Journal of Telematics and Informatics (JTI)
Vol.7, No.2, June 2019, pp. 60~68
ISSN: 2303-3703

Software Defect Prediction Using Synthetic Minority Over-sampling Technique and Extreme Learning Machine

Khadijah, Priyo Sidik Sasongko
Department of Computer Science/ Informatics, Universitas Diponegoro
Semarang, Indonesia
e-mail: khadijah@ive.undip.ac.id

Abstract

Software testing is one of the crucial processes in software development life cycle which will influence the software quality. One of the strategies to help testing process is predicting the part or module of software which is prone to defect. Then, the testing process can be more focused to those parts. In this research a classifier model for predicting software defect was built. One of the most important problems in software defect prediction is imbalance data distribution between samples of positive class (prone to defect) and of negative class. Therefore, in this research SMOTE is implemented to handle imbalance data problem and extreme learning machine is implemented as a classification algorithm. As a comparison to SMOTE-ELM, a modification of ELM which directly copes with imbalance problem, weighted-ELM, is also observed. This research used NASA MDP dataset PC1, PC2, PC3 and PC4. The results of experiment using 10-fold cross validation show that directly classification using ELM obtain the worse result compared to SMOTE-ELM and weighted-ELM. When the value of imbalance ratio is not very small, the SMOTE-ELM is better than weighted-ELM. When the value of imbalance ratio is very small, the g-mean of weighted-ELM is higher than the g-mean of SMOTE-ELM, but the accuracy of weighted-ELM is lower than the accuracy of SMOTE-ELM. Therefore, in this software defect prediction case it can be concluded that SMOTE is effective to increase the generalization performance of classifier in minority class as long as the value of imbalance ratio is not very small.

Keywords*: extreme learning machine (ELM), software defect prediction, SMOTE, weighted-ELM*

1. Introduction

Software testing is one of the crucial processes in software development life cycle. Software testing is aimed to find errors of software before it is deployed to end user [1]. The results of software testing will influence the software quality. Incomplete software testing can result in bad software quality. Comprehensive software testing activities need big resources in term of cost, human and time, especially if the developed software is complex and large scale [2].

One of the strategies to do testing process effeciently is by predicting the part or module of software which is prone to defect. Then, the testing process can be more focused to those parts. One of the approaches for predicting software defect is based on software metrics which describe the properties of software [3]. Therefore, an automated software defect prediction model can be built based on those software metrics. In term of machine learning, such kind of model can be built by using supervised learning or classification algorithm.

The various classification algorithm have been implemented for predicting software defect, such as Artificial Neural Network (ANN) [4] [5], Support Vector Machine (SVM) [6] and Naïve-Bayes classifier [7]. Naïve-Bayes classifier assumes that every attribute (software metric) is conditional independence to the class of software defect [8], while in the case of software

defect prediction these attributes is not truly conditional independence to the class of software defect. When using ANN, gradient-descent learning algorithm is usually implemented. This learning algorithm usually needs many iteration to reach convergency and involves many parameter in its training process. The other learning algorithm for ANN that can solve the limitation of gradient-descent learning is extreme learning machine (ELM). The training process of ELM is extremely fast because it only needs one iteration. ELM also involves less number of parameter in its training process [9]. ELM is also faster and more scalable than SVM. Moreover in binary classification case the generalization performance of ELM is better than ANN and similar to SVM [10].

In spite of classification algorithm, the other important problem in software defect prediction is the imbalance ratio of number of samples between possitive class (prone to defect) and negative class. The value of imbalance ratio can reach 1/100, even 1/1000 in certain dataset. Classifications on imbalanced dataset usually result in good accuracy, but very low sensitivity. For example, when number of samples of positive class is 10 and number of negative class is 90 and classifier always classifies each sample into negative class, then the classifier can achieve 90% accuracy. However, it can not be claimed that it is a good classifier because it never classifies a sample into possitive class, as the positive class is the topic of interest. Therefore, it is needed an additional method for handling imbalanced data distribution in order to get optimal results [11].

One of the approaches for handling imbalanced data distribution is resampling (data-level method). Resampling method modifies the distribution of dataset to achieve balance ratio between number of positive class and negative class. The advantages of this method is its flexibility because it is independence to the classification algorithm [12]. Synthetic Minority Over-sampling Technique (SMOTE) is one of the popular resampling methods and has been frequently used in many cases [13]. SMOTE even still has good performance when the number of samples in minority class is quite small [11]. Some implementation of SMOTE on classification with imbalanced dataset has proven that SMOTE is able to improve the generalization performance of classifier in minority class [14] [15] [16].

As a comparison to SMOTE, the other approach for handling imbalance dataset, algorithm-level method, is also implemented. This method does not modify the dataset, but modifies the classification algorithm directly [12]. This research implements weighted-ELM which is the modification of original ELM to cope directly with the imbalanced data problem. Experiment results show that weighted-ELM has better performance than original ELM as classifier on imbalanced dataset [17].

This research is aimed to create a model for predicting software defect using SMOTE as a resampling method to achieve balanced data distribution and ELM as a classifier. Then, as a comparison to SMOTE, a model for software defect prediction is also built using special modification of ELM for imbalanced data distribution, weighted-ELM. The resulting models are then evaluated to measure their performance.

2. Research Method

There are two process involved to build a model for software defect prediction, the training and testing process. The training is aimed to create a model and testing is aimed to evaluate the resulting model. This section describes the dataset, methods (SMOTE and ELM) and evaluation metric used in this research.

Tabel 1. Description of Dataset

Dataset	d	N	Number of negative samples	Number of positive samples	Imbalance ratio
PC1	37	735	674	61	0.0905
PC2	36	1493	1477	16	0.0108
PC3	37	1099	961	138	0.1436
PC4	37	1380	1201	179	0.1490

2.1. Dataset

This research used dataset from NASA MDP dataset PC1, PC2, PC3 and PC4. Version of dataset used is the cleaned data [18]. Those dataset can be downloaded in https://github.com/klainfo/NASADefectDataset. The description of each dataset are shown at Table 1 where d is the number of feature attributes and N is number of samples. The imbalance ratio is ratio between number of samples in positive class (prone to defect) and number of samples in negative class. Each dataset consists of some feature attributes and one decision attribute. The feature attributes are numeric software metrics that are used as an input feature for prediction, while the decision attribute is the output class which shows the existence of software defect (yes or no).

2.2. Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE is a resampling method to increase the number of samples in minority class by creating some synthetic samples. SMOTE creates a synthetic sample by choosing a point randomly in the line segment connecting a sample and its nearest neighbor. Figure 1 shows the pseudocode of SMOTE algorithm. There are two parameters in this algorithm. The first parameter is N, the amount of over-sampling. The second parameter is k, the number of nearest neighbor selected for creating synthetic samples. For example, if the number of samples in minority class is 10, the value of N is set to 200% and the value of k is set to 5, SMOTE will creates 2 synthetic samples for every samples in the minority class. Each time SMOTE creates a synthetic sample, SMOTE will choose a point randomly on the line connecting its sample and one of the five nearest neighbors of its sample. Therefore, the total number of synthetic sample is 2 x 10 = 20 and the total number of samples in minority class now is 10 + 20 = 30. This over-sampling is applied on training data, so that in the training process the ratio of samples in posstive class and negative is balanced or almost balanced. [13]

```
Function SMOTE(Sample[1..T][1..numattrs] :array of array of real, T : integer, numattrs: integer, N : integer, k: integer) → 
<array of array of real >
{This function is used for over-sampling minority class samples. The parameters of this function are:
Sample[][] is array original minority class samples
T is number of samples in minority class
numatrrs is the number of sample's attributes
N is percentage of over-sampling (N ≥ 100%)
k is number of nearest neighbors

Kamus lokal:
  i : integer {counter for original sample}
  nnarray : array of integer {array for storing index of sample's nearest neighbors}
  nn: integer {index of the choosen nearest neighbor of a sample}
  attr : integer {counter for sample's attributes}
  dif : real {the difference of attribute value between synthetic sample and original samples}
  gap : real {random value between 0..1 for multiplying dif}
  newindex : integer {counter for generated synthetic samples}
  Synthetic : array of array of real {array of generated synthetic samples}

Algoritma:
  {initialization}
  newindex ← 0

  i traversal [1..T]
        {obtain k nearest neighbor for i-th sample using Euclidean distance formula and store
        location index of its nearest negihbors into nnarrays}
        <nnarray> ← compute_nearest_neighbor(Sample[i][1..numattrs])

        {generate synthetic samples}
        N ← (integer)(N/100)

        while N ≠ 0
            {choose a sample from k nearest neighbors of i-th sample randomly }
            nn ← random(1,k) {nn is random value between  1..k}

            attr traversal [1..numattrs]
                {calculate the difference between attribute value of sample's nearest neighbor
                and its sample}
                dif ← Sample[nnarray[nn]][attr] - Sample[i][attr]
                gap ← random(0,1) {gap is random value 0..1}
                Synthetic[newindex][attr] = Sample[i][attr] + gap*dif
            {end traversal attr}

            newindex ← newindex + 1
            N ← N-1
        {endwhile N ≠ 0}
```

```
{end traversal i}

→ Synthetic[0..newindex-1][1..numattrs]
```

Figure 1. Pseudocode of SMOTE [13]

2.3. Extreme Learning Machine (ELM)

ELM is a learning algorithm for artificial neural network which has single hidden layer and feedforward architecture. The learning algorithm of ELM only needs one iteration, so that the training process is extremely fast [9] [10]. Figure 2 shows the architecture of artificial neural network used in this research. This network has d input nodes which is equal to number of feature attributes in each dataset, L hidden nodes dan one output node which represents the decision attribute or existence of defect. This network apply radial basis function (RBF) hidden node, so that parameter $(\mathbf{a}_j, b_j)_{j=1}^{L}$ from input nodes to hidden nodes are center vector $\mathbf{a}_j$ and scaling parameter b_j in the j-th hidden node. The weights from hidden nodes to output node is vector $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^T$ [19].

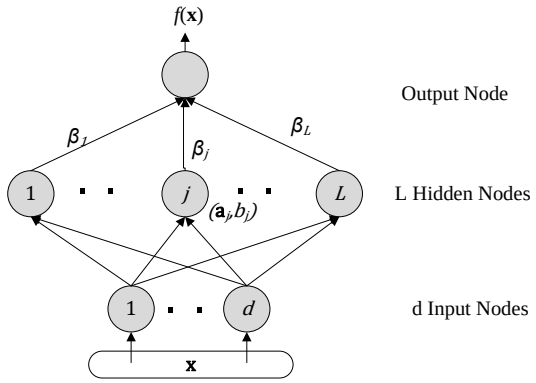

Figure 2. The Architecture of Network for ELM [19]

Training algorithm of ELM uses N pairs of training sample, $(\mathbf{x}_i, t_i)_{i=1}^{N}$ where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^T \in R^d$ is input vector and t_i is output target. The value of input and target are first normalized into [-1,1] using min-max normalization. Then the algorithm of the training is described as following [10]:

1. Initialize parameter $(\mathbf{a}_j, b_j)_{j=1}^{L}$ from each input node to $1..L$ hidden node randomly.
2. Calculate matrix $\mathbf{H}$, the output of each hidden node $1..L$ for all training samples from $1..N$, using a specific activation function as equation (1). This research used multiquadric activation function as shown in equation (2).

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \ldots & h_L(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & \ldots & h_L(\mathbf{x}_N) \end{bmatrix}_{N \times L} \quad (1)$$

$$h_j(\mathbf{x}_i) = (\|\mathbf{x} - \mathbf{a}_j\|^2 + {b_j}^2)^{1/2} \quad (2)$$

3. Calculate vector $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^T$, the weights from hidden nodes to output node, as equation (3). Output vector $\mathbf{T}$ as equation (4) is target value for each training samples from $1..N$, $\mathbf{I}$ is identity matrix, and C is regularization parameter to increase the generalization performance.

$$\boldsymbol{\beta} = \begin{Bmatrix} \mathbf{H}^{\mathrm{T}}\left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^{\mathbf{T}}\right)^{-1}\mathbf{T} & \text{if } N < L \\ \left(\frac{\mathbf{I}}{C} + \mathbf{H}^{\mathbf{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{T} & \text{if } N \gg L \end{Bmatrix} \tag{3}$$

$$\mathbf{T} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix} \tag{4}$$

In the testing process, class of an input sample $\mathbf{x}$ can be predicted by using equation (5) followed by equation (6). Equation (5) is aimed to calculate the value of output network using parameter $(\mathbf{a}_j, b_j)_{j=1}^{L}$ and $\boldsymbol{\beta}$ and also activation function from the training process, while equation (6) is aimed to predict the class of output $f(\mathbf{x})$. [10]

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} \tag{5}$$

$$class(\mathbf{x}) = \text{sign}(f(\mathbf{x})) \tag{6}$$

ELM algorithm is not especially designed for used in imbalanced dataset. Therefore, in the imbalanced dataset to maximize the accuracy, the boundary line of classifier usually is moved toward to the side of minority class. As a result, the generalization performance of classifier in the majority class is high, while in the minority class is low. But, the low performance in the minority class does not give significant effect on the end accuracy of classifier because the number of samples in the minority class is small enough. Therefore, ELM is modified into weighted-ELM which is especially designed for handling imbalanced data distribution. [17]

Weighted-ELM modifies the training algorithm of ELM by adding diagonal weight matrix $\mathbf{W} = diag\{w_{ii}\},\ i = 1, \dots, N$. Each row of matrix $\mathbf{W}$ is associated to each sample in the training data. The addition of this matrix is aimed to move boundary line toward to the area of majority class in order to increase the generalization performance in the minority class. But how much the boundary line is moved, should be carefully decided so that the generalization performance in the majority class does not decrease. Therefore, the value of w_{ii} is calculated using the golden ratio, 0,618:1 as shown at equation (7) where $\#(t_i)$ is the number of samples which is belong to the class of i-th sample. Then, the output weight matrix can be calculated by using equation (8). [17]

$$w_{ii} = \begin{cases} \dfrac{0.618}{\#(t_i)} & ,if\ t_i \text{ is majority class} \\ \dfrac{1}{\#(t_i)} & ,if\ t_i \text{ is minority class} \end{cases} \tag{8}$$

$$\boldsymbol{\beta} = \begin{Bmatrix} \mathbf{H}^{\mathrm{T}}\left(\frac{\mathbf{I}}{C} + \mathbf{W}\mathbf{H}\mathbf{H}^{\mathbf{T}}\right)^{-1}\mathbf{W}\mathbf{T} & \text{if } N < L \\ \left(\frac{\mathbf{I}}{C} + \mathbf{H}^{\mathbf{T}}\mathbf{W}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} & \text{if } N \gg L \end{Bmatrix} \tag{9}$$

2.4. Evaluation Metrics

Binary classifier can be evaluated based on its accuracy, sensitivity and specifity. Accuracy is the comparison between number of samples predicted correctly by classifer and the total number of samples. Sensitivity is the accuracy of positive class, while sensitivity is the accuracy of negative class [8]. The ratio between number of samples in positive class and negative class in the NASA MDP dataset is imbalanced, so that the use of accuracy as an evaluation metric is not quite representative. Therefore, in this research classifier is also evaluated by $g - mean$ as shown at equation (10). The value of $g - mean$ is not influenced by the balanceness of dataset because it considers the performance of classifier in both class, positive and negative [20].

$$G - mean = \sqrt{sensitivity \times specifity} \quad (10)$$

3. Results and Analysis

In order to get comparating result, in this research the classifier for predicting software defect was built by using three scenarios: 1) directly classification using ELM; 2) over-sampling minority class using SMOTE followed by classification using ELM; 3) directly classification using modification of ELM, weighted-ELM. This experiment applied 10-cross validation to divide dataset into training and testing data. First, dataset is divided into 10 parts where each part has the same ratio of positive samples and negative samples, then each experiment uses 9 part as training data and 1 part as testing data. The experiment is repeated until 10 times using different part as testing data, then the reported result is the mean of result of 10 experiments [21].

There are two parameters in the ELM training algorithm. They are number of hidden nodes L and regularization parameter C. In this research, number of hidden nodes is set to 1000. Some prior research used 1000 hidden nodes for various dataset and concluded that as long as the number of hidden nodes is quite large, it does not make significant effect to the end result [10]. Therefore, there is only one remaining parameter to be adjusted, regularization parameter C. The value of C is search from range $\{2^1, 2^2, \dots, 2^{24}, 2^{25}\}$ to obtain optimal result.

In the over-sampling using SMOTE, there are also two parameters, they are number of nearest neighbors k and percentage of over-sampling N. The value of N in each dataset is set into certain number to make the ratio of positive and negative samples in training data close to 1. Then, the value of k is set to 10 because in each dataset there are 6 to10 synthetic samples that would be created for each minority sample, except in the PC2 dataset the value k is set 12 because there are only 12 to 13 samples of positive class in the training data. Table 2 shows the combination of SMOTE parameters in four datasets.

Tabel 2. The Combination Value of SMOTE parameters

dataset	k	N
PC1	10	1000
PC2	12	1200
PC3	10	600
PC4	10	500

The results of experiment using ELM, SMOTE-ELM and weighted ELM for all dataset are shown at Table 3, 4 and 5, respectively. Then, the graphic at Figure 3 and 4 show the comparison of accuracy and g-mean, respectively.

Tabel 3. The Results of Experiment Using ELM

Dataset	C	Akurasi	Sensitivity	Specifity	G-mean
PC1	25	86,52 ± 3,5	42,62 ± 1,1	90,49 ±3,3	62,10
PC2	25	97,66 ± 1,1	35,00 ±41,2	98,38 ± 0,8	58,68
PC3	24	83,44 ±2,6	34,18 ± 8,2	90,53 ± 3,0	55,62
PC4	20	87,60 ± 2,5	45,49 ± 9,5	93,84 ± 3,1	65,34

Tabel 4. The Results of Experiment Using SMOTE-ELM

Dataset	C	Akurasi	Sensitivity	Specifity	G-mean
PC1	7	82,73 ± 6,5	74,05 ± 23,2	83,51 ± 6,9	78,64
PC2	9	98,26 ± 1,0	45 ± 43,78	98,92 ± 0,9	66,72
PC3	8	77,16 ± 4,9	72,42 ± 15,2	77,83 ± 4,6	75,08
PC4	9	84,19 ±3,5	82,65 ±11,6	84,44 ± 4,3	83,53

Tabel 5. The Results of Experiment Using Weighted-ELM

Dataset	C	Akurasi	Sensitivity	Specifity	G-mean
PC1	12	72,92 ± 6,3	86,67 ± 13,1	71,64 ± 7,1	78,80
PC2	4	76,36 ± 3,4	85,00 ± 24,2	76,31 ± 3,5	80,54

PC3	24	74,88 ±4,0	69,62 ± 10,6	75,65 ± 3,7	72,57
PC4	16	76,28 ± 3,7	93,30 ± 6,8	73,78 ± 4,3	82,97

The results on Table 3 show that directly classification using ELM without any handling for imbalanced data problem, tend to achieve the highest accuracy and the lowest g-mean, compared to the other methods (SMOTE-ELM and weighted-ELM). It is caused by the high value of specifity, but the low value of sensitivity, so that the the value of g-mean become low. However, because the number of positive samples is far smaller than the number of negative samples, especially in PC2 dataset, the value of accuracy is still high.

In the second scenario, when SMOTE is added for over-sampling dataset before classification, it can be observed that the value of g-mean and sensitivity in all dataset are higher compared to the first scenario, but the value of accuracy and specifity decrease, except on PC2 dataset. This condition show that when SMOTE is added, the generalization performance on positive class increase, but it is also followed by drecreasing generalization performance on negative class. It can be observed that the increasement values of g-mean are higher than the decreasement values of accuracy from ELM to SMOTE-ELM. The condition is slightly different on PC2 dataset, because PC2 has very low number of positive samples so that after oversampling the dataset is still imbalanced and the classifier still obtain bad generalization performance on positive (minority class) class.

Same as the second scenario, when using weighted-ELM the generalization performance on positive class (sensitivity) also increase, but it is also followed by drecreasing generalization performance on negative class (specifity). Compared to SMOTE-ELM, the values of g-mean are lower on PC3 and PC4 dataset, slightly higher on PC1 dataset, and higher on PC2 dataset. But, the accuracy of weighted-ELM are lower than accuracy of SMOTE-ELM in all dataset. On PC2 dataset, the value of imbalance ratio is very small so that in the learning algorithm of weighted-ELM the minority samples are given high value in the calculation of weight matrix **W** so that the boundary line is moved toward to the area of majority class. It causes the generalization performance of minority class increase, but the generalization of majority class decrease. Because the number of minority class is very small, the value of accuracy also decrease. Therefore, it can be concluded that in this prediction case SMOTE-ELM is better than weighted-ELM, especially when the imbalance ratio is not very small.

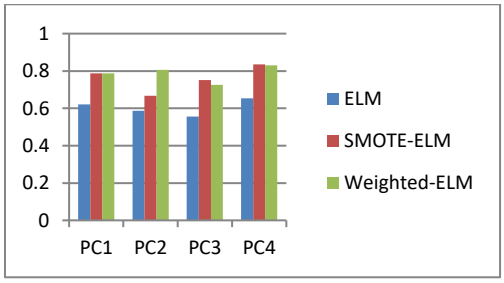

Figure 3. The Comparison of g-mean in All Dataset

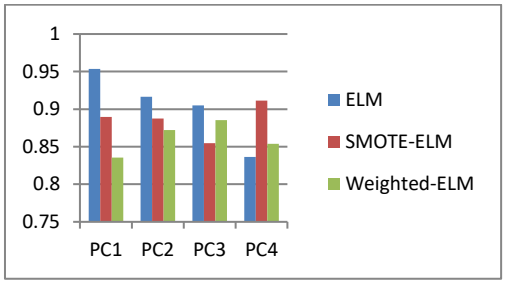

Figure 4. The Comparison of Accuracy in All Dataset

4. Conclusion

In this research the classifiers for software defect prediction was built. For building the classifier model, four dataset from NASA MDP were used, they are PC1, PC2, PC3 and PC4 dataset. One of the important problems in those dataset is the imbalance data distribution

between samples of positive class and samples of negative class. Therefore, in this research a method for handling this imbalance data problem was added. In order to get comparating result, the classifiers were built by using three scenarios: 1) directly classification using ELM; 2) over-sampling minority class using SMOTE followed by classification using ELM; 3) directly classification using modification of ELM, weighted-ELM.

The results of experiment using 10-fold cross validation show that directly classification using ELM obtain the worse result compared to the other scenarios. When the value of imbalance ratio is not very small the SMOTE-ELM is better than weighted-ELM because SMOTE-ELM can achieve higher value, both in g-mean and accuracy. When the value of imbalance ratio is very small, the g-mean of weighted-ELM is higher than the g-mean of SMOTE-ELM, the accuracy of weighted-ELM is lower than the accuracy of SMOTE-ELM. Therefore in this software defect prediction case, it can be concluded that SMOTE is effective to increase the generalization performance in minority class as long as the value of imbalance ratio is not very small.

The future research can be focused for selecting features or attributes that play signigicant role to the prediction class. The reduction of number of features as input to a classfier can also increase the performance of classifier.

References

[1] Pressman RS. y Software Engineering: A Practitioner's Approach Seventh Edition. New York: McGraw-Hill. 2009.

[2] Galin D. Software Quality: Concepts and Practice. Hoboken: IEEE Computer Society. 2018.

[3] Thi P, Phuong M, Thong PH. Empirical Study of Software Defect Prediction : A Systematic Mapping. *Symmetry*. 2019;11(212):1–28.

[4] Sethi T, Gagandeep. *Improved Approach for Software Defect Prediction using Artificial Neural Networks*. 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). Noida. 2016: 480–485.

[5] Irawan E, Wahono RS. Penggunaan Random Under Sampling untuk Penanganan Ketidakseimbangan Kelas pada Prediksi Cacat Software Berbasis Neural Network. *Journal of Software Engineering*. 2015;1(2):92–100.

[6] Rong X, Li F, Cui Z. A Model for Software Defect Prediction Using Support Vector Machine Based on CBA. *Intenational Journal of Intelligent Systems Technologies and Applications*. 2016;15(1):19–34.

[7] Putri SA, Wahono RS. Integrasi SMOTE dan Information Gain pada Naive Bayes untuk Prediksi Cacat Software. *Journal of Software Engineering*. 2015;1(2):86-91.

[8] Han J, Kamber M. Data Mining: Concepts and Techniques Second Edition. San Farnsisco: Elsevier Inc. 2006.

[9] Huang G, Zhu Q, Siew C. Extreme Learning Machine : Theory and Applications. *Neurocomputing*. 2006;70:489–501.

[10] Huang G-B, Zhou H, Ding X, Zhang R. Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*. 2012;42(2):513–28.

[11] Haixiang G, Yijing L, Shang J, Mingyun G, Yuanyue H, Bing G. Learning from Class-imbalanced Data: Review of Methods and Applications. *Expert System Applications*. 2017;73:220–39.

[12] Douzas G, Bacao F, Last F. Improving Imbalanced Learning Through a Heuristic Oversampling Method Based on K-means and SMOTE. *Infomation Sciences*. 2018;465:1–20.

[13] Chawla N, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*. 2002;16:321–57.

[14] Sarakit P, Theeramunkong T, Haruechaiyasak C. *Improving Emotion Classification in*

Imbalanced YouTube Dataset Using SMOTE Algorithm. 2nd International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA). Chonburi. 2015.

[15] Demidova L, Klyueva I. *SVM Classification : Optimization with the SMOTE Algorithm for the Class Imbalance Problem*. 6th Mediterranean Conference on Embedded Computing (MECO). Montenegro; 2017:17–20.

[16] Chang Z. *The Application of C45 Algorithm Based on SMOTE in Financial Distress Prediction Model*. 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC). Dengleng. 2011:5852–5855.

[17] Zong W, Huang G Bin, Chen Y. Weighted Extreme Learning Machine for Imbalance Learning. *Neurocomputing*. 2013;101:229–42.

[18] Shepperd M, Song Q, Sun Z, Mair C. Data Quality : Some Comments on the NASA Software Defect Datasets. I*EEE Transactions on Software Engineering*. 2013;39(9):1208–1215.

[19] Huang G. *Extreme Learning Machine - Learning Without Iterative Tuning*. Tutorial in IJCNN2012/WCCI2012. Brisbane. 2012.

[20] Timotius IK, Miaou SG. *Arithmetic Means of Accuracies: A Classifier Performance Measurement for Imbalanced Dataset*. International Conference on Audio, Language and Image Processing (ICALIP). Shanghai. 2010:1244–1251.

[21] Haykin S. Neural Networks - A Comprehensive Foundation. Second Edition. India: Pearson Education; 2005.