Computer and Information Science; Vol. 15, No. 3; 2022
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Homogenous Multiple Classifier System for Software Quality Assessment Based on Support Vector Machine

Udoinyang G. Inyang[1,2], Olufemi S. Adeoye[1,2], Edward N. Udo[1,2], Edidiong F. Bassey[1,2], Enefiok A. Etuk[3], Fidelia N. Ugwoke[3], Emmanuel B. Usoro[1,2]

[1] Department of Computer Science, Faculty of Science, University of Uyo, Nigeria

[2] TETFund Centre of Excellence in Computational Intelligence Research, University of Uyo, Nigeria

[3] Department of Computer Science, Michael Okpara University of Agriculture, Umudike, Nigeria

Correspondence: Udoinyang G. Inyang, Department of Computer Science, University of Uyo, Nigeria.

Received: June 19, 2022 Accepted: July 19, 2022 Online Published: July 25, 2022

doi:10.5539/cis.v15n3p47 URL: https://doi.org/10.5539/cis.v15n3p47

Abstract

In today's society, almost all human endeavours depend on software products. Lack of quality software is one of the software industry's most important problems. Hence, it would be beneficial to access the quality of software to improve and enhance software products while increasing customer satisfaction. This paper assesses software product quality using a Support Vector Machine-based ensemble classifier. The ISO/IEC-9126 (International Organization for Standardization 2001) software quality (SQ) framework was adopted in this work. Dimension reduction of the product metric category dataset and the entire PM dataset was conducted using linear discriminant analysis (LDA). SVM kernel functions (linear, quadratic, cubic, fine gaussian, medium gaussian and coarse gaussian) were used to model each classifier. The combinations of the results from the multiple SVMs used AdaBoost, bagging, and random subspace ensemble methods for the assessment of SQ. All three ensemble learning methods performed better than the individual SVM, however, the bagging stood out with an accuracy of 93.0%. Hence, it was adopted in the fusion of the SVM results and classification of SQ into classes. Results from the confusion matrix and receivers' operating characteristics were greater than 97.99% and confirm significant improvements with an ensemble of homogenous classifiers based on SVM.

Keywords: support vector machine, ensemble learning, software quality, software defect prediction, SVM Kernel

1. Introduction

The world's dependency on software products is increasing due to the progression in the applications of computers and advancements in computer technology. Software Engineering (SE) is an emerging area of computer science because of the dependency on software products by industries, businesses, organizations, and other facets of society. Software Testing (ST) is a critical step in the software development (SD) cycle. Testability is chief among the software quality attributes (SQA) that largely impact software product quality (Anand and Uddin, 2019). ST involves confirmation and authentication of the services of software or its application by assessing whether it conforms to users' expectations. ST ensures that software quality (SQ) is incorporated and enshrined during the SD process. SQ is a long-standing paradigm of software design, it focuses on the design of error-free programs, as well as guaranteeing efficiency when put to use. The concern of improving the quality of software products is motivated by users' increasing demand for efficient and effective software products. Software product quality (SPQ) is the extent to which a system (or components of the process) conforms to documented software functional and performance requirements that are expected by users and other stakeholders (Sharkov & Stoeva, 2021). A quality model for software products includes a set of features and in-between associations that serve as the foundation for stipulating specifications and evaluation. Evaluate models for SQ have been constructed; they define the fundamental features (qualities) and the sub-features that are assigned metrics for the real evaluation. The dearth of error-free software is one of the recurrent problems experienced by software engineers owing to the wide scope and intricacies of the systems (Huda et al., 2018). Hence, it is necessary to assess the quality of software products in order to improve and enhance their effectiveness while increasing users' satisfaction. One of the promising approaches to quality assurance is an analytical investigation of software modules. The early discovery of defective modules is strongly recommended

because it minimizes the effort involved in error corrections at later stages of the software development life cycle (Iqbal et al., 2019).

Machine learning (ML) methodologies have been demonstrated as promising tools for effective and efficient software defect detection and prediction (Iqbal, et al., 2019). These approaches span, supervised (Iqbal, et al, 2019), unsupervised (Ekpenyong and Inyang 2016; Inyang, et al., 2019, Inyang and Inyang, 2011), and hybridized systems (Inyang, and Akinyokun, 2014). Both supervised and unsupervised learning approaches acquire or enhance pieces of knowledge through the training process. The training data comprises input and target records (labelled data) for supervised learning while only samples from the input space (unlabeled data) are required during unsupervised learning. The key goal is to construct an appropriate model(s) that describes relationships, distribution, rules, and trends exhibited by the input features when mapped to the target class. The resultant classifier is then validated through the assignment of class labels to testing instances, where the values of the input samples are given, but value target classes are not provided. Data mining is a significant application of ML while classification task is the widely used data mining technique (Soofi and Awan, 2017). The classification approach is used to predict the class of a specified group of data points. The widely used classification algorithms include Support Vector Machine (SVM), Decision Tree (ID3 and C4.5), K-Nearest Neighbor, Bayesian Network, Naïve Bayes (NB), Multi-Layer Perceptron (MLP) and Random Forest etc. These classification approaches have their strengths and limitations, as they largely rely on the type of data being analyzed and this, therefore, has a relative significance. In Reddivari & Raman (2019) a comparative analysis of software defect and mentainance predictions was performed with eight ML algorithms including SVM, Bayesian Network and Random forest. Although Bayesian Network performed best with an AUC slightly above 84.9%, the score of SVM was the least with an AUC below 50%. In the menatinance dataset, Random Forest stood out with an AUC score of 81.40% while SVM produced a marginal improvement with 68.1% score.

SVM has proven to be one of the prominent and suitable tools for dealing with problems requiring learning, (classification, and prediction of data points) even when the dataset exhibits non-regularity. Once an SVM model is trained, even in small sample size environment, it depicts an acceptable generalization rate even in high dimensional space and can adapt effectively to nonlinear functional relationships that are demanding with other approaches. All these characteristics make SVM appropriate for SQ modeling as such conditions are typically encountered during classification and predictive analytics (Gaye, et al, 2021). This was confirmed by Reshi, & Singh, (2018), when smell prediction model based on SVM was used to predict defects in the subsequent releases of the eclipse software. The lack of diversity during decision-making with a single SVM classifier system is a major determinant of the degraded performance of SVM, especially when presented with a large number of input features. This requires a method of combining more than one learning algorithm ─. Ensemble Learning (EL). EL consists of several meta-algorithms that can combine many ML approaches into one predictive model in order to minimize variance, bias, or improve predictions. This will produce the diversity of predictions made by member classifiers which in turn guarantees the ensemble's ability to outperform the best individual classifiers (Ponti, 2011). This shows that predicting ensemble classifiers for software quality methods have not been fully exploited. Therefore, there is a need for a comparative analysis of multiple SVMs using ensemble methods on software quality product metrics datasets.

This research focuses on the assessment of software product quality based on multiple SVM and also adopts EL mechanisms to combine the results. The utilization of ML algorithms has proven to be of great practical value in solving diverse problems in SE including the prediction of failures, faults and fault-proneness (Sharma and Mona, 2016). Degraded SQ leads to dissatisfaction, and requires a certain cost to revert to its original quality. This, therefore, ascertains the need to initially assess the quality of software through intelligent approaches, before being put to use. The performance of single classifiers in pattern recognition problems especially on high dimensionality and noisy data sets, and a large number of classes is poor and unreliable. This paper aims at combining multiple SVMs using ensemble base learners to assess and predict with high accuracy, the software product quality. A review of related literature is described in Section 2, while the system framework and methodological workflow are presented in Section 3. Section 4 has the results of the implementation and discussion while conclusions are drawn in Section 5.

2. Related Works

2.1 Software Quality (SQ)

SQ is an abstract concept that is conceptualized and described differently based on one's personal view and interest. In order to lessen this ambiguity, ISO/IEC-9126 (International Organizational for Standardization, 2001) provides a framework for the evaluation of SQ (Agu and Bakpo, 2011). Six SQ attributes are defined and

grouped into sets of sub-characteristics to really assess software product quality. Functionality ─ the competence of particular software to provide services that will meet the explicitly stated and implied expectations of users within some specified conditions of usage, reliability (property of software to sustain its level of performance over a specified period of time), usability (the ease of comprehensibility, learnability, interpretability, and usage under specified conditions of usage), efficiency (provision of expected performance level, based on available resources and under some stated conditions), maintainability (the ease of modification, improvements or adaptations on the software), portability (the ease of transferring the software from one environment to another). Khan & Ansari, (2020) adopted multi-criteria decision-making and fuzzy sets approach to predict software defects due to the high dimensionality of software attributes and the complexity of the process.

Static Code analysis is expensive and ineffective since it is mostly carried out manually; making the localization of code issues an arduous task while requiring a longer time to eliminate the bugs. Several works on SQ assessment have been performed; Farid et al. (2017), propose gauging the quality of software products using product metrics in order to provide a roadmap for developing software products within allocated time and cost without compromising SQ. A review and analysis of software complexity metrics in structural testing are reported in Debbarma, et al., (2013) due to the inability to measure customer satisfaction not only at the delivery but at other stages of the development process. This was carried out to discover how to minimize software development costs and improve testing efficacy and quality by testers (developers). Static analysis of software complexity metrics like size and control flow metrics was carried out on four program characteristics that are responsible for complexity measures; Lines of Code (LOC), McCabb's complexity metrics (MCC), Halstead's Software Science Complexity (HSSC), and NPATH Complexity (NC), with programs written in C. The result showed that static analysis could lead to a reduction in software development cost, improvement of testing efficacy, and software quality by evaluating software complexity metrics with LOC, MCC, HSSC and NC.

2.2 Support Vector Machine

SVM is a member of the most active supervised learning classifiers and a widely used tool in many application domains. Support vectors are samples of any dataset that are positioned to be closest to the decision surface. SVM discovers a line that optimally distinguishes between data from different classes (hyperplane) and is represented as points in space. This is achieved by building a model using examples in the training dataset and thereafter classifying unknown instances through the built model. It has the potential for the classification of data vectors through hyperplane in massive dimensionality space. The main strength of SVM lies in its efficiency in dealing with diverse classification tasks, especially those characterized by an increased amount of input vectors, non-linearly separable patterns, missing values, and inconsistencies in data vectors (Jun, 2021). The requirement of key parameters before achieving excellent classification results reduces the acceptance rate of SVMs. Singh et al. (2009) applied SVMs for the prediction of software fault proneness by modelling the association between fault proneness and object-oriented based metrics. The work confirms that the construction of SVM models is feasible, adaptable to object-oriented systems, and useful in predicting fault-prone classes. Xing et al., (2005) presented a novel method for SQ prediction based on SVM. The work was motivated by the lack of an SVM approach for the prediction of SQ in the software engineering field. The objectives were to adopt SVM in the classification of software modules based on complexity metrics and explain the Transductive SVM (TSVM) approach that considers particular test samples as well as training samples in building classifiers. Although TSVM achieved the best correct classification rate of 90.03% in the early detection of software errors, it consumes more training time than other methods.

Gupta (2015) used SVM based technique to predict software defects in NASA's Promise datasets by developing Matrix Laboratory (MATLAB) interface. Training and testing were done using ANT 1.7 datasets (Khuat, & Le, 2019). The accuracy of the prediction was above very good 89.0%. Rong et al., (2016) applied a centroid-based Bat Algorithm (CBA) to optimize SVM parameters and produced a CBA-SVM model to find out the most error-prone modules in software accurately for timely correction. This model performed better when compared with other models because it takes advantage of the non-linear computing ability of the SVM model and the optimization capacity of the bat algorithm with centroid strategy.

Thangaval and Pugazendi (2017) also proposed an optimized Minimum Redundancy – Maximum Relevance (MRMR) and SVM with a firefly algorithm to predict software defects using KC1 datasets, which is National Aeronautics and Space Administration (NASA) software metrics data program that supports predictive software engineering models. The proposed model improved classification accuracy. In (Iqbal et al., 2019), a performance-wise comparative analysis of the various supervised ML approaches on software defect detection and utilized twelve (12) NASA datasets for experimentations. Recall, F-measure, precision produced a positive

correlation while Accuracy and ROC did not show a negative reaction on the class imbalance issue. Utilizing SVM in regression analysis for time series prediction was implemented by automatically varying and estimating the loss function parameter rather than using the standard SVM regression model. The main aim was to use multiple SVM models rather than using a global model was proposed. Each of the classifier models used different datasets for training. The results showed that training multiple SVMs rather than one global learning model leads to a significant enhancement of classification performance (Evgeniou & Pontil, 2001). SVM based hybidized systems for software defect prediction have been demonstrated in Cai et al, (2020), Alsghaier & Akour, (2020), Prabha & Shivakumar, 2020) and Rhmann et al, (2020).

2.3 Ensemble Learning

Ensemble learning (EL) is a paradigm in ML that pursues healthier predictive performance through the combination of predictions from a set of multiple component models when dealing with future instances. Since an ensemble achieves high precision than its constituent learners, it has become topical in recent years. Regarding the pattern of training base learners, current ensemble algorithms fall into two classes; sequential and parallel training. These two approaches shrink the variance of each estimate via a combination of more than one estimate from diverse models. An instance of the first class is Adaboost, which provides a combination of predictions that belong to different classifiers while bagging, an example of the latter, combines predictions that belong to the same type (Gonz á lez, 2020).

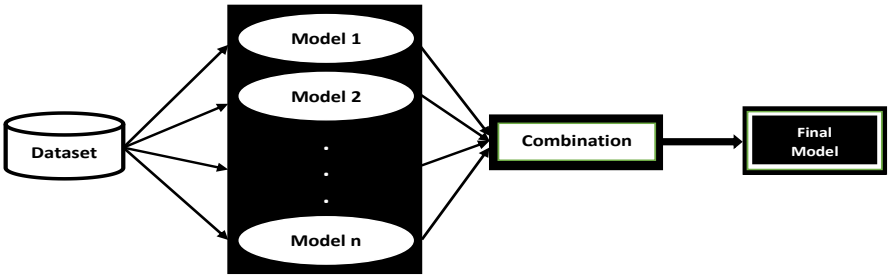


Figure 1. A generic structure of EL

Recently, researchers have reported on SQ assessment based on EL technique; focusing on the development of an assessment framework for SQ using a more suitable SVM kernel function with an EL method. Karthikeyan and Veni (2016) applied an improved SVM classifier to model and predict software defects. They were motivated by low prediction accuracy and high false alarm associated with predictions using SVM. The work aimed at proposing a new algorithm for software defect prediction using an improved support vector machine (ISVM) classifier. The PROMISE software engineering repository was used to get datasets on CMI, KCI, and PCI datasets with the following input vectors; number of attributes, number of total instances, and number of defect-prone modules were used. From the result, it was shown that ISVM attained better software defect prediction accuracy than SVM in addition to false alarm rate reduction. Asmita and Shukla (2014) reviewed the structure, algorithm, and combinational strategies of EL algorithms. In (Shanthini et al., 2014) the effect of hybridization of principal component analysis (PCA) and SVM in software fault prediction was proposed. The work was motivated by poor classification performance of single SVM models in terms of classification rate on software fault prediction, aimed at analyzing the performance of some feature selection strategies using bagged SVM classifier for various metrics level datasets on fault prediction. The objectives were to analyze each filter performance with root mean square error (RMSE) and receiver operative characteristic (ROC). The result from the ROC curve showed that, across the metrics level dataset, hybridized SVM produces improved performance in terms of classification rate. Shanthini and Chandrasekaran (2013) presented the results from ensemble methods used for metrics-level software fault prediction. The comparative analysis on different ensemble approaches in relation to different aspects of taxonomy was conducted using Waikato Environment for Knowledge Analysis (WEKA) environment. The following ensemble methods were compared; Bagging, Boosting, Stacking, and Voting methodologies, using class, method, and package levels on software metrics dataset using RMSE. Alhazmi and Khan (2020) adopted EL bagging as the base learner, with Linear regression, sequential minimal optimization regression (SMOReg), Muli-Layered Perceptron, random forest, reduced error pruning tree (REPTree), and M5 Rule for prediction of software effort. The dataset used is China dataset with 499 ×19 feature spaces; (18 autonomous variables and 1 ward variable), while each row corresponds to a project). The feature selection (FS) algorithms; BestFit and Genetic Algorithm was implemented to examine their functionality. The results showed that the Mean Magnitude Relative error of the Bagging M5 rule with Genetic

Algorithm as FS is 10%, which makes it outperforms other algorithms.

3. Methodology

This system is designed as the integration of components that will enable preprocessing of SQ attributes, classification, and ranking of the software quality product metrics, as well as combining the classifiers through the base learners. The system framework is depicted in Figure 2 and consists of the database, multiple classifier sub-system, inference engine, and ensemble learners. The database component holds values of SQ metrics. The system undergoes five stages during implementation: 1) dataset selection, 2) feature Selection, 3) dimension reduction 4) classification, and 5) aggregation. A detailed description of the stages is given in the following sections:

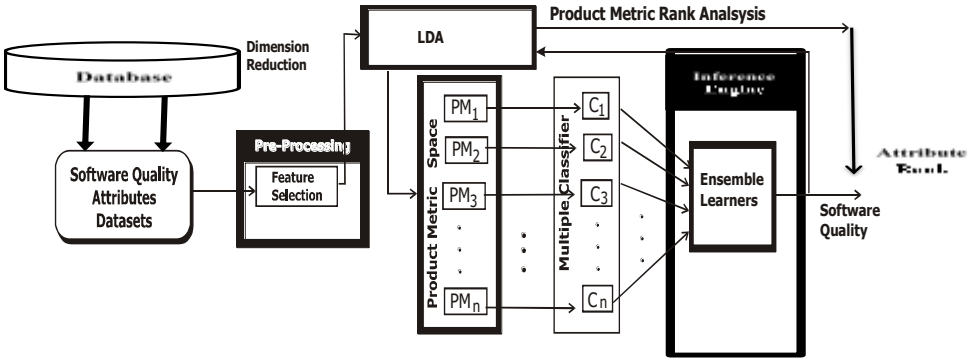


Figure 2. System Framework

3.1 Dataset Description and Pre-processing

The dataset which consists of one thousand, five hundred (1500) instances, representing product metric of software, was collected from PRedictOr Models In Software Engineering (PROMISE) software engineering repository ─ an online repository produced by the School of Information Technology and Engineering, University of Ottawa, Canada, to make datasets and models accessible by software engineering community (Shippey, et al, 2016, Colakoglu, et al, 2021).

Table 1. Attributes of Software Product Quality Dataset (Colakoglu et al, 2021)

S/N	Field Code	Description	Type
1	WMC	Weighted Methods Per Class	Input
2	NOC	Number of Children	Input
3	RFC	Response for A Class	Input
4	CBO	Coupling Between Objects	Input
5	DIT	Depth of Inheritance Tree	Input
6	LCM	Lack of Cohesion On Methods	Input
7	NOM	Number of Methods	Input
8	NCS	Number of Classes	Input
9	LOC	Lines of Code	Input
10	COP	Comment Percentages	Input
11	CYC	Cyclomatic Complexity	Input
12	QDX	Quality Index	Output

The software product metrics space comprises; usability, maintainability, functionality, efficiency, reliability, and portability categories of attributes. A data point in the repository corresponds to a single software package and is described by a set of input variables - WMC, NOC, RFC, CBO, DIT, LCM, NOM, NCS, LOC, COP, CYC, and output variables - QDX. Definitions of the input and output features are provided in Table 1. The mapping of the attributes to the software product metric space is presented in Fig. 6.

As shown in Fig. 3, DIT belongs to three product metric categories while NOC and NC are members of the maintainability and reliability categories respectively. CBO is a member of three product categories as well as LOC. Dataset pre-processing eliminates irrelevant and redundant attributes from the original dataset (Tsai and Chen, 2010), thereby lessening the dimensionality, computational complexity, and enhancing interpretation and visualization efforts of the dataset. Several studies have investigated the effectiveness of feature selection methods on the performance of defect prediction models (Almayyan, 2021). Linear Discriminant Analysis (LDA) is used for feature dimension reduction and rank analysis (Dada et al. 2021). It discovers a linear mapping of features that can effectively discriminate different classes, via exploitation of parameters relating to inter-class while those associated with intra-class are minimized simultaneously. The LDA realizes good outcome,

especially when the population is characterized with a normal joint distribution, and produces a consistent covariance for different classes (Dada *et al*, 2021). One reasonable drawback of LDA is that it is exceptionally susceptible to outliers. The model of LDA adopted in this work is as described in Dada *et al*, (2021) was used to transform the SQ dataset.

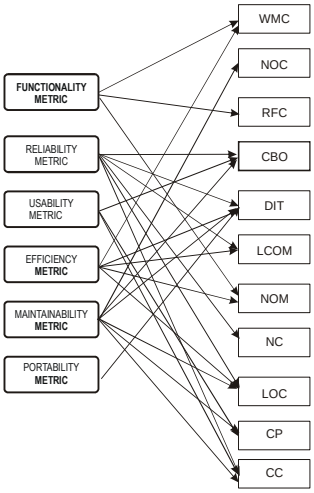


Figure 3. Attributes mapping to product metrics categories

3.2 SVM Classifier System

The SVM classifier has a committee of six SVMs, each acting on a category of SQ dataset. Table 2 gives a description of the SVM model in relation to their respective dataset.

Table 2. SVM models and associated dataset category

S/N	SVM model	Dataset category
1	SVM_M	Maintainability attribute dataset
2	SVM_E	Efficiency attribute dataset
3	SVM_F	Functionality attribute dataset
4	SVM_R	Reliability attribute dataset
5	SVM_U	Usability attribute dataset
6	SVM_P	Portability attribute dataset

The SVM Classifier system performs supervised learning, through the implementation of multiple SVM models, each on their respective LDA transformed SQ dataset. This runs in three phases; train, test and validate, after the assignment of datasets. The datasets were randomly partitioned into three in the ratio of 8:1:1, as follows: train (80%) yielding 1200 instances, test (10%) yielding 150 instances, and validating (10%) yielding 150 instances. Multiple SVMs with different kernel functions ─ linear, quadratic, cubic, fine gaussian, medium gaussian and coarse Gaussian, were employed with ensemble base learners ─ bagging, AdaBoost, and random subspace methods, for training the homogenous set of classifiers.

4. Results and Discussion

Each SVM model acts on a software product metric category in three phases; training, testing, and validation, after the assignment of datasets. Every phase has parameters tuning while the individual dataset was randomly partitioned into three; training (80%) yielding 1200 instances, testing (10%) yielding 150 instances, and validating (10%) yielding 150 instances. The SVM classification process spans several iterations involving different kernel-function and regularization tuning to achieve greater accuracy on each of the categories of the transformed SQ datasets. Table 3 and Fig. 4 show the performance of SVM models on their respective datasets with six (6) kernel functions. Although the fine Gaussian kernel depicted the worst performance across product metric categories (average accuracy of 75.0%), other kernels yielded accuracy > 79.9% across all dataset configurations. The linear kernel was outstanding in all dataset scenarios (average accuracy = 85.6%) and yielded the best accuracy of 86.2% when trained with the SVM_M model.

Table 3. Kernel-based Performance of SVM models on SQ classification

Kernel Function	SVM Models					Total	Average
	SVM_M	SVM_E	SVM_F	SVM_R	SVM_U		
Linear	86.6	86.2	85.2	85.0	85.7	428.7	85.7
Quadratic	86.1	85.2	84.9	83.5	85.2	424.9	85.0
Cubic	82.7	82.5	85.1	79.9	85.3	415.5	83.1
Fine Gaussian	67.5	72.1	82.7	67.7	84.8	374.8	75.0
Medium Gaussian	85.5	83.9	84.7	82.4	85.3	421.8	84.4
Coarse Gaussian	84.2	83.7	84.1	81.5	85.1	418.6	83.7
Total	492.6	493.6	506.7	480	511.4	2484.3	496.9
Average	82.1	82.3	84.5	80	85.2	414.1	82.8

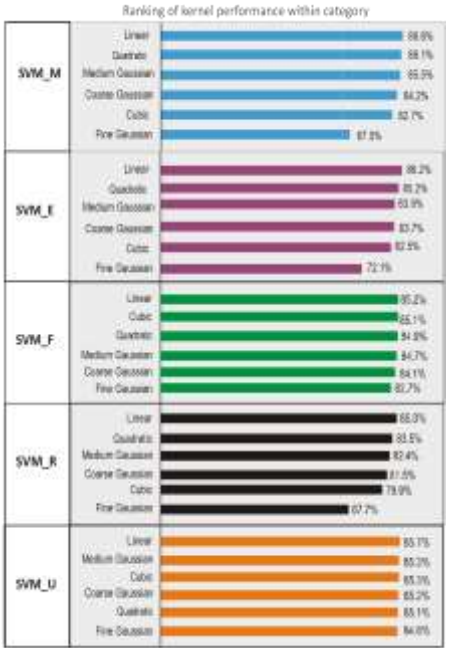


Figure 4. Visualized ranking of SVM-kernel performances within the category

Usability category reveals negligible differences in the performances of SVM kernels (between 84.8% and 85.7%), producing the least standard deviation (±0.29) closely followed by functionality (SD ±0.94). The product metric category that produced the highest difference in kernel performances is maintainability (SD = ±7.29). A statistical test for significant difference, at a 95% confidence level, between the mean performances of kernels across product metric levels, yielded (df=5, f=6.02, p=0.001) depicting a statistically significant difference while insignificantly different values are obtained from SVM datasets (df=4, f=2.49, p=0.076). Turkey's post hoc test (Inyang *et al*, 2021) confirms the Fine Gaussian kernel (mean=75.0%) as producing statistically significantly different performances. This means all kernels except Fine Gaussian produce performances that do not differ significantly and are candidates for better performances.

Three methods; AdaBoost, bagging, and random subspace, were adopted for combining the six-member committee of SVM learners' outputs at a learning rate = 0.1, number of learners = 6, Number of splits = 20. EL design followed an error-correcting output codes (ECOC) model where the multi-classification problem was reduced to a set of binary classifiers via One versus One (OVO) coding design having k(k-1)/2 binary learners (BL), where k is the number of classes. The resultant BL coding matrix is described in Table 4. Each row represents the state of each SQ class with its associated BL.

Table 4. Coding design for the SVM BL

Ordered Class Names	BL									
	BL 1	BL 2	BL 3	BL 4	BL 5	BL 6	BL 7	BL 8	BL 9	BL 10
High	1	1	1	1	0	0	0	0	0	0
Low	-1	0	0	0	1	1	1	0	0	0
Moderate	0	-1	0	0	-1	0	0	1	1	0
Very High	0	0	-1	0	0	-1	0	-1	0	1
Very Low	0	0	0	-1	0	0	-1	0	-1	-1

As shown in Table 5 and Figure 5, the performance of the EL methods green-lights bagging (with an accuracy of 93.0%), as the best approach followed by random subspace (92.7%). In terms of the training cost, the higher the accuracy the less time used for training.

Table 5. Performance evaluation for Ensemble methods

Learner's Method	Accuracy	Training Time (seconds)
AdaBoost	92.3%	27.864
Bagging	93.0%	5.9008
Random Subspace	92.7%	10.412

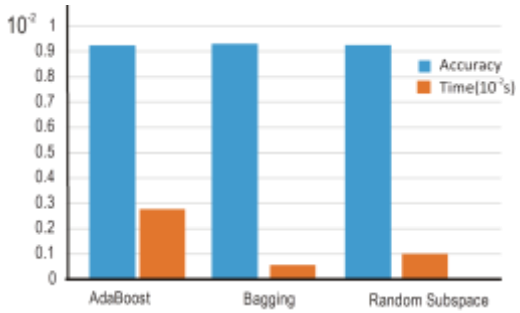
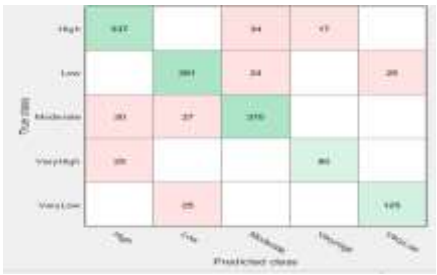


Figure 5. Comparison of performances of EL methods

It can be concluded that all three learning methods performed better than the individual SVM models whose best performance stood at 86.6%. This further confirms the fact that EL improves the prediction levels of classifiers especially when bagging is the base learner.

4.1 SVM Classifiers' Performance Analysis

The classification of the SQ demonstrated in this paper adopts linear kernel function, box constraint of 1, a learning rate of 0.01 as parameters and OVO pairwise strategy for the classification of SQ. The SQ index comprises five classes ─very high, high, moderate, low and very low. The confusion matrix and ROC curve were tools used for the performance assessment of each SVM model. The SVM_E model which earns an accuracy of 86.2% with a duration of 60.875 seconds for training (Fig. 6). Data points numbering 337 instances (87%) in the High class were correctly classified while 13% (51) were incorrectly classified. Out of 430 data points in the Low class, 89% (381) were correctly classified as Low while 11% (49) were incorrectly classified (6% as members of moderate class and 6% as member of Very Low Class). The number of cases in the moderate, Very High and Very Low that were correctly classified is 370 (87%), 80 (76%) and 125 (83%) respectively. This shows a satisfactory performance of the SVM_E classifier, therefore suitable for the prediction of Efficiency of a Software Product. The accuracy of all SVM models are summarized in Table 6 and Fig. 6.



a.) Number of observations

b.) TPR/TNR

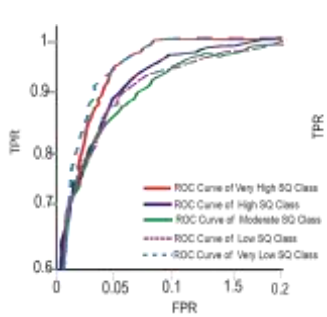
Figure 6. Confusion matrix for SVM_E

Table 6. Correctly predicted instances by SVM models

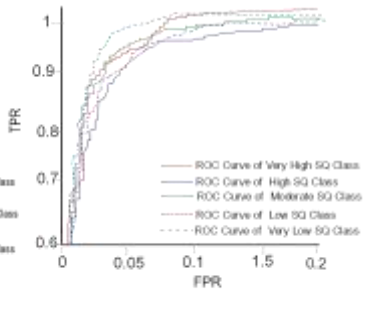
	SVM_M	SVM_E	SVM_F	SVM_R	SVM_U	SVM_P
Very High	76	82	74	74	76	78
High	86	86	85	85	86	88
Moderate	88	86	86	85	86	86
Low	89	88	87	88	88	90
Very Low	87	83	85	83	85	79

Similar results were obtained from other SVM classifiers. The very high class yielded the lowest percentage of correctly classified software defects in each SVM model whereas the best classification rate points to the Low class across SVM models. The overall assessment of the performances of the SVM models regarding the accuracy of the classifications, all the models performed satisfactorily. The poor performance recorded when instances of the Very High Class are predicted may not be unconnected with limited training samples due class imbalance. It therefore calls for the treatment of the dataset before classification ROC curves (Fig.7a-f) were used to measure the performances of the SVM models in predicting individual classes. Values from three ROC parameters; Sensitivity (SE), Specificity (SP) and AUC, were obtained and presented in Table 7.

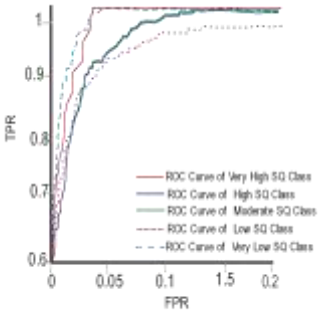
The area AUC, summarizes the overall location of the ROC curve and quantifies the ability of the bagging method to discriminate between SQ classes. As shown in Table 6 and Fig. 7a-f, the AUC values in the range of 97% to 99% ($97 \leq AUC \leq 99$), portray strong evidence of the diagnostic capacity of SVM models in discriminating among the five classes of SQ. The within-class AUC are the same for SVM_F, SVM_E, and SVM_U classifiers; and SVM_M and SVM_P have common values across classes. Across classes and SVM models, the AUC values are marginally different. This was confirmed via a test of significance result $p>0.5$ at 95% confidence intervals.



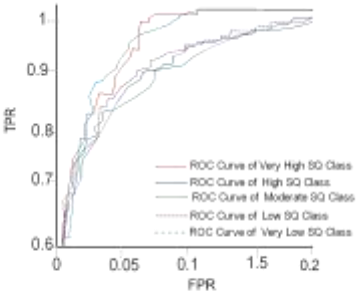
a) ROC Curve for SVM F



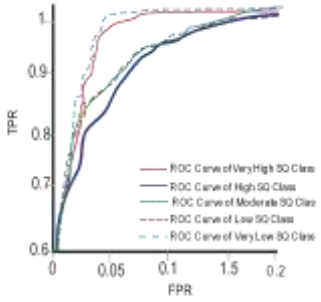
b)) ROC Curve for SVM_E



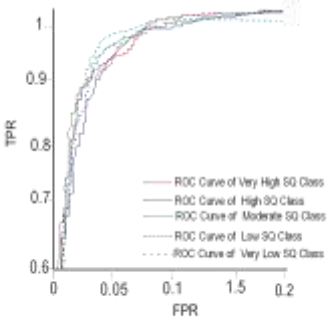
c) ROC Curve for SVM_R



d) ROC Curve for SVM_U



e.) ROC Curve for SVM_M



f.) ROC Curve for SVM_P

Figure 7. ROC Curves for SVM Models

Table 7. Summary of SVM classifiers' Performances

SQ Index	SVM_F		SVM_E		SVM_R		SVM_U		SVM_M		SVM_P		Mean	
	SE (SP) %	AUC (%)	SE (SP) %	AUC %	SE (SP) %	AUC %	SE (SP) %	AUC (%)	SE (SP) %	AUC %	SE (SP)%	AUC %	SE (SP) %	AUC
Very High	74 (99)	99	76 (99)	99	74 (83)	99	76 (99)	99	76 (86)	99	78 (98)	99	75.67 (94)	99.0
High	85 (95)	97	87 (95)	97	97 (95)	97	86 (95)	97	86 (95)	98	88 (95)	98	88.17 (95)	97.3
Mod-erate	86 (94)	97	87 (95)	97	85 (94)	97	86 (94)	97	88 (96)	98	86 (96)	98	86.33 (94.8)	97.3
Low	87 (95)	98	89 (95)	98	88 (95)	97	88 (95)	98	99 (96)	99	90 (94)	98	90.16 (95)	97.8
Very Low	85 (98)	99	83 (98)	99	83 (98)	99	85 (98)	99	87 (98)	99	79 (98)	99	83.67 (98)	99.0
Mean	83.4 (96.2)	98	84.4 (96.4)	98	90.4 (93)	97.8	84.2 (96.2)	98	87.2 (94.2)	98.4	84.2 (96.2)	98.4		

In terms of SE (True Positive Rate), although lower scores (74≤SP≤78) are associated with the Very High class and better results (SP≥85) are exhibited by the other classes of SQ, the within-class difference (df=4, f=18.95, p=0.000) is statistically significant as against across product metric levels which do not depict statistically difference in mean values (df=5, f=0.19, p=0.964). A post-hoc pair-wise comparison test using the Turkey method reveals that the SE associated with the prediction of the Very High SQ class (mean=75.667) is significantly lower than others classes at P>0.05 confidence interval. The rate at which, software defects that do not belong to a particular class are identified correctly as not members of the specified class (specificity) performed significantly well, both at product metric and SQ index levels, without any statistically significant difference (p<0.05). This implies that SVM models minimize false negative ratio and therefore performed better when discriminating software defect instances.

4.2 Ensemble Classifiers' Performance Analysis

Bagging is the adjudged best-performing methodology with an accuracy of 93.0%. The assessment of the overall performance of the system vis-à-vis EL methods is based on the bagging method with linear kernels of the SVM models. The confusion matrix shows that individual class 93% (361) instances of data points in the High SQ class were correctly classified while 7% (27) were incorrectly classified. Out of 430 data points in the Low SQ class, 91% (393) were correctly classified as Low while 9% (37) were incorrectly classified (4% as members of the moderate class and 4% as members of the Very Low class). The number of cases in the moderate, very high and very low that were correctly classified are 373(87%), 67 (64%) and 123 (82%) respectively. This shows a satisfactory performance of the SQ Bagging ensemble method, therefore suitable for the prediction of SQ index.

a) Confusion Matrix (Numbers) b) Confusion Matrix (Percentage)

Figure 8. Confusion Matrix for Homogenous Ensemble Classifier

The AUC from The results depicts a near perfect classification of the respective classes in the SQ using Bagging method. For Very Low and Very High classification, the AUC is 0.99. This implies that the probability of classifying instances of SQ correctly as members of Very High and Very Low classes respectively is 0.99 at 95% confidence level. Very High SQ class also shows specificity of 100%, followed by Very Low SQ class of 99%. The specificity and sensitivity values are also excellent as presented in Table 8. These results show that the Bagging ensemble method outputs are suitable for SQ assessment and prediction. The result reported in this work indicate a signification improvment on results of previous works as described in Table 9. The proposed methodology yields the highest AUC indicating best performance of homogenous ensemble of SVM in the prediction of software defects.

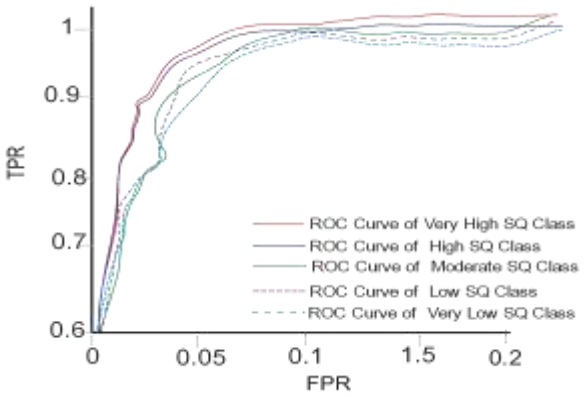


Figure 7. ROC curve for EL

Table 8. ROC Performance values for SQ Bagging EL

SQ Index	SE (%)	SP (%)	AUC (%)
Very High	64	100	99
High	93	94	98
Moderate	87	96	98
Low	91	95	98
Very Low	82	99	99

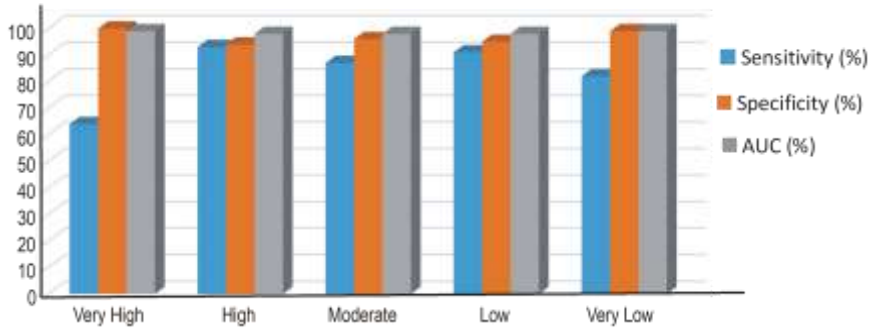


Figure 8. Visualized ROC Performance values for SQ Bagging EL

Table 9. Comparison of the proposed SVM Ensemle with previous works based on SVM

Work	Approach	AUC
Xing et al., 2005	SVM	90.03
Khuat & Le (2019)	SVM	89.0%
Reddivari & Raman, 2019	SVM	49.9%
Shanthini et al, 2013	Bagged SVM	0.79%
Reshi & Singh, 2018	SVM based Smell-Defect	0.652%
Proposed approach	SVM ensemble	93.0%

5. Conclusion

Quality is one of the key factors in the market growth and success of any product. It would be beneficial to assess the quality of software in order to improve and justify the dependency on software products while increasing customers' satisfaction. The experiment was characterized by six SVM kernel functions (KF) — linear, quadratic, cubic, fine gaussian, medium gaussian and coarse Gaussian and three EL base learners (AdaBoost, Bagging and Random Subspace). The results indicate that linear KF performed better than other KFs with an average accuracy of 85.7%. All three EL methods performed better than the individual SVM models, however, the Bagging approach had the highest accuracy of 93.0%. It was therefore used for SQ class discrimination into very high, high, moderate, low and very low classes. Moreso, linear kernel and Bagging learning methods are more suitable for the assessment of SQ. As further work, the imbalance nature of instances of SQ classes would be corrected and the effectiveness of different feature selection methods for predicting software defects prediction would be investigated.

Acknowledgements

We would like to thank the Tertiary Education Trust Fund (TETFund) for their support and funding of this research through the TETFund Centre of Excellence in Computational Intelligence Research, University of Uyo and the University of Uyo management for the enabling environment.

References

Agu, M., & Bakpo, F. (2011). Lessons and challenges from software quality assessment: The case of space systems software. *Global Journal of Pure and Applied Sciences*, *17*(1), 93-99.

Alhazmi, O., & Khan, M. (2020). Software Effort Prediction using Ensemble Learning Methods. *Journal of Software Engineering and Applications, 13*(7), 143-160. https://doi.org/10.4236/jsea.2020.137010

Almayyan, W. (2021). Towards Predicting Software Defects with Clustering Techniques. *International Journal of Artificial Intelligence and Applications (IJAIA), 12*(1). https://doi.org/10.5121/ijaia.2021.12103

Alsghaier, H., & Akour, M. (2020). Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier. *Software: Practice and Experience, 50*(4), 407-427. https://doi.org/10.1002/spe.2784

Anand, A., & Uddin, A. (2019). Importance of Software Testing in the Process of Software Development. *International Journal for Scientific Research and Development, 12*(6).

Asmita, S., & Shukla, K. K. (2014). Review on the architecture, algorithm and fusion strategies in ensemble learning. *International Journal of Computer Applications, 108*(8), 21-28. https://doi.org/10.5120/18932-0337

Cai, X., Niu, Y., Geng, S., Zhang, J., Cui, Z., Li, J., & Chen, J. (2020). An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. *Concurrency and Computation: Practice and Experience, 32*(5), e5478. https://doi.org/10.1002/cpe.5478

Colakoglu, F. N., Yazici, A., & Mishra, A. (2021). *Software product quality metrics: A systematic mapping study.* IEEE Access. https://doi.org/10.1109/ACCESS.2021.3054730

Dada, E. G., Oyewola, D. O., Joseph, S. B., & Duada, A. B. (2021). Ensemble Machine Learning Model for Software Defect Prediction. *Advances in Machine Learning & Artificial Intelligence, 2*(1), 11-21. https://doi.org/10.33140/AMLAI.02.01.03

Debbarma, M. K., Debbarma, S., Debbarma, N., Chakma, K., & Jamatia, A. (2013). A review and analysis of software complexity metrics in structural testing. *International Journal of Computer and Communication Engineering, 2*(2), 129-133. https://doi.org/10.7763/IJCCE.2013.V2.154

Ekpenyong, M. E., & Inyang, U. G. (2016). *Unsupervised Mining of Under-resourced Speech Corpora for Tone Feature Classification.* International Joint Conference on Neural Networks (IJCNN), 2374-2381. https://doi.org/10.1109/IJCNN.2016.7727494

Evgeniou, T., & Pontil, M. (2001). Support vector machines: Theory and applications. In *Advanced Course on Artificial Intelligence* (pp. 249-257). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44673-7_12

Farid, S., Alam, M., Akbar, A., Iqbal, M. M., & Siddiqui, F. A. (2017). Gauging the quality of software products using metrics. *Technical Journal, University of Engineering and Technology (UET) Taxila, 22*(1), 121-127.

Gaye, B., Zhang, D., & Wulamu, A. (2021). Improvement of support vector machine algorithm in big data background. Mathematical Problems in Engineering, 2021. https://doi.org/10.1155/2021/5594899

Gonz ález, S., Garc á, S., Del Ser, J., Rokach, L., & Herrera, F. (2020). A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. *Information Fusion, 64*, 205-237. https://doi.org/10.1016/j.inffus.2020.07.007

Gupta, Rishu (2015). Software Defects Prediction using Support Vector Machine. *International Journal of Computer Science and Software Engineering, 1*(1), 39-48.

Huda, S., Alyahya, S., Ali, M. A., Abawajy, V., Al-Dossari, J., & Yearwood, J. (2018). Framework for Software Defect Prediction and Metric Selection. *IEEE Access, 6*, 2844-2858. https://doi.org/10.1109/ACCESS.2017.2785445

Inyang, U. G., & Akinyokun, O. C. (2014). A hybrid knowledge discovery system for oil spillage risks pattern classification. *Artificial Intelligence Research, 3*(4), 77-86. https://doi.org/10.5430/air.v3n4p77

Inyang, U. G., & Inyang, M. U. (2011). A Neuro-Fuzzy Decision Support Framework for Tenants' Satisfaction Assessment in Residential Properties. *World Journal of Applied Science and Technology, 3*(1), 146-154.

Inyang, U. G., Robinson, S. A., Ijebu, F. F., Udo, I. J., & Nwokoro, C. O. (2021). Optimality Assessments of Classifiers on Single and Multi-labelled Obstetrics Outcome Classification Problems. *International Journal of Advanced Computer Science and Applications, 12*(2). https://doi.org/10.14569/IJACSA.2021.0120260

Inyang, U., Umoh, U., Nnaemeka, I., & Robinson, S. (2019). Unsupervised Characterization and Visualization of Students' Academic Performance Features. *Computer and Information Science; 12*(2). https://doi.org/10.5539/cis.v12n2p103

Iqbal, A., Aftab, S., Ullah, B. M., & Saeed, M. (2019). A Feature Selection based Ensemble Classification Framework for Software Defect Prediction. *International Journal of Modern Education and Computer Science, 9*, 54-64. https://doi.org/10.5815/ijmecs.2019.09.06

Jiang, Y., & Zhou, Z. H. (2004). SOM ensemble-based image segmentation. *Neural Processing Letters, 20*(3), 171-178. https://doi.org/10.1007/s11063-004-2022-8

Jun, Z. (2021). The development and application of support vector machine. *Journal of Physics: Conference Series, 1748*(5), 052006. IOP Publishing. https://doi.org/10.1088/1742-6596/1748/5/052006

Karthikeyan, M., & Veni, S. (2016). Software defect prediction using improved Support Vector Machine Classifier. *International Journal of Mechanical Engineering and Technology (IJMET), 7*(5), 417-421.

Khan, M., & Ansari, M. D. (2020). Multi-criteria software quality model selection based on divergence measure and score function. *Journal of Intelligent & Fuzzy Systems, 38*(3), 3179-3188. https://doi.org/10.3233/JIFS-191153

Khuat, T. T., & Le, M. H. (2019). Binary teaching–learning-based optimization algorithm with a new update mechanism for sample subset optimization in software defect prediction. *Soft Computing, 23*(20), 9919-9935. https://doi.org/10.1007/s00500-018-3546-6

Ponti Jr, M. P. (2011). Combining classifiers: from the creation of ensembles to the decision fusion. In *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2011 24th SIBGRAPI Conference on*, 1-10p. IEEE. https://doi.org/10.1109/SIBGRAPI-T.2011.9

Prabha, C. L., & Shivakumar, N. (2020, June). Software defect prediction using machine learning techniques. In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184) (pp. 728-733). IEEE. https://doi.org/10.1109/ICOEI48184.2020.9142909

Reddivari, S., & Raman, J. (2019). *Software quality prediction: an investigation based on machine learning.* In 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI) (pp. 115-122). IEEE. https://doi.org/10.1109/IRI.2019.00030

Reshi, J. A., & Singh, S. (2018). *Predicting software defects through SVM: an empirical approach.* arXiv preprint arXiv:1803.03220.

Rhmann, W., Pandey, B., Ansari, G., & Pandey, D. K. (2020). Software fault prediction based on change metrics using hybrid algorithms: An empirical study. *Journal of King Saud University-Computer and Information Sciences, 32*(4), 419-424. https://doi.org/10.1016/j.jksuci.2019.03.006

Rong, X., Li, F., & Cui, Z (2016). A Model for Software Defect Prediction using Support Vector Machine based on CBA. *International Journal of Intelligent Systems Technologies and Applications, 15*(1), 19-34. https://doi.org/10.1504/IJISTA.2016.076102

Shanthini, A., & Chandrasekaran, R. M. (2013). Effect of Ensemble Methods for Software Fault Prediction at Various Metrics Level. *International Journal of Applied Information Systems (IJAIS), 5*(2), 51-55.

Shanthini, A., Chandrasekaran, R. M., & Vinodhini, G. (2014). Effect of Principle Component Analysis and Support Vector Machine in Software Fault Prediction. *International Journal of Computer Trends and Technology (IJCTT), 7*(3), 131-136. https://doi.org/10.14445/22312803/IJCTT-V7P131

Shanthini, A., Vinodhini, G., & Chandrasekaran, R. M. (2013). Bagged SVM classifier for software fault prediction. *International Journal of Computer Applications, 62*(15), 21-24. https://doi.org/10.5120/10156-5030

Sharkov, G., & Stoeva, M. (2021). *Introducing Software Quality Maturity Models in Software Engineering Education and Small Organizations.* In Proc. of 14th conference on Information Systems and Grid Technologies–ISGT 2021, Sofia, Bulgaria.

Sharma, M., & Mona, P. (2016). Default prediction in software module by using Support Vector Machine (SVM). *IRACST- International Journal of Computer Science and Information Technology and Security (IJCSITS), ISSN: 2249-9555, 6*(2), 142-148.

Shippey, T., Tracy, H., Steve, C., & David, B. (2016). So You Need More Method Level Datasets for Your Software Defect Prediction? Voila!" In 10th Acm/Ieee International Symposium on Empirical Software Engineering and Measurement (Esem'16), 12:1-12:6. ESEM'16. New York, NY, USA: ACM. https://doi.org/10.1145/2961111.2962620

Singh, Y., Kaur, A., & Malhotra, R. (2009). Software fault proneness prediction using support vector machines. *Proceedings of the world congress on engineering, 1*(1), 1-3.

Soofi, A., & Awan, A. (2017). Classification Techniques in Machine Learning: Applications and Issues. *Journal of Basic and Applied Sciences, 2017*(13), 459-465. https://doi.org/10.6000/1927-5129.2017.13.76

Thangavel, M., & Pugazendi, R. (2017). Optimized Support Vector Machine for Software Defect Prediction.

International Journal of Engineering Research and Development, 13(12), 1-11

Xing, F., Guo, P., & Lyu, M. R. (2005). A novel method for early software quality prediction based on support vector machine. In *Software Reliability Engineering. ISSRE 2005. 16th IEEE International Symposium on* Software Reliability Engineering (ISSRE 2005) held at Chicago, USA. 213-224.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).