Software defect detection based on selected complexity metrics using fuzzy association rule mining and defective module oversampling

Mohammad Farid Naufal
Department of Informatics
Universitas Surabaya
Surabaya, Indonesia
faridnaufal@staff.ubaya.ac.id

Selvia Ferdiana Kusuma
Department of Informatic
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
selvia1805@mhs.its.ac.id

***Abstract*—Software defect is a major problem in software development. The cost of software development will be minimized when the software defects are detected earlier. Complexity metric is a mathematic calculation to calculate code complexity. It could be used to consider software defect detection. But, not all of complexity metrics influent on the occurrence of software defect, therefore it needs feature selection to select the most influent complexity metrics. Correlation-based Feature Selection (CFS) is used for selecting the most influent complexity metrics. This study conducted experiments on NASA Metric Data Program (MDP) datasets. NASA MDP contains software defect history logs based on several complexity metrics. But, there is an imbalanced distribution of defective and not defective modules in NASA MDP. The distribution of defective modules is less than not defective modules. It can reduce software defect detection performance. The distribution of defective module need to be reproduced. In this study, Synthetic Minority Oversampling Technique (SMOTE) is used to balance the distribution between defective and not defective modules. Software defect detection using Fuzzy Association Rule Mining (FARM) which is combined with the selection of complexity metrics using CFS and dataset balancing using SMOTE has sensitivity 85.51% and accuracy 91.63% in detecting software defective modules on NASA MDP dataset.**

***Index Terms*— Software Defects, Fuzzy Association Rule Mining, CFS, SMOTE, Complexity Metric**

I. Introduction

Developing good quality software is an expensive task. Minimizing the occurrence of software defect can also minimize the software development cost. Software defect detection is useful for software engineer to pay attention to defective modules (e.g. function, method, or classes) [1].

NASA MDP [2] are open source datasets which shows the occurrence of software defect module with its following software complexity metrics. NASA has publicly accessible datasets which is easy to compare this study with the previous researches which use the same dataset.

Complexity metric selection as attributes or features in software defect detection still be an issue today. Not all of complexity metric influent the occurrence of software defect [3]. It needs a method to select the most influential complexity metrics. CFS is used to analyze the attributes relation in classification [4]. It can be applied to select the most influential complexity metric in software defect detection.

The software defect distribution in training dataset is also important for detecting software defect. NASA MDP consist of two type distributions which are defective and not defective modules. If one of the distributions is more than the other, it causes the class imbalance. This situation can affect on software defect detection performance [5]. The majority class is the class which has higher distribution and the minority class is a class which has fewer distribution. Undersampling or oversampling technique can be used to solve this problem [6]. Undersampling is used for reducing the majority class and oversampling is used for increasing the minority class.

Defective modules distribution is less than not defective modules in NASA MDP dataset. It requires oversampling technique to increase defective modules distribution. SMOTE [7] is a technique to oversample the minority class. In this study SMOTE performs data oversampling on defective modules.

FARM is association rule mining extension with excess faster and more efficient in large quantitative dataset [8]. FARM generate the pattern of defective modules which is useful for detecting defective modules with complexity metric as attributes. FARM can be expected as a good approach to detect defective modules. This study proposes detection of software defect by applying FARM combined with attributes selection in complexity metric by applying CFS and defective modules oversampling using SMOTE.

II. Background

A. Related Work

Burak [8] uses Branch Count, McCabe, and Halstead metric as features for software defect detection. CM1, JM1, KC1, KC2, and PC1 are NASA MDP dataset which are used in this study. This method is not considering the most influent complexity metrics to detect software defect but has fast computing time.

Ruchika [9] uses Support Vector Machine (SVM) to detect software defect on one CMI dataset in NASA MDP. This method uses Chidamber & Kemerer (CK) Metrics [10] as features. But this study only tried on one dataset. Gabriela using relational association rule to detect software defect [11]. It uses CK metrics (Fan-in, Fan-Out, DIT, and NOC).

978-1-7281-0719-6/19/$31.00 ©2019 IEEE

TABLE I. COMPLEXITY METRICS IN NASA MDP

Symbol	Metric
loc	line count of code
v(g)	cyclomatic complexity
ev(g)	essential complexity
Iv(g)	design complexity
n	total operators + operands
v(g)	Volume
L	program length
…	…

TABLE II. THE DESCRIPTION OF NASA MDP DATASET [2]

Dataset	Language	Description
CM1	C	Instrument of NASA spacecraft
JM1	C	Real-time predictor of ground system: Generate predictions using simulation.
KC1	C++	Management of storage for processing and receiving ground data
KC2	C++	Another part of KC1 project. Data processing of science.
PC1	C	Flight application for earth orbiting satellite

The weakness of this method is representing complexity metric. It converts complexity metric into Boolean relational association rule.

Omer Faruk [12] combine Artificial Bee Colony (ABC) and Artificial Neural Network (ANN) to detect software defect. Branch Count, McCabe, and Halstead metric are used as features. The NASA MDP datasets which are used in this study are CM1, JM1, KC1, KC2, and PC1. This method is easy to implement in quantitative complexity metrics values. But it's performance is not significant while compared with other software defect detection method.

B. Complexity Metrics

NASA MDP provides the occurrence of software defect with related complexity metrics. These complexity metrics can be used as features for software defect detection. Branch Count, Halstead metric [13], and McCabe metrics [14] are widely used software metric which represent the code complexity. Table 1 shows list of some complexity metrics.

C. Correlation-based Features Selection

A useful feature is a feature which is correlated with predicted class, otherwise it is irrelevant feature [4]. Correlation-based feature selection select the most influent features by considering the correlation between features-features and features-predicted class. The calculation of CFS evaluation is shown in Eq. (1).

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \qquad (1)$$

M_S is a Merit or heuristic value of S which containing k features. $\overline{r_{cf}}$ is value of average correlation between predicted classes and features (f ∈ S). $\overline{r_{ff}}$ is value of average correlation between features and other features. Rank of the most influent metric to predict class is shown in Eq. (1). Value of $\overline{r_{cf}}$ and $\overline{r_{ff}}$ are calculated by using the Symmetrical Uncertainty (*SU*). The equation of *SU* is shown in Eq. (2).

$$SU = 2.0\ x\ \left[\frac{H(X)-H(Y|X)}{H(Y)+H(X)}\right] \qquad (2)$$

The entropy H(X) and H(Y) are the possible emergence of an attribute (x ∈ X) or (y ∈ Y) in a transaction. The calculation of H(X), H(Y), and H(X|Y) are shown in Eq. (3), (4), and (5).

$$H(Y) = -\sum_{(y \in Y)} p(y) \log_2(p(y)) \qquad (3)$$

$$H(X) = -\sum_{(x \in X)} p(x) \log_2(p(x)) \qquad (4)$$

$$H(X|Y) = -\sum_{(x \in X)} p(x) \sum_{(y \in Y)} p(y|x) \log_2(p(y|x)) \qquad (5)$$

D. Synthetic Minority Oversampling Technique

SMOTE solve the class imbalanced problem by reproducing or oversampling of the minority class distribution [7]. Oversampling is the way to create synthetic data by merging k nearest neighbors of minority class samples. K-nearest neighbors is randomly selected depend on the amount of desired synthetic data. Synthetic data in software defective modules is formed in the following way:

a. Select one of defective module randomly.
b. Take the k-nearest neighbors of that defective module based on its complexity metric value.
c. Randomly select n of selected defective modules.
d. Calculate the differences value between selected defective module's complexity metric with its N nearest neighbors.
e. Multiply a random value from [0,1] with previous differences value.
f. Add the selected defective module's complexity metric with the multiplied differences from step e. Eq. (6) is the calculation to form oversampled defective module. *N, a,* and *b* are vector of defective module which consist of complexity metrics m. *N* is oversampled dataset vector *N* (m_{n1}, m_{n2}, m_{n3}, …), *a* is selected defective module vector a (m_{a1}, m_{a2}, m_{a3}, …), *r* is random value [0,1], and *b* is one of nearest neighbor vector of *a, b* (m_{b1}, m_{b2}, m_{b3}, …).

$$N = a + r(a - b) \qquad (6)$$

III. Fuzzy Association Rule Mining

FARM [15] is Association Rule Mining (ARM) expansion. ARM represent the appearance of item in transaction as a Boolean value. In FARM represent the data as a membership function between 0 and 1. FARM is a good method for detecting software defect which has quantitative complexity metric attributes.

TABLE III. SOFTWARE DEFECT HISTORY LOGS IN NASA MDP DATASET

ID	loc	v(g)	ev(g)	iv(g)	n	other metrics	defect status
1	24	5	1	3	63	…	FALSE
2	20	4	4	2	47	…	FALSE
3	31	4	1	2	141	…	TRUE
4	29	5	1	3	111	…	TRUE

$$Supp(A,B) = \frac{All\ transactions\ contain\ A\ and\ B}{All\ transactions} \quad (7)$$

$$Conf(A,B) = \frac{All\ transactions\ contain\ A\ dan\ B}{All\ transactions\ contains\ A} \quad (8)$$

There are several methods for finding association rules in fuzzy sets. F-APACS [16] describes quantitative attributes into fuzzy sets by using a linguistic term. Apriori [17] performs fuzzification of quantitative attributes then categorize them into membership functions. Finally, association rules are generated using fuzzy sets. But, Apriori efficiency is not good enough. Han [18] propose Frequent-Pattern growth (FP-Growth) as a new mining method to improve Apriori efficiency. This method more efficient because it does not generate all item set candidates. FP-Growth categorize every transaction into Frequent Pattern tree (FP-Tree). Then it calculates the support and confidence for every pattern. Eq. (7) and Eq. (8) show support and confidence calculation.

IV. METHODOLOGY

The detail of this study approaches will be described in this section.

A. Collecting Dataset

NASA MDP dataset are used for evaluating our method. NASA MDP datasets contain several software defect history logs based on Branch Count, McCabe, and Halstead metric. This dataset publicly accessible and used in many previous researches.

This study uses 5 NASA MDP datasets which are CM1, JM1, KC1, KC2, and PC1. Table 2 shows the description of each dataset and Table 3 shows the software defect history logs in NASA MDP dataset. Every row is a module, it could be a method or a function and every column is a complexity metric calculation for each module. The rightmost column is a defective status for each module. For example, at the first row is a method with ID 1, it has 24 line of codes (loc), 5 cyclomatic complexity (v(g)), 1 essential complexity (ev(g)), 3 design complexity (iv(g)), 63 total operators and operands (n), and defective status of that module is false. Another example at the third row is a method with id 3, it has 31 line of codes (loc), 4 cyclomatic complexity (v(g)), 1 essential complexity (ev(g)), 2 design complexity (iv(g)), 141 total operators and operands (n), and defective status of that module is true.

B. Preprocessing

In the preprocessing step, CFS and SMOTE are used to select the most influent complexity metric and oversample the training dataset. From 21 complexity metrics on NASA MDP dataset, CFS will select the most influent metric for detecting defective modules on each dataset. Good complexity metric combinations selection is based on merit value on Eq. 1. The best complexity metric combination has the highest merit value. CFS and SMOTE are executed using WEKA tools [19]. SMOTE will reproduce the defective modules on training dataset. This phase has been completed after selection of complexity metric using CFS. The example of oversampling process are as follows:

TABLE IV RANDOMLY SELECTED DEFECTIVE MODULE

ID	ev(g)	b	IOComment
1	32	2	0.43

TABLE V RANDOMLY 5 NEAREST NEIGHBORS

ID	ev(g)	b	IOComment
1	41	6	0.33
2	**23**	**5**	**0.08**
3	**38**	**5**	**0.35**
…	**…**	**…**	**…**

TABLE VI. NEW DEFECTIVE MODULE FROM OVERSAMPLING PROCESS

ID	ev(g)	b	IOComment
N2	33.8	1.4	0.5
N5	30.2	1.1	0.454

a. Table 4 shows one of randomly selected defective module which will be oversampled.
b. Select the k nearest neighbors of Table 4 randomly. Table 5 shows the 5 randomly nearest neighbors of Table 4 and ID 2 and 5 are selected nearest neighbors.
c. Calculate oversampled defective module using Eq (6) as follows:
   $N2 = (32, 2, 0.43) + 0.2\ x\ [(32,2,0.43) - (23,5,0.08)]$
   $N5 = (32, 2, 0.43) + 0.3\ x\ [(32,2,0.43) - (38,5,0.35)]$
   N2 and *N5* are new defective modules from oversampling process of selected defective module on Table 4. Table 6 shows the new defective module of N1 and N2.

C. Training

In this step, software defect pattern will be obtained. FP Growth is used for conducting training step. Then FP-Tree is obtained based on complexity metrics. The detail of training steps are as follows:

1) Determining Fuzzy Membership Function

Each complexity metric of defective modules will be categorized its fuzzy membership function value. The kind of fuzzy membership functions are trapezoidal, triangular, gaussian function. This study uses triangular fuzzy membership function because it can more precisely write the more and less value [20] in this case complexity metric. The membership functions are "low", "medium", and "high". The x-axis is complexity metric value and y-axis is membership value of complexity metric to each category.

The limit value of a, b, and c for every complexity metric in training dataset defines as:

$$y = \text{complexity metric value} \quad (8)$$

a = the minimum value of y (9)
b = y mean value (10)
c = 2 x y mean value (11)

2) Mapping Complexity Metric to Fuzzy Membership

In this step the membership function of complexity metric μ(x) will be calculated in each category (low, medium, and high). In Table 7, the complexity metric has been converted into three categories (low, medium, and high) fuzzy membership. For example, at the first row, a module with ID 1 has low of ev(g) and low medium of b and low medium of IOComment. b and IOComment are metric which have two fuzzy memberships. Low medium means that complexity metric has two fuzzy memberships which are low and medium.

3) Identify Defective Patterns

The defective pattern will be identified by using FP Growth. The next step is calculating each defective pattern confidence. Then this pattern will be used for detecting defective modules. Table 8 shows the defective pattern based on complexity metric which are generated from training step. For example at the first row the defective pattern with ID 1 is ev(g).Low, ev(g).Med, b.High, lOComment.High and has confidence value 1. It means the module which has low medium of ev(g) and high of b and high of IOComment has 100% probability to be a defective module. Another example at the third row the defective pattern with ID 3 is ev(g).Low, ev(g).Med, b.Med, b.High, lOComment.Med, lOComment.High and has confidence value 0.875. It means the module which has low medium of ev(g) and medium high of b, and medium high of IOComment has 87.5% probability to be a defective module.

D. Defective Modules Detection

The generated defective modules patterns from training step will be used to detect defect on testing dataset. A module is detected as defective module if it is matched with one of defective modules pattern and vice versa.

E. Performance Evaluation

True Positive (TP) is the defective module number which successfully detected as defect, True Negative (TN) is the not defective module number which successfully detected as not defect, False Positive (FP) is the not defective module number which classified as defect, False Negative (FN) is the defective modules number which classified as defect. Accuracy, sensitivity, specificity, precision, nad probability of false alarm are used as performance evaluation.

F. Results

This study has been done on five NASA MDP datasets. Testing scenario has been done without oversampling (FARM-CFS) and with oversampling process (FARM-CFS-SMOTE). This scenario is done to compare the effect of oversampling on defect detection performance. This proposed approach use weka to run CFS and SMOTE.

We are dividing the dataset randomly into two parts, 50% for training and 50% for testing. The reason is we assume in real software project, the model of defect detection is formed in the middle of software construction. We also compare the accuracy, sensitivity, precision, specificity and probability of false alarm (pof) of this detection approach with another detection method using Naïve Bayes (NB), Support Vector Machine (SVM), Relational Association Rule (RAR), Artificial Neural Network + Artificial Bee Colony (ANN+ABC).

Table 9 shows the selected metric of each dataset. From 21 complexity metrics in Table 1, CFS select only 3 complexity metrics on CM1, 7 on JM1, 8 on KC1, 8 on KC2, and 3 on PC1. It can be seen that IOComment is the most influent metric in all dataset, followed by locCodeAndComment and ev(g) which are influent metric in 3 datasets, IOBlank, uniqOp, iv(g), loc, b, and I are influent metric at least in 2 datasets.

The result of this approach can be clearly seen on Table 10. Table 10 shows the performance average of FARM-CFS and FARM-CFS-SMOTE at every confidence threshold value.

Accuracy comparison between FARM-CFS and FARM-CFS-SMOTE is not significant but when the confidence threshold above 0.3 FARM-CFS-SMOTE is better than FARM-CFS. This indicate that when confidence threshold is above 0.3, the defective pattern of FARM-CFS-SMOTE can detect more defective pattern as a defect than detecting not defective pattern as defect. In other words FARM-CFS-SMOTE has more true positive and true negative value than FARM-CFS.

Specificity comparison between FARM-CFS and FARM-CFS-SMOTE are significant when the confidence value under 0.3, it is caused by true negative value of FARM-CFS is always better than FARM-CFS-SMOTE. Moreover defective pattern which formed by oversampling process is not always accurate to detect defective modules. In another word FARM-CFS-SMOTE has more false positive value than FARM-CFS.

Sensitivity comparison between FARM-CFS and FARM-CFS-SMOTE are significant. FARM-CFS-SMOTE sensitivity is better than FARM-CFS at every confidence threshold. It caused by oversampling process which reproduce more defective pattern can detect more defective module. In other word, FARM-CFS-SMOTE has more true positive value.

Precision comparison between FARM-CFS and FARM-CFS-SMOTE are significant when confidence threshold above 0.6. It is caused by true positive value of FARM-CFS-SMOTE more than FARM-CFS compared by the false positive value.

Probability of false alarm comparison between FARM-CFS and FARM-CFS-SMOTE are not significant but FARM-CFS is better at every confidence threshold. It is caused by defective pattern which is formed by oversampling process sometimes detect not defective modules as defect.

The comparison of these two scenarios can be clearly seen on Table 10. It shows the comparison of the best value of each performance evaluation at each confidence threshold. FARM-CFS has better specificity, precision, and probability of false alarm than FARM-CFS-SMOTE by a margin of each are 9.35%, 1.14%, and 1.32%. However, the imbalance of defective and not defective modules in the dataset make detecting the defective modules is more important. FARM-CFS-SMOTE which has a higher sensitivity value still better assessed because it more able to detect more defective modules. In other words FARM-CFS-SMOTE has more true positive value than FARM-CFS.

Table 11 shows the performance comparison between this approach with NB, SVM, RAR, and ANN+ABC. FARM-CFS-SMOTE has the highest sensitivity than the NB, SVM, RAR, ANN + ABC, and FARM-CFS. Although the

TABLE VII. THE COMPLEXITY METRIC HAS BEEN CONVERTED INTO THREE FUZZY MEMBERSHIP CATEGORIES

ID	ev(g)			b			IOComment			Defect Status
	Low	Medium	High	Low	Medium	High	Low	Medium	High	
1	1	0	0	1	1	0	1	1	0	FALSE
2	1	0	0	1	1	0	1	0	0	FALSE
3	1	0	0	1	1	0	1	0	0	FALSE

TABLE VIII. THE DEFECTIVE PATTERN BASED ON COMPLEXITY METRIC

ID	Defective Pattern	Confidence
1	ev(g).Low, ev(g).Med, b.High, lOComment.High	1
2	ev(g).Low, ev(g).Med, b.High, lOComment.Med, lOComment.High	1
3	ev(g).Low, ev(g).Med, b.Low, b.Med, lOComment.Low, lOComment.Med	0.844

TABLE IX. SELECTED COMPLEXITY METRIC OF NASA MDP DATASET

CM1	JM1	KC1	KC2	PC1
ev(g)	loc	ev(g)	IOComment	i
b	v(g)	iv(g)	IOCodeAndComment	IOComment
IOComment	ev(g)	v(g)	IOBlank	IocCodeAndComment
	iv(g)	IOCode	uniq_Op	loc
	e	IOComment	loc	ev(g)
	IOComment	IOBlank	ev(g)	i
	locCodeAndComment	uniq_Op	i	b
		total_Opnd	b	

TABLE X. PERFORMANCE COMPARISON BETWEEN FARM-CFS AND FARM-CFS-SMOTE

Confidence	FARM-CFS					FARM-CFS-SMOTE				
	Accuracy	Specificity	Sensitivity	Precision	Pof	Accuracy	Specificity	Sensitivity	Precision	Pof
0.1	87.96%	90.15%	**69.56%**	55.61%	9.82%	84.08%	83.61%	**85.51%**	48.57%	16.36%
0.2	88.15%	95.31%	54.18%	71.83%	4.66%	85.74%	86.27%	82.31%	51.91%	13.70%
0.3	87.57%	95.64%	47.29%	72.65%	4.34%	90.18%	92.98%	70.74%	63.36%	6.99%
0.4	89.00%	96.17%	38.20%	**75.86%**	3.81%	89.94%	94.07%	62.87%	65.56%	5.90%
0.5	88.24%	96.55%	30.96%	73.03%	3.42%	**90.41%**	96.03%	54.62%	72.54%	3.95%
0.6	89.11%	99.09%	21.77%	46.66%	0.90%	89.95%	**96.57%**	45.54%	74.06%	3.41%
0.7	**89.13%**	99.73%	19.08%	54.94%	0.26%	89.73%	96.57%	42.40%	**74.72%**	3.21%
0.8	88.41%	99.76%	15.46%	55.25%	0.24%	89.78%	97.37%	39.47%	75.93%	2.60%
0.9	88.40%	**99.76%**	15.39%	55.23%	**0.24%**	88.90%	98.41%	26.24%	72.35%	**1.56%**

accuracy, specificity, precision, and probability of false alarms FARM-CFS-SMOTE is not the best, but sensitivity is the priority in defect detection. The higher value of sensitivity, it is better in detecting defective modules that have lower distribution in the dataset. The imbalance distribution of defective and not defective modules makes detection of defective modules which have lower distribution becomes more essential.

G. Conclusion

From the results, it can be concluded that the addition of oversampling process on defective modules can significantly increase the sensitivity up to 15.96% compared with no oversampling process. Although the oversampling process does not improve accuracy significantly, it is only 1.18% compared without oversampling process.

FARM-CFS-SMOTE has the best sensitivity compared with SVM, NB, RAR, ANN+ABC, and FARM-CFS. It means that FARM-CFS-SMOTE can detect more defective modules than the other methods. The lower distribution of defective modules compared with not defective modules make detection of defective module is more essential. The defective pattern which generated from oversampling process can detect the defective modules correctly. Although several not defective modules are classified as defect (false positive) but the value is not significant if compared with defective modules which are classified as defect (true positive). Software defect detection performed in this study is based on complexity metrics McCabe, Halstead, and Branch Count at NASA MDP dataset.

TABLE XI. PERFORMANCE COMPARISON WITH ANOTHER CLASSIFICATION METHOD

Defect Detection Method	Accuracy	Specificity	Sensitivity	Precision	Pof
NB	87.95%	95.17%	50.10%	71.86%	4.84%
SVM	**92.48%**	**99.95%**	54.36%	**99.24%**	**0.04%**
RAR	87.69%	89.33%	82.80%	69.83%	10.70%
ANN+ABC	69.60%	56.41%	76.60%	22.55%	29.80%
FARM-CFS	91.09%	99.76%	69.56%	93.54%	0.24%
FARM-CFS-SMOTE	91.63%	98.41%	**85.51%**	80.35%	1.56%

H. Future Work

In the future software defect detection can be developed and applied to another software development project. Furthermore, determining the confidence threshold value can be detailed in smaller units to get the better performance.

V. REFERENCES

[1] P. He and B. Li, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology,* vol. 59, pp. 170-190, 2015.

[2] NASA, "PROMISE," University of Ottawa, 2006. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html. [Accessed 3 May 2015].

[3] V. T. A. S. Ishani Arora, "Open Issues in Software Defect Prediction," *Procedia Computer Science,* pp. 906-912, 2015.

[4] M. A. Hall, "Correlation-based feature selection for machine learning," 1998.

[5] M. Galar, A. Fernández, E. Barrenechea, H. Bustince and F. Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions,* pp. 463-484, 2012.

[6] R. Barandela, R. Valdovinos, J. Sánchez and F. Ferri, "The imbalanced training sample problem: Under or over sampling?," *Structural, Syntactic, and Statistical Pattern Recognition,* pp. 806-814, 2004.

[7] Chawla, N. V. a. Bowyer, K. W. a. Hall, L. O. a. Kegelmeyer and W. Philip, "SMOTE: Synthetic Minority Over-sampling Technique," *CoRR,* pp. 321--357, 2002.

[8] B. Turhan and A. Bener, "Analysis of Naive Bayes assumptions on software fault data: An empirical study," *Data & Knowledge Engineering,* pp. 278-290, 2009.

[9] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing,* vol. 27, p. 504–518, 2015.

[10] M. Hitz and B. Montazeri, "Chidamber and Kemerer's metrics suite: a measurement theory perspective," in *IEEE Transactions on Software Engineering*, 1996.

[11] G. Czibula, "Software defect prediction using relational association rule," *Information Sciences,* pp. 260-278, 2014.

[12] N. Omar, S. S. Haris, R. Hassan, H. Arshad , M. Rahmat, N. F. A. Zainal and R. Zulkifli, "Automated Analysis of Exam Questions According to Bloom's Taxonomy," *Science Direct,* pp. 297-303, 2012.

[13] M. H. Halstead, "Elements of Software Science (Operating and Programming Systems Series," 1977.

[14] T. McCabe, "A Complexity Measure," *Software Engineering, IEEE Transaction,* Vols. SE-2, pp. 308-320, 1976.

[15] C.-H. Wang and W.-H. Lee, "Applying Fuzzy FP-Growth to Mine Fuzzy Association Rules," 2010.

[16] W. Au and K. Chan, "An effective algorithm for discovering fuzzy rules in relational databases," in *IEEE World Congress on Computational Intelligence*, 1998.

[17] A. Inokuchi, T. Washio and H. Motoda, "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data," in *Principles of Data Mining and Knowledge Discovery*, 2002.

[18] H. Jiawei, P. Jian and Y. Yiwe, "Mining Frequent Patterns Without Candidate Generation," *SIGMOD Rec.,* vol. 29, no. 2, pp. 1-12, 2000.

[19] Hall, M. a. Frank, E. a. Holmes, G. a. Pfahringer, B. a. Reutemann, P. a. Witten and I. H, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.,* vol. 11, pp. 10-18, 2009.

[20] A. K. Shyamal and M. Pal, "Triangular Fuzzy Matrices," *Iranian Journal of Fuzzy Systems ,* vol. 4, no. 1, pp. 75-87, 2007.

2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE 2019)

Chonburi, Thailand
10 – 12 July 2019

IEEE

IEEE Catalog Number: CFP1932P-POD
ISBN: 978-1-7281-0720-2

****** This is a print representation of what appears in the IEEE Digital Library. Some format issues inherent in the e-media version may also appear in this print version.***

IEEE Catalog Number:	CFP1932P-POD
ISBN (Print-On-Demand):	978-1-7281-0720-2
ISBN (Online):	978-1-7281-0719-6
ISSN:	2372-1642

Additional Copies of This Publication Are Available From:

Curran Associates, Inc
57 Morehouse Lane
Red Hook, NY 12571 USA
Phone: (845) 758-0400
Fax: (845) 758-2633
E-mail: curran@proceedings.com
Web: www.proceedings.com