Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 132 (2018) 1412–1421

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Intelligence and Data Science (ICCIDS 2018)

Software Bug Prediction Prototype Using Bayesian Network Classifier: A Comprehensive Model

Sushant Kumar Pandey*, Ravi Bhushan Mishra, Anil Kumar Triphathi

Department of Computer Science and Engineering, Indian Institute of Technology-BHU, Uttar Pradesh, India-221005

Abstract

Software bug prediction becomes the vital activity during software development and maintenance. Fault prediction model able to engaged to identify flawed software code by utilizing machine learning techniques. Naive Bayes classifier has often used times for this kind problems, because of its high predictive performance and comprehensiveness toward most of the predictive issues. Bayesian network(BN) able to construct the simple network of a complex problem using the fewer number of nodes and unexplored arcs. The dataset is an essential phase in bugs prediction, NASA/Eclipse free-ware are freely available for better results. ROC/AUC is a performance measure for classification of fault-prone or non-fault prone, H-measure is also useful while prediction technique, we will explore every parameter and valuable expects for experiment perspective.

Keywords: Bayesian network; bug prediction; classification techniques

1. Introduction

In the modern age of ubiquitous computing, it has given massive rise in research work namely artificial intelligence, machine learning, software engineering whereas research development in telecommunication, medicine, and image/audio/video processing. Although software fault prediction has still so much scope to work. Machine learning is one of the best ways to handle this kind of problems.

Different approaches have been applied for software fault prediction(SFP) such as statistical method, expert-driven models, and various machine learning algorithms[1]. Although some advanced methods such as support vector machine(SVM) [2], neural network[3], swarm intelligence[4] and genetic programming[5]. The Naive Bayes model[6] also involved in SFP with limitations. In spite of restriction of network architecture, it applies assumption of conditional interdependency. The performance of Naive Bayes can be hike compared with other classification techniques by some modification such as relaxing conditional independence; then it will allow us to construct more complex

* Corresponding author. Tel.: +91-9451837790.
E-mail address: (sushantkp.rs.cse16, mishravi.cse, aktripathi.cse)@iitbhu.ac.in

10.1016/j.procs.2018.05.071

network structure. Some famous example included General Bayesian network and Augmented Naive Bayes classifier. Then it will not impose any restriction on network architecture and able to learn appropriate network architecture. The two such techniques jointly with different Augmented Naive Bayes classifier and selecting of benchmark classifier combines and make technology, and that method considers in our study.
In the next sections, we will elaborate related work, there are different NB classifier models which are explored in various NB classifier and how the experiments are going to conduct that section will be elaborate in experimental design.

2. Related work

Faulty of software is analysis from the different point of view, regarding failure probability, every time a software component performs the risk of failure is available. Some models associate information with each other over different elements regarding reliability of data [7]. For proper presentation of related work, total seven categories have been made which is given below.

(i) *Studies related to different classification techniques which depend on heuristic or intelligent perspective*. Chill et al. [8] justify the result between SVM and Naive Bayes after experimenting using nine different datasets from NASA MDP. They identified SVM is more successful than SVM. Carrozza [9] et al. compare the results between Decision tree, Bayesian Network, SVM, Logistic Regression using industrial dataset gave a much better result. The overall study suggests that performance of all the techniques are similar, but Random Forest(RF) and K2 are better amongst them.

(ii) *Analysis of combined methods to achieve successful result:*According to research, after combing the methods will able to recognizing faulty modules in the software system. The experiment performed by Cortroneo[10] includes NB+ logarithmic transformation(log), LB+log, DT+log uses Linux based operating system, MYSQL, DBMS project related dataset for an experiment, it was about to telecommunication by Kasai et al. [11]. The combined techniques by Chen et al.[12] Random sampling(Rs) + NB + C4.5 and RS + K-nearest neighbor(KNN) using promise and NASA MDP dataset. Li. et al.[13] experimented using combine clustering algorithm + C4.5 using Eclipse dataset.

(iii) *Uniques approach for SFP:* The idea of a self-organized map by Abaei et al[14], the concept which had not used for SFP problem. The method uses AR3, AR4, AR5 datasets from promise repository. The fuzzy inference system(FIS) for SFP was first used by Ertuk and Sezer applying promise dataset repository[15]. SFP applying relational association rule proposed by Czibula et al.[16] by using promise dataset.

(iv) *Uses of various metrcis for SFP:*The research that explores software metrics for SFP attempt to figure-out relevant metrics for SFP. Catal and Dri[1] first investigate the class level/component level of metrics. Then Oyentoyan et al.[17] present discussion about cyclic dependencies and explore metrics for some industrial application using various project dataset. The first experiment performed using Chidamber-Kemerer metrics(object-oriented metrics), process metrics and McCabe metrics using KC1 dataset from promise repository by Ertuk et al[18].

(v) *Studies that suggest preprocessing techniques:*Preprocessing help to remove extraneous parameters. Cahill et al.[8] introduced Rank sum although Chen et al.[19] introduced two phase of preprocessing. In the first period, Chi-square, symmetrical uncertainty is applied, and in second phase clustering method was employed. Shivaji et al.[20] are explore various preprocessing methods example Wrapper method, Chi-squared attribute selection, Relief-F attribute selection and significance attribute evaluation. Lu.et al.[12] explored about multidimensional scaling as preprocessing technique and verified on KC1, PC1, PC2 and PC3 NASA MDP datasets.

(vi) *Cross-project fault prediction research:* The idea of cross-project to learn about project A from project B. Ma et al.[21] applied cross-project fault prediction concept in an experiment by using Transfer NB method, which is mainly transfer learning method, it was involved in seven different promise datasets. Canfora et al.[22] applied the multi-objective genetic algorithm and verified over ten different promise datasets. Turhan et al.[23] explored recall increment and uncertainty of false alarm rate using NB over six datasets from promise repository.

(vii) *Research that applies solutions for incomplete data in initial phase of software life-cycle:* Kamiya et al.[24] proposed four checkpoints for fault proneness prediction in design/implementation stage by using Chidamber-Kermerer metrics. The experiment was executed over a private project. Nagappan and Ball[25] used two different static tools for analysis(PREfix & PREfast) to identify previous fault density for windows. Lu et al.[26] recommended confident fits method which is half supervised technique; it can able to predict defect for limited size data.

3. Bayesian Network Classification Techniques

This section introduced by the Bayesian network(NB) and various NB classifiers. Explanation of substitute BN classifier and relaxing the premise of conditional independence. Two other ML techniques also explained which have an essential role in this study.
Following are some annotation which was employed in the entire paper.
$k\rightarrow$ Identifier for attribute
$i\rightarrow$ Identifier for instance
$n\rightarrow$ Total number of attributes
$N\rightarrow$ Total number of instances
$x_{i(k)} \epsilon R\rightarrow$ Scalar representation of value i^{th} instance on k^{th} attribute
$y_i\rightarrow$ Bidirectional target implies an instance is faulty(y_i=*1*) or non faulty(y_i=*0*)
After using annotation, the new task is to develop a learning mathematical model M for fault prediction.

$$M = train(x_i, y_i)_{i=1}^{N} \tag{1}$$

where $x_{i(j)} \epsilon R^N$ implies feature characteristics of i^{th} instance and y_i represents availability of fault, when $y_i = 1$ faulty and $y_i = 0$ non-faulty. SFP model delivers the mapping to the instances x_i to the probability of faulty class code portion, $P(y_i = 1|x_i)$. As the formal equation looks like.

$$f(x_i) : R^n \rightarrow P(y_i = 1|x_i) \tag{2}$$

3.1. The Bayesian Network

The combined probability of stochastic variables which can be continuous or discrete is represented by BN. It can be expressed in the graph, every individual variable (x_i) represents as nodes and dependencies among variables imply as directed arcs. If there are no arcs between two variables(two nodes), it means the variables are conditionally independence to each other given in parents graph. Each node is associated with probability table, which contains probability distribution of every node(variable) to direct parent(s) into graph[27]. The Bayesian theorem contrives the posterior fault probability by prior fault probability using conditional probability.

$$P(y_i = 1|x_i) = \frac{P(x_i|y_i = 1)P(y_i = 1)}{P(x_i)} \tag{3}$$

where $P(x_i)$ behave as normalizing constant, it can be ignored according to requirement.
BN consists two elements B=(G,Θ), here G is directed acyclic graph, which carries information within dataset about direct dependence relationship whereas Θ illustrate conditional probability distribution for every variable. According to notation of Cooper et al.[28] $\Phi_{x_{(j)}}$ showed set straight parents $x_{(j)}$ into G. Θ consist $\theta_{x_{(j)}|\Phi_{x_{(j)}}} = P_B(x_{(j)}|\Phi_{x_{(j)}})$ for every

variable of $x_{(j)'}$. The representation of joint probability of network B are showing below.

$$P_{B(x_{(1)},....x_{(n)})} = \sum_{k=1}^{N} P_B(x_{(k)}|\Phi_{x_{(k)}}) = \sum_{k=1}^{N} \theta_{x_{(k)}|\Phi_{x_{(k)}}} \tag{4}$$

The learning task for BN is divided into two subtasks which will orderly executed. Firstly, the complete architecture of network G will be determined. It is impossible to understand all possible network architecture. After entire determination architecture of network G, every node is related with some parameters, and that needs to be evaluated. M_{train} are utilized to evaluate the empirical frequencies which are observed by parameters.

$$\theta_{x_{(k)}|\Phi_{x_{(k)}}} = P_{M_{train}}(x_{(k)}|\Phi_{x_{(k)}}) \tag{5}$$

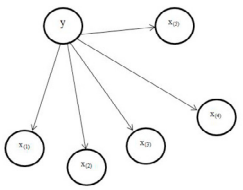
The above equation will further improved by Laplace correlation.

4. Naive Bayes Classifier

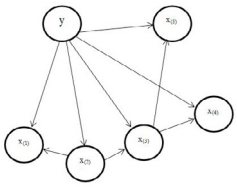
The very first classifier constructed based on Bayesian network principle is NB classifier[29]. The NB classifier pursue by determining posterior probability of every class of variables inputs($x_{i(1)}, ...x_{i(n)}$). Using Bayesian equation 3, by assuming conditional independence, the probability of class conditional is as follows.

$$P(x_{i(1)}, ...x_{i(n)}|y_i = y) = \sum_{k=1}^{n} P(x_{i(k)}|y_i = y) \tag{6}$$

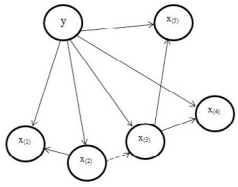
Estimation of probabilities $P(x_{i(k)}|y_i = y)$ are done using frequency count regarding normal or discrete variables and kernel density approach regarding continuous variables. After presumption of conditional independence, NB classifier is uncomplicated to build since network shape is prior and no shape learning period is needed. Computational efficiency is another benefits with NB classifier when the model in product form it can be easily converted in sum by applying logarithmic transformation.



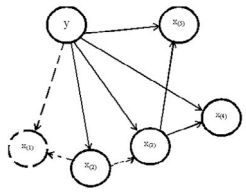
(a) Basic NB structure



(b) Selective Tree Augmented NB structure



(c) Selective Forrest Augmented NB structure



(d) Selective Forrest Augmented NB with discarding structure

Fig. 1: Various types of NB structures

5. Augmented NB classification Techniques

Several modifications have been done and relaxation on several parameters for conditional independence to achieve the better result. As suggested by Karel et al.[30], the amendment mainly involved adding arcs furthermore in between

variables regarding the availability of dependencies in data and removing unrelated or irrelevant variables from network architecture. In the Fig 1, the various NB structures are shown. The basic structure of NB is demonstrated in

Table 1: A various operators: Augmented NB model

Dependency Operator	**Explanation**
Selective Augmented Naive Bayes (SAN)	*In the start, the set will be empty, SAN searches for the all possible arcs over the class variable(y), to some another variable(x_k) for optimize various quality parameters. The selected variable are associated with arcs then added to the network, then one of the augmented operators passed in the system. The variable organized dependencies over every variables(x_k) independent to their connection with class node.*
Selective Augmented Naive Bayes with Discarding (SAND)	*It is similar to SAN operator, in which the operator link with class node according to dependent attribute. The basic difference between SAN and SAND is it rejects every variable which is independent of a class node, before passing to augmenting variable. The variable which was rejected will be not part of the network. The figure 1c and 1d are the basic SAN and SAND structure respectively.*

Table 2: Various Augmented operators

Augmented operators	**Explanation**
Tree Augmenter	*From the available set of attributes, the operator built a maximum number of spanning tree. The unconditional and conditional probability of x_i&$x_{k'}$ are mainly dependent on the availability of the attribute and arc linking the class node. Class variable need not to be connected with each attribute. The resulted network is tree augmented NB classifier, see fig1b.*
Forest Augmenter	*This operator is also for creating dependencies among attribute but a bit flexible. By the number of disjoint trees, the forest augmenter establishes dependencies among variables. There is no need of undirected path between two attribute, which not passes from the class node. As shown in Fig 1b,1c.*

Fig1a, whereas the selective forest with discarding option is shown in Fig 1d. In Table 1 different augmented operators for NB models are shown, whereas in the Table 2 different augmented operators are shown. After combining augmenting operators from Table1 and Table2, the conditional dependency between two attribute is shown in equation 7. There are four different possibility arises. Note that the augmented operator can be directly applied to the Bayesian

network.

$$\begin{cases} \sum_{x_{(k)},x_{(k')}} P(x_{(k)}x_{(k')}|k) \log \dfrac{P(x_{(k)}, x_{(k')}|y)}{P(x_{(k)}|y)P(x_{(k')})} \\ when \quad y \prec x_{(k)} \\ \sum_{x_{(k)},x_{(k')}} P(x_{(k)}x_{(k')}|k) \log \dfrac{P(x_{(k)}, x_{(k')}|y)}{P(x_{(k)})P(x_{(k')}|y)} \\ when \quad y \prec x_{(k')} \\ \sum_{x_{(k)},x_{(k')}} P(x_{(k)}x_{(k')}|k) \log \dfrac{P(x_{(k)}, x_{(k')}|y)}{P(x_{(k)}|y)P(x_{(k')}|y)} \\ when \quad y \prec x_{(k)}.x_{(k')} \\ \sum_{x_{(k)},x_{(k')}} P(x_{(k)}x_{(k')}|k) \log \dfrac{P(x_{(k)}, x_{(k')})}{P(x_{(k)})P(x_{(k')})} \\ when \quad y \perp x_{(k)}.x_{(k')} \end{cases} \tag{7}$$

6. Experiment Design

In this section, we explore about experiment steps required in bugs prediction. There is different free-ware dataset available for bugs prediction, Promise, NASA, etc. The preprocessing and classification evaluation are two most crucial phase of the experiments. The final steps are statistical testing.

6.1. Datasets

Datasets have an essential credential in an experiment, we have considered two open source datasets, first is NASA datasets which freely available and can be directly downloaded from the repository(Metric data program) of NASA. Second is Eclipse dataset; it is also openly can be obtained from Eclipse foundation. Both of them datasets are publicly available to inspire researcher for validation and finding better results. The static code parameters are heterogeneous, which includes object-oriented(OO) metrics, Halstead metrics, LOC, and McCabe complexity. Table 3 and 4 give an complete overview about NASA and Eclipse datasets. The information regarding dataset is the very important phase; researcher should have the full description of the dataset. The faulty modules in the Table 3 and 4 give information about the percentage of the faulty phase in the complete software.

6.2. Data Preprocessing

The crucial initial stage in any data mining related employment is preprocessing data. Every observation(software module) in the dataset contains unique identification number(ID), various static features, number of error count. Firstly data ready to learn and validates, then selected, so ID or attribute which have null variance will be abandon. Furthermore, if any observation with zero line of the count is logically erroneous and will be removed. In NASA dataset, the ”error density” parameters will also remove. The ”error count” converted into Boolean values, 0 implies error-free recorded and 1 for existence of error. Many of the Bayesian learner are not capable to manage continuous features, so it is converted into discrete values by algorithm of Irani and Faiyad[31].
ML techniques will be more efficient when the data many more datasets are available for learning. Every dataset needs to partition into two mutually exclusive sets randomly, one for(2/3) training and other for testing(1/3), for avoiding sampling bias this partitioned need to be iterate to ten to fifteen times. After performing all these things, there is less chance of handling missing values.

6.3. Classifier assessment

The performance of the classifier is measured using various parameters. A common performance measurement tools are ROC(receiver operating characteristics) method. Every classifier obtain a score s(x_i) to each instance i.

Table 3: NASA dataset: An Overview

NASA	About	Size	Faulty modules
PC1	*C program of flight software(no more usable)*	*40(KLOC)*	*76 / 1059 (7.19%)*
PC2	*Attitude control system program in C*	*26(KLOC)*	*23 / 4.5 (0.51%)*
PC3	*C program of flight software*	*40(KLOC)*	*160 / 1.51 (10.60%)*
PC4	*C program of flight software*	*36(KLOC)*	*178 / 1.34 (13.2%)*
PC5	*Safety enhancement software in C++*	*165(KLOC)*	*503 / 15400 (3.25%)*
KC1	*Storage management system software in C++, from rough data*	*43(KSLOC)*	*325 / 2108 (15.43%)*
MC1	*Software for combustion experiment in C/C++*	*63(KLOC)*	*68 / 4625 (1.48%)*
JM1	*C program for real time system*	*315(KSLOC)*	*2.102 / 10878 (19.32%)*

Table 4: Eclipse dataset: An Overview

NASA	About	Size	Faulty modules
ECL-2.0	*Publicly available on 27^{th} June 2002*	*797(KLOC)*	*975 / 6730 (14.50%)*
PC2	*Publicly available on 27^{th} March 2003*	*988(KLOC)*	*855 / 7.89 (10.84%)*
PC3	*Publicly available on 25^{th} June 2004*	*1309(KLOC)*	*1570 / 10595 (14.80%)*

Classification occurs by that score; a threshold t is defined, when the score of any instance is less than t, it will be categories as non-faulty prone. The curve that shows fraction that classified faulty instance versus one minus fraction that non-faulty instance is ROC curve. The area under ROC curve(AUC) is defined by, let $p_l(s)$ be the probability density function of s(score), and l is the classes belongs to 0,1, $P_l(s)$ is the corresponding cumulative distribution function. The AUC will defined as follows:

$$AUC = \int_{-\infty}^{+\infty} P_0(s)p_l(s)ds \tag{8}$$

Let the cost of the misclassifying faulty instance as non-faulty be c_0, and c_1 be the faultless instance which is categories as fault-prone. Then the anticipated loss of minimum misclassification is showing in equation showing below.

$$L = P_{(b)} \int_0^1 [c\delta_0(1 - P_0(T)) + (1 - c)\delta_1 P_1(T)]u(c)dc \tag{9}$$

where δ_0 is the prior probability of fault-prone δ_1 the prior probabilities of non-fault prone instance, and the optimal threshold for given values of c and t is T.

When their is no extra information about c is available, then a symmetric distribution is propose with $\alpha = \beta = 2$ called *H-measure parameters*. There is no especially cost for fault prediction, using these default values H-measure will be calculated. Misclassification of a faulty instance as non-faulty is very significant issues as compared to the opposite of it.

7. Testing

There are two main points for consideration , first is what type of Bayesian classifier used and second the feature selection for model. The Friedman test is considers for this two factors, it is a nonparametric way to detect multiple test attempt. The Friedman testing is defined as

$$\psi_F^2 = \frac{12P}{k(k+1)} \sum_m R_m^2 - \frac{k(k+1)^2}{4} \tag{10}$$

where R_m is average rank of every treatment m = 1, 2,....,k on P number of test strive. The ψ_F^2 distributed over k-1 degree of freedom, when k and P are very large(k*P>30).
When the Friedman rejected null hypothesis, Nemenyi test is applied for comparing all the factor with each other. The Nemenyi is also a nonparametric test and defined as

$$C = q_a \sqrt{\frac{k(k+1)}{6P}} \tag{11}$$

Where q_a is critical value and based on Studentized range, which is divided by $\sqrt{2}$, and C is critical difference which is performance difference between treatment and their average ranks.

8. Conclusion and future work

The objective of this paper to compare various Bayesian network and classifier and how they are useful for bugs prediction. Regarding ROC, AUC and lastly introduced H-measure. As from the study, the augmented classifier can be better performing than another Bayesian classifier. If we want to boost performance than, then the model will become complicated. Then general Bayesian networks over performed concerning another Bayesian learner of created very complex network architecture. Toward development point of view, the cost of misclassification instances or not faulty on a non transparent model is very much different. Our supportive model rather then BN can be the random forest for better classification, the model can be more predictive and less costly. A testing condition can be more difficult, but contradictory the Augmented Bayesian network will be the better choice. There can be many queries that performance of the Support vector machine and Neural network under such situation, that area is still unexplored.
According to latest research the addition information besides static code in fault prediction model. That additional information is about inter-module relation and metrics requirements. By context development, the cost associated with misclassification a faulty instance, some another model needs to be more discriminative.

Acknowledgment

This research paper is part of the Ph.D. thesis of the first Author, and IIT-BHU, India support it. The author would like to thank department of computer science and engineering of IIT-BHU for providing such an outstanding research platform.

References

[1] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[2] Karim O Elish and Mahmoud O Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.

[3] Tong-Seng Quah and Mie Mie Thet Thwin. Application of neural networks for software quality prediction using object-oriented metrics. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 116–125. IEEE, 2003.

[4] Olivier Vandecruys, David Martens, Bart Baesens, Christophe Mues, Manu De Backer, and Raf Haesen. Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and software*, 81(5):823–839, 2008.

[5] Matthew Evett, Taghi Khoshgoftar, Pei-der Chien, and Edward Allen. Gp-based software quality prediction. In *Proceedings of the Third Annual Conference Genetic Programming, volume*, pages 60–65, 1998.

[6] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13, 2007.

[7] Swapna S Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Transactions on dependable and secure computing*, 4(1), 2007.

[8] Jaspar Cahill, James M Hogan, and Richard Thomas. Predicting fault-prone software modules with rank sum classification. In *Software Engineering Conference (ASWEC), 2013 22nd Australian*, pages 211–219. IEEE, 2013.

[9] Gabriella Carrozza, Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. Analysis and prediction of mandelbugs in an industrial software system. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 262–271. IEEE, 2013.

[10] Domenico Cotroneo, Roberto Natella, and Roberto Pietrantuono. Predicting aging-related bugs using software complexity metrics. *Performance Evaluation*, 70(3):163–178, 2013.

[11] Norimitsu Kasai, Shuji Morisaki, and Kenichi Matsumoto. Fault-prone module prediction using a prediction model and manual inspection. In *Software Engineering Conference (APSEC), 2013 20th Asia-Pacific*, volume 1, pages 106–115. IEEE, 2013.

[12] Wangshu Liu, Shulong Liu, Qing Gu, Jiaqiang Chen, Xiang Chen, and Daoxu Chen. Empirical studies of a two-stage data preprocessing approach for software fault prediction. *IEEE Transactions on Reliability*, 65(1):38–53, 2016.

[13] Biwen Li, Beijun Shen, Jun Wang, Yuting Chen, Tao Zhang, and Jinshuang Wang. A scenario-based approach to predicting software defects using compressed c4. 5 model. In *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pages 406–415. IEEE, 2014.

[14] Golnoush Abaei, Zahra Rezaei, and Ali Selamat. Fault prediction by utilizing self-organizing map and threshold. In *Control System, Computing and Engineering (ICCSCE), 2013 IEEE International Conference on*, pages 465–470. IEEE, 2013.

[15] Ezgi Erturk and Ebru Akcapinar Sezer. Software fault prediction using mamdani type fuzzy inference system. *International Journal of Data Analysis Techniques and Strategies*, 8(1):14–28, 2016.

[16] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. Software defect prediction using relational association rule mining. *Information Sciences*, 264:260–278, 2014.

[17] Tosin Daniel Oyetoyan, Daniela S Cruzes, and Reidar Conradi. A study of cyclic dependencies on defect profile of software components. *Journal of Systems and Software*, 86(12):3162–3182, 2013.

[18] Ezgi Erturk and Ebru Akcapinar Sezer. Software fault prediction using fuzzy inference system and object-oriented metrics. In *Proceedings of the 13th IASTED International Conference on Software Engineering Austria*, pages 101–108, 2014.

[19] Jiaqiang Chen, Shulong Liu, Xiang Chen, Qing Gu, and Daoxu Chen. Empirical studies on feature selection for software fault prediction. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, page 26. ACM, 2013.

[20] Shivkumar Shivaji, E James Whitehead, Ram Akella, and Sunghun Kim. Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, 39(4):552–569, 2013.

[21] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54(3):248–256, 2012.

[22] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Multi-objective cross-project defect prediction. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 252–261. IEEE, 2013.

[23] Burak Turhan, Ayşe Tosun Mısırlı, and Ayşe Bener. Empirical evaluation of the effects of mixed project data on learning defect predictors. *Information and Software Technology*, 55(6):1101–1118, 2013.

[24] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Prediction of fault-proneness at early phase in object-oriented development. In *Object-Oriented Real-Time Distributed Computing, 1999.(ISORC'99) Proceedings. 2nd IEEE International Symposium on*, pages 253–258. IEEE, 1999.

[25] Nachiappan Nagappan and Thomas Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 580–586. ACM, 2005.
[26] Huihua Lu, Bojan Cukic, and Mark Culp. A semi-supervised approach to software defect prediction. In *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pages 416–425. IEEE, 2014.
[27] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.
[28] Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.
[29] Richard O Duda, Peter E Hart, and David G Stork. Pattern classification and scene analysis 2nd ed. *ed: Wiley Interscience*, 1995.
[30] Karel Dejaeger, Thomas Verbraken, and Bart Baesens. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering*, 39(2):237–257, 2013.
[31] Usama Fayyad and Keki Irani. Multi-interval discretization of continuous-valued attributes for classification learning. 1993.