Software Defect Prediction in Imbalanced Data Sets Using Unbiased Support Vector Machine

Teerawit Choeikiwong, Peerapon Vateekul

Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University, Bangkok, Thailand

Teerawit.Ch@student.chula.ac.th, Peerapon.V@chula.ac.th

Abstract. In the software assurance process, it is crucial to prevent a program with defected modules to be published to users since it can save the maintenance cost and increase software quality and reliability. There were many prior attempts to automatically capture errors by employing conventional classification techniques, e.g., Decision Tree, k-NN, Naïve Bayes, etc. However, their detection performance was limited due to the imbalanced issue since the number of defected modules is very small comparing to that of non-defected modules. This paper aims to achieve high prediction rate by employing unbiased SVM called "R-SVM," our version of SVM tailored to domains with imbalanced classes. The experiment was conducted in the NASA Metric Data Program (MDP) data set. The result showed that our proposed system outperformed all of the major traditional approaches.

Keywords: software defect prediction; imbalanced issue; threshold adjustment

1 Introduction

Software defect is any flaw or imperfection in a software product or process. It is also referred to as a fault, bug, or error. It is considered as a major cause of failures to achieve the software quality, and the maintenance cost can be very high in order to amend all of the defects in the software production stage. Hence, it is very important to detect all of the bugs as early as possible before publishing the program. Software defect prediction is recognized as an important process in the field of software engineering. It is an attempt to automatically predict the possibility of having a defect in the software. Note that the detection can be in many levels, i.e., method, class, etc.

Data miners have been aware of the software defect issue and proposed several works in defect prediction applying conventional classification techniques[1, 2, 3, 4, 5, 6]. Although their experimental results were reported showing a promising detection performance, all of them were evaluated based on accuracy, which is not a proper metric in this domain. For example, assume the percentage of defected modules is only 10%, while the remaining modules (90%) are non-defected. Although the detection system incorrectly classifies all modules as non-defected ones, the accuracy is still 90%!

K.J. Kim (ed.), *Information Science and Applications,*
Lecture Notes in Electrical Engineering 339, DOI 10.1007/978-3-662-46578-3_110

For the sake of comparison, it is necessary to evaluate the system based on standard benchmarks. NASA Metrics Data Program (MDP) [7] is a publicly available and widely used data set. There are 12 projects described by many software metrics, such as McCabe's and Halstead's metrics. It is common in the software prediction benchmark, including MDP, that the number of defected modules is very small comparing to the non-defected ones. In the MDP benchmark, an average of the defected modules in all of the projects is 20%, and the PC2 project has the lowest percentage of defected modules with only 2.15%. This circumstance is referred to as "imbalanced data set," which is known to tremendously drop classification performance. There were many works [1, 2, 3, 4, 5, 6] using the MDP benchmark; however, most of them ignore the imbalanced issue, so their reported results should be impaired.

In this paper, we aim to propose a novel defect prediction system by applying a support vector machine technique called "R-SVM," our previous work [8] that is customary to induce SVM in the imbalanced training set. In R-SVM, the separation hyperplane is adjusted to reduce a bias from the majority class (non-defect). The experiment is conducted based on the MDP benchmark and, then, the result is compared to several classification techniques: Naive Bayes, Decision Tree, k-NN, SVM (Linear), and SVM (RBF) in terms of the measures including PD, PF, F1, and G-mean.

The rest of paper is organized as follows. Section 2 presents an overview of the related work. Section 3 describes the proposed method in details. The description of the data sets and tools are found in Section 4. Section 5 shows the experimental results. Finally, this paper is concluded in Section 6.

2 Related Work

2.1 Software Metrics

Measurement is known as a key element in engineering process. It can be used to control the quality and complexity of software. For building one product or system, we use many software measures to assess the quality of product and also to define the product attribute like in the MDP benchmark. These software metrics provide various benefits, and one of the major benefits is that it offers information for software defect prediction. Currently there are many metrics for defect prediction in software.

McCabe's cyclomatic complexity [9] and Halstead's theory [10] are ones of the well-known software metrics. They can be applied into any program languages and were introduced in order to describe as "code features," which are related to software quality. McCabe's metric is used to show the complexity of a program. From a program source code, it can directly measure the number of linearly independent paths. Moreover, it can be used as an ease of maintenance metric. Halstead's theory is widely used to measure complexity in a software program and the amount of difficulty involved in testing and debugging of software development. In addition, there are also other metrics, e.g., the Chidamber and Kemerer metrics [11] and Henry and Kafura's information flow complexity.

Table 1 shows the features used in the MDP benchmark, which does not only depend on McCabe's and Halstead's metrics, but also the statistics of program codes, Line Count.

Table 1. The software metrics used as features in the MDP benchmark.

McCabe (4 metrics)	*v(G)*	cyclomatic_complexity		
	ev(G)	essential_complexity		
	iv(G)	design_complexity		
	loc	loc_total(one line = one count)		
Line Count (5 metrics)		loc_blank		
		loc_code_and_comment		
		loc_comments		
		loc_executable		
		branch_count		
Halstead (12 metrics)	N_1	num_operands	V	volume: $V = N * \log_2 \mu$
	N_2	num_operators	D	difficulty: $D = 1/L$
	μ_1	num_unique_operands	E	effort: $E = V/\tilde{L}$
	μ_2	num_unique_operators	T	prog_time: $T = E/18$ seconds
	B	error_est	L	level: $L = V^*/V$ where $V^* = (2+\mu_2^*)\log_2(2+\mu_2^*)$
	N	length: $N = N_1 + N_2$	I	content: $I = \tilde{L} * V$ where $\tilde{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$

2.2 Prior Works in Defect Prediction

In the field of software defect prediction, there were many trials in applying machine learning techniques to the MDP benchmark. Menzies et al. [1] applied Naïve Bayes to construct a classifier to predict software defects. There was an investigation on the feature selection using information gain. The proposed system obtained 71 % of the mean probability detection (PD) and 25 % of the mean false alarm rate (PF). Bo et al. [2] proposed a system called "GA-CSSVM". It built around SVM and used Genetic Algorithm (GA) to improve the cost sensitive parameter in SVM. The result showed that it achieved a promising performance in terms of AUC. Seliya et al. [3] introduced an algorithm called "RBBag". It employed the bagging concepts into two choices of classifiers: Naïve Bayes and C4.5. The result showed that RBBag outperformed the classical models without the bagging concept. Moreover, RBBag is more effective when it applied to Naïve Bayes than C4.5. Unfortunately, all of these works discard the imbalanced issue, so their reported results should be limited.

Shuo et al. [4] was aware of the imbalanced issue in the software defect prediction. There was an investigation on many strategies to tackle the imbalanced issue including resampling techniques, threshold moving, and ensemble algorithms. The result showed that AdaBoost.NC is the winner, and it also outperformed other conventional techniques: Naïve Bayes and Random Forest. Furthermore, a dynamic version of AdaBoost.NC was proposed and proved to be better than the original one.

Recently, support vector machine (SVM) have been applied in the field of software defect prediction showing a good prediction performance. Gray et al. [5] employed SVM to detect errors in the MDP data set. The analysis from the SVM results revealed that if a testing module has a large average of the decision values (SVM scores), there is high chance to found defects in it. Elish et al. [6] compared SVM to eight traditional classifiers, such as DT, NB, etc., on the MDP data set. The experiment demonstrated that SVM is the winner method. Thus, this is our motivation to apply a method built around SVM called "R-SVM" to detect the software errors.

2.3 Assessment in Defect Prediction

In the domain of binary classification problem (defect vs. non-defect), it is necessary to construct a confusion matrix, which comprises of four based quantities: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) as shown in Table 2.

Table 2. A confusion matrix.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

These four values are used to compute Precision (Pr), Probability of Detection (PD), Probability of False Alarm (PF), True Negative Rate (TNR), F-measure [11], and G-mean [12], which is a proper metric that frequently used to tackle the class imbalance problem as shown in Table 3.

Table 3. Prediction Performance Metrics

Metrics	**Definition**	**Formula**
Precision (Pr)	a proportion of examples predicted as defective against all of the predicted defective	$\frac{TP}{TP+FP}$
Probability of Detection (PD), Recall, TPR	a proportion of examples correctly predicted as defective against all of the actually defective	$\frac{TP}{TP+FN}$
Probability of False Alarm (PF), FPR	a proportion of examples correctly predicted as non-defective against all of the actually non-defective	$\frac{FP}{TN+FP}$
True Negative Rate (TNR)	a proportion of examples correctly predicted as non-defective against all of the actually non-defective	$\frac{TN}{TN+FP}$
G-mean	the square root of the product of TPR (PD) and TNR	$\sqrt{(TPR)\cdot(TNR)}$
F-measure	a weighted harmonic mean of precision and recall	$\frac{2\times \mathrm{Pr}\times \mathrm{Re}}{\mathrm{Pr}+\mathrm{Re}}$

3 Our Proposed Method

3.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) [13, 14] is one of the most famous classification techniques which was presented by Vapnik. It was shown to be more accurate than other classification techniques, especially in the domain of text categorization. It constructs a classification model by finding an optimal separating hyperplane that maximizes the margin between the two classes. The training samples that lie at the margin of the class distributions in feature space called support vectors.

The purpose of SVM is to induce a hyperplane function (Equation 1), where $\vec{w}$ is a weight vector referring to "orientation" and b is a bias.

$$h(\vec{w},b) = \vec{w}\cdot x + b \tag{1}$$

The optimization function to construct SVM hyperplane is shown in (2), where C is a penalty parameter of misclassifications.

$$\begin{aligned} &\underset{w,b,\xi}{Minimize} \quad \frac{1}{2}w^T w + C\sum_{i=1}^{n}\xi_i \\ &subject\ to \quad y_i\left(w^T\phi(x_i)+b\right) \geq 1-\xi_i \quad ,\xi_i \geq 0 \end{aligned} \tag{2}$$

In a non-linear separable problem, SVM handles this by using a kernel function (non-linear) to map the data into a higher space, where a linear hyperplane cannot be used to do the separation. A kernel function is shown in (3).

$$K(x_i, x_j) \equiv \phi(x_i)\phi(x_j) \tag{3}$$

Unfortunately, although SVM has shown an impressive result, it still suffers from the imbalanced issue like other traditional classification techniques.

3.2 Threshold Adjustment (R-SVM)

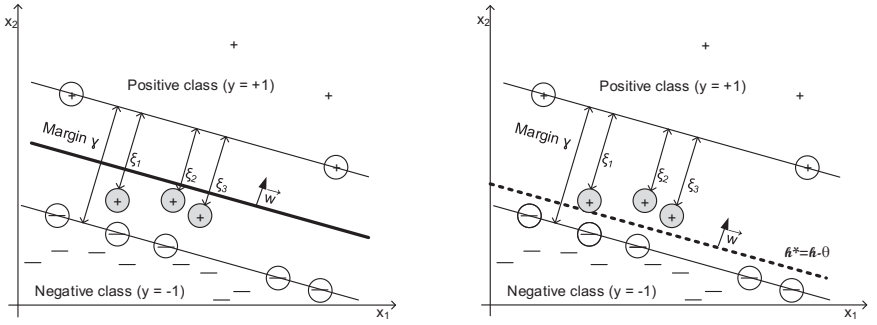
Although SVM has shown a good classification performance in many real-world data sets, it often gives low prediction accuracy in an imbalanced scenario. R-SVM [8] is an our earlier attempt that focuses to tackle this issue by applying the threshold adjustment strategy. To minimize a bias of the majority class, it translates a separation hyperplane in (1) without changing the orientation $\vec{w}$ by only adjusting b. After the SVM hyperplane has been induced from the set of training data mapped to SVM scores, L. The task is to find a new threshold, θ, that selected from the set of candidates thresholds, Θ, which gives the highest value of a user-defined criterion, $perf$ (e.g., the F_1 metric):

$$\left\{\theta \in \Theta \middle| \theta = \max\left(perf\left(L, \Theta\right)\right)\right\} \tag{4}$$

To avoid overfitting issue, the output θ is an average of the thresholds obtained from different training subsets. Finally, the SVM function is corrected as below:

$$h^*(x_i) = h(x_i) - \theta \tag{5}$$

Fig. 1 shows how "shifting" the hyperplane's bias downward in the bottom graph corrects the way SVM labels the three positive examples misclassified by the original hyperplane in the bottom graph (note that the hyperplane's orientation is unchanged).



(a) SVM hyperplanes before threshold adjustment. (b) SVM hyperplanes after threshold adjustment.

Fig. 1. SVM hyperplanes before (a) and after (b) threshold adjustment. The classification of three examples is corrected.

However, the enhanced hyperplane can reach an overfitting issue since the adjustment is based on just a single training data set. To increase a generalizability of the model, the resampling concept is applied to R-SVM (R stands for "resampling"). Moreover, the under-sampling concept is also employed in order to speed up the adjustment process.

4 Data Set and Tools

4.1 Data Set

The benchmark used in this experiment is the NASA Metric Data Program (MDP) [7] that comes from the NASA IV&V Facility MDP Repository, which contains a series of real software defect data from NASA spacecraft software. In this benchmark, there are 12 projects (data sets). The defect statistics is shown in Table 4. From the statistics, it has shown that the MDP data set suffers from the imbalanced issue. An average percentage of the defects is 20%, and the minimum percentage is only 2.15% in the PC2 data sets.

Table 4. Defect Statistics for Each Data Set.

Name	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5	Avg.
#Samples	327	7782	1183	194	1988	125	253	705	745	1077	1287	1711	1448
#Attributes	38	22	22	40	39	40	38	38	37	38	38	39	36
Defect Class	42	1672	314	314	46	44	27	61	16	134	177	471	277
Non-Defect Class	285	6110	869	158	1942	81	226	644	729	943	1110	1240	1195
%Defect	12.84	21.49	26.54	66.53	2.31	35.20	10.67	8.65	2.15	12.44	13.75	27.53	20

4.2 Tools

WEKA [15] is a popular machine learning software for data mining tasks. It is a product of University of Waikato in New Zealand and was first implemented in 1997. It supports several data mining process, such as preprocessing, regression, classification, and so on. All comparison methods in this experiment are carried out in Weka.

HR-SVM [16] is our SVM software that handles the imbalanced issue by using the threshold adjustment strategy. It is built on top of LIBSVM and HEMKit, and can run in any operating system. Moreover, it supports any kinds of classification tasks: single label classification (Binary classification), multi-class classification, multi-label classification, and hierarchical multi-label classification. "R-SVM" is a part of the HR-SVM software that is used for multiclass and multi-label data.

5 Experiments and Results

5.1 Experimental Setup

In this section, shows how to conduct the experiments in this paper. It starts from the data preprocessing by scaling all values into a range of [0, 1], which is suggested in [17]. Then, we compare the detection performance among different approaches as in the following steps. Note that all experiments are based on 10 fold-cross validation and measured using PD, PF, F1, and G-mean.

- Step1: find the baseline method which is the winner of the famous traditional classifiers: Naïve Bayes (NB), Decision Tree, k-NN, and SVM.
- Step2: find the best setting for R-SVM whether or not the feature selection is necessary.
- Step3: compare R-SVM (Step2) to the baseline method (Step1) along with a significance test using unpaired t-test at a confidence level of 95%

5.2 Experimental Results

The comparison of the conventional methods. In order to get the baseline for each data set, five classifiers: Naïve Bayes (NB), Decision Tree (DT), k-NN, SVM (Linear), and SVM (RBF), were tested and compared in terms of PD, PF, F1, and G-Mean (Table 5 – 8). For each row in the tables, the boldface method is a winner on that data set. From the result, k-NN and NB showed the best performance in almost all data sets, while the standard SVM gave the worst performance since it cannot detect any errors resulting 0% detection in many data sets. For Table 7 – 8, it is interesting that F1 and G-mean unanimously showed the same winners. Since F1 and G-mean are suitable metrics for imbalanced data, we selected the winner as a baseline using F1 and G-mean as summarized in Table 9.

Table 5. Prediction performance: PD

Name	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
CM1	0.190	0.143	**0.262**	0.000
JM1	0.228	**0.333**	0.189	0.090
KC1	**0.354**	0.417	0.322	0.048
KC3	0.333	0.139	**0.389**	0.028
MC1	0.065	0.304	**0.311**	0.000
MC2	0.523	**0.545**	0.386	0.364
MW1	0.148	**0.259**	0.000	0.037
PC1	0.213	**0.426**	0.350	0.000
PC2	0.000	0.000	**0.125**	0.000
PC3	0.261	0.336	**0.903**	0.000
PC4	**0.525**	0.492	0.375	0.198
PC5	0.482	**0.495**	0.206	0.161
Avg.	0.277	0.324	0.318	0.077
SD	0.172	0.166	0.219	0.112

Table 6. Prediction performance: PF

Name	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
CM1	**0.091**	0.147	0.130	0.000
JM1	0.091	0.176	0.056	**0.020**
KC1	0.113	0.181	0.124	**0.028**
KC3	**0.101**	0.146	0.114	0.000
MC1	**0.005**	0.011	0.088	0.000
MC2	0.185	0.198	**0.086**	0.111
MW1	0.044	0.115	0.000	**0.009**
PC1	0.032	0.053	0.071	**0.002**
PC2	**0.007**	0.021	0.080	0.000
PC3	**0.066**	0.077	0.744	0.000
PC4	0.059	0.085	0.048	**0.004**
PC5	0.143	0.187	0.053	**0.033**
Avg.	0.078	0.116	0.133	0.017
SD	0.054	0.066	0.196	0.032

Table 7. Prediction performance: F1

Name	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
CM1	0.211	0.133	**0.244**	0.000
JM1	0.292	**0.337**	0.271	0.155
KC1	0.424	**0.435**	0.386	0.289
KC3	0.375	0.156	**0.412**	0.054
MC1	0.102	**0.341**	0.122	0.000
MC2	0.561	**0.571**	0.500	0.478
MW1	0.195	**0.233**	0.000	0.067
PC1	0.271	**0.430**	0.331	0.000
PC2	0.000	0.000	**0.053**	0.000
PC3	0.303	**0.357**	0.254	0.000
PC4	**0.554**	0.486	0.447	0.324
PC5	**0.519**	0.498	0.307	0.259
Avg.	0.317	0.332	0.277	0.136
SD	0.178	0.171	0.155	0.164

Table 8. Prediction performance: G-mean

Name	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
CM1	0.416	0.349	**0.477**	0.000
JM1	0.455	**0.524**	0.422	0.298
KC1	0.560	**0.585**	0.531	0.420
KC3	0.547	0.344	**0.587**	0.167
MC1	0.255	**0.549**	0.533	0.000
MC2	0.653	**0.662**	0.594	0.576
MW1	0.376	**0.479**	0.000	0.192
PC1	0.454	**0.635**	0.570	0.000
PC2	0.000	0.000	**0.339**	0.000
PC3	0.494	**0.557**	0.481	0.000
PC4	**0.703**	0.671	0.598	0.444
PC5	**0.643**	0.634	0.442	0.395
Avg.	0.416	0.499	0.465	0.208
SD	0.455	0.191	0.166	0.213

The comparison of R-SVM with and without feature selection. In this section, we intend to provide the best setting for R-SVM by testing whether or not the feature selection can improve the prediction performance. The F1-results in Table 10 illustrate that the feature selection should not be employed into the system since it tre-

mendously decreased the performance. This should be because the number of attributes in the data sets is already small, so it cannot be further reduced.

Table 9. The winner of the baseline method for each data set in terms of F1 and Gmean.

Name	Winner	F1	G-mean
CM1	NB	0.244	0.477
JM1	k-NN	0.337	0.524
KC1	k-NN	0.435	0.585
KC3	NB	0.412	0.587
MC1	k-NN	0.341	0.549
MC2	k-NN	0.571	0.662
MW1	k-NN	0.233	0.479
PC1	k-NN	0.430	0.635
PC2	NB	0.053	0.339
PC3	k-NN	0.357	0.557
PC4	DT	0.554	0.703
PC5	DT	0.519	0.643
Avg.	-	0.374	0.562
SD	-	0.149	0.099

Table 10. A comparison of R-SVM between With and Without Feature Selection.

Name	F1 of R-SVM	
	With	*Without*
CM1	0.000	0.354**
JM1	0.008	0.411**
KC1	0.847	0.853
KC3	0.900	0.891
MC1	0.000	0.126*
MC2	0.333	0.574*
MW1	0.000	0.456**
PC1	0.000	0.392**
PC2	0.000	0.095*
PC3	0.000	0.384**
PC4	0.155	0.573**
PC5	0.845	0.846
Avg.	0.257	0.496
SD	0.379	0.264

Table 11. Comparison prediction performance measures between R-SVM and the baseline method. the boldface method is a winner on that dataset.

Name	PD		PF		F1		G-mean	
	Baseline	R-SVM	Baseline	R-SVM	Baseline	R-SVM	Baseline	R-SVM
CM1	0.262	**0.405**	**0.091****	0.130	0.244	**0.354**	0.477	**0.593****
JM1	0.333	**0.587****	**0.020****	0.301	0.337	**0.411****	0.524	**0.616****
KC1	0.354	**0.952****	**0.028****	0.777	0.435	**0.853****	**0.585****	0.461
KC3	0.389	**0.956****	**0.101****	0.722	0.412	**0.891****	**0.587**	0.508
MC1	**0.311**	0.174	**0.005****	0.030	**0.341***	0.126	**0.549****	0.409
MC2	0.545	**0.591**	**0.086****	0.346	0.571	**0.574**	**0.662**	0.657
MW1	0.259	**0.481**	**0.009****	0.075	0.233	**0.456***	0.479	**0.667****
PC1	0.426	**0.492**	**0.002****	0.096	**0.430**	0.392	0.635	**0.667**
PC2	0.125	0.125	**0.007****	0.033	0.053	**0.095**	0.339	**0.348**
PC3	**0.903****	0.463	**0.066****	0.135	0.357	**0.384**	0.557	**0.633****
PC4	0.525	**0.729****	**0.004****	0.130	0.554	**0.573**	0.703	**0.796****
PC5	0.495	**0.950****	**0.033****	0.764	0.519	**0.846**	**0.643****	0.458
Avg.	0.411	0.575	0.038	0.295	0.374	0.496	0.562	0.568
SD	0.197	0.282	0.038	0.293	0.149	0.264	0.099	0.131

* and ** represent a significant difference at a confidence level of 95% and 99%, respectively.

The comparison of R-SVM and the baseline methods. In this section, we compare R-SVM to the baseline methods, which are obtain from the first experiment as shown in Table 5-8. In Table 11 shows a comparison in terms of PD, PF, F1, and G-mean. All of the metrics give the same conclusion that R-SVM outperforms the baseline methods in almost all of the data sets. From 12 data sets, R-SVM *significantly* won 5, 3, and 5 on PD, F1, and G-mean, respectively. On average, F1-result of R-SVM outperforms that of the baselines for 32.62%, especially for the KC3 data set showing 116.26% improvement. Hence, this illustrates that it is effective to apply R-SVM as a core mechanism to early detect erroneous software modules.

6 Conclusion

Early defect detection is an important activity in the software development. Unfortunately, most of the prior works discarded the imbalanced issue, which is commonly found in the field of defect prediction, and it has known to severely affect the prediction accuracy. To tackle this issue, we proposed to employ our version of SVM called "R-SVM," which reduces a bias of the majority class by using the concept of threshold adjustment. The NASA Metric Data Program (MDP) was selected as our benchmark. It comprises of 12 projects (data sets). In the experiment, we compared R-SVM to five traditional classification techniques: Naïve Bayes, Decision Tree, k-NN, SVM (Linear), and SVM (RBF). The results showed that R-SVM overcame the imbalanced issue and significantly surpassed those classifiers.

References

[1] Menzies, T., Greenwald, J., Frank, A.: Data Mining Static Code Attributes to Learn Defect Predictors. In: IEEE Transactions on SE, vol. 33(1), pp. 2-13 (2007)
[2] Bo, S., Haifeng, L., Mengjun, L., Quan, Z., Chaojing, T.: Software Defect Prediction Using Dynamic Support Vector Machine. In: 9th International Conference on Computational Intelligence and Security (CIS), 2013, pp. 260-263. China (2013)
[3] Seliya, N., Khoshgoftaar, T.M., Van Hulse, J.: Predicting Faults in High Assurance Software. In: 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE), pp. 26-34. San Jose, CA(2010)
[4] Shuo, W., Xin, Y.: Using Class Imbalance Learning for Software Defect Prediction. In: IEEE Transactions on Reliability, vol. 62(2), pp. 434-443 (2013)
[5] Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B.: Software defect prediction using static code metrics underestimates defect-proneness. In: The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1-7. Barcelona (2010)
[6] Elish, K.O., Elish, M.O.: Predicting defect-prone software modules using support vector machines. In: Journal of System Software. vol. 81(5), pp. 649-660 (2008)
[7] NASA IV & V Facility. Metric Data Program, http://MDP.ivv.nasa.org/.
[8] Vateekul, P., Dendamrongvit, S., Kubat, M.: Improving SVM Performance in Multi-Label Domains: Threshold Adjustment. International Journal on Artificial Intelligence Tools (2013)
[9] McCabe, T.J.: A Complexity Measure. Software Engineering, In: IEEE Transactions on SE, vol. 2(4), pp. 308-320 (1976)
[10] Halstead, M.H.: Elements of Software Science. Elsevier Science Inc., (1977)
[11] Chidamber, S. R., Kemerer, C. F.: A metrics suit for object oriented design. In: IEEE Transactions on SE, vol. 20, pp. 476-493 (1994)
[12] Kubat, M., Matwin, S.: Addressing the curse of imbalanced training seta: One-sided selec-tion, pp. 179-186, 1997
[13] Han, J., Kamber, M.,: Data Mining: Concepts and Techniques, 2 ed., s.l.: Morgan Kaufmann, 2006.
[14] Cortes, C., Vapnik,V.: Support-Vector Networks. Machine Learning, pp.273-297(1995)
[15] WEKA, http://www.cs.waikato.ac.th.nz/ml/weka.
[16] Vateekul, P., Kubat, M., Sarinnapakorn, K.: Top-down optimized SVMs for hierarchical multi-label classification: A case study in gene function prediction. Intelligent Data Analysis (in press)
[17] Hsu, C.W., Chang, C.C., Lin, C.J.: A practical guide to support vector classification. Department of Computer Science and Information Engineering, National Taiwan University, (2003)