Inteligencia Artificial 24(68), 72-88
doi: 10.4114/intartif.vol24iss68pp72-88

INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

# The Effect of the Dataset Size on the Accuracy of Software Defect Prediction Models: An Empirical Study

Mashaan A. Alshammari[1] and Mohammad Alshayeb[2,3,A]

[1]Information and Computer Science Department, University of Ha'il, Ha'il, Saudi Arabia

[2]Mohammad Alshayeb[2,3,A], Information and Computer Science Department, Interdisciplinary Research Center for
[3]Intelligent Secure Systems, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
[A]alshayeb@kfupm.edu.sa

**Abstract.** The ongoing development of computer systems requires massive software projects. Running the components of these huge projects for testing purposes might be a costly process; therefore, parameter estimation can be used instead. Software defect prediction models are crucial for software quality assurance. This study investigates the impact of dataset size and feature selection algorithms on software defect prediction models. We use two approaches to build software defect prediction models: a statistical approach and a machine learning approach with support vector machines (SVMs). The fault prediction model was built based on four datasets of different sizes. Additionally, four feature selection algorithms were used. We found that applying the SVM defect prediction model on datasets with a reduced number of measures as features may enhance the accuracy of the fault prediction model. Also, it directs the test effort to maintain the most influential set of metrics. We also found that the running time of the SVM fault prediction model is not consistent with dataset size. Therefore, having fewer metrics does not guarantee a shorter execution time. From the experiments, we found that dataset size has a direct influence on the SVM fault prediction model. However, reduced datasets performed the same or slightly lower than the original datasets.

**Keywords:** Software Defect Prediction, Support Vector Machine, Feature Selection

## 1. INTRODUCTION

The rapid growth in computer applications and telecommunication systems has resulted in software projects becoming massive and more complex. Performing testing on such complex systems might be too costly for an organization. Software defect prediction models provide the ability to estimate defects without actually running the system. This estimation is based on software metrics recorded from the system. Several studies have been carried out on improving and enhancing software defect prediction models. Some advantages of software defect prediction models are: making the system more dependable, improving the test process by focusing on fault proneness modules and improving software quality [1]. Industrial organizations heavily use software defect prediction models. Tosun et al. (2010) delivered a case study on practical considerations for defect prediction models [2].

There are two approaches, considered in the literature, to predicting faults in software systems: the machine learning approach [3] and the statistical approach. The machine learning approach uses learning algorithms such as k-Nearest Neighbor (k-NN) to train on a specific part of the data and then test (predict) on the remaining part. The

statistical approach concentrates on the statistical relationships between metrics such as mutual information (MI) and correlation. The statistical approach is more vulnerable to outliers and cannot handle problems such as high dimensionality and imbalanced classes. Therefore, researchers often use a machine learning approach for software defect prediction.

A support vector machine (SVM) is a well-known classification technique in machine learning. It has been tested on various applications in pattern recognition, including software defect prediction [4-9]. Vapnik developed the principle of SVM in 1995 [10]. The SVM algorithm comprises two steps, first, finding the optimal hyperplane with the longest distance from the nearest training patterns and second, drawing the support vectors with some margin on the sides of the hyperplane [11]. An essential feature of the SVM classifier is that the complexity of the resulting classifier depends on the support vectors, not on the transformed feature space. In other words, the SVM classification process does not depend on the number of metrics (features) used in the prediction model, whereas other machine learning techniques do. For instance, in the naïve Bayes (NB) classifier, the computation time of the classification process is proportional to the number of metrics used for prediction. Another advantage of SVM is that it is less prone to over-fitting problems than other methods [11].

Despite these advantages of SVM, researchers have found that the SVM fault prediction model has a weak performance in predicting defects compared to other machine learning algorithms. This motivated us to investigate the reasons for the low performance of SVM fault prediction models. In particular, we focus on the impact of dataset size and the number of metrics used and whether these factors affect SVM performance. Furthermore, we want to answer another question: does the size of the dataset influence the running time of SVM software defect prediction? We found that reducing the dataset size and keeping only the influential features improves SVM software defect prediction. However, we have not found a strong relationship between reducing the dataset size and the running time.

The paper is organized as follows: in Section 2, we review the related work, and we discuss the experimental setup in Section 3. We detail the empirical study in Section 4 and in Section 5 we discuss the results. The threats to validity are discussed in Section 6 and we provide the conclusion and suggestions for future work in Section 7.

## 2. LITERATURE REVIEW

Due to the importance of software defect prediction models, many studies using several techniques have been conducted. Nevertheless, it is not a new field of research, as we can see in [12-14], where they used methods like a feed-forward neural network to predict software reliability. Another notable effort was undertaken by Tantithamthavorn et al. to evaluate model validation techniques for defect prediction models [15]. However, these techniques seem to be simple compared to modern pattern recognition techniques. In fact, as the artificial intelligence techniques became more robust and dependable, software defect prediction models developed as well.

Wang et al. used 15 datasets from three real systems to empirically evaluate the performance of the SVM ranker using imbalanced datasets. The metrics were filtered from datasets using the SVM ranker and were then passed through five different learners: naïve Bayes (NB), multilayer perceptron (MLP), k-Nearest Neighbor (k-NN), SVMs and linear regression (LR). As their conclusion, they recommended how the parameters of the SVM ranker should be set to improve the prediction model [16]. Jin et al. proposed a software fault-proneness model (SFPM) as a new model for fault prediction. The proposed model consists of three components: an artificial neural network (ANN) for metrics selection, a function to compute the contribution of each metric and an SVM learner for prediction. This model was tested using four datasets from (PRedictOr Models in Software Engineering) PROMISE[1] repository and it was compared with five other learners: SVM, LR, k-NN, NB and the decision tree classifier (DT). Unlike the benchmark learners, the proposed model delivered a good and consistent performance with all datasets [17].

Some studies implemented a feature selection step before constructing the prediction model. Gao et al. [18] conducted a very deep controlled study to evaluate the metrics participating in the fault prediction model. Metrics data were collected over four releases from the large legacy telecommunication software system (LLTS). Additionally, the experiments were carried out using seven feature ranking algorithms, four feature selection techniques (None, ES, HS, and AHS). Finally, the data was passed through five well-known learners (NB, MLP, SVM, LR, and k-NN). In fact, the authors were in a strong position to argue that the SVM learner has the lowest performance and k-NN has the second-lowest. Another study by Khoshgoftaar et al. [19] addressed the problem of high dimensionality and class imbalance in software prediction models. The study investigated the performance of

[1] https://code.google.com/p/promisedata/

two feature selection (FS) approaches (individual FS and repetitive sampled FS) for software defect prediction. For individual FS, a single FS algorithm was applied directly to the data. In contrast, the repetitive sampled FS created a balanced version of the data using under-sampling or oversampling techniques before applying the FS algorithm. They also examined two options for building the prediction model (boosting and plain learner). The study concludes that repetitive sampled FS had better performance than individual FS using both learning techniques. Turhan et al. investigated the feasibility of using cross-project data to build a defect prediction model [20]. This idea represents a cost-effective solution since open-source data can be used to build the defect prediction model. However, the study concluded that it is not feasible to collect data from other projects, whereas the data within the project is available. Nevertheless, suppose the data within the project is not adequate to build the prediction model, in that case it is recommended to use data across projects since it delivers a similar performance of the data within the project.

Further research was conducted to investigate the impact of feature selection methods. Okutan and Yıldız (2014) used Bayesian networks to build prediction models for software defects. They used nine open-source data sets from the Promise repository and found that response for class (RFC), lines of code (LOC), and lack of coding quality (LOCQ) are the most effective metrics for the prediction model [21]. Kumar et al. proposed a defect prediction model using the least-squares support vector machine (LSSVM) using 30 open-source Java projects. They used then different feature selection techniques and 20 source code metrics. They found that the proposed prediction model is more suitable for projects with faulty classes less than the threshold value [22]. Manjula and Florence [23] proposed an approach that combines a genetic algorithm (GA) for feature optimization with a deep neural network (DNN) for classification for software defect prediction using software metrics. They found that the proposed approach to the prediction of software defects outperforms existing models. Majid et al. [24] proposed a technique for software defect prediction using a deep-learning model at statement-level metrics (SLDeep). They validated the proposed deep learning model and found it to be effective at statement-level software defect prediction. Aljamaan and Alazba [25] conducted an experiment to investigate the prediction performance of seven Tree-based ensembles in defect prediction. using 11 publicly available MDP NASA software defect datasets. They found that Random Forest and Extra Trees ensembles outperformed other Tree-based boosting ensembles.

Several studies investigated the influence of independent variables on software defect prediction models. Jiang et al. investigated the variance and its impact on fault prediction accuracy. From 12 datasets with different parameters, it was concluded that the variance of samples affected fault prediction results [26]. Additionally, Muzaffar and Ahmed investigated the factors that affect the accuracy of fuzzy logic systems and concluded that the prediction model is highly dependent on the system structure, the participating parameters, and the training algorithms [27]. Similarly, Catal and Diri inspected the impact of 3 independent factors: dataset size, metric set and metric selection algorithm on fault prediction model accuracy. In addition, they inspected the best techniques to deal with datasets of different sizes. The study used datasets from the PROMISE repository and 9 algorithms for the prediction (learning) process. The study concluded that using the random forest (RF) algorithm achieved better performance on large datasets. On the other hand, the naive Bayes (NB) algorithm achieved better performance on small datasets [1]. D'Ambros et al. presented a benchmark for defect prediction for publicly available datasets, compared the well-known bug prediction approaches and also proposed their own [28].

Due to the large number of studies in the software defect prediction field, researchers conducted systematic literature reviews to summarize the main differences between studies and direct new researchers' efforts. Hall et al. conducted a systematic literature review by collecting 208 papers published between 2000 and 2010 on fault prediction models. This study investigated the prediction model development, the independent variables and the techniques used for building the models. The study was able to answer important questions; for instance, the models which used object-oriented (OO) metrics performed better than the models which used source code metrics. Furthermore, the models that used support vector machines (SVM) had worse performance than those used other techniques, whereas NB and LR achieved the best performance of all the learning algorithms. Accordingly, it was concluded that the best performance models had optimized data, optimized independent variable selection and the modeling technique were the most suitable for the data [29]. Another review study by Catal surveyed 90 software defect prediction papers that use the machine learning approach and the statistical approach. Based on the review outcomes, the study encouraged researchers to focus on unlabeled program modules and limited fault data [30]. Malhotra conducted a systematic review of machine learning techniques for software fault prediction. The study compared machine learning and statistical techniques and also compared the performance of the different machine learning techniques. The study concluded that machine learning techniques are better than LR techniques in predicting software defects. They also found that RF was the best method of defect prediction [31]. Tantithamthavorn et al. conducted an empirical comparison of model validation techniques for defect prediction models. In their study, they identified the most common evaluation techniques used for model validation. Based on

their empirical evaluation, the authors recommended using out-of-sample bootstrap validation instead of single-repetition holdout validation to maintain the best balance between bias and variance in the selected features [15].

It can be noted from the literature review that the SVM fault prediction model achieved a weak performance in predicting defects compared to the other machine learning algorithms. This observation was stated clearly by Gao et al. [18] and Hall et al. [29]. Therefore, in this study, we investigate the reasons for the low performance of SVM fault prediction models. Specifically, we inspected the direct effect of two independent variables: dataset size and number of used metrics. The dataset size effect on SVM has been investigated in applications other than software defect prediction [32, 33]. Dataset size represents the number of samples (records) in the fault prediction model while the number of used metrics represents which measurements are used to build the fault prediction model. Additionally, we examine some of the techniques that can enhance the performance of the SVM fault prediction model. Table 1 summarizes the related work.

Table 1: Summary of the Related Work

| Author and year | Data | Type of Prediction Model | Machine learning Technique | Results |
|---|---|---|---|---|
| Catal and Diri, 2009 [1] | CM1, JM1, KC1, KC2, and PC1 from PROMISE repository. | Software defects | J48, RF, NB, Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, and AIRS2Parallel. | RF algorithm produced better performance for large datasets whereas NB was the most suitable for small datasets. |
| Jiang et al., 2009 [26] | MDP as well as PROMISE repositories. | Software defects | RF, BAG, LOG, BST, and NB | The variance of the samples has a direct influence on fault prediction accuracy. |
| Muzaffar an d Ahmed, 2010 [27] | Five artificial datasets generated by COCOMO. | Effort prediction | Fuzzy logic systems | Effort prediction model is affected by system structure, fuzzy logic parameters, and training algorithms. |
| Wang et al., 2011 [16] | Telecommunications system, Eclipse, and NASA- project. | Software defects | NB, MLP, k-NN, SVM, and LR | Predictor tends to perform better by changing the number of metrics removed by SVM ranker per iteration. |
| Gao et al., 2011 [18] | Data was collected over 4 releases from Large Legacy Telecommunication software System (LLTS). | Software defects | 3 selection techniques (ES, HS, and AHS) in addition to 4 learners (NB, MLP, SVM, LR, and k-NN) | For selection techniques, AHS provides good predictions. For learners, NB was the best performer whereas SVM was the lowest. |
| Jin et al., 2012 [17] | NASA software projects: PC1, CM1, KC1, and KC3. | Software defects | SFPM, SVM, LR, kNN, NB, and DT | SFPM performs consistently in addition to its automated implementation. |
| Turhan et al., 2013 [20] | MDP repository and Turkish software company. | Software defects | NB | Data across projects delivered similar performance to the data within project. |
| Okutan and Yıldız, 2014 [21] | Promise data repository | Software defects | Bayesian networks | RFC, LOC and LOCQ are the most effective metrics for defect prediction. |
| Kumar et al. 2017 [22] | 30 Open-Source Java projects | Software defects | Least Squares Support Vector Machine | The prediction model is more suitable for projects with faulty classes less than the threshold value. |
| Manjula and Florence, 2019 [23] | Promise data repository | Software defects | A hybrid approach deploying GA and DNN | Improved performance is reported using the proposed approach. |
| Majid et al., 2020 [24] | 100,000 C/C++ programs | Software defects | Deep learning | SLDeep seems to be effective at statement-level software defect prediction |
| Aljamaan and Alazba, 2020 [25] | MDP | Software defects | Tree-based ensembles | Tree-based bagging ensembles (Random Forest and Extra Trees ensembles) outperforms other Tree-based boosting ensembles. |
| This work | CM1, KC1, PC1, and PC4. | Software defects | SVM, GA, Gain Ratio, and ReliefF. | Dataset size impacts the performance of SVM prediction model. |

# 3. EXPERIMENTAL SETUP

In this section, we discuss the datasets, methodologies, tools, and algorithms that we used in this study.

## 3.1 Datasets

Datasets from the PROMISE repository have been widely used by researchers in defect prediction research [29]. CM1 and KC1 are two datasets used for defect prediction. With 498 samples in the CM1 dataset and 2109 samples in the KC1 dataset, CM1 plays the role of the smaller dataset in terms of size. Nevertheless, both datasets have 22 features including the target one to maintain the same length of the feature vector. The features' description can be found in Appendix A.

However, datasets in the PROMISE repository were the subject of some criticism [34, 35]. To avoid misleading conclusions, we used PC1 and PC4 datasets from the NASA MDP website. Two variants of preprocessed datasets exist: D" and D' however, the developers recommend the D' variant. Both PC1 and PC4 have the same length of feature vector, which is 38 including the class attribute. The list of features can be found in Appendix B.

## 3.2 Research Approach

Our objective is to investigate the reasons why SVM does not perform well in fault prediction models. To investigate this claim, we perform predictions using datasets of different sizes. As shown in Figure 1, both datasets are used in the prediction model with their original set of metrics. Afterwards, the datasets go through a metric selection algorithm the prediction model uses them. It is important to mention that the datasets vary in their number of samples whereas the number of metrics is the same in both datasets. In other words, we are fixing the independent variable "number of metrics".

In this study, we ran 2 independent experiments using 4 datasets and 4 feature selection algorithms. The feature selection algorithms were distributed over both experiments to provide a valid observation independent from the used feature selection algorithm. In the first experiment, we tested the CM1 and KC1 datasets from the PROMISE repository [36]. The feature selection algorithms used in this experiment are: 1) Correlation-Based Feature Selection with Exhaustive Search, 2) Correlation-Based Feature Selection with Genetic Search. Furthermore, PC1 and PC4 datasets from the NASA Metrics Data Program (MDP) website2 were used to perform the second experiment. Another two feature selection algorithms used in this experiment are: 1) the Gain Ratio algorithm, 2) ReliefF algorithm.

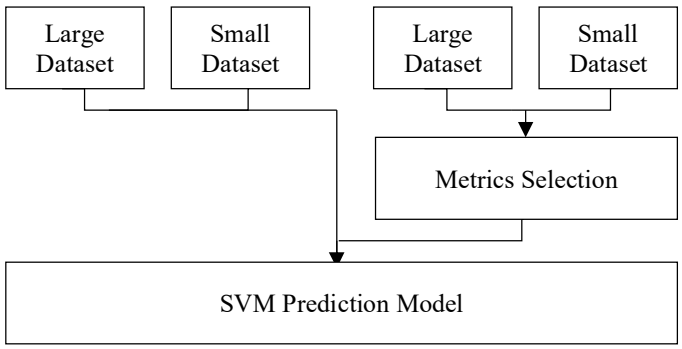


Figure 1. Investigating the influence of dataset size and metrics selection

## 3.3 Metrics Selection Algorithms

Metrics can be categorized from a prediction point of view into three categories: relevant, irrelevant and redundant. Relevant metrics have a strong influence on the predictor's decision. On the other hand, irrelevant metrics have little effect on the predictor's decision. Finally, redundant metrics have the same impact on the prediction process. One representative metric for these redundant metrics might be enough for making a decision. To avoid a biased prediction process, the metrics selection algorithm was performed on the small dataset. Running selection

[2] http://nasa-softwaredefectdatasets.wikispaces.com/

algorithms on large datasets requires a lot of computation time due to the large number of samples. The metrics selection process implemented in this study is shown Figure 2. Four metrics selection algorithms were used in this study; the algorithms were divided into two experiments.

### 3.3.1 Metrics Selection Algorithms in Experiment 1

The metrics selection algorithms used in the first experiment are: Correlation-Based Feature Selection with Exhaustive Search (CFS ES) and Correlation Based Feature Selection with Genetic Search (CFS GS). Both search algorithms eliminate the correlated features; however, they differ in how they explore the search space. The exhaustive search algorithm searches for the most effective subset of metrics by examining all possible combinations and selects the subset of metrics that minimizes the error rate. Nevertheless, the implementation of an exhaustive search is time consuming due to its search method. On the other hand, the genetic search algorithm searches for the optimum set of metrics by optimizing the search space.

A genetic search is one of the evolutionary algorithms. It starts with an initial set of chromosomes; in this study, it is a set of metrics combinations. A genetic search explores the search space by performing two operations: crossover and mutation. Crossover and mutation operations are responsible for producing new offspring based on existing parents. The effectiveness of each chromosome is measured by a fitness function, which the programmer designs to accomplish a predefined objective. Unlike the exhaustive search, a genetic search algorithm is not time-consuming due to its capability to optimize the search space.

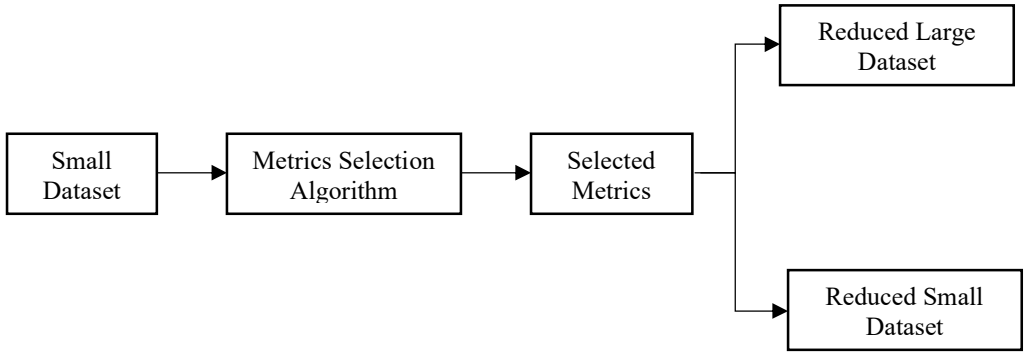


Figure 2. Metrics Selection Process

### 3.3.2 Metrics Selection Algorithms in Experiment 2

In the second experiment, the Gain Ratio and ReliefF algorithms were used to reduce the feature space. The gain ratio is a divide-and-conquer algorithm. It tackles the problem by constructing decision trees. It was derived from the information gain algorithm, which tests the outcomes based on the absence of the feature. While information gain tends to favor features with large numbers of possible values, gain ratio overcomes this problem by considering the number and the size of the child nodes into which the feature splits the dataset, regardless of the impact of class information [37].

The ReliefF algorithm is an extension of the Relief algorithm. The principle of the Relief algorithm is to assign a contribution weight to each feature in the training model. This weight represents its contribution to the target attribute. The weight calculation is done by calculating the distance between the instances of the same class ”near-hit” and the instances of the other class ”near-miss” [38]. The ReliefF algorithm was proposed by Kononenko et al. to overcome the shortcomings in the Relief algorithm such as noise, incompleteness, and multi-class datasets. They improved the heuristic by applying Manhattan distance instead of Euclidean distance to calculate the difference between “near-hit” and “near-miss” instances. Additionally, they applied absolute differences instead of square differences. They also contributed to the algorithm by suggesting the use of conditional probability to estimate the missing features [39].

### 3.4 Tools

The Waikato Environment for Knowledge Analysis (WEKA 3) is a machine learning software based on Java developed at the University of Waikato, New Zealand [40]. WEKA is free software available under the GNU General Public License. The latest stable release is WEKA 3.6, which was released in 2008 and is Java-based. It is a very popular software in the machine learning community, and it provides various techniques to execute machine-learning algorithms. In this study, we used WEKA to perform the metrics selection operations.

The SVM fault prediction models were built using DTREG (predictive modeling software) [41]. The classification testing method used in this study is 10-fold cross-validation where the data is divided into 10 sets of size n/10, then 9 of the datasets are used for training and 1 dataset is used for testing. The experiment is replicated 10 times and each iteration uses a different testing dataset. This method is preferable over dividing the dataset into training and testing portions because it provides more statistically reliable results due to replication.

### 3.5 SVM Parameter Optimization

For our SVM approach, we used a radial basis function (RBF) as a kernel function. This selection was built on a recommendation by DTREG documentation. The performance of the SVM model can be strongly influenced by the selection of SVM parameter values [42]. Using RBF, SVM tends to have two influential parameters: the complexity parameter (C) and the Gaussian free parameter (Gamma). C controls the level of the support vector margins; in other words, it specifies the misclassification cost. A low C value leads to soft margins with a low misclassification cost, whereas a high C value produces rigid margins with high misclassification cost. On the other hand, Gamma defines the effect of a single training sample. A low Gamma value defines the far range of the effect, whereas a high Gamma value means the effect of the sample will be close. In general, Gamma is very useful if the training data is not separated linearly; tuning Gamma controls the shape of the decision border.

SVM parameters can be optimized using many search techniques, however the search techniques provided by DTREG are grid search and pattern search. Grid search is the simplest algorithm in parameter optimization since it performs an exhaustive search within the limits provided by the user. Nevertheless, the pattern search performs parameter optimization more intelligently. It starts at some point in the search space then performs steps in each direction. If the evaluation criteria improve the algorithm, it transfers its center to the new point. If all the surrounding points fail to produce an improvement, the step size is reduced. The pattern search algorithm stops when the search step size ($\Delta_k$) reaches a specified limit.

DTREG provides the ability to combine both grid and pattern searches. First, the grid search operates using a specified number of points to find the best point. Afterwards, the pattern search operates on a narrow space surrounding the grid search best point to find an optimal point to be used in the SVM model. This combination overcomes the pattern search deficiency since it might get trapped in local optima.

## 4. EMPIRICAL STUDY

To investigate the effect of dataset size on SVM defect prediction, two experiments were conducted in this study. The first experiment uses two datasets from the PROMISE repository whereas the second experiment uses two datasets from the MDP repository. Each experiment has 3 sub-experiments: 1) SVM defect prediction using the complete set of metrics, 2) SVM defect prediction after applying the first metrics selection algorithm, 3) SVM defect prediction after applying the second metrics selection algorithm. The distribution of the datasets and metrics selection algorithms is described in Table 2.

Table 2: Distribution of Datasets and Metrics Selection Algorithms

| | Dataset | Language | Number Of Metrics | Number Of Samples | First FS Algorithm | Second FS Algorithm |
|---|---|---|---|---|---|---|
| Experiment 1 | CM1 | C | 22 | 498 | CFS ES | CFS GS |
| | KC1 | C++ | 22 | 2109 | | |
| Experiment 2 | PC1 | C | 38 | 759 | Gain Ratio | ReliefF |
| | PC4 | C | 38 | 1399 | | |

[3] http://www.cs.waikato.ac.nz/ml/weka/

## 4.1 Research Hypotheses

It is recommended that several hypotheses be formulated before conducting empirical studies in software engineering [43].The hypothesis structure for this study is as follows:

$H_1$: "The impact of dataset size on the SVM defect prediction model"

- Null hypothesis $H_{10}$: Dataset size has no direct impact on SVM defect prediction output.
- $H_{11}$: Dataset size has a direct impact on SVM defect prediction output. (Null hypothesis: Dataset size has no direct impact on SVM defect prediction output.)

$H_2$: "The impact of dataset size on the execution time of the SVM defect prediction model"

- Null hypothesis $H_{20}$: Dataset size has no direct impact on SVM defect prediction execution time.
- $H_{21}$: Dataset size has a direct impact on SVM defect prediction execution time.

## 4.2 Experiment 1: SVM Defect Prediction Model using PROMISE Datasets

In this section, we discuss the details of the first experiment and its sub-experiments.

### 4.2.1 SVM Defect Prediction using the Complete Set of Metrics

The SVM fault prediction model was built based on the complete set of metrics for datasets: CM1 and KC1. In CM1, 50 instances were misclassified out of 498 leading to 89.9% prediction accuracy. However, in KC1, 296 were predicted wrongly out of 2109, resulting in an accuracy rate of 85.9%. It can be noted from Figure 3 that the accuracy rate dropped in KC1 prediction results compared to CM1 prediction even though both datasets have the same number of metrics. This drop can be explained by the rapid growth in the number of samples while the number of metrics remains unchanged. This increase in the number of samples leads to some confusion during the prediction process.

### 4.2.2 SVM Defect Prediction using a Subset of Metrics Reduced by CFS ES

The most effective set of metrics was selected by correlation-based feature selection using the exhaustive search algorithm. Eight metrics out of the 22 available metrics were selected, thus reducing the feature space by almost 63%. The selected metrics are 7 predictors (1, 4, 9, 11, 14, 15, 17) in addition to the target attribute 22 (description is shown in Appendix A).

The SVM defect prediction model was built based on this set of metrics. For the CM1 set, the results remain unchanged even after using the reduced set of metrics. For the KC1 set, the number of misclassified samples increased by 5 after using the reduced set. As shown in Figure 3, the CM1 accuracy rate does not change whereas the KC1 accuracy rate has a small drop of 0.2 compared to the accuracy rate using the full set of features. The drop caused by the large number of samples shows that the reduced set was not effective as it was with the small number of samples in CM1.

### 4.2.3 SVM Defect Prediction using a Subset of Metrics Reduced by CFS GS

The second metric selection algorithm used in this experiment is correlation-based feature selection with genetic search. Seven metrics were selected (6 predictors and 1 target) of the 22 available metrics, thus reducing the complete set of metrics by almost 68%. The selected metrics are 1, 4, 14, 15, 17, 18 and 22 (the description is given in Appendix A). We used the default options of genetic search provided by WEKA, which are specified as follows: the population size is 20 chromosomes, the number of generations was set to 20 generations, the probability of crossover is 0.6 and the probability of mutation is 0.033. After building the SVM defect prediction model using this subset of metrics, the performance of CM1 was improved where the number of misclassified samples was reduced by one sample compared to the previous sub-experiments. On the other hand, the number of misclassified samples for the KC1 dataset was increased by 11 samples compared to the first sub-experiment and by 6 samples compared to the second sub-experiment. The SVM prediction accuracy rates are shown in Figure 3.

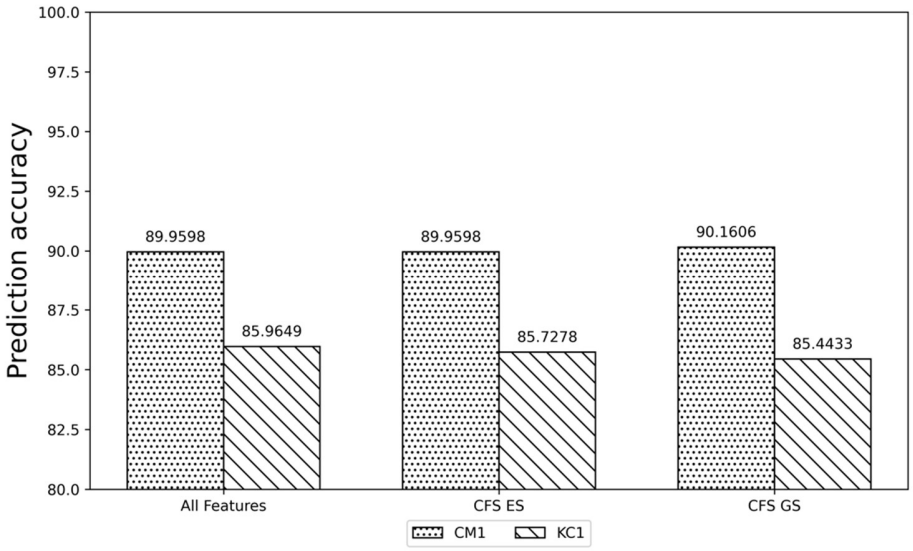


Figure 3. SVM prediction accuracy rates for CM1 and KC1 from the PROMISE dataset

To evaluate the model, we also use area under the ROC curve (AUC) and the results are shown in Figure 4. Unlike the accuracy outcomes, the KC1 dataset (dataset with more instances) has a higher score in terms of AUC. This observation can be explained by the distribution of instances over classes. The CM1 has 449 instances of class (defective = False) and 49 instances of class (defective = True), while the KC1 has 1783 instances of class (defective = False) and 326 instances of class (defective = True). This distribution in the KC1 dataset provides a sophisticated SVM training model that has a high AUC score instead of classifying all instances towards the dominating class.

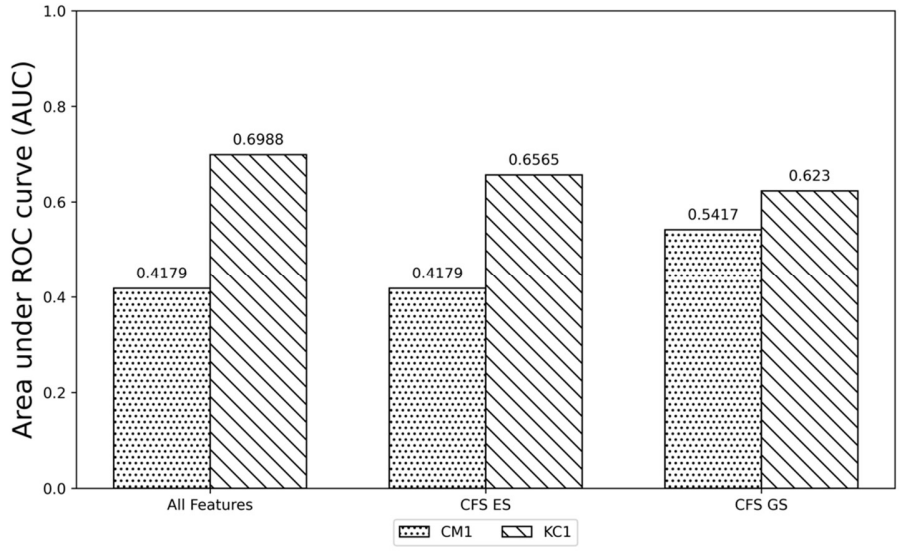


Figure 4. AUC for CM1 and KC1 from the PROMISE dataset

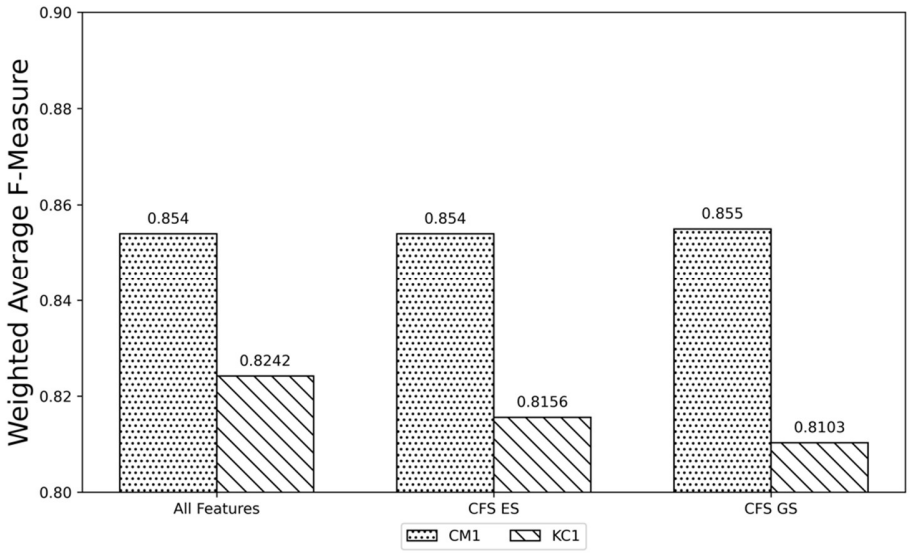


Figure 5. Weighted Average F-Measure for CM1 and KC1 from the PROMISE dataset

## 4.3 Experiment 2: SVM Defect Prediction Model using MDP Datasets

In this section, we discuss the details of the second experiment and its sub-experiments.

### 4.3.1 SVM Defect Prediction using the Complete Set of Metrics

After building the SVM prediction model using the full set of metrics, 61 samples of PC1 were misclassified of the total number of 759 samples, leading to a prediction accuracy of 91.96%. This accuracy dropped a little for PC4 with 135 misclassified samples of the total 1399 samples. The prediction rates are shown in Figure 4.

### 4.3.2 SVM Defect Prediction using a Subset of Metrics Reduced by Gain Ratio

We used the ranker method provided by WEKA to rank the metrics based on their relevance to the final outcome. After ranking the metrics using the Gain Ratio algorithm, we selected the top 15 metrics as predictors in addition to the target attribute. The top 15 metrics provided by the Gain Ratio algorithm are 18, 5, 35, 37, 4, 16, 30, 1, 33, 8, 25, 21, 36, 13, 32 (for the metrics description see Appendix B).

PC1 maintains the same prediction rate achieved by the complete set of metrics even though it used only 42% of its entire feature space. On the other hand, PC4 had 143 misclassified instances forcing the prediction accuracy to drop by 0.6% from what was achieved by the complete set of metrics. The results of this sub-experiment are shown Figure 6.

### 4.3.3 SVM Defect Prediction using a Subset of Metrics Reduced by ReliefF

Similar to the reduction performed by Gain Ratio, in this sub-experiment, we used the ranker technique to rank the metrics using the ReliefF algorithm. The top 15 metrics are 12, 36, 26, 15, 17, 8, 18, 30, 3, 23, 5, 1, 35, 4, 34 (for the metrics description see Appendix B). These metrics were selected to serve as predictors in the SVM prediction model.

The prediction rates using the PC1 and PC4 datasets dropped by 0.4% and 0.2% respectively compared to the prediction rates achieved by the complete set of metrics, as shown in Figure 6.

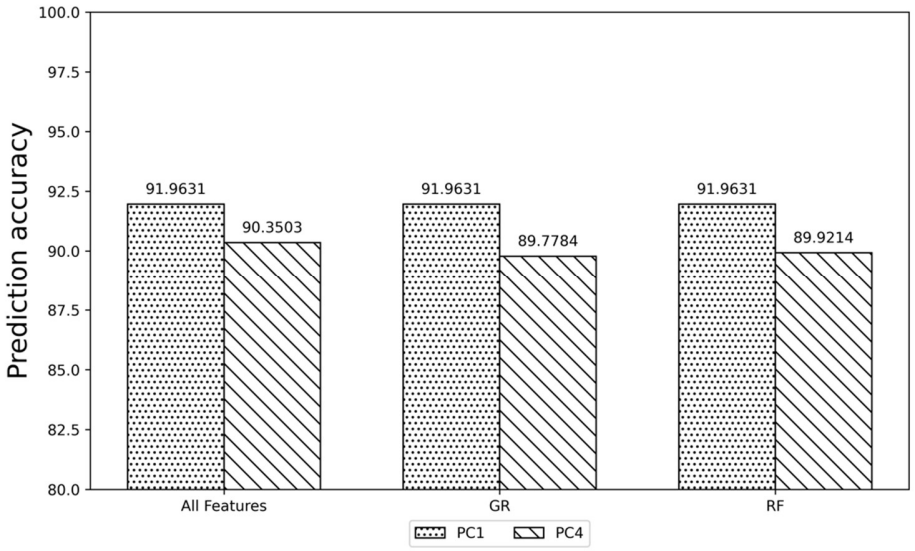


Figure 6. SVM prediction accuracy rates for PC1 and PC4 from MDP datasets

We also use AUC to evaluate the model and the results are shown in Figure 7. As we observed in experiment 1, the dataset with the larger number of instances (PC4 in this experiment) has the higher AUC score. PC1 has 698 instances of class (defective = False) and 61 instances of class (defective = True), whereas PC4 has 1221 instances of class (defective = False) and 178 instances of class (defective = True). In PC4, the class (defective = True) has a significant share of instances to train the SVM model on this class which produced a high AUC score.

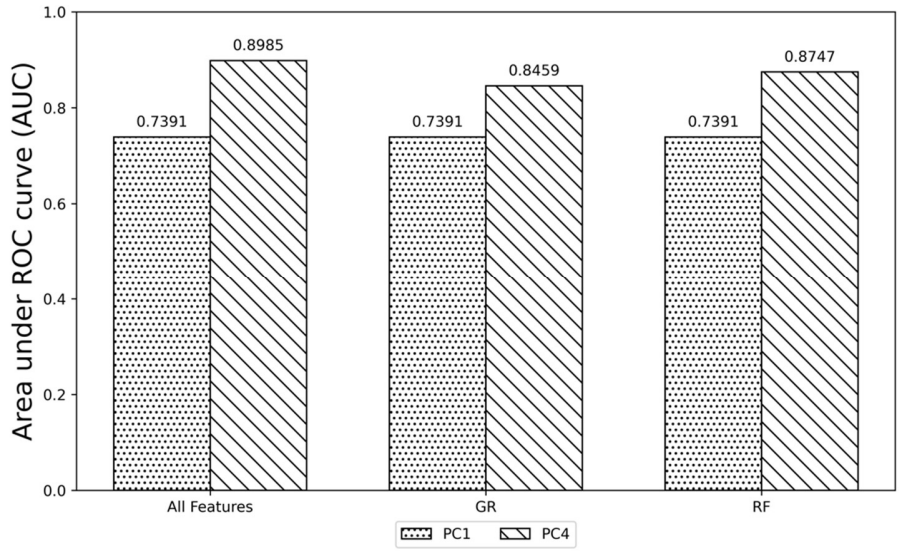


Figure 7. AUC for PC1 and PC4 from the MDP datasets

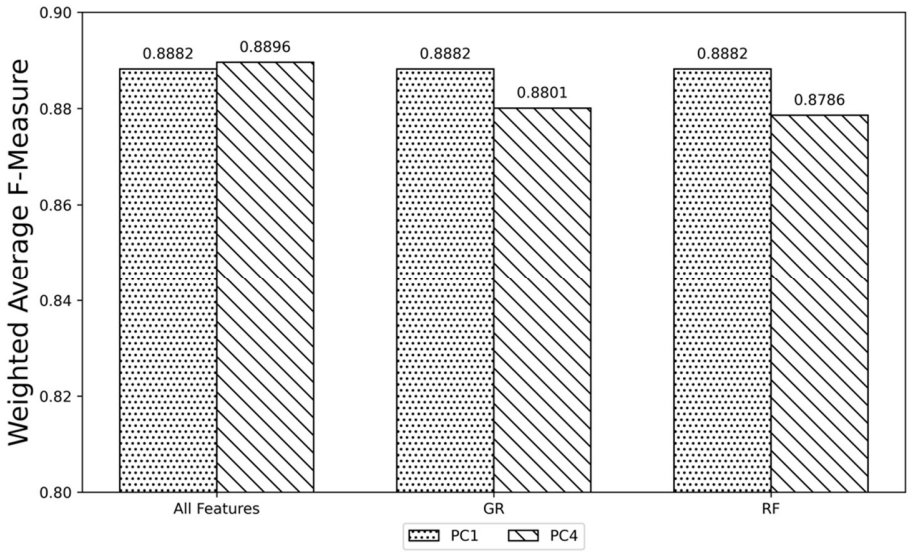


Figure 8. Weighted Average F-Measure for PC1 and PC4 from the MDP dataset

## 4.4 Running Time Analysis

While executing the SVM fault prediction model using the CM1 dataset, the running time was recorded. For the complete set of metrics, CM1 has 22 metrics, whereas it has 8 and 7 metrics respectively after running correlation feature selection for exhaustive and genetic searches. As shown in Figure 9, CM1 Genetic Search has a longer running time than CM1 Exhaustive Search although CM1 Genetic Search has 7 metrics and CM1 Exhaustive Search has 8 metrics. This shows that the number of metrics and the running time for the prediction model are not directly related however, they are influenced by other factors such as the complexity of the metrics value.

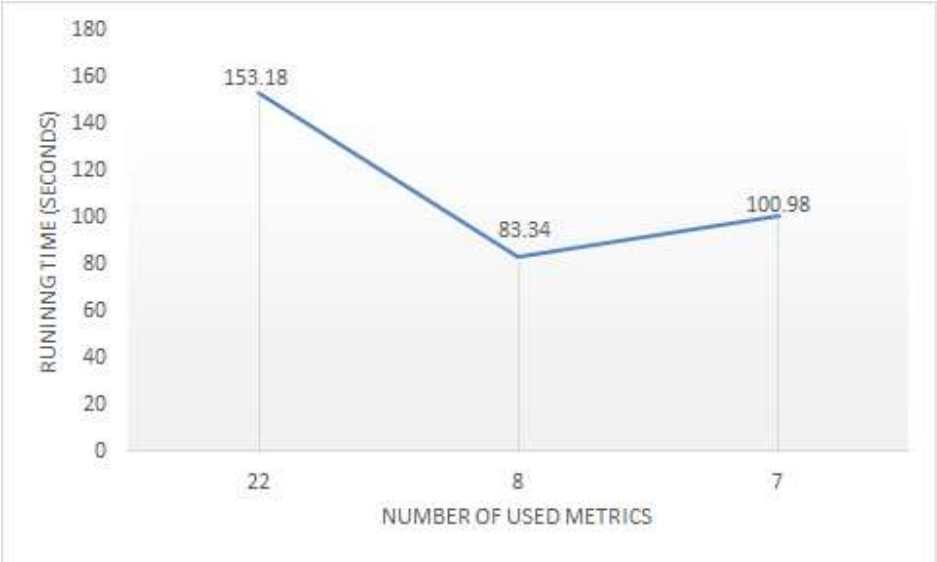


Figure 9. Running time for the SVM defect prediction model using the CM1 dataset.

## 4.5 Discussion

For the PROMISE datasets, when the complete set of metrics was used, the smaller dataset (CM1) scored the highest prediction rate because the 498 instances can be predicted sufficiently with the complete set of metrics.

However, this is not the case with the 2109 instances in the KC1 dataset, which led to a drop in the prediction rate. The difference in the prediction rate between CM1 and KC1 continues to appear after applying correlation-based feature selection with exhaustive and genetic searches.

In the same line, the MDP datasets have similar behavior to the PROMISE datasets. Using the complete set of metrics, the smaller dataset (PC1) achieved higher accuracy than the larger dataset (PC4). This observation does not change after applying the Gain Ratio and ReliefF algorithms.

It can be noted that the difference in the prediction accuracy between PC1 and PC4 was not as large as the difference between CM1 and KC1. This can be explained by the difference in the number of instances between the large and small datasets. PC1 and PC4 have a difference of 640 instances whereas the difference in the number of instances between CM1 and KC1 is 1611 instances.

The observations obtained by the AUC demonstrate that the dataset with a larger number of instances has a higher AUC score. This is explained by the distribution of instances among classes. In the small datasets, the class (defective = True) has a small share of instances, which provides incomplete training for the SVM, therefore most of instances are misclassified causing a low AUC score. On the other hand, when the dataset has a good number of instances of the class (defective = True), this provides sufficient training for the SVM and high AUC score as a result.

Therefore, in all six experiments, we experienced the same observation, which is the smaller dataset always has a higher accuracy score than the larger dataset and a lower AUC score. This observation continues to appear even though the number of metrics was reduced. As a result, we accept the first hypothesis (H1) and reject the null hypothesis.

For the experiment on running time, it was observed that the running time does not depend on the number of used metrics. A dataset with a smaller number of metrics had a longer processing time than a dataset with a larger number of metrics. This can be justified by the complexity of the metric itself. Intuitively, processing float values is much more complex than processing integer values.

Therefore, dataset size has an impact on SVM prediction execution time. This impact does not necessarily have a positive correlation with execution time. Consequently, we accept the second hypothesis and reject the null hypothesis.

### 4.6. Threats to Validity

Some possible threats that may affect the validity of the findings of this study. Using two projects with four datasets for prediction may impact on our conclusion that dataset size has a direct impact on the SVM defect prediction model. However, we tried to minimize this impact by testing four datasets from two different repositories. Additionally, measuring the running time of the SVM defect prediction model using a standard PC clock might be considered a threat to construct validity. A standard PC clock can differ from one PC to another based on the PC's capabilities, such as CPU and memory. Nevertheless, the running time analysis was executed multiple times using different timings to minimize this threat. Another possible internal threat to validity is that the non-stable running time of the SVM fault prediction model can be affected by the conditions during the experiment, such as CPU load at the time of the experiment. Therefore, to minimize this threat, we ran the experiment multiple times. The datasets used in this study were selected from public repositories (PROMISE and its variant MDP repositories) making them valid for verification and replication by the research community. However, this can be a threat to external validity since different data sets from different fields may provide different results.

## 5. CONCLUSION AND FUTURE WORK

The SVM learning method is one of the commonly used classifiers in the machine learning community. Our experiment results show that SVM provides sturdy performance in tackling datasets with a considerable feature space, although some studies claim that the SVM learning algorithm has a weak performance in predicting defects in fault prediction models compared to other machine learning algorithms. In this empirical study, we investigated the impact of dataset size on the SVM defect prediction model. We used four metric selection algorithms to observe the impact of the SVM fault prediction model on datasets with reduced metrics. The experiment findings showed that the dataset size directly effects on the performance of the SVM defect prediction model. Applying the SVM defect prediction model on datasets with a reduced number of metrics may enhance the accuracy of the fault prediction model, although it directs the test effort to maintain the most influential set of metrics. Moreover, the running time

of the SVM fault prediction model using a dataset with a reduced number of metrics is not stable; therefore, having fewer metrics does not guarantee a shorter execution time.

In our future work, we plan to conduct more experiments with different datasets to further confirm or negate our findings. We also plan to study and compare the impact of dataset size on different machine learning algorithms.

## Acknowledgement

M. Alshammari would like to acknowledge the support of University of Ha'il and M. Alshayeb acknowledges the support of King Fahd University of Petroleum & Minerals.

## References

[1] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences,* vol. 179, no. 8, pp. 1040-1058, 2009.

[2] A. Tosun, A. Bener, B. Turhan, and T. Menzies, "Practical considerations in deploying statistical methods for defect prediction: A case study within the Turkish telecommunications industry," *Information and Software Technology,* vol. 52, no. 11, pp. 1242-1257, 2010.

[3] M. S. Rawat and S. K. Dubey, "Software defect prediction models for quality improvement: a literature study," *International Journal of Computer Science Issues,* vol. 9, no. 5, pp. 1694-0814, 2012.

[4] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software,* vol. 81, no. 5, pp. 649-660, 2008.

[5] I. Gondra, "Applying machine learning to software fault-proneness prediction," *Journal of Systems and Software,* vol. 81, no. 2, pp. 186-195, 2008.

[6] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics," in *Engineering Applications of Neural Networks*, vol. 43, D. Palmer-Brown, C. Draganova, E. Pimenidis, and H. Mouratidis, Eds. (Communications in Computer and Information Science: Springer Berlin Heidelberg, 2009, pp. 223-234.

[7] S. D. Martino, F. Ferrucci, C. Gravino, and F. Sarro, "A genetic algorithm to configure support vector machines for predicting fault-prone components," presented at the Proceedings of the 12th international conference on Product-focused software process improvement, Torre Canne, Italy, 2011.

[8] H. A. Al-Jamimi and L. Ghouti, "Efficient prediction of software fault proneness modules using support vector machines and probabilistic neural networks," in *5th Malaysian Conference in Software Engineering (MySEC)*, 2011, pp. 251-256.

[9] S. Agarwal, D. Tomar, and Siddhant, "Prediction of software defects using Twin Support Vector Machine," in *Information Systems and Computer Networks (ISCON), 2014 International Conference on*, 2014, pp. 128-132.

[10] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning,* vol. 20, no. 3, pp. 273-297, 1995.

[11] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[12] N. Karunanithi, Y. K. Malaiya, and D. Whitley, "Prediction of software reliability using neural networks," in *International Symposium on Software Reliability Engineering*, 1991, pp. 124-130.

[13] N. Li and Y. K. Malaiya, "Enhancing accuracy of software reliability prediction," in *Fourth International Symposium on Software Reliability Engineering*, 1993, pp. 71-79.

[14] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering,* vol. 25, no. 5, pp. 675-689, 1999.

[15] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," *IEEE Transactions on Software Engineering,* vol. 43, no. 1, pp. 1-18, 2017.

[16] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "An Empirical Study of Software Metrics Selection Using Support Vector Machine," in *23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, 2011, pp. 83-88.

[17] C. Jin, S.-W. Jin, and J.-M. Ye, "Artificial neural network-based metric selection for software fault-prone prediction model," *IET software,* vol. 6, no. 6, pp. 479-487, 2012.

[18] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience,* vol. 41, no. 5, pp. 579-606, 2011.

[19] T. Khoshgoftaar, K. Gao, and A. Napolitano, "Improving software quality estimation by combining feature selection strategies with sampled ensemble learning," in *15th International Conference on Information Reuse and Integration (IRI)*, 2014.

[20] B. Turhan, A. T. Mısırlı, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Information and Software Technology,* vol. 55, no. 6, pp. 1101-1118, 2013.

[21] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Empirical Software Engineering,* journal article vol. 19, no. 1, pp. 154-181, February 01 2014.

[22] L. Kumar, S. K. Sripada, A. Sureka, and S. K. Rath, "Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM)," *Journal of Systems and Software,* 2017/04/20/ 2017.

[23] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing,* vol. 22, no. 4, pp. 9847-9863, 2019/07/01 2019.

[24] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features," *Expert Systems with Applications,* vol. 147, p. 113156, 2020/06/01/ 2020.

[25] H. Aljamaan and A. Alazba, "Software defect prediction using tree-based ensembles," presented at the Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Virtual, USA, 2020. Available: https://doi.org/10.1145/3416508.3417114

[26] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," in *20th International Symposium on Software Reliability Engineering (ISSRE'09)*, 2009, pp. 99-108.

[27] Z. Muzaffar and M. A. Ahmed, "Software development effort prediction: A study on the factors impacting the accuracy of fuzzy logic systems," *Information and Software Technology,* vol. 52, no. 1, pp. 92-109, 2010.

[28] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering,* vol. 17, no. 4-5, pp. 531–577, 2012.

[29] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering,* vol. 38, no. 6, pp. 1276-1304, 2012.

[30] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications,* vol. 38, no. 4, pp. 4626-4636, 2011.

[31] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing,* vol. 27, pp. 504-518, 2015/02/01/ 2015.

[32] T. Kavzoglu and I. Colkesen, *The effects of training set size for performance of support vector machines and decision trees*. 2012.

[33] A. Althnian *et al.*, "Impact of Dataset Size on Classification Performance: An Empirical Evaluation in the Medical Domain," vol. 11, no. 2, p. 796, 2021.

[34] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the nasa metrics data program data sets for automated software defect prediction," in *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 2011, pp. 96-103.

[35] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," *IEEE Transactions on Software Engineering,* vol. 39, no. 9, pp. 1208-1215, 2013.

[36] T. Menzies *et al.*, "The PROMISE Repository of empirical software engineering data," ed, 2012.

[37] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[38] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence,* vol. 97, no. 1, pp. 273-324, 1997.

[39] I. Kononenko, E. Simec, and M. Robnik-Sikonja, "Overcoming the myopia of inductive learning algorithms with RELIEFF," *Applied Intelligence,* vol. 7, no. 1, pp. 39-55, 1997.

[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter,* vol. 11, no. 1, pp. 10-18, 2009.

[41] P. H. Sherrod. (2003). *DTREG predictive modeling software*. Available: http://www.dtreg.com

[42] A. C. Lorena, Andr, #233, and C. P. L. F. d. Carvalho, "Evolutionary tuning of SVM parameter values in multiclass problems," *Neurocomput.,* vol. 71, no. 16-18, pp. 3326-3334, 2008.

[43] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, "Reporting experiments in software engineering," in "Guide to advanced empirical software engineering," Springer2008.

**Appendix A.** Features Description for CM1 and KC1 datasets

| No. | Metric Symbol | Metric Type | Metric Description |
|---|---|---|---|
| 1 | Loc | Numeric | McCabe's line count of code |
| 2 | v(g) | Numeric | McCabe "cyclomatic complexity" |
| 3 | ev(g) | Numeric | McCabe "essential complexity" |
| 4 | iv(g) | Numeric | McCabe "design complexity" |
| 5 | N | Numeric | Halstead total operators + operands |
| 6 | V | Numeric | Halstead "volume" |
| 7 | L | Numeric | Halstead "program length" |
| 8 | D | Numeric | Halstead "difficulty" |
| 9 | I | Numeric | Halstead "intelligence" |
| 10 | E | Numeric | Halstead "effort" |
| 11 | B | Numeric | Halstead |
| 12 | T | Numeric | Halstead's time estimator |
| 13 | lOCode | Numeric | Halstead's line count |
| 14 | lOComment | Numeric | Halstead's count of lines of comments |
| 15 | lOBlank | Numeric | Halstead's count of blank lines |
| 16 | lOCodeAndComment | Numeric | |
| 17 | uniq Op | Numeric | unique operators |
| 18 | uniq Opnd | Numeric | unique operands |
| 19 | totalOp | Numeric | total operators |
| 20 | total_Opnd | Numeric | total operands |
| 21 | branchCount | Numeric | of the flow graph |
| 22 | Defects | Boolean | module has/has not one or more defects |

**Appendix B.** Features Description for PC1 and PC4 datasets

| No. | Metric Type | Metric Description |
|---|---|---|
| 1 | Numeric | LOC_BLANK |
| 2 | Numeric | BRANCH_COUNT |
| 3 | Numeric | CALL_PAIRS |
| 4 | Numeric | LOC_CODE_AND_COMMENT |
| 5 | Numeric | LOC_COMMENTS |
| 6 | Numeric | CONDITION_COUNT |
| 7 | Numeric | CYCLOMATIC_COMPLEXITY |
| 8 | Numeric | CYCLOMATIC_DENSITY |
| 9 | Numeric | DECISION_COUNT |
| 10 | Numeric | DECISION_DENSITY |
| 11 | Numeric | DESIGN_COMPLEXITY |
| 12 | Numeric | DESIGN_DENSITY |
| 13 | Numeric | EDGE_COUNT |
| 14 | Numeric | ESSENTIAL_COMPLEXITY |
| 15 | Numeric | ESSENTIAL_DENSITY |
| 16 | Numeric | LOC_EXECUTABLE |
| 17 | Numeric | PARAMETER_COUNT |
| 18 | Numeric | HALSTEAD_CONTENT |
| 19 | Numeric | HALSTEAD_DIFFICULTY |
| 20 | Numeric | HALSTEAD_EFFORT |
| 21 | Numeric | HALSTEAD_ERROR_EST |
| 22 | Numeric | HALSTEAD_LENGTH |
| 23 | Numeric | HALSTEAD_LEVEL |
| 24 | Numeric | HALSTEAD_PROG_TIME |
| 25 | Numeric | HALSTEAD_VOLUME |
| 26 | Numeric | MAINTENANCE_SEVERITY |
| 27 | Numeric | MODIFIED_CONDITION_COUNT |
| 28 | Numeric | MULTIPLE_CONDITION_COUNT |
| 29 | Numeric | NODE_COUNT |
| 30 | Numeric | NORMALIZED_CYLOMATIC_COMPLEXITY |
| 31 | Numeric | NUM_OPERANDS |
| 32 | Numeric | NUM_OPERATORS |
| 33 | Numeric | NUM_UNIQUE_OPERANDS |
| 34 | Numeric | NUM_UNIQUE_OPERATORS |
| 35 | Numeric | NUMBER_OF_LINES |
| 36 | Numeric | PERCENT_COMMENTS |
| 37 | Numeric | LOC_TOTAL |
| 38 | Boolean | Defective |