Agreement-based SVM One-Class Classification: an Ensemble Method for Software Defect Prediction

Luigi Lavazza
Università degli Studi dell'Insubria
Varese, Italy
email:luigi.lavazza@uninsubria.it

Sandro Morasca
Università degli Studi dell'Insubria
Varese, Italy
email:sandro.morasca@uninsubria.it

Gabriele Rotoloni
Università degli Studi dell'Insubria
Varese, Italy
email:gabriele.rotoloni@uninsubria.it

***Abstract*—One-class Support Vector Machine (SVM) classification is well-suited to train classifiers when the training dataset is very imbalanced. So, it is increasingly used for software defect prediction, since faulty modules are usually relatively rare, compared to non-faulty ones. Though a minority, faulty modules often are a non-negligible fraction of the total; hence, completely ignoring them when training a one-class model can lead to discarding potentially useful information: we investigate how to exploit the information embedded in the minority class when using one-class SVM classification. To this end, we devised an ensemble software defect prediction method that takes advantage of one-class SVM classification without ignoring the minority class modules. Two one-class SVM defectiveness classification models are built: one based on the majority class modules alone and the other on the minority class modules alone. If both models agree on the defectiveness of a module, it is classified accordingly; otherwise, as "uncertain." Uncertainty can then be treated in many different ways. Here, we adopt a "prudential" approach: a module is classified non-faulty only if it is estimated non-faulty by both one-class classifiers; otherwise, faulty. The proposed approach was tested by comparing its results with the ones obtained via two-class and one-class SVM classifiers. The results show that our technique is a valid alternative to both two-class and one-class classifications. Specifically, our approach achieves lower global cost when the cost of a false negative (i.e., a faulty module incorrectly believed non-faulty) is sufficiently higher than the cost of a false positive (a non-faulty module incorrectly believed faulty).**

Keywords-Software defect prediction; ensemble methods; machine Learning; Support Vector Machine; One-class classification; One-class SVM.

I. Introduction

Traditional Support Vector Machine (SVM) classification distinguishes two or more classes by learning from a training set with instances from all the classes. Instead, One-Class Classification (OCC) identifies instances of a specific class amongst all instances, by learning from a training set with only instances of that class.

OCC is particularly useful when instances of one class are rare, so that the available training sets are severely imbalanced. This is often the case with software defect prediction, since usually a small subset of the modules (e.g., classes) in a software system is faulty, so most datasets are imbalanced in favor of non-faulty modules. For datasets with medium to high imbalance, OCC methods may perform better than traditional methods [1].

However, completely ignoring information on the minority class instances may not be a good idea, especially because faulty software modules may still account for 10–20% of the total (see Section III-A). Faulty modules are definitely fewer than non-faulty ones, but they are not really extremely rare, in absolute terms.

In this paper, we investigate whether useful information can be obtained by considering both the majority and minority classes, while using OCC SVM. We define an ensemble method that uses the results of two OCC SVM models, one based on the majority class and the other on the minority class. We call this method ABOCC (Agreement-Based One-Class Classification).

When the two OCC models agree that a module is non-faulty, ABOCC classifies it as non-faulty. Likewise, when both models classify it as faulty, the module is classified as faulty by ABOCC. A module that receives conflicting classifications from the two OCC models is classified as "uncertain." There are several ways to deal with this uncertainty. We here investigate a "prudential" approach, in which uncertain modules are classified as faulty.

We carried out an empirical study to evaluate ABOCC and compare it with both traditional SVM and OCC SVM.

We find that "prudential" ABOCC is the best at minimizing false negatives, but it yields the most false positives. Therefore, when releasing a faulty module has severe consequences, ABOCC is economically more beneficial than traditional classification. These results show that considering both the minority class and majority class can be important in some contexts.

The paper is organized as follows. Section II describes the proposed ensemble method. Section III describes the empirical study. The execution and results of the study are discussed in Section IV, along with hints on how to use the proposed method. Section V discusses the threats to the validity of the empirical study. Section VI accounts for related work. Section VII draws some conclusions and outlines future work.

II. An Ensemble Classification Approach Based on One-class Classifications

Building prediction models using the data from just the majority class of instances has the obvious drawback of not considering the knowledge embedded in minority class instances. To avoid this issue, ABOCC uses data from both classes.

The idea is to train two OCC classifiers, one only on positive instances and one only on negative instances, and then combine

them via simple voting. If both classifiers agree, there is reasonable confidence that the classification is correct (more than with a classification by a single classifier, whether obtained on the entire dataset or on one class only). If, instead, the two classifications are conflicting, leaving a module's classification uncertain seems reasonable, at least at first.

A. Classification with Uncertainty

The ABOCC classifier is defined as follows.

1) A binary classifier is built via one-class classification, using only positive instances. We denote it as bc_p.
2) A binary classifier is built via one-class classification, using only negative instances. We denote it as bc_n.
3) An instance i is then classified using bc_p and bc_n: let $c_p(i)$ and $c_n(i)$ be the obtained classifications, respectively. The classification $c_e(i)$ by ABOCC is
   $c_e(i) = positive \Leftrightarrow c_p(i) = positive \wedge c_n(i) = positive$
   $c_e(i) = negative \Leftrightarrow c_p(i) = negative \wedge c_n(i) = negative$
   $c_e(i) = uncertain \Leftrightarrow c_p(i) \neq c_n(i)$

The one-class classification method used in steps (1) and (2) above is one-class SVM, a kernel-based method [2] of SVM, proposed by Schölkopf and Müller in 2001 [3][4]. The main idea of one-class SVM is that training data in input space are mapped into feature space via a kernel to find a hyperplane with a maximum margin in feature space to separate the transformed data from the origin.

Since the two OCC models are trained on two separate datasets, ABOCC mitigates a problem of vote-based ensemble models, i.e., the possible correlation among the training set features. Take two classifiers, one using the lines of code and the other the number of statements, trained on a dataset. The two metrics are highly correlated, so the two classifications will be very similar. If these models, along with a third model, are used in a vote-based ensemble model, their classification always prevails on the third model.

B. The Prudential Classification Approach

Since uncertain instances must be classified one way or the other, one possibility is to consider all uncertain instances as positive. This decision corresponds to a prudential attitude, which in software engineering is justified by the cost of not properly treating defective modules, which can cause severe consequences if released to end-users. The corresponding ensemble classifier, ABOCC$_p$ (where '*p*' is for prudential or pessimistic), is defined as follows.

Given bc_p and bc_n described in Section II-A, the classification by the binary prudential ensemble method $c_{bp}(i)$ is
$c_{bp}(i) = negative \Leftrightarrow c_p(i) = negative \wedge c_n(i) = negative$;
$c_{bp}(i) = positive \Leftrightarrow c_p(i) = positive \vee c_n(i) = positive$.
Unlike ABOCC, which returns three classes, i.e., positive, negative, and uncertain, ABOCC$_p$ is a proper binary classifier.

III. Empirical Evaluation: Goals and Method

The proposed method ABOCC$_p$ was evaluated via an empirical study. We here describe the goals and organization of the study; its execution and results are described in Section IV.

To investigate whether ignoring one of the classes has an ill effect on classification, we evaluate ABOCC$_p$ compared to traditional classification models with the following research question.

Research Question: How well does ABOCC$_p$ perform compared to "regular" SVM, one-class SVM, and random classification?

To answer the research question, we consider the performance in predicting (1) actually faulty and actually non-faulty modules separately, and (2) the entire test set, containing both faulty and not-faulty modules. To this end, we use widely used performance metrics.

Though performance metrics provide quantitative indications of how good a model is at predicting faultiness, software developers are not all that interested in optimizing this kind of abstract evaluations. Instead, they want to minimize total software costs, which include the cost of development activities and the costs due to software failures after the release to end-users. If we assume that modules estimated faulty undergo some quality improvement activity, and modules estimated non-faulty are released as-is, costs are clearly related to the performance of faultiness predictors. Thus, we could evaluate predictors based on the costs they imply. These considerations are out of the scope of this paper and will be the object of future work.

A. Dataset

We perform cross-version prediction: the data from a version are the training set, and the data from the following version are the test set. For example, when the versions available in the collection are `project-1.1`, `project-1.2`, and `project-2.0`, first we use `project-1.1` for training and `project-1.2` for testing, then we use `project-1.2` for training and `project-2.0` for testing.

We used the dataset collection by Jureczko and Madeyski (J&M in what follows) [5], available from Zenodo [6]. Specifically, we used the datasets providing data from multiple versions of the same project, namely those concerning projects `ant`, `ivy`, `jedit`, `lucene`, `poi`, `synapse`, `velocity`, `xalan`, `xerces`. Overall, 24 training-testing dataset pairs were used. Descriptive statistics for the dataset collections are in Table I.

TABLE I. Descriptive statistics.

	mean	stdev	median	min	max
Modules	423	212	351	178	909
Prevalence	0.340	0.248	0.258	0.022	0.988

B. Model Building

We built software defect prediction models via "traditional" binary SVM, one-class SVM, and ABOCC$_p$. We also built prediction models based on random classification: these models were used as baselines, since one expects that a practically useful prediction model performs better than random classification.

As for hyperparameter tuning, SVM was tuned on cost, epsilon, and gamma. We used the `R` function `tune.svm`, from library `e1071`, which performs a Grid Search with cross validation; the process was repeated four times and we selected the best performing model.

For the one-class models, the most important parameter to tune is ν, the expected proportion of outliers in the training set. Since there is no real consensus on its possible values, we considered both low values, like 0.05, as in the literature [7] and higher values for different possible models, specifically 0.1, 0.2, 0.5 and 0.8. Since `tune.svm` does not support the tuning of one-class models, we wrote a function to perform the Grid Search with cross validation that chooses the best performing model. Like before, the process is repeated four times and the best performing model is selected.

We built models by using the training set "as-is," without balancing, and by using training sets balanced via SMOTE [8]. We used the `R` library `performanceEstimation` and its `smote` function, which performs both over-sampling and under-sampling.

Over-sampling generates elements of the minority class using the K-nearest neighbors. We set K=5, like Chawla et al. did in their original paper. The number of elements generated depends on parameter `perc.over`. For example, with `perc.over=2`, the algorithm generates twice the amount of minority class elements.

Under-sampling randomly selects elements of the majority class. Parameter `perc.under` indicates the amount of elements to pick in relation to the amount of elements generated during the over-sampling. For example, `perc.under=3` means that the algorithm will pick three times the amount of elements generated by the over-sampling. Thus, the size of the resulting dataset depends on the number of elements of the minority class, meaning that the final result can be smaller than the starting dataset, especially when the ratio between minority and majority is very small. The larger the values of `perc.over` and `perc.under`, the larger and less balanced the resulting dataset. To achieve a good trade-off between size and balance, we set `perc.over=perc.under=2`, which makes prevalence (the proportion of positive instances) equal to $3/7$, since the positives are the minority class. The result is a relatively large datasets with total amount of elements seven times the original number of minority class elements.

Random classification was performed by estimating every module faulty with probability equal to ρ, the proportion of faulty modules in the test set. The performance of random classification is computed on the mean values of TP, FN, TN and FP.

C. Model Evaluation and Comparison

For each training set, each model yields a confusion matrix.

Given a dataset containing n modules, of which AP actually positive (i.e., faulty) and AN actually negative (i.e., non-faulty), each model estimates EP modules positive and the remaining EN modules negative. Of the EP estimated positives, TP are true positives, i.e., they are correctly estimated; the remaining FN are false negatives, i.e., they are positive, but are incorrectly estimated negative. Similarly, of the EN estimated negatives, TN are true negatives, i.e., they are correctly estimated; the remaining FP are false positives, i.e., they are negative, but are incorrectly estimated positive.

TABLE II. A CONFUSION MATRIX.

	Actual Neg.	Actual Pos.	
Est. Negative	TN	FN	EN=TN+FN
Est. Positive	FP	TP	EP=FP+TP
	AN=TN+FP	AP=FN+TP	n=AN+AP=EN+EP

AP and AN, hence n=AP+AN, are properties of the dataset and so is prevalence $\rho = \frac{AP}{n}$, the proportion of faulty instances. A perfectly balanced dataset has $\rho = \frac{1}{2}$; the closer ρ to zero or one, the more imbalanced is the dataset.

Usually, the performance of a given classifier is expressed via performance metrics, defined as functions of the confusion matrix cells. We evaluate classifiers via the following performance metrics.

- $TPR = \frac{TP}{AP}$. The True Positive Rate is the proportion of actually faulty modules that were correctly classified. TPR is a quite popular metric, also known as recall or sensitivity.
- $TNR = \frac{TN}{AN}$. The True Negative Rate is the proportion of actually non-faulty modules that were correctly classified. It is also known as specificity. Note that TNR=1–FPR, where FPR is the False Positive Ratio, used—along with TPR—to define Receiver Operating Characteristic (ROC) curves.
- $Gmean = \sqrt{TPR\ TNR}$, the geometric mean of TPR and TNR.
- $MCC = \frac{TP\ TN – FP\ FN}{\sqrt{AP\ AN\ EP\ EN}}$, the Matthews Correlation Coefficient [9].

TPR and TNR are used to represent a model's performance in correctly classifying faulty and, respectively, non-faulty modules. Gmean and MCC are used to provide "global" evaluations of the model, thus summarizing the contents of the confusion matrix. We use both Gmean and MCC as "overall" performance indicators because Gmean, being the geometric mean if TPR and TNR, does not depend on the (im)balance of the test set, while MCC takes into account the proportion of faulty modules, along with TPR and TNR.

For the random model, the mean performance metrics are $\text{TPR}_{rnd} = \rho$, $\text{TNR}_{rnd} = 1 - \rho$, and $\text{MCC}_{rnd} = 0$ [10]. We are not aware of a formula for computing $\text{Gmean}_{rnd}(\rho)$, hence we use values of $\text{Gmean}_{rnd}(\rho)$ obtained via simulation.

IV. EXPERIMENTAL EVALUATION: RESULTS

We report the results obtained for five types of models:

– SVM. Since the literature reports that better results are achieved with balanced datasets, we used SMOTE to obtain fairly balanced datasets to train SVM models.
– One-class SVM. Since this method is deemed useful for unbalanced datasets, we used it without balancing the datasets.

– Random classification is carried out estimating each module with a probability (the proportion of faulty modules) that depends on the testing set: there is no training set, and using a balancing technique like SMOTE is just not applicable.
– Concerning ABOCC$_p$, we investigated whether it could benefit from training set balancing. Hence, we created ABOCC$_p$ models both with and without SMOTE.

A summary of the performance metrics obtained with the prediction methods is in Table III. In the table and afterwards, the methods are labeled as follows: OCSVM indicates one-class SVM, SVM_SMOTE indicates traditional two-class SVM with SMOTE, ABOCC_SMOTE and ABOCC indicate ABOCC$_p$ with and without SMOTE.

TABLE III. SUMMARY OF PERFORMANCE METRICS PER METHOD.

		mean	sd	median	min	max
OCSVM	TPR	0.596	0.206	0.584	0.081	0.944
	TNR	0.553	0.197	0.589	0.081	1.000
	Gmean	0.543	0.129	0.581	0.218	0.708
	MCC	0.114	0.167	0.14	-0.234	0.376
SVM_SMOTE	TPR	0.497	0.248	0.550	0.178	0.958
	TNR	0.686	0.212	0.743	0.192	1.000
	Gmean	0.539	0.123	0.540	0.383	0.765
	MCC	0.147	0.136	0.121	-0.140	0.419
ABOCC	TPR	0.801	0.145	0.808	0.599	1.000
	TNR	0.341	0.192	0.377	0.007	0.606
	Gmean	0.476	0.174	0.525	0.081	0.697
	MCC	0.127	0.110	0.120	-0.075	0.360
ABOCC_SMOTE	TPR	0.868	0.117	0.886	0.605	1.000
	TNR	0.264	0.168	0.291	0.000	0.565
	Gmean	0.422	0.194	0.487	0.000	0.618
	MCC	0.121	0.082	0.134	-0.004	0.255
random	TPR	0.340	0.248	0.257	0.023	0.988
	TNR	0.660	0.248	0.742	0.013	0.978
	Gmean	0.393	0.107	0.433	0.114	0.500
	MCC	0.000	0.000	0.000	0.000	0.000

We illustrate the comparisons among predictors' performances via boxplots, whose indications are confirmed by the Wilcoxon Sign Rank test. The results of the Wilcoxon Sign Rank tests are not reported here because of space reasons; however, all of them were significant at the usual $\alpha = 0.05$ level.

Figure 1 shows the boxplots of the distributions of TPR and TNR obtained by faultiness predictors; the blue triangles represent the mean values. TPR quantifies the performance of the predictors applied to actually faulty modules, while TNR represents the performance of the predictors applied to actually non-faulty modules. Figure 1 shows that no model performs best for both TPR and TNR:

- ABOCC$_p$, both with and without SMOTE, has the best TPR and the worst TNR.
- SVM with SMOTE and random classification perform best in terms of TNR, but have low TPR, i.e., they are quite inaccurate when predicting actually faulty modules.
- One-class SVM performs decently with both faulty and non-faulty modules. Nonetheless, it is outperformed by ABOCC$_p$ at predicting faulty modules and by SVM and random classification at predicting non-faulty modules.

ABOCC$_p$ achieves more extreme performances (i.e., better TPR and worse TNR) when trained with datasets balanced via SMOTE.

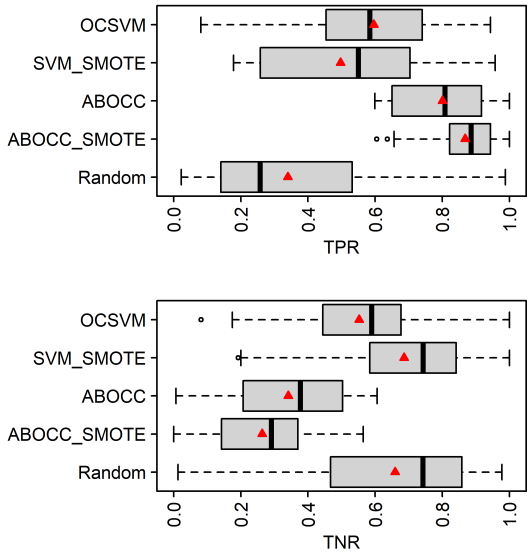

Figure 1. Boxplots of TPR and TNR.

Random classification is quite good at predicting non-faulty modules. This is not surprising, since non-faulty modules are the vast majority in most datasets (the median prevalence is 0.258, according to Table I). Hence, random classification predicts modules non-faulty with high probability, which leads to having high values of EN and TN. Symmetrically, random classification gives low EP and TP, hence it achieves low TPR.

As no model has both the best TPR and TNR, we check which method has the best "overall" performance, in terms of Gmean and MCC (see Section III-C). Figure 2 shows Gmean and MCC boxplots.

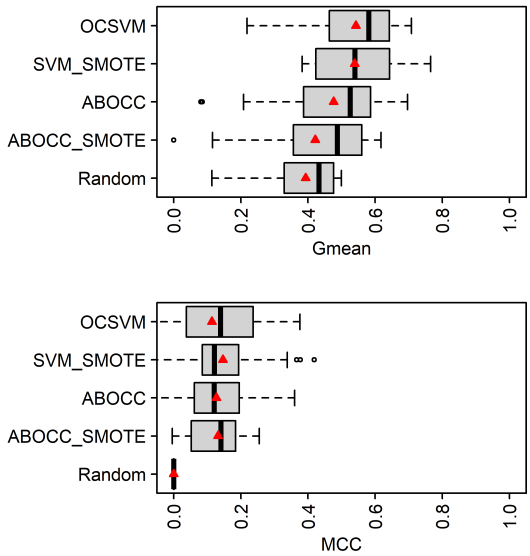

Figure 2. Boxplots of Gmean and MCC.

Coherently with Figure 2, the Wilcoxon Sign Rank tests indicate that i) OCSVM is better than all other methods; ii) SVM and ABOCC$_p$ without SMOTE are equivalent to each

other and both better than ABOCC$_p$ with SMOTE; iii) random classification is worse than all other methods.

It is now interesting to look at the evaluations based on MCC: since MCC computes the effect size of a model with respect to random classification, it indirectly shows how much a method is better or worse than another. In this respect, Figure 2 shows that all models achieve similar improvements over random estimation.

All of the models have relatively low MCC values (means and medians are just above 0.15), which should not be interpreted as indicating bad performance. MCC evaluates models with respect to the random classification, whose performance is not really bad, as Figure 2 shows.

The above results show that none of the methods appears better than the others across the board. Specifically, ABOCC$_p$ appears to be the best of the lot at estimating positive (i.e., defective) modules and the worst at estimating negative (i.e., non-defective) modules. This is a consequence of the "prudential" approach that classifies as faulty the modules that are deemed uncertain by the basic ABOCC.

As for overall performance, ABOCC$_p$ achieves Gmean and MCC values that are quite close to those of the other methods.

Thus, we can conclude that ABOCC$_p$ is competitive with respect to other faultiness prediction methods, but choosing the "best" model depends on what is deemed more important by the users of the predictive models: maximizing TPR (hence minimizing false negatives) or maximizing TNR (hence minimizing false positives).

V. Threats to Validity

Internal Validity. A possibly relevant threat concerns the correctness of the used data. In this respect, the fact that the J&M dataset has been used by many researchers, shows that it is a trustable (or at least widely trusted) dataset.

Another potential internal validity issue concerns the methods through which the machine learning models were built. This threat was mitigated by using well-established SVM and one-class SVM computation programs, as specified in Section III-B. However, we cannot exclude that different parameterizations of the machine learning (ML) models could lead to different results. Similarly, different balancing techniques and parameters could have been used, so that—depending on the nature of the data and the balancedness of the datasets—different results could have been obtained.

Construct Validity. A potential construct validity issue concerns the way faultiness prediction errors were measured and evaluated. To this end, note that, when dealing with software defect prediction, it is hardly possible to identify a performance metric that satisfies completely all the evaluation needs and that fits equally well in all development processes. We addressed this threat by using two basic indicators, TPR and TNR, which provide a clear view of the performance of each model in identifying faulty and, respectively, non-faulty modules. TPR and TNR do not depend on prevalence ρ either, thus making the provided indication less context-dependent. As overall performance indicators, we used Gmean, the geometric mean of TPR and TNR, which is also independent of prevalence ρ, and MCC, which depends on ρ.

External Validity. As for the generalizability of our results, the datasets we used collect modules with a wide range of code features (size, complexity, cohesion, coupling, etc.); nonetheless, it is still possible that some software development environments do not match the situation represented by the datasets we used.

VI. Related Work

One-class classification has raised the interest of researchers in several areas, as witnessed by multiple reviews [11]–[13]. Noticeably, it has been studied for anomaly and defect detection in several industrial domains, not just software development.

The One-class SVM has been tested and compared with other ML techniques in multiple occasions and in different contexts. Shin et al. [14] compared it with the multilayer perceptron for fault detection and classification in electro-mechanical machinery. Mahadevan and Shah [15] compared it with Principal Component Analysis (PCA) and Dynamic Principal Component Analysis (DPCA) for fault detection and diagnosis in process data. Hejazi and Singh [16] compared it with the traditional SVM for the detection of fraudulent credit card transactions. In these studies, the One-class model showed better performance than the more traditional approaches. However, in the research by Shin et al., one-class SVM appeared to be sensitive to the parameters and the choice of kernel.

Moussa et al. [17] conducted a study by comparing the one-class SVM with other ML techniques, including SVM, for cross-version and cross-project defect detection. The one-class model outperformed other traditional techniques for cross-version and cross-project detection, but had poor performance for within-project detection. Ciubotariu et al. [18] compared the performance of traditional SVM and one-class SVM. The one-class models had worse FPR than the traditional SVM, but better TPR. Even though the experiment was not conclusive, they suggest that the joint usage of both approaches, in order to compensate each other weaknesses, should be investigated further. Finally, Shehab et al. [1] compared different one-class SVM, Isolation Forest, and k-NN to their traditional counterparts. Their tests showed that, for datasets with medium to high imbalanced ratio, the one-class variants performed better than the traditional models, with and without balancing, using cross and time-sensitive validation approaches.

Ahmad and Bezawada studied the performance of One-Class Classification by Ensembles of Regression model (OCCER) [19]. This model, originally proposed by Noto et al. [20], consists of multiple regression models, one for each feature of the one-class training set. The test data are labeled as outliers depending on the Outlier Score computed by every regression model. Their test showed that OCCER performs well in relation to other one-class solutions.

VII. Conclusions and Future Work

Ignoring one of the classes during one-class classification leads to a loss of potentially useful information. To exploit

the information embedded in the minority class, though still using one-class estimation, we defined ABOCC, an ensemble method that performs one-class SVM classification twice, first using the majority class, then using the minority class, and finally estimating module defectiveness using both the resulting models. The "prudential" approach ABOCC$_p$ classifies as non-faulty only the modules that are so classified by both models.

ABOCC$_p$ has been tested via an empirical study, in which it has been compared with regular SVM (i.e., with SVM trained using both classes) and one-class SVM. The performances of the considered models were evaluated both globally and considering the ability to detect actual positive and actual negative modules separately. This is important, because in some contexts minimizing false negatives could be more important than minimizing false positives, while in other contexts it could be the opposite. Therefore, in some situations, using the ensemble method ABOCC based on one-class classification can be more advantageous with respect to traditional one class and two-class classification methods.

ACKNOWLEDGMENT

The work reported here was partly supported by Fondo per la Ricerca di Ateneo, Università degli Studi dell'Insubria.

REFERENCES

[1] M. A. Shehab, W. Khreich, A. Hamou-Lhadj, and I. Sedki, "Commit-time defect prediction using one-class classification," *Journal of Systems and Software*, vol. 208, p. 111 914, 2024.

[2] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM trans. on intelligent systems and technology*, vol. 2, no. 3, pp. 1–27, 2011.

[3] K.-R. Müller, S. Mika, K. Tsuda, and K. Schölkopf, "An introduction to kernel-based learning algorithms," in *Handbook of neural network signal processing*, CRC Press, 2018, pp. 4–1.

[4] C.-Y. Yeh and S.-J. Lee, "A kernel-based two-stage one-class support vector machines algorithm," in *Int. Symp. on Neural Networks*, Springer, 2007, pp. 515–524.

[5] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–10.

[6] *Zenodo*, https://zenodo.org/records/6342328.

[7] M. Zhang, B. Xu, and J. Gong, "An anomaly detection model based on one-class svm to detect network intrusions," in *2015 11th Int. Conf. on Mobile Ad-hoc and Sensor Networks (MSN)*, 2015, pp. 102–107.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[9] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[10] L. Lavazza and S. Morasca, "Common problems with the usage of f-measure and accuracy metrics in medical research," *IEEE Access*, 2023.

[11] S. S. Khan and M. G. Madden, "One-class classification: Taxonomy of study and review of techniques," *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014.

[12] S. Alam, S. K. Sonbhadra, S. Agarwal, and P. Nagabhushan, "One-class support vector classifiers: A survey," *Knowledge-Based Systems*, vol. 196, p. 105 754, 2020.

[13] N. Seliya, A. Abdollah Zadeh, and T. M. Khoshgoftaar, "A literature review on one-class classification and its potential applications in big data," *Journal of Big Data*, vol. 8, no. 1, pp. 1–31, 2021.

[14] H. J. Shin, D.-H. Eom, and S.-S. Kim, "One-class support vector machines—an application in machine fault detection and classification," *Computers & Industrial Engineering*, vol. 48, no. 2, pp. 395–408, 2005.

[15] S. Mahadevan and S. L. Shah, "Fault detection and diagnosis in process data using one-class support vector machines," *Journal of process control*, vol. 19, no. 10, pp. 1627–1639, 2009.

[16] M. Hejazi and Y. P. Singh, "One-class support vector machines approach to anomaly detection," *Applied Artificial Intelligence*, vol. 27, no. 5, pp. 351–366, 2013.

[17] R. Moussa, D. Azar, and F. Sarro, "Investigating the use of one-class support vector machine for software defect prediction," *arXiv preprint arXiv:2202.12074*, 2022.

[18] G. Ciubotariu, G. Czibula, I. G. Czibula, and I.-G. Chelaru, "Uncovering behavioural patterns of one: And binary-class svm-based software defect predictors," in *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, Rome, Italy: SciTePress, 2023, pp. 5881–588.

[19] A. Ahmad and S. Bezawada, "One-class classification by ensembles of regression models–a detailed study," *arXiv:1912.11475*, 2019.

[20] K. Noto, C. Brodley, and D. Slonim, "Anomaly detection using an ensemble of feature models," in *2010 IEEE International Conference on Data Mining*, 2010, pp. 953–958.