0.59	0.59	**0.58**	0.59	0.56
KC3	0.62	0.62	**0.70**	0.63	0.61
MC2	0.58	0.58	**0.69**	0.67	0.54
MC1	0.62	0.54	**0.60**	0.68	0.38
PC5	0.53	0.53	**0.60**	0.54	0.59
PC2	0.53	0.53	**0.50**	0.38	0.50

Table 5.8: AUC - Neural Gas Clustering

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.71	0.50	0.70	**0.73**	0.49
Equinox	0.68	0.71	0.70	**0.71**	0.70
Lucene	0.64	0.64	0.59	**0.63**	0.57
Mylyn	0.63	0.62	0.65	**0.63**	0.55
PDE	0.66	0.66	0.67	**0.65**	0.60
CM1	0.65	0.63	0.70	**0.64**	0.48
MW2	0.71	0.70	0.70	**0.70**	0.59
PC1	0.68	0.66	0.75	**0.74**	0.54
PC3	0.70	0.66	0.50	**0.71**	0.51
PC4	0.65	0.65	0.68	**0.65**	0.65
JM1	0.64	0.64	0.64	**0.63**	0.59
KC1	0.63	0.63	0.63	**0.63**	0.63
KC3	0.61	0.61	0.69	**0.67**	0.62
MC2	0.68	0.67	0.68	**0.67**	0.63
MC1	0.67	0.68	0.73	**0.71**	0.64
PC5	0.64	0.63	0.61	**0.64**	0.40
PC2	0.74	0.75	0.70	**0.70**	0.50

Table 5.9: AUC - Spectral Clustering

Chapter 6

Comparison of classifiers

Since the evaluation metrics to compare different methods do not follow a normal distribution and are highly skewed, it is necessary to compare them using a nonparametric test [17] (because it does not make assumptions on the data). The samples are not independent, because the methods have been compared on the same 17 datasets: therefore, the Friedman test is chosen. This is the last step of the thesis, as seen in Figure 6.1 The first Section contains a description of the Friedman test [19] and its application on the results of this thesis. The second Section contains a description of Nemenyi test [31] for multiple comparisons, that is applied if the Friedman test rejects the null hypothesis.

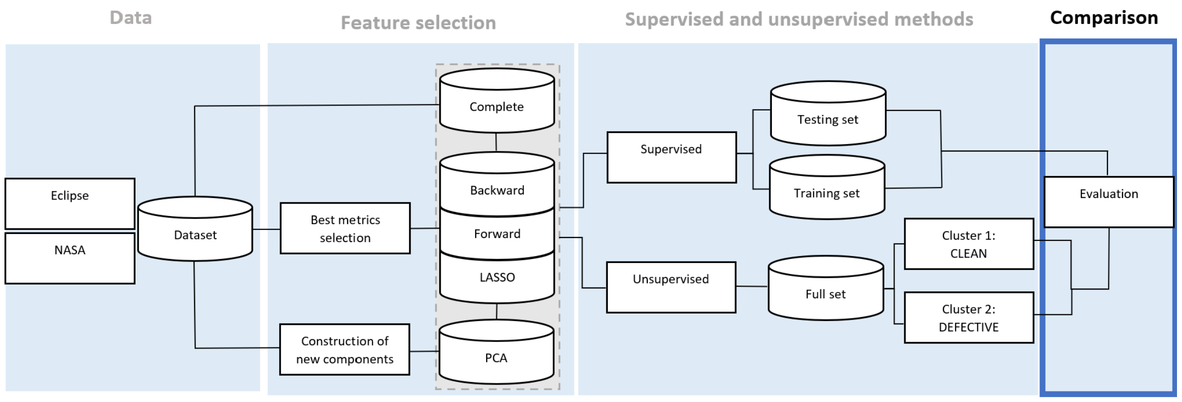


Figure 6.1: Methodology flow-chart: comparison of classifiers

6.1 Friedman test

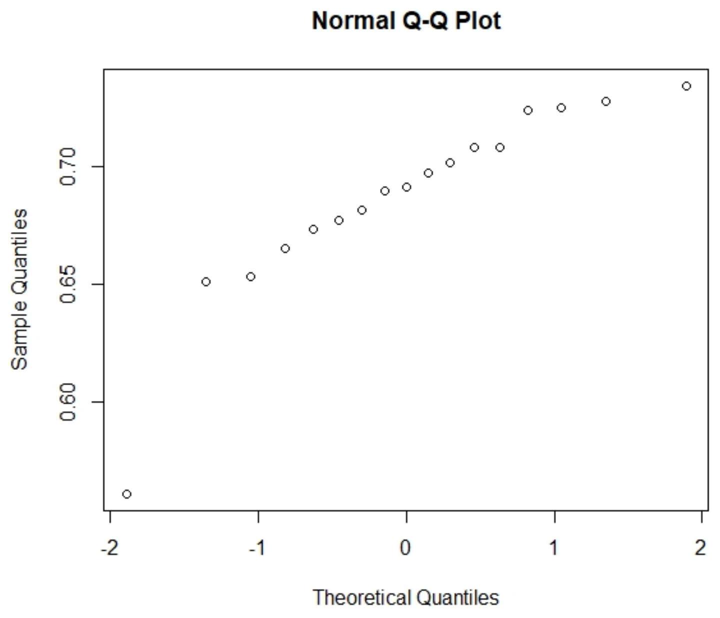
To compare the results produced in the previous Chapter, different types of tests can be used. Parametric tests rely on two main assumptions: the data is normally distributed and is spherical (the samples generated by each random variable have the same variance). By taking for example Table 6.1, it can be seen that some of the accuracy distributions of spectral clustering are not normally distributed, while others are. This means the normality assumption is not always respected. In this particular Table, backward and lasso selection accuracy results are not normally distributed, as can be seen by the quantile plots, Figure 6.2.

And by the respective values of the Shapiro-Wilk test, which rejects the null hypothesis of normality for $\alpha = 0.05$:

$$W = 0.850, \quad p\text{-}value = 0.011$$

$$W = 0.612, \quad p\text{-}value = 0.000$$

Furthermore the data is not spherical, as can be seen by the standard deviations in Table 6.2.



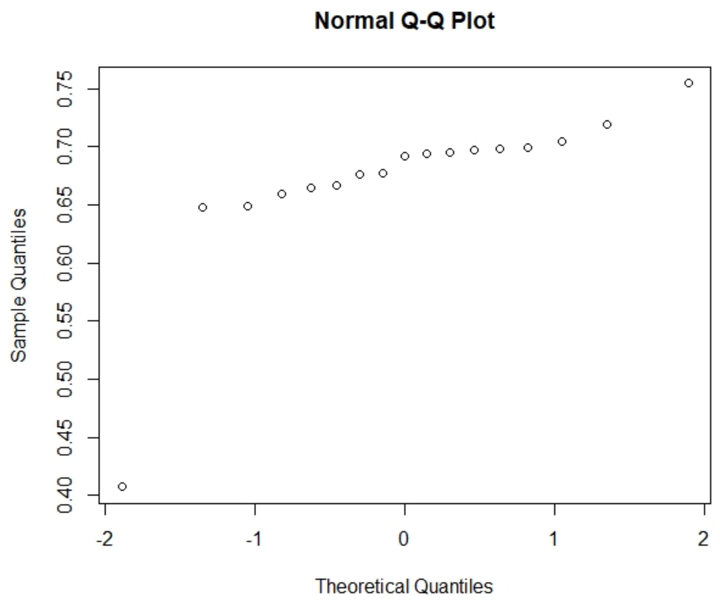


Figure 6.2: Quantile plots of backward accuracy results and lasso accuracy results for spectral clustering

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.78	0.56	0.73	0.75	0.36
Equinox	0.71	0.73	0.72	0.72	0.70
Lucene	0.69	0.68	0.64	0.70	0.43
Mylyn	0.71	0.65	0.67	0.66	0.55
PDE	0.70	0.70	0.73	0.70	0.42
CM1	0.74	0.73	0.61	0.69	0.34
MW2	0.70	0.71	0.66	0.65	0.80
PC1	0.72	0.71	0.70	0.70	0.43
PC3	0.72	0.65	0.61	0.69	0.37
PC4	0.68	0.67	0.67	0.66	0.50
JM1	0.70	0.69	0.69	0.70	0.48
KC1	0.67	0.68	0.67	0.67	0.60
KC3	0.68	0.66	0.61	0.66	0.45
MC2	0.72	0.70	0.69	0.68	0.63
MC1	0.68	0.72	0.58	0.65	0.66
PC5	0.73	0.73	0.59	0.70	0.35
PC2	0.67	0.69	0.59	0.41	0.86

Table 6.1: Accuracy - Spectral Clustering

Complete	Backward	Forward	LASSO	PCA
0.030	0.041	0.050	0.073	0.159

Table 6.2: Standard deviations of accuracy results for spectral clustering

Since some of the results do not follow a normal distribution and the data is not spherical, it is better to apply a nonparametric test [21] for comparison, because it does not make assumptions on the underlying data. This is a common solution in comparing machine learning classifiers and especially in software defect prediction.

The Friedman test [19] is chosen for this purpose. It tests the null hypothesis that there's no variation between each column of results $H_0 : \Delta_1 = \Delta_2 = ... = \Delta_k$, by computing a χ-squared test statistic with $k-1$ degrees of freedom :

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{6.1}$$

Where N is the number of observations in a column, k is the number of columns to be compared and R_j^2 is the square of the average rank of every column, described as: $R_j = \frac{1}{N}\sum_{i=1}^{N} r_i^j$. The values of r_i^j are obtained by ranking the values of the Table.

Alternatively, an F test statistic with $k-1$ and $(k-1)(N-1)$ degrees of freedom is computed:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{6.2}$$

Since the Friedman test relies on a χ-squared test statistic or on a F test statistic (which is a transformation of the χ-squared), it is an upper tail test.

Table 6.3 shows the p-values of the Friedman test for every classifier (or clustering method) with respect to the 6 evaluation measures considered. If the p-value is higher than the threshold $\alpha = 0.05$, it means that the different feature selection methods do not influence the performance of the algorithm. If the p-value is lower than the threshold $\alpha = 0.05$, it means that the algorithms perform better with some feature selection methods but worse with others, and this is worth exploring further.

From the Table of p-values, a first interpretation is clear: in the supervised methods (Naive Bayes and Random Forest) the different methods of feature selection do not have an influence on the results of the evaluation measures. The difference between backward, forward, LASSO and PCA feature

Evaluation Measures	Naive Bayes	Random Forest	Neural Gas Clustering	Spectral Clustering
Accuracy	0.324	0.972	**0.000**	**0.000**
F-score	0.190	0.967	**0.000**	**0.000**
AUC	0.621	0.141	**0.006**	**0.000**
Recall	0.109	0.766	**0.000**	**0.000**
Precision	**0.000**	0.145	**0.001**	**0.002**
Specificity	0.090	**0.023**	**0.000**	**0.000**

Table 6.3: Friedman test p-values over each results dataset

selection does not influence significantly the performance of each supervised classifier (except for the performance of precision in Naive Bayes and specificity in Random Forest: since the p-values are $0.0002 < 0.05$ and $0.0401 < 0.05$, the null hypotheses are rejected). Instead, with regards to the unsupervised methods of clustering, the feature selection method definitely has an influence on the performance of the algorithms, since all the p-values are lower than the threshold $\alpha = 0.05$ (or even of the stricter threshold $\alpha = 0.01$). The Friedman test does not show which of the feature selections is better for the overall performance of the unsupervised algorithms, and that is why when the null hypothesis is rejected, it is necessary to use a further test for multiple comparisons. This test for multiple comparisons will be applied and explained in the next Section.

6.2 Nemenyi test for multiple comparisons

When the Friedman test rejects the null hypothesis, all the k columns of the Table are compared using the Nemenyi test for multiple comparisons [31]. This test computes a critical distance (CD), described as:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{6.3}$$

Where q_α is the studentized range statistic divided by $\sqrt{2}$. So this critical value, Table 6.4, depends on the number of classifiers and on the threshold selected:

k	2	3	4	5	6	7	8	9	10
$q_{0.05}$	1.960	2.343	2.569	**2.728**	2.850	2.949	3.031	3.102	3.164
$q_{0.10}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

Table 6.4: Critical values for Nemenyi test

When comparing the different feature selection methods (complete, backward, forward, lasso, pca) the k columns are 5, $\alpha = 0.05$, and so $q_{0.05} = 2.728$. Therefore, the critical distance is $CD = 2.728\sqrt{\frac{5\cdot6}{6\cdot17}} = 1.479$. With the Nemenyi test, the rankings of each feature selection method are computed. When a method differs from another by at least the CD 1.479, it means the methods are significantly different.

The feature selection methods, for the purpose of the next Tables (6.5 and 6.6), are coded as:

- Complete: C
- Backward: B
- Forward: F
- LASSO: L
- PCA: P

Evaluation Measure	Rank 1 (lowest)	Rank 2	Rank 3	Rank 4	Rank 5 (best)
Accuracy	1.353 P	3.088 F	3.294 L	3.471 C	**3.794 B**
F-score	1.235 P	3.000 F	3.088 L	3.588 C	**4.088 B**
AUC	2.118 P	2.647 C	2.735 B	3.618 L	3.882 F
Recall	1.441 P	2.676 L	2.794 F	3.735 C	**4.353 B**
Precision	2.000 P	2.618 B	2.706 C	3.676 L	4.000 F
Specificity	1.235 P	3.029 L	3.294 F	3.529 C	**3.912 B**

Table 6.5: Nemenyi multiple comparison for feature selection methods applied to neural gas clustering

For Neural Gas Clustering, it seems that the best performing feature selection method is either backward elimination or forward selection. However, since all methods (except PCA) do not differ by more than the critical distance 1.479, it means that their performance is similar. The only method significantly different (and significantly worse) than the others is PCA.

Evaluation Measure	Rank 1 (lowest)	Rank 2	Rank 3	Rank 4	Rank 5 (best)
Accuracy	1.588 P	2.559 F	2.971 L	3.588 B	**4.294 C**
F-score	1.588 P	2.529 F	3.000 L	3.647 B	**4.235 C**
AUC	1.529 P	2.706 B	3.294 C	3.706 L	3.765 F
Recall	1.588 P	2.441 F	3.029 L	3.618 B	**4.324 C**
Precision	2.000 P	2.588 B	2.765 C	3.824 F	3.824 L
Specificity	1.176 P	3.118 L	3.176 F	3.294 B	**4.235 C**

Table 6.6: Nemenyi multiple comparison for feature selection methods applied to spectral clustering

For Spectral Clustering, leaving the dataset as it is (using the complete dataset) or applying LASSO are the best choices for good performance of the model. Again, PCA performs the worst, along with forward selection (and backward elimination in cases of AUC and Precision measures).

Therefore, LASSO or forward selection can be used as well on both methods, but specifically in case of neural gas clustering backward elmination is the best and in case of spectral clustering the complete datasets give the best performance.

Finally, a comparison between the classifiers is necessary. The best feature selection method is selected for the supervised methods by checking the median of the AUC values. The AUC is chosen as the best evaluation measure for this because it is the most common in software defect prediction. According to that, Naive Bayes performs best with backward elimination, while Random Forest performs best with forward selection. Furthermore, backward elimination is selected as the best feature selection method for Neural Gas Clustering, while the complete datasets are taken for Spectral Clustering. This is done according to the Nemenyi multiple comparisons above.

Now, the Friedman test is applied to Table 6.7, containing AUC values of the 4 classifiers (each classifier selected by their best feature selection method).

Project	NB - Backward	RF - Forward	NG - Backward	SC - Complete
Eclipse	0.667	0.698	0.515	0.709
Equinox	0.654	0.648	0.594	0.683
Lucene	0.628	0.566	0.554	0.642
Mylyn	0.619	0.592	0.547	0.628
PDE	0.592	0.545	0.592	0.658
CM1	0.665	0.505	0.589	0.650
MW2	0.807	0.691	0.713	0.711
PC1	0.702	0.528	0.659	0.676
PC3	0.615	0.545	0.494	0.703
PC4	0.776	0.754	0.556	0.649
JM1	0.623	0.574	0.545	0.636
KC1	0.590	0.629	0.593	0.632
KC3	0.525	0.778	0.623	0.608
MC2	0.723	0.659	0.577	0.679
MC1	0.474	0.550	0.545	0.672
PC5	0.681	0.673	0.525	0.636
PC2	0.769	0.497	0.528	0.738

Table 6.7: Comparison of classifiers through AUC values

The p-value obtained by the Friedman test is 0.001. This means there is a significant difference between the performance of these methods. Therefore, it is necessary to apply the Nemenyi test for multiple comparisons, which computes a $CD = 1.138$ and the following ranking (Table 6.8):

Rank 1 (worst)	Rank 2	Rank 3	Rank 4 (best)
NG - Backward	RF - Forward	NB - Backward	SC - Complete
1.647	**2.118**	**2.941**	**3.294**

Table 6.8: Ranking of best performing classifiers according to AUC - Nemenyi multiple comparison

The best performing classifier is Spectral Clustering on the complete datasets. However, there is not a significant difference between the performance of Random Forest, Naive Bayes and Spectral Clustering, since their ranking difference is not bigger than the critical distance $CD = 1.1376$. This is more clear from the plot of Nemenyi multiple comparisons, Figure 6.3.

Thus, the two supervised methods perform as well as Spectral Clustering, but the unsupervised method is often preferred in software defect prediction as it happens that many datasets are unlabeled or lack historical data. The fact that an unsupervised method can perform as well as the best supervised methods is extremely useful.

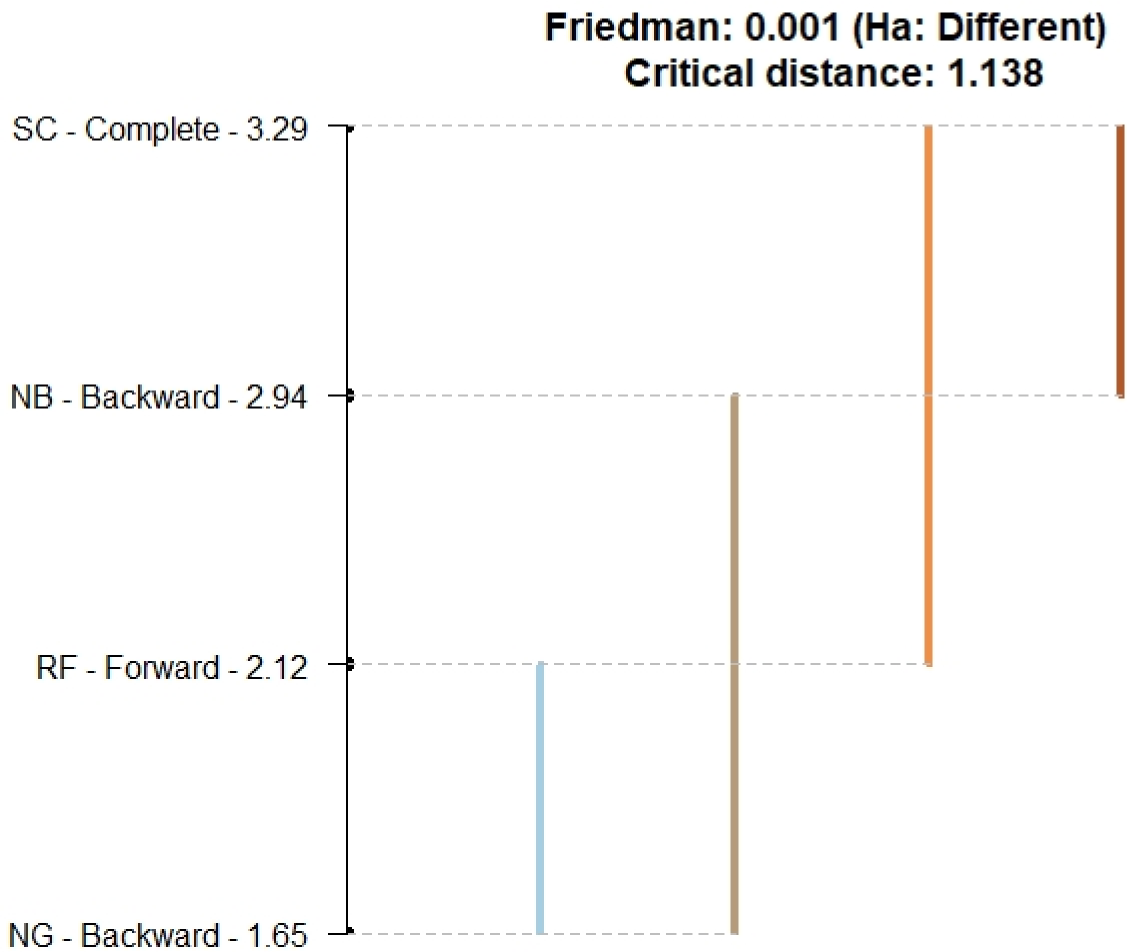


Figure 6.3: Nemenyi multiple comparison - best performing classifiers according to AUC

Chapter 7

Conclusion

In this thesis, different supervised and unsupervised machine learning methods have been studied to identify defective instances of code. Furthermore, different feature selection methods have been considered, in order to achieve better performance and, when necessary, reduce correlation between software metrics. In particular, Neural Gas and Spectral Clustering have been studied and compared to Naive Bayes and Random Forest, to see if they could achieve good performances in predicting software defects. All of these methods have been applied on a subset of features selected by stepwise approaches (backward and forward selection), LASSO and PCA, and also on the complete datasets. To correctly interpret the outcomes of this thesis, the evaluation metrics computed have been compared through nonparametric tests, allowing for multiple comparisons. Now, it is possible to answer the research questions stated in Chapter 2.

RQ1. Do current unsupervised methods perform better or worse than the supervised ones?

Spectral Clustering, when applied to the complete datasets, actually performs better than Naive Bayes and Random Forest according to the AUC values. However, the Nemenyi test for multiple comparisons shows how there is no significant difference between this unsupervised clustering method and the two supervised methods, since their ranking in the AUC performance values does not differ by a critical distance CD. Therefore, applying the correct feature selection to Spectral Clustering yields a slightly better performance than the supervised methods taken into consideration. Instead, Neural Gas Clustering performs significantly worse than all the other methods (both supervised and unsupervised) used in this thesis.

RQ2. Which feature selection method is better for supervised methods?

For supervised methods, two different feature selection methods are required to reach best performance (according to AUC values). For Naive Bayes, backward elimination is the best. For Random Forest, forward selection gives the best results. However, again there is not a significant difference in the performance of feature selection methods to reach good supervised prediction: this means that the other methods such as LASSO and PCA can be applied as well, without a significant loss of performance. In general, supervised methods perform well in software defect prediction, especially Random Forest.

RQ3. Which feature selection method is better for unsupervised methods?

For Neural Gas clustering, backward elimination is the best feature selection method, as can be seen from the Nemenyi test for multiple comparisons. All the feature selection methods (including the use of the complete datasets) give similar performance results for AUC, except for PCA which appears to be significantly worse than the other methods. For Spectral Clustering, keeping the complete datasets is the best way to select the features. This can be seen, again, from the Nemenyi test. As in Neural Gas Clustering, PCA appears to be significantly worse than the other methods, while the others give similar performance results.

A good performance can be achieved by unsupervised methods (in this case achieved by Spectral Clustering) for software defect prediction. This is an important result because it happens often that the software defects datasets are not labeled or lack historical data about, for example, the pre-release number of defects. To reduce collinearity between variables, different methods of feature selection can be applied, depending on the prediction model being supervised or unsupervised. In case of application of Spectral Clustering it is better, however, to run the algorithm on the complete datasets (without reducing the number of features, if possible).

Being able to apply unsupervised methods that perform as well as some of the best supervised ones is really important, especially when it is necessary to explore new software defects datasets. Further research can be done by experimenting with new datasets and new unsupervised methods, in particular by exploring more in depth the performance of connectivity-based classifiers such as Spectral Clustering with other feature selection methods.

Chapter 8

Appendix

8.1 Histograms

Figure 8.1 contains the histogram of cbo variable for Eclipse dataset. Figure 8.2 contains the histograms of cbo variable for Equinox and Lucene datasets. Figure 8.3 contains the histograms of DECISION_COUNT variable for PC1 and PC3 datasets.

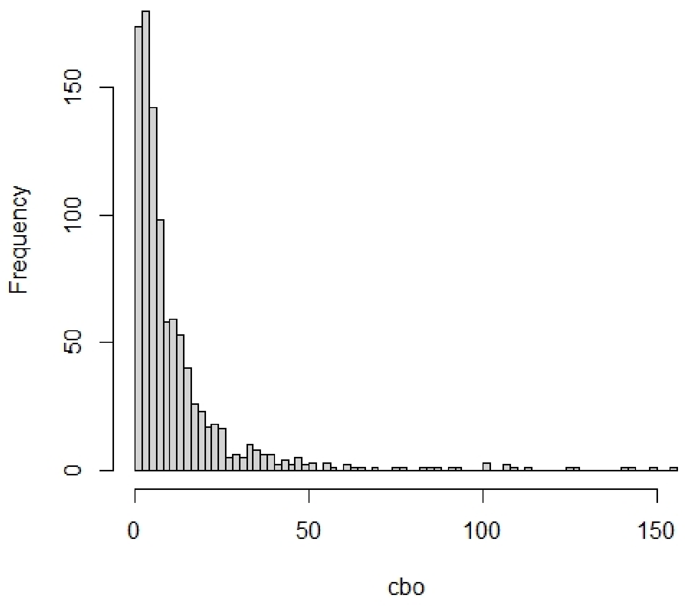
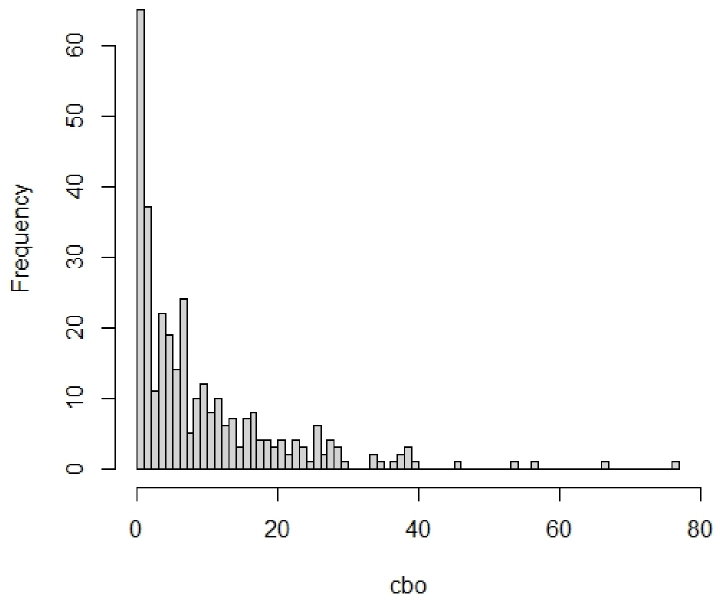


Figure 8.1: Histogram of cbo variable - Eclipse dataset



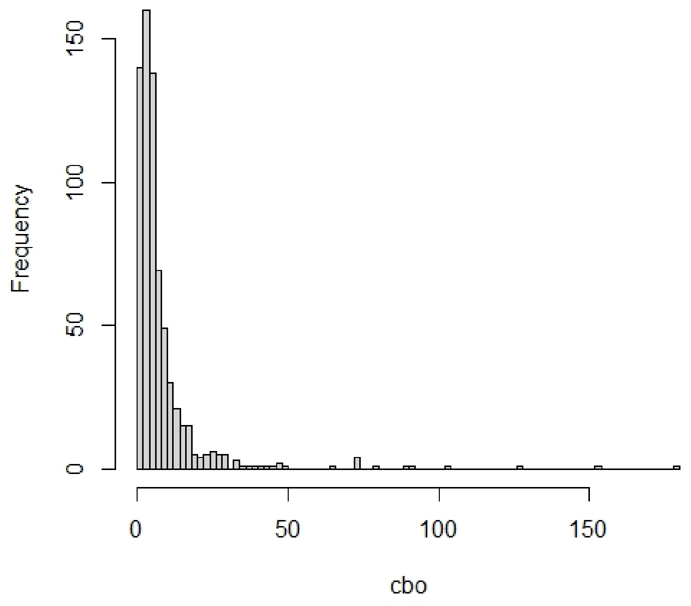
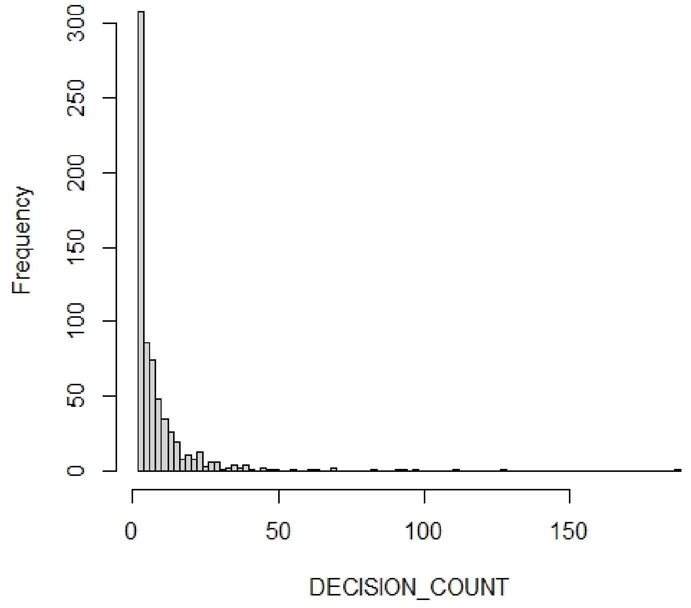


Figure 8.2: Histograms of cbo variable for Equinox and Lucene datasets



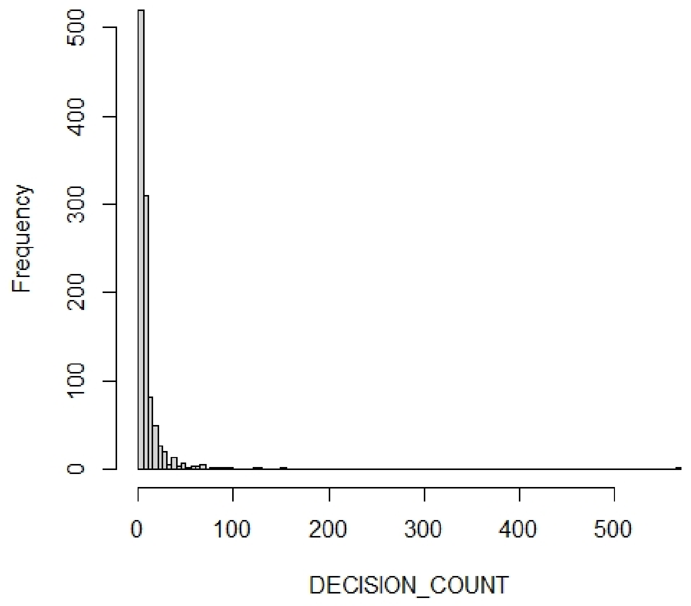


Figure 8.3: Histograms of DECISION_COUNT variable for PC1 and PC3 datasets

8.2 Shapiro-Wilk test on the NASA projects

These are the Tables (8.1, 8.2, 8.3, 8.4) containing the Shapiro-Wilk test statistics for all the variables of the NASA projects, groups 2, 3, 4 and 5 (group 1 is already included in Section 3.2.3 of the thesis).

Variable	W_JM1	W_KC1
LOC_BLANK	0.41	0.63
BRANCH_COUNT	0.35	0.67
LOC_CODE_AND_COMMENT	0.20	0.24
LOC_COMMENTS	0.33	0.45
CYCLOMATIC_COMPLEXITY	0.28	0.67
DESIGN_COMPLEXITY	0.22	0.65
ESSENTIAL_COMPLEXITY	0.36	0.48
LOC_EXECUTABLE	0.32	0.73
HALSTEAD_CONTENT	0.61	0.87
HALSTEAD_DIFFICULTY	0.60	0.86
HALSTEAD_EFFORT	0.04	0.42
HALSTEAD_ERROR_EST	0.29	0.65
HALSTEAD_LENGTH	0.39	0.72
HALSTEAD_LEVEL	0.77	0.56
HALSTEAD_PROG_TIME	0.04	0.42
HALSTEAD_VOLUME	0.29	0.65
NUM_OPERANDS	0.41	0.71
NUM_OPERATORS	0.38	0.72
NUM_UNIQUE_OPERANDS	0.45	0.83
NUM_UNIQUE_OPERATORS	0.40	0.96
LOC_TOTAL	0.35	0.73

Table 8.1: Shapiro-Wilk on NASA datasets - group 2

Variable	W_KC3	W_MC2
LOC_BLANK	0.79	0.65
BRANCH_COUNT	0.75	0.62
CALL_PAIRS	0.80	0.68
LOC_CODE_AND_COMMENT	0.29	0.60
LOC_COMMENTS	0.57	0.66
CONDITION_COUNT	0.72	0.62
CYCLOMATIC_COMPLEXITY	0.75	0.63
CYCLOMATIC_DENSITY	0.91	0.84
DECISION_COUNT	0.72	0.64
DECISION_DENSITY	0.46	0.58
DESIGN_COMPLEXITY	0.74	0.62
DESIGN_DENSITY	0.77	0.94
EDGE_COUNT	0.74	0.65
ESSENTIAL_COMPLEXITY	0.61	0.53
ESSENTIAL_DENSITY	0.70	0.72
LOC_EXECUTABLE	0.71	0.69
PARAMETER_COUNT	0.83	0.77
GLOBAL_DATA_COMPLEXITY	0.74	0.60
GLOBAL_DATA_DENSITY	0.84	0.75
HALSTEAD_CONTENT	0.80	0.84
HALSTEAD_DIFFICULTY	0.88	0.81
HALSTEAD_EFFORT	0.47	0.44
HALSTEAD_ERROR_EST	0.64	0.67
HALSTEAD_LENGTH	0.70	0.71
HALSTEAD_LEVEL	0.89	0.61
HALSTEAD_PROG_TIME	0.47	0.44
HALSTEAD_VOLUME	0.64	0.67
MAINTENANCE_SEVERITY	0.94	0.96
MODIFIED_CONDITION_COUNT	0.72	0.61
MULTIPLE_CONDITION_COUNT	0.72	0.62
NODE_COUNT	0.74	0.66
NORMALIZED_CYLOMATIC_COMPLEXITY	0.92	0.88
NUM_OPERANDS	0.70	0.73
NUM_OPERATORS	0.71	0.69
NUM_UNIQUE_OPERANDS	0.77	0.83
NUM_UNIQUE_OPERATORS	0.97	0.98
NUMBER_OF_LINES	0.72	0.70
PERCENT_COMMENTS	0.70	0.95
LOC_TOTAL	0.71	0.70

Table 8.2: Shapiro-Wilk on NASA datasets - group 3

Variable	W_MC1	W_PC5
LOC_BLANK	0.64	0.30
BRANCH_COUNT	0.34	0.35
CALL_PAIRS	0.64	0.55
LOC_CODE_AND_COMMENT	0.38	0.38
LOC_COMMENTS	0.40	0.31
CONDITION_COUNT	0.33	0.34
CYCLOMATIC_COMPLEXITY	0.30	0.36
CYCLOMATIC_DENSITY	0.79	0.83
DECISION_COUNT	0.31	0.33
DESIGN_COMPLEXITY	0.21	0.35
DESIGN_DENSITY	0.62	0.82
EDGE_COUNT	0.34	0.39
ESSENTIAL_COMPLEXITY	0.37	0.23
ESSENTIAL_DENSITY	0.20	0.62
LOC_EXECUTABLE	0.51	0.36
PARAMETER_COUNT	0.34	0.26
GLOBAL_DATA_COMPLEXITY	0.22	0.35
GLOBAL_DATA_DENSITY	0.62	0.72
HALSTEAD_CONTENT	0.76	0.08
HALSTEAD_DIFFICULTY	0.30	0.52
HALSTEAD_EFFORT	0.02	0.21
HALSTEAD_ERROR_EST	0.28	0.23
HALSTEAD_LENGTH	0.34	0.29
HALSTEAD_LEVEL	0.78	0.63
HALSTEAD_PROG_TIME	0.02	0.21
HALSTEAD_VOLUME	0.28	0.23
MAINTENANCE_SEVERITY	0.63	0.86
MODIFIED_CONDITION_COUNT	0.35	0.33
MULTIPLE_CONDITION_COUNT	0.33	0.33
NODE_COUNT	0.35	0.38
NORMALIZED_CYLOMATIC_COMPLEXITY	0.67	0.77
NUM_OPERANDS	0.33	0.31
NUM_OPERATORS	0.35	0.28
NUM_UNIQUE_OPERANDS	0.67	0.25
NUM_UNIQUE_OPERATORS	0.84	0.79
NUMBER_OF_LINES	0.58	0.37
PERCENT_COMMENTS	0.85	0.90
LOC_TOTAL	0.51	0.37

Table 8.3: Shapiro-Wilk on NASA datasets - group 4

Variable	W_PC2
BRANCH_COUNT	0.27
CALL_PAIRS	0.52
LOC_CODE_AND_COMMENT	0.34
LOC_COMMENTS	0.47
CONDITION_COUNT	0.24
CYCLOMATIC_COMPLEXITY	0.28
CYCLOMATIC_DENSITY	0.88
DECISION_COUNT	0.22
DECISION_DENSITY	0.42
DESIGN_COMPLEXITY	0.27
DESIGN_DENSITY	0.83
EDGE_COUNT	0.30
ESSENTIAL_COMPLEXITY	0.18
ESSENTIAL_DENSITY	0.39
LOC_EXECUTABLE	0.23
PARAMETER_COUNT	0.91
HALSTEAD_CONTENT	0.38
HALSTEAD_DIFFICULTY	0.70
HALSTEAD_EFFORT	0.23
HALSTEAD_ERROR_EST	0.32
HALSTEAD_LENGTH	0.41
HALSTEAD_LEVEL	0.87
HALSTEAD_PROG_TIME	0.23
HALSTEAD_VOLUME	0.32
MAINTENANCE_SEVERITY	0.88
MODIFIED_CONDITION_COUNT	0.26
MULTIPLE_CONDITION_COUNT	0.24
NODE_COUNT	0.31
NORMALIZED_CYLOMATIC_COMPLEXITY	0.96
NUM_OPERANDS	0.42
NUM_OPERATORS	0.41
NUM_UNIQUE_OPERANDS	0.42
NUM_UNIQUE_OPERATORS	0.89
NUMBER_OF_LINES	0.39
PERCENT_COMMENTS	0.89
LOC_TOTAL	0.32

Table 8.4: Shapiro-Wilk on NASA datasets - group 5

8.3 F1-score Tables

Tables 8.5, 8.6, 8.7 and 8.8 contain the F1-scores for Naive Bayes, Random Forest, Neural Gas Clustering and Spectral Clustering, each with the complete datasets and the 4 feature selection methods.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.880	0.860	0.909	0.891	0.879
Equinox	0.718	0.709	0.759	0.767	0.776
Lucene	0.938	0.912	0.921	0.937	0.913
Mylyn	0.922	0.928	0.926	0.921	0.893
PDE	0.875	0.838	0.866	0.867	0.877
CM1	0.832	0.885	0.850	0.840	0.851
MW2	0.906	0.920	0.845	0.912	0.909
PC1	0.911	0.882	0.895	0.915	0.943
PC3	0.852	0.863	0.907	0.885	0.906
PC4	0.866	0.894	0.867	0.830	0.906
JM1	0.817	0.812	0.832	0.825	0.837
KC1	0.762	0.743	0.763	0.784	0.836
KC3	0.773	0.791	0.857	0.722	0.864
MC2	0.731	0.846	0.893	0.826	0.792
MC1	0.970	0.966	0.981	0.988	0.966
PC5	0.806	0.800	0.784	0.793	0.793
PC2	0.919	0.929	0.982	0.967	0.964

Table 8.5: Naive Bayes - F1 score

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.896	0.894	0.915	0.889	0.911
Equinox	0.800	0.760	0.695	0.748	0.800
Lucene	0.955	0.964	0.951	0.927	0.940
Mylyn	0.934	0.925	0.929	0.919	0.924
PDE	0.936	0.937	0.922	0.911	0.905
CM1	0.925	0.915	0.897	0.915	0.940
MW2	0.925	0.977	0.966	0.950	0.932
PC1	0.955	0.965	0.959	0.960	0.960
PC3	0.927	0.936	0.930	0.935	0.939
PC4	0.931	0.937	0.951	0.935	0.948
JM1	0.880	0.879	0.876	0.880	0.882
KC1	0.868	0.847	0.869	0.854	0.848
KC3	0.878	0.899	0.930	0.913	0.906
MC2	0.737	0.814	0.766	0.806	0.711
MC1	0.988	0.987	0.992	0.989	0.987
PC5	0.859	0.857	0.857	0.862	0.865
PC2	0.988	0.983	0.984	0.996	0.993

Table 8.6: Random Forest - F1 score

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.895	0.766	0.898	0.909	0.804
Equinox	0.785	0.787	0.793	0.792	0.778
Lucene	0.936	0.941	0.934	0.936	0.866
Mylyn	0.919	0.919	0.924	0.922	0.894
PDE	0.900	0.901	0.902	0.896	0.817
CM1	0.894	0.905	0.702	0.884	0.787
MW2	0.864	0.905	0.875	0.872	0.862
PC1	0.939	0.932	0.886	0.890	0.755
PC3	0.917	0.928	0.769	0.915	0.768
PC4	0.881	0.883	0.872	0.866	0.573
JM1	0.883	0.879	0.879	0.882	0.879
KC1	0.844	0.850	0.843	0.847	0.835
KC3	0.869	0.845	0.789	0.782	0.581
MC2	0.796	0.796	0.780	0.774	0.631
MC1	0.940	0.981	0.954	0.909	0.744
PC5	0.843	0.843	0.769	0.845	0.777
PC2	0.985	0.987	0.957	0.851	0.922

Table 8.7: Neural Gas - F1 score

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.859	0.685	0.819	0.833	0.401
Equinox	0.767	0.778	0.775	0.765	0.734
Lucene	0.808	0.797	0.769	0.808	0.577
Mylyn	0.817	0.767	0.780	0.778	0.683
PDE	0.803	0.802	0.824	0.803	0.555
CM1	0.840	0.830	0.719	0.802	0.439
MW2	0.810	0.814	0.775	0.765	0.887
PC1	0.830	0.819	0.810	0.812	0.564
PC3	0.817	0.767	0.742	0.796	0.477
PC4	0.787	0.782	0.774	0.776	0.604
JM1	0.795	0.788	0.790	0.795	0.544
KC1	0.766	0.772	0.760	0.768	0.684
KC3	0.782	0.772	0.706	0.757	0.590
MC2	0.788	0.773	0.753	0.733	0.685
MC1	0.805	0.838	0.729	0.783	0.793
PC5	0.824	0.825	0.671	0.790	0.401
PC2	0.796	0.813	0.740	0.565	0.922

Table 8.8: Spectral Clustering - F1 score

8.4 Recall Tables

Tables 8.9, 8.10, 8.11 and 8.12 contain the recall values for Naive Bayes, Random Forest, Neural Gas Clustering and Spectral Clustering, each with the complete datasets and the 4 feature selection methods.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.899	0.849	0.896	0.873	0.866
Equinox	0.689	0.684	0.719	0.773	0.789
Lucene	0.926	0.938	0.917	0.928	0.918
Mylyn	0.906	0.895	0.889	0.875	0.902
PDE	0.920	0.905	0.914	0.911	0.880
CM1	0.905	0.948	0.855	0.887	0.851
MW2	0.955	0.984	0.907	0.905	0.893
PC1	0.944	0.958	0.950	0.943	0.948
PC3	0.941	0.899	0.925	0.927	0.917
PC4	0.940	0.961	0.943	0.924	0.913
JM1	0.831	0.848	0.827	0.858	0.836
KC1	0.830	0.819	0.775	0.832	0.833
KC3	0.810	0.837	0.849	0.833	0.795
MC2	0.792	0.846	0.893	0.731	0.731
MC1	0.982	0.986	0.979	0.987	0.979
PC5	0.840	0.838	0.802	0.808	0.808
PC2	0.984	0.995	0.964	0.975	0.981

Table 8.9: Naive Bayes - Recall

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.843	0.876	0.878	0.853	0.879
Equinox	0.781	0.657	0.688	0.878	0.727
Lucene	0.914	0.934	0.910	0.864	0.896
Mylyn	0.892	0.873	0.893	0.872	0.879
PDE	0.887	0.896	0.868	0.843	0.856
CM1	0.860	0.862	0.856	0.862	0.896
MW2	0.873	0.970	0.950	0.919	0.920
PC1	0.925	0.937	0.935	0.932	0.932
PC3	0.875	0.898	0.872	0.891	0.910
PC4	0.881	0.899	0.932	0.918	0.917
JM1	0.811	0.809	0.809	0.814	0.809
KC1	0.807	0.771	0.805	0.790	0.799
KC3	0.782	0.851	0.930	0.922	0.857
MC2	0.724	0.686	0.857	0.675	0.615
MC1	0.978	0.977	0.985	0.979	0.974
PC5	0.795	0.805	0.802	0.804	0.809
PC2	0.981	0.967	0.974	0.991	0.986

Table 8.10: Random Forest - Recall

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.991	0.735	0.991	0.980	0.809
Equinox	0.995	0.995	0.985	0.969	0.928
Lucene	0.959	0.967	0.947	0.959	0.807
Mylyn	0.960	0.963	0.961	0.959	0.922
PDE	0.915	0.917	0.918	0.908	0.786
CM1	0.891	0.916	0.554	0.867	0.712
MW2	0.791	0.867	0.809	0.800	0.791
PC1	0.942	0.918	0.832	0.840	0.644
PC3	0.958	0.988	0.685	0.950	0.683
PC4	0.886	0.889	0.860	0.850	0.459
JM1	0.983	0.966	0.960	0.980	0.970
KC1	0.908	0.924	0.915	0.918	0.911
KC3	0.880	0.829	0.696	0.703	0.475
MC2	0.950	0.950	0.775	0.750	0.588
MC1	0.898	0.979	0.924	0.840	0.601
PC5	0.977	0.981	0.751	0.974	0.778
PC2	0.990	0.994	0.938	0.758	0.873

Table 8.11: Neural Gas - Recall

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.834	0.602	0.756	0.774	0.269
Equinox	0.800	0.800	0.795	0.759	0.687
Lucene	0.707	0.691	0.656	0.708	0.429
Mylyn	0.742	0.663	0.675	0.675	0.553
PDE	0.714	0.713	0.748	0.715	0.422
CM1	0.775	0.761	0.575	0.712	0.295
MW2	0.702	0.711	0.649	0.636	0.858
PC1	0.734	0.718	0.692	0.697	0.404
PC3	0.722	0.651	0.644	0.685	0.327
PC4	0.689	0.682	0.663	0.671	0.442
JM1	0.741	0.729	0.730	0.741	0.394
KC1	0.717	0.730	0.707	0.718	0.576
KC3	0.715	0.696	0.570	0.652	0.487
MC2	0.812	0.787	0.725	0.688	0.625
MC1	0.678	0.725	0.575	0.647	0.662
PC5	0.852	0.857	0.570	0.773	0.296
PC2	0.664	0.688	0.589	0.394	0.873

Table 8.12: Spectral Clustering - Recall

8.5 Precision Tables

Tables 8.13, 8.14, 8.15 and 8.16 contain the precision values for Naive Bayes, Random Forest, Neural Gas Clustering and Spectral Clustering, each with the complete datasets and the 4 feature selection methods.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.862	0.871	0.922	0.910	0.892
Equinox	0.750	0.736	0.804	0.761	0.763
Lucene	0.951	0.887	0.926	0.947	0.908
Mylyn	0.939	0.964	0.967	0.972	0.884
PDE	0.835	0.780	0.823	0.828	0.874
CM1	0.770	0.830	0.845	0.797	0.851
MW2	0.863	0.863	0.790	0.919	0.926
PC1	0.880	0.816	0.845	0.888	0.938
PC3	0.779	0.829	0.890	0.847	0.895
PC4	0.802	0.837	0.803	0.753	0.899
JM1	0.802	0.778	0.836	0.796	0.838
KC1	0.705	0.680	0.751	0.742	0.839
KC3	0.739	0.750	0.865	0.636	0.946
MC2	0.679	0.846	0.893	0.950	0.864
MC1	0.958	0.948	0.983	0.990	0.954
PC5	0.774	0.765	0.768	0.778	0.778
PC2	0.863	0.872	1.000	0.960	0.949

Table 8.13: Naive Bayes - Precision

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.957	0.913	0.956	0.928	0.945
Equinox	0.820	0.902	0.702	0.652	0.889
Lucene	1.000	0.995	0.995	1.000	0.990
Mylyn	0.980	0.984	0.968	0.972	0.973
PDE	0.990	0.981	0.982	0.991	0.960
CM1	1.000	0.974	0.943	0.976	0.989
MW2	0.984	0.985	0.983	0.983	0.945
PC1	0.988	0.995	0.984	0.990	0.990
PC3	0.986	0.978	0.996	0.983	0.970
PC4	0.987	0.978	0.970	0.953	0.982
JM1	0.961	0.962	0.955	0.958	0.968
KC1	0.939	0.939	0.944	0.931	0.905
KC3	1.000	0.952	0.930	0.904	0.960
MC2	0.750	1.000	0.692	1.000	0.842
MC1	0.998	0.996	1.000	1.000	1.000
PC5	0.935	0.917	0.920	0.930	0.931
PC2	0.995	1.000	0.995	1.000	1.000

Table 8.14: Random Forest - Precision

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.817	0.800	0.821	0.847	0.799
Equinox	0.649	0.651	0.664	0.670	0.670
Lucene	0.915	0.917	0.921	0.915	0.934
Mylyn	0.882	0.880	0.890	0.887	0.867
PDE	0.885	0.885	0.886	0.884	0.850
CM1	0.898	0.894	0.958	0.901	0.879
MW2	0.952	0.947	0.953	0.957	0.947
PC1	0.935	0.946	0.947	0.948	0.912
PC3	0.880	0.875	0.877	0.882	0.877
PC4	0.876	0.877	0.884	0.882	0.763
JM1	0.802	0.807	0.811	0.802	0.804
KC1	0.788	0.787	0.781	0.787	0.771
KC3	0.858	0.862	0.909	0.881	0.750
MC2	0.685	0.685	0.785	0.800	0.681
MC1	0.986	0.983	0.986	0.990	0.975
PC5	0.742	0.740	0.788	0.746	0.776
PC2	0.979	0.979	0.978	0.971	0.978

Table 8.15: Neural Gas - Precision

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.885	0.795	0.893	0.903	0.783
Equinox	0.736	0.757	0.756	0.771	0.788
Lucene	0.943	0.943	0.930	0.941	0.882
Mylyn	0.910	0.911	0.924	0.917	0.891
PDE	0.917	0.917	0.917	0.916	0.808
CM1	0.917	0.912	0.959	0.919	0.857
MW2	0.958	0.952	0.961	0.960	0.919
PC1	0.956	0.953	0.975	0.973	0.933
PC3	0.942	0.933	0.876	0.950	0.880
PC4	0.916	0.918	0.929	0.920	0.953
JM1	0.857	0.858	0.861	0.857	0.875
KC1	0.824	0.818	0.822	0.825	0.843
KC3	0.863	0.866	0.928	0.904	0.748
MC2	0.765	0.759	0.784	0.786	0.758
MC1	0.991	0.991	0.996	0.994	0.989
PC5	0.798	0.795	0.814	0.807	0.619
PC2	0.994	0.994	0.993	1.000	0.978

Table 8.16: Spectral Clustering - Precision

8.6 Specificity Tables

Tables 8.17, 8.18, 8.19 and 8.20 contain the specificity values for Naive Bayes, Random Forest, Neural Gas Clustering and Spectral Clustering, each with the complete datasets and the 4 feature selection methods.

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.544	0.463	0.559	0.548	0.536
Equinox	0.548	0.571	0.636	0.615	0.636
Lucene	0.333	0.368	0.200	0.300	0.211
Mylyn	0.351	0.274	0.179	0.133	0.416
PDE	0.455	0.404	0.492	0.466	0.288
CM1	0.600	0.500	0.368	0.429	0.071
MW2	0.571	0.750	0.444	0.455	0.250
PC1	0.333	0.588	0.556	0.500	0.438
PC3	0.500	0.400	0.457	0.512	0.429
PC4	0.605	0.714	0.553	0.629	0.440
JM1	0.391	0.468	0.387	0.495	0.400
KC1	0.578	0.500	0.456	0.551	0.415
KC3	0.385	0.300	0.200	0.500	0.250
MC2	0.583	0.600	0.667	0.462	0.533
MC1	0.091	0.000	0.188	0.111	0.000
PC5	0.597	0.597	0.541	0.423	0.520
PC2	0.400	0.667	0.000	0.167	0.200

Table 8.17: Naive Bayes - Specificity

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.388	0.530	0.441	0.447	0.500
Equinox	0.632	0.489	0.595	0.818	0.475
Lucene	0.048	0.133	0.136	0.062	0.043
Mylyn	0.167	0.167	0.216	0.187	0.200
PDE	0.155	0.128	0.108	0.086	0.103
CM1	0.000	0.000	0.067	0.133	0.000
MW2	0.100	0.000	0.400	0.375	0.143
PC1	0.235	0.176	0.071	0.176	0.211
PC3	0.070	0.091	0.093	0.079	0.133
PC4	0.323	0.352	0.538	0.426	0.400
JM1	0.194	0.178	0.194	0.185	0.156
KC1	0.344	0.306	0.315	0.282	0.294
KC3	0.077	0.300	0.625	0.429	0.273
MC2	0.385	0.214	0.625	0.235	0.167
MC1	0.000	0.000	0.100	0.077	0.000
PC5	0.375	0.407	0.426	0.364	0.414
PC2	0.000	0.000	0.000	0.000	0.000

Table 8.18: Random Forest - Specificity

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.811	0.225	0.833	0.805	0.230
Equinox	0.960	0.962	0.914	0.857	0.741
Lucene	0.235	0.300	0.283	0.235	0.188
Mylyn	0.366	0.348	0.452	0.421	0.113
PDE	0.337	0.344	0.350	0.322	0.101
CM1	0.295	0.314	0.216	0.283	0.146
MW2	0.254	0.318	0.271	0.274	0.242
PC1	0.280	0.301	0.198	0.206	0.067
PC3	0.188	0.000	0.123	0.220	0.125
PC4	0.238	0.244	0.257	0.237	0.033
JM1	0.555	0.491	0.498	0.531	0.473
KC1	0.506	0.538	0.493	0.523	0.434
KC3	0.406	0.357	0.342	0.309	0.117
MC2	0.692	0.692	0.600	0.592	0.400
MC1	0.058	0.089	0.064	0.058	0.008
PC5	0.567	0.571	0.403	0.600	0.397
PC2	0.125	0.200	0.022	0.000	0.022

Table 8.19: Neural Gas - Specificity

Project	Complete	Backward	Forward	LASSO	PCA
Eclipse	0.478	0.209	0.410	0.439	0.203
Equinox	0.652	0.669	0.664	0.644	0.604
Lucene	0.167	0.164	0.133	0.164	0.073
Mylyn	0.232	0.204	0.228	0.218	0.158
PDE	0.255	0.254	0.274	0.253	0.097
CM1	0.256	0.236	0.224	0.226	0.122
MW2	0.212	0.207	0.194	0.188	0.200
PC1	0.170	0.158	0.186	0.185	0.090
PC3	0.257	0.213	0.123	0.250	0.125
PC4	0.239	0.239	0.247	0.237	0.199
JM1	0.351	0.345	0.351	0.351	0.255
KC1	0.396	0.395	0.388	0.398	0.353
KC3	0.286	0.284	0.299	0.312	0.110
MC2	0.615	0.585	0.560	0.537	0.483
MC1	0.037	0.042	0.038	0.040	0.033
PC5	0.512	0.511	0.359	0.450	0.211
PC2	0.052	0.056	0.043	0.036	0.022

Table 8.20: Spectral Clustering - Specificity

8.7 R software metrics analysis

The rsma package, available at [3], contains R functions to perform software metrics analysis for the purpose of software defect prediction. The functions of rsma used in this thesis are the following:

- **shapiro.wilk.sample:** applies the Shapiro-Wilk test to a sample of a variable, if the number of observations exceeds 5000. Default sample size is set to 2000.
- **binary.trans:** transforms the response variable of a dataset from discrete to continuous, assigning label 1 if the observation has one or more defects, otherwise assigning label 0.
- **select.backward:** performs backward elimination on a dataset and returns a new dataset containing only the selected variables and the response variable.
- **select.forward:** performs forward selection on a dataset and returns a new dataset containing only the selected variables and the response variable.
- **select.lasso:** performs variable selection on a dataset through LASSO and returns a new dataset containing only the selected variables and the response variable.
- **select.lasso.conv:** iterates the function select.lasso on a dataset for a set number of times, and select the subset of variables that appear the most. From this subset of variables, creates a new dataset containing the subset and the response variable. Default number of iterations is set to 50.
- **select.pca.prcomp:** performs principal components analysis on a dataset through the R function prcomp. By default, it selects the first principal components that explain at least 95% of the total variability (however, a different percentage can be set). It creates a new dataset containing the principal components selected.
- **add.response:** given the original dataset and a dataset selected by the function select.pca.prcomp, adds the response variable to the PCA selected dataset.
- **spectral.clustering:** performs the Spectral Clustering algorithm on a dataset and identifies two different clusters, one containing the defective instances and one containing the clean instances.

References

[1] URL: https://bug.inf.usi.ch/download.php.

[2] URL: https://github.com/klainfo/NASADefectDataset/tree/master/CleanedData/MDP/D’’.

[3] URL: https://github.com/Gianluca2022/rsma.

[4] URL: https://github.com/Gianluca2022/rsma.eclipse.

[5] URL: https://github.com/Gianluca2022/rsma.nasa.

[6] URL: http://www.intooitus.com/.

[7] URL: www.moosetechnology.org.

[8] URL: http://churrasco.inf.usi.ch.

[9] Hirotogu Akaike. “Information theory and an extension of the maximum likelihood principle”. (1998), pp. 199–213.

[10] Farah Atif et al. “A Survey on Data Science Techniques for Predicting Software Defects”. (2021), pp. 298–309.

[11] Ermiyas Birhanu. “Analysis of Software Quality Using Software Metrics”. *International Journal on Computational Science Applications* 8 (2018), pp. 11–20.

[12] Alexandre Boucher and Mourad Badri. “Predicting fault-prone classes in object-oriented software: an adaptation of an unsupervised hybrid SOM algorithm”. (2017), pp. 306–317.

[13] Leo Breiman. “Random forests”. *Machine learning* 45.1 (2001), pp. 5–32.

[14] Fernando Canales and Max Chacón. “Modification of the growing neural gas algorithm for cluster analysis”. (2007), pp. 684–693.

[15] Shyam R. Chidamber and Chris F. Kemerer. “A metrics suite for object oriented design”. *IEEE Transactions on software engineering* 20.6 (1994), pp. 476–493.

[16] Marco D’Ambros, Michele Lanza and Romain Robbes. “An extensive comparison of bug prediction approaches”. (2010), pp. 31–41.

[17] Janez Demšar. “Statistical comparisons of classifiers over multiple data sets”. *The Journal of Machine Learning Research* 7 (2006), pp. 1–30.

[18] Geanderson Esteves et al. “Understanding machine learning software defect predictions”. *Automated Software Engineering* 27.3 (2020), pp. 369–392.

[19] Milton Friedman. “The use of ranks to avoid the assumption of normality implicit in the analysis of variance”. *Journal of the american statistical association* 32.200 (1937), pp. 675–701.

[20] Bernd Fritzke. “A growing neural gas network learns topologies”. *Advances in neural information processing systems* 7 (1994).

[21] M. Hollander, D.A. Wolfe and E. Chicken. *Nonparametric Statistical Methods*. Wiley Series in Probability and Statistics. Wiley, 2013.

[22] Muzamil Jah. “Software metrics: usability and evaluation of software quality”. (2008).

[23] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[24] George H. John and Pat Langley. “Estimating continuous distributions in Bayesian classifiers”. *arXiv preprint arXiv:1302.4964* (2013).

[25] Ian T. Jolliffe and Jorge Cadima. “Principal component analysis: a review and recent developments”. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), pp. 1–16.

[26] Ruchika Malhotra and Ankita Jain. “Fault prediction using statistical and machine learning methods for improving software quality”. *Journal of Information Processing Systems* 8.2 (2012), pp. 241–262.

[27] Md Abdullah Al Mamun, Christian Berger and Jörgen Hansson. “Effects of measurements on correlations of software code metrics”. *Empirical Software Engineering* 24.4 (2019), pp. 2764–2818.

[28] Aris Marjuni, Teguh B. Adji and Ridi Ferdiana. “Unsupervised software defect prediction using median absolute deviation threshold based spectral classifier on signed Laplacian matrix”. *Journal of Big Data* 6.1 (2019), pp. 1–20.

[29] Thilo Mende and Rainer Koschke. “Effort-aware defect prediction models”. (2010), pp. 107–116.

[30] Jaechang Nam and Sunghun Kim. “Clami: Defect prediction on unlabeled datasets (t)”. (2015), pp. 452–463.

[31] Peter B. Nemenyi. “Distribution-free multiple comparisons.” (1963).

[32] Oluwaseyi Olorunshola et al. “Evaluation of machine learning classification techniques in predicting software defects”. *Trans Mach Learn Artif Intell* 8.5 (2020), pp. 01–15.

[33] Ross J. Quinlan. “Induction of decision trees”. *Machine learning* 1.1 (1986), pp. 81–106.

[34] Parvati Reena and Rajan Binu. “Software Defect Prediction System–Decision Tree Algorithm With Two Level Data Pre-processing”. *International Journal of Engineering Research & Technology (IJERT)* 3.3 (2014).

[35] Irina Rish. “An empirical study of the naive Bayes classifier”. 3.22 (2001), pp. 41–46.

[36] Samuel Sanford Shapiro and Martin B. Wilk. “An analysis of variance test for normality (complete samples)”. *Biometrika* 52.3/4 (1965), pp. 591–611.

[37] Martin Shepperd et al. “Data quality: Some comments on the nasa software defect datasets”. *IEEE Transactions on Software Engineering* 39.9 (2013), pp. 1208–1215.

[38] Jianbo Shi and Jitendra Malik. “Normalized cuts and image segmentation”. *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[39] Petre Stoica and Yngve Selen. “Model-order selection: a review of information criterion rules”. *IEEE Signal Processing Magazine* 21.4 (2004), pp. 36–47.

[40] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[41] Meng Yan et al. “File-level defect prediction: Unsupervised vs. supervised models”. (2017), pp. 344–353.

[42] Meng Yan et al. “Self-learning Change-prone Class Prediction.” (2016), pp. 134–140.

[43] Feng Zhang et al. “Cross-project defect prediction using a connectivity-based unsupervised classifier”. (2016), pp. 309–320.