



Effective software defect prediction using support vector machines (SVMs)

Somya Goyal^{1,2}

Received: 28 July 2020 / Revised: 19 August 2021 / Accepted: 25 August 2021 / Published online: 14 September 2021
© The Society for Reliability Engineering, Quality and Operations Management (SREQOM), India and The Division of Operation and Maintenance, Lulea University of Technology, Sweden 2021

Abstract Software defect prediction (SDP) plays a key role in the timely delivery of good quality software product. In the early development phases, it predicts the error-prone modules which can cause heavy damage or even failure of software in the future. Hence, it allows the targeted testing of these faulty modules and reduces the total development cost of the software ensuring the high quality of end-product. Support vector machines (SVMs) are extensively being used for SDP. The condition of unequal count of faulty and non-faulty modules in the dataset is an obstruction to accuracy of SVMs. In this work, a novel filtering technique (FILTER) is proposed for effective defect prediction using SVMs. Support vector machine (SVM) based classifiers (linear, polynomial and radial basis function) are designed utilizing the proposed filtering technique over five datasets and their performances are evaluated. The proposed FILTER enhances the performance of SVM based SDP model by 16.73%, 16.80% and 7.65% in terms of accuracy, AUC and F-measure respectively.

Keywords Defect prediction · Class imbalance · Support vector machine (SVM) · ROC · AUC · F-measure

1 Introduction

Software defect prediction (SDP) is an essential activity in software project management as it is dedicated to enhancing the quality of software product. It has become non-separable part of software development process and machine learning based classifiers have found huge application in SDP. Such classifiers predict in advance those modules which will be requiring more testing efforts and hence this prediction catalyzes the debugging process. The awareness of faulty modules in prior allows the tester to perform effective testing and overall quality of the product is improved. The faults which remain veiled during the testing phase may become defects in future and can incur heavy maintenance cost in operational times. Such faults which remained undetected during development phases have already played havoc in Software industry becoming defect during the operational phases of software like NASA Mass Climate Orbiter (MCO) spacecraft worth \$125 million lost in the space due to small data conversion bug (NASA 2015).

Machine learning techniques are being widely accepted in the software industry for early defect prediction. It includes neural networks, decision trees, Bayesian approach, and support vector machines (Kumar et al. 2018; Goyal 2020; Goyal and Bhatia 2019, 2020a, b). SVMs have seized a lot of attention from researchers in the SDP domain due to its speed and better performance with small datasets (Cai et al. 2019; Wang et al. 2021; Rong et al. 2016; Jaiswal and Malhotra 2018; Erturk and Sezer 2015). The performance of these classifiers solely depends upon the training dataset fed to them. Sometimes, the dataset is out of class-balance in terms of the count of modules belonging to faulty and non-faulty categories. Class

✉ Somya Goyal
somyagoyal1988@gmail.com

¹ Manipal University Jaipur, Jaipur, Rajasthan 303007, India

² Guru Jambheshwar University of Science & Technology, Hisar, Haryana 125001, India

balance means $|faulty| = |non-faulty|$; where $|faulty|$ denotes the count of fault-prone modules and $|non-faulty|$ denotes the count of the clean modules without any faults. But if $|faulty| \neq |non-faulty|$, then there is loss of balance, it implies one of the classes has more instance count than the other one. It is denoted as class imbalance issue in terms of classification. In this situation of class-imbalance, the classification accuracy of the SVM classifiers is threatened.

In SDP, the category of faulty modules is of high significance, and this is the class which is scarce too. It implies that the class ‘faulty’ has lesser instances than the class ‘non-faulty’. The degree of imbalance can be computed as the imbalanced ratio using Eq. (1):

$$IR = \frac{\text{total count of majority class instances}}{\text{total count of minority class instances}} = \frac{|non-faulty|}{|faulty|} \quad (1)$$

The value of IR equals to 1; shows there exists class-balance. The IR value higher than unity reflects that ‘faulty’ class which is more valuable in software development process has become minority class. This class imbalance (Guo et al. 2017; Chen et al. 2018) situation causes biasing while training the SVM Classifiers which in turn results into the ignorance of faulty data-points. Hence, the overall classification accuracy is adversely impacted due to class-imbalance nature of dataset.

1.1 Motivation

In the domain of SDP, multiple approaches have been deployed at the data level (Felix and Lee 2019; Cai et al. 2019; Kaur and Gossain 2019; Malhotra and Kamal 2019) to improve accuracy of ML classifiers, by resolution of class imbalance problem. But these are dependent either on the datasets or on the techniques used to solve the problem. An accurate software defect prediction in the class-imbalance condition is still an open problem.

A novel filtering technique is proposed for better accuracy of SVM classification based SDP models. This study is directed to attain following research goals:

- G1-To devise a novel filtering technique (FILTER) and assess the effectiveness of FILTER.
- G2-To build SVM based SDP prediction model with variations in the kernel over the FILTERed dataset.
- G3-To evaluate the accuracy of proposed models empirically and to find which prediction model outperforms other models.

1.2 Contribution

This work is contributing to improve the prediction power of SVM based SDP models by filtering the training dataset using the proposed FILTER. This work contributes a novel technique (FILTER) to find a more balanced dataset and hence improves the performance of SVM classifiers to predict in advance the software modules which can be faulty.

1.3 Organization

The paper is organized as follows. Section 2 covers the current state-of-the-art and review of the literature. The research methodology is explained, and research questions are formulated under Sect. 3. The experimental setup along with the datasets and evaluation metrics in use are given in Sect. 4. In Sect. 5, the experimental results are reported and analysed to answer the research questions. The conclusions are drawn in Sect. 6 with remarks on future scope of work.

2 Related works

This section discusses the literature work done in the field of SDP using SVM and filtering methods to achieve accurate models for Software Defect Prediction (SDP). The current state-of-art is summed as Table 1.

From the literature survey, it is found that SVM has potential for software defect prediction (Wang and Yao 2013; Siers and Islam 2015; Goyal 2021a, b; Wang et al. 2021; Huda et al. 2018). NASA and PROMISE Data Repository are the most popular public data sources. Apparently, 67% of total research work has been done using these two repositories (Rathore and Kumar 2019; Yang et al. 2017) carried out their research using Object-Oriented metrics. Complexity metrics are the third largest adopted metrics (Rathore and Kumar 2017; Ozakıncı and Tarhan 2018; Song et al. 2018; Chen et al. 2019; Son et al. 2019; Tsai et al. 2019). Wang and Yao (2013) proposed random under-sampling as data pre-processing method to tackle class imbalance problem. Chen et al. (2019) proposed a SWAY method and confirmed experimentally that under-sampling is essential to optimize the performance of SDP models. Tsai et al. (2019) proposed modified random under-sampling using cluster-based instance selection to select the data-points to be removed from majority class. Rao and Reddy (2020) devised an algorithm to under-sample the data-points and handled the class-imbalance effectively. Sun et al. (2020) stated that under-sampling attains a balance between two classes and proposed a ranking method to under-sample the data-points.

Table 1 State-of-the-art: SDP using SVMs; SDP with filtering techniques

S.no	Year	Study	Technique	Dataset	Performance metrics	Observation drawn (Goyal, S.)
1	2012	Afzal et al. (2012)	Sampling Techniques	PROMISE Software Engineering repository data	AUC, ROC	<i>Comparative study of sampling methods deployed for SDP</i>
2	2013	Wang and Yao (2013)	RUS, SMOTE-Resampling	PROMISE Software Engineering repository data	AUC, ROC, G-Mean	<i>Data space is optimized with sampling</i>
3	2014	Ma et al. (2014)	RUS	PROMISE Software Engineering repository data	AUC, ROC	<i>Emphasised on usage of Random Undersampling for classification problem-SDP instance</i>
4	2017	Yang et al. (2017)	Random Under Sampling	OSS (Bugzilla)	F1-score	<i>Improved results than baseline models</i>
5	2018	Chen et al. (2019)	Random Under sampling	PROMISE	G-mean, AUC	<i>Results are improved with under sampling and ensembles</i>
6	2018	Huda et al. (2018)	Oversampling	PROMISE	AUC, Accuracy, Recall	<i>Imbalance is dealt effectively with oversampling</i>
7	2019	Cai et al. (2019)	Under Sampling + SVM	PROMISE	pd, pf, G-mean	<i>Results are improved with under sampling</i>
8	2020	Sun et al. (2020)	Undersampling	PROMISE Software Engineering repository data	AE, RE, PRED(l)	<i>Proposed undersampling brings better results</i>
9	2020	Rao and Reddy (2020)	Undersampling	NASA MDP	F-Measure	<i>Proposed undersampling brings better results</i>
10	2021	Goyal (2021a)	Filtering Technique	NASA	ROC, AUC, Accuracy, PD	<i>Proposed technique performed better for imbalanced data</i>
11	2021	Goyal (2021b)	Neighbourhood undersampling	NASA	AUC, Accuracy	<i>Proposed method is suitable to handle class imbalance</i>
12	2015	Erturk and Sezer (2015)	SVM	PROMISE Software Engineering repository data	AUC-ROC	<i>Parameter reduction is done beautifully in a practical way</i>
13	2016	Rong et al. (2016)	SVM	NASA, PC1, PC4, MC1	AUC, G-mean	<i>Feature selection techniques are reported with their effect on SVMs</i>
14	2018	Jaiswal and Malhotra (2018)	SVM	NASA data sets	Accuracy	<i>Comparative study of ML techniques including SVMs</i>
15	2020	Goyal and Bhatia (2020b)	SVM	NASA	ROC, AUC, Accuracy, PD	<i>Comparative analysis among five most popular ML classifiers is made</i>
16	2021	Wang et al. (2021)	SVM	NASA	MCC, AUC, F-measure	<i>Proposed model is robust to dataset availability</i>

3 Research methodology

This section reports the research methodology followed to conduct this study. The two most basic strategies to handle class-imbalance are data level solution and algorithmic solution (Guo et al. 2017). In this paper, I propose SDP classifier with a novel filtering technique to improve the Imbalanced Ratio (IR) and SVM classifier which is robust

to handle class imbalance. Total six models are being developed which are as follows—(1) SVM-Linear Kernel without filtering, (2) SVM-Linear Kernel with filtering, (3) SVM-RBF Kernel without filtering, (4) SVM-RBF Kernel with filtering, (5) SVM-Polynomial Kernel without filtering, and (6) SVM-Polynomial Kernel with filtering.

3.1 Research questions

To steer the research in this above-mentioned direction, the following research questions are formulated:

RQ1 Does the proposed filtering technique (FILTER) improve the condition of dataset to be used for the training of the SVM based SDP classifier?

RQ2 Which SVM variant trained with the FILTERed dataset, performs the best as a Software Defect Prediction model?

RQ3 Does the answer to above mentioned research question RQ2 bear the statistical proof?

3.2 Proposed filtering technique (FILTER)

The proposed filtering technique filters out the data-points from majority class. It maximizes the visibility of minority data-points by achieving better balance in the count of the majority class and minority class.

In SDP, the minority class is the set of data instances which are faulty and the majority class is the set of data instances which are non-faulty.

Assume $\{(x_{ij}, y_i)\}$ denotes the dataset \mathcal{F} with m attributes and n instances (data-points) for SDP as 2-class classification problem; where $1 \leq i \leq n$ (instances), $1 \leq j \leq m$ (attributes). x_{ij} denotes j^{th} attribute for i^{th} instance in the dataset. $X_i = \{x_{ij} \mid 1 \leq j \leq m \text{ for all } i\}$. y_i denotes the class for i^{th} instance in the dataset and $y_i \in \{\text{faulty, non-faulty}\}$.

The proposed technique (FILTER) identifies few non-faulty data-points strategically and filters these out. In this way, the IR value is minimized to achieve more balanced class distribution. The strategy to select the non-faulty data-points for filtering is explained below as **Algorithm_1**.

The key idea is to locate the non-faulty instances which are in the proximity of faulty instances. The presence of large number of non-faulty instances in close surroundings of faulty instances, hinders the visibility of faulty instances. Subsequently, the training of SDP classifiers gets negatively affected and so as the classification accuracy of predictors.

Hence, the proposed FILTER removes selected few non-faulty data-instances to bring balance in the training dataset.

Algorithm_1: Proposed Filtering Technique (FILTER)

Inputs: Training Dataset $\mathcal{F} = \text{Faulty} \cup \text{Non_Faulty}$, where *Faulty* denotes those instances which are from class (faulty) and *Non_Faulty* denotes those instances which are from class (non-faulty).

Output: \mathcal{F}' , the filtered dataset after the application of FILTER

Begin:

Step 1. Set **Filter_Proximity**[]:= {}; **In_Proximity**[]:= {}; **Proximity_size**:=0;

To_Filter[]:= {}; **P**:=Imbalanced Ratio; \mathcal{F}' := {}.

Step 2. for each data-point **faulty** instance from the **training dataset** \mathcal{F} say $\langle X_i, y_i \rangle$

Step 3. find **P** instances in proximity of $\langle X_i, y_i \rangle$ using Euclidean distance metric

Step 4. **In_Proximity**:= instances obtained from step3

Step 5. **Filter_Proximity** := Search (**In_Proximity**, **Non_Faulty** instances)

Step 6. **Proximity_size** := count (**Filter_Proximity**)

Step 7. if **Proximity_size** greater than **Zero**

Step 8. Sort (**Proximity_size**, distance_metric, non-decreasing order)

Step 9. **To_Filter** := **Filter_Proximity** [1... (((**Proximity_size**-1) mod [**P**/3])) + 1]

Step 10. Update **Non_faulty**:= **Non_faulty** - **To_Filter**;

End if;

End for;

Step 11. Assign \mathcal{F}' := **Faulty** \cup **Non_Faulty**;

Step 12. return \mathcal{F}' ;

End.

The FILTER works as follows:

- (1) For each faulty instance of training dataset, its proximity is scanned for P closest instances; where P is imbalance ratio of the original dataset \mathcal{F} . (Refer Step 3 and Step 4)
- (2) From the data instances obtained, non-faulty data-instances are searched for (Refer Step 5)
- (3) Sort the non-faulty instances in proximity in non-decreasing order by the distance metrics (Refer Step 8)
- (4) The instances to be filtered are identified by applying Step 9. The top $[(((\text{Proximity_size}-1) \bmod P/3) + 1)]$ instances from the sorted list are removed.
- (5) Updated dataset is returned with more balanced class distribution.

FILTER is demonstrated for training subset from KC1 dataset. Figure 1 shows the original dataset with $IR \approx 6.27$. Figure 2 highlights the non-faulty instances which qualify as Filter_Proximity. Figure 3 shows the filtered dataset with reduced $IR \approx 5.30$.

The proposed technique FILTER is effective for SVM classifiers due to robust nature of SVM with availability of small datasets (Wu et al. 2007). It filters out at maximum ‘one-third of IR’ non-faulty data-instances. Hence, it minimizes the loss of information along with improving the visibility of faulty data instances.

4 Experimental set-up

In previous section, the research methodology is explained. Now, this section covers the experimental design, set-up and the description of datasets utilized for the experiment.

4.1 Experimental design

The proposed experimental design is depicted in Fig. 4 which is logically divided into four phases: Phase-I comprises of dividing the dataset into training and testing data subsets. The dataset is divided randomly in two partitions with 80% and 20% of total data-points. The partition with 80% data-samples is to be used as training dataset and the portion with 20% data-samples is to be used as testing dataset for the classification algorithms; Phase-II includes filtering of training dataset to balance the class distribution by applying FILTER technique; Phase-III is training the SDP models using the FILTERed dataset and making the predictions. Phase-IV is testing the performance of SDP models and drawing comparative analysis.

4.1.1 Description to dataset and software metrics used

The experiment is designed using five fault prediction datasets named CM1, KC1, KC2, PC1, and JM1. The Data is collected from NASA projects using McCabe metrics which are available publicly in the PROMISE repository (Sayyad and Menzies 2005; PROMISE). Table 2 describes the dataset used for this study (Rao and Reddy 2020; Rong et al. 2016; Chen et al. 2018).

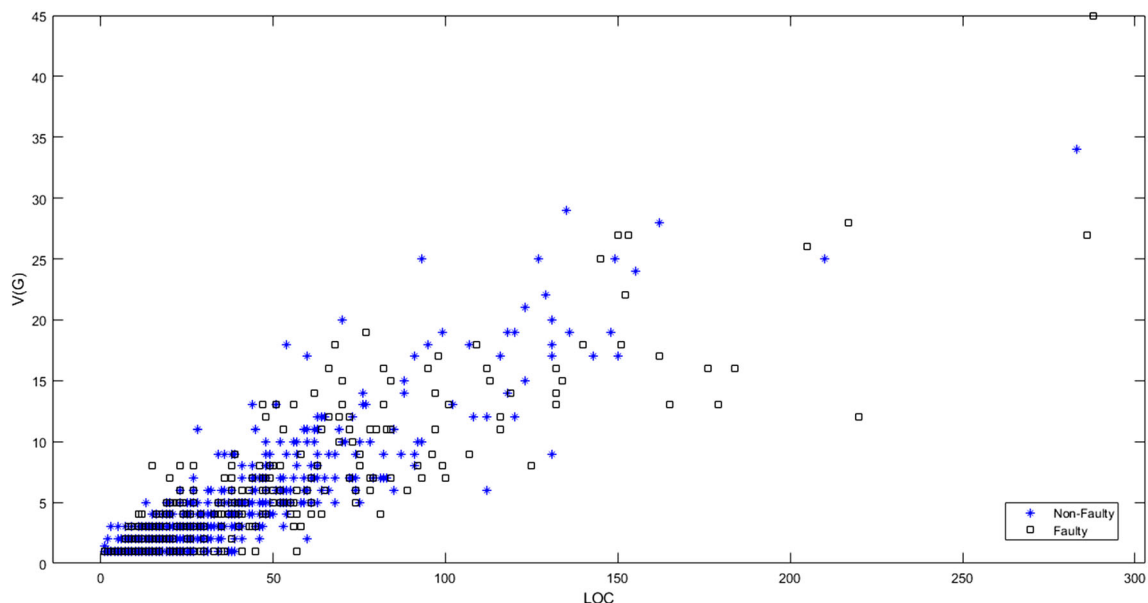


Fig. 1 KC1 DATASET ($IR \approx 6.27$)

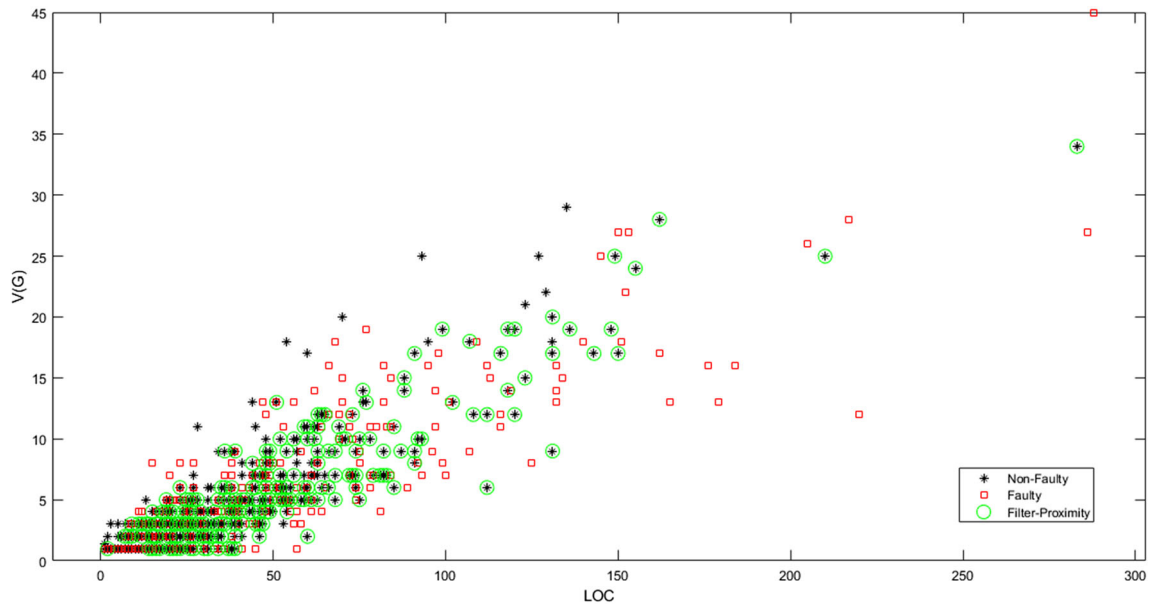


Fig. 2 Demonstration of Filter_Proximity instances

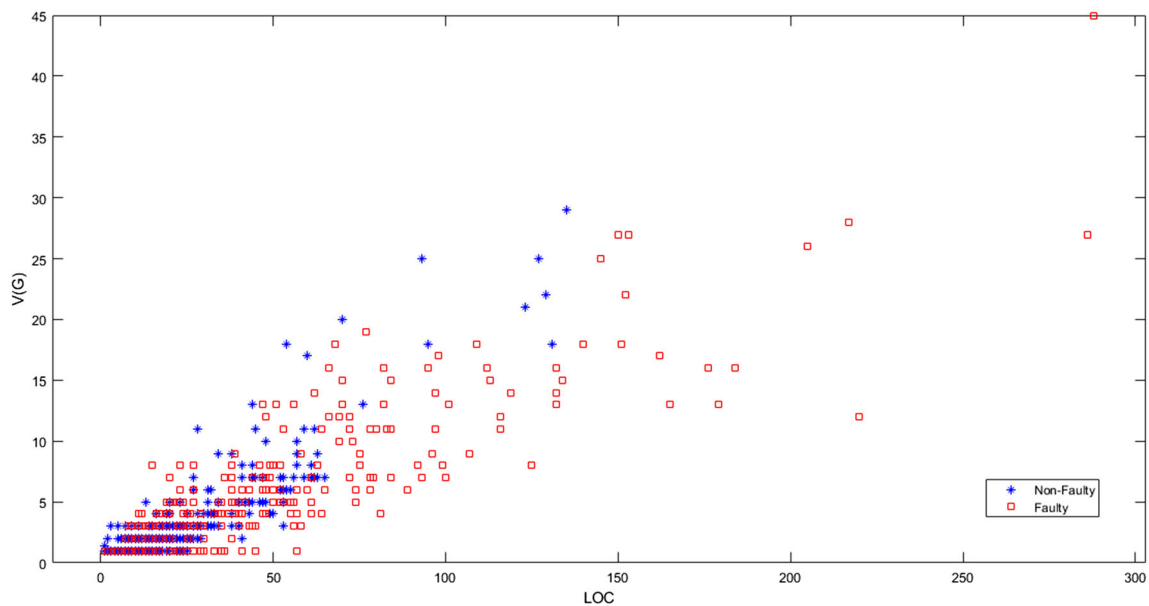


Fig. 3 Updated KC1 dataset after applying FILTER ($IR \approx 5.30$)

The datasets are comprising of the most popular static code metrics (Huda et al. 2018) including McCabe's and Halstead's complexity metrics (Thomas 1976; Menzies et al. 2007). All five datasets possess 21 metrics and 1 response variable (tabulated as Table 3). The effectiveness of these metrics is empirically proven and accepted (Song et al. 2018; Son et al. 2019; Tsai et al. 2019).

4.1.2 Parameter setting for SDP models

Phase-III involves the training of our SVM classifiers using the training dataset received from the previous phase and Phase IV tests the trained classifier over the test dataset. The parameter settings for all three classifiers are given as Table 4.

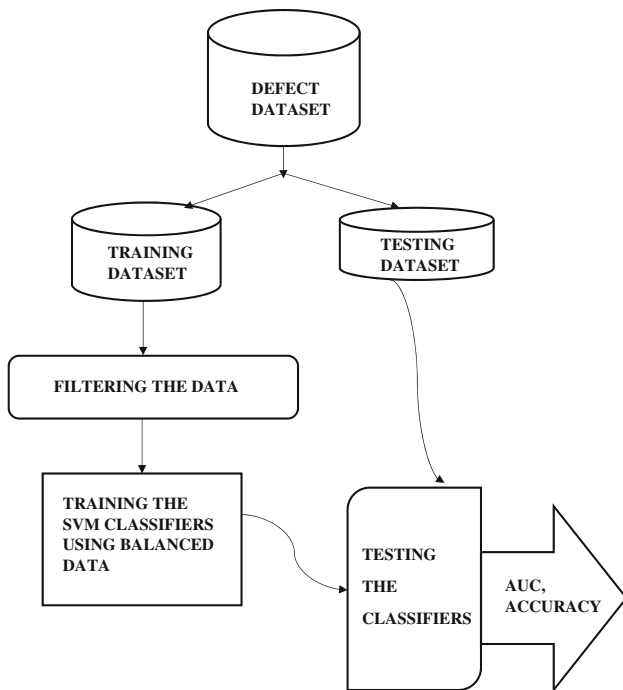


Fig. 4 Experimental design of proposed work

4.2 Performance evaluation criteria

The performance of proposed SVM classifiers is evaluated using the widely accepted evaluation metrics namely Confusion matrix, ROC, AUC, Accuracy and F-measure. The opted evaluation criteria for this paper are explained below:

- *Confusion matrix* contains information about the actual values and predicted values for the output class variable in the form of a matrix (as in Fig. 5). The predicted values for the classifications done by the fault prediction model are compared and performance is evaluated (Kumar et al. 2018).
- The *sensitivity (recall)* of the model is defined as the percentage of the ‘buggy’ modules that are predicted ‘buggy’ and the *specificity* of the model is defined as the percentage of the ‘clean’ modules that are predicted ‘clean’. These are computed as Eqs. (2) and (3).

Table 3 Metrics set in used dataset

#	Feature name	Feature definition
1	loc	LOC (McCabe’s)
2	v_(g)	Cyclomatic Complexity (McCabe’s)
3	ev_(g)	Essential Complexity (McCabe’s)
4	iv_(g)	Design Complexity (McCabe’s)
5	_n’	Operator and Operand total (Halstead)
6	_v’	Volume (Halstead)
7	_l’	Program Length (Halstead)
8	_d’	Difficulty (Halstead)
9	_i’	Intelligence (Halstead)
10	_e’	Effort (Halstead)
11	_b’	(Halstead)
12	_t’	Time (Halstead)
13	IOC_Code	LOC (Halstead)
14	IOC_Comment	Line of Comments (Halstead)
15	IoFBlank	Line of Blanks (Halstead)
16	IoFCode + Comment	Line of Code and Comment (Halstead)
17	Uniq-Op	No. of unique operator
18	Uniq-Opnd	No. of unique operand
19	Total-Op	Total operator
20	Total-Opnd	Total operands
21	Branch-Count	Count of Branch
22	Reponse_Variable	{Faulty,Non-Faulty}

$$sensitivity \text{ (or recall)} = \frac{true\ positive}{true\ positive + false\ negative} \tag{2}$$

$$specificity = \frac{true\ negative}{true\ negative + false\ positive} \tag{3}$$

- *Receiver Operating Characteristics (ROC) curve* is analyzed to evaluate the performance of the prediction model. During the development of the ROC curves, many cutoff points between 0 and 1 are selected; the sensitivity and $(1 - specificity)$ at each cut off point is calculated (see Fig. 6). It is interpreted that closer the classifier gets to the upper left corner, better is its performance. To compare the performance of classifiers, the one above the other is considered better (see Fig. 7).

Table 2 Dataset description

#	Data-set name	# total instances	#buggy instances	# clean instances
1	CM1	498	49	449
2	KC1	2109	326	1783
3	KC2	522	107	415
4	PC1	1109	77	1032
5	JM1	10,885	2106	8779

- *Area Under the ROC Curve (AUC)* is a combined measure of the sensitivity and specificity. It gives the averaged performance for the classifier over different situations. $AUC = 1$ is considered ideal.
- *Accuracy* is the measure of the correctness of prediction model. It is defined as the ratio of correctly classified instances to the total number of the instances (Hanley and McNeil 1982) and computed as Eq. (4)

$$Accuracy = \frac{true\ positive + true\ negative}{true\ positive + false\ positive + true\ negative + false\ negative} \tag{4}$$

- F-measure is harmonic mean of precision and recall and computed as Eq. (5)

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5}$$

The above criteria are widely used in the literature for the evaluation of predictor performance (Goyal and Bhatia 2020b; Rathore and Kumar 2017; Ozakıncı and Tarhan 2018; Song et al. 2018; Chen et al. 2019; Son et al. 2019; Tsai et al. 2019). This set of metrics is appropriate and suitable for comparing the performance of multiple SDP classifiers.

5 Result analysis and discussion

In this section, the results are being reported which are obtained from the experimental work and answers are drawn to the research questions mentioned in the former section of this paper. All three research questions are discussed one by one in this section to find the answers.

5.1 Proposed filtering technique (FILTER) improves the condition of dataset to be used for the training of the SVM based SDP classifier—finding answer to RQ1

The author investigates the impact of FILTER technique on training datasets to find the answer to first research

question of this work; RQ1. Table 5 reports the experimental results of applying the proposed filtering technique i.e. FILTER over all five selected datasets.

Table 6 reports the accuracy of classifiers with and without the application of filtering technique, so that the impact of proposed filtering technique can be measured on the performance of SDP classifiers (also plotted as Fig. 8). (Bold values highlight the best values).

The inferences drawn are:

- FILTER is effective to reduce the IR by 17.3%, 20.9%, 15.4%, 24.1% and 16.3% for CM1, JM1, KC1, KC2 and PC1 dataset respectively.
- The average value of reduction in IR value is 18.81%. It implies that on average FILTER will reduce the $False_Negative_{Faulty \gg Non-Faulty}$ by 18.81%.
- FILTERed dataset results in improved accuracy of SVMs. It increases the performance by 9.32% for SVM-Linear, by 16.74% for SVM-RBF and by 14.06% for SVM-Polynomial SDP models.

Answer to RQ1: From the above experimental results, analysis and inferences; YES, the proposed filtering technique (FILTER) improves the condition of dataset to be used for training the SVM based SDP classifier.

ACTUAL CLASS	Classifier	
BUGGY	True_Positive _{Buggy>>Buggy}	False_Negative _{Buggy>>Clean}
CLEAN	False_Positive _{Clean>>Buggy}	True_Negative _{Clean>>Clean}
Confusion Matrix	BUGGY	CLEAN
	PREDICTED CLASS	

Fig. 5 Confusion matrix

Table 4 Parameter settings for classifier

Classifier	Parameter: value
Support vector machine (SVM) with Linear Kernel	C = 10
Support vector machine (SVM) with RBF Kernel	C = 10, $\gamma = 0.1$
Support vector machine (SVM) with Polynomial Kernel	C = 10, $\gamma = 0.01$

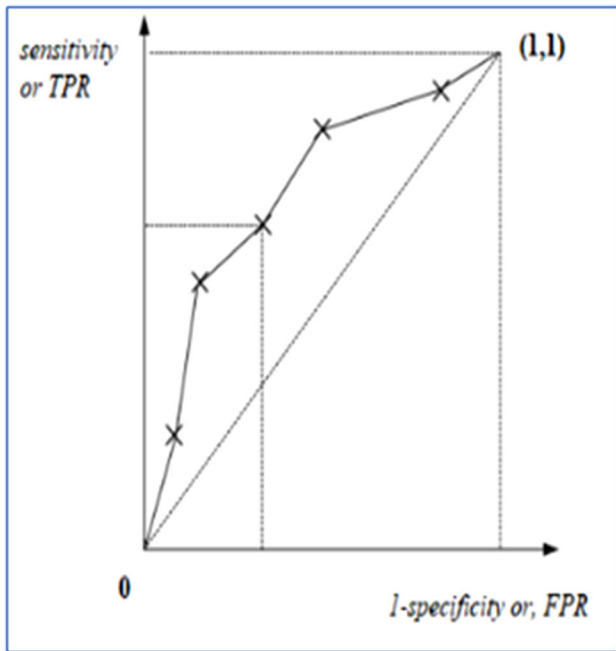


Fig. 6 ROC

5.2 Best SVM variant trained with the FILTERed dataset, for defect prediction—finding answer to RQ2

To answer RQ2, the performance of all 03 variants of SVM classifiers are evaluated over 5 datasets using evaluation criteria mentioned in former section. The observation is recorded for both scenario—(1) without the application of FILTER and (2) with the application of FILTER. Tables 7

Table 5 Impact of application of proposed filtering technique on the datasets

Dataset	#Faulty	#Non-Faulty	#Total	IR
CM1	35	364	399	10.4
FILTER_CM1	35	301	336	8.6
JM1	1045	7663	8708	7.33
FILTER_JM1	1045	6061	7106	5.8
KC1	232	1456	1688	6.27
FILTER_KC1	232	1231	1463	5.30
KC2	68	349	417	5.13
FILTER_KC2	68	265	333	3.89
PC1	67	821	888	12.2
FILTER_PC1	67	687	754	10.2

Table 6 Accuracy measure for all classifiers with and without filtering technique on the datasets

Dataset	Classification algorithm		
	SVM-Linear	SVM-RBF	SVM-Polynomial
CM1	0.791	0.77	0.75
Filtered CM1	0.83	0.881	0.825
JM1	0.764	0.801	0.791
Filtered JM1	0.821	0.967	0.867
KC1	0.83	0.821	0.80
Filtered KC1	0.94	0.973	0.962
KC2	0.812	0.863	0.815
Filtered KC2	0.923	0.976	0.902
PC1	0.89	0.843	0.79
Filtered PC1	0.954	0.987	0.945

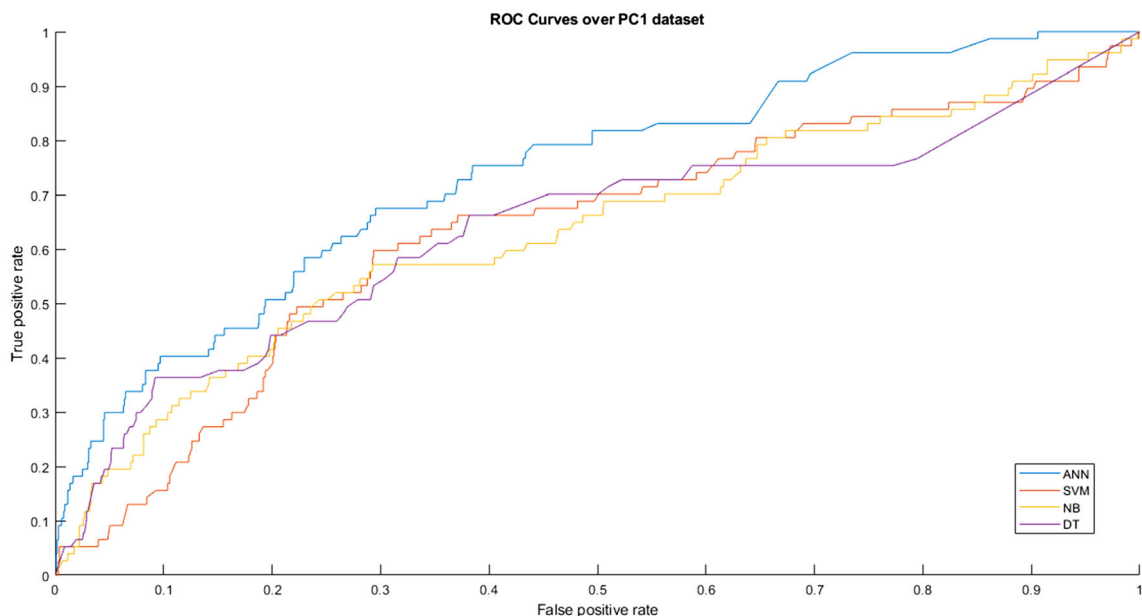


Fig. 7 Multiple ROC

Fig. 8 Accuracy measure for all classifiers with and without filtering technique on the datasets

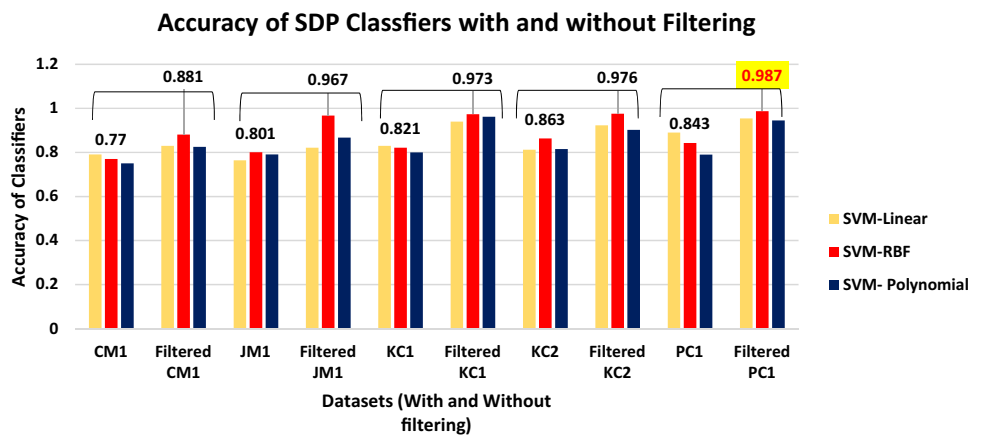


Table 7 AUC measure for all classifiers with and without filtering technique on the datasets

AUC	Classification algorithm		
	SVM-Linear	SVM-RBF	SVM-Polynomial
Dataset			
CM1	0.754	0.787	0.76
Filtered CM1	0.8216	0.845	0.723
JM1	0.723	0.919	0.80
Filtered JM1	0.787	0.979	0.853
KC1	0.768	0.781	0.643
Filtered KC1	0.829	1	0.787
KC2	0.835	0.853	0.801
Filtered KC2	0.901	1	0.845
PC1	0.735	0.79	0.74
Filtered PC1	0.776	1	0.798

Table 8 F-Measure for all classifiers with and without filtering technique on the datasets

F-Measure	Classification algorithm		
	SVM-Linear	SVM-RBF	SVM-Polynomial
Dataset			
CM1	0.8216	0.91	0.723
Filtered CM1	0.856	0.967	0.821
JM1	0.801	0.901	0.85
Filtered JM1	0.821	0.968	0.903
KC1	0.81	0.884	0.85
Filtered KC1	0.901	0.941	0.907
KC2	0.814	0.821	0.782
Filtered KC2	0.861	0.89	0.812
PC1	0.805	0.891	0.844
Filtered PC1	0.934	0.978	0.928

and 8 report the AUC and F-measure for all 06 models. It is observed that out of all classifiers, the SVM-RBF classifier with proposed filtering technique performs best for all 5 datasets in terms of AUC, Accuracy and F-measure.

From the results, it is clear that SVM-RBF over FILTERED dataset performs best (shown with bold values in Tables 7 and 8) among all SDP models based on SVM variants.

It is desirable to gain deep insight into the obtained results before reporting the final answer to RQ2.

At initial step, the performance of all three variants of SVM is compared over 5 datasets. The recorded values for AUC, Accuracy and F-measure are shown as box plots in Figs. 9, 10 and 11 respectively. It is seen that the classifier ‘SVM_RBF’ shows the highest median value of ‘1’ for the criteria (AUC, accuracy, F-measure) and minimum outliers.

Hence, it can be inferred that SVM variant with RBF kernel is best among all three SVM variants.

At next level, the behaviour of SDP model built using SVM_RBF variant is observed ‘with the application of FILTER’ in contrast to ‘without the FILTER’.

For this agenda, the ROC curves for SVM-RBF ‘with the application of FILTER’ in contrast to ‘without the FILTER’ are plotted in Fig. 12. Figure 12 has 5 sub-figures, one for each of the five datasets namely Fig. 12a–e. It shows that the classifier with ROC curve closer to top-left corner is better performer. Through, all sub-figures, on the average the ROC for FILTERED dataset with SVM-RBF classifier is above the ROCs of SDP model without FILTER.

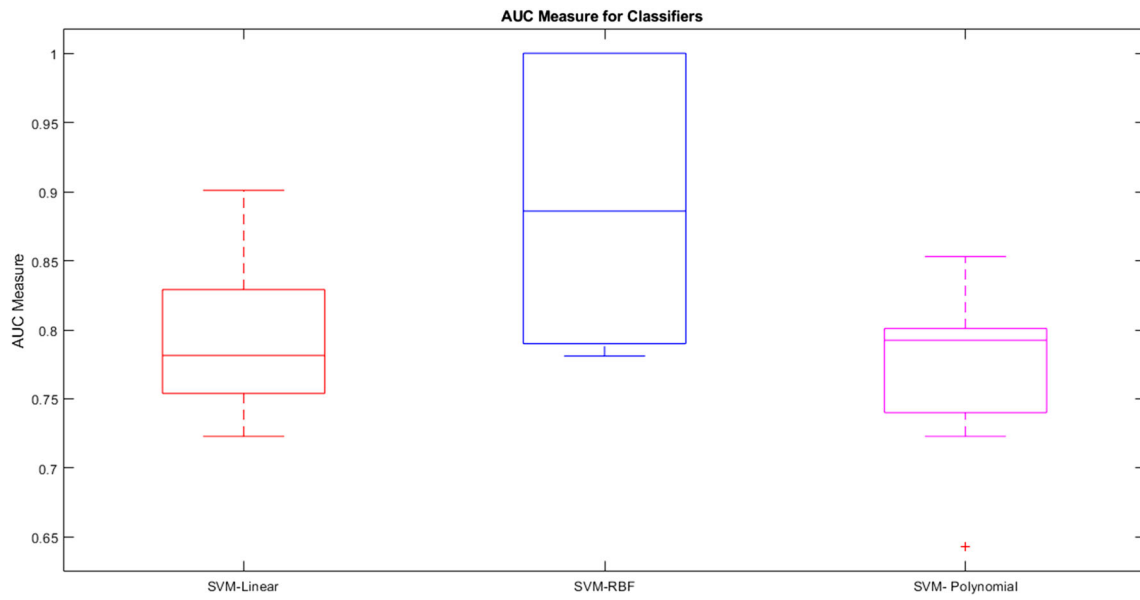


Fig. 9 Box-plots for performance over AUC

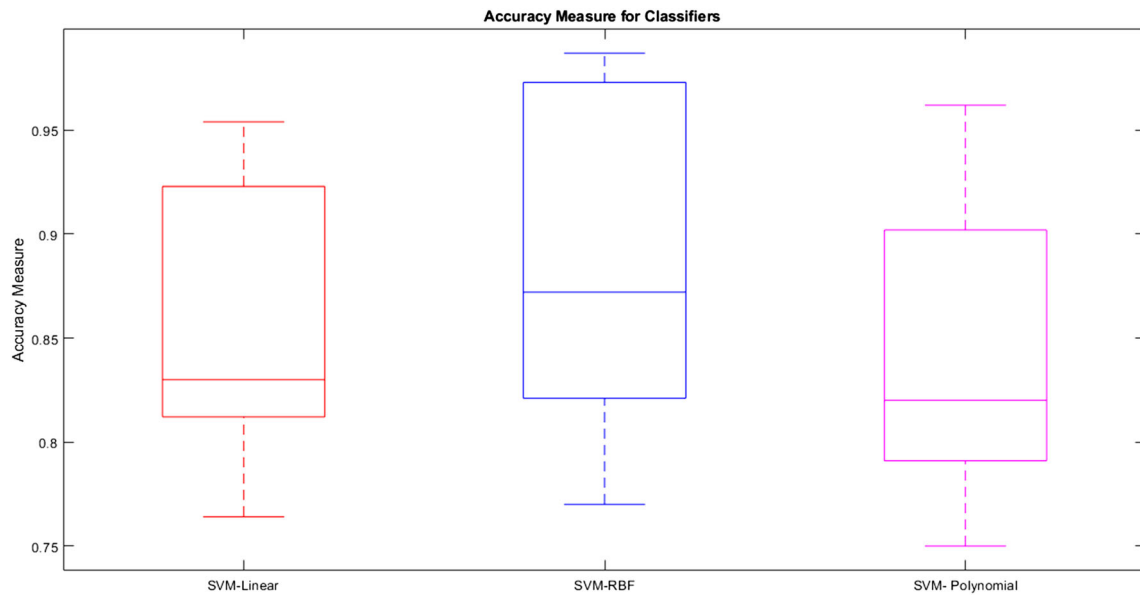


Fig. 10 Box-plots for performance over accuracy

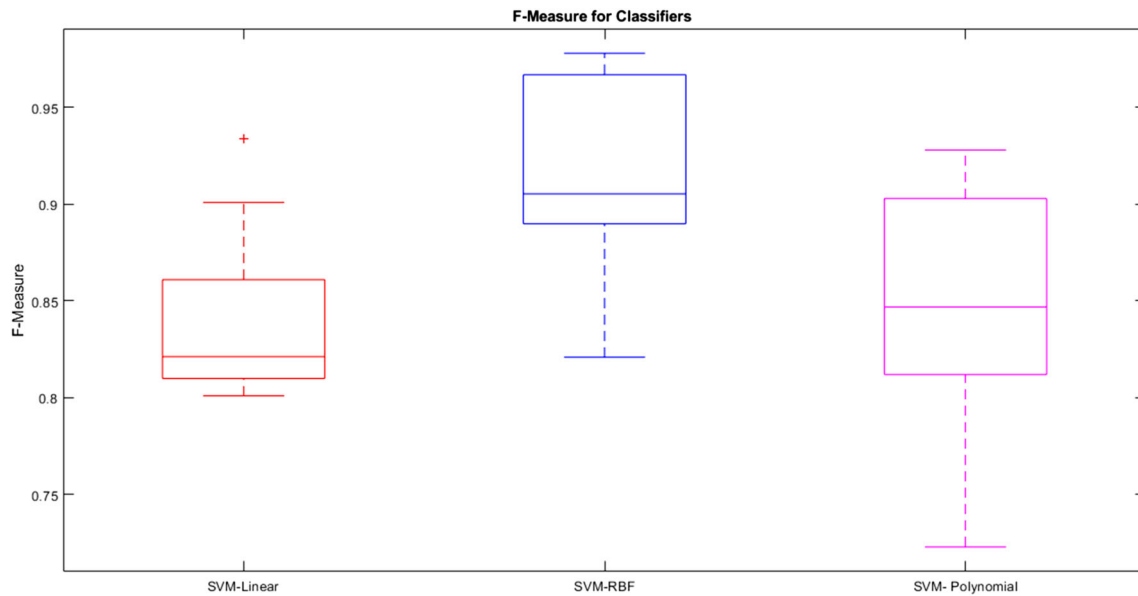


Fig. 11 Box-plots for performance over F-measure

Here the ‘SVM-RBF with FILTERed dataset’ can be inferred as the best SDP model with following observations made:

- i. Minimum Type-II error; reaching to the value of *zero*
- ii. Highest value for AUC, F-measure, accuracy metric and the closest ROC curve to the top left corner.
- iii. Generalized behaviour over five public datasets.
- iv. The application of FILTER improves the performance of SVM-RBF model by 16.73%, 16.8% and 7.65% in terms of accuracy, AUC and F-measure.

Answer to RQ2: From the above experimental results, analysis and inferences; *SVM variant with RBF kernel trained with the FILTERed dataset, is the best SDP model.*

5.3 Statistical evidence for being the best classifier for software defect prediction—finding answer to RQ3

Now, the investigation begins for the statistical evidence in support of the answer reported for RQ2. The statistical evidence is essential to confirm that answer to RQ2; the proposed SVM-RBF with FILTER is the best performer; is not subjective, or due to just by the chance. The hypothesis testing framework is adopted, and non-parametric tests are performed to find statistical evidence. Non-parametric tests

namely Friedman tests are found to be suitable for our research work as we want to compare the performance of multiple classification algorithms over several datasets (Lehmann and Romano 2008; Ross 2005).

In Figs. 13, 14 and 15, the test statistic p values for Friedman test at confidence level of 95% are shown over AUC, accuracy and f-measure metrics respectively. It is to be noted that p -static value less than 0.05. Hence, it can be inferred that the performance of SVM-RBF with FILTER in comparison to rest of the classifiers over 5 datasets in terms of AUC, accuracy and f-measure is statistically significant.

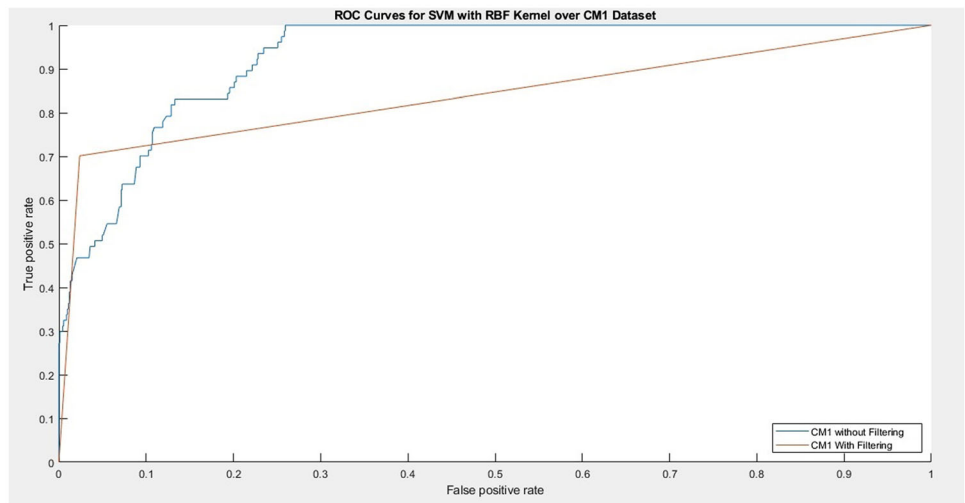
Answer to RQ3: From the reported results, analysis and drawn inferences-*YES, the proposed SVM-RBF with FILTER statistically outperforms other classifiers.*

6 Conclusion and future work

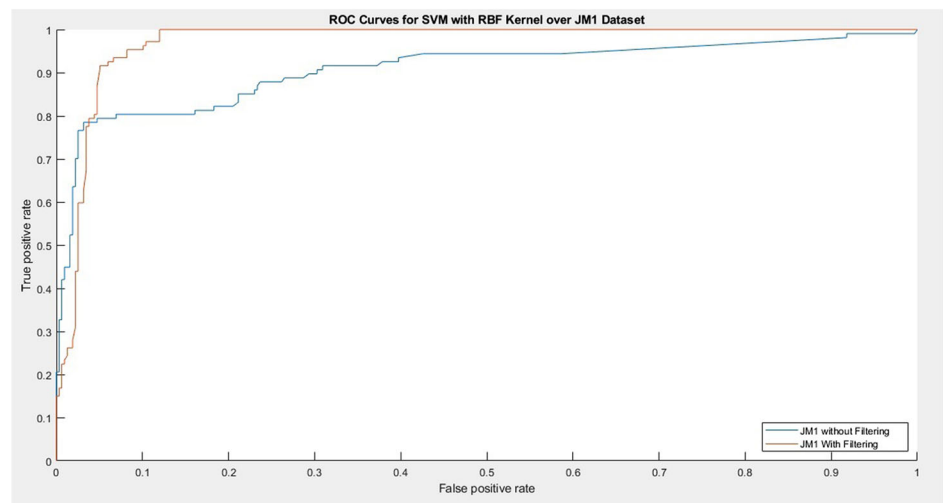
This study proposed a novel filtering technique (FILTER) for SVM variants to construct effective SDP prediction model. The experiments are conducted in MATLAB and the findings are:

1. It is observed that the proposed FILTER effectively reduces the IR by 18.81% and conditions the dataset for the training of SVM based SDP models.

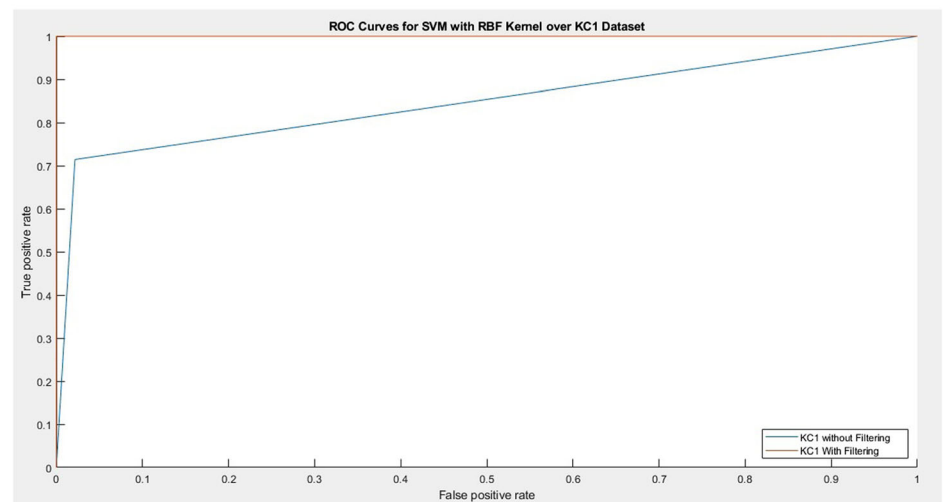
Fig. 12 (a) ROC curve over dataset CM1. (b) ROC curve over dataset JM1. (c) ROC curve over dataset KC1. (d) ROC curve over dataset KC2. (e) ROC curve over dataset PC1



(a)

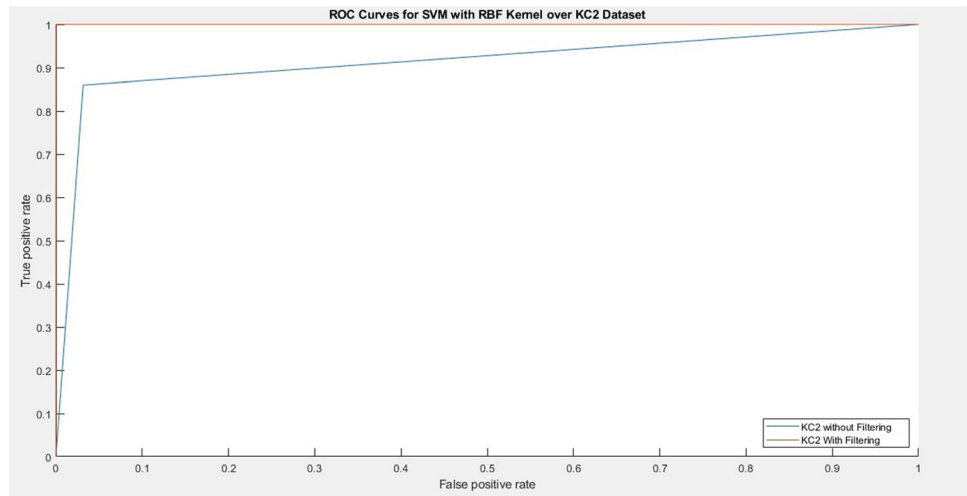


(b)

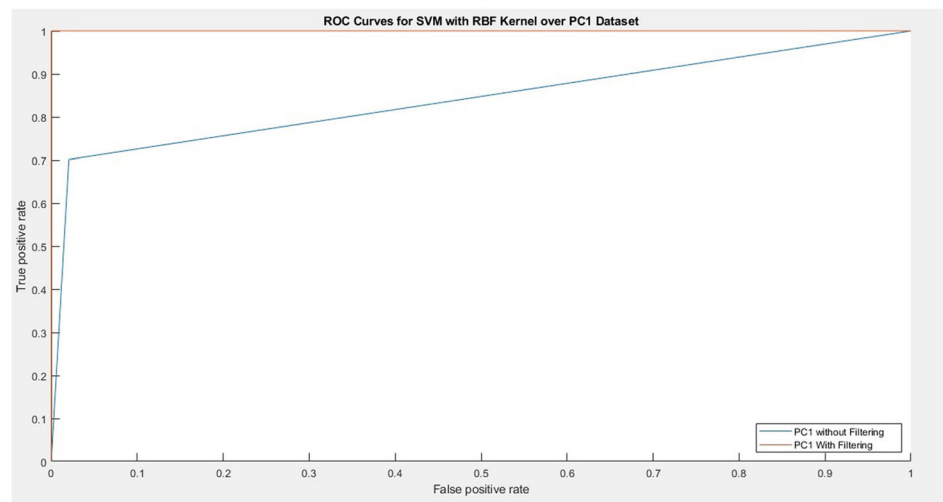


(c)

Fig. 12 continued



(d)



(e)

Fig. 13 p-statistic value (AUC)

Friedman's ANOVA Table					
Source	SS	df	MS	Chi-sq	Prob>Chi-sq
Columns	63.1	5	12.62	18.03	0.0029
Error	24.4	20	1.22		
Total	87.5	29			

Fig. 14 p-statistic value (accuracy)

Friedman's ANOVA Table					
Source	SS	df	MS	Chi-sq	Prob>Chi-sq
Columns	53.9	5	10.78	15.4	0.0088
Error	33.6	20	1.68		
Total	87.5	29			

Fig. 15 p-statistic value (F-measure)

Friedman's ANOVA Table					
Source	SS	df	MS	Chi-sq	Prob>Chi-sq
Columns	61.9	5	12.38	17.69	0.0034
Error	25.6	20	1.28		
Total	87.5	29			

- From the experiments, it can be inferred that the performance of SVM variant with RBF kernel performs best when trained with FILTERed dataset with average values of 95.68%, 96.48% and 94.88% for accuracy, AUC and F-measure respectively.
- Proposed filtering technique (i.e., FILTER) improves the prediction power of SVM-RBF based SDP model by 16.73%, 16.80% and 7.65% in terms of accuracy, AUC and F-measure respectively.

Further, the results are statistically validated using Friedman's test at the confidence level of 95% with $\alpha = 0.05$. Therefore, it can be concluded that the results obtained in the experiments are statistically significant.

In future, the author proposes to extend this work to predict the number of faults with deep learning model. The work can be replicated with larger industry based real-life dataset.

Funding No funding has been availed for this work.

Declarations

Conflict of interest The author has no conflict of interests.

References

- Afzal W, Torkar R, Feldt R (2012) Resampling methods in software quality classification. *Int J Softw Eng Knowl Eng* 22(2):203–223
- Cai X, Niu Y, Geng S, Zhang J, Cui Z, Li J, Chen J (2019) An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. *Concurr Comput Pract Exp* 32:e5478
- Chen L, Fang B, Shang Z et al (2018) Tackling class overlap and imbalance problems in software defect prediction. *Softw Qual J* 26:97–125. <https://doi.org/10.1007/s11219-016-9342-6>
- Chen J, Nair V, Krishna R, Menzies T (2019) "Sampling" as a baseline optimizer for search-based software engineering. *IEEE Trans Softw Eng* 45(6):597–614. <https://doi.org/10.1109/TSE.2018.2790925>
- Erturk E, Sezer EA (2015) A comparison of some soft computing methods for software fault prediction. *Expert Syst Appl* 42:1872–1879
- Felix EA, Lee SP (2019) Systematic literature review of preprocessing techniques for imbalanced data. *IET Softw* 13(6):479–496
- Goyal S (2020) Heterogeneous stacked ensemble classifier for software defect prediction. In: 2020 sixth international conference on parallel, distributed and grid computing (PDGC), Wanknaghat, Solan, India, pp 126–130. <https://doi.org/10.1109/PDGC50313.2020.9315754>
- Goyal S (2021a) Predicting the defects using stacked ensemble learner with filtered dataset. *Autom Softw Eng* 28:14. <https://doi.org/10.1007/s10515-021-00285-y>
- Goyal S (2021b) Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction. *Artif Intell Rev.* <https://doi.org/10.1007/s10462-021-10044-w>
- Goyal S, Bhatia P (2020b) Comparison of machine learning techniques for software quality prediction. *Int J Knowl Syst Sci (IJKSS)* 11(2):20–40
- Goyal S, Bhatia PK (2019) A non-linear technique for effective software effort estimation using multi-layer perceptrons. In: 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon), Faridabad, India, pp 1–4. <https://doi.org/10.1109/COMITCon.2019.8862256>
- Goyal S, Bhatia PK (2020) Feature selection technique for effective software effort estimation using multi-layer perceptrons. In: Proceedings of ICETIT 2019. Lecture notes in electrical engineering, Springer, Cham, vol 605, pp 183–194. https://doi.org/10.1007/978-3-030-30577-2_15
- Guo H, Li Y, Jennifer S, Gu M, Huang Y, Gong B (2017) Learning from class-imbalanced data: review of methods and applications. *Expert Syst Appl* 73:220–239
- Hanley J, McNeil BJ (1982) The meaning and use of the area under a Receiver Operating Characteristic ROC curve. *Radiology* 143:29–36
- Huda S, Liu K, Abdelrazek M, Ibrahim A, Alyahya S, Al-Dossari H, Ahmad S (2018) An ensemble oversampling model for class imbalance problem in software defect prediction. *IEEE Access* 6:24184–24195. <https://doi.org/10.1109/access.2018.2817572>
- Jaiswal A, Malhotra R (2018) Software reliability prediction using machine learning techniques. *Int J Syst Assur Eng Manag* 9(1):230–244
- Kaur P, Gossain A (2019) FF-SMOTE: a metaheuristic approach to combat class imbalance in binary classification. *J Appl Artif Intell* 33(5):420–439
- Kumar L, Sripada SK, Sureka A, Rath SK (2018) Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM). *J Syst Softw* 137:686–712
- Lehmann EL, Romano JP (2008) Testing statistical hypothesis: springer texts in Statistics. Springer, New York
- Ma Y, Pan W, Zhu S, Yin H, Luo J (2014) An improved semi-supervised learning method for software defect prediction. *J Intell Fuzzy Syst* 27:2473–2480. <https://doi.org/10.3233/IFS-141220>
- Malhotra R, Kamal S (2019) An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data. *Neurocomputing* 343(28):120–140. <https://doi.org/10.1016/j.neucom.2018.04.090>

- Menzies T, DiStefano J, Orrego A, Chapman R (2007) Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng* 32(11):1–12
- NASA (2015) https://www.nasa.gov/sites/default/files/files/Space_Math_VI_2015.pdf
- Ozakinci R, Tarhan A (2018) Early software defect prediction: a systematic map and review. *J Syst Softw* 144:216–239. <https://doi.org/10.1016/j.jss.2018.06.025>
- PROMISE. <http://promise.site.uottawa.ca/SERRepository>
- Rao KN, Reddy CS (2020) A novel under sampling strategy for efficient software defect analysis of skewed distributed data. *Evol Syst* 11:119–131. <https://doi.org/10.1007/s12530-018-9261-9>
- Rathore S, Kumar S (2017) Towards an ensemble-based system for predicting the number of software faults. *Expert Syst Appl* 82:357–382
- Rathore SS, Kumar S (2019) A study on software fault prediction techniques. *Artif Intell Rev* 51(2):255–327. <https://doi.org/10.1007/s10462-017-9563-5>
- Rong X, Li F, Cui Z (2016) A model for software defect prediction using support vector machine based on CBA. *Int J Intell Syst Technol Appl* 15(1):19–34
- Ross SM (2005) *Probability and statistics for engineers and scientists*, 3rd edn. Elsevier Press, Amsterdam (ISBN: 81-8147-730-8)
- Sayyad S, Menzies T (2005) The PROMISE repository of software engineering databases. University of Ottawa, Canada. <http://promise.site.uottawa.ca/SERRepository>
- Siers MJ, Islam MZ (2015) Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem. *Inf Syst* 51:62–71
- Son LH, Pritam N, Khari M, Kumar R, Phuong PTM, Thong PH (2019) Empirical study of software defect prediction: a systematic mapping. *Symmetry*. <https://doi.org/10.3390/sym11020212>
- Song Q, Guo Y, Shepperd M (2018) A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Trans Softw Eng*. <https://doi.org/10.1109/TSE.2018.2836442>
- Sun Z, Zhang J, Sun H, Zhu X (2020) Collaborative filtering based recommendation of sampling methods for software defect prediction. *Appl Soft Comput* 90:106–163
- Thomas J (1976) McCabe, a complexity measure. *IEEE Trans Softw Eng* 2(4):308–320
- Tsai CF, Lin WC, Hu YH, Yao GT (2019) Under-sampling class imbalanced datasets by combining clustering analysis and instance selection. *Inf Sci* 477:47–54
- Wang S, Yao X (2013) Using class imbalance learning for software defect prediction. *IEEE Trans Reliab* 62(2):434–443
- Wang K, Liu L, Yuan C, Wang Z (2021) Software defect prediction model based on LASSO-SVM. *Neural Comput Appl* 33(14):8249–8259
- Wu XD, Kumar V, Quinlan JR, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou ZH, Steinbach M, Hand DJ, Steinberg D (2007) Top 10 algorithms in data mining. *Knowl Inf Syst* 14:1–37. <https://doi.org/10.1007/s10115-007-0114-2>
- Yang X, Lo D, Xia X, Sun J (2017) TLEL: a two-layer ensemble learning approach for just-in-time defect prediction. *J Inf Softw Technol* 87:206–220

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.