Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 10, October 2019, pp. 2578-2588
DOI: 10.23940/ijpe.19.10.p3.25782588

Proposed Intelligent Software System for Early Fault Detection

Manu Banga*, Abhay Bansal, and Archana Singh

ASET, Amity University, Noida, 201313, India

Abstract

The major challenge in designing an Intelligent Information Software System for fault detection is to detect faults at an early stage unless it becomes a failure. This can be achieved by using feature selection and effective classification applied on failure datasets. Support Vector Machines (SVM) are used for efficient and accurate feature selection by finding unknown model parameters using local and global kernel parameter optimization. Research shows that previously many attempts were made using classical classifiers as decision tree, naïve bayes, and k-NN for software fault prediction. In earlier research, class imbalance problems in software fault datasets were not addressed. In this paper, we propose an intelligent hybrid algorithm that is based on feature selection hybrid kernel function SVM and entropy-based bagging for efficient classification to reduce the class imbalance problem. The proposed model is compared with traditional approaches. The improved hybrid algorithm based on entropy-based bagging and mixed kernel SVM can effectively improve the classification accuracy of NASA Metric Data Program (MDP) faulty datasets. This paper presents an empirical study on using the proposed hybrid algorithm and results showed that our proposed approach enhances the classification accuracy when compared with existing methods.

Keywords: software fault classification; model parameter estimation; support vector machines; entropy; bagging

(Submitted on September 2, 2019; Revised on October 21, 2019; Accepted on October 30, 2019)

1. Introduction

Software has played a crucial role in manufacturing, security, finance, government, business and a lot more, but still fails at being trustworthy at all times. Detecting failures in software is critical as the development process affects the effective utilization of staff, cost and time. The software defect can cause failure thereby reducing required software quality. So, a software defect prediction system should predict defects automatically. Among protective measures, the first concern is fault diagnosis in the software datasets, as it is an important part of overall software reliability and maintenance system. When a fault occurs in software the effort and development cost increases.

However, many of these faults have intricately related connections as some faults remained unidentified and make it difficult to detect the type of fault generated due to imperfect debugging or error generation. Therefore, it is necessary to use the most suitable fault classification algorithm to classify the collected data accurately for the best fault diagnosis. Software metrics are used for software defect prediction such as the McCabe's and Halstead's metrics [1]. To compare results, we are using the NASA Metrics Data Programs datasets, which comprise 13 project data, with an average number of defective modules at 19 percent. The PC2 dataset has the lowest at 0.4 percent defected modules. This unequal distribution of defective and non-defective modules is referred to as the "class imbalance problem", which tremendously drops classification performance [2-3]. Wang et. Al (2010) and Gray et al.(2011) proposed several defect prediction models using classical available techniques in machine learning with MDP datasets. However, they ignored the class imbalance issue, making their results biased and erroneous [4-5]. Adopting a systematic approach, we propose a novel defect prediction system using feature selection for removing noisy and irrelevant attributes and entropy for the class imbalance problem. After an empirical study, we compared our approach with various classifiers in terms of precision, recall, F-Score, and accuracy.

* Corresponding author.
E-mail address: manubanga@gmail.com; abansal1@amity.edu

In this paper, we propose a novel algorithm for improving the accuracy of software reliability estimation using feature selection with SVM by reducing noise and selecting relevant attributes and thereby applying the entropy-based bagging algorithm on relevant attributes. We then compared our results with various existing classification approaches. Our approach enables the classification of defective modules with higher accuracy and addresses the class imbalance problem.

Section II presents a literature review of related work on defect prediction. Section III presents the research methodology used. Section IV presents the proposed approach of hybrid algorithms and in Section V, the experiment and results on the NASA Dataset are shown. Section VI discusses the conclusion of our research with its outcome.

2. Related Work Done

Rodriguez et al. (2014) proposed a key for defect removal and defect prediction. He made a list of metrics first and then used it for software defect prediction [4, 6]. Elish et al. (2008) used the Naïve Bayes classifier for predicting software defects and achieved 71 percent accuracy on fault detection [7-8]. Catal et al. (2009) used the Naïve Bayes classifier for defect detection from static code. They compared its result with the entropy approach and concluded that Naive Bayes is an effective classifier for static codes [9-10]. Challagulla et al. (2008) used the Principal Component Analysis (PCA) for reducing multi-collinear complexity and achieved a 10 percent higher accuracy in misclassification rate. These researchers used classifiers for software assurance on digital communication system designing [2, 11]. Jiang et al. (2008) devised an Artificial Immune System on the human immune system for defect prediction. They extended the bayesian approach with poisson distribution and proposed a novel software reliability model [12]. Shepperd et al. (2013) used Support Vector Machines for detecting errors in the NASA MDP data set and showed that modules with the highest SVM scores in the testing phase are more defect-prone [13]. Gray et al. (2010) used the association rule for the learning phase and used the correlation coefficient to compare results, achieving a higher accuracy by 23 percent in the prediction rate. In 2011, they surveyed the performance of the classification algorithm. They used NASA MDP datasets on twenty-two classifiers with accuracy outcome nearly the same [14-15]. Jiang et al. (2017) proposed a comprehensive approach in choosing classifiers for faulty datasets [3, 15]. Shuo et al. (2013) surveyed the resampling techniques, threshold and ensemble algorithms on Naïve Bayes, Random Forest, and AdaBoost, and they concluded that AdaBoost is an effective classifier in achieving the highest classification accuracy [15-16]. Similarly, Shukla et al. (2015) attempted to classify data using the Naïve Bayes classifier to predict defects at an early stage using the Bayes Theorem, and they achieved significant results in fault detection at inductive learning. Furthermore, in 2014 they proposed using evolutionary algorithms like genetic algorithms and modelled faults control using the mutation detection rate [17-18]. Nassif et al. (2019) used software metrics for predicting software module reuse and concluded with classifying reuse modules into four classes for fault prediction [5, 16, 19].

3. Research Methodology

We extract feature selection on NASA MDP datasets by Support Vector Machines (SVM) as it searches from local and global kernel by choosing only relevant features using the sigmoid kernel function and optimizes it, thereby decreasing the dimensionality and learning time of a classifier. It further enhances the classification accuracy, thereby obtaining an optimal feature subset giving optimal performance in fault prediction [20]. Feature Selection is mainly of three types: Embedded Feature Selection, Wrapper Feature Selection, and Filter Based Feature Selection. In Embedded Feature Selection, regularization of the dataset is done to avoid over-fitting of variance. In Filter Based Feature Selection, known characteristics of the data based on a metrics is used. In Wrapper Based Feature Selection, learning from the training and dataset is done and is verified on test data to get optimal parameters. We chose the Wrapper Based Feature Selection approach as it has the best accuracy in feature selection.

3.1. Data Sets Used

We used the NASA Metric Data Program (MDP) dataset [21] from the Promise Software Engineering Repository. It has data from 13 projects of NASA spacecraft software: CM1, JM1, KC1, KC3, KC4, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5. CM1 project has 505 samples of recorded value 20200 with 40 features; the percentage of defected modules is 10 percent. JM1 project has 10878 samples of recorded value 228438 with 21 features; the percentage of defected modules is 19 percent. KC1 project has 2107 samples of recorded value 44247 with 21 features; the percentage of defected modules is 15 percent. KC3 project has 458 samples of recorded value 18320 with 40 features; the percentage of defected modules is 9 percent. KC4 project has 125 samples of recorded value 5000 with 40 features; the percentage of defected modules is 49 percent. MC1 project has 9466 samples of recorded value 369174 with 39 features; the percentage of defected modules is 0.7 percent. MC1 project has 9466 samples of recorded value 369174 with 39 features; the percentage of defected modules is 0.7 percent. MC2 project has 161 samples of recorded value 6440 with 40 features; the percentage of defected modules is 32 percent. MW1 project has 403 samples of recorded value 16120 with 40 features; the percentage of defected modules is 8

percent. PC1 project has 1107 samples of recorded value 44280 with 40 features; the percentage of defected modules is 7 percent. PC2 project has 5589 samples of recorded value 223560 with 40 features; the percentage of defected modules is 04 percent. PC3 project has 1563 samples of recorded value 62520 with 40 features; the percentage of defected modules is 10 percent. PC4 project has 1468 samples of recorded value 58320 with 40 features; the percentage of defected module is 12 percent. PC5 project has 17186 samples of recorded value 670254 with 39 features; the percentage of defected module is 3 percent. From 13 projects, we found the defect percentage as 19 percent with maximum defect percentage in JM1 project and minimum defect percentage as 0.4 percent in PC2. The various project values are as depicted in Table 1.

Table 1. Details of NASA MDP datasets

Name	Language	Features	Instances	Recorded values	Percentage of defective modules
CM1	C	40	505	20200	10
JM1	C	21	10878	228438	19
KC1	C++	21	2107	44247	15
KC3	Java	40	458	18320	9
KC4	Perl	40	125	5000	49
MC1	C & C++	39	9466	369174	0.7
MC2	C	40	161	6440	32
MW1	C	40	403	16120	8
PC1		40	1107	44280	7
PC2	C	40	5589	223560	0.4
PC3		40	1563	62520	10
PC4		40	1468	58320	12
PC5	C++	39	17186	670254	3

3.2. Hybrid Kernel Support Vector Machines for Feature Selection

Support vector machine (SVM) are used for selecting relevant features from software faulty NASA MDP datasets. SVM maximizes the "margin" and thus relies on the concept of "distance" between different points. It is up to you to decide if "distance" is meaningful. As a consequence, one hot encoding for categorical features is a requirement. Furthermore, min-max or other scaling is highly recommended as a preprocessing step. Since the introduction of SVM in recent years, many new algorithms have been developed using it. The Least Square Twin Value SVM (LST-SVM) proposed to cope with the considerable noise in the dataset using kernel-based learning methods. Mean Value-Least Square Twin Support Vector Machines (MW-LSTSVM) is based on eigen values, which determine two nonparallel planes for solving the granular data problem and for improving classification accuracy.

3.2.1. Data Normalization

Data normalization is an indispensable part of training an SVM. In the sampled data, the difference in the range of numerical values is very large. For features with such a large range, we ignored the fractional (decimal) portion of the values as the large range already affects the classifier to a much greater extent than features with a smaller range of values. We map a given attribute to the range [0, 1] using the normalization:

$$x_m = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times (high - low) + low \tag{1}$$

Where x is the value before the feature value is processed, $x_{\min}$ is the minimum of all of the original features, and $x_{\max}$ is the maximum of all of the original features. *high* and *low* are the maximum and minimum values of the mapping interval, respectively.

3.2.2. Data Cleaning

The widely used SVM classification technique performs the task satisfactorily when no noise is present, but performs less well with noise in the dataset. Moreover, a given sample may differ significantly from normal data and have a greater similarity to abnormal data. Noise has characteristics that indicate that they are equivalent to discrete points. Therefore, admitting a noise sample into the final calculation can make a significant difference between the computed result and the actual value, leading to serious errors in the classifier. So, accurate classification requires us to pre-process the existing

training samples to remove the noise samples from the initial training sample set. This noise filtering greatly improves classification accuracy. To make the classifier more robust and less sensitive to noise performance, we propose a scheme to enhance these characteristics greatly. Prior to using the dataset to train the SVM, we remove the outliers using high-dimensional spatial de-noising to complete the de-noising process.

3.2.3. SVM Model Evaluation using Kernel Function Selection

A kernel function is selected based on the dataset used as each kernel functions and its parameters produce different effects even for the same sample data, thereby decreasing feature selection performances. Therefore, selecting appropriate kernel parameters to solve the applicable faulty dataset is a necessary condition. Commonly used kernel functions include Linear Kernel Functions, Polynomial Kernel Functions, Gauss Radial Basis Kernel Functions, and Sigmoid Kernel Functions. As the NASA MDP faulty dataset is nonlinear, we have chosen the Sigmoid Kernel Function as it has a single parameter σ and can handle the relationship between attribute and category well. It has a high performance when compared with functions in mapping the original nonlinear samples to the high-dimensional feature space of the faulty dataset [18].

3.2.4. Kernel Parameter Optimization

After selecting the kernel function, we must select appropriate kernel parameters. For the Sigmoid

Kernel Parameter σ, experimental data shows that, if the distance between σ and the sample point is very small, $\sigma \to 0$. Conversely, if the distance between σ and the sample point is large, $\sigma \to \infty$. When σ is very small, the discriminant function obtained by the Sigmoid Kernel Function SVM is almost a constant, which leads to over-fitting and a reduction in the classification accuracy rate. A large value of σ leads to reduced classification accuracy as well. Therefore, finding optimal parameter values is necessary for the best classification performance. The traditional separation interval method (SIM) aids in selecting the kernel parameter by taking the smallest distance from the same sample data to the centre point of its category.

We use two sample sets: $X_1 = \{(x_i, y_i) \mid y_i = 1\}$ and $X_2 = \{(x_i, y_i) \mid y_i = -1\}$. The data quantities are n_1 and n_2, and the respective center points of the sample sets are q_1 and q_2. From this, we obtain:

$$q_1 = \frac{1}{n_1}\sum_{i=1}^{n_1} x_i \ , \ q_2 = \frac{1}{n_2}\sum_{i=1}^{n_2} x_i \tag{2}$$

We set the kernel function to be $K(x_i, x_j) = <\phi(x_i), \phi(x_j)>$ with kernel parameter σ. After the kernel function maps the selected samples from the lower-dimensional to the higher-dimensional space, the distance between the centre points q_1 and q_2 is:

$$Q = \|q_1 - q_2\| = \sqrt{\left\| \frac{1}{n_1}\sum_{i=1}^{n_1}\phi(x_i) - \frac{1}{n_2}\sum_{i=1}^{n_2}\phi(x_i) \right\|^2} \tag{3}$$

Then, we express the optimization kernel parameter as:

$$\max_{\sigma}\left(Q(\sigma)\right) = \max_{\sigma}\sqrt{\left\| \frac{1}{n_1}\sum_{i=1}^{n_1}\phi(x_i) - \frac{1}{n_2}\sum_{i=1}^{n_2}\phi(x_i) \right\|^2} \tag{4}$$

This method only needs to solve for the maximum value in Formula (4) to obtain the value of the kernel parameter. In each type of dataset, a distinct feature is always present: the sample data belonging to the same category are always close to each other, and the distribution is relatively aggregated. SIM first solves their centre points q_i according to the sample data of each category and then solves the sum of the distances of different types of sample data to the centre points of other categories. As a simple example, consider the use of two categories. For the low-dimensional space, there are two different categories of nonlinear sample sets:

$$X_1 = \{(x_i, y_i) \mid y_i = 1\}, \ i = 1, 2, \cdots, n_1 \tag{5}$$

$$X_2 = \{(x_i, y_i) \mid y_i = -1\},\ i = 1, 2, \cdots, n_2 \tag{6}$$

In the above formula, n_1 and n_2 are used to indicate the number of the samples that two categories of data sets contain respectively, and y_i represents the category of the sample data. If two data belong to the same category, then their *y* values are equal. Conversely, if they do not belong to the same category, then their *y* values are not equal when calculating the centre points of two different categories of data based on the sample set:

$$Q_1 = \frac{1}{n_1}\sum_{i=1}^{n_1} x_i\ ,\ Q_2 = \frac{1}{n_2}\sum_{i=1}^{n_2} x_i \tag{7}$$

The average distance of the data in the category of X_1 to Q_2 is:

$$X_{12} = \frac{1}{n_1}\sum_{i=1}^{n_1}\|x_i - Q_2\| = \frac{1}{n_1}\sum_{i=1}^{n_1}\sqrt{\left\|x_i - \frac{1}{n_2}\sum_{i=1}^{n_2} x_i\right\|^2} \tag{8}$$

Similarly, the calculation of the average distance from the data in the X_2 category Q_1 is:

$$X_{21} = \frac{1}{n_2}\sum_{i=1}^{n_2}\|x_i - Q_1\| = \frac{1}{n_2}\sum_{i=1}^{n_2}\sqrt{\left\|x_i - \frac{1}{n_1}\sum_{i=1}^{n_1} x_i\right\|^2} \tag{9}$$

Then, we use the ISIM selection method to determine the kernel parameter σ :

$$\max(\sigma) = \max(X_{12} + X_{21}) \tag{10}$$

For the Sigmoid Kernel Function used in this paper, $K(x_i, x_j) = \exp(\frac{-\|x_i - x_j\|^2}{2\sigma^2})$, the above sample set is mapped to a higher-dimensional space. The mapping is represented by ϕ . Thus, after the mapping, we can convert Formulas (6)-(8) into the following expressions:

$$Q_1 = \frac{1}{n_1}\sum_{i=1}^{n_1}\phi(x_i)\ ,\ Q_2 = \frac{1}{n_2}\sum_{i=1}^{n_2}\phi(x_i) \tag{11}$$

$$X_{12} = \frac{1}{n_1}\sum_{i=1}^{n_1}\|\phi(x_i) - Q_2\| \tag{12}$$

$$X_{21} = \frac{1}{n_2}\sum_{i=1}^{n_2}\|\phi(x_i) - Q_1\| = \frac{1}{n_2}\sum_{i=1}^{n_2}\sqrt{1 + \frac{1}{n_1^2}\sum_{j=1}^{n_1}\sum_{k=1}^{n_1}\exp\left(\frac{-\|x_j - x_k\|^2}{2\sigma^2}\right) - \frac{1}{n_1}\sum_{j=1}^{n_1}\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)} \tag{13}$$

After mapping to a higher-dimensional space, the full expansion of Formula (9) is:

$$\begin{aligned}\max(\sigma) = \max\Bigg(&\frac{1}{n_1}\sum_{i=1}^{n_1}\sqrt{1 + \frac{1}{n_2^2}\sum_{j=1}^{n_2}\sum_{k=1}^{n_2}\exp\left(\frac{-\|x_j - x_k\|^2}{2\sigma^2}\right) - \frac{1}{n_2}\sum_{j=1}^{n_2}\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)} \\ &+ \frac{1}{n_2}\sum_{i=1}^{n_2}\sqrt{1 + \frac{1}{n_1^2}\sum_{j=1}^{n_1}\sum_{k=1}^{n_1}\exp\left(\frac{-\|x_j - x_k\|^2}{2\sigma^2}\right) - \frac{1}{n_1}\sum_{j=1}^{n_1}\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)}\Bigg)\end{aligned} \tag{14}$$

After selecting the kernel function, we must select appropriate kernel parameters. For the Sigmoid Kernel Parameter σ, experimental data shows that:

- Case 1: If the distance between σ and the sample point is very small$\sigma \to 0$. When σ is very small, the discriminant function obtained by the Sigmoid Kernel Function SVM is almost a constant, which leads to over-fitting and a reduction in the classification accuracy rate.
- Case 2: If the distance between σ and the sample point is large$\sigma \to \infty$. A large value of σ leads to reduced classification accuracy as well.

The preceding describes our optimization method for the kernel parameters. The following steps in Figure 1 give a specific process for kernel parameter optimization [22].

Step 1: First, obtain the sample dataset and incorporate the sample data of each category into equation 14 to obtain their actual expressions.
Step 2: Select a range of values (f_1, f_2), where f_1, is the minimum value of the interval and f_2, is the maximum in the interval, for the kernel parameter σ
Step 3: Solve for the values $f_3 = {f_1 + f_2}/{2}\ \mathrm{f_3} = (\mathrm{f_1} + \mathrm{f_2})/2$ and obtain $\max(f_1)$ and $\max(f_2)$
Step 4: If $\max(f_1) > \max(f_2)$ set $f_3 = f_2\ \mathrm{f_2} = \mathrm{f_3}$. If $\max(f_1) < \max(f_2)$, set.$f_3 = f_1$
Step 5: If $|\max(f_1) - \max(f_2)| \le e\ |\max(\mathrm{f_1})\text{-}\max(\mathrm{f_2})| \le \mathrm{e}$, the optimal value is $\left({f_1 + f_2}/{2}\right)\ (\mathrm{f_1} + \mathrm{f_2})/2$, and the entire optimization process ends. Otherwise, repeat Step 3

Figure 1. Algorithm for kernel parameter optimization

3.2.5. Penalty Parameter

The penalty parameter *C* is another important factor affecting the performance of SVM algorithms by balancing error and risk. This parameter adjusts the ratio of the confidence range to the empirical risk of the SVM model, improving the SVM's generalization ability.

Case 1: If the value of *C* is too small, there is a smaller empirical error, and the obtained error becomes greater, increasing the empirical risk value of the SVM, resulting in an "under-learning" condition.

Case 2: If C is too large, the accuracy of the model improves at the expense of its generalization ability, and the "over-learning" condition occurs. So, reasonable value of the penalty parameter is a must for model to be in a stable state. Therefore, we need to choose the penalty parameter efficiently.

3.3. Information Entropy Calculation using Renyi Entropy

Entropy (Shannon, 1948) is most commonly used in thermodynamics in order to measure system disorder and randomness. This approach came out to be very successful when considering multi-attribute problems. According to the approach, entropy is inversely proportional to the contribution in the comprehensive evaluation. It also maintains a balance between the criterions chosen and determines the weight obtained from the chosen values. It has been observed that the criteria weights have large values when the considerations consist of larger differences in it, thus, reducing the entropy. The information entropy calculated using Renyi Entropy [23] is:

$$H_\alpha(X) = \frac{1}{1-\alpha} \log \sum p_x^\alpha(x) \tag{15}$$

Where α is the degree of randomness and p_x is the probability of event occurrence.

3.4. Bagging

The application of bagging is used to improve the results of classification techniques used. The technique Bootstrap results in z fitted models. The models are further brought together to apply regression or generate a random selection of models. The bagging classifier segregates *z* datasets into z' training sets. The model is now built with the help of z' dataset, and voting for each model results in the model reducing variance and evading over-fitting of the model [24]. For improving classification, the Bagging technique is used as shown in Figure 2.

Input: Training Original Dataset M of size z
Apply Bootstrap Sampling on S generating modified dataset as M' of size z'
Case 1: if($z' < z$), by a uniform sampling of S with replacement, probability of repeating some samples in M'
Case 2: if ($z' = z$) then for M' is expected to have $\left(1 - \frac{1}{e}\right)$ unique samples, rest repeating samples.

Figure 2. Implementing bagging algorithm

4. Proposed Approach-Software Defect Prediction Framework

In the proposed approach, the hybrid of algorithms is used to improve the accuracy of software reliability estimation using feature selection by Hybrid Kernel SVM. Reducing noise and selecting relevant attributes are obtained after *optimized kernel parameter* and we use *information entropy-based Bagging* to improve classification.

Figure 3 and Figure 4 depicts Extracting Feature Selection using the Hybrid Proposed Algorithm

Step 1: Let $= sample\_count$;
/*Load the dataset with attribute*/
Step 2: Selected Feature Subset $= \{\phi\}$
For $p = 1$ to $desired\_feature\_count$;
For $c = 1$ to $feature\_count$;
$Data\_all$ =data(Selected_Feature + feature(c));
For $= 1$ to $class\_number$;
$Train\_data\_class(i) = partition(rand(Data\_all(class == i)),0.10)$;
/* Divide dataset into a training set and testing data set based on tenfold cross-validation, with $i \leq m$*/
$Test\_data\_class(i) = partition(rand(Data\_all(class == i)), others)$;
Step 3: Generate new dataset with a number of N features $N = \{1, \ldots\ldots, m\}$ and obtain $desired\_feature\ class\ C = \{c_1 \cup c_{2\ldots\ldots} \cup c_m\}$ and
if $N > m$, proceed and compare results with other features subset,
else $i = i + 1$ and repeat loop from Step 2.
Step 4: Calculate Information Entropy using Renyi Entropy
Step 5: Implement Bagging and select a subset of the highest accuracy
Step 6: Obtain optimal parameters values for classification of faulty and non-faulty modules

Figure 3. Proposed algorithm for software defect prediction

5. Experiment and Results

To maximize classification accuracy, we solve the fitness function in the algorithm. We looked for values of the kernel parameter σ in the range $[f_1, f_2]$ to find the optimal parameter value and conducted an empirical study using a Hybrid Kernel SVM Algorithm with parameter values of Kernel Parameter σ in the range [0, 100] and Penalty parameter C = 100 (constant).

5.1. Model Validation

To validate our approach, we divide the dataset into k-fold cross-validation for the learning and testing data. The value of k depends on the group in which the dataset has to be split for checking accuracy of a model. The first fold is treated as the validation set and is used to fit the remaining k-1 folds. We use the 10-fold cross-validation as it generally results in less biased. An estimate of the model [19] is depicted in Figure 5.

5.1.1. Performance Measure

The classifier performance is depicted in two-dimensions by the ROC curve. To compare the classifiers, an ROC performance must be represented in a single scalar vector of expected performance [4]. The commonly used technique is to calculate the area under the ROC curve. The area under the curve of a classifier is equivalent to the probability that:

An instance is positive and classified as positive – True Positive (TP)
An instance is positive and classified as negative – False Negative (FN)
An instance is negative and classified as negative – True Negative (TN)
An instance is negative and classified as positive – False Positive (FP)

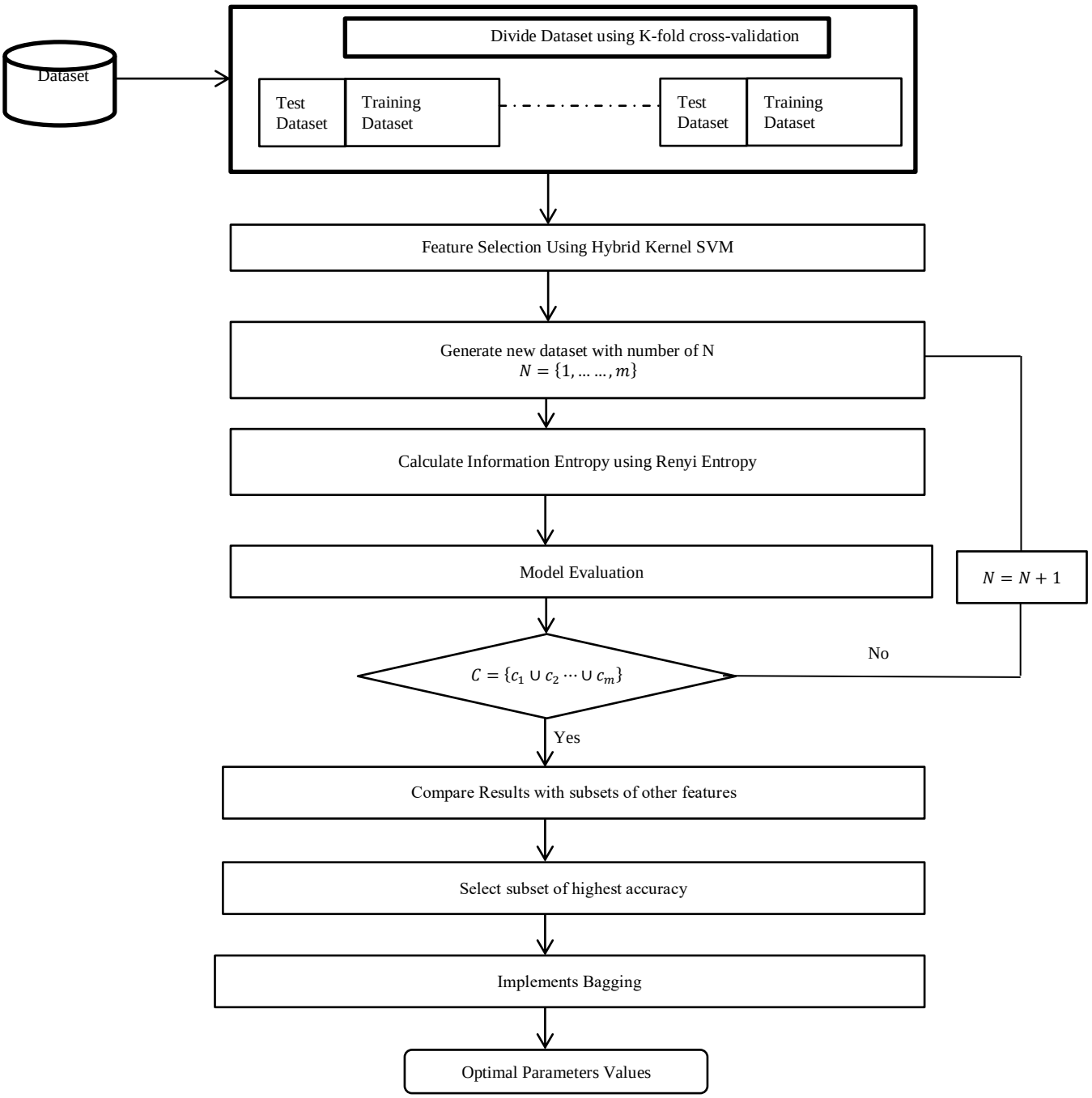

Figure 4. Flow chart proposed approach

5.1.2. Model Evaluation

The performance of classifiers used on our model is classified as: 90% and above is Excellent, 80% to 90% is Good, 70% to 80% is Fair, 60% to 70% is Not Acceptable, below 50% is Failure.

5.1.3. Model Comparison using t-Test

We used *t*-tests to compare two samples of statistical differences between paired values of two samples and obtained a unique value after considering the variation of values within each sample. Significance level, α, signifies the probability of rejecting the null hypothesis if it is true. We chose a significance level $\propto = 0.05$.

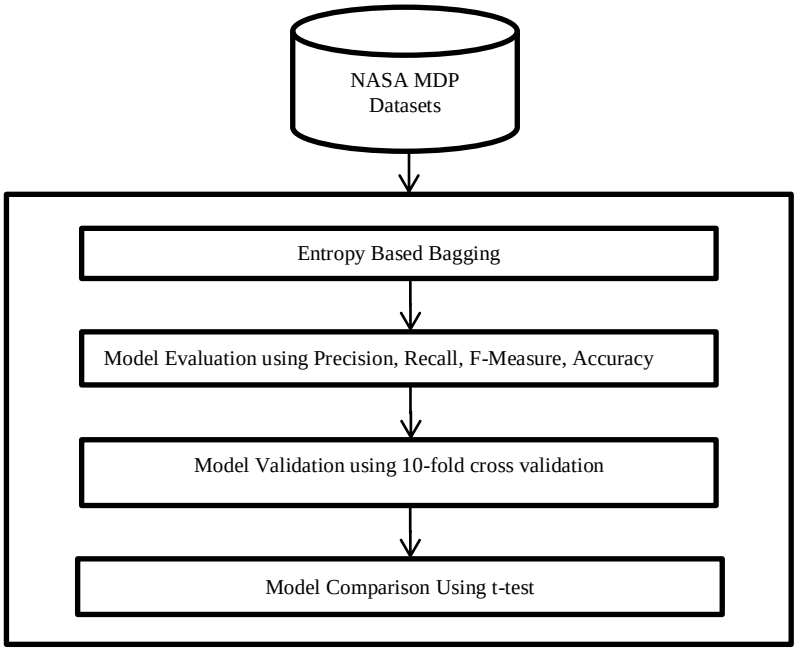

Figure 5. Model validation

5.2. Results and Discussion

In the experiments, the classifiers, Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM), used on the NASA metrics data program (MDP) dataset were:

- Without Feature Selection as shown in Table 2.
- Without Feature Selection as shown in Table 3.
- With Feature Selection and Entropy as shown in Table 4.

Table 2. Accuracy of classification on NASA MDP datasets (without feature selection)

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
SVM	0.753	0.752	0.642	0.761	0.714	0.79	0.534	0.75	0.79
LSTSVM	0.734	0.786	0.67	0.739	0.732	0.781	0.811	0.756	0.838
MS-LSTSVM	0.713	0.791	0.647	0.71	0.625	0.784	0.918	0.79	0.833

Table 3. Accuracy of classifiers on NASA MDP datasets (feature selection extracted using hybrid kernel SVM)

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
SVM	0.753	0.752	0.642	0.761	0.659	0.774	0.139	0.47	0.89
LSTSVM	0.734	0.786	0.67	0.739	0.724	0.799	0.811	0.75	0.861
MS-LSTSVM	0.713	0.795	0.647	0.761	0.742	0.852	0.822	0.81	0.903

Table 4. Accuracy of classifiers on NASA MDP datasets (using feature selection and entropy)

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
SVM	0.721	0.725	0.642	0.761	0.659	0.774	0.139	0.47	0.895
LSTSVM	0.756	0.786	0.867	0.739	0.724	0.799	0.811	0.75	0.861
MS-LSTSVM	0.617	0.95	0.547	0.761	0.742	0.852	0.822	0.81	0.903

To observe a statistical difference in the performance of classifiers Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM) we conducted *t*-tests as:

- Without feature selection and entropy as shown in Table 5.
- With feature selection as shown in Table 6.

- With feature selection and entropy as shown in Table 7.

Table 5. *t*-Test of classifiers without feature selection or entropy

Classifiers	Value of *t*-test	Result
SVM	0.005	Significant
LSTSVM	0.008	Significant
MS-LSTSVM	0.004	Significant

Table 6. *t*-Test of feature selection

Classifiers	Value of *t*-test	Result
SVM	0.008	Significant
LSTSVM	0.005	Significant
MS-LSTSVM	0.005	Significant

Table 7. *t*-Test of feature selection with entropy

Classifiers	Value of *t*-test	Result
SVM	0.007	Significant
LSTSVM	0.004	Significant
MS-LSTSVM	0.005	Significant

The various classifiers Support Vector Machine (SVM), Least Squares Twin Support Vector Machines (LSTSVM), and Mean Weighted Least Squares Twin Support Vector Machine (MW-LSTSVM) with feature selection and the entropy model are validated by comparing parameters including precision, recall, F-Measure, and accuracy. Results are shown in Table 8.

Table 8. Performance measures of different classifiers

Parameters	SVM	LSTSVM	MW-LSTSVM
Precision	81.56	82.84	86.61
Recall	81.46	82.46	88.64
F-Measure	81.89	82.93	87.83
Accuracy	84.56	82.46	89.85

After obtaining the above results, a comparison between various SVM Models was done and is shown in Figure 6.

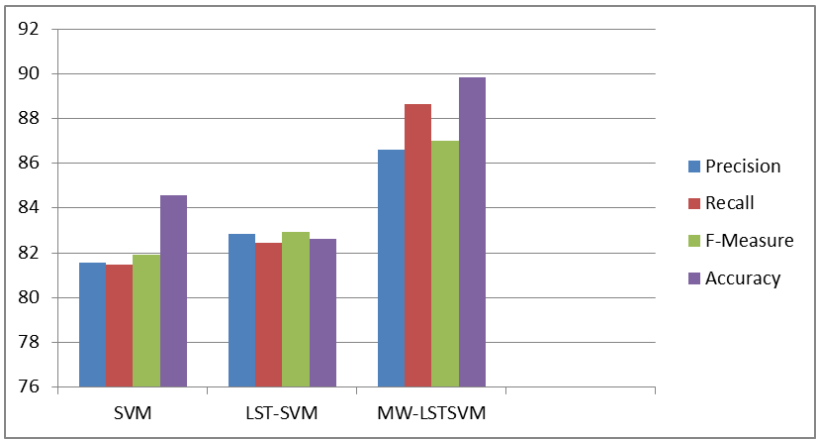

Figure 6. Comparative analysis of SVM models

6. Conclusions and Future Scope

In this paper, an attempt was made to find software defects using the hybrid approach. Identifying defects is the most crucial step as defect-prone software could lead to hazard and loss of reliability on the software. The avoidance of the class imbalance problem led to a decrease in the accuracy of the classifier. In this paper, we have used a hybrid SVM for feature selection and entropy-based bagging techniques to increase classification accuracy of defective or non-defective modules on different classifiers SVM, LSTSVM, and MW-LSTSVM. The experimental results show that the MW-LSTSVM classification algorithm achieved highest accuracy among all other classifiers with 91.3% accuracy, thus increasing the efficiency by 9.8% than the existing approaches. In our future work, this accuracy can be extended by using deep learning

techniques to have better feature engineering out of the datasets, leading to higher accuracy, thereby improving the software reliability of the system.

References

1. C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technology Journal*, Vol. 27, No. 1928, pp. 379-423, 1948
2. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Software Defect Prediction using Static Code Metrics Underestimates Defect-Proneness," in *Proceedings of the 2010 International Joint Conference on Neural Networks* (*IJCNN*), IEEE, 2010
3. H. W. Zhang, J. W. Xie, J. Ge, W. L. Lu, and B. F. Zong, "An Entropy-based PSO for DAR Task Scheduling Problem," *Applied Soft Computing*, Vol. 73, pp. 862-873, 2018
4. Q. Yu, S. J. Jiang, R. C. Wang, and H. Y. Wang, "A Feature Selection Approach based on a Similarity Measure for Software Defect Prediction," *Frontiers of Information Technology and Electronic Engineering*, Vol. 18, No. 11, pp. 1744-1753, 2017
5. Y. Y. Zheng, Y. Zhou, and J. H. Qu, "An Improved PSO Clustering Algorithm with Entropy-based Fuzzy Clustering," *WSEAS Transactions on Computing*, Vol. 14, pp. 88-96, 2018
6. T. M. Khoshgoftaar, N. Seliya, and Y. Liu, "Genetic Programming-based Decision Trees for Software Quality Classification," in *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence*, pp. 374-383, 2003
7. T. Wang and W. H. Li, "Naive Bayes Software Defect Prediction Model," in *Proceedings of 2010 International Conference on Computational Intelligence and Software Engineering*, pp. 1-4, 2010
8. T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on Software Engineering*, Vol. 33, No. 1, pp. 2-13, 2016
9. 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 *Proceedings of the 15th Annual Conference on Evaluation and Assessment in Software Engineering* (*EASE 2011*), pp. 96-103, 2011
10. S. C. Jiang, et al., "Modified Genetic Algorithm-based Feature Selection Combined with Pre-Trained Deep Neural Network for Demand Forecasting in Outpatient Department," *Expert Systems with Applications*, Vol. 82, pp. 216-230, 2017
11. V. U. B. Challagulla, F. B. Bastani, I. L. Yen, and R. A. Paul, "Empirical Assessment of Machine Learning based Software Defect Prediction Techniques," *International Journal on Artificial Intelligence Tools*, Vol. 17, No. 2, pp. 389-400, 2008
12. D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014
13. 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, 2018
14. Y. Jiang, B. Cuki, T. Menzies, and N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, pp. 11-18, May 2018
15. C. Catal and B. Diri, "Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem," *Journal of Information Sciences*, Vol. 179, No. 8, pp. 1040-1058, 2009
16. 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
17. B. Turhan and A. Bener, "Analysis of Naive Bayes' Assumptions on Software Fault Data: An Empirical Study," *IEEE Transactions on Data and Knowledge Engineering*, Vol. 68, No. 2, pp. 278-290, 2009
18. R. Malhotra, "A Systematic Review of Machine Learning Techniques for Software Fault Prediction," *Applied Soft Computing*, Vol. 27, pp. 504-518, 2015
19. G. Chandrashekar and F. Sahin, "A Survey on Feature Selection Methods," *Computers and Electrical Engineering*, Vol. 40, No. 1, pp. 16-28, 2014
20. H. S. Shukla and D. K. Verma, "A Review on Software Defect Prediction," *International Journal of Advanced Research in Computer Engineering and Technology* (*IJARCET*), Vol. 4, No. 12, pp. 4387-4394, 2015
21. M. Hamill and K. Goseva-Popstojanova, "Exploring Fault Types, Detection Activities, and Failure Severity in an Evolving Safety-Critical Software System," *Software Quality Journal*, Vol. 23, No. 2, pp. 229-265, 2015
22. S. Wang and X. Yao, "Using Class Imbalance Learning for Software Defect Prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, pp. 434-443, 2013
23. A. B. Nassif, M. Azzeh, A. Idri, and A. Abran, "Software Development Effort Estimation using Regression Fuzzy Models," Computational Intelligence and Neuroscience, 2019
24. P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-Validation," *Encyclopedia of Database Systems*, pp. 532-538, 2009