

**POLITECNICO DI TORINO**

**Corso di Laurea Magistrale in Ingegneria Informatica**



**Thesis Master Degree**

**BUG PREDICTION:**

**Log approach – version approach**

Candidato: Lucio Ciraci (s240345)

Relatore: Elio Piccolo

Correlatore: Javier Segovia

Anno accademico 2018-2019

# **RINGRAZIAMENTI**

Un grande ringraziamento a mia madre e mio padre che, con il loro instancabile sostegno, sia morale che soprattutto economico, mi hanno permesso di arrivare fin qui e di permettermi di realizzarmi.

Grazie alla pazienza e alla professionalità del Politecnico di Torino (professori, dottorandi e assistenti) affinché mi hanno permesso di arrivare fin qui.

Gracias a la maravillosa experiencia Erasmus que he podido hacer en Madrid y en las muchísimas gente que he conocido.

I wish to express my most sincere gratitude to all my friends come from each part of Italy and world about the wonderful experiences that we had together.

# INDEX

RINGRAZIAMENTI.....	1
1. INTRODUCTION .....	4
2. OVERVIEW .....	6
2.1. Granularity level.....	6
2.2. Developers approach.....	8
2.3. Different types of approach.....	8
2.4. Last study with same dataset.....	10
3. BACKGROUND.....	11
3.1. NEURAL NETWORK.....	12
3.2. DECISION TREE .....	21
3.3. LINEAR REGRESSION .....	33
3.4. R-SQUARE.....	37
3.5. SPEARMAN CORRELATION.....	38
3.6. WHERE USE SPEARMAN .....	41
3.7. SPEARMAN OR $R^2$ .....	45
4. METRICS DESCRIPTION.....	46
4.1. CHANGE METRICS.....	47
4.2. PREVIOUS DEFECTS .....	48
4.3. SINGLE VERSION CK-OO.....	49
4.4. ENTROPY OF CHANGE.....	51
4.5. CHURN OF SOURCE CODE METRICS.....	54
4.6. ENTROPY OF SOURCE CODE METRICS .....	56
5. DATASET DESCRIPTION .....	58
5.1. DETAILS .....	62

6. DESCRIPTION WORK.....	63
6.1. ANALYSIS WITH PRINCIPAL METRICS.....	64
6.2. COMMENTS .....	80
7. ANALYSIS WITH A METRICS COMBINATION .....	82
7.1. COMMENTS .....	85
8. METRIC ADJUNCT.....	87
8.1. EXPERIMENTS WITH A NEW METRIC.....	91
8.2. COMMENTS NEURAL NETWORK & DECISION TREE .....	97
8.3. COMMENTS LINEAR REGRESSION.....	98
9. BINARY CLASSIFICATION .....	100
9.1. EXPLICATION BINARY CLASSIFICATION.....	100
9.2. SIMULATION.....	104
10. CONCLUSIONS .....	124
INDEX FIGURES .....	126
INDEX GRAPH .....	127
INDEX TABLES.....	128
BIBLIOGRAFIA .....	129

# 1. INTRODUCTION

Bug prediction has generated widespread interest for a long time. The general scenario have been many applications with different approaches. The raising of properties of programming languages made it necessary a study about the correlation between classes, packages and files. In order to improve the quality of code and the developers satisfaction.

In the last 20 years there are studied many metrics to evaluate a coding and to limit bugs. In further detail for instance, the Chidamber and Kemerer (CK) object-oriented metrics suite, based on coupling between objects, numbers of children or depth of an inheritance tree.

We will focus on different levels of granularity (class, method or package), paying attention on a featuring selection like to reduce or to combine metrics.

In this work using the last Data analysis approach (neural network, decision tree, linear regression), it was possible to obtain models in order to predict bugs based on five relevant software system (Eclipse JDT Core - Eclipse PDE UI etc.). There models based on the purpose to have an exact prediction of numbers of bugs or to have the knowledge of a present / absence of bugs during the execution or programming.

This is useful, essentially for the developers to understand the relationship between bugs and software, because the modern laugages programming have a lot dependencies, just think about library, operative system and the several versions.

Using the CK metrics have prove that correlation between metrics is more reliable with some software system (eclipse and equinox). To obtain the same level of correlation with other software we need to combine in a specific approach the metrics (with also some interpolations).

We have developed a hybrid metric, correlated with the bug accurence return a level about 0.8 and 0.9 of correlation using the neural network and decision tree.

While the classification approach, (bug present or absence) developed for the first time with these programs, using the combined metrics, the result will show very good values of precision, recall and accuration.

## 2. OVERVIEW

Bug prediction is one of relevant problem into Software Engineering world. The idea to limit this issue is to use a knowledge system, like database that contains for each class or package if this one can lead an error or not.

Another way to recognize errors into the workplace is exchange reference, opinions between colleagues, just dropping off the bug's reference [i] but also use the bugs history. Bearing in mind, we have to try a trade-off between bug management and the coding.

For instance the bug history of 70.000 bugs used by [ii] was able to predict predict the overall bug-count trend monthly. So for the nexts months knowing the past the system can avoid bugs.

### 2.1. Granularity level

It is relevant which granularity level we will focus on.

We can work with different level to predict bugs (package-level, file-level, class-level and method-level). Below are explain the principal study about different level approach.

In the 2010, it was apply a correlated-study between bugs and effort-aware models [iii]. The **package-level** prediction can be unsuitable if we consider the effort.

For effort have considered the lines of code (LOC) like proposed by [iv]. It was compared with a file-level approach and this benchmark show that using only 20% test to detect up over 70% of bugs but using package-level we have almost 60%. This happens for a high dependency between files and release in a package-approach.

Now a **file-level** prediction combined with a classifier are able to recognize bugs clean or 'buggy' (hidden). It makes to several advantages like small granularity infact it is not necessary a semantic information about the source code and it works well with different projects and programms languages [v].

The **method-level** prediction have various benefit combined always with a classifier. First, when the file is too large, the developer needs a lot times to examine all method into a file, it is increased the granularity model. In this casa, it's suggest to use the AUC (Area under the curce ROC) to create a better model, this because when selecting randomly a bug-prone (inclined to error) and a not bug-prone method, AUC represents the probability that a given classifier assigns a higher rank to the bug-prone method. [vi]

A successful approach is **graph-level**, it is able to develop a software evolution. In the work [vii] there are studied various graphs to detect propertied (Assortativity, Clustering Coefficient, Modularity Ratio etc.) about software process and these results are applied with many software. They finally prove that some measure related with the bugs as for instance 'Modules with higher ModularityRatio have lower associated maintenance effort'. When for ModularityRadio aims ratio between the total number of intra-modules and total number of inter-modules.

A bug-fix time prediction model with Multivariate Regression Testing is a way to predict information based on times and the correlation between independent variable and dependent variables. [viii]

Another type is historical metric-based. To evaluate this approach there were used prediction methods on different level (packages, files, methods). During the first analysis it showed that the method-prediction have larger percentage of bugs found this because this approach was apply with LOC (lines of code) and intuitively, comparing the median value of the LOC, methods are almost ten times smaller than files, and are from thirty to threehundred times smaller than packages.[ix]



## 2.2. Developers approach

The software's quality is also and good improvements. Experimentally changing the developer's number also change the number of defect fixex by them, this change the defect prediction quality. [x] Using a concept drift, seeing the prediction quality over time and using different features, so metrics shows that changing the number of author we have a negative impact.

Another negative impact comes from the number of lines added / removed to fix bugs relative to total number of lines operates (This feature reflects the fraction of work performed to fix bugs relative to total work done).

Beyond all, it is interesting the developer's behavior with the bug prediction. There are developed algorithms in relation with the developer-preference in terms of characteristics. Finally, this is evidenced by the developers prefer the algorithm that have file with a larger numbers of closed bugs through the human inspection techniques. [xi]

Another point of analysis is about the relationship between the metric and the developers. In the study was studied specific metrics to understand the relationship between the developers. Using an Eclipse plug-in, they have captured the variations of the files in correlation with the developers. The analysis performed using regression and decision tree, with the cross-validation technique have been able to return level of prediction about 80%. [xii]

## 2.3. Different types of approach

It is significant reducing features: but which type of features can we erase? To answer that question it is interesting the approach by [xiii]. Here it is used an iterative algorithm Gain Ratio based, which evidence an improvement of scalability and response-time to find a optimal classification performance into of a iterative-algorithm using the accuracy and F-measure.

In order to reduce the features the approach explained in [xiv] is intuitive. Here it is used a method capable to use simply metrics to predict bugs based on trade-off

between cost and accuracy. In this case, a good metric is LOC (Lines of codes) and it is the metrics that we will study more detailed into the metric section, however the same approach to find a metric is used in the our metric (Section 7). The final consideration cited in the paper used within the analysis, which are the LOC, CBO and LCOM use to optimize the minimum metric subset.

We can also combine different features such that to obtain a good predictive capability. Related with the likelihood of inserting a fault during development or the likelihood of discovering and fixing a fault prior to product release. <sup>[xv]</sup> During the test-phase, they realized that some features combined, return a good contribution using various methods (wrappers and filters). Wrappers are algorithms that use feedback from a learning algorithm to understand which attributes used to build a predictive model. The last method (filters) does not require a learning algorithm to determine the attributes selection.

A useful analysis could be the Mining Software Repositories (MSR) using topic models, which it allows to scale thousands or millions of documents. <sup>[xvi]</sup>

The part of selection defects bugs is usually trial (right to think about the memory corruption, that allow to manifest a bugs after long times). In this works it is allow to each bug predictor to signal anomalies into the software, to achieve this, it is developed a dynamic forward slices of each anomaly and retain only those anomalies whose forward slices contain the point of failure. Using this approach, they have used this method with gcc compiler, which performs many bugs, without using, any hardware's support and it obtain very relevant results. <sup>[xvii]</sup>

The bug estimation work using clustering techniques with the information stored into the bug's repositories is innovative. Using cluster to separate different bugs and to join similar bugs is possible to resolve problems that have been in the past, in order to save time. <sup>[xviii]</sup>

Reducing the data scale and improve the accuracy of bug triage can save many resources. The heart of the algorithm is the instance selection and the feature selection with the removing of the noise or duplicate information. They prove that the feature selection can supplement the loss of accuracy by instance selection. Thus, we apply instance selection and feature selection to reduce simultaneously the data scales. <sup>[xix]</sup>

An important point where that should be in-depth is about the re-opened bugs. Based into two phases: In the training phase, our goal is to build a classifier from the historical bug reports, which have known. In the prediction phase, this classifier predict whether an unknown bug report would be re-open or not. In the results shows that the Bagging and Decision Tree had a good performance (up on 90 %).  
[<sup>xx</sup>]

## 2.4. Last study with same dataset

In the study developed into [<sup>xxi</sup>] was apply the linear regression with the same Data set. Here are developed different methods also with a combination of different metrics valuated using Spearman correlation and  $R^2$ .

These indexes have different behaviour, so:

- **Spearman to evaluate the predictive power or the exactly number of a bugs for each classes;**
- **$R^2$  to evaluate the explanative power or the present or absence of a bug for a class.**

The principal results come from this study were:

- Approaches based on churn and entropy of source code metrics have good and stable explanative and predictive power than the other combined metrics;
- Using CK+OO metrics is not necessary historical information to predict bugs;
- The use of a single metrics are inefficient (this is true also with our approach);
- The combination of OO metrics with WCHU and LDHH the explanative and predictive power are high level.

### 3. BACKGROUND

Fayyad defines a Data Mining like a process of nontrivial extraction of implicit, previously unknown and potentially useful of information from the data stored in a database. [xxii]

The wide variety of Data mining techniques made possible to resolve problems in a faster way. The domains applied are multiple.

The information collects into the data hide a very useful knowledge that in many cases are underestimated.

Into the healthcare organizations, using these types of methods is possible to reduce the cost on clinical test, to speed up the diagnosis and to improve the quality of a clinical decision. In the hospital long time ago, data used to answer at simple statistics questions, but the real goal is to answer to complex queries, more specifically.

In the works developed by [xxiii] are used three principal data mining tools (Decision Tree, Neural Network and Naïve Bayes) to predict heart disease. The results prove that the almost the 80% of the heart patients have been predicted with these techniques. Given patients, medical profiles predict those who diagnosed with heart disease are. Alternatively, identify the relationship and influences in the medical input associated with the predictable state heart disease, or determine the attribute values that differentiate nodes favoring and disfavoring the predictable states: patients with heart disease patients with no heart disease. [xxiv]

Another valuable work is the Heart Diseases from [xxv] done in India. The dataset contains 303 rows with 76 attributes. The accuracy of the prediction with the different techniques used is always up on 80%.

This has shown us the good performance about the use of the Data Mining tools respect to the precision and to the time-response. These two parameters will be fundamental for our approach.

Moving on, an inspirational work of the Data mining have been developed by [xxvi] for the financial distress in China between the years 2011-2018. The use of Data Mining algorithm is for grasping the profitability situation and to anticipate the investment related losses, using the three most famous classifiers (Neural network,

Decision Tree and Support Vector Machine). The goal is to foresee the sign of financial deterioration of the firm early on, using 107 companies with 31 financial indicator.

The model with a higher accuracy was the Neural Network because it did not make any assumption about the statistic distribution or properties of the data and because the neural network had an ability to fit nonlinear data and could approximate accurately complex data.

### **3.1. NEURAL NETWORK**

The neural network is a predictive model with a number of parameters. The disadvantages is the not-clearly of the interpretation of the model. Therefore, when we have inserted the target of the model we can choose different algorithms to implement the neural network. We can choose four three of objective: Standard Model, Enhance model accuracy and Enhance model stability (See Modeler References). [xxvii]

The algorithm about the building of neural network based with back-propagation method. The advantages about these approaches are good successfully, reliability estimation, software cost prediction, ability to approximate complex nonlinear function. It is better for the supervised algorithm than the unsupervised algorithm. In the paper [xxviii] is explained the way to operate of the algorithm respect to detect bugs.

The work [xxix] shows a study using the classic metrics (Code Churn, Chidamber and Kemerer etc.) with a subclasses of eclipse. In this work with the spearman-correlation, (it allows no assumptions about the distributions, variances and the type of relationship) shows that the use of neural network is not always adapt to predict bugs. In particular, in this case the target was to identify the present or the absence of bugs for a set software system.

The hybrid model ‘Artificial Neural Network (ANN) optimized by Artificial Bee Colony (ABC)’, is a study developed in [xxx]. In this work, the dataset divided in order to use a subset of features, instead to use all metrics, with also the PCA. By combining the algorithm genetic idea and using a specific fitness function, it is able to obtain the better neural, like a greedy-approach. Based with the classic approach (recall, precision, ROC, etc.) they have to prove that using less features is possible to obtain a good level of accuracy and cost level.

A work using NNA (neural network algorithm) proposed [xxxi].based with three methods:

- 1) reused the training set in order to obtain a betel level of effort
- 2) In order to prevent the bad interpretation of any features it is performed a feedback to train the model again iteratively to obtain prediction more precisely
- 3) It is used a Bayed method to see if some prediction could be found in a simple way.

A comparison about different types of bugs using various machine-learning techniques is the study [xxxi]. Here there are used Support vector machine, Naïve Bayes, K-nearest neighbors and neural network. Using their dataset using neural network have obtained a better accuracy than the SVM and K-NN. This it clear with the 10 cross-validation that allow seeing the accuracy about 96% for NN respect 90% for SVM. This is one of several reasons about the use of NN.

Another point about the prediction of bugs is the location where the bugs can verify. The method proposed by [xxxi] is fast and reliable. The neural network is developed by back-propagation and into the last layer is shows the success or the failure of a program. Using this method, they are able to identify suspicious code of a given program in terms of its probability to contain bugs.

## Explication neural network

A neural network taken the inspiration by the human's neuron. Like in the nervous system, the neurons interconnected by synapse into the neural networks the synapses are the weight of the interconnection between nodes.

The neural network composed by three levels:

- Input layer: This layer has in input the different fields of analysis. In our case, it made with the metrics, which we considered.
- Hidden layer: This layer adapt the weight to minimize the error using the back propagation algorithm. In this phase, we have only numbers that will be adapted in order to obtain a good error.
- Output layer: The last layer represents the output of the model. In this case, it can be assume two values (bugs or not-bugs).

Each neurons of the models have an activation function that describes the relationship between the input and the output of the neurons.

Possible activation function are:

- Binary activation function (Discrete or Digital)  $\{0,1\}$  or  $\{-1,1\}$  :

$$a_i(t+1) = \begin{cases} 1, & \text{if } \sum_j w_{ij} * a_j(t) - \theta_i \geq 0 \\ 0, & \text{Otherwise} \end{cases} \quad (3.1)$$

$$a_i(t+1) = \begin{cases} 1, & \text{if } \sum_j w_{ij} * a_j(t) - \theta_i \geq 0 \\ -1, & \text{Otherwise} \end{cases} \quad (3.2)$$

- Sigmoidal logistic function  $[0,1]$  :

$$f(x) = \frac{1}{(1 + e^{-x})} \quad (3.3)$$

- Hyperbolic tangent [-1,1] :

$$th(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad (3.4)$$

- Rectifier function :

$$f(x) = \max(0, x) \quad (3.5)$$

- Softmax activation function :

(Used in the output layer of a Neural Network. It corresponds to a class. Softmax function  $f_i$  for neuron  $i$ , out of neural network.)

$$f_i(net_i) = \frac{e^{net_i}}{\sum_{k=1}^N e^{net_k}} \quad (3.6)$$

The algorithm of the neural network divided into two categories:

- Supervised: It is used when is know the possible values of the output a priori. Which is the output values fixed. In our case, this is a supervised algorithm because there are possible only two values for output: bugs or not bugs. Other supervised algorithm are in medicine when we are interesting to a specific disease but we know what the possible disease we can have.
- Unsupervised: Apply when the possible output is unknown. In particular, we cannot understand a priori what the domain is. This is most oriented to the artificial intelligence because the network is capable to learn and the possible values of output are unknown.



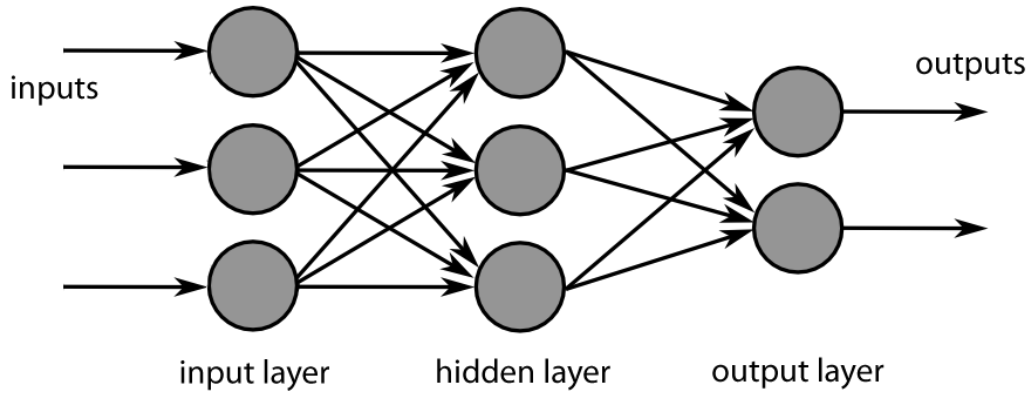


Figura 3.1. Multi-Layer Neural Network.

### Back-propagation algorithm

Back propagation algorithm consists on efficient back-propagation error calculations through the layer of a NN. It is based with a general-purpose numerical optimization method and it rapresents the central learning algorithm. (Known as the reverse mode of automatic differentiation).

Then explained briefly the back-propagation algorithm:

1. Feed a particular training pattern  $p$  to the network input and move forward to calculate all the neuron net inputs and outputs:

$$net_i = \sum_j w_{ij} * S_{jp} ;$$

$$S_{ip} = F (net_{ip}) \quad (3.7)$$

2. Calculate  $\delta_{ip}$  for the output layer:

$$\delta_{ip} = (t_{ip} - S_{ip}) * F'(net_i) \quad (3.8)$$

3. Gradient back-propagation.

Calulate  $\delta_{ip}$  for the layer preceding the output layer right back to the input neurons using the recursive formula:

$$\delta_{ip} = F'(net_i) \sum_k \delta_{kp} * w_{ki} \quad (3.9)$$

The index k follows the numbering of all the nodes to which the ith neuron was connected.

4. Calculate the weight increments of the connections making up the neural network

$$\Delta_p w_{ij} = \mu \delta_{ip} * S_{jp} \quad (3.10)$$

5. Update the values of the network weights:

Incremental (on-line) updating. Weights updated for each p.

$$w_{ij}(t+1) = w_{ij}(t) + \Delta_p w_{ij} \quad (3.11)$$

**Batch updating.** The weight increments accumulated for all the learning patterns. Then the update computed by.

$$w_{ij}(t+1) = w_{ij}(t) + \sum_p \Delta_p w_{ij} \quad (3.12)$$

6. The algorithm finishes (stop condition) when the mean square error (MSE) is less than a stated threshold:

$$MSE = \frac{1}{P} \sum_p E_p \quad (3.13)$$

The calculation of  $\delta_{ip}$  involves the derivative of the activation function  $F(x)$ . Sigmoid or tanh activation functions. They are themselves involved in their respective derivatives expressions, what speeds up learning:

Logistic:

$$f(x) = \frac{1}{(1 + e^{-x})} \quad F'(x) = F(x) * [1 - F(x)]$$

Hyperbolic tangent:

$$th(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad F'(x) = 1 - F^2(x)$$

Obviously, there are many disadvantages about the use of neural network. Some examples are:

- Vanishing or exploding gradients. Fundamental Deep Learning Problem  
Cumulative back-propagated error signals either shrink rapidly, or grow out of bounds. They decay exponentially in the number of layers, or they explode. The result is that the final trained network converges to a poor local minimum.
- Slow convergence. Choice of the learning rate  $\mu$ : If  $\mu$  is too large, the optimum can overshoot, leading to oscillations. An overly small value for  $\mu$  can draw out the learning time.
- Overfitting. Multiple non-linear hidden layers make deep ANNs very expressive (powerful) models that can learn very complicated relationships between inputs and outputs, including noise not existing in testing data. This leads to poor predictive performance, without a generalization, capabilities correctly predict unseen data.
- High powerful computational resources as deep learning involves many hidden layers.

- Local optima or early convergence. The random choice  $(-1, 1)$  of the set of weights as a starting point to the algorithm is not adequate.

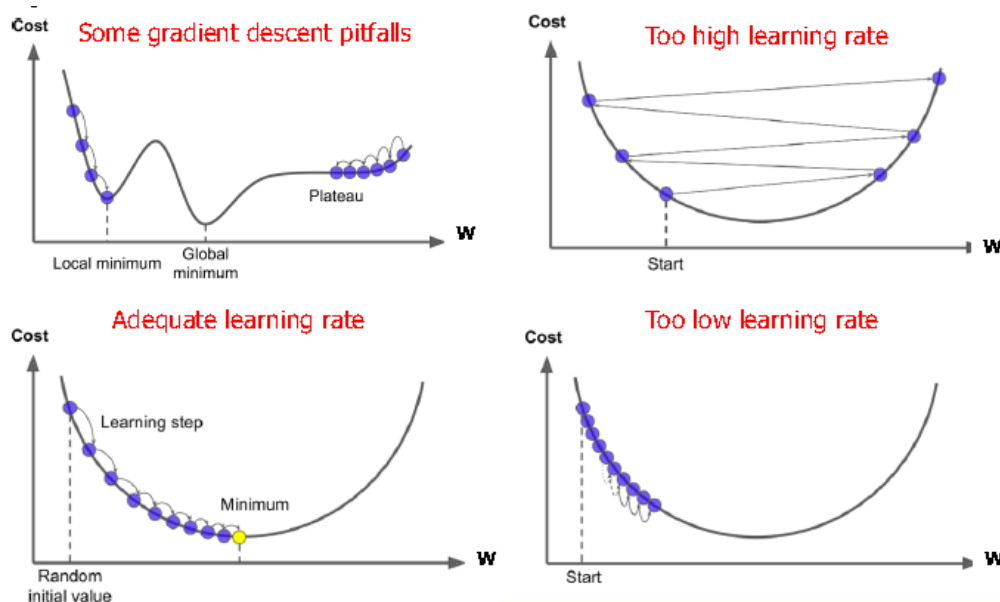


Figure 3.2 Slow convergence.

### Solution to slow convergence

- Again, adequate kernels initialization.
- Momentum factor and variations.
- Adaptive momentum and learning rates: from large to small momentum factor. On the contrary, from high learning rate (typically 0.1) to small (exponential decay).
- Neuroevolution and memetic approaches.
- Unsupervised pre-training with autoencoders

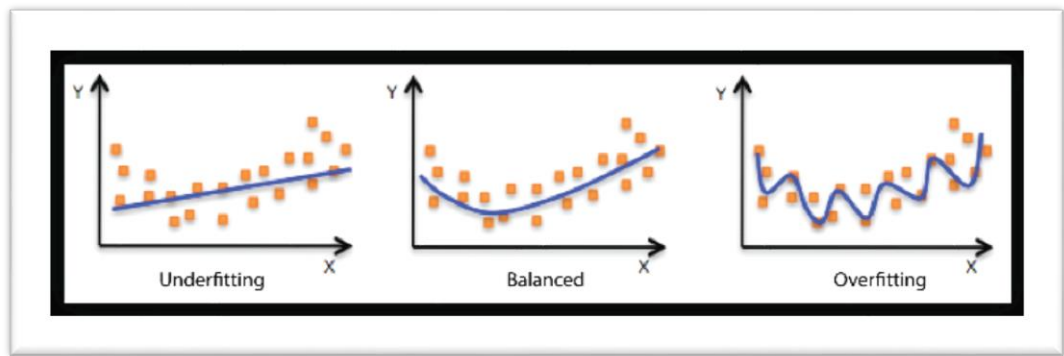


Figura 3.4 Overfitting

### Solution to overfitting

- Early stopping as soon as performance on a validation set starts to get worse.
- Introduce weight penalties of various kinds such as  $\ell_1$  and  $\ell_2$  regularization.
- Dropout technique. It randomly drop hidden and input neurons along with their connections from the NN during training. A neuron is present with probability  $p$ .  $P=0.5$  for hidden neurons and close to 1 for input neurons.
- Simplify NN (make it shallower) by encoding raw input data in a less redundant or more compact way through autoencoders.
- Unsupervised pre-training. E.g. Autoencoders. Evolutionary search for simple neural architectures

## 3.2. DECISION TREE

The decision tree is a predictive model very easy to understand, because just need to walk down in the tree to know the objective-target. This model built using the C&R-tree algorithm. Therefore, the node starts examining the input field to it to find the best split using an impurity index. Each separation is in two subset choosing the node with a best measure. In the setting is possible change the maximum number of fields, numbers of leaves etc.

### OVERVIEW DECISION TREE

The speed to have a result is important in each intelligent system. In fact, the best way to have bigger performance is about the choice of the fields to use. For example, we can develop a very accurate model but it need about minutes or hours to try a solution and it is unnecessary. The work [xxxiv] shows a study to reduce the time calculations using the decision tree. More in details the researchers were interested about the newly attributes inserted and about the effects which them caused. To allow this the target field divided into two possible values (fast or slow) related to a specific period time. The experiments based with Eclipse and Mozilla show that with the decision tree that level of recall is lower than 0.5. It means that the model misses more than half of fast-bugs. On the other hands with Mozilla, the accuracy with slow-bugs is low (about 0.6). The next study argued is related with the post-submission data of bug reports improves prediction models. It developed just divided the time of analysis in different periods after the creation data of the bug report. This is very performant with Eclipse (with a precision of 0.8) but quite accurate.

In this approach is used also the Decision Tree. In this data mining tool is used the Gini index to evaluate the impurity of a node. The Gini index used to investigate how changes made to source code distributed among the developer population. The Gini coefficient initially was a measure for inequality in economy.

In these work the gini coefficient used with different measures of software, means that the gini index based on change data correlate negatively with the number of bugs.[<sup>xxxv</sup>]

A simple way to be able to predict bugs is with matching with others bugs using various similarity index (Dice Similarity, Cosine Similarity, TF-IDF Similarity and Jaccard Similarity).[<sup>xxxvi</sup>]

The parallel work proposd by [<sup>xxxvii</sup>] in order to improve the certification results using the decision tree is useful. Here, there were combined different source code changes come from developers. Of course, the certification-time could be very large and it can take the total working-time. Using the decision tree proposed this problem can be resolved, calculating the probability of a build failing or passing the certification procedure. There are many factor, which cause certification issue (Social, Technical, coordination, etc).

The decision tree is devoped based with various effects to ensure the validity and the reliability.

Returning about save-time, in the 2014 by [<sup>xxxviii</sup>] has been develop a method to reduce the median time to identify a blocking bug. A blocking bug are software defects that threaten other software systems.

Using decision tree was develop a way to reduce the blocking bugs for the developers. Using different resources cited in the article the algorithm can predict bugs that became blocking bugs with an accuration explained in terms of F-measure about 15% – 40 %. Just see the resulting-model is possible to select the most important factors to determine the blocking bugs and it are reletes with the CC metrics.

The algorithm to predict the total amount of time to fix the bugs based with an empirical distribution and a Markov method to predicting the numbers of bugs.

Briefly, using the Markov model, we can create a matrix where each element represent the probability of a defect transferring from fixed or not fixed. We can estimate at the next period time the total numberof bugs; just apply a regression analysis with the historical data.

Additionally, we are able to estimate the total time to resolve bugs using a Monte Carlo based method with the  $R^2$  and the Standard error.

Finally using the classic method of comparing (recall, precision, F-measure) the best model is the decision tree in terms of speed to fix a bug. [xxxix]

### **Explication Decision Tree**

The Decision Tree is a tool that use the tree-structure to apply a classification. Having a simply interpretation, just going down, to solve the question on the nodes, we obtain a classification.

Many parameters to considerate and two principal algorithm develop a decision tree (C&R and CHAID).

Important factors are the split and pruning index. However into the machine learning the decision tree is a predict model where each nodes represents a variable and an edge towards child-node represent a possible path.

There are advantages and disadvantages related to this one.

The advantages are:

- Easy to understand and to interpret. (especially with the rules-set extract)
- Allow the addition of new possible scenarios
- Can be apply to different date categories
- Very fast

The disadvantages are:

- The results can need a lot of time if are too similar.
- It can be innacurate
- It is useful set a stopping time appropriated



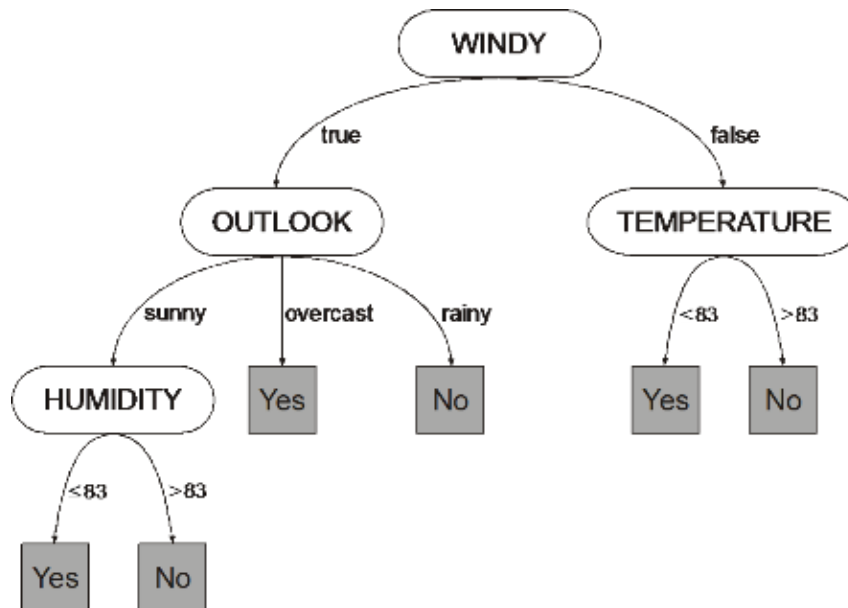


Figura 3.5 Decision Tree example

## CHAID

CHAID (Chi-squared Automatic Interaction detection) is a statistical technique to evaluate predictor fields. It selects the best predictor to be able to have the first branch of the tree such that each child have homogeneous values. The process continues in an iterative procedure until the tree is completed.

We use the F-Test if the target is continuous and Chi-Squared if the target is categorical.

There is an extension of this algorithm called Exhaustive-Chaid. Here, in order to obtain the best split for the prediction the results are compared using the p-value.

Some possible options:

### **Frequency fields:**

Using a frequency fields we can reduce the dataset.

### **Case weights:**

The contribution of the record weighted in proportion to the numbers of the elements that the records represent.

### **Binning of scale-level predictors**

It discretized automatically ordinal categories, like grades or ranks (1 – 10):

- Data  $y_i$  are sorted;
- Calculate the relative weights frequency ( $cf_i = \sum_{y_k < y_i} w_k$ )
- Determine the bin to which it belongs the value checked with the relative frequencies

$$bin\ index = \frac{g}{W + 1} * 10$$

$W =$  total weight frequency

$$\sum_i w_i$$

$$g = \begin{cases} c f_{i-1} + \frac{w_{i+1}}{2} & \text{if } w_i \geq 0 \\ c f_{i-1} + \frac{w_i}{2} & \text{if } w_i < 0 \end{cases} \quad (3.14)$$

Generally Chainf create k=10 bins by default.

### CHAID Algorithm

Each final categories of a predictor field X will represent a child node, id X is used to splite a node.

1. If X has one or two categories, start to split:
2. To fins the category X that has the minimum different by p-value
3. If a pair has a high p-value, so the pair has merged. Otherwise go to step 6
4. If the users allow splitting of merged categories it splits in three or more categories. It is finds the best binary split with the best p-value
5. Continue to merge categories from step 1 for this fiels
6. Every categories with a numbers of fields minor than the alfa (specific by user) is merged in a similar category.

#### F-Test

The F-Test for the continuous target is calculated:

$$F = \frac{\sum_{i=1}^I \sum_{n \in D} w_n f_n I(x_n = i)(y_i - y)^2 / (I - 1)}{\sum_{i=1}^I \sum_{n \in D} w_n f_n I(x_n = i)(y_i - y)^2 / (N_f - I)} \quad (3.15)$$

$$y_i = \frac{\sum_{n \in D} w_n f_n y_n I(x_n = i)}{\sum_{n \in D} w_n f_n I(x_n = i)} \quad (3.15)$$

$$y = \frac{\sum_{n \in D} w_n f_n y_n}{\sum_{n \in D} w_n f_n} \quad (3.16)$$

$$N_f = \sum_{n \in D} f_n \quad (3.17)$$

$F(I-1, N_f-1)$  is a random variable on F-Distribution.

### Chi-Squared

Chi-Squared is used for the categorical field.

$$\chi^2 = 2 \sum_{i=1}^I \sum_{j=1}^J n_{ij}^* \ln\left(\frac{n_{ij}^*}{n_{ij}}\right) \quad (3.18)$$

$n_{ij}$  = frequency observed ;  $n_{ij}^*$  = frequency expected

If the target field y is ordinal:

$$H^2 = 2 \sum_{i=1}^I \sum_{j=1}^J n_{ij}^* \ln\left(\frac{n_{ij}^*}{n_{ij}}\right) \quad (3.19)$$

### Stopping rules

There are many conditions to decide to stop the split in the tree:

- Node pure
- All records have the same value for a field
- The tree depth is the maximum tree depth
- $N^\circ$  records < minimum parent node size
- $N^\circ$  records in a child node < minimum child node size

### C&RT

C&RT (Classification and Regression Tree) is an approach that separates data into two subset homogeneous. It allows an equal cost to be considered in the growing process. It also allows to specify the priori probability and the distribution to apply an automatically cost-complexity pruning.

### Frequency fields:

Using a frequency fields we can reduce the dataset.

**Case weights:**

The contribution of the record during the analysis weighed in proportion to the numbers of the elements that the records represent.

**Parameters**

C&RT works with the priority of nodes to split the nodes.

How build the C&RT Tree:

- Numerics fields: Sort the fields. Choose each point like the split node and calculate the impurity. Finally select the best field with the impurity index.
- Categorical field: Examine each possible combination of values of two subset and calculate the impurity. Finally select the best field with the impurity.

Identify the field whose best split gives the greatest decrease in impurity for the node and select the best splitted field.

**Blank handling**

To manage the blank fields is apply a surrogate splitting:

If the best field selected to the split have a blank or missing value, it is selected another similar field and its value is used to assign the record to the child node.

Selected by the probability:

$$p_f(t) = \sum_j \frac{\pi(j)Nf_j(t)}{Nf_j} \quad (3.20)$$

Where:

- $Nf_j(j)$  = sum frequency weight;
- $Nf_j$  = sum of frequency weight for the total records.

### Predictive measure of association

Let  $h_x^* \Omega_x$  be the set of learning cases without missing values.

Let  $P(s^* \simeq s_x | t)$  be the probability of sending a case in  $h_x^* \Omega_x$  to the same child by both  $s^*$  and  $s_x$ ,  $\hat{s}$  be the split with maximized probability:

$$P(s^* \simeq s_x | t) = \max P(s^* \simeq s_x | t)$$

The predictive measure of association is:

$$\lambda(s^* \approx \hat{s}_x | t) = \frac{\min(P_L, P_R) - (1 - P(s^* \simeq s_x | t))}{\min(P_L, P_R)} \quad (3.21)$$

where  $P_L$ (resp.  $P_R$ ) is the relative probability that the best split  $s^*$  at node  $t$  sends a case with non-missing value of  $X^*$  to the left (resp. right) child node. And where:

$$P(s^* \simeq s_x | t) = \begin{cases} \frac{\sum_j \pi(j) N_{wj}(s^* \simeq s_x | t)}{N_{w,j}(X^* \cap X)} & \text{if } Y \text{ is categorical} \\ \frac{N_w(s^* \simeq s_x, t)}{N_w(X^* \cap X)} & \text{if } Y \text{ is continuous} \end{cases} \quad (3.22)$$

### Stopping rules

There are many conditions to decide to stop the split in the tree:

- Node pure
- All records have the same value for a field
- The tree depth is the maximum tree depth
- $N^\circ$  records  $<$  minimum parent node size
- $N^\circ$  records in a child node  $<$  minimum child node size

**Profits**

This is the number value associated to the category of a field. It is used to calculate the gain.

$$\sum_j f_j(t)p_j \quad (3.23)$$

Where:

- $j$  = category
- $f_j(t)$  = frequency sum of the record on the node  $j$
- $p_j$  = used-defined profit to the category  $j$

**Pruning**

Pruning refers to remove insignificant nodes. It uses an index to measure the miss-classification and the complexity of the tree.

$$R_\alpha(t) = R(t) + \alpha|\check{T}| \quad (3.24)$$

Where:

- $R(t)$  = miss-classification
- $\alpha$  = complexity cost
- $|\check{T}|$  = Numbers of terminal nodes

**Impurity measures**

The impurity measure depends on the target fields

(Symbolic we can use GINI or TWOING – Continuous we can use LSD)

## GINI

The GINI index defined like:

$$g(t) = \sum_{j \neq i} p(j|t)p(i|t) \quad (3.25)$$

Where:

- $i$  and  $j$  are categories of the target field
- $p(j|t) = \frac{p(j,t)}{p(t)}$
- $p(j, t) = \frac{\pi(t)N_j(t)}{N_j}$
- $p(j) = \sum_i p(j, t)$
- $\Pi(j) = \text{Prob. Category } j$
- $N_j(t) = \text{Numbers of records of the category } j \text{ to the node } t$
- $N_j = \text{Numbers of records of the category } j \text{ from the root node}$

Gini index can be also like:

$$g(t) = 1 - \sum_j p^2(j|t) \quad (3.26)$$



## TWOING

This method splits a target in two superclasses  $C_1$  and  $C_2$  then chosen the best split based on those two categories.

$$c_1 = \{j = p(j|t_L) \geq p(j|t_R)\}$$

$$c_2 = c - c_1$$

The criteria function defined is:

$$\varphi(s, t) = p_L p_R \left[ \sum_j |p(j|t_L) - p(j|t_R)| \right]^2 \quad (3.27)$$

## LSD

LSD (Least-Squared Deviation) is definen:

$$r(t) = \frac{1}{N_w(t)} \sum_i w_i f_i (y_i - \hat{y}(t))^2 \quad (3.28)$$

Where:

- $N_w(t)$  = Numbers of weights of records to the node  $j$
- $f_i$  = frequency for the field  $i$
- $y_i$  = target field value
- $\hat{y}(t)$  = means of the weights to the node  $t$

The criteria function defined is:

$$\varphi(s, t) = R(t) - p_L R(t_L) - p_R R(t_R)$$

### 3.3. LINEAR REGRESSION

In the approach of [xl] the same software system are engineered using only the linear regression. In this method used the classic metrics to evaluate the software. Using CK+OO metrics are able to predict bugs especially for Eclipse, Equinox and PDE.

Using the Weight Churn metric, that perform very good correlation with Spearman and  $R^2$  like in our study with neural network and decision tree.

In the work of [xli] are explained the power of the linear regression to predict bugs, in this case with Eclipse. The principal problem is related with the not knowledge how the bugs are linked to a classes. Here have been studied the software six months before and after the release.

In particular, with file-level only some metrics (TLOC, FOUT) have correlation above 0.4. On the other hands on package-level, many metrics show a level of correlation above 0.4.

A different approach of linear regression proposed into [xlii]. In fact, using non-linear model and temporal features are able to predict bugs with a high correlation. Using more or less the same features (LOC, Releases, AgeMonth,) but only with a flag for the bug (bugs, not-bugs), have to predict a high percentage of bugs comparable with our study. (There are used the same coefficient like Spearman and ROC curve for the model).

### Explication Linear Regression

The algorithm used to represent a linear regression use the least squared with four methods to insert / remove variables.

The summary statistics  $X'$  and covariance  $S_{ij}$  computed using provisional means algorithms to update the values:

$$X'_{i(k)} = X'_{i(k-1)} + (x_{ik} - X'_{i(k-1)}) \frac{wk}{Wk} \quad (3.29)$$

and

$$S_{ij} = \frac{C_{ij}}{C - 1} \quad (3.30)$$

Where:

- $X'_k$  = Sample mean for the  $k$ th input field
- $x_{ki}$  = The value of the  $k$ th input field for record  $i$
- $W$  = The sum of weights across records ( $\sum_{i=1}^l w_i$ )
- $w_i = c_i * g_i$
- $c_i$  = Case weight for record  $i$ ;
- $g_i$  = Regression weight for record  $i$ ;

The regression model has the form:

$$Y_i = \beta_0 + \beta_1 X_{1j} + \beta_2 X_{2j} + \cdots + \beta_p X_{pj} + e_i \quad (3.31)$$

The algorithm compute the least squared estimates b of  $\beta$  and the associated regression statistics.

## Variable entry and Removal

### Automatic field selection

Let  $r_{ij}$  be the element in the current checked matrix associated with  $X_i$  and  $X_j$ . Variables inserted or removed one for time.

$X_k$  is eligible for entry if it is an input field not currently in the model such that:

$$r_{kk} > t \quad \text{and} \quad \left( r_{jj} - \frac{r_{jk}r_{kj}}{r_{kk}} \right) t \leq 1 \quad t \text{ is the tolerance.}$$

The second condition above imposes so that entry of the variable does not reduce the tolerance of variables already in the model to unacceptable levels.

The F-toenter value for  $X_k$  defined as:

$$F_{toenterk} = \frac{(C - p^* - 1)V_k}{r_{jj} - V_k} \quad (3.32)$$

Where:

- 1 and  $C - p^* - 1$  = degrees of freedom
- $p^*$  = number of coefficient currently in the model
- $V_k = \frac{r_{yk}r_{ky}}{r_{kk}}$

The F-toremove value for  $X_k$  defined as:

$$F_{toremovek} = \frac{(C - p^*)|V_k|}{r_{yy}} \quad (3.33)$$

Four methods for entry and removal of variables are available. The selection process repeated until no more independent variables qualify for entry or removal.

Here we have the algorithms for these four methods:

- Enter: The selected input fields are all included in the model, with no field selection applied.
- Stepwise: If there are independent variables, choose  $X'_k$  with the minimum  $F_{to-reove}$ .  $X_k$  is removed if  $F_{to-remove} < F_{out}$ . If there are not present independent variables in the model, or if no variable can be removed, choose  $X_k$  such that  $F_{to-enter}$  is maximum. At each step, all variables considered for removal or input are.
- Forward: This procedure is the entry phase of the stepwise procedure.
- Backward: This procedure starts with all input fields in the model and applies the removal phase of the stepwise procedure.

### 3.4. R-SQUARE

The big capacity to elaborate data made possible to develop a various types of models. A good strategy is to use approximation model, which referred to metamodels.

Metamodelling have good success in many engineering applications. An important study has been proposed by [xliii] able to compare four techniques (polynomial regression - multivariate adaptive regression splines - radial basis functions – kriging) based on different criteria.

Obviously, there are not present methods, which work well than to other. Every times, it is trade-off between efficiency, robustness, model transparency and simplicity.

To prove the accuracy of a model there are possible three metrics:

**R Square:**

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - y_i^*)^2} \quad (3.34)$$

Where:  $\hat{y}_i$  is the corresponding predicted value for the observed value  $y_i$ ;  $y_i^*$  is the mean of the observed values.

The larger the value of R Square, the more accurate the metamodel.

**Relative Average Absolute Error (RAAE):**

$$RAAE = \frac{\sum_{i=1}^N |y_i - \hat{y}_i|}{n * STF} \quad (3.35)$$

Where: STD stands for standard deviation.

The smaller the value of RAAE, the more accurate the metamodel.

### Relative Maximum Absolute Error (RMAE)

$$RMAE = \frac{\max(|y_1 - \hat{y}_1|), (|y_2 - \hat{y}_2|), \dots, (|y_n - \hat{y}_n|)}{STD} \quad (3.36)$$

While the RAAE usually highly correlated with MSE and thus R Square, RMAE is not necessarily. A small RMAE is preferred.

(In our study, we have used the R-Squared). We have choose this one because:

- 1) it was used in the last study so it is useful to compare;
- 2) Like it was discovered by<sup>[xliv]</sup> the R-square works very well with non-linear and linear system to see the difference between models that presents similar features. However, using R-squared we can select the best models to use for engineering applications.

## 3.5. SPEARMAN CORRELATION

Starting with all metrics available it is performed an accuate analysis foo each metrics group using the two tools (Neural Network and Decision Tree).

We have used the Spearman correlation to understand the predictive power and the  $R^2$  to understand the explanative power. We will use also other graphics model to understand in the best way the results. We do not use any particular criteria to characterize the different value obtained in the various version of the model. In some interesting cases, we show the coincidence's matrix to show the some relevant errors between bugs.

## Valuation model

To validate a model there are many techniques which are developed in particular cases, that using specific mathematic functions.

The most relevant coefficient used are Spearman and Pearson. The Pearson coefficient is the covariance of two factors separated by the result of their standard deviation. Used in different environments. While the Spearman coefficient is a measure of a monotone affiliation between two variables without making any assumption about the frequency distribution of the variables.

Both of them have a range between +1 and -1, they are closed to 1 when have a comparative rank in the other case have a divergent rank between two factors.

In this study it is used the Spearman coefficient because the variables are ordinal, if the variable of interest would be continues we have to use the Pearson coefficient.

### Spearman correlation:

$$r_s = 1 - \frac{6 \sum_i d_i^2}{n(n^2 - 1)} \quad (3.37)$$

Where:

- $d_i$  = is the difference between the two ranks of each observation
- $n$  = is the number of observations



### Pearson correlation

$$r_p = \frac{N \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{\sqrt{[N \sum_i x_i^2 - (\sum_i x_i)^2][N \sum_i y_i^2 - (\sum_i y_i)^2]}} \quad 3.38)$$

Where:

- $N$  = Number of pair to score
- $x_i$  = x score
- $y_i$  = y score

To undersrand better the use of these two coefficient based on the study of Hauke and Kossowski [<sup>xlv</sup>]. In fact, in this work, there are studied dataset with both of them and the conclusions are that it depends of the case. We do not assume that one is the best, because sometimes where Spearman returns positive values Pearson return negative values.

### Summary

Here we have used two indicators (Spearman correlation and  $R^2$ ) to indicate the goodness of the model and to compare the results.

Briefly the spearman correlation give us a predictive power more sensitive instead  $R^2$  is correlated with the explanative power in a more generically way.

### **3.6. WHERE USE SPEARMAN**

Spearman correlation is a non-parametric measure of rank correlation. It estimate the relationship between two variables using a monotonic function.

The spearman correlation between two variables is similar to the Pearson correlation between the ranks values of those two variables studied. Pearson used for a linear relationship, spearman assesses monotonic relationship that can be linear or not.

Therefore, the spearman correlation between two variables will be high (up to a maximum to 1) when observations have a similar rank between the two variables. The spearman correlation will be low (up to a minimum of -1) when observations have a dissimilar rank between the two variables.

Now, we can see some example to point out the principal difference with some particular cases.

- Perfect correlation

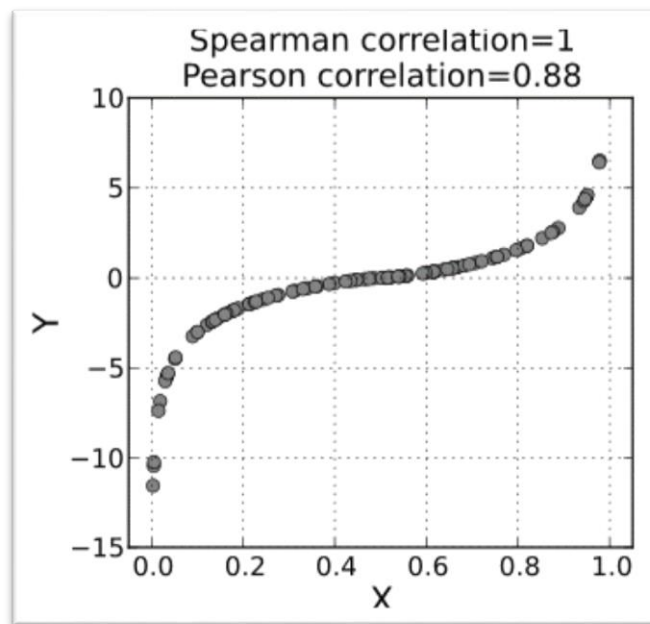


Figura 3.7 Spearman 1

<b>x</b>	5	8	10	12	15	13	8	7	5	3
<b>y</b>	8	15	18	21	25	22	15	13	10	4

A spearman correlation is equal to 1 when the two variables are monotonically related. So for each value of x correspond a value of y at the same grade. If x increase also y increase respect to the other values.

Only in this case we have a perfect correlation for Spearman and Pearson coefficient is close to 0.90

- Bad correlation

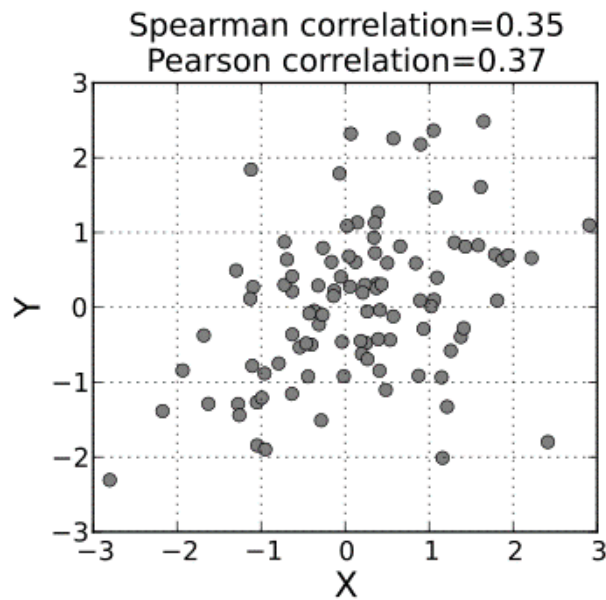


Figura 3.8 Spearman 2

<b>x</b>	5	8	10	12	15	13	8	7	5	3
<b>y</b>	7	18	1	45	13	11	33	57	9	50

When the data are random distributed the coefficient of spearman and Pearson are more or less the same. In this case is not possible to extract any useful information

- Correlation and outlier

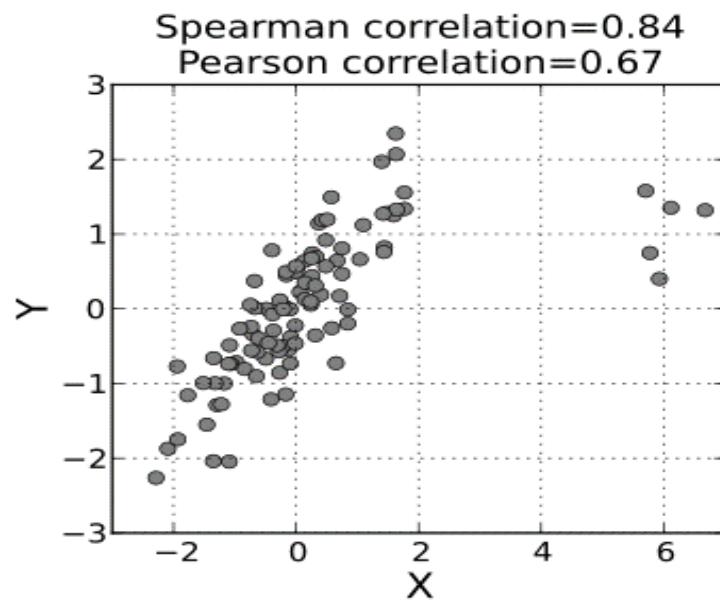


Figura 3.9 Spearman 3

<b>x</b>	5	8	10	12	15	13	8	7	5	3
<b>y</b>	8	15	1	21	25	22	80	13	10	4

In case of outlier is better the use of Pearson. Spearman is less sensitive with the presence of outliers. This behaviour related with the definition of spearman that adapt the value of x and y in a same range, in this way spearman limits the outlier to the value of its rank.

### 3.7. SPEARMAN OR $R^2$

In our analysis to evaluate a model are used two coefficient (Spearman and  $R^2$ ).

Both of coefficient have a range between -1 and 1. 1 means a perfect correlation and -1 a opposite correlation.

In our study of the bugs with different software sytems, the means of these values is different:

- We must see Spearman where we want know exacly the number of bugs for each class in a spefic software based on a metric
- We see  $R^2$  when we are interesting only at the present or absence of bugs in a software based with a metrics. It is used for a more generically vison of the problem.

## **4. METRICS DESCRIPTION**

The metrics used to detect prediction based in two macro-category, ‘change log approach’ and ‘single version approach’. The first one use the bug information obtained from recent or frequently bugs, the second studies the current state with particular metrics to avoid bugs, based especially with mathematical formulations (logarithm, exponentials etc.).

## 4.1. CHANGE METRICS

The table `Change_Metrics` is composed of a set of metrics that combine two approach (Moser and Graves). Each rows in the table represents the bugs comes from the versioning system. Here you have explained each fields how works:

<b>numberOfVersionsUntil</b>	<i>Number of version of these class</i>
<b>numberOfFixesUntil</b>	<i>Numbers of times that a class is cause of a bug</i>
<b>numberOfRefactoringsUntil</b>	<i># of times that a class is re-scheduled</i>
<b>numberOfAuthorsUntil</b>	<i># of authors that has revised a class</i>
<b>linesAddedUntil</b>	<i># of lines added</i>
<b>maxLinesAddedUntil:</b>	<i>Maximum lines added</i>
<b>avgLinesAddedUntil</b>	<i>Average lines added</i>
<b>linesRemovedUntil</b>	<i># lines removed</i>
<b>maxLinesRemovedUntil</b>	<i>Maximum lines removed</i>
<b>avgLinesRemovedUntil</b>	<i>Average lines added</i>
<b>codeChurnUnti</b>	<i>(linesAddedUntil – linesRemovedUntil) for a class</i>
<b>maxCodeChurnUntil</b>	<i>Maximum CodeChurn for all revision of a class</i>
<b>avgCodeChurnUntil</b>	<i>Everage CodeChurn for all revision of a class</i>



<b>ageWithRespectTo</b>	<i>Age of a class in weeks</i>
<b>weightedAgeWithRespectTo</b>	<i>Sum of numbers of weeks for revision 'i' per the numbers of lines added at revision 'i', divided for the sum of the numbers of lines added at revision i</i>

In this study, it was tested whose the best set of metrics are, without considered the splits using by Zimmermann and Graves (NFI and NR).

## 4.2. PREVIOUS DEFECTS

This approach is the most intuitive because we recognize the bugs based with the past bugs. So we can prove to identify a correlation between future bugs and past bug fixed. Here we have selected each fields in order to obtain the best correlation. We have obtained a good model using this metrics. This is easy to understand because we have a big correlation between bugs but it is not good if we want to generalize this model using this type of metrics. Here we have the classification of the different bugs:

<b>numberOfBugsFoundUntil</b>	<i># bugs found until that time</i>
<b>numberOfNonTrivialBugsFoundUntil</b>	<i># no-trivial bugs found until that time</i>
<b>numberOfMajorBugsFoundUntil</b>	<i># major bugs found until that time</i>
<b>numberOfCriticalBugsFoundUntil</b>	<i># critical bugs found until that time</i>

<b>numberOfHighPriorityBugsFoundUntil</b>	<i># high priority bugs found until that time</i>
<b>nonTrivialBugs</b>	<i>Bug classified like non-trivial bug</i>
<b>majorBugs</b>	<i>Bug classified like major bug</i>
<b>criticalBugs</b>	<i>Bug classified like critical bug</i>
<b>highPriorityBugs</b>	<i>Bug classified like high priority bug</i>

### 4.3. SINGLE VERSION CK-OO

The most relevant metric is from Chidamber and Kemerer (CK). There particular metrics respects six properties: Non Coarseness – Non uniqueness – Monotonicity – Nonequivalence of interaction – Interaction increases complexity <sup>[xlv]</sup>. Here you have explained the behavior of these metrics:

<b>WMC</b>	<i>Weighted methods per class. This represent the sum of the complexity of the methods of each class</i>
<b>DIT</b>	<i>Depth of Inheritance tree of the class</i>
<b>RFC</b>	<i>Response for a class. This is equal to the response set, which represents a set of methods that can be executed in response to a message received by an object of the class.</i>

<b>NOC</b>	<i>Number of children. This is the number of immediate subclasses subordinated to a class in the class hierarchy.</i>
<b>CBO</b>	<i>Coupling between object classes. CBO for a class is a count of the number of other classes to which it is coupled.</i>
<b>LCOM</b>	<i>Lack of cohesion in methods. This is measured using a set of instance variables used by method that compose the class.</i>

The other metrics are a set of object-oriented metrics. These metrics depend mostly from the technical property of the class and how it was written. Here there are the description of each field:

<b>FANIN</b>	<i># of other classes that reference the class</i>
<b>FANOUT</b>	<i># of other classes referenced by the class</i>
<b>NOA</b>	<i># of attributes</i>
<b>NOPA</b>	<i># of public attributes</i>
<b>NOPRA</b>	<i># of private attributes</i>
<b>NOAI</b>	<i># of attributes inherited</i>
<b>LOC</b>	<i># of lines of code</i>
<b>NOM</b>	<i># of methods</i>
<b>NOPM</b>	<i># of public methods</i>

<b>NOPRM</b>	<i># of private methods</i>
<b>NOMI</b>	<i># of methods inherited</i>

#### 4.4. ENTROPY OF CHANGE

Here the measure calculated over the time. In particular have been taken measure of the changes during interval of two weeks, with this one it is possible to compute the Hassan Entropy [<sup>xlvii</sup>]. In the last study are used different types of Entropy, but how we can see in further detail after, only two are significant. Here we have the details of each Entropy:

<b>HCM</b>	<i>History of complexity metric</i>
<b>WHCM</b>	<i>Weighted history of complexity metric</i>
<b>EDHCM</b>	<i>Exponentially decayed Hcm</i>
<b>LDHCM</b>	<i>Linearly decayed Hcm</i>
<b>LGDHCM</b>	<i>Logarithmically decayed Hcm</i>

The basic formule is:

$$H_n = - \sum_{k=1}^n p_k * \log_2 p_k \quad (4.1)$$

Where:

$P_k$  = it is the probability that the file k changes during the considered time interval

Considered an example with three files and three-time interval:

<b>T1</b>	$p_A = \frac{2}{4}, p_B = \frac{1}{4}, p_C = \frac{1}{4}$
<b>T2</b>	$p_A = \frac{2}{7}, p_B = \frac{1}{7}, p_C = \frac{4}{7}$
<b>T3</b>	$p_A = \frac{1}{3}, p_B = \frac{1}{3}, p_C = \frac{1}{3}$

In T1 the entropy  $H_1 = - \left( \frac{2}{4} * \log_2 \frac{2}{4} + \frac{1}{4} * \log_2 \frac{1}{4} + \frac{1}{4} * \log_2 \frac{1}{4} \right) = 1$

In T2 the entropy  $H_2 = - \left( \frac{2}{7} * \log_2 \frac{2}{7} + \frac{1}{7} * \log_2 \frac{1}{7} + \frac{4}{7} * \log_2 \frac{4}{7} \right) = 1.378$

In T3 the entropy  $H_3 = - \left( \frac{1}{3} * \log_2 \frac{1}{3} + \frac{1}{3} * \log_2 \frac{1}{3} + \frac{1}{3} * \log_2 \frac{1}{3} \right) = 1.585$

Like cited in [xlvi], to compute the likelihood that a file changes, it is used the amount of change by measuring the number of modified lines. The Adaptive Entropy defined:

$$H_n' = - \sum_{k=1}^n p_k * \log_{\tilde{n}} p_k \quad (4.2)$$

Where:

$\tilde{n}$  = number of recently modified files.

To use the entropy of code change as a bug predictor, Hassan defined the History of Complexity Metric (HCM) of a file  $j$  as

$$HCM_{\{a,...,b\}}(j) = \sum_{i \in \{a,...,b\}} HCPF_i(j) \quad (4.3)$$

and

$$HCPF_i(j) = \begin{cases} c_{ij} * H_i' & j \in F_i \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

Where:

- $i$  = is the period with entropy  $H_i'$ ;
- $F_i$  = is the set of files modified in the period  $i$
- $j$  = is the file belonging to  $F_i$

Now we can define more specifically the metrics:

- HCM:  $c_{ij} = 1$ . Every file modified in the period  $I$  gets the entropy of the system in the considered time interval;
- WHCM:  $c_{ij} = p_j$ . Each file have the entropy of the system with the probability of the file being modified
- EDHCM :  $\sum_{i \in \{a,...,b\}} \frac{HCPF_i(j)}{e^{\varphi_1 * (|\{a,...,b\}| - i)}} \quad (4.5)$
- LDHCM :  $\sum_{i \in \{a,...,b\}} \frac{HCPF_i(j)}{\varphi_2 * (|\{a,...,b\}| + 1 - i)} \quad (4.6)$
- LGDHCM :  $\sum_{i \in \{a,...,b\}} \frac{HCPF_i(j)}{\varphi_3 * (|\{a,...,b\}| + 1.01 - i)} \quad (4.7)$

Where:

- $\varphi_1$ ,  $\varphi_2$  and  $\varphi_3$  are decay factors.

## 4.5. CHURN OF SOURCE CODE METRICS

This metric provides for sampling the history of the source code every two weeks and it computed the deltas for each consecutive pair of samples [<sup>xlix</sup>]. So we have created two tables one for the CC metrics and another for the OO metrics. Finally, It was calculated the churn for each class of the software system that we have explained here:

<b>CHU</b>	<i>Churn. This compute the sum of deltas between samples for a class</i>
<b>WCHU</b>	<i>Weight churn. This is a particular type of churn with a weight of a 0.01</i>
<b>LDCHU</b>	<i>Linear Churn.</i>
<b>EDCHU</b>	<i>Exponential Churn</i>
<b>LGDCU</b>	<i>Logarithmic Churn</i>

For each source metrics, we have created a matrix. The rows represents the classes and the columns the sampled versions (every two weeks). If the class does not exist in a specific version is used the value -1. After we create the matrix of deltas, where each cell is the absolute value of the difference between two consecutive of a metric for a class (if the class does not exist in a version is used the value -1).

Below show an example:

	Release				
classes	1.1	1.2	1.3	1.4	1.5
a	5	5	5	5	-1
b	6	6	7	7	9
c	-1	-1	5	5	5

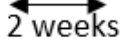

  
2 weeks

Figura 4.1. Churn metrics

The accurate formula are show here:

- $CHU(i) = \sum_{j=1}^C \begin{cases} 0 & \text{delta}(i,j) = -1 \\ PCHU(i,j) & \text{otherwise} \end{cases} \quad (4.8)$

$$PCHU(i,j) = \text{delta}(i,j)$$

- $WPCHU(i,j) = 1 + \alpha * \text{delta}(i,j) \quad \alpha = 0.01 \quad (4.9)$

- $EDPCHU(i,j) = \frac{1+\alpha*\text{delta}(i,j)}{e^{\varphi_1*(C-j)}} \quad (4.10)$

- $LDPCHU(i,j) = \frac{1+\alpha*\text{delta}(i,j)}{\varphi_2*(C+1-j)} \quad (4.11)$

- $LGDPCCHU(i,j) = \frac{1+\alpha*\text{delta}(i,j)}{\varphi_3*\ln(C+1.01-j)} \quad (4.12)$

Using, for example the formule WPCHU, we obtain:

- $a \rightarrow 1+0.01*\text{delta}(5,5) + 1+0.01*\text{delta}(5,5) + 1+0.01*\text{delta}(5,5) + 1+0.01*\text{delta}(5,-1) = 1+0.01*0+1+0.01*0+1+0.01*0+1+0.01*0 = 4$
- $b \rightarrow 1+0.01*\text{delta}(6,6) + 1+0.01*\text{delta}(6,7) + 1+0.01*\text{delta}(7,7) + 1+0.01*\text{delta}(7,9) = 1+0.01*0+1+0.01*1+1+0.01*0+1+0.01*2 = 5.03$
- $c \rightarrow 1+0.01*\text{delta}(-1,-1) + 1+0.01*\text{delta}(-1,5) + 1+0.01*\text{delta}(5,5) + 1+0.01*\text{delta}(5,5) = 1+0.01*0+1+0.01*0+1+0.01*0+1+0.01*0 = 4$

$$1 + 0.01 * \text{delta}(5,5) + 1 + 0.01 * \text{delta}(5,5) + 1 + 0.01 * \text{delta}(5,5) + 1 + 0.01 * \text{delta}(5,-1) = 1$$



## 4.6. ENTROPY OF SOURCE CODE METRICS

The last metric is a combination of two metrics (Entropy and Churn). Here it was calculated the measure of entropy using the deltas history of the class and we have obtained different types of metrics which are useful in different cases.

For example, if in the system the metric CBO changed by 200 (but only a class is involved) the entropy is minimum. Otherwise, is 10 classes are involved with a local change of 10 CBO the entropy is higher.

Like in the churn metrics, to compute the entropy of source code metric we start from the matrices of deltas. We define the entropy for each metric (column) and classes (rows).

<b>HH</b>	<i>Entropy</i>
<b>HWH</b>	<i>Weight entropy</i>
<b>LDHH</b>	<i>Linear Entropy</i>
<b>EDHH</b>	<i>Exponential Entropy</i>
<b>LDGHH</b>	<i>Logarithmic Entropy</i>

The formule more detailed are explained here:

$$H'_{WMC}(j) = - \sum_{i=1}^R \begin{cases} 0 & \text{if } \Delta(i,j) = -1 \\ p(i,j) * \log_{R_j} p(i,j) & \text{otherwise} \end{cases} \quad (4.13)$$

$R$  = numbers of rows of the metrix

$R_j$  = numbers of cells of the column  $j$  greater than 0

$p(i,j)$  = measure of the frequency of change of the class  $i$  for a given metric

$$p(i,j) = \frac{\Delta(i,j)}{\sum_{k=1}^R \begin{cases} 0 & \text{if } \Delta(i,j) = -1 \\ \Delta(i,j) & \text{otherwise} \end{cases}} \quad (4.14)$$

Given a metric, for example CBO, and a class corresponding a row  $i$  the history of entropy is:

$$\bullet \quad HH_{CBO}(i) = \sum_{j=1}^C \begin{cases} 0 & \text{delta}(i,j) = -1 \\ PHH_{CBO}(i,j) & \text{otherwise} \end{cases} \quad (4.15)$$

$$PHH_{WMC}(i,j) = H'_{WMC}(j)$$

This approach is apply for each metric define in the Source Code Metric respect to each classes.

Here the variants:

$$\bullet \quad HWH(i,j) = p(i,j) * H'(j) \quad (4.16)$$

$$\bullet \quad EDHH(i,j) = \frac{H'(j)}{e^{\varphi_1 * (C-j)}} \quad (4.17)$$

$$\bullet \quad LDHH(i,j) = \frac{H'(j)}{\varphi_2 * (C+1-j)} \quad (4.18)$$

$$\bullet \quad LGDHH(i,j) = \frac{H'(j)}{\varphi_3 * \ln(C+1.01-j)} \quad (4.19)$$

## 5. DATASET DESCRIPTION

The Dataset is composed by six different software systems, each of them represented through the measure of metrics described before for every classes.

The Dataset are:

- Eclipse JDT Core;
- Eclipse PDE UI;
- Equinox Framework;
- Lucene;
- Mylyn;

### Eclipse JDT Core

Eclipse JDT Core is the Java infrastructure of the Java IDE. It includes many components like:

- An incremental Java compiler. Implemented as an Eclipse builder, it based on technology evolved from VisualAge for Java compiler. In particular, it allows to run and debug code which still contains unresolved errors
- A Java Model that provides API for navigating the Java element tree. The Java element tree defines a Java centric view of a project. It surfaces elements like package fragments, compilation units, binary classes, types, methods, fields.
- A Java Document Model providing API for manipulating a structured Java source document.
- Code assist and code select support.
- An indexed based search infrastructure that is used for searching, code assist, type hierarchy computation, and refactoring. The Java search engine can accurately find precise matches either in sources or binaries.
- Evaluation support in either a scrapbook page or a debugger context.
- Source code formatter

Our classes investigated have 997 classes that represents the plug-in ‘org.eclipse.jdt.core’.

## Eclipse PDE UI

Eclipse PDE UI provides a comprehensive set of tools to create, develop, test and debug Eclipse plug-ins, fragments, features, update sites and RCP products. It also provides comprehensive OSGi tooling, which makes it an ideal environment for component programming, not just Eclipse plug-in development.

The components including:

- Form-Based Manifest Editors - multi-page editors that centrally manage all manifest files of a plug-in or feature.
- RCP Tools - wizards and a form-based editor that allow you to define, brand, test and export products to multiple platforms.
- New Project Creation Wizards - create a new plug-in, fragment, feature, feature patch and update sites.
- Import Wizards - import plug-ins and features from the file system.
- Export Wizards - wizards that build, package and export plug-ins, fragments and products with a single click.
- Launchers - test and debug Eclipse applications and OSGi bundles.
- Views - PDE provides views that help plug-in developers inspect different aspects of their development environment.
- Miscellaneous Tools - wizards to externalize and clean up manifest files.
- Conversion Tools - wizard to convert a plain Java project or plain JARs into a plug-in project.
- Integration with JDT - plug-in manifest files participate in Java search and refactoring.
- User Assistance Tools - Editors and tools for developing user help and other UA documents.
- Declarative Services Tools - Editors and validation for OSGi declarative services.

The Dataset contains 1497 classes derived from 'org::eclipse::pde'.

## **Equinox Framework**

Eclipse Equinox is an implementation of the OSGi R6 core framework specification, a set of bundles that implement various optional OSGi services and other infrastructure for running OSGi-based systems.

More generally, the goal of the Equinox project is to be a first class OSGi community and to promote the vision of Eclipse as a landscape of bundles. As part of this, it is responsible for developing and delivering the OSGi framework implementation used for all of Eclipse

The principal components are:

- Implementation of all aspects of the OSGi specification (including the EEG, MEG and VEG work)
- Investigation and research related to future versions of OSGi specifications and related runtime issues
- Development of non-standard infrastructure deemed to be essential to the running and management of OSGi-based systems
- Implementation of key framework services and extensions needed for running Eclipse (e.g., the Eclipse Adaptor, Extension registry) and deemed generally useful to people using OSGi

Equinox, like every projects Eclipse, ships with all the major releases. The various other bundles developed here may ship independently and on different schedule

Our dataset is composed by 324 classes in different levels ('org::eclipse::osgi:internal' or 'org::eclipse::osgi::framework').

## Lucene

The Apache Lucene™ project develops open-source search software, including:

- Lucene Core, our flagship sub-project, provides Java-based indexing and search technology, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities.
- Solr a high performance search server built using Lucene Core, with XML/HTTP and JSON/Python/Ruby APIs, hit highlighting, faceted search, caching, replication, and a web admin interface.
- PyLucene is a Python port of the Core project.

It is composed by 691 classes derived from ‘org::apache::lucene’.

## Mylyn

Mylyn is the task and application lifecycle management (ALM) framework for Eclipse.

It provides:

- The revolutionary task-focused interface (Realigns the IDE around tasks so that you see only the code that's relevant )
- A task management tool for developers (Averaging 1 million downloads/month, Mylyn is the most popular IDE tool for ALM)
- A broad ecosystem of Agile and ALM integrations (Dozens of extensions integrate Mylyn with ALM and developer collaboration tools)

Mylyn's task-focused interface reduces information overload and makes multitasking easy. Mylyn makes tasks a first class part of the IDE, integrates rich and offline editing for ALM tools, and monitors your programming activity to create a "task context" that focuses your workspace and automatically links all relevant artifacts to the task-at-hand.

It contains 2862 classes derived from ‘org::eclipse::mylyn::tasks’ and ‘org::eclipse::mylyn::internal’.

## 5.1. DETAILS

Each software is described with 6 metrics into different formats:

Change Metrics into the csv file calls 'change-metrics.csv'.

Previous Defects into the csv file calls 'bug-metrics.csv'.

Source Code Metrics into the file 'single-version-ck-oo.csv'

Entropy of Change into the file 'complexity-code-change.csv'

Churn of source code metrics into the archive churn which contains every type of churn (churn, weight-churn, lin-churn, exp-churn and log-churn).


Entropy of source code metrics into the archive entropy with the different types (ent, weighted-ent, lin-ent, exp-ent and log-ent).

The metrics based on Source code metrics are calculated using the file biweekly-ck-value and bi-weekly-oo collecting each measures every 2 week.

## 6. DESCRIPTION WORK

Our workspace is MODELER.

Modeler is a software developed by IBM, which allow creating models Data-Mining-likes. Therefore, it is possible to personalize each algorithm in every phases of the prediction (Data understanding, data preparation...). The software is drags&drops-likes and there are many options.

In this approach, we want to define the best model to predict bugs using not only the linear regression, like in the last works, but using each possible models that we can apply. In fact, before to apply specific algorithm we have used a ‘Numeric Automatic’ tools.  Just to have the idea about the principal models, which we will use, based on the Linear Correlation and  $R^2$ . Using this tool, we have selected two principal models: Neural Networks and Decision Tree (based with C&RT and CHAID).

The output related with the linear colleration and relative error. First, we can note that the output are better than the linear regression. This approach has a disanvantages which is the not exacly prediction, because these tool use an automatic selection of the options. Therefore, it is necessary a study more detailed of our models.

We have apply the classical metrics and in the majority of cases, we have noted that the best model is the decision tree (CHAID algorithm).

The values of correlation are between 0.6 and 0.9 and the relative error is between 0.2 and 0.7. This is useful to be able to create a specific model for each software system.



## 6.1. ANALYSIS WITH PRINCIPAL METRICS

In this part of the job, we will work with neural network and decision tree in details. Starting with Eclipse and continuing with the others, it was tried every possible metrics and selected the best for each set of metric.

In the majority of cases, we have selected the same metrics used in the combined approach in the last work. The result are very different and more exhaustive.

Now, we use the metrics that have been studied in the last paper only with the linear regression, using neural network and decision tree

### CHANGE METRICS

NEURAL NETWORK										
	R <sup>2</sup>					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
	0.406	0.2	0.634	0.038	0.46	0.647	0.458	<b>0.811</b>	0.227	0.69
	0.167	0.045	0.494	0.028	0.242	0.43	0.236	0.726	0.205	0.515
	0.382	0.126	0.529	0.109	0.505	0.682	0.368	0.748	0.349	0.721
	0.392	0.129	0.630	0.066	0.509	0.636	0.373	<b>0.809</b>	0.281	0.724
MOSER										
NFIX										
NR										
NFIX + NR										

### DECISION TREE

	R <sup>2</sup>					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
MOSER	0.765	0.481	0.707	0.367	0.582	<b>0.878</b>	0.698	<b>0.852</b>	0.613	0.771
NFIX	0.112	0.048	0.335	0.004	0.078	0.361	0.243	0.615	0.133	0.325
NR	0.279	0.063	0.389	0.06	0.114	0.542	0.272	0.655	0.271	0.367
NFIX + NR	0.295	0.108	0.424	0.061	0.158	0.556	0.344	0.679	0.272	0.428

**Table 6.1 Change metrics.**

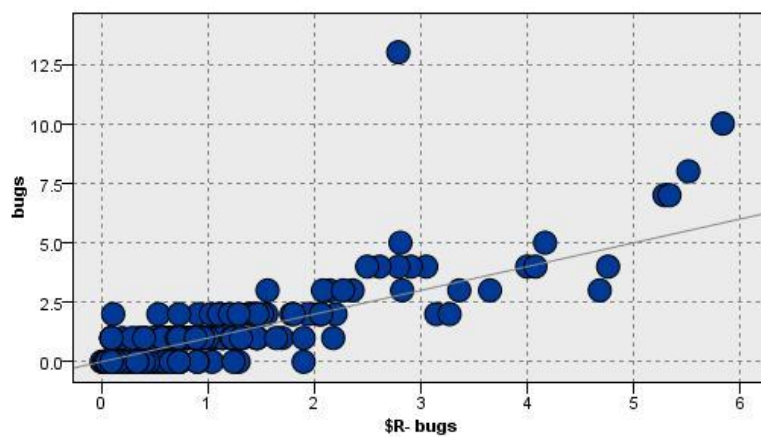
In the Table 6.1 we can see the results of our study in terms of R<sup>2</sup> and Spearman correlation for the first type of metric, Change Metrics. Like in the linear regression this metric not have any very relevant features. Only with Equinox (the same thing happened with linear regression), we obtain a good level of correlation especially with neural network.

The worst software system is PDE this is due for the different type of classes that it has. In fact, also in the last study PDE had a lower level prediction with this metric.

An interesting thing is about Eclipse: using our decision tree the predictive power with MOSER is not bad also in terms of explanative power that is the  $R^2$  that is equal to 0.765.

About Change Metrics type, the best use that we can do is for Equinox software system because the results with more techniques present stable results.

These plots rapresent the relation between bugs and predictive bugs with decision tree and linear regression.



**Figura 6.1 Linear regression Equinox**

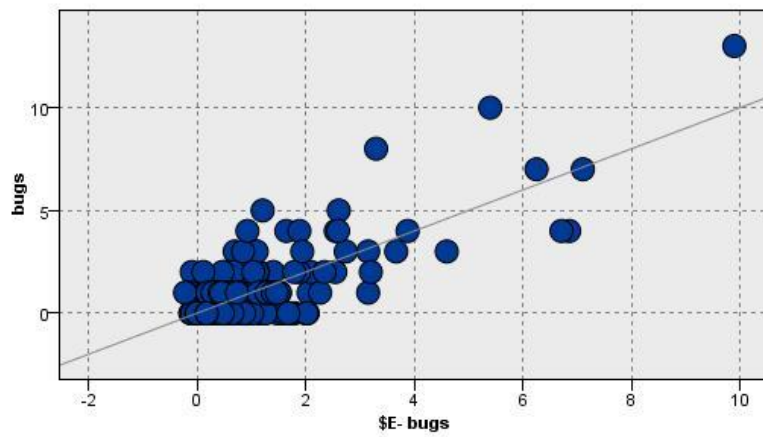


Figura 6.2 Decision TreeEquinox

## PREVIOUS DEFECTS

### NEURAL NETOWRK

	R <sup>2</sup>					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
BF	0.491	0.135	0.442	0.153	0.527	<u>0.708</u>	0.381	0.691	0.406	<u>0.736</u>
BUG-CAT	0.398	0.632	0	0.029	0	0.641	0.798	0.045	0.207	0

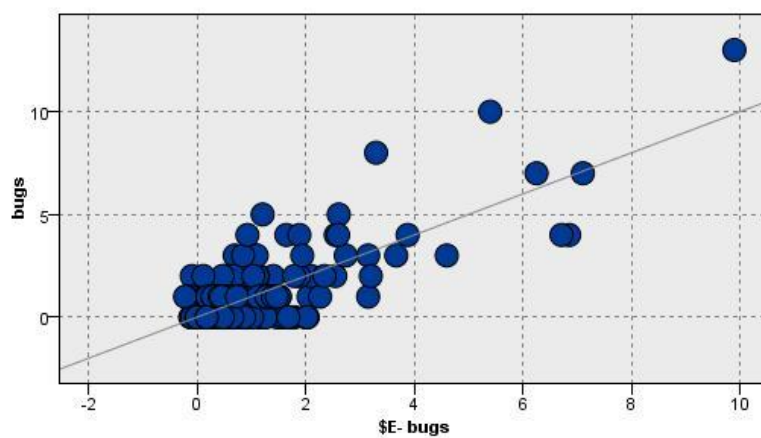
### DECISION TREE

	R <sup>2</sup>					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
BF	0.448	0.178	0.498	0.130	0.167	0.678	0.432	<u>0.728</u>	0.377	0.438
BUG-CAT	0.373	0.552	0	0.191	0	0.621	0.747	0.068	0.45	0

Table 6.2 Previous Defects.

The previous defects metric is one of the less generic metric. The failure of generality cause this behavior. The use of this metric whit the neural network and decision tree are more or less the same, in particular the bad performance for Equinox and Lucene with the BugCat metric.

In the last work, this metric was good with Eclipse, PDE and Lucene but now also with Equinox using the BF, how we can see in the figure.



**Figura 6.3 Decision Tree Equinox BF**

The low validation of this works is because the metric in the majority of the cases are unrelated.

Previous defect prove the level of correlation and Explanative power with the same performance in the cases of the Data Mining algorithm.

## SOURCE CODE METRICS

### NEURAL NETOWRK

	R^2					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
<b>CK+OO</b>	0.473	0.192	0.520	0.046	0.084	0.696	0.449	0.742	0.244	0.334
<b>CK</b>	0.410	0.120	0.502	0.034	0.227	0.65	0.36	0.731	0.219	0.501
<b>OO</b>	0.530	0.165	0.435	0.019	0.113	0.735	0.687	0.687	0.182	0.374
<b>LOC</b>	0.224	0.053	0.330	0.009	0.104	0.49	0.253	0.661	0.152	0.362

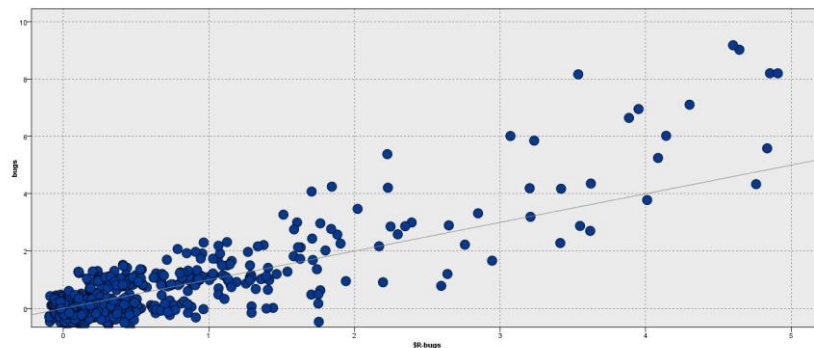
### DECISION TREE

	R^2					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
<b>CK+OO</b>	0.781	0.485	0.757	0.347	0.657	0.886	0.701	0.879	0.597	0.817
<b>CK</b>	0.540	0.234	0.577	0.175	0.398	0.741	0.493	0.778	0.432	0.646
<b>OO</b>	0.793	0.460	0.737	0.332	0.699	0.893	0.683	0.868	0.584	0.842
<b>LOC</b>	0.139	0.036	0.302	0.012	0.20	0.396	0.217	0.59	0.16	0.222

Table 6.3 Source Code Metrics Defects.

In the table 6.3 there are calculated the predictive and explanative power for the metric SOURCE CODE METRIC. A strange phenomenon happens for the Neural Network. We do not have any significant model that is better than the linear regression.

Instead, using the Decision Tree we have obtained very good results. The Eclipse case is more reliable like in the linear regression, using only OO metric we can obtain almost 0.9 of correlation .



**Figura 6.4 Decision Tree Eclipse OO**

The worst case of this metric is for Mylyn but an effect we have taken also in the last work.

Equinox and Lucene produce useful results that we do not have without these skills (Neural Network and Decision Tree).

*The use of Source Code Metric recommended with Eclipse using Linear Regression and Decision Tree but in particular for Equinox and Lucene.*

## ENTROPY OF CHANGE

### NEURAL NETOWRK

	R^2					SPEARMAN				
	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene
HCM	0.475	0.045	0.535	0.063	0.373	0.697	0.237	<u>0.752</u>	0.276	0.626
WHCM	0.431	0.034	0.348	0.158	0.302	0.665	0.213	0.625	0.412	0.568
EDHCM	0.390	0.094	0.264	0.295	0.415	0.634	0.323	0.558	0.552	0.658
LDHCM	0.414	0.093	0.504	0.229	0.442	0.653	0.321	<u>0.732</u>	0.489	0.663
LGDHCM	0.443	0	0.537	0.143	0.222	0.674	0.034	<u>0.753</u>	0.394	0.496



# DECISION TREE

	R <sup>2</sup>					SPEARMAN				
	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene
HCM	0.319	0.020	0.361	0.06	0.137	0.578	0.175	0.634	0.27	0.405
WHCM	0.295	0.048	0.323	0.081	0.121	0.556	0.243	0.606	0.307	0.384
EDHCM	0.241	0.058	0.260	0.087	0.112	0.507	0.261	0.555	0.316	0.373
LDHCM	0.261	0.067	0.339	0.085	0.106	0.526	0.278	0.618	0.321	0.364
LGDHCM	0.258	0.010	0.364	0.074	0.123	0.524	0.146	0.637	0.295	0.386

**Table 6.4 Entropy of Change**

The table 6.4 shall collect the findings of our analysis with the Entropy's metric.

The system Mylyn is not suitable in this case, in fact the value of R<sup>2</sup> is everywhere less 0.1 with ant type of Data Mining techniques (also Linear Regression).

Overall, this metric used for Eclipse and Equinox but without any successful results. This happened because the entropy used for calculated the prediction is too general and the differences between various types of classes are discard.

Entropy of change metric is a metric too overall, also in the Linear Regression case.  
The utility is for the improvement that can be apply with some revision.

## CHURN OF SOURCE CODE METRICS

### NEURAL NETOWRK

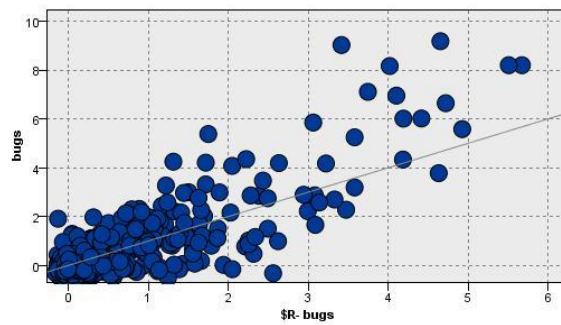
	R^2					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
CHU	0.411	0.174	0.639	0.213	0.306	0.651	0.428	<b>0.814</b>	0.473	0.572
WCHU	0.474	0.128	0.440	0.140	0.584	0.696	0.371	0.69	0.39	<b>0.773</b>
LDCHU	0.470	0.160	0.366	0.291	0.301	0.694	0.411	0.638	0.549	0.568
EDCHU	0.498	0.167	0.367	0.290	0.363	<b>0.713</b>	0.42	0.369	0.547	0.618
LGDCHU	0.488	0.1	0.446	0.324	0.404	<b>0.706</b>	0.332	0.695	0.578	0.65

### DECISION TREE

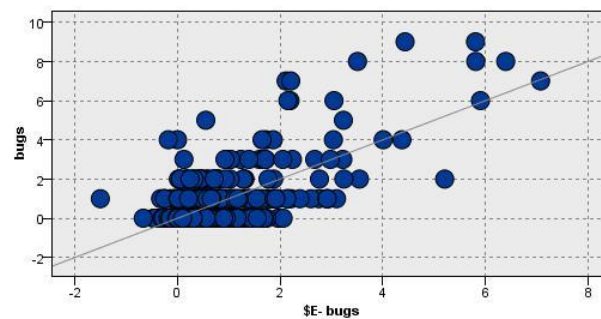
	R <sup>2</sup>					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
CHU	0.749	0.374	0.708	0.313	0.641	0.869	0.617	0.853	0.568	0.807
WCHU	0.742	0.338	0.695	0.315	0.643	0.865	0.588	0.845	0.57	0.809
LDCHU	0.422	0.159	0.444	0.174	0.428	0.659	0.41	0.693	0.43	0.668
EDCHU	0.472	0.152	0.414	0.166	0.482	0.695	0.403	0.672	0.421	0.706
LGDCHU	0.490	0.198	0.442	0.261	0.332	0.707	0.455	0.692	0.521	0.594

Table 6.5 Churn of Source Code Metrics

In the table 6.5 we can see the behavior of the metric ‘Churn of source code metrics’ apply with the Decision Tree and Neural Network. The best correlation is with Eclipse using Churn of Weight metrics in addition with the decision tree we can see the different about the use of linear regression or decision tree.



**Figura 6.6 Decision Tree W Churn Decision Tree**



**Figura 6.7 Linear regression Eclipse OO**

The worst cases is another times with Mylyn which represent a short level of accuracy. We can see many improvements with Lucene always with the classic churn or weigh churn. This behavior was very different in the last paper.

A reduction of the performance is for the explanative power ( $R^2$ ) for PDE's systems that with both of methods we have obtained low results.

The metric Churn of source code metric works very well with Decision Tree algorithm. The level of correlation is high and sometimes we can get levels between 0.8 and 0.9.

Finally, we propose three good models for Eclipse Equinox and Lucene:

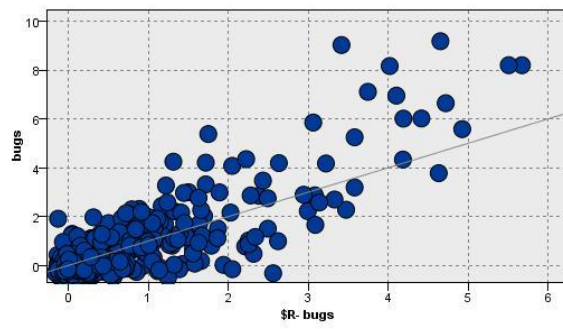


Figura 6.8 Decision Tree Eclipse

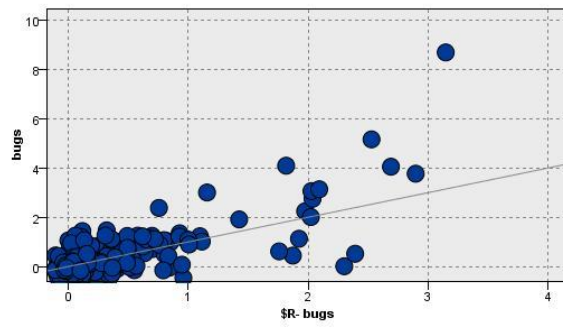


Figura 6.9 Decision Tree Lucene

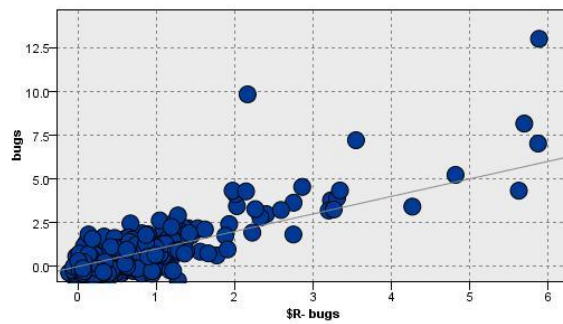


Figura 6.10 Decision Tree Equinox

## Entropy of source code metrics

### NEURAL NETOWRK

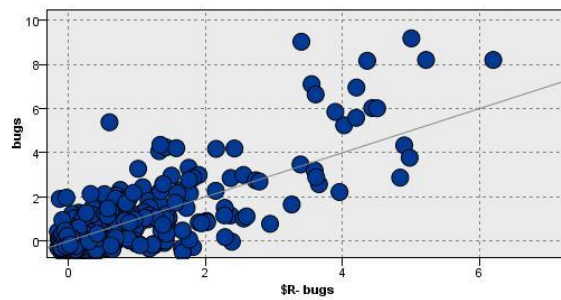
	R^2					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
HH	0.471	0.136	0.440	0.042	0.401	0.695	0.382	0.69	0.235	0.648
HWH	0.478	0.130	0.481	0.272	0.382	0.699	0.374	0.717	0.531	0.633
LDHH	0.495	0.119	0.430	0.312	0.319	0.711	0.359	0.683	0.567	0.583
EDHH	0.443	0.166	0.300	0.259	0.227	0.675	0.419	0.578	0.519	0.5
LGDHH	0.451	0.090	0.570	0.150	0.198	0.68	0.316	0.734	0.403	0.472

## DECISION TREE

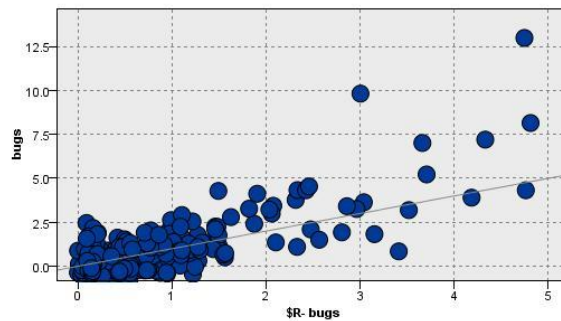
	R^2					SPEARMAN				
	Eclipse	Mylyn	Equinox	Pde	Lucene	Eclipse	Mylyn	Equinox	Pde	Lucene
HH	0.690	0.306	0.647	0.286	0.516	<b>0.834</b>	0.561	<b>0.818</b>	0.544	<u>0.728</u>
HWH	0.536	0.146	0.484	0.249	0.421	<u>0.739</u>	0.394	<u>0.719</u>	0.51	0.663
LDHH	0.462	0.167	0.489	0.171	0.272	0.688	0.42	<u>0.722</u>	0.428	0.542
EDHH	0.442	0.161	0.457	0.167	0.322	0.674	0.413	<u>0.702</u>	0.423	0.585
LGDHH	0.407	0.197	0.486	0.253	0.305	0.648	0.454	<u>0.721</u>	0.513	0.571

Table 6.6 Entropy of Source Code Metrics

The last table (Table 7.6) have the results of the last metric. As we can see in the table, the best way to use this metric is with Eclipse and Equinox (Fig 11 and Fig 12).



**Figura 6.11 Decision Tree Eclipse**



**Figura 6.12 Decision Tree Equinox**

An improvement of this metric is with the Decision Tree, infact with Mylyn and PDE (the software systems with the lowest level of confidence) there are good results. In the last work both of them had level of correlation and  $R^2$  very bad, but now the decision tree developed can be used to predict bugs.

Only the Neural network that we have created in the cases of Eclipse and Lucene present the same performance respect to the linear regressin, especially in term of explanative power.

*This type of metric can be used with the Decision Tree withput any improvements (Weight version, linear version etc.). The level of correlation are right for a good prediction.*



## 6.2. COMMENTS

The simulation work using the decision tree and neural network respect to the last metric could be complete. Overall, the results studied with the linear regression are satisfied. In particular, the last best metrics discovered like WCHU and LDHH still having a good performance for the majority of the software system, with some improvements, just to see the Lucene software.

In more detail, the metric with improvements is the Source code metrics (C). Here we have obtained level of correlation close to 0.9 for three softwares (Eclipse, Equinox and Lucene) but also the other have obtained good results. In this way during the software development for these systems, can be increased the quality, the accuracy and the reliability.

On the other hands, the worst metric with these techniques is the Previous Defects (B). Overall we can see which are present some null values, incurred for the bad use of this metric with the decision tree cases. Nevertheless, in any cases, using neural network, we can develop a level of confidence between 0.5 and 0.7 for the majority of the systems.

The eclipse software shows a good level of prediction (0.9) using the metric OO in the Source code Metric (C). This one is the best results that we have obtained correlated to an  $R^2$  of 0.79 which means a very good proximity to the studied bugs.

Mylyn have a bad behavior with the Entropy of Change (D), like in the linear regression. The best model is in the Previous defects (B) with the bugs fixed metrics, using a neural network. Obviously, the results is not good like Eclipse but is better respect also a linear regression cases.

Equinox has the same performance of Eclipse. In fact, we have founded the best model (with 0.9 of correlation) with the Source code metrics (C) in the CC + OO

cases, always with a decision tree. This aspect was respect also with the linear regression. Because the two software have equal classes, (like cited previously).

The PDE system had some problems to try a good model. The best model that we are able to obtain is using the neural network in the Churn of source code metric (D) using the logarithmic case (with a correlation of 0.58 and  $R^2$  of 0.7). This result in terms of quality is the same with Mylyn.

The final software, Lucene has encountered high improvements. In fact using the metric Source code metric (C) OO with the decision tree is possible to develop a model with high performance. The result here obtained is not comparable with the linear regression model. In the last work, the level of performance was about 0.65 but now is close to 0.9.

## 7. ANALYSIS WITH A METRICS COMBINATION

Now, in the second part of the experiments, we have developed a mixed approach with the best metrics using also some adjusted operation for each software system.

We have designed five models. To be able to develop these we have started with the selection of the best metric respect to correlation level. In some cases, we have changed the domain. There are five possible operations:

- Inverse:  $\frac{1}{x}$  ;
- Logarithm  $\log_{10}(x)$  ;
- Logarithm  $\log_n(x)$
- Exponential  $e^x$  ;
- Square root  $\sqrt[2]{x}$  ;

The inverse and logarithm operation have some problems with the null values. To solve this it was added 1 to an each cell to the metric matrix but the performance decreased respect to not use the mathematical operation.

Only one transformation appears good, the Square root. Using this particular operation, we can see after in the analysis, the robustness of the model is more reliable.

Sperimentally we can see that just changing the domain the interpretations of the metrics and the relative model appears more accurated.

In the work [1] was studied a ranking approach that have an impact on the domain knowledge. The features are connected with the lessical gap to connect natural language terms in the bug report. They have created a linear combination of features able to resolve bugs. In particular had been used similarity measures to dig up at best method to solve bugs.

Another problem is related with the too vagueness caused by metrics and because they are not focused into the domain. In the paper [li] has been studied correlation between specific warnings and bugs. It shows experimentally that there is a relation between domain specific warnings and defects.

The generalization of the categories is too expensive and laborious. The approach proposed by [lii] is able to automatize the categorization of software applications with an API using the extraction of stakeholders. Based with statistical operations they found the categories, which are able to provide more information. As already mentioned the API packages provides better performance than API classes.

The model developed are synthetized here:

- Model 1: This model have every fields of the metric B - Previous Defects. In this case using every types of bugs is possible to predict very well problems. The inconvenience is that this method is not much generic so cannot be used in a general-purpose scenario.
- Model 2: In this case we have used the square root transformation such that to improve the performance. Therefore with the square root the data are more balanced and the prediction appears better.
- Model 3: The Model 3 provides for using CC + OO metric of the Source Code Metric with weights.
- Model 4: The Model 4, instead, provides of using CC + OO metric of Entropy of source code metric with the linear transformation.
- Model 5: The last model provides a union of the best metric. Here we have joined the metrics C and F (linear Entropy)
- Model 6: This model has connected the metrics E (weight churn) and F (linear entropy).

The table 8 collect the results of the study with the best models in terms of explanative and predictive power.

# DECISION TREE

	R^2					SPEARMAN				
	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene
<b>Model 1</b>	0.657	0.653	0.502	0.276	0.167	<b>0.815</b>	<b>0.811</b>	0.731	0.535	0.438
<b>Model 2</b>	0.750	0.485	0.757	0.347	0.657	<b>0.869</b>	0.701	<b>0.879</b>	0.597	<b>0.817</b>
<b>Model 3</b>	0.704	0.360	0.693	0.289	0.605	<b>0.843</b>	0.606	<b>0.844</b>	0.547	0.787
<b>Model 4</b>	0.766	0.374	0.651	0.314	0.575	<b>0.879</b>	0.617	<b>0.821</b>	0.569	0.767
<b>Model 5</b>	<b>0.823</b>	0.654	<b>0.814</b>	0.386	0.710	<b>0.909</b>	<b>0.811</b>	<b>0.909</b>	0.628	<b>0.848</b>
<b>Model 6</b>	<b>0.805</b>	0.532	0.676	0.341	0.661	<b>0.899</b>	0.733	<b>0.835</b>	0.592	<b>0.819</b>
NEURAL NETOWRK										

	R^2					SPEARMAN				
	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene	ECLIPSE	Mylyn	EQUINOX	PDE	Lucene
<b>Model 1</b>	0.706	0.786	0.673	0.566	0.557	<b>0.844</b>	<b>0.888</b>	<b>0.833</b>	0.756	0.755
<b>Model 2</b>	0.462	0.160	0.413	0.131	0.111	0.688	0.412	0.671	0.378	0.371
<b>Model 3</b>	0.538	0.147	0.572	0.1	0.338	0.74	0.396	0.755	0.288	0.598
<b>Model 4</b>	0.510	0.139	0.490	0.250	0.235	0.722	0.386	0.723	0.511	0.508
<b>Model 5</b>	0.623	0.220	0.443	0.245	0.272	0.794	0.479	0.692	0.506	0.542
<b>Model 6</b>	0.344	0.166	0.490	0.160	0.272	0.598	0.419	0.723	0.415	0.543

Table 7.1 BestModel results

## 7.1. COMMENTS

The results of these models are satisfactory. In fact for each models except to PDE we are able to develop a model with a level of confidence higher than 0.8.

Using these models, a good quality of the bug-prediction is obtain. The best model are the models 5 for Equinox and Eclipse using the Decision Tree, here it was obtained a correlation of 0.9 and the predicted model is able to intercept almost the totally bugs. A good use of these metrics is for Mylyn, in fact before is not possible to develop any enough model, but now with the Model 1 mylyn has a prediction capability very high.

The disadvantages of these models were in terms of CPU-time to arrive to the solution, because the nodes are several. This problem is not serious because the time increase of some tenth so it is irrelevant.

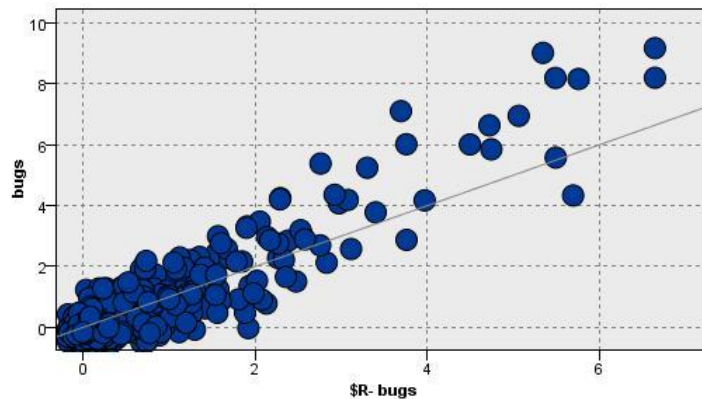
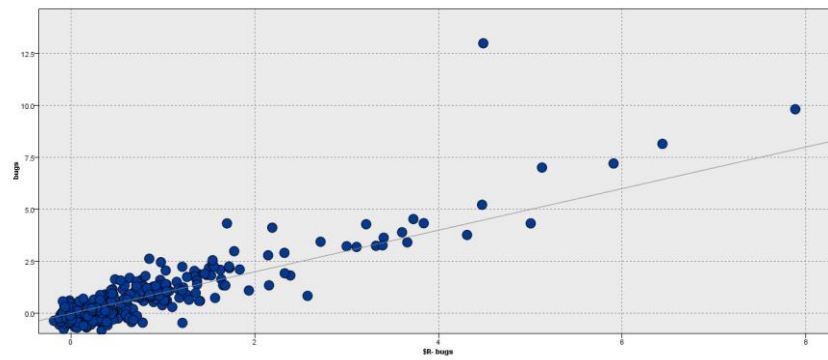
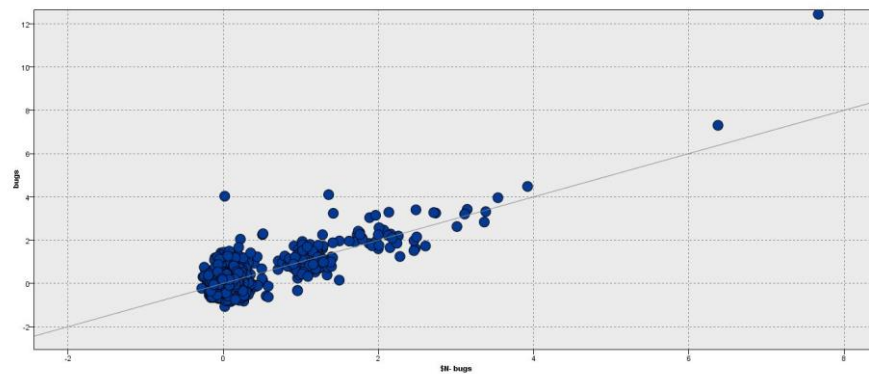


Figura 7.1 Decision Tree Eclipse Model 5



**Figura 7.2 Decision Tree Equinox Model 5**



**Figura 7.3 Neural Network Mylyn Model 1**

There are shows some graphs to represent the reliability of our some best model (in terms of Spearman correlation and  $R^2$ ).

## 8. METRIC ADJUNCT

Based with the last metrics used in the last projects, new metrics are developed. This metric generates a good level of correlation using the Data-mining techniques like Neural Network and Decision Tree, but it has a bad result with the linear regression.

The based form of the metric is this:

$$M(i) = \sum_{j=0}^c \begin{cases} 0, & \text{if } d(i,j) = 0 \\ 1 + d(i,j) + \log_2(d(i,j)), & \text{otherwise} \end{cases} \quad (8.1)$$

Where:

- $d(i,j)$  = Difference between two measures in two periods of time adjacent in the same row corresponding at the same class for each metric
- $C$  = numbers of elements captured in a period of time for each classes

Like in the Entropy of Change - D (or Churn of Source metrics – E or Entropy of Source code metric – F) we have started from the metrics, Source Code Metrics for each classes, collect for each types each two weeks (1). Using these metrics, we have calculated the metrics for each classes and metrics-types, collected in a matrix (2).



Below is proposed an example using CBO and LOC metrics with the easiest case of this type of metric (M).

- 1:

Metric						
CBO						
Classes	A	5	5	5	5	5
	B	-1	-1	7	8	-1
	C	4	4	7	7	9
	D	5	4	-1	-1	8

Metric						
LCO						
Classes	A	8	8	8	9	10
	B	-1	-1	5	6	6
	C	-1	-1	-1	-1	-1
	D	10	10	10	5	-1

- 2:

Metric			
Classes		CBO	LOC
	A	4	6
	B	5	5
	C	11.58	4
	D	5	11.32

This approach is apply for each metric developed by Chidamber and Kemerer (Source Code Metrics). The values obtained used for input for each prediction-model (Desion Tree and Neural Network).

To develop this metric we have started with the metric E (Churn of source code metric) and we have changed same factor in order to increase the performance of the model. We have used the same concept of delta between two samples of measure. Here it has added the standard distance plus the logarithm of the distance, each of which multiply for a specific factor ( $\alpha$ ,  $\beta$  and  $\gamma$ ) which were chosen opportunely with experimental measures (used for the advanced version of this hybrid metric).

Like in the Churn version, adding a supplement with the four classic version (Linear, Exp, Log and Weight). Experimentally these metrics works very well in particular cases for example for the Eclipse the best is Linear Version with the Decision Tree, which return a 0.81 level of correlation.

The basic idea was used the rules developed in the study from Jiarpakdee [<sup>liii</sup>]. The study respects these steps:

Hypothesis – Design appropriate metric – Define a model – Develop analytical model (Neural Network or Decision Tree) – Testing ( $R^2$ )

To validate the metric I have to be careful about the introduction of inconsistency was apply the variable clustering and the variance inflation factor techniques like cited in the paper.

The constants calculated in simulation. I have used these values:

- $\alpha = 0.9$
- $\beta = 0.01$
- $\gamma = 0.1$

A strange effect was about the Linear Regression that do not have any relevant change if we modify the constants. In fact this metric have sense if is used in correlation with the neural network or decision tree.

There are developed different versions of the metrics:

$$\begin{array}{ll} \text{Weighted} & \\ \text{metric:} & MW(i) = \sum_{j=0}^c \begin{cases} 0 & d(i,j) = 0 \\ 1 + \alpha * d(i,j) + \beta \log_2(d(i,j)) & \text{otherwise} \end{cases} \end{array} \quad (8.2)$$

$$\begin{array}{ll} \text{Exponential} & \\ \text{metric:} & ME(i) = \sum_{j=0}^c \begin{cases} 0 & d(i,j) = 0 \\ \frac{1 + \alpha * d(i,j) + \beta \log_2(d(i,j))}{e^{\gamma(C+1-j)}} & \text{otherwise} \end{cases} \end{array} \quad (8.3)$$

$$\begin{array}{ll} \text{Linear} & \\ \text{metric:} & ML(i) = \sum_{j=0}^c \begin{cases} 0 & d(i,j) = 0 \\ \frac{1 + \alpha * d(i,j) + \beta \log_2(d(i,j))}{\gamma * (C + 1 - j)} & \text{otherwise} \end{cases} \end{array} \quad (8.4)$$

$$\begin{array}{ll} \text{Logarithmic} & \\ \text{metric:} & MLG(i) = \sum_{j=0}^c \begin{cases} 0 & d(i,j) = 0 \\ \gamma * \log_2(C + 1 - j) & \text{otherwise} \end{cases} \end{array} \quad (8.5)$$

## 8.1. EXPERIMENTS WITH A NEW METRIC

These are metrics developed using Matlab. Here we can see the code for the first metric (the others change only some factor):

```
...
[data, text] = xlsread(File Name with the samples);
s=0;
for i=1: length(data)
    a=data(i,1);
    for j=2 : length(data(1,:))
        b=data(i,j);
        if (a == b) || ((a == -1) && (b == -1))
            s=s;
        else
            if (a== -1) && (b>0)
                c=1+b+alfa*log2(b);
                s=s+c;
            else
                c=1+abs(b-a)+beta*log2(abs(b-a));
                s=s+c;
            end
        end
        a=data(i,j);
    end
    M(i+1,2)=s;
    s=0;
end
```

**Figura 8.1** Code MyMetric.

The operation is easy. For each element in the row, correspondent to a class, we calculate the difference between two adjacent elements, which correspond samples taken in 2 weeks. If the difference is zero we do not calculate nothing, otherwise we calculate the difference and the logarithm of the difference. We apply this approach for each samples until we do not have any values. Therefore, we sum every difference for every class of the software system.

Here, we can see the effect for each type of metric with our dataset, using the two methods, Decision Tree and Neural Network.

	DECISION TREE		NEURAL NETWORK	
	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.473	0.716	0.593	0.774
<b>MW</b>	0.154	0.792	0.571	0.76
<b>ME</b>	0.127	0.762	0.425	0.658
<b>ML</b>	0.461	0.699	0.161	0.433
<b>MLG</b>	0.202	0.801	0.528	0.732

Table 8.1 Eclipse

	DECISION TREE		NEURAL NETWORK	
	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.665	0.818	0.797	0.894
<b>MW</b>	0.660	0.814	0.793	0.892
<b>ME</b>	0.627	0.795	0.639	0.802
<b>ML</b>	0.374	0.617	0.382	0.136
<b>MLG</b>	0.665	0.818	0.791	0.891

Table 8.2 Mylyn

	DECISION TREE		NEURAL NETWORK	
	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.429	0.683	0.604	0.794
<b>MW</b>	0.171	0.676	0.615	0.78
<b>ME</b>	0.124	0.425	0	0.163
<b>ML</b>	0.463	0.706	0.568	0.772
<b>MLG</b>	0.354	0.629	0.471	0.711

Table 8.3 Equinox

	DECISION TREE		NEURAL NETWORK	
	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.358	0.606	0.614	0.787
<b>MW</b>	0	0.584	0.703	<b>0.838</b>
<b>ME</b>	0.299	0.555	0.273	0.532
<b>ML</b>	0.202	0.462	0.048	0.248
<b>MLG</b>	0	0.573	0.583	0.763

Table 8.4 PDE

	DECISION TREE		NEURAL NETWORK	
	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b><i>M</i></b>	0.495	0.715	0.348	0.607
<b><i>MW</i></b>	0.375	0.657	0.290	0.604
<b><i>ME</i></b>	0.235	0.508	0	0.172
<b><i>ML</i></b>	0.298	0.638	0.293	0.583
<b><i>MLG</i></b>	0.445	0.68	0.342	0.602

**Table 8.5 LUCENE**

Below is develop the metric using the Linear Regression for each software system:

	LINEAR REGRESSION	
	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.415	0.651
<b>MW</b>	0.414	0.65
<b>ME</b>	0.420	0.654
<b>ML</b>	0.414	0.65
<b>MLG</b>	0.414	0.65

**Table 8.6 ECLIPSE**

### LINEAR REGRESSION

	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.763	<b>0.875</b>
<b>MW</b>	0.763	<b>0.875</b>
<b>ME</b>	0.765	<b>0.876</b>
<b>ML</b>	0.02	0.113
<b>MLG</b>	0.763	<b>0.875</b>

**Table 8.7 MYLYN**

### LINEAR REGRESSION

	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.099	0.397
<b>MW</b>	0.189	0.414
<b>ME</b>	0	0.181
<b>ML</b>	0	0.252
<b>MLG</b>	0.022	0.293

**Table 8.8 EQUINOX**



### LINEAR REGRESSION

	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.392	0.633
<b>MW</b>	0.401	0.633
<b>ME</b>	0.408	0.645
<b>ML</b>	0	0.065
<b>MLG</b>	0.400	0.632

**Table 8.9 PDE**

### LINEAR REGRESSION

	<b>R<sup>2</sup></b>	<b>Spearman Correlation</b>
<b>M</b>	0.02	0.181
<b>MW</b>	0	0.168
<b>ME</b>	0	0.173
<b>ML</b>	0.02	0.196
<b>MLG</b>	0.01	0.203

**Table 8.10 LUCENE**

## 8.2. COMMENTS NEURAL NETWORK & DECISION TREE

Overall, this metric has a good behavior for each software system in terms of Correlation and  $R^2$ .

The best software system is Mylyn, which has very good values for both the models using every types of metrics. In fact here we have the best model, which is the neural Network using the classic metric without any improvement (with a correlation of 0.894). Under we can see the matching between bugs and predictive bugs

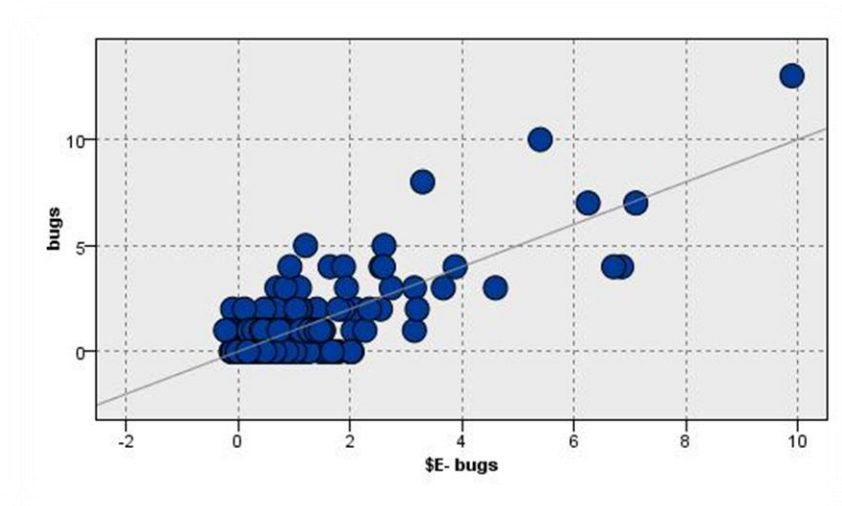


Figura 8.2 Neural Network Mylyn

Our experimental metric works good with the Decision Tree, just see the Eclipse case using the Logarithm type, with a correlation of 0.801 but with a bad  $R^2$ .

The principal reason of our negative values using the linear regression are:

This is a metric developed especially for the neural network and decision tree without consider other data-mining techniques. Another reason is for the dataset, that the database how we can understand from the other study is not able to obtain level of correlation like in these cases.

The worst case is the Linear for the Mylyn system, with a correlation 0.136 and a  $R^2$  equal to 0.382 for the neural network. In general, the metric, which returns results worst than the others metric, is the Exponential case. This behavior does not surprise us because that is what happens for each software system in the last work.

Lucene is the only software system that does not have any very interesting results, in some cases also nulls because the Dataset, have less rows ( $< 1000$ ) respect to the other dataset that have many values.

### **8.3. COMMENTS LINEAR REGRESSION**

As regards the linear regression the performance are insufficient. The reason is correlated for the greater addressing between the new metric and the neural network (or decision tree) algorithm. Therefore, this find best correlation and exploration with the first techniques.

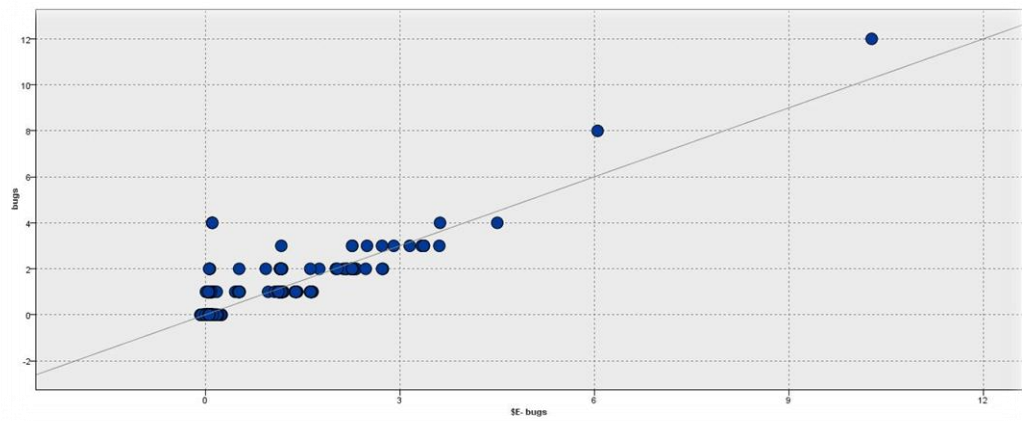
However, there are some cases where this metric is good for the linear regression.

Great success there are for the software Mylyn. For each kind of metrics the performance are great, only for the linear case the model is not capable to predict exactly the bugs.

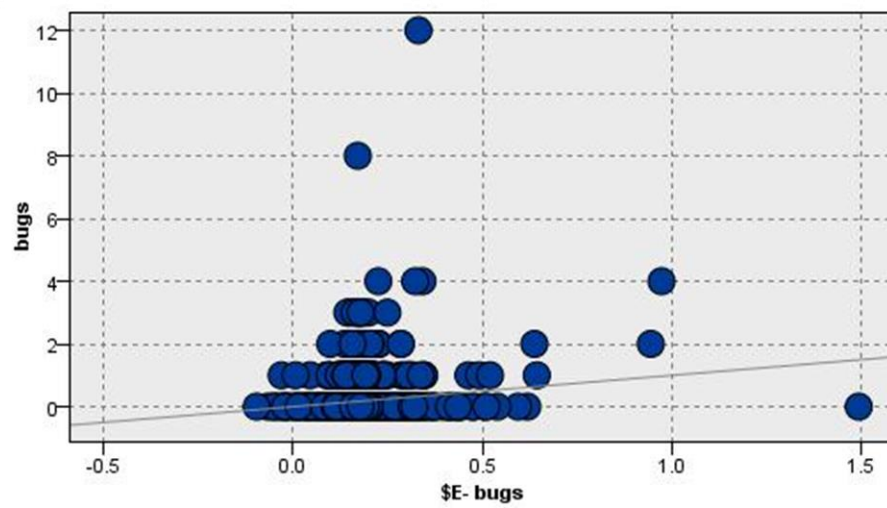
Eclipse contains a mean level of correlation about 0.4 without null value. In the other software there are present also null value. Meaning the absolutely capability of no-prediction, for both the exactly number of bugs for a class (spearman correlation) and for a global vision of the system ( $R^2$ ).

Especially for Lucene the model is not capable in the bugs-prediction.

Below we can see two graphs to show the best and the worst cases with the use of this hybrid metric:



**Figura 8.3 Linear Regression Mylyn metric M**



**Figura 8.4 Linear Regression Mylyn metric ML**

## 9. BINARY CLASSIFICATION

Last analysis involve the binary classification to predict bugs using neural network and decision tree.

To apply this model we need to change the interpretation of bugs in the Dataset. Means the field ‘bugs’ now can assume only two values (bugs or not-bugs). We have used this strategy:

$$\begin{cases} n^{\circ} \text{ bug} \geq 1 & \text{bug} = 1 \\ n^{\circ} \text{ bug} = 0 & \text{bug} = 0 \end{cases}$$

Using this strategy is possible to create a classification model able to predict the present (or absence) of a bugs for each class.

Starting from the metrics (described previously) apply with the best model (section 8) we have create some classificatory with neural network and decision tree.

### 9.1. EXPLICATION BINARY CLASSIFICATION

Binary classification is a task of classification that divided the dataset into two groups.

To evaluate the classification there are many factors (different with the last one). In this case, we cannot use spearman or Pearson but there are specific measure of performance.

In the literature there are numerous metrics (recall, specificity, F-measure) and each oh them have a specific role and should be use in a specific scenario.

Each of metrics start with the **confusion matrix**:

		DEFECTS ARE OBSERVED	
		True	False
MODEL PREDICTS DEFECTS	Positive	True positive (TP)	False Positive (FP)
	Negative	False negative (FN)	True negative (TN)

A confusion matrix is a table typically used to evaluate the performance of a supervised algorithm. Each rows represents the predicted value and each column represent the value of the real class. Therefore, in this way we can count the numbers of true (and false) positive and the numbers of True (and False) negative respect to the membership class.

From this table we can study the accuracy of the model with different metrics (Recall, Precision, F-measure and Accuracy). Using the ROC (Receiver operating characteristic), we can evaluate with a area the difference between different classification model.

- **Recall:**

The recall relates the number of true positives (predicted and observed as defect-prone) to the number of classes that actually had defects.

A goog value is close to one, means that every class that had defects observed was predict to have defects.

$$recall = \frac{TP}{(TP + FN)} \quad (9.1)$$

- **Precision:**

The precision relates the number of true positives (predicted and observed as defectprone) to the number of classes predicted as defect-prone.

A good value is close to one, means that every class that was predicted to have defects actually had defects

$$precision = \frac{TP}{(TP + FP)} \quad (9.2)$$

- **Accuracy:** The accuracy relates the number of correct classifications (true positives and true negatives) to the total number of classes. A value of accuracy equal to one means that the model classified perfectly without any single mistake.

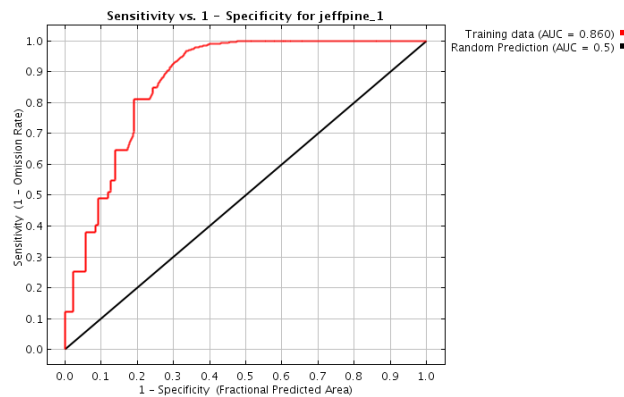
$$accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \quad (9.3)$$

- **F-measure:** The F-measure is a harmonic mean of recall and precision. We preferred this type of mean because the harmonic mean is commonly appropriate when averaging rates or frequencies. F-measure allows differential weighting of Recall and Precision but commonly they are given equal weight.

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \quad (9.4)$$

## ROC

The ROC (Receiver operating characteristic) curve is a plot that shows the performance of a binary classifier. The ROC is built by plotting the True-positive-rate (Y) and False-positive-rate (X).



**Figure 9.1 ROC curve example**

The area under the curve (AUC) is the overall measure of the classifier

Larger is the AUC better classifier performance are. For example the best classification is with TP-rate = 1 and FP-rate = 0, with a AUC = 1 (we have the total area under the curve). Instead, the worst classifier is with TP-rate = 0 and

FP-rate = 1, in this case the classifier is a opposite-classifier.

Considering a classifier with TP= 0.9 and FP=0.2 this means:

- 90 % of the observation are classifier well;
- 20 % of the observation are wrongly assigned to the positive class



## 9.2. SIMULATION

We apply the classification model with the last metrics that have studied (Best metrics in the chapter 8). We can see that every model have very good level of precision, recall, accuracy and F-measure. To choose the best model we have use also the ROC. So these metrics used to create a binary classifier for the bug prediction.

### Eclipse

#### Model 1

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	770	21	Predicted condition	No Bugs	698	93
	Bugs	104	102		Bugs	54	152
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.93	0.88	0.92	0.87	0.88	0.92	0.90	0.85

Table 9.1 Eclipse Model 1

## Model 2

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	760	26	Predicted condition	No Bugs	686	98
	Bugs	109	102		Bugs	61	152
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.96	0.87	0.91	0.86	0.88	0.91	0.90	0.85

Table 9.2 Eclipse Model 2

## Model 3

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	758	33	Predicted condition	No Bugs	738	53
	Bugs	99	107		Bugs	20	186
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.96	0.88	0.90	0.83	0.93	0.97	0.95	0.92

Table 9.3 Eclipse Model 3

### Model 4

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	754	37	Predicted condition	No Bugs	729	62
	Bugs	129	77		Bugs	23	183
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.95	0.85	0.90	0.83	0.97	0.90	0.93	0.89

Table 9.4 Eclipse Model 4

### Model 5

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	769	23	Predicted condition	No Bugs	756	26
	Bugs	84	122		Bugs	12	1940.
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.97	0.90	0.93	0.89	0.96	0.98	0.97	0.97

Table 9.5 Eclipse Model 5

## Model 6

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	735	56	Predicted condition	No Bugs	750	41
	Bugs	75	131		Bugs	21	185
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.93	0.90	0.92	0.86	0.95	0.97	0.96	0.94

Table 9.6 Eclipse Model 6

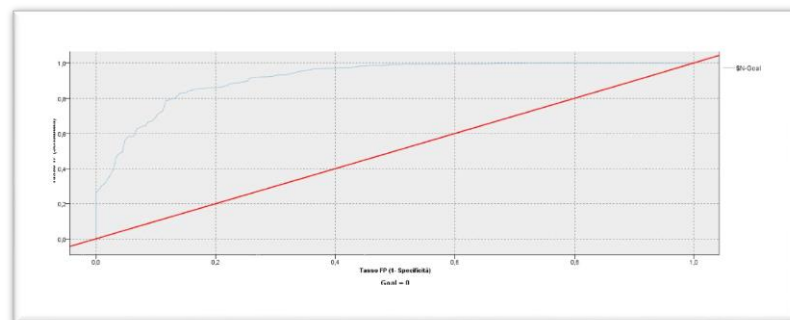


Figura 9.2 ROC model 5 Eclipse (Decision Tree)

# Mylyn

## Model 1

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1614	3	<i>Predicted condition</i>	No Bugs	1497	120
	Bugs	84	161		Bugs	52	193
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.95	0.97	0.95	0.92	0.96	0.95	0.90

Table 9.7 Mylyn Model 1

## Model 2

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1576	41	<i>Predicted condition</i>	No Bugs	1446	171
	Bugs	181	64		Bugs	16	229
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.97	0.89	0.93	0.88	0.89	0.98	0.93	0.89

Table 9.8 Mylyn Model 2

### Model 3

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	1589	28	Predicted condition	No Bugs	1126	491
	Bugs	213	32		Bugs	23	222
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.88	0.93	0.87	0.69	0.97	0.81	0.72

Table 9.9 Mylyn Model 3

### Model 4

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1608	9	<i>Predicted condition</i>	No Bugs	1157	460
	Bugs	232	13		Bugs	26	219
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.87	0.93	0.87	0.71	0.97	0.82	0.73

Table 9.10 Mylyn Model 4

## Model 5

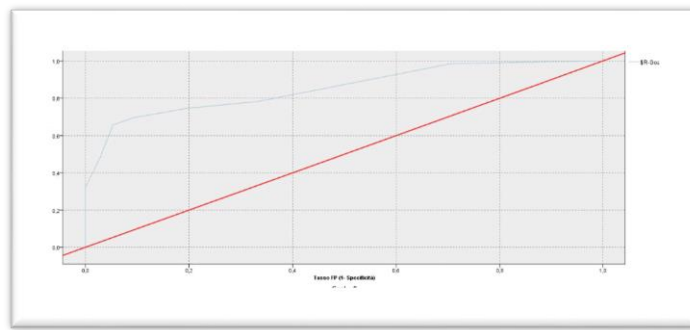
NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	1589	28	Predicted condition	No Bugs	1520	97
	Bugs	204	41		Bugs	18	227
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.88	0.93	0.87	0.94	0.98	0.96	0.93

Table 9.11 Mylyn Model 5

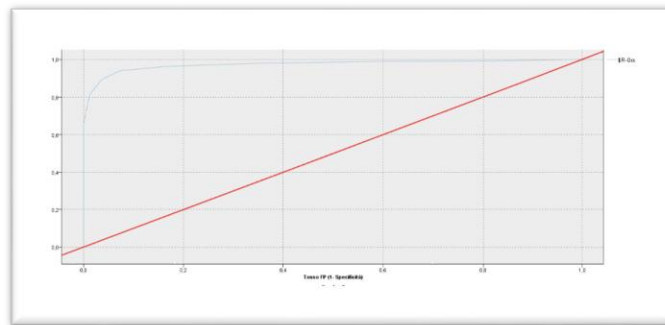
## Model 6

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1583	34	<i>Predicted condition</i>	No Bugs	1209	408
	Bugs	206	39		Bugs	21	224
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.97	0.88	0.93	0.87	0.74	0.98	0.84	0.76

Table 9.12 Mylyn Model 6



**Figura 9.3 ROC model 3 Mylyn (Decision Tree)**



**Figura 9.4 ROC model 5 Mylyn (Decision Tree)**

In these two graphs, we can see a couple of model of Mylyn. In particular, we can see the worst (Model 3 Decision Tree) and the best (Model 5 decision tree).

Like suggested from the metrics like F-measure the model 5 have a F-measure and accuracy higher than 0.9, so the AUC is almost perfect.



## Equinox

### Model 1

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	165	30	Predicted condition	No Bugs	159	56
	Bugs	40	89		Bugs	56	93
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.84	0.80	0.82	0.78	0.81	0.81	0.81	0.77

Table 9.13 Equinox Model 1

### Model 2

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	152	43	<i>Predicted condition</i>	No Bugs	178	17
	Bugs	46	83		Bugs	10	119
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.77	0.76	0.77	0.72	0.91	0.94	0.92	0.91

Table 9.14 Equinox Model 2

### Model 3

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	167	28	Predicted condition	No Bugs	172	23
	Bugs	57	72		Bugs	16	113
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.85	0.74	0.79	0.73	0.88	0.91	0.89	0.87

Table 9.15 Equinox Model 3

### Model 4

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	171	24	Predicted condition	No Bugs	174	21
	Bugs	61	68		Bugs	17	112
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.87	0.73	0.80	0.73	0.89	0.91	0.90	0.88

Table 9.16 Equinox Model 4

## Model 5

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	157	38	Predicted condition	No Bugs	176	19
	Bugs	49	80		Bugs	4	125
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.80	0.76	0.78	0.73	0.90	0.97	0.93	0.82

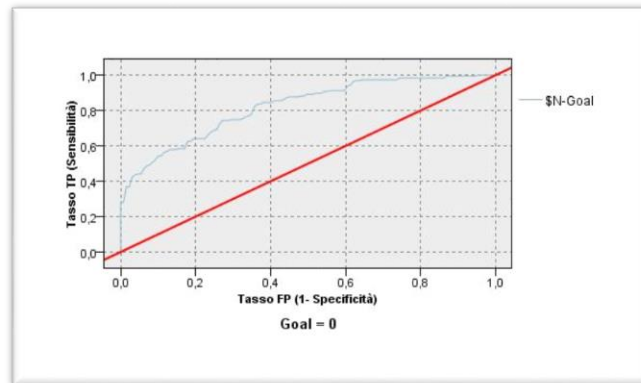
Table 9.17 Equinox Model 5

## Model 6

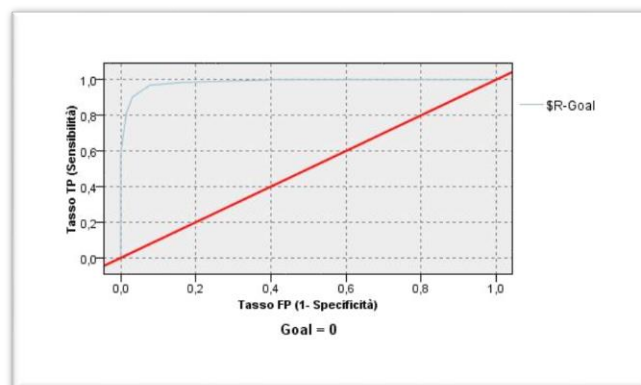
NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	165	30	<i>Predicted condition</i>	No Bugs	176	19
	Bugs	50	75		Bugs	4	125
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.80	0.76	0.7	0.73	0.90	0.97	0.93	0.93

Table 9.18 Equinox Model 6

In this case the performance are limited, especially for the model 2 with neural network. The best is for the model 5 (Decision Tree). This depends for the bugs distribution that is inefficient for thid type of prediction.



**Figura 9.5 ROC model 2 Equinos (Neural Network)**



**Figura 9.6 ROC model 5 Equinos (Decision Tree)**

## PDE

### Model 1

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	1283	5	Predicted condition	No Bugs	1142	146
	Bugs	141	68		Bugs	86	123
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.90	0.94	0.90	0.88	0.93	0.90	0.84

**Table 9.19 PDE Model 1**

### Model 2

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1614	3	<i>Predicted condition</i>	No Bugs	1288	0
	Bugs	84	161		Bugs	209	0
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.95	0.97	0.95	1	0.86	0.92	0.86

**Table 9.20 PDE Model 2**

### Model 3

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	1273	15	Predicted condition	No Bugs	1158	130
	Bugs	167	42		Bugs	46	166
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.88	0.93	0.87	0.89	0.96	0.92	0.88

Table 9.21 PDE Model 3

### Model 4

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1264	24	<i>Predicted condition</i>	No Bugs	1214	74
	Bugs	172	37		Bugs	49	160
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.88	0.92	0.86	0.94	0.96	0.95	0.91

Table 9.22 PDE Model 4

## Model 5

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	1264	24	Predicted condition	No Bugs	1242	46
	Bugs	128	81		Bugs	19	190
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.90	0.94	0.89	0.96	0.98	0.97	0.95

Table 9.23 PDE Model 5

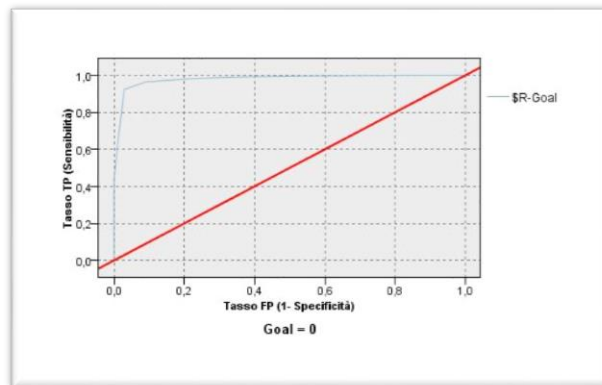
## Model 6

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	1274	14	<i>Predicted condition</i>	No Bugs	1197	91
	Bugs	186	23		Bugs	46	163
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.87	0.92	0.86	0.92	0.96	0.94	0.90

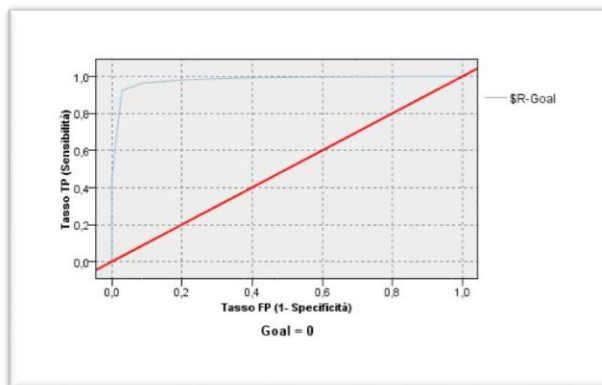
Table 9.24 PDE Model 6

PDE is more consistent with the classification prediction. We have different cases where the accuracy is very high (Model 2 with neural network and model 5 with Decision Tree).

The number and distribution of bugs is balanced and this be able to obtain performance higher.



**Figura 9.7 ROC model 2 PDE (Neural Network)**



**Figura 9.8 ROC model 5 PDE (Decision Tree)**



## Lucene

### Model 1

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	626	1	<i>Predicted condition</i>	No Bugs	495	132
	Bugs	51	13		Bugs	12	46
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.92	0.96	0.92	0.78	0.97	0.87	0.78

Table 9.25 lucene Model 1

### Model 2

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	626	1	<i>Predicted condition</i>	No Bugs	615	12
	Bugs	59	5		Bugs	4	60
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.91	0.95	0.91	0.98	0.99	0.98	0.97

Table 9.26 lucene Model 2

### Model 3

NEURAL NETWORK		Real condition		DECISION TREE		Real condition	
		No Bugs	Bugs			No Bugs	Bugs
Predicted condition	No Bugs	620	7	Predicted condition	No Bugs	610	17
	Bugs	45	19		Bugs	12	52
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.93	0.96	0.92	0.97	0.98	0.97	0.95

Table 9.27 lucene Model 3

### Model 4

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	616	11	<i>Predicted condition</i>	No Bugs	601	260
	Bugs	51	13		Bugs	11	53
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.92	0.95	0.91	0.69	0.98	0.81	0.70

Table 9.28 lucene Model 4

## Model 5

NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	620	7	<i>Predicted condition</i>	No Bugs	621	6
	Bugs	53	11		Bugs	6	58
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.98	0.92	0.95	0.91	0.99	0.99	0.99	0.98

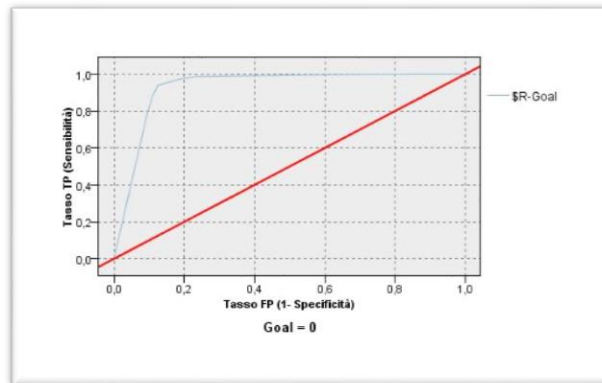
Table 9.29 lucene Model 5

## Model 6

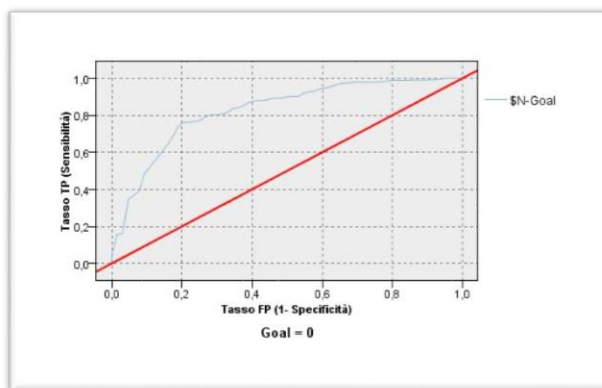
NEURAL NETWORK		<i>Real condition</i>		DECISION TREE		<i>Real condition</i>	
		No Bugs	Bugs			No Bugs	Bugs
<i>Predicted condition</i>	No Bugs	626	1	<i>Predicted condition</i>	No Bugs	614	10
	Bugs	57	7		Bugs	9	55
precision	recall	f-measure	accuracy	precision	recall	f-measure	accuracy
0.99	0.91	0.95	0.91	0.98	0.98	0.98	0.97

Table 9.30 lucene Model 6

The last software system have performance almosto perfectaly for each model and each methods. We can see that for the models 5 and 6 False positive is less than 10 fo the prediction is good, like we can see in some ROC.



**Figura 9.9 ROC model 3 Lucene (Decision Tree)**



**Figura 9.10 ROC model 4 Lucene (Neural Network)**

## 10. CONCLUSIONS

The neural network and decision tree developed in this work make it possible a prediction of bugs with level of correlation and prediction about 0.8 and 0.9 with the Software Sets proposed.

Very relevant model are for example:

- Mooser with Eclipse and Equinox
- BF with Equinox
- WCHU with Lucene

Relevant concept discovered are about the correlation measure (Spearman and  $R^2$ ). Spearman correlation interpreted like the exactly knowledge that joins what we want that it happen with the possible output of the system.

$R^2$  just said if a events can happen or not, in our cases if a classes X using a metric M can give a bug or not.

We apply the classification model with the last metrics that have studied (. We can see that every model have very good level of precision, recall, accuracy and F-measure. To choose the best model we have use also the ROC. So these metrics can be use to create a binary classifier for the bug prediction.

These results are possible only using neural network and decision tree (not with linear regression that performe in some past works).

Using mixed metrics we have create six models with a different interpretations.

Model 1 apply the formula of our metrics. Model 5 and 6 with linear and logarithmic improvements.

Model 1 developed with a mixed of metrics perform prediction for Mylyn with a Spearman correlation of 0.81

The Model 5 and 6 return values almost like a perfect prediction.

In this work we have covered the bug prediction problem based with Spearman correlation and  $R^2$  including the explanation of the use of these factors. Explaining thoroughly the principal's metrics to understand the software about bugs and developing different model with different goals for each software system.

Thanks to this work, we can use different approaches to predict bugs in order to discover where the bug is present and how to avoid errors.

An interesting point is about the use of different coefficients to understand better the prediction. We have discovered that if we are interesting on the exactly numbers of bugs we should use Spearman correlation. Instead, if we want to know the presence of bugs we need to use the  $R^2$  coefficient. This rule is true with any types of model that we have described.

An alternative approach could be with the mobile software to highlight the bugs' problems relative with graphics and the different platforms.

The Model 5 and 6 return values almost like a perfect prediction.

In this work we have covered the bug prediction problem based with Spearman correlation and  $R^2$  including the explanation of the use of these factors. Explaining thoroughly the principal's metrics to understand the software about bugs and developing different model with different goals for each software system.

Thanks to this work, we can use different approaches to predict bugs in order to discover where the bug is present and how to avoid errors.

An interesting point is about the use of different coefficients to understand better the prediction. We have discovered that if we are interesting on the exactly numbers of bugs we should use Spearman correlation. Instead, if we want to know the presence of bugs we need to use the  $R^2$  coefficient. This rule is true with any types of model that we have described.

An alternative approach could be with the mobile software to highlight the bugs' problems relative with graphics and the different platforms.

## INDEX FIGURES

- Figura 3.1. Multi-Layer Neural Network.
- Figura 3.2 Slow convergence.
- Figura 3.4 Overfitting
- Figura 3.5 Decision Tree example
- Figura 3.7 Spearman 1
- Figura 3.8 Spearman 2
- Figura 3.9 Spearman 3
- Figura 9.1 ROC curve example

# INDEX GRAPHS

- Figura 6.1 Linear regression Equinox
- Figura 6.2 Decision Tree Equinox
- Figura 6.3 Decision Tree Equinox BF
- Figura 6.4 Decision Tree Eclipse OO
- Table 6.4 Entropy of Change
- Table 6.5 Churn of Source Code Metrics
- Figura 6.6 Decision Tree W Churn Decision Tree
- Figura 6.7 Linear regression Eclipse OO
- Figura 6.8 Decision Tree Eclipse
- Figura 6.9 Decision Tree Lucene
- Figura 6.10 Decision Tree Equinox
- Figura 6.11 Decision Tree Eclipse
- Figura 6.12 Decision Tree Equinox
- Figura 7.1 Decision Tree Eclipse Model 5
- Figura 7.2 Decision Tree Equinox Model 5
- Figura 7.3 Neural Network Mylyn Model 1
- Figura 8.1 Code MyMetric.
- Figura 8.2 Neural Network Mylyn
- Figura 8.3 Linear Regression Mylyn metric M
- Figura 8.4 Linear Regression Mylyn metric ML
- Figura 9.2 ROC model 5 Eclipse (Decision Tree)
- Figura 9.3 ROC model 3 Mylyn (Decision Tree)
- Figura 9.4 ROC model 5 Mylyn (Decision Tree)
- Figura 9.5 ROC model 2 Equinox (Neural Network)
- Figura 9.6 ROC model 5 Equinox (Decision Tree)
- Figura 9.7 ROC model 2 PDE (Neural Network)
- Figura 9.8 ROC model 5 PDE (Decision Tree)
- Figura 9.9 ROC model 3 Lucene (Decision Tree)
- Figura 9.10 ROC model 4 Lucene (Neural Network)



# INDEX TABLES

- Table 6.1 Change metrics.
- Table 6.2 Previous Defects.
- Table 6.3 Source Code Metrics Defects.
- Table 6.4 Entropy of Change
- Table 6.5 Churn of Source Code Metrics
- Table 6.6 Entropy of Source Code Metrics
- Table 7.1 BestModel results
- Table 8.1 Eclipse
- Table 8.2 Mylyn
- Table 8.3 Equinox
- Table 8.4 PDE
- Table 8.5 LUCENE
- Table 8.6 ECLIPSE
- Table 8.7 MYLYN
- Table 8.8 EQUINOX
- Table 8.9 PDE
- Table 8.10 LUCENE
- Table 9.1 Eclipse Model 1
- Table 9.2 Eclipse Model 2
- Table 9.3 Eclipse Model 3
- Table 9.4 Eclipse Model
- Table 9.5 Eclipse Model 5
- Table 9.6 Eclipse Model
- Table 9.7 Mylyn Model 1
- Table 9.8 Mylyn Model 2
- Table 9.9 Mylyn Model 1
- Table 9.10 Mylyn Model 4
- Table 9.11 Mylyn Model 5
- Table 9.12 Mylyn Model 6
- Table 9.13 Equinox Model 1
- Table 9.14 Equinox Model 2
- Table 9.15 Equinox Model 3
- Table 9.16 Equinox Model 4
- Table 9.17 Equinox Model 5
- Table 9.18 Equinox Model 6
- Table 9.19 PDE Model 1
- Table 9.20 PDE Model 2
- Table 9.21 PDE Model 3
- Table 9.22 PDE Model 4
- Table 9.23 PDE Model 5
- Table 9.24 PDE Model 6
- Table 9.25 lucene Model 1
- Table 9.26 lucene Model 2
- Table 9.27 lucene Model 3
- Table 9.28 lucene Model 4
- Table 9.29 lucene Model 5
- Table 9.30 lucene Model 6

# BIBLIOGRAPHY

---

- <sup>i</sup> “Barriers to Effective Use of Knowledge Management Systems in Software Engineering” Kevin C. Desouza
- <sup>ii</sup> “Local and Global Recency Weighting Approach to Bug Prediction” Hemant Joshi, Chuanlei Zhang, S. Ramaswamy and Coskun Bayrak
- <sup>iii</sup> “Revisiting Common Bug Prediction Findings Using Effort-Aware Models” Yasutaka Kame, Shinsuke Matsumoto and Akito Monden
- <sup>iv</sup> “Effort-aware defect prediction models” T. Mende and R. Koschke
- <sup>v</sup> “Classifying Software Changes: Clean or Buggy?” Sunghun Kim, E. James Whitehead Jr and Yi Zhang
- <sup>vi</sup> “Method-Level Bug Prediction” Emanuel Giger, Marco D’Ambros and Martin Pinzger
- <sup>vii</sup> “Graph-Based Analysis and Prediction for Software Evolution”, Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu and Michalis Faloutsos
- <sup>viii</sup> “Bug-fix Time Prediction Models: Can We Do Better?” Pamela Bhattacharya and Iulian Neamtiu
- <sup>ix</sup> “Bug Prediction Based on Fine-Grained Module Histories” Hideaki Hata, Osamu Mizuno, and Tohru Kikuno
- <sup>x</sup> “Tracking Concept Drift of Software Projects Using Defect Prediction Quality” Jayalath Ekanayake, Jonas Tappolet, Harald C. Gall and Abraham Bernstein
- <sup>xi</sup> “Does Bug Prediction Support Human Developers? Findings From a Google Case Study” Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou and E. James Whitehead Jr.
- <sup>xii</sup> “Micro Interaction Metrics for Defect Prediction” Taek Lee, Jaechang Nam, DongGyun Han, Sunghun Kim and Hoh Peter
- <sup>xiii</sup> “Reducing Features to Improve Bug Prediction” Shivkumar Shivaji, E. James Whitehead, Ram Akella and Sunghun Kim
- <sup>xiv</sup> “An empirical study on software defect prediction with a simplified metric set” Peng He, Bing Li, Xiao Liu, Jun Chen and Yutao Ma
- <sup>xv</sup> “Choosing software metrics for defect prediction: an investigation on feature selection techniques” Kehan Gao, Taghi M. Khoshgoftar, Huanjing Wang and Naeem Seliya
- <sup>xvi</sup> “Mining Software Repositories with Topic Models” Stephen W. Thomas
- <sup>xvii</sup> “Anomaly-Based Bug Prediction, Isolation, and Validation: An Automated Approach for Software Debugging” Martin Dimitrov and Huiyang Zhou
- <sup>xviii</sup> “A Data Mining Model to Predict Software Bug Complexity Using Bug Estimation and Clustering” Naresh Kumar Nagwani and Ashok Bhansali
- <sup>xix</sup> “Towards Effective Bug Triage with Software Data Reduction Techniques” Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo and Xindong Wu
- <sup>xx</sup> “A Comparative Study of Supervised Learning Algorithms for Re-opened Bug Prediction” Xin Xial, David Lo, Xinyu Wang, Xiaohu Yang, Shanping Li and Jianling Sun
- <sup>xxi</sup> “An Extensive Comparison of Bug Prediction Approaches” Marco D’Ambros, Michele Lanza and Romain Robbes
- <sup>xxii</sup> “Intelligent Heart Disease Prediction System Using Data Mining Techniques” Sellappan Palaniappan and Rafiah Awang
- <sup>xxiii</sup> “Intelligent Heart Disease Prediction System Using Data Mining Techniques” Sellappan Palaniappan and Rafiah Awang
- <sup>xxiv</sup> “Intelligent Heart Disease Prediction System Using Data Mining Techniques” Sellappan Palaniappan and Rafiah Awang

- 
- <sup>xxv</sup> “Early Prediction of Heart Diseases Using Data Mining Techniques” Vikas Chaurasia and Saurabh Pal
- <sup>xxvi</sup> “Prediction of financial distress: An empirical study of listed Chinese companies using data mining” Ruibin Geng, Indrani Bose, Xi Chen
- <sup>xxvii</sup> [ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/AlgorithmsGuide.pdf](http://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/AlgorithmsGuide.pdf)
- <sup>xxviii</sup> “BP Neural Network-Based Effective Fault Localization” W. ERIC WONG\* and YU QI
- <sup>xxix</sup> “Comparing Fine-Grained Source Code Changes And Code Churn For Bug Prediction” Emanuel Giger, Martin Pinzger and Harald C. Gall
- <sup>xxx</sup> “Software defect prediction using cost-sensitive neural network” Ömer Faruk Arara and Kürs Ayan
- <sup>xxxi</sup> “Predicting Defect Priority Based on Neural Networks” Lian Yu1, Wei-Tek Tsai, Wei Zhao and Fang Wu
- <sup>xxxii</sup> “Predicting the priority of a reported bug using machine learning techniques and cross project validation” Meera Sharm, Punam Bedi, K.K. Chaturvedi and V. B. Singh
- <sup>xxxiii</sup> “Effective software fault localization using an BFN neural network” W. Eric Wong, Vidroha Debroy, Richard Golden, Xiaofeng Xu and Bhavani Thuraisingham
- <sup>xxxiv</sup> “Predicting the Fix Time of Bugs” Emanuel Giger, Martin Pinzger and Harald Gall
- <sup>xxxv</sup> “Using the Gini Coefficient for Bug Prediction in Eclipse” Emanuel Giger, Martin Pinzger and Harald Gall
- <sup>xxxvi</sup> “Predictive Data Mining Model for Software Bug Estimation Using Average Weighted Similarity” Naresh Kumar Nagwani and Dr. Shrish Verma
- <sup>xxxvii</sup> “Using Decision Trees to Predict the Certification Result of a Build” Ahmed E. Hassan and Ken Zhang
- <sup>xxxviii</sup> “Characterizing and Predicting Blocking Bugs in Open Source Projects” Harold Valdivia Garcia and Emad Shihab
- <sup>xxxix</sup> “Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects” Hongyu Zhang, Liang Gong, Steve Versteeg, Jue Wang, Zeqi Shen and Janine Radford
- <sup>xl</sup> “An Extensive Comparison of Bug Prediction Approaches” Marco D’Ambros, Michele Lanza and Romain Robbes
- <sup>xli</sup> “Predicting Defects for Eclipse” Thomas Zimmermann, Rahul Premraj and Andreas Zeller
- <sup>xlii</sup> “Improving Defect Prediction Using Temporal Features and Non Linear Models”, Abraham Bernstein, Jayalath Ekanayake and Martin Pinzger
- <sup>xliii</sup> “Comparative studies of metamodelling techniques under multiple modelling criteria” R. Jin, W. Chen and T.W. Simpson
- <sup>xliv</sup> “Comparative studies of metamodelling techniques under multiple modelling criteria” R. Jin, W. Chen and T.W. Simpson
- <sup>xlv</sup> “*Analysing the Concrete Compressive Strength using Pearson and Spearman*” Chandrasegar Thirumalai, Swapna Anupriya Chandhini and Vaishnavi M
- <sup>xlvi</sup> “A Metrics Suite for Object Oriented Design” Shyam R. Chidamber and Chris F. Kemerer
- <sup>xlvii</sup> “Predicting Faults Using the Complexity of Code Change” Ahmed E. Hassan
- <sup>xlviii</sup> “Predicting Faults Using the Complexity of Code Change” Ahmed E. Hassan
- <sup>xlix</sup> “An Extensive Comparison of Bug Prediction Approaches” Marco D’Ambros, Michele Lanza and Romain Robbes
- <sup>l</sup> “Learning to Rank Relevant Files for Bug Reports using Domain Knowledge” Xin Ye, Razvan Bunescu, and Chang Liu
- <sup>li</sup> “Domain specific warnings: Are they any better?” Andr’e Hora, Nicolas Anquetil, St’eephane Ducasse, Simon Allier
- <sup>lii</sup> “On using machine learning to automatically classify software applications into domain categories” Mario Linares, Vásquez, Collin McMillan, Denys Poshyvanyk and Mark Grechanik
- <sup>liii</sup> “The Impact of Correlated Metrics on Defect Models” Jirayus Jiarapakdee, Chakkrit Tantithamthavorn and Ahmed E. Hassan