

# Detecting Outliers in Terms of Errors in Embedded Software Development Projects Using Imbalanced Data Classification

Kazunori Iwata, Toyoshiro Nakashima, Yoshiyuki Anan and Naohiro Ishii

**Abstract** This study examines the effect of undersampling on the detection of outliers in terms of the number of errors in embedded software development projects. Our study aims at estimating the number of errors and the amount of effort in projects. As outliers can adversely affect this estimation, they are excluded from many estimation models. However, such outliers can be identified in practice once the projects have been completed; therefore, they should not be excluded while constructing models and estimating errors or effort. We have also attempted to detect outliers. However, the accuracy of the classifications was not acceptable because of a small number of outliers. This problem is referred to as data imbalance. To avoid this problem, we explore rebalancing methods using k-means cluster-based undersampling. This method aims at improving the proportion of outliers that are correctly identified while maintaining the other classification performance metrics high. Evaluation experiments were performed, and the results show that the proposed methods can improve the accuracy of detecting outliers; however, they also classify too many samples as outliers.

---

K. Iwata (✉)

Department of Business Administration, Aichi University, 4-60-6, Hiraike-cho, Nakamura-ku, Nagoya, Aichi 453-8777, Japan  
e-mail: kazunori@vega.aichi-u.ac.jp

T. Nakashima

Department of Culture-Information Studies, Sugiyama Jogakuen University, 17-3 Moto-machi, Hoshigaoka, Chikusa-ku, Nagoya, Aichi 464-8662, Japan  
e-mail: nakasima@sugiyama-u.ac.jp

T. Nakashima

Institute of Managerial Research, Aichi University, 4-60-6, Hiraike-cho, Nakamura-ku, Nagoya, Aichi 453-8777, Japan

Y. Anan

Base Division, Omron Software Co., Ltd., Higashiiru, Shiokoji-Horikawa, Shimogyo-ku, Kyoto 600-8234, Japan  
e-mail: yoshiyuki\_anan@oss-g.omron.co.jp

N. Ishii

Department of Information Science, Aichi Institute of Technology, 1247 Yachigusa, Yakusa-cho, Toyota, Aichi 470-0392, Japan  
e-mail: ishii@aitech.ac.jp

© Springer International Publishing AG 2018

R. Lee (ed.), *Computational Science/Intelligence and Applied Informatics*, Studies in Computational Intelligence 726, DOI 10.1007/978-3-319-63618-4\_6

**Keywords** Embedded software · Imbalanced dataset · Support vector machine · k-means clustering algorithms · Undersampling

## 1 Introduction

The growth and expansion of our information-based society has resulted in an increasing number of information products. In addition, the functionality of these products is becoming ever more complex [6, 14]. Guaranteeing the quality of software is particularly important because it relates to reliability. Therefore, it is increasingly important for corporations that develop embedded software to implement efficient processes while guaranteeing timely delivery, high quality, and low development costs [2, 12, 15, 16, 18–21]. Companies and divisions involved in developing of such software focus on a variety of improvements, particularly in their processes. Estimating the number of errors and the amount of effort is necessary for new software projects and guaranteeing product quality is particularly important because the number of errors is directly related to the product quality and the amount of effort is directly related to cost, which affect the reputation of the corporation. Previously, we investigated the estimation of total errors and effort using an artificial neural network (ANN) and showed that ANN models are superior to regression analysis models for estimating errors and effort in new projects [8, 9]. We proposed a method to estimate intervals for the amount of effort using a support vector machine (SVM) and an ANN [7, 10]. These models were constructed with data that excluded outliers. The outliers can be identified in practice once the projects have been completed. Hence, they should not be excluded while constructing models and estimating effort. We attempted to classify embedded software development projects based on verifying whether the amount of efforts was an outlier using an ANN and SVM [11]. However, the accuracy of the classifications was not acceptable because of a small number of outliers. This problem occurs in most machine learning methods and is referred to as data imbalance. It exists in a broad range of experimental data [1, 22]. Data imbalance occurs when one of the classes in a dataset has a very small number of samples compared to the number of samples in other classes. When the number of instances of the majority class exceeds that of the minority class by a significant amount, most samples are classified into a class to which the majority samples belong. Therefore, the number of the outliers is small, and they are classified as normal values. To avoid this problem, we explored rebalancing methods in terms of errors using k-means [5] cluster-based undersampling. Evaluation experiments were performed to compare the classification accuracy using k-means undersampling with that of random undersampling and no undersampling using ten-fold cross-validation.

## **2 Related Work**

### ***2.1 Undersampling***

Undersampling is one of the most common and straightforward strategies for handling imbalanced datasets. Samples of the majority class are dropped to obtain a balanced dataset. Simple undersampling randomly drops samples to generate a balanced dataset.

### ***2.2 Cost-Sensitive Learning***

Unlike cost-insensitive learning, cost-sensitive learning is a type of learning that considers misclassification costs [17]. Additionally, cost-sensitive learning imposes different penalties for different misclassification errors. It aims at classifying samples into a set of known classes with high accuracy. Cost-sensitive learning is a common approach that solves the problem associated with imbalanced datasets.

#### **2.2.1 Cost-Sensitive SVMs**

SVMs have proven to be effective in many practical applications. However, the application of SVMs has limitations when applied to the problem of learning from imbalanced datasets. A cost-sensitive SVM, which assigns different misclassification costs, is good solution to address the problem [3, 13]. Such an SVM is developed using different error costs for the positive and negative classes, and can improve classification accuracy for a small number of classes.

### ***2.3 Our Contribution***

The above algorithm has a certain level of classification accuracy for some imbalanced datasets; however, it cannot improve the accuracy for highly imbalanced datasets. Therefore, in this research, we proposed a rebalancing method using k-means cluster-based undersampling.

### 3 Datasets and Outliers

#### 3.1 Original Datasets

Using data from a large software company, the classification methods divide the number of anticipated errors into normal values and outliers. The data consist of the following features:

*Class:*

This indicates whether the total number of errors for an entire project is a normal value or an outlier. Predicting this value is the objective of the classification.

Volume of newly added steps ( $V_{new}$ ):

This feature denotes the number of steps in the newly generated functions of the target project.

Volume of modification ( $V_{modify}$ ):

This feature denotes the number of steps modified or added to existing functions that were needed to use the target project.

Volume of the original project ( $V_{survey}$ ):

This feature denotes the original number of steps in the modified functions and the number of steps deleted from the functions.

Volume of reuse ( $V_{reuse}$ ):

This feature denotes the number of steps in a function of which an external specification is only confirmed and which are applied to the target project design without confirming the internal content.

#### 3.2 Determination of Outliers

This study examined the classification of outliers in terms of the number of errors in a project. Fig. 1 shows the distribution of the number of errors, whereas Fig. 2 is a boxplot of this metric. The lowest datum of the boxplot is 0, which is the lowest possible number of errors in the projects and higher than 1.5 times the interquartile range (IQR) of the lower quartile. The highest valid datum is within 1.5 times the IQR of the upper quartile. The outliers are denoted by circles. Here, the values are spread along the Y-axis to more clearly present the distribution of the outliers; however, the Y-coordinate has no other meaning. Of the total of 1,419 data points, 143 are outliers. Detailed values of the boxplot are listed in Table 1.

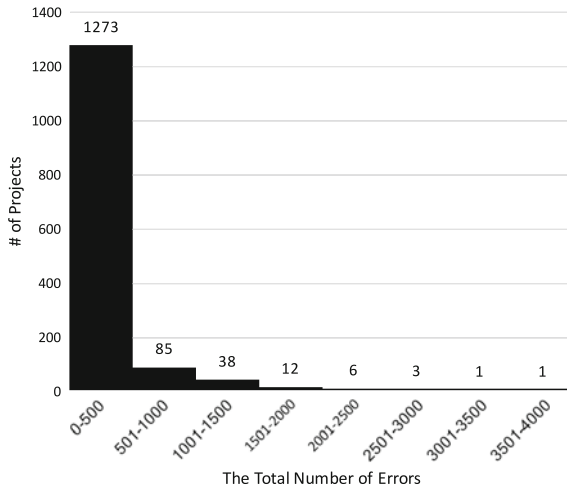


Fig. 1 Distribution of the total number of errors (in intervals of 500 errors)

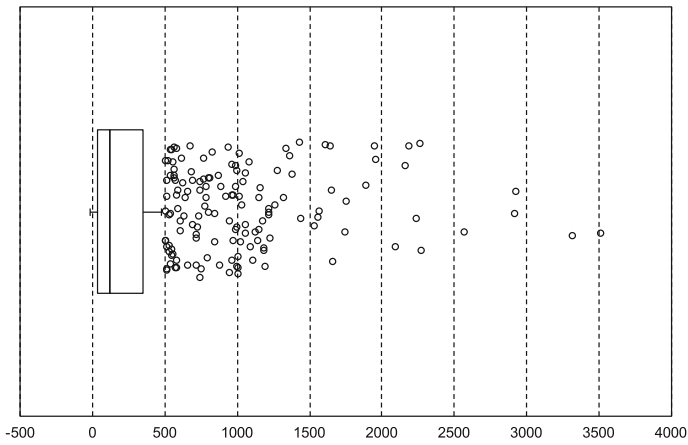


Fig. 2 Boxplot of the number of errors

Table 1 Detailed information of the boxplot shown in Fig. 2

	IQR	Minimum	Lower quartile	Median	Upper quartile	Maximum
Values	189.0	0.0	35.0	86.0	224.0	507.5

## 4 Classification Methods

The following classification methods were created to compare their accuracy:

- SVM without undersampling (SVM w/o).
- Cost-sensitive SVM without undersampling (CSSVM).
- SVM with random undersampling (W/Random).
- SVM with k-means cluster-based undersampling (W/ $n$  Clusters), where  $n$  is the number of clusters, which is varied from 2 to 15.

### 4.1 K-Means Cluster-Based Undersampling

The k-means clustering algorithm aims at finding the positions of clusters that minimize the distance from the data points to  $k$  clusters. The algorithm is often presented as a method that assigns samples to the nearest cluster by distance. The main steps of k-means are to select the initial cluster centers, change the classification of the data based on Euclidean distance and adjust the cluster centers according to the classification result. The clustering results are largely dependent on the initial cluster assignment. In this research, the clustering algorithm is applied to undersampling. The k-means cluster-based undersampling algorithm is shown in Algorithm 1.

---

#### Algorithm 1 Algorithm of K-Means Cluster-Based Undersampling

---

**Require:**  $L$ : majority class samples from a training set,  $S$ : minority class samples from a training set,  $k$ : the number of clusters.

**Ensure:**  $L_u$ : under-sampled samples

- 1: Initialize  $L_u$  to empty-set
- 2: Use k-means clustering to form clusters on  $L$  denoted by  $C_i$ , where  $1 < i \leq k$
- 3: **for**  $i = 1$  to  $k$  **do**
- 4: Calculate the number of samples for all the clusters denoted by:

$$n_i = \frac{|C_i|}{|L|} \times |S|$$

, where  $|X|$  indicates the number of elements in  $X$

- 5: Set  $T$  to  $n_i$  elements selected randomly from  $C_i$
  - 6: Assign the union  $L_u$  and  $T$  to  $L_u$
  - 7: **end for**
  - 8: **return**  $L_u$
-

## 5 Evaluation Experiment

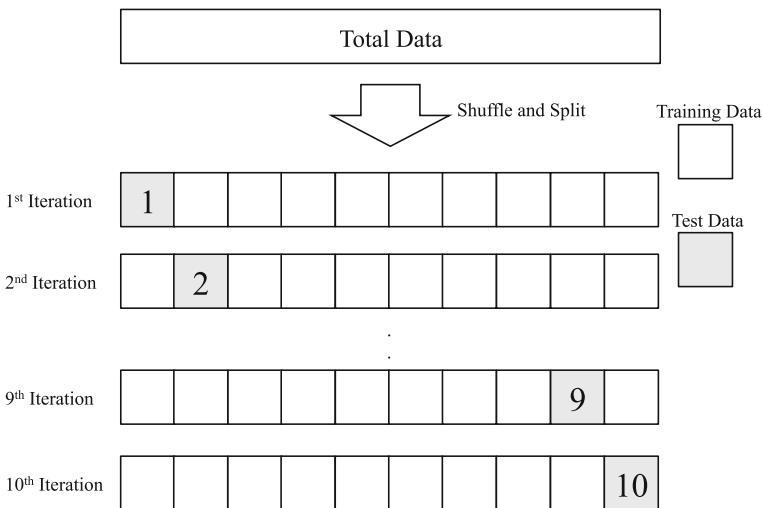
### 5.1 Data Used in the Evaluation Experiment

To evaluate the performance of the proposed technique, we performed ten-fold cross-validation on data from 1,419 real projects. The original data were randomly partitioned into 10 equally sized subsamples (each subsample having data from 141 or 142 projects). One of the subsamples was used as the validation data for testing the model, while the remaining nine subsamples were used as training data. The cross-validation process was repeated 10 times, with each of the 10 subsamples used exactly once as validation data. An example of ten-fold cross-validation is shown in Fig. 3.

### 5.2 Evaluation Criteria

This study focused on the imbalance problem wherein the minority class (outliers) has much lower precision and recall than the majority class (normal values). Accuracy metrics place more weight on the majority class than on the minority class, which makes it difficult for a classifier to perform well on the minority class.

By convention, the class label of the minority class is positive, whereas that of the majority class is negative. The True Positive (*TP*) and True Negative (*TN*) values, as summarized in Table 2, denote the number of positive and negative samples



**Fig. 3** Ten-fold cross-validation

**Table 2** Confusion matrix

	Predict positive	Predict negative
Actual positive	$TP$	$FN$
Actual negative	$FP$	$TN$

that are correctly classified, while the False Positive ( $FP$ ) and False Negative ( $FN$ ) values denote the numbers of samples incorrectly classified as positive and negative, respectively.

The following eight criteria were used as performance measures for the classification methods. The best value of these criteria is 1.0, whereas the worst is 0.0.

- (1) Accuracy ( $ACC$ ) is the proportion of the total number of predictions and is calculated as the number of all correct predictions divided by the total number of samples using the following equation:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \quad (1)$$

- (2) Precision ( $PREC$ ) is the proportion of correctly predicted positive cases. It is calculated as the number of accurate positive predictions divided by the total number of positive predictions using the following equation:

$$PREC = \frac{TP}{TP + FP}. \quad (2)$$

- (3) Sensitivity ( $SN$ , recall, or  $TP$  rate) is the proportion of positive cases that are correctly identified. It is calculated as the number of correct positive predictions divided by the total number of positives using the following equation:

$$SN = \frac{TP}{TP + FN}. \quad (3)$$

- (4) Specificity ( $SP$  or  $TN$  rate) is defined as the proportion of negative cases that are correctly classified and calculated as the number of correct negative predictions divided by the total number of negatives using the following equation (4):

$$SP = \frac{TN}{TN + FP}. \quad (4)$$

- (5, 6, 7) The F-measure ( $F_\beta$ ) is the harmonic mean of precision and sensitivity. It is calculated as a weighted ( $\beta$ ) average of the precision and sensitivity as follows:

$$F_\beta = \frac{(1 + \beta^2) \times PREC \times SN}{\beta^2 \times PREC + SN}. \quad (5)$$



**Table 3** Classification results obtained using the support vector machine without undersampling (SVM w/o) Method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	65	78
	Normal values	16	1260

**Table 4** Classification results obtained using the cost-sensitive SVM without undersampling (CSSVM) Method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	26	117
	Normal values	0	1276

F-measures  $F_{0.5}$ ,  $F_1$ , and  $F_2$  are commonly used. The larger  $\beta$ , the more importance sensitivity has in the equation.

- (8) The G-measure ( $G$ ) is based on the sensitivity of both the positive and negative classes, calculated as follows:

$$G = \sqrt{SN \times SP}. \tag{6}$$

This paper aimed to detect all of outliers; however, there is a trade-off between precision and sensitivity. In generally, precision improves at the expense of sensitivity and sensitivity improves at the expense of precision [4]. Thus, improving  $SN$ ,  $F_2$  and  $G$  was important while maintaining the other classification performance metrics high.

### 5.3 Results and Discussion

For each method described in Sect.4, the confusion matrices of the experimental results for all projects using ten-fold cross-validation are presented in Tables 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 and 20. The values in the tables represent the aggregate over 10 experiments. The results in the tables are summarized in Figs. 4, 5, 6, and 7. High values in Figs. 4 and 5 mean high accuracy. In contrast, low values in Figs. 6 and 7 suggest high accuracy.

Table 20 summarizes the results of the criteria for all methods. The underlined values indicate the best results. The top five methods for each criterion are indicated in bold type. The results of the criteria for SVM w/o, CSSVM, W/Random and W/14 Clusters appear in Fig. 8. The results of W/ $n$  Clusters except W/14 Clusters are omitted because these show similar features.

**Table 5** Classification results obtained using the SVM with random undersampling (W/Random) Method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	115	28
	Normal values	137	1139

**Table 6** Classification results obtained using the W/2 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	131	12
	Normal values	302	974

**Table 7** Classification results obtained using the W/3 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	130	13
	Normal values	309	967

**Table 8** Classification results obtained using the W/4 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	136	7
	Normal values	301	975

**Table 9** Classification results obtained using the W/5 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	131	12
	Normal values	310	966

**Table 10** Classification results obtained using the W/6 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	133	10
	Normal values	299	977

**Table 11** Classification results obtained using the W/7 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	137	6
	Normal values	313	964

**Table 12** Classification results obtained using the W/8 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	131	12
	Normal values	312	964

**Table 13** Classification results obtained using the W/9 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	133	10
	Normal values	299	977

**Table 14** Classification results obtained using the W/10 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	133	10
	Normal values	311	965

**Table 15** Classification results obtained using the W/11 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	132	11
	Normal values	292	984

**Table 16** Classification results obtained using the W/12 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	137	6
	Normal values	306	970

**Table 17** Classification results obtained using the W/13 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	130	13
	Normal values	299	977

**Table 18** Classification results obtained using the W/14 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	134	9
	Normal values	289	987

**Table 19** Classification results obtained using the W/15 clusters method

		Predicted classes	
		Outliers	Normal values
Actual classes	Outliers	134	9
	Normal values	321	955

**Table 20** Accuracy comparison for all methods

	<i>ACC</i>	<i>PREC</i>	<i>SN</i>	<i>SP</i>	<i>F<sub>0.5</sub></i>	<i>F<sub>1</sub></i>	<i>F<sub>2</sub></i>	<i>G</i>
SVM w/o	<b>0.9338</b>	<b>0.8025</b>	0.4545	<b>0.9875</b>	<b>0.6959</b>	<b>0.5804</b>	0.4977	0.6700
CSSVM	<b>0.9175</b>	<b>1.000</b>	0.1818	<b>1.0000</b>	<b>0.5263</b>	0.3077	0.2174	0.4264
W/Random	<b>0.8837</b>	<b>0.4563</b>	0.8042	<b>0.8926</b>	<b>0.4996</b>	<b>0.5823</b>	<b>0.6978</b>	<b>0.8473</b>
W/2 Clusters	0.7787	0.3025	0.9161	0.7633	0.3493	0.4549	0.6517	0.8362
W/3 Clusters	0.7731	0.2961	0.9091	0.7578	0.3423	0.4467	0.6429	0.8300
W/4 Clusters	0.7829	0.3112	<b>0.9510</b>	0.7641	<b>0.3596</b>	<b>0.4690</b>	<b>0.6739</b>	<b>0.8525</b>
W/5 Clusters	0.7731	0.2971	0.9161	0.7571	0.3435	0.4486	0.6466	0.8328
W/6 Clusters	0.7822	0.3079	0.9301	0.7657	0.3554	0.4626	0.6624	0.8439
W/7 Clusters	0.7759	0.3051	<b>0.9580</b>	0.7555	0.3533	0.4628	<b>0.6709</b>	<b>0.8508</b>
W/8 Clusters	0.7717	0.2957	0.9161	0.7555	0.3420	0.4471	0.6453	0.8319
W/9 Clusters	0.7822	0.3079	0.9301	0.7657	0.3554	0.4626	0.6624	0.8439
W/10 Clusters	0.7738	0.2995	0.9301	0.7563	0.3465	0.4532	0.6545	0.8387
W/11 Clusters	<b>0.7865</b>	<b>0.3113</b>	0.9231	<b>0.7712</b>	0.3589	0.4656	0.6627	0.8437
W/12 Clusters	0.7801	0.3093	<b>0.9580</b>	0.7602	0.3577	<b>0.4676</b>	<b>0.6749</b>	<b>0.8534</b>
W/13 Clusters	0.7801	0.3030	0.9091	0.7657	0.3497	0.4545	0.6494	0.8343
W/14 Clusters	<b>0.7900</b>	<b>0.3168</b>	<b>0.9371</b>	<b>0.7735</b>	<b>0.3651</b>	<b>0.4735</b>	<b>0.6734</b>	<b>0.8514</b>
W/15 Clusters	0.7674	0.2945	<b>0.9371</b>	0.7484	0.3413	0.4482	0.6524	0.8375

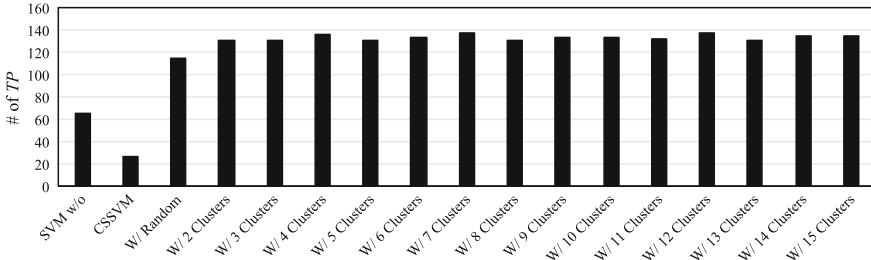


Fig. 4 Number of TP for methods

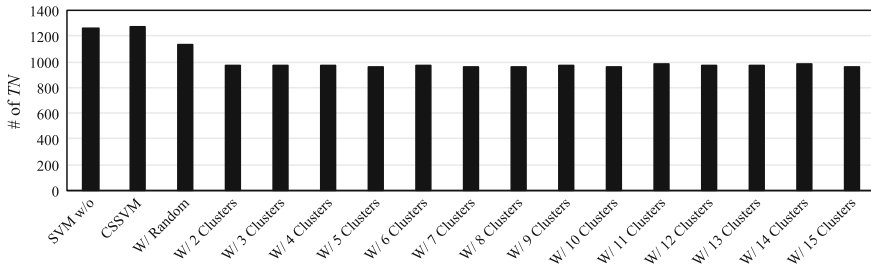


Fig. 5 Number of TN for methods

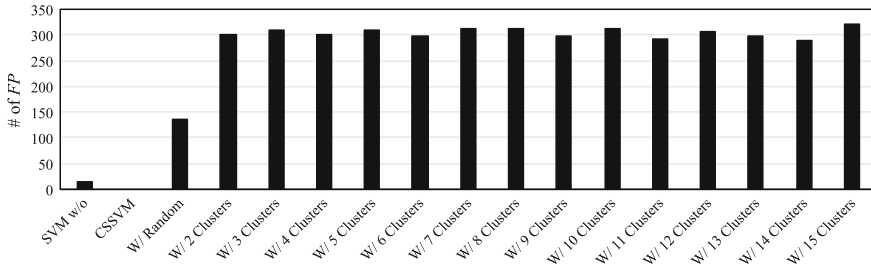


Fig. 6 Number of FP for methods

The SVM w/o method shows the best ACC; however it is second worst with respect to SN. This is because most outliers are classified into a class of normal values (the majority class) specified in Table 3, which is a common problem with imbalanced datasets.

The CSSVM method obtains a perfect result for PREC; however, it obtains the worst result for SN. This is because the method imposes heavier costs for misclassifying outliers and can reduce FP. At the same time, it increases FN. The results indicate that CSSVM can accurately detect some outliers; however it overlooks most outliers. In other words, the outliers predicted by CSSVM must be actual outliers; however, only a few outliers are detected by it.

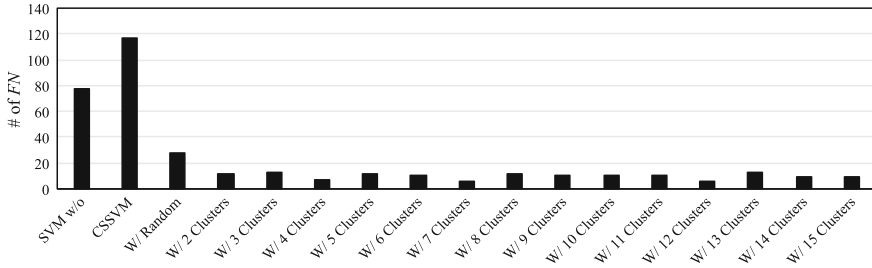


Fig. 7 Number of FN for methods

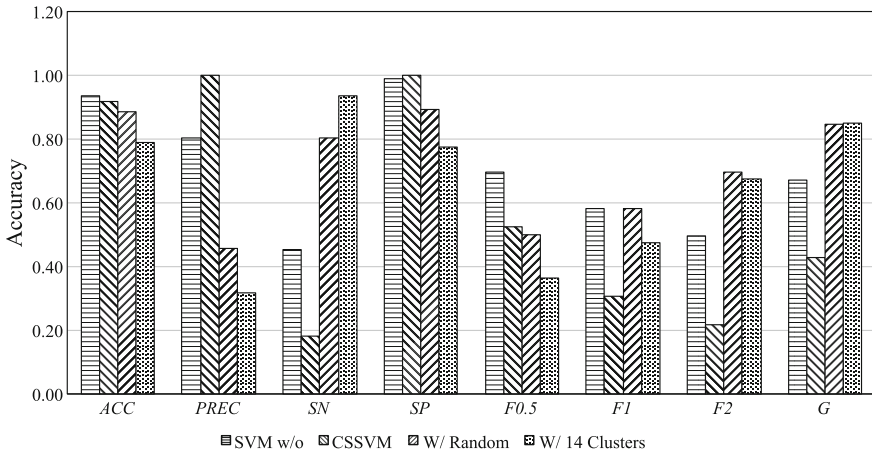


Fig. 8 Accuracy comparison of four methods

The results of the methods with undersampling show similar results. They have higher  $SN$ ,  $F_2$  and  $G$  but lower  $PREC$  than those of the methods without undersampling. The W/Randoms method obtains higher  $PREC$  than those of the methods with k-means cluster-based undersampling. In contrast, the W/ $n$  Clusters methods have better results in terms of  $SN$ . In addition, the classification criteria accuracy depends on the number of clusters, and all results of the W/14 Clusters method are within the top five. These results show that the proposed methods can improve the accuracy of detecting outliers; however they tend to classify too many samples as outliers.

## 6 Conclusion

This research examined the ability of undersampling to detect outliers in terms of the number of errors in embedded software development projects. The undersampling method was based on the k-means clustering algorithm. The method aimed at

improving the proportion of outliers that were correctly identified while maintaining the other classification performance metrics high.

Evaluation experiments were conducted to compare the prediction accuracy of the methods with k-means undersampling, random undersampling and without undersampling using ten-fold cross-validation.

The results indicated that the methods with undersampling have higher sensitivity and lower precision than those of the methods without undersampling. The results further indicated that the proposed methods improved the accuracy of detecting outliers but classified too many samples as outliers.

In future research, we plan to investigate the following:

1. We plan to apply the oversampling method to improve the accuracy of precision, while keeping sensitivity high.
2. We intend to consider other methods to detect outliers.
3. More data are needed to further support our research. In particular, data for projects that include outliers are essential for improving the models.

**Acknowledgements** This work was supported by JSPS KAKENHI Grant Number JP16K00310 and JP17K00317.

## References

1. Barandela, R., Sánchez, J.S., Garca, V., Rangel, E.: Strategies for learning in class imbalance problems. *Pattern Recognit.* **36**(3), 849–851 (2003). <http://dblp.uni-trier.de/db/journals/pr/pr36.html>
2. Boehm, B.: Software engineering. *IEEE Trans. Softw. Eng.* **C-25**(12), 1226–1241 (1976)
3. Fumera, G., Roli, F.: Cost-sensitive learning in support vector machines. In: VIII Convegno Associazione Italiana per L'Intelligenza Artificiale (2002)
4. Gordon, M., Kochen, M.: Recall-precision trade-off: a derivation. *J. Am. Soc. Inf. Sci.* **40**(3), 145–151 (1989)
5. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. *J. R. Stat. Soc. Series C (Appl. Stat.)* **28**(1), 100–108 (1979). <http://www.jstor.org/stable/2346830>
6. Hirayama, M.: Current state of embedded software (in japanese). *J. Inf. Process. Soc. Jpn (IPSJ)* **45**(7), 677–681 (2004)
7. Iwata, K., Liebman, E., Stone, P., Nakashima, T., Anan, Y., Ishii, N.: Bin-Based Estimation of the Amount of Effort for Embedded Software Development Projects with Support Vector Machines, pp. 157–169. Springer International Publishing (2016)
8. Iwata, K., Nakashima, T., Anan, Y., Ishii, N.: Error estimation models integrating previous models and using artificial neural networks for embedded software development projects. In: Proceedings of 20th IEEE International Conference on Tools with Artificial Intelligence, pp. 371–378 (2008)
9. Iwata, K., Nakashima, T., Anan, Y., Ishii, N.: Improving accuracy of an artificial neural network model to predict effort and errors in embedded software development projects. In: Lee, R., Ma, J., Bacon, L., Du, W., Petridis, M. (eds.) Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2010, *Studies in Computational Intelligence*, vol. 295, pp. 11–21. Springer Berlin Heidelberg (2010). doi:[10.1007/978-3-642-13265-0\\_2](https://doi.org/10.1007/978-3-642-13265-0_2)
10. Iwata, K., Nakashima, T., Anan, Y., Ishii, N.: Estimating interval of the number of errors for embedded software development projects. *Int. J. Softw. Innov. (IJSI)* **2**(3), 40–50 (2014). doi:[10.4018/ijsi.2014070104](https://doi.org/10.4018/ijsi.2014070104)

11. Iwata, K., Nakashima, T., Anan, Y., Ishii, N.: Effort estimation for embedded software development projects by combining machine learning with classification. In: Proceedings of 3rd ACIS International Conference on Computational Science/Intelligence and Applied Information, pp. 265–270 (2016)
12. Komiyama, T.: Development of foundation for effective and efficient software process improvement. *J. Inf. Process. Soc. Jpn (IPSJ)* **44**(4), 341–347 (2003) (in japanese)
13. Masnadi-Shirazi, H., Vasconcelos, N.: Risk minimization, probability elicitation, and cost-sensitive svms. In: J. Fürnkranz, T. Joachims (eds.) ICML, pp. 759–766. Omnipress (2010). <http://dblp.uni-trier.de/db/conf/icml/icml2010.html>
14. Nakamoto, Y., Takada, H., Tamaru, K.: Current state and trend in embedded systems. *J. Inf. Process. Soc. Jpn (IPSJ)* **38**(10), 871–878 (1997) (in japanese)
15. Nakashima, S.: Introduction to model-checking of embedded software. *J. Inf. Process. Soc. Jpn (IPSJ)* **45**(7), 690–693 (2004) (in japanese)
16. Ogasawara, H., Kojima, S.: Process improvement activities that put importance on stay power. *J. Inf. Process. Soc. Jpn (IPSJ)* **44**(4), 334–340 (2003) (in japanese)
17. Sammut, C., Webb, G.I. (eds.): *Encyclopedia of Machine Learning and Data Mining*. 2. Springer US (2017)
18. Takagi, Y.: A case study of the success factor in large-scale software system development project. *J. Inf. Process. Soc. Jpn (IPSJ)* **44**(4), 348–356 (2003) (in japanese)
19. Tamaru, K.: Trends in software development platform for embedded systems. *J. Inf. Process. Soc. Jpn (IPSJ)* **45**(7), 699–703 (2004) (in japanese)
20. Ubayashi, N.: Modeling techniques for designing embedded software. *J. Inf. Process. Soc. Jpn (IPSJ)* **45**(7), 682–692 (2004) (in japanese)
21. Watanabe, H.: Product line technology for software development. *J. Inf. Process. Soc. Jpn (IPSJ)* **45**(7), 694–698 (2004) (in japanese)
22. Weiss, G.M.: Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.* **6**(1), 7–19 (2004). doi:[10.1145/1007730.1007734](https://doi.org/10.1145/1007730.1007734)