International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 02 - Issue 12 || December 2017 || PP. 49-58

Prediction of Software Defects Using Zero R

Abstract: In the software field the quality and the reliability are the important factors which have to be greatly handled with the help of software defect prediction. During the period of development and the maintenance of the software detecting and rectifying the software defects is really more expensive. By designing prediction model which accurately determines the occurrence of defect in software greatly assist in efficient software testing, reducing the cost and considerably improvising testing process of software by focusing on fault prone modules. Machine Learning Classifiers have emerged as a way to predict the fault in the software system. This paper focuses on predicting the software defect contributed by NASA repository dataset. In this work three different classifiers are used namely naïve bayes, simple logistics and Zero R. The experimental result shows that the performance of predicting software is highly classified using Zero R than remaining two classifiers. Thus it helps in increasing the quality and reliability of software.

Keywords: Software, defect, quality, detect, fault, predict.

Introduction

With the help of software metrics and the software fault dataset which was collected from the earlier developed software or projects based on the prediction model is trained and developed for software defect detection. This well trained model can then be applied to unknown defect data of any software module. The performance of the defect prediction is greatly influenced by the attributes of the software metrics which increases the efficiency of the software considerably. As sovereign testing team, it is significant to map and administer the test implementation behavior in order to convene the tight limit for releasing the software to end-users. Since the aspire of test carrying out is to determine as many fault as possible, testing team is typically put into encumber to guarantee all defects are establish and set by the developers inside the system testing stage.

Extra number of days has to be added to the timeline to contain testing team in effecting their test with the trust that all defects have been originate and set. On the other hand, the stakeholders would also ask the difficult team on the forecasted defects in the software so that they could choose whether the software is reasonable and robust for release. This is owing to the environment that system testing is the last gate before the software is made visible to end-users, thus as the custodian of executing system testing, the autonomous testing team has to take liability to ensure software to be unrestricted is of high excellence.

Therefore, the ability to predict how many defects that can be found at the start of system testing shall be a good way to tackle this issue. This becomes the reason for conducting this study. Besides serving as a target on how many defects to capture in system testing, defect prediction can also become an early quality indicator for any software entering the testing phase. Testing team can use the predicted defects to plan, manage and control test execution activities. This could be in the form aligning the test execution time and number of test engineers assigned to particular testing project. Having defect prediction as part of the testing process allows testing team to strengthen their test Strategies by adding more exploratory testing and user experience testing to ensure known defects are not escaped and re-introduced to end-users.

Related Work

Software defect prediction is not a new thing in software engineering domain. To come out with the right defect prediction model various related studies and approaches have been conducted. Understanding what defect really means is important so that the term defect is not confused with error, mistake or failure. In the event the defect have taken place, when the software or system fails to perform its desired function [1]. Defect is also observed as the deviation from its specification [2] as well as any imperfection related to software itself and its related work product [3]. Consequently, defect can be referred as its work product and something that is not according to requirement for software. Since, the defects means it is the structure the prediction model for defects, it is used to know how defects are introduced as part of verification and validation (V&V) activities [3]. Defects predicting can be characterized in the proactive process of many types of defects that can be found in software's content, design and codes in producing high quality product [4]. To predict defect density Rayleigh model was also used for different phases of project life cycle [5]. In [6] product and project metrics collected from design review, code testing, code peer review as well as product release usage and defect validation can be constructed using the model to predict defects. Linear regression was applied to these metrics via product metrics only, project metrics only and both. As the result, both product and project metrics provided better correlation between defects and the predictors using linear regression. It demonstrated the feasibility of using regression analysis to build defect prediction model at the same time. To predict defects an approach was carried out using mathematical distributions that serve as quality prediction model [7].

In order to identify and predict the highest defects in the large software systems will prone to more defect is investigated was performed in it. The important factor for the prediction and its impact to the model quality is development information will be the result of the investigation, which focuses on three metrics: number of developers who modified the file during the prior release; the number of new developers who modified the file during the prior release; and the cumulative number of distinct developers who modified the file during all releases through the prior release [8].

We also study to investigate on how to defect fault-proneness in the source code of the open source Web and e-mail suite called Mozilla.To conduct the investigation it used object-oriented metrics proposed by Chidamber and Kemerer [9]. On the other hand, [10] to build defect prediction model was proposed several inputs to simulate the system test phase, in which those inputs could be considered as potential predictors. The defect prediction was based on simple Bayesian Network in a form of Defect Type Model (DTM) that predicts defects based on severity minor, major and minor was the another approach to defects prediction [11]. To come out with defect inflow prediction for large software projects either short-term defect inflow prediction or long-term defect inflow prediction [12] is used by Multivariate linear regression. [13] To predict defect density statistical approach in Six Sigma methodology is applied. In this case, Statistical method was used against the function point as the base metrics to predict defect density before releasing software to production. Defect prediction can also be observed from different perspective which is by predicting remaining total number of defects while the testing activities are still on-going [14], which is called as defect decay model. This model depends on on-going test execution data instead of historical data. [15] Case studies can be presented on building and assisting their organization to assess testing effectiveness and predict the quantity of post release defects and enables quantitative decision about production go-live readiness the defect prediction model was used.

Their model was mostly focused on predicting defects in receiving test or manufacture which involves estimate total possible defects based on defined thorough requirements, applying defect elimination efficiency and finally estimates the defects per phase as well as post discharge defects. It display a 1% defect removal efficiency improvement which equals to $20,000 for implementing this model, The defect prediction would be difficult However, if past data is not available. Sample-based defect prediction was proposed to overcome this difficulty by using a small sample of modules to construct cost-effective defect forecast models for large scale systems, in which Co Forest, a semi supervised learning method was applied [16]. For defect prediction testing resources portion could be optimized, [17] on predicting defects of cross-project when chronological data is not in place possibility study must be conducted.

The training data is very significant for machine learning based defect prediction provided that the data is carefully selected from the projects was demonstrated as results. Building of defect prediction system, it is necessary to couple with the technique to find its success. In [18] the authors proposed to compute the percent of faults establish in the recognized files as one of the ways to review the efficiency of the prediction Systems. In addition that, the model is said to be a good if it can help in the resource planning in order to maintain the software and insure based on the software system itself is insured [19]. However, it is firm to discover an recognized standard specific for defect prediction. An attempt was taken by given that an all-embracing contrast of well-known bug prediction approaches, jointly with narrative approaches using openly available dataset consisting of numerous software systems [20]. The findings showed that there is still a difficulty with observe to exterior soundness in defect prediction. It necessitate larger mutual data set towards having a noteworthy target of defect prediction

Methodology

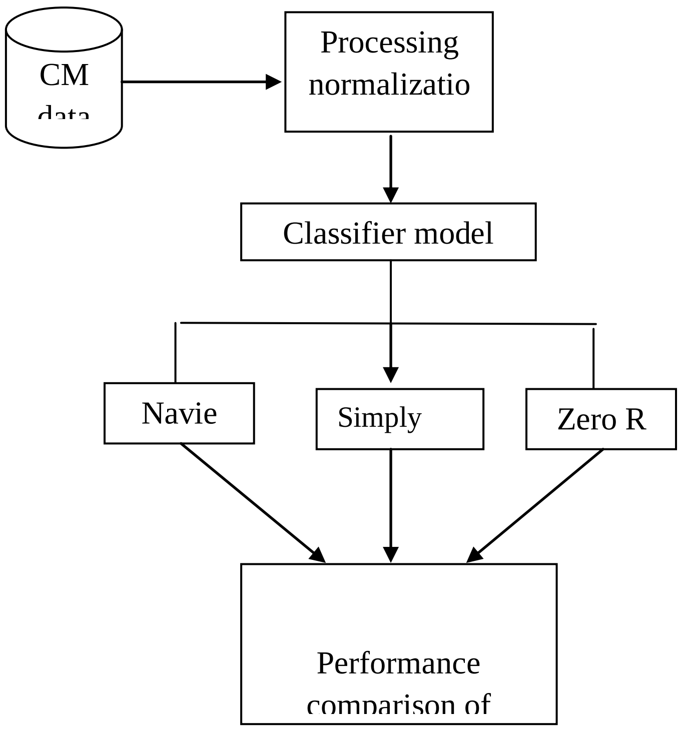


Figure 1 Architecture of proposed software defect detection model

The figure 1 depicts the work flow of proposed model in this the dataset named as CM1 is collected from NASA repository. The selected raw dataset is preprocessed using normalization approach in which the dataset are converted to the values of the same range. The three different classification models namely naïve bayes, simple logistics and Zero R are applied on the selected dataset to classify whether the given software module is prone to defect or not. The performance comparison is done using weka tool kit. The result shows that the Zero R classifier produces that higher percentage of accuracy in software defect prediction

ZeroR

ZeroR is the simplest classification method which relies on the target and ignores all predictors. ZeroR classifier simply predicts the majority category (class). Although there is no predictability power in ZeroR, it is useful for determining a baseline performance as a benchmark for other classification methods.
ZeroR is the simplest classification method which relies on the target and ignores all predictors .ZeroR classifier [25] simply predicts the majority category (class). Although there is no predictability power in ZeroR, it is useful for determining a baseline performance as a benchmark for other classification methods. **Algorithm** Construct a frequency table for the target and select its most frequent value. **Predictors Contribution** There is nothing to be said about the predictors contribution to the model because ZeroR does not use any of them.
Model Evaluation The ZeroR only predicts the majority class correctly. As mentioned before, ZeroR is only useful for determining a baseline performance for other classification methods.

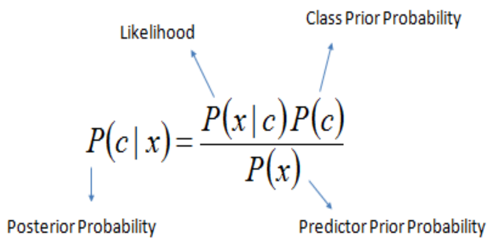
Naive Bayes classifiers

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s, [21] and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. [22] It also finds application in automatic medical diagnosis. [23]
Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, [21] which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the statistics and computer science literature, Naive Bayes models are known under a variety of names, including **simple Bayes** and **independence Bayes**. [24] All these names reference the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not (necessarily) a Bayesian method. [21][24].

Algorithm

Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

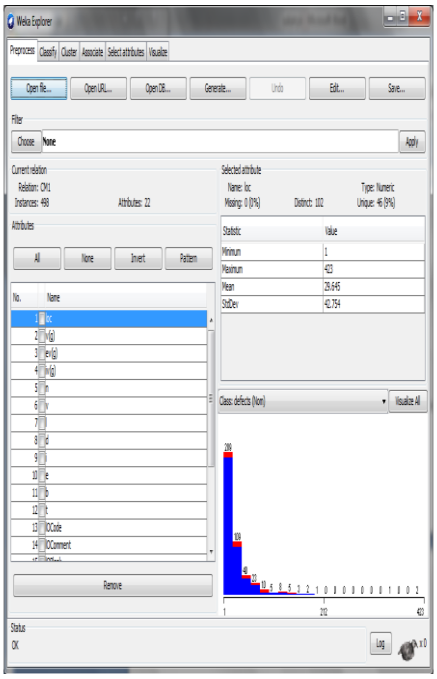
- *P*(*c*|*x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

Simple Logistic Regression

Classifier for building linear logistic regression models. LogitBoost with simple regression functions as base learners is used for fitting the logistic models. The optimal number of LogitBoost iterations to perform is cross-validated, which leads to automatic attribute selection.

Experimental result

This section discuss on performance of three different classifiers towards software defect prediction. The dataset used in this paper is a complexity metric of a software. The weka tool kit is used for performing prediction process



Current relation
Relation: CM1
Instances: 498 Attributes: 22

Selected attribute
Name: loc Type: Numeric
Missing: 0 (0%) Distinct: 102 Unique: 46 (9%)

Statistic	Value
Minimum	1
Maximum	423
Mean	29.645
StdDev	42.754

No.	Name
1	loc
2	v(g)
3	ev(g)
4	iv(g)
5	n
6	v
7	l
8	d
9	i
10	e
11	b
12	t
13	lOCode
14	lOComment

Class: defects (Nom)

Status
OK

Figure 2 Loading software defect dataset in weka

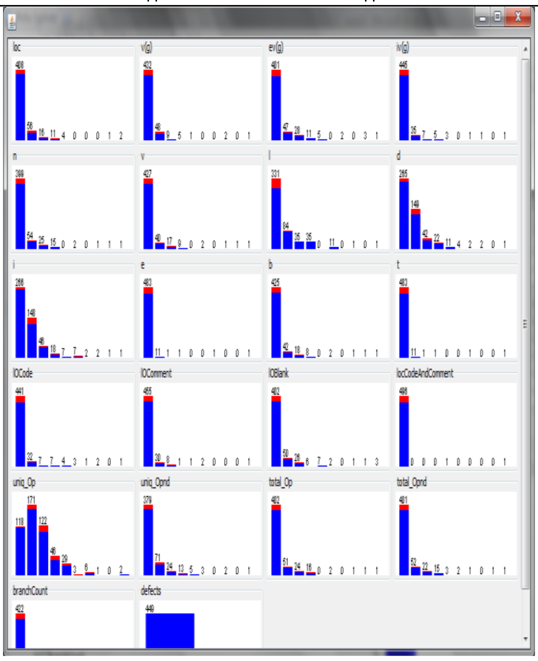


Figure 3 Distribution of attribute values from the selected dataset

Evaluation Metrics

This work used NASA dataset namely CM1. In order to classify the software module as defect or no defect using naïve bayes, simple logistics and Zero R classifier are applied and their performance is compared using the following metrics.

Table 3: The confusion matrix for the software defect prediction is depicted in the following table 3.

		Module actually has defects	
		No (P)	Yes (N)
Classifier predicts no defects	No (P)	a (TP)	b (FP)
Classifier predicts some defects	Yes (N)	C (FN)	d (TN)

TP = true positives: number of examples predicted positive that are actually positive
FP = false positives: number of examples predicted positive that are actually negative
TN = true negatives: number of examples predicted negative that are actually negative
FN = false negatives: number of examples predicted negative that are actually positive

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{tp+tn}{tp+tn+fp+fn} \qquad (18)$$

$$\text{Probability of Detection} = \text{PD} = \text{Recall} = \frac{d}{b+d} = \frac{tp}{tp+fn} \qquad (19)$$

$$\text{Probability of False Alarm} = \text{PF} = \frac{c}{a+c} = \frac{fp}{tp+fp} \qquad (20)$$

Precision = $\frac{d}{c+d} = \frac{tp}{tp+fp}$ (21)

F –Measure = $2 * \frac{\Pr ecision.recall}{\Pr ecision + recall}$ (22)

Relative Absolute Error = $\frac{|p_1 - a_1| + ... + |p_n - a_n|}{|\bar{a} - a_1| + ... + |\bar{a} - a_n|}$ (23)

Root relative squared error = $\frac{(p_1 - a_1)^2 + ... + (p_n - a_n)^2}{(\bar{a} - a_1)^2 + ... + (\bar{a} - a_n)^2}$ (24)

Where,

- Actual target values: a1 a2 … an
- Predicted target values: p1 p2 … pn
- Mean value of actual target values: $\bar{a}$

Dataset Description

This subsection describes the datasets used in this work. Four public data sets CM1, JM1, PC1 and KC1obtained from NASA MDP [1, 2] Repositorymade available by PROMISE [3]are used for evaluation of the proposed work. Each of these dataset consists of a set of features characterized by static code metrics, such as LOC counts, Halstead and McCabe complexity metrics. These features are characterizing objectively the software quality. The In McCabe metrics are a collection of four software metrics:Essential complexity, cyclomatic complexity, design complexity and Lines of Code.CM1 dataset consists of 498 instances, JM1 consists of 10885, KC1 dataset consists o 2109 instances and PC1 contains 1109 instances. All the 4 dataset consists of 22 attributes. In the dataset 5 of the attributes are used for representing different lines of code measure, 3 attributes represents the McCabe metrics, 4 attributes refers to Halstead measures, 8 attributes refers derived Halstead measures, a branch-count attribute, and 1 goal field attribute which is called as class which classifies the instance as presence or absence of defect.

Table 4 shows the description of each attributes used in the four dataset of this research work.

Table 4: Attribute Description of the four Dataset

S.No	Variables	Description
1	Loc	McCabe's line count of code
2	v(g)	McCabe "cyclomatic complexity"
3	ev(g)	McCabe "essential complexity"
4	iv(g)	McCabe "design complexity"
5	N	Halstead total operators + operands
6	V	Halstead "volume"
7	L	Halstead "program length"
8	D	Halstead "difficulty"
9	I	Halstead "intelligence"
10	E	Halstead "effort"
11	B	Halstead
12	T	Halstead's time estimator
13	lOCode	Halstead's line count
14	lOComment	Halstead's count of lines of comments
15	lOBlank	Halstead's count of blank lines
16	lOCodeAndComment	
17	uniq_Op	unique operators
18	uniq_Opnd	unique operands

19	total_Op	total operators
20	total_Opnd	total operands
21	branchCount	% of the flow graph
22	Defects	Yes/No module has/has not one or more

Performance of Naïve Bayes Classifier

Performance of Naïve Bayes Classifier	
Correctly classified	85.3414
Incorrectly classified	14.6586
Mean absolute error	0.1524
Root mean squared error	0.38
Relative absolute error	85.2218 %
Root relative squared error	127.572 %
TPR	0.911
FP rate	0.673
Precision	0.925
Recall	0.911
F measure	0.918

The table 2 shows the performance of the classifier naïve bayes which holds 85.3% as correctly classified and 14.65% as incorrectly classified. The relative absolute error is 85% and the root relative squared error is 127.57%. The True positive rate and false positive rate is .91 and .67 respectively. The naïve bayes on the whole performs lowest while comparing the other two methods.

Performance of SL Classifier	
Correctly classified	89.1566
Incorrectly classified	10.8434
Mean absolute error	0.3682
Root mean squared error	0.4327
Relative absolute error	205.8763 %
Root relative squared error	145.2821 %
TPR	0.989
FP rate	1
Precision	0.901
Recall	0.989
F measure	0.943

The table 3 shows the performance of the classifier simple logistics which holds 89% as correctly classified and 10.8% as incorrectly classified as presence of software defect or not. The relative absolute error is 205% and the root relative squared error is 145.57%. The True positive rate and false positive rate is .98 and 1 respectively.

The table 4 shows the performance of the classifier Zero R which holds 90% as correctly classified and 9.8% as incorrectly classified as presence of software defect or not. The relative absolute error is 100% and the root relative squared error is 100%. The True positive rate and false positive rate is 1 and 1 respectively.

Performance of Zeror Classifier	
Correctly classified	90.1606
Incorrectly classified	9.8394
Mean absolute error	0.1789
Root mean squared error	0.2979
Relative absolute error	100 %
Root relative squared error	100 %
TPR	1
FP rate	1
Precision	0.902
Recall	1
F measure	0.948

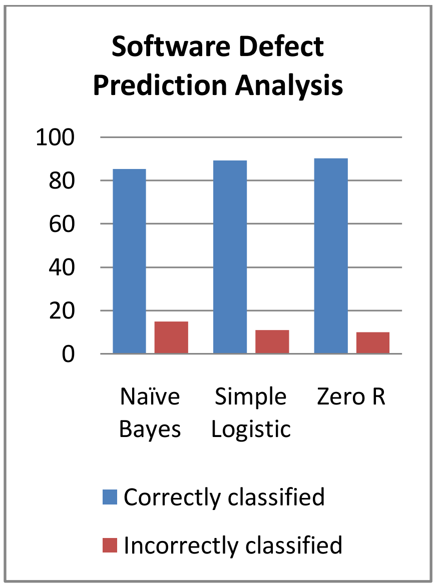


Figure 4 Software defect prediction analysis of three classifier based on correctly and incorrectly classified instances

The figure 4 and 5shows that the performance of the ZeroR performs better than the remaining two classifiers by analyzing the correctly and incorrectly classified instances.

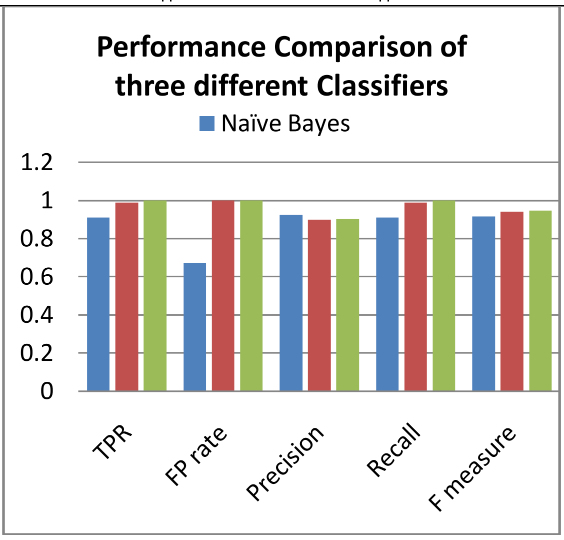


Figure 5 Performance comparisons of three classifiers

Conclusion

Predicting the software defect in advance is a very challenging problem. To overcome this and handle efficiently the data mining techniques greatly assist in finding it. This paper aims at developing the best classifier for predicting the software defect presence. The experimental result is done using the weka toolkit. The three classifiers namely naïve bayes, simple logistics and zero r are analyzed for determining the predicting process. The result shows that the performance of Zero R plays more accurately in predicting the presence of software defect.

Reference

[1]. G. Graham, E.V. Veenendaal, I. Evans, R. Black, “Foundations of Software Testing: ISTQB Certification”, Thomson Learning, United Kingdom, 2007.
[2]. N.E. Fenton, M. Neil, “A Critique of Software Defect Prediction Models”, IEEE Transactions on Software Engineering, vol. 25, no.5, pp.675-689, 1999.
[3]. B. Clark, D. Zubrow, “How Good is the Software: A Review of Defect Prediction Techniques”, Carnegie Mellon University, USA, 2001.
[4]. V. Nayak, D. Naidya, “Defect Estimation Strategies”, Patni Computer Systems Limited, Mumbai, 2003.
[5]. M. Thangarajan, B. Biswas, “Software Reliability Prediction Model”, Tata Elxsi Whitepaper, 2002.
[6]. D. Wahyudin, A. Schatten, D. Winkler, A.M. Tjoa, S. Biffl, “Defect Prediction using Combined Product and Project Metrics: A Case Study from the Open Source “Apache” MyFaces Project Family” In Proceedings of Software Engineering and Advanced Applications (SEAA '08), 34th Euromicro Conference, pp. 207-215, 2008.
[7]. I. Sinovcic, L. Hribar, “How to Improve Software Development Process using Mathematical Models for Quality Prediction and Element of Six Sigma Methodology”, In Proceedings of the 33rd International Conventionions 2010 (MIPRO 2010), pp. 388-395, 2010.
[8]. E.J. Weyuker, T.J. Ostrand, R.M. Bell, “Using Developer Information as a Factor for Fault Prediction”, In Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), pp.8, 2007.
[9]. T. Gyimothy, R. Ferenc, I. Siket, “Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction”, IEEE Transactions on Software Engineering, vol. 31, no.10, pp. 897-910, 2005.
[10]. J.S. Collofello, “Simulating the System Test Phase of the Software Development Life Cycle”, In Proceedings of the 2002 Summer Software Computer Simulation Conference, 2002.
[11]. L. RadliRski, “Predicting Defect Type in Software Projects”, Polish Journal of Environmental Studies, vol.18, no. 3B, pp. 311-315, 2009.
[12]. M. Staron, W. Meding, “Defect Inflow Prediction in Large Software Projects”, e-Informatica Software Engineering Journal, vol. 4, no. 1, pp. 1-23, 2010.
[13]. T. Fehlmann, “Defect Density Prediction with Six Sigma”, Presentation in Software Measurement European Forum, 2009.

[14]. S.W. Haider, J.W. Cangussu, K.M.L. Cooper, R. Dantu, "Estimation of Defects Based on Defect Decay Model: ED3M", IEEE Transactions on Software Engineering, vol. 34, no. 3, pp. 336-356, 2008.

[15]. L. Zawadski, T. Orlova, "Building and Using a Defect Prediction Model", Presentation in Chicago Software Process Improvement Network, 2012.

[16]. M. Li, H. Zhang, R. Wu, Z.H. Zhou, "Sample-based Software Defect Prediction with Active and Semi-supervised Learning", Journal of Automated Software Engineering, vol. 19, no. 2, pp. 201-230, 2012.

[17]. Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, "An Investigation on the Feasibility of Cross-Project Defect Prediction", Journal of Automated Software Engineering, vol. 19, no. 2, pp. 167-199, 2012.

[18]. T.J. Ostrand, E.J. Weyuker, "How to Measure Success of Fault Prediction Models", In Proceedings of Fourth International Workshop on Software Quality Assurance 2007 (SOQUA '07), pp. 25-30, 2007.

[19]. L.P. Li, M. Shaw, J. Herbsleb, "Selecting a Defect Prediction Model for Maintenance Resource Planning and Software Insurance", In Proceedings of 5th Workshop on Economics-Driven Software Engineering Research (EDSER '03), pp. 32-37, 2003.

[20]. M. D'Ambros, M. Lanza, R. Robbes, "Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison, Journal of Empirical Software Engineering, vol. 17, no. 4-5, pp. 531-577, 2012.

[21]. Russell, Stuart*;* Norvig, Peter *(2003) [1995].* Artificial Intelligence: A Modern Approach *(2nd ed.). Prentice Hall.* ISBN 978-0137903955.

[22]. *Rennie, J.; Shih, L.; Teevan, J.; Karger, D. (2003).* Tackling the poor assumptions of Naive Bayes classifiers *(PDF). ICML.*

[23]. Rish, Irina (2001). An empirical study of the naive Bayes classifier (PDF). IJCAI Workshop on Empirical Methods in AI.

[24]. Hand, D. J.; Yu, K. (2001). "Idiot's Bayes — not so stupid after all?". International Statistical Review. 69 (3): 385–399. ISSN 0306-7734. doi:10.2307/1403452.

[25]. http://chem-eng.utoronto.ca/~datamining/dmc/zeror.htm