Chapter 1
Software Defect Prediction Through a Hybrid Approach Comprising of a Statistical Tool and a Machine Learning Model

Ashis Kumar Chakraborty and Barin Karmakar

Abstract Traditional statistical learning algorithms perform poorly in case of learning from an imbalanced dataset. Software defect prediction (SDP) is a useful way to identify defects in the primary phases of the software development life cycle. This SDP methodology will help to remove software defects and induce to build a cost-effective and good quality of software products. Several statistical and machine learning models have been employed to predict defects in software modules. But the imbalanced nature of this type of datasets is one of the key characteristics, which needs to be exploited, for the successful development of a defect prediction model. Imbalanced software datasets contain non-uniform class distributions with most of the instances belonging to a specific class compared to that of the other class. We propose a novel hybrid model based on Hellinger distance-based decision tree (HDDT) and artificial neural network (ANN), which we call as hybrid HDDT-ANN model, for analysis of software defect prediction (SDP) data. This is a newly developed model which is found to be quite effective in predicting software bugs. A comparative study of several supervised machine learning models with our proposed model using different performance measures is also produced. Hybrid HDDT-ANN also takes care of the strength of a skew-insensitive distance measure, known as Hellinger distance, in handling class imbalance problems. A detailed experiment was performed over ten NASA SDP datasets to prove the superiority of the proposed method.

Keywords Software defect prediction · Class imbalance · Hellinger distance · Artificial neural network · Hybrid model

Introduction

Software defect prediction (SDP) is an important topic in software reliability engineering literature. It helps software engineers to allocate the available resources to detect the defect-prone modules, if one can predict the number of defects in a module

A. K. Chakraborty · B. Karmakar (✉)
SQC and OR Unit, ISI Kolkata, Kolkata, India
e-mail: barinsaheb1996@gmail.com

A. Gunasekaran et al. (eds.), *Applications of Operational Research in Business and Industries*, Lecture Notes in Operations Research,
https://doi.org/10.1007/978-981-19-8012-1_1

in the early stage of the software development process. Several researchers tried to estimate the number of defects present in a software system [20]. Different statistical techniques like capture-recapture sampling have been applied to estimate the number of defects. Some authors also have used the number of defects detected during code inspection as a co-variate and used regression type of analysis to find out the estimate of reliability of a software [20]. A variant of this problem has been considered to estimate the optimum time for software release [10, 14]. An important feature of SDP problem is that the real-life data are highly imbalanced in nature between defect and non-defect classes. This basically means that the incidence of one of the classes, say non-defect classes are much more compared to the occurrence of defect classes. Some studies [12] noted this important feature for these types of datasets and suggested a hybrid approach which uses statistical and machine learning methods together to improve prediction accuracy. The authors [12] noted an important property of sample defect prediction datasets, that is in majority cases the modules that are non-defective (majority) are much more higher in number in comparison with the defective modules (minority). Noticing this class imbalance problem, several researchers [29] started using imbalance learning techniques to improve the performance for defect prediction. Earlier several SDP models have been developed with a goal to predict the possibility of occurrence of a defect in the unseen (future) version of a software product [7, 22, 35]. A software module with defects often fails and prevents to produce preferable results; so, it is required to have an early warning (detection) system of software defects in the system. In the starting phases of software development life cycles, there is possibility to correct the set of identified defects [38] in order to achieve better performance of the system. Furthermore, if it is possible to restrict such defects to propagate to the latter stages, then this could be cost-saving and worthwhile for the producer. Thus, constructing an efficient SDP model will be helpful in order to develop better quality software products which will reduce maintenance costs and accounts for higher customer satisfaction.

In this article, we use a novel hybrid approach where decision tree, which is based on Hellinger distance, and artificial neural network with one hidden layer is used to predict the number of software bugs. To validate the power of the hybrid HDDT-ANN method, we performed this method on several standard imbalanced SDP datasets. This new approach helped to increase minority class prediction accuracy and appears to be more efficient than other traditional supervised classification models for predicting defects and improving the overall performance.

This article is constructed as follows. In Section "Literature Review", we provide the literature review, while in Section "Proposed Model" we develop the proposed hybrid model. Section "Experimental Analysis" gives experimental analysis of the proposed model, wherein we apply the model on several SDP datasets. We conclude this article in Section "Conclusion". Section "Code" refers to the computer code used for implementation of the model developed.

Literature Review

Earlier, several authors have developed SDP models which were applied for predicting the occurrence of a defect that may be present in unseen (future) version of a software product [22, 29]. So a valid SDP model is required to develop good quality software products which helps in reducing maintenance costs and also increases customer satisfaction. Recent studies demonstrated that around 80% of the defects occurred in a few modules which consists of 20% [51] of total number of modules. This fact suggests that the defective class can be considered as a minority class and non-defective class is the majority class. This kind of imbalanced class frequency distribution is an important feature of SDP datasets. It is noted further that the more penalty is associated when minority (defective) classes are misclassified though they are less in number. Hence, for defect prediction in SDP area, it is essential to address the issue of misclassification of the elements of minority class for the unseen data to reduce future defect percentages and successful development of a software. In the binary pattern classification problem, when the source dataset (labeled data) contains one class of data, which are very high in number, compared to the other class, the concept of imbalance in the dataset originates. By convention, the majority class is the over-represented class whereas minority class is the other class having a lower number of instances. In software defect prediction, the probability of finding defect instances, which is from the minority class, are very less compared to finding a non-defect case which is from the majority class.

The objective of imbalance learning is to develop classification model that can correctly classify the instances from minority and majority class simultaneously. This kind of problem is tackled through different approaches, but almost all the traditional classifiers used to put higher weightage to the instances from majority class and as a result the instances from minority classes are often misclassified compared to the majority class instances [25]. In order to tackle this data imbalance, problem several methods have been developed utilizing the properties of data and also at algorithm levels. Some methods based on utilizing the properties of the dataset use various sampling techniques, where by manipulating training data we try to balance the skewed class distributions. The methods based on algorithm have modified the training procedure and used cost-sensitive and ensemble learning techniques in order to achieve increased accuracy on the minority class instances. The most common technique which has been used for solving class imbalance problems, is the well-known sampling techniques where we oversample or undersample the original dataset to modify the data class distributions. Random oversampling (ROS) or undersampling is one of the common techniques used for imbalanced datasets. In random oversampling, one randomly selects examples from the minority class, with replacement, and add them to the training dataset; on the other hand, random undersampling technique randomly selects instances from the majority class and deletes them from the training set. Another way to balance the class distribution is to synthesize new examples from the minority class. SMOTE (Synthetic Minority Oversampling Technique) is the most widely used approach to synthesizing new examples. SMOTE selects instances that

are adjacent within the feature space. After that it draws a line between the instances in the feature space and then it selects a new sample at a position along that line. Shatnawi [43] showed the efficacy of random oversampling (ROS) where training instances are added in the defective class of the datasets, and it is more effective compared to SMOTE technique. Lopez et al. [36] have presented that oversampling may effect in overfitting since there is possibility that the input training dataset may contain duplicate instances. These approaches have shortcomings of often overfitting or underfitting. To get rid of these problems, hybrid sampling, the technique where the approach is not only balancing the class distribution but also removing noisy instances which are lying on the wrong side of the decision boundaries, is used. Some examples are the combined techniques of oversampling and undersampling like synthetic minority oversampling technique (SMOTE) + Tomek link (TL) and ROS + TL [2, 23]. Oversampling like SMOTE alone is not enough to counter the problems with imbalanced structure, like overfitting. So, undersampling like Tomek link (TL) is further applied in the preprocessing stage to get rid of the problems like class clusters are overlapped between some majority and minority class space. Only majority class examples that participate in a Tomek link are removed on the basis of the following principles: If two instances create a Tomek link (TL), then at least one of the instances is noise or both are considered as borderline, since minority class examples are considered too rare to be discarded. These hybrid methods have been implemented on the SDP datasets, and it is found more effective than the individual methods. But ensemble methods [25] work better compared to these methods.

Some authors used ensemble learning techniques where they use combination of finite number of classifiers and allocate different values of weights to the component methods to tackle the imbalance classification problems. It is seen that ensemble learning utilizes the strength of individual learners and improves overall performance of the model by combining them. Wang and Yao [48] introduced a dynamic version of the AdaBoost.NC (ABNC) which is used in SDP for tackling the imbalanced classification problem. On the other hand AdaBoost-based kernel ensemble learning method was also experimentally presented to be quite effective for the projects which are made on NASA SDP datasets [29]. One can also think to formulate the SDP problems as a binary classification problem where software modules are classified as either defect-prone or non-defect-prone based on a set of features whose components are software metrics.

In software reliability engineering, a set of static code attributes like McCabe and Halstead are used to describe the complexity of a module, which is defined as the smallest unit of functionality and is extracted from previous software releases with the log files of defects. As the complexity of a module increases, the more likely it is to be fault-prone in nature. The values of the metrics which are mentioned above have been used as features to construct classifiers for defective module prediction for the upcoming phase of release. This leads to detect the segments of the software that have greater probability of containing defects. PROMISE repository [3] is an open-source of defect prediction datasets from real-world projects and several researchers used in their academic study. Several statistical and machine learning tools have been implemented to solve SDP problems, such as decision tree

[6, 30], Random Forest (RF) [27, 45], Deep Feed Forward Neural Network [28], support vector machines [26], and artificial immune systems [8]. Some authors also considered ensemble learning [46], kernel-based technique [26], learning based on cost-sensitive [49], improved subclass discriminant analysis (ISDA) technique [29], value-cognitive boosting along with support vector machine (VCB-SVM) method [40] transfer learning [47] in order to build software defect prediction models. It seems that no single method is good enough for developing models based on the SDP datasets. But all the methods like RF and cost-sensitive learning seem to be working better for the most of the datasets [48] compared to others.

However, many of the previous studies had not taken into account the class imbalance scenario of SDP datasets, though some researchers have implemented ensemble and other techniques for tackling the imbalanced structure of the dataset [25]. Some research demonstrated the effectiveness of resampling techniques in case of tree-based learners [48]. Ensemble and cost-sensitive learning approaches were also found quite effective in the cases where an appropriate cost ratio is fixed [33]. Recently, Gong et al. [25] proposed a new method known as stratification embedded nearest neighbor (STr-NN) approach. Also, Sun et al. [46] presented a ensemble learning approach which is coding-based. But these approaches have some deficiencies. For example, while using sampling techniques, the actual datasets are modified. Also, the choice of optimized weights in constructing ensembles is not straight forward. To get rid of these drawbacks, some authors introduced "imbalanced data-oriented" classifiers [17]. These classifiers, e.g., Hellinger distance-based decision tree (HDDT) and the ensemble technique like Hellinger distance-based random forest (HDRF) can tackle the problem of class imbalance without modifying the original dataset and hence it can be a possible solution to deal with SDP problems. Even though HDDT takes care of class imbalance, still it has some deficiencies. Since it is a greedy algorithm, i.e., at each stage, it finds the best feature for splitting and as a result it may stick to local minima and also may lead to overfitting [9] when the tree size is very large compared to the number of training data present. Also, HDDT does not use pruning techniques, where we add complexity parameter to obtain optimal subtree, so it may overfit the dataset since we are using the whole tree [4]. Some authors have tackled these deficiencies of HDDT problems using hybridization like Hellinger net [12], where the following steps are associated to generate the model: convert an HDDT into rules, construct two hidden layered DFFNN (Deep feed forward neural network) architecture from the generated rules, and finally train the DFFNN using stochastic gradient descent back propagation method.

Different hybrid models have been constructed combining the decision tree algorithm along with neural networks. Sirat and Nadal [44] introduced a method called Neural tree (NT), which is an example of this kind of model. Such techniques combine both CART which have the advantages of a hierarchical organization and ANN, because of the perceptron's ability to handle many input variables and they further proposed NT model which works for multiclass classification problems also. Sakar and Mammone [41] introduced a new method known as neural tree networks (NTN), where neural networks are associated in a tree structure. In neural tree network method, the neural networks are used recursively to segment the feature space

into disjoint sub-spaces at each tree node, which results in a preferable efficient classification performance. Foresti and Dolso [24] introduced a new concept known as adaptive neural tree (ANT). Later on, flexible neural tree (FNT) is introduced by Chen et al. [15], which works as a feature selection technique and also deal with problem of intrusion detection. Chen et al. [16] presented an experimental result where it is shown that the efficiency of FNT model to forecast the small time scale traffic measurement. Bouaziz et al. proposed a model called Flexible Beta Basis Function NT model [5], and it was shown that the model performs better compared to other related methods in case of dealing with some standard problems drawn from the control system and time series prediction. Sethi introduced [42] Entropy nets, a mapping of decision trees into a multilayer neural network structure, which has superiority since it contains comparatively fewer number of neural connections. In this framework, the number of neurons present in the neural network's input layer equals to the number of decision tree's internal nodes. These neurons go through hidden layer, and the number of neurons in the output layer is same as number of distinct classes. A recent study [12] has shown how hybrid model famously known as Hellinger net helps to improve accuracy of the model in the context of dealing with imbalanced dataset. In this article, we propose a hybrid HDDT-ANN model which will increase the performance of the model in terms of different accuracy metrics.

Proposed Model

In software defect prediction, static code attributes are extracted from older releases of software containing the log files of defects and these are used to construct models with an objective to predict the defective modules for the future releases. This is an advantage to discover parts of the software which are more probable to contain defects. We have presented our proposed hybrid HDDT-ANN model to tackle the class imbalance problem in context of SDP in this section. We first discuss the theoretical frameworks for constructing Hellinger distance-based decision tree (HDDT) and later on the work flow of our proposed model where HDDT and ANN two distinct classifiers are used jointly to make a decision.

Hellinger Distance Decision Tree

Hellinger distance, which is a symmetric and non-negative measure of distributional divergence, is associated to the Bhattacharyya's distance and the Kullback–Leibler divergence. Chawla [17] proposed how to use Hellinger distance as a decision tree splitting criterion for modeling an imbalanced dataset for classification. This method considers two distributions say U and V, which is the normalized frequencies of feature values considering the problem as a binary classification. Cieslak and Chawla [17] have introduced the concept of affinity between two distributions U and V as a

criterion for tree splitting. The goal is to split tree nodes on the features which are having minimal affinity which implies maximal Hellinger distance. This approach uses the idea of splitting of features based on how well the examples seen so far in the training data can be discriminated by them. Whereas other tree-based models used to split on the feature, which represents the highest possible number of data points seen so far (for an example: information gain discussed in Breiman's CART [6]). Hellinger distance is skew insensitive in nature, since examples of one particular class are higher in number compared to other class which only leads to make its sample distribution more inclined to the actual distribution. Considering the situation that a feature is a good class discriminator, then whatever the class balance is, it will remain same. The formal definition of Hellinger distance is as follows.
Definition: Let (δ, γ) denote a measurable space and assume that U and V be two continuous distributions with respect to the parameter γ having the densities u and v in a continuous space Ω, respectively [1]. One can define Hellinger distance (HD) as follows:

$$d_{\mathrm{H}}(U, V) = \sqrt{\int_{\Omega} (\sqrt{u} - \sqrt{v})^2 \mathrm{d}\gamma}$$

Note that Hellinger distance does not depend on the choice of the parameter γ. Some important properties of Hellinger distance are given below.

1. $d_{\mathrm{H}}(U, V)$ is in $[0, \sqrt{2}]$.
2. Hellinger distance (HD) is non-negative $d_{\mathrm{H}}(U, V) \geq 0, \forall\, U, V$.
3. HD is symmetric, i.e., $d_{\mathrm{H}}(U, V) = d_{\mathrm{H}}(V, U), \forall\, U, V$.
4. As the Hellinger distance increases, the discrimination power of the feature also gets better.

Hellinger Distance Decision Tree (HDDT) utilizes Hellinger distance for tree splitting and constructs the tree on the basis of methods proposed by Breiman's CART [6] as discussed in [17]. In this methodology, once the root node is found, the criterion is applied recursively until next decision node is found. This criterion determines in which input direction split will take place and where the cut should be initiated. HDDT works on feature input spaces in order to make a hierarchical axes-parallel split of it, similar like CART, the only exception is the split criterion. Here, each of the tree node represents the partitioned subsets in the input feature space. In this case, we consider only binary tree consisting of two child nodes or zero child nodes are constructed [17]. The details of the algorithm of splitting criteria of HDDT are provided below:

Consider $\underline{X}$ to be the input feature space and let p be the number of attributes and let $\underline{Y}$ be the set of binary class labels which consists of elements from the response column. Suppose, there is a training sample (labeled dataset) with k as the number of instances, $D_k = \{(X_j, Y_j);\ j = 1, 2, \ldots k\}$ where $X_j = (X_{j1}, X_{j2}, \ldots, X_{jp}) \in \underline{X}$, and Y_j represent their respective labels. The source SDP dataset D_k contains identically distributed independent instances (X_j, Y_j), which are distributed as the

pair denoted by $(\underline{X}, \underline{Y})$, where $X \in R_+^p$ and Y $\in \{0, 1\}$. One may note that for the source SDP dataset, the range of values of static code attributes of a software module are positive and numeric whereas response class value is discrete in nature (that is defective or non-defective). In our proposed hybrid approach [13], the second model is an artificial neural network model. Before modeling, we need to normalize the features of SDP dataset, i.e., we are mapping each element of each column to [0, 1] range and keeping the response column as it is, so that in latter stage, while we are training the neural network, the model becomes efficient. We are using min-max scaling for normalization. The problem is given with a total of n instances on p input features where $\underline{X} \in C^p$ and $\underline{Y} \in \{0, 1\}$. The goal is to construct a binary pattern classifier that predicts whether the software module is defect-prone or not, given the observed features of the software module. HDDT does hierarchical splitting of the input feature space. Each tree node in the feature space represents as one of the segregated subsets in C^p. Cieslak et al. [17] introduced a splitting criterion to create the HDDT is as follows:

$$\text{Hellinger distance (HD)} = d_{\text{H}}(X_+, X_-) = \sqrt{\sum_{r=1}^{p} \left(\frac{|X_{+r}|}{|X_+|} + \frac{|X_{-r}|}{|X_-|}\right)^2} \tag{1.1}$$

In this methodology, given there are two class distributions X_+ and X_-, the goal is to calculate the "distance" in terms of the normalized frequencies combined over all the partitions. Assuming feature space as a countable space, we are discretizing all continuous features into p partitions. $|X_+|$ denotes the frequency of instances which are from the majority class (non-defective class) in the training dataset and $|X_+r|$ is a subset of the training set for the feature X, comprising of majority class and with value r. A similar description follows for $|X_-|$ and $|X_-r|$ which are defined for the minority class (defect class). As Hellinger distance (HD) increases, the features become more discriminative in nature. HDDT methods also have a feature selection mechanism, and the first feature (first split node) selected has the property of carrying minimal affinity with respect to the classes. The Hellinger distance is not influenced by prior probability; hence, it is insensitive to class distribution. HDDT is also skew insensitive in nature since it is not taking account the prior probability while calculating distance. However, the split criterion mentioned in (1) only works in case of classification problem where only two distinct classes are present. But often HDDT may overfit the data. To get rid of this problem, we go for a hybrid method. Generally, we combine classifiers which use distinct pattern representations to make a decision. Pattern classifiers are designed in order to achieve the best possible classification performance for the unseen data or the task in hand [32]. Decision trees and artificial neural networks are both well-known and competitive methodologies for building classification problems. Classification trees are hierarchical classifiers in nature and it is comparatively superior to artificial neural network (ANN) in case of readability of knowledge [37]. ANN works better in case of implementation of comprehensive inference over the inputs [50].

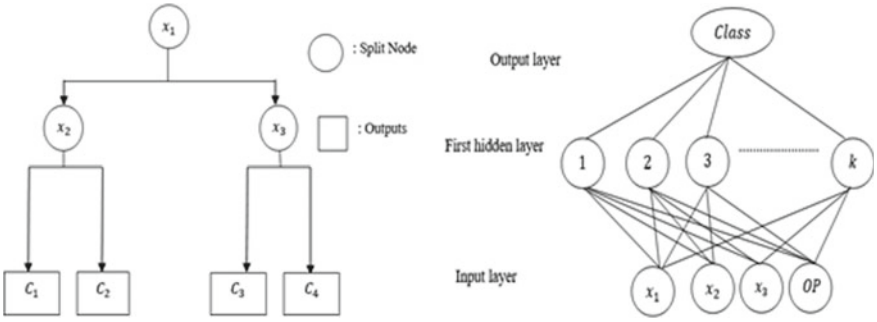

Fig. 1.1 Example of hybrid CT-ANN classification model where $x_i, i = 1, 2, 3$, represents important features extracted through CT and $c_i, i = 1, 2, 3$, are leaf nodes and OP as CT output [13]

Flowchart of the Proposed Model

In our proposed methodology, the feature space is split into disjoint segments by HDDT algorithm. The constructed tree chooses important features and removes redundant ones. Later, we build an ANN model with one hidden layer [39] where we have used the important features obtained through HDDT algorithm as input feature and additionally the prediction results produced by HDDT method is also used as input feature, i.e., if we obtain m number of important features then we provide $(m + 1)$ number of features in the input layer of neural networks. We first provide a picture in Fig. 1.1 of a simple hybrid model called CT-ANN Model [13]. This helps us to understand how a hybrid approach is used.

The informal work flow of the proposed hybrid model called HDDT-ANN model is given. Figure 1.2 represents the flowchart of the proposed HDDT-ANN model.

1. Normalize the source SDP dataset using min-max method.
2. Split the scaled dataset randomly by 80:20 ratio, where 80% data is used for training and 20% for testing.
3. Apply HDDT algorithm on training set to build a tree model which will extract important features.
4. HDDT shortlists the important features, which are contributing for building the tree and neglect the rest ones.
5. The prediction result produced by HDDT algorithm is considered as an additional feature along with the important features shortlisted, as described in step 5 are used in the input layer of Artificial neural network (ANN) model.
6. The important features extracted by HDDT method with an additional input feature are exported to the Artificial neural network (ANN) model with one hidden layer and with sigmoid activation function.
7. Then, we optimize the weights and number of neurons present in hidden layer of artificial neural network as mentioned in [13] and run ANN algorithm till acceptable accuracy is achieved.
8. Then, the classifier will be finally ready to produce results.

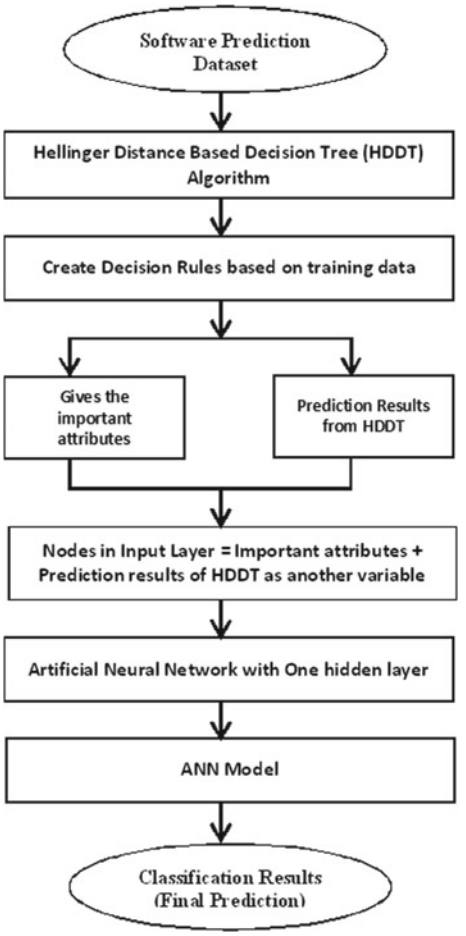

Fig. 1.2 Flowchart of HDDT-ANN model

Insights of the Proposed Model

The key point of our proposed HDDT-ANN model is the extraction of important features and utilization of the class levels produced through HDDT model, which is then followed by ANN model. Hornik and Stinchcombe [28] showed that if sufficient number of hidden units are present then standard multilayer feedforward networks consisting of one hidden layer is able to approximate any Borel measurable function from one finite dimensional space to another space to any desired level of accuracy. An additional input feature in the form of HDDT output with the extracted important features from HDDT method increases the dimension of feature space and will also increase the class separability, which will help ANN for drawing better decision boundary. Cover [18] proposed the idea that shows that if feature space is not densely populated, then in the higher dimensional space it becomes linearly sepa-

rable, compared to a lower dimensional space in a complex pattern classification problem. Experimental results by Lee and Srihari [34] have presented that as more information is included, then the performance of combination of decision tree and ANN algorithm also increases. One of the key characteristics of our proposed hybrid model is choosing the optimum number of neurons to be used in ANN model in the hidden layer. It is shown that too few nodes in hidden layer may restrict network generalization capabilities, on the other side, too many hidden layers may lead into the situation of overtraining by the network [39]. To deal with this problem, we used the approach of optimizing weights and also varying the number of neurons to be used in the hidden layer until we achieve sufficient accuracy. This algorithm is a two-step problem-solving approach where it initially selects features based on HDDT and then use optimum ANN technique to improve the model. The theoretical properties of our proposed model have been proved in the article [11].

By implementing this model, future defects based on code attributes can be predicted accurately compared to other models and appropriate actions can be taken. We will experimentally illustrate that the proposed model is superior in comparison with the other supervised models, which is discussed in the literature for 10 NASA datasets available at promise repository. Our proposed methodology can be utilized for choosing features of items that will satisfy a specific goal and also can be engaged for modeling such complex scenarios.

Experimental Analysis

Description of Datasets

The SDP datasets consist of various features (numeric values) of a software module (e.g., various measures of lines of code, base Halstead measures, derived Halstead measures, McCabe metrics, and branch counts are some features of the SDP datasets), along with the response column consisting of class labels (true or false i.e., whether the module reported defects or not) [31]. Our objective is to classify between defect ("true" class) and non-defect ("false" class) distributions of software modules. The dataset is splitted into training set (80% of the dataset) for building the model and test set (20% of the dataset) to examine the performance of our model. Such a split is quite usual in literature [15]. The response variable indicates whether the software module is defective or non-defective. Sample datasets and their characteristics are shown in Table 1.1. Each of the datasets contains two classes and is highly imbalanced in nature. As an example, analysis of CM1 dataset is shown here. The number of attributes of CM1 dataset contains 22 features (lines of code measure of 5 different types, 3 different McCabe metrics, 4 different base Halstead measures, 8 different derived Halstead measures, a branch-count, and 1 goal field). There are no missing values in any of attributes. The class value (defects) is discrete in nature, and 90.87% are false instances compared to 9.13% true instances. We provide the summary of

Table 1.1 Description of 10 SDP datasets

Summary				
Dataset	Classes	Objects	Attributes	Defects %
CM1	2	327	22	9.13
KC3	2	194	39	18.6
MC1	2	1988	38	2.3
MC2	2	125	39	35.2
MW1	2	253	37	10.7
PC1	2	705	37	8.7
PC2	2	745	36	2.1
PC3	2	1077	37	12.4
PC4	2	1287	37	13.8
PC5	2	1711	38	27.5

ten SDP datasets in Table 1.1. It is clear from Table 1.1 that the true instances vary from 2.1 to 35.2%, which means that the false instances vary from 68.8 to 97.9% indicating a clear imbalance between the two class instances.

Performance Evaluation Metrics

For classification problems, the efficiency of a model is presented as a matrix representation of the classification results known as confusion matrix. The elements of the matrix are actual and predicted classification results produced by the classification model. After the confusion matrix for a model is generated, we calculate different metrics like precision, recall, F-measure and accuracy percentages to verify how good the model is performing. The ratio of number of positive instances which are predicted accurately to the total number of instances that are predicted positive is defined as precision and recall is defined as the ratio of number of positive instances which are predicted accurately to the total number of instances that belong to the actual class. F-measure is defined as the harmonic mean of precision and recall, and the ratio of number of correctly predicted data points to the total number of data points is defined as accuracy. In general, particularly dealing with balance data the model, which produces maximum accuracy may be considered as the best model but in case of imbalanced datasets, it may not be a good metric to evaluate model performance. For symmetric datasets where the number of false negatives and false positives are almost equal, the metric accuracy gives the best measure. However, in case of dealing with asymmetric class distribution in case of binary classification scenario, F-measure is usually more logical and capable than accuracy [19]. A F-measure value closer to 1 recommends a desired model. The formula of the different performance measures are given in the following:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN+FP}}$$

$$\text{AUC} = \frac{\text{Recall+Specificity}}{2}$$

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

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}$$

where TP (True Positive) is number of correctly positive predictions and FP (False Positive) is number of positive prediction which are incorrect, whereas the number of correctly negative prediction is denoted as TN (True Negative) and FN (False Negative) is the number of negative prediction which are incorrect, done by the model.

Analysis of Results

The hybrid methodology presented in this article has been applied to ten NASA datasets [3] mentioned in Table 1.1. Since most of the datasets mentioned in Table 1.1 are mostly imbalanced in nature, the decision tree developed on the Hellinger distance [17] has been applied. We present here the analysis on CM1 (one of the 10 NASA SDP dataset) dataset. The HDDT has been grown on the randomly splitted training set and tested on remaining 20% instances. From there, we extract the important features. The important features are LO-Comment which is Halstead's count of lines of comments, LOC-COde-And-comment. We have predicted the class of each instances based on this HDDT model. Then, we took the prediction made by HDDT as an extra input along with the important features extracted by HDDT in the input layer of ANN. Several performance metrics have been considered in this context like F-measure, AUC, and Recall since data are highly imbalanced in nature. The F-measure of the HDDT model further improved to 90%, AUC is 66%, and recall is 88%. The optimal classification tree based on Hellinger distance is given in Fig. 1.3. ANN model is quite often used as a supervised learning methodology for predicting class of unknown instances in a classification problem. We have performed the normalization of the data prior to training a neural network, the major reason behind is that if we avoid normalization then the training process may become very difficult and time-consuming. We have used min-max normalization technique to scale the data in the interval [0, 1]. Hornik et al. [28] proved the universal consistency of the feedforward neural networks. We have chosen optimum number of neurons as mentioned in [13].

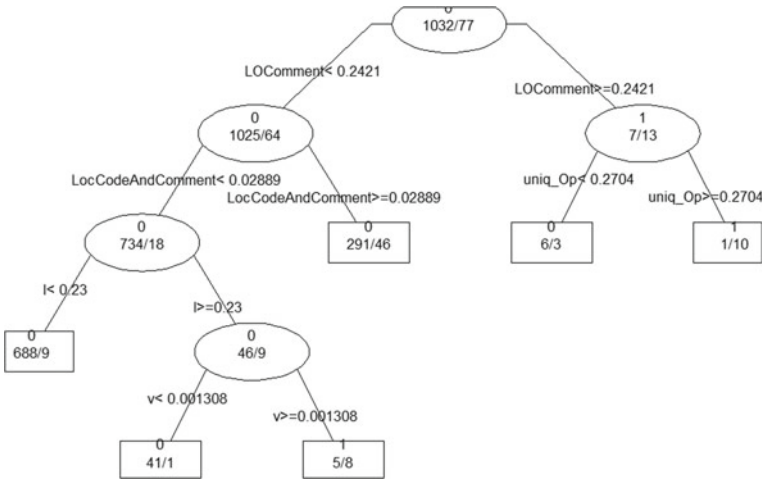

Fig. 1.3 Hellinger distance-based decision tree output on CM1 dataset

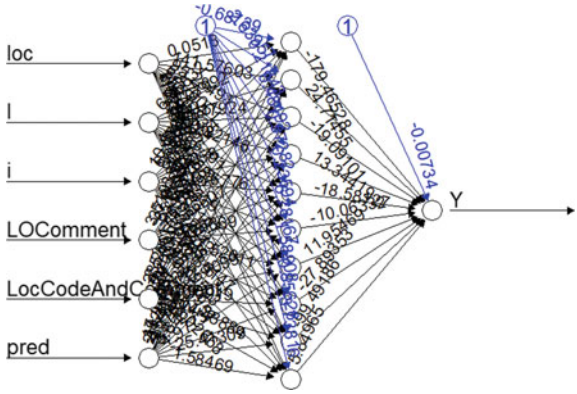

Fig. 1.4 Hybrid HDDT-ANN output on CM1 dataset

We have taken 10 neurons in the hidden layer in case of CM1 dataset. We used sigmoid activation function in the output layer.

The ANN model is presented in Fig. 1.4. After hybridization the F-measure improved to 96%, AUC improves to 90% and recall to 95%. Since the sample is drawn randomly, in each iteration we will get different values for each of metrics. We iterate 10,000 times and calculate mean and standard deviation for each of the metrics.

Table 1.2 gives the various performance metrics obtained for different models applied to these datasets, including the one of the recent methods [12]. The performance metrics of the proposed model are shown in the right most column of Table 1.2 and it clearly indicates that the proposed model is the best model so far used for imbalanced classification problems of this type, though for a few instances other models have done better occasionally.

Table 1.2 Mean (S.D.) of different metrics of various models applied on 10 SDP datasets

Dataset	Measure	ISDA	STr-NN	SMB+DT	VCB-SVM	ABNC+DT	HDDT	HDRF	Hellinger net	HDDT+ANN
CM1	*F*-measure	0.84 (0.02)	0.845 (0.012)	0.792 (0.01)	0.76 (0.03)	0.842 (0.02)	0.785 (0.015)	0.82 (0.054)	0.865 (0.052)	0.90 (0.031)
	AUC	0.55 (0.032)	0.566 (0.02)	0.653 (0.017)	0.595 (0.04)	0.66 (0.019)	0.573 (0.04)	0.671 (0.049)	0.706 (0.04)	0.66 (0.051)
	Recall	0.30 (0.01)	0.38 (0.02)	0.593 (0.001)	0.395 (0.04)	0.320 (0.005)	0.30 (0.01)	0.491 (0.049)	0.66 (0.034)	0.88 (0.053)
KC3	*F*-measure	0.73 (0.01)	0.77 (0.02)	0.78 (0.06)	0.80 (0.02)	0.732 (0.006)	0.892 (0.014)	0.902 (0.014)	0.832 (0.027)	0.94 (0.043)
	AUC	0.56 (0.05)	0.656 (0.04)	0.748 (0.03)	0.626 (0.017)	0.541 (0.07)	0.532 (0.12)	0.737 (0.01)	0.744 (0.02)	0.87 (0.045)
	Recall	0.35 (0.02)	0.5 (0.015)	0.348 (0.024)	0.465 (0.01)	0.33 (0.07)	0.584 (0.03)	0.536 (0.008)	0.60 (0.02)	0.91 (0.037)
MC1	*F*-measure	0.975 (0.001)	0.95 (0.003)	0.965 (0.002)	0.97 (0.001)	0.97 (0.005)	0.94 (0.005)	0.97 (0.004)	0.985 (0.003)	0.95 (0.002)
	AUC	0.64 (0.02)	0.76 (0.03)	0.69 (0.01)	0.66 (0.025)	0.59 (0.01)	0.532 (0.06)	0.70 (0.01)	0.80 (0.03)	0.80 (0.015)
	Recall	0.18 (0.00)	0.58 (0.001)	0.42 (0.001)	0.36 (0.008)	0.17 (0.00)	0.333 (0.003)	0.50 (0.01)	0.60 (0.001)	0.91 (0.005)
MC2	*F*-measure	0.67 (0.02)	0.71 (0.02)	0.66 (0.01)	0.61 (0.04)	0.75 (0.01)	0.57 (0.06)	0.68 (0.01)	0.73 (0.04)	0.94 (0.011)
	AUC	0.638 (0.01)	0.65 (0.01)	0.59 (0.01)	0.57 (0.02)	0.69 (0.01)	0.43 (0.06)	0.60 (0.01)	0.673 (0.008)	0.90 (0.025)
	Recall	0.60 (0.01)	0.5 (0.00)	0.33 (0.00)	0.26 (0.02)	0.4 (0.01)	0.5 (0.00)	0.55 (0.01)	0.54 (0.02)	0.92 (0.04)
MW1	*F*-measure	0.80 (0.04)	0.91 (0.014)	0.88 (0.144)	0.779 (0.132)	0.832 (0.090)	0.772 (0.120)	0.795 (0.134)	0.902 (0.017)	0.94 (0.04)
	AUC	0.612 (0.08)	0.816 (0.02)	0.70 (0.016)	0.643 (0.014)	0.58 (0.10)	0.535 (0.04)	0.627 (0.1)	0.790 (0.023)	0.56 (0.03)
	Recall	0.40 (0.01)	0.70 (0.02)	0.50 (0.06)	0.35 (0.02)	0.620 (0.05)	0.40 (0.01)	0.55 (0.04)	0.65 (0.03)	0.92 (0.015)
PC1	*F*-measure	0.915 (0.02)	0.890 (0.05)	0.89 (0.032)	0.88 (0.03)	0.92 (0.002)	0.882 (0.01)	0.902 (0.04)	0.945 (0.008)	0.96 (0.04)
	AUC	0.586 (0.08)	0.808 (0.027)	0.786 (0.008)	0.67 (0.010)	0.632 (0.07)	0.532 (0.05)	0.719 (0.08)	0.854 (0.022)	0.90 (0.04)
	Recall	0.842 (0.108)	0.858 (0.087)	0.796 (0.118)	0.838 (0.080)	0.815 (0.107)	0.832 (0.090)	0.839 (0.108)	0.87 (0.07)	0.95 (0.04)
PC2	*F*-measure	0.97 (0.00)	0.97 (0.00)	0.96 (0.01)	0.97 (0.01)	0.96 (0.00)	0.95 (0.00)	0.97 (0.00)	0.98 (0.005)	0.94 (0.001)
	AUC	0.5 (0.00)	0.5 (0.00)	0.56 (0.01)	0.45 (0.005)	0.620 (0.04)	0.432 (0.00)	0.570 (0.01)	0.66 (0.02)	0.74 (0.02)
	Recall	0 (0.00)	0.1 (0.00)	0 (0.00)	0 (0.00)	0.05 (0.00)	0.1 (0.00)	0.1 (0.00)	0.2 (0.00)	0.96 (0.04)
PC3	*F*-measure	0.888 (0.01)	0.840 (0.05)	0.82 (0.03)	0.81 (0.02)	0.880 (0.02)	0.796 (0.03)	0.883 (0.017)	0.907 (0.04)	0.94 (0.04)
	AUC	0.598 (0.018)	0.756 (0.012)	0.635 (0.03)	0.654 0.018	0.65 (0.02)	0.532 (0.04)	0.614 (0.02)	0.835 (0.009)	0.795 (0.001)
	Recall	0.48 (0.01)	0.67 (0.02)	0.45 (0.03)	0.48 (0.02)	0.38 (0.028)	0.49 (0.01)	0.64 (0.02)	0.70 (0.005)	0.93 (0.002)
PC4	*F*-measure	0.905 (0.015)	0.86 (0.01)	0.88 (0.01)	0.79 (0.02)	0.89 (0.04)	0.82 (0.006)	0.87 (0.01)	0.90 (0.02)	0.94 (0.022)
	AUC	0.805 (0.01)	0.836 (0.01)	0.80 (0.03)	0.67 (0.04)	0.744 (0.03)	0.679 (0.05)	0.783 (0.01)	0.842 (0.03)	0.72 (0.02)
	Recall	0.50 (0.06)	0.80 (0.03)	0.71 (0.01)	0.47 (0.07)	0.53 (0.04)	0.41 (0.09)	0.63 (0.03)	0.82 (0.03)	0.93 (0.01)
PC5	*F*-measure	0.74 (0.05)	0.72 (0.10)	0.75 (0.02)	0.78 (0.02)	0.73 (0.05)	0.78 (0.02)	0.80 (0.01)	0.825 (0.01)	0.96 (0.02)
	AUC	0.65 (0.003)	0.66 (0.04)	0.62 (0.06)	0.68 (0.01)	0.56 (0.05)	0.62 (0.005)	0.64 (0.06)	0.70 (0.05)	0.71 (0.04)
	Recall	0.45 (0.00)	0.360 (0.00)	0.40 (0.00)	0.26 (0.00)	0.35 (0.00)	0.25 (0.00)	0.40 (0.00)	0.50 (0.00)	0.90 (0.021)

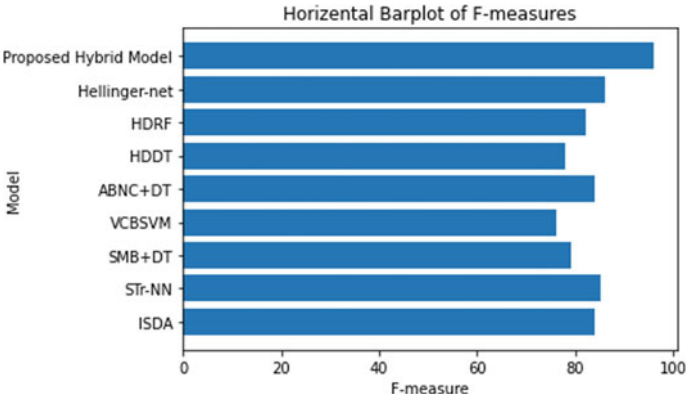

Fig. 1.5 Horizontal barplot of F-measure of different models applied on CM1 dataset

Conclusion

The objective of this article is to develop a model, which can predict whether a software module is defective or non-defective. In our study, we have produced a HDDT-ANN model which is a combination of artificial neural network and Hellinger distance-based classification tree, and it gives better results in terms of different metrics used for classification problems, specially for imbalanced datasets than all other traditional models as shown in Fig. 1.5. Splitting the data into training and testing set in the ratio 80:20 is quite popular in machine learning literature [13]. However, it is noted that when the data comes from an extremely imbalanced dataset, there are non-zero (although very low) chances of obtaining a sample which has only one class of data. But, since the samples generated randomly are in thousands, the effect of the positive probability of having a one-class sample will be much low, which can be neglected. In this study, we have used HDDT for extracting important features from imbalanced dataset and found that the hybrid HDDT-ANN model outperforms the other supervised models in the context of SDP problems. Significant accuracy compared to traditional machine learning algorithms has been achieved through the use of our experimentally optimized model for the 10 NASA SDP datasets. For a module which may be newly introduced, if we know the features of the new module in terms of Halstead's count of lines of comments, LOC-COde-And-comments, etc., then the proposed model will be able to say whether the module is defect-prone or not. The proposed hybrid HDDT-ANN model may be used for similar problems like in medical diagnosis contexts as well. Also, it is known that all defects are not of same importance. However, severity of the effects of a defect instance can be looked at as a future research topic. Some recent works [21] may be looked at for tackling such problems.

Code

We used R language to implement this hybrid HDDT-ANN methodology. The link for the code is given here: https://github.com/KARMAKAR03/Barin001.

References

1. Akash, P. S., Kadir, M. E., Ali, A. A., & Shoyaib, M. (2019). Inter-node Hellinger distance based decision tree. *IJCAI,* 1967–1973.
2. Batista, G., Bazan, A., & Monard, M. (2003). Balancing training data for automated annotation of keywords: A case study. In *Proceedings of the Second Brazilian Workshop on Bioinformatics* (pp. 35–43).
3. Boetticher, G. (2007). The promise repository of empirical software engineering data. http://promisedata.org/repository
4. Boonchuay, K., Sinapiromsaran, K., & Lursinsap, C. (2017). Decision tree induction based on minority entropy for the class imbalance problem. *Pattern Analysis and Applications, 20*(3), 769–782.
5. Bouaziz, S., Dhahri, H., Alimi, A. M., & Abraham, A. (2013). A hybrid learning algorithm for evolving flexible beta basis function neural tree model. *Neurocomputing, 117,* 107–117.
6. Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC Press.
7. Briand, L. C., Emam, K. E., Freimut, B. G., & Laitenberger, O. (2000). A comprehensive evaluation of capture-recapture models for estimating software defect content. *IEEE Transactions on Software Engineering, 26*(6), 518–540.
8. Catal, C., & Diri, B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences, 179*(8), 1040–1058.
9. Chaabane, I., Guermazi, R., & Hammami, M. (2019). Enhancing techniques for learning decision trees from imbalanced data. In *Advances in Data Analysis and Classification* (pp. 1–69). Springer.
10. Chakraborty, A. K., & Arthanari, T. S. (1994). Optimum testing time for software under an exploration model. *OPSEARCH, 31,* 202.
11. Chakraborty, T., & Chakraborty, A. K. (2020). Superensemble classifier for improving predictions in imbalanced datasets. *Communications in Statistics: Case Studies, Data Analysis and Applications, 6*(2), 123–141.
12. Chakraborty, T., & Chakraborty, A. K. (2021). Hellinger net: A hybrid imbalance learning model to improve software defect prediction. *IEEE Transactions on Reliability, 70*(2), 481–494.
13. Chakraborty, T., Chattopadhyay, S., & Chakraborty, A. K. (2018). A novel hybridization of classification trees and artificial neural networks for selection of students in a business school. *OPSEARCH, 55*(2), 434–446.
14. Chakraborty, A. K., Basak, G. K., & Das, S. (2019). Bayesian optimum stopping rule for software release. *OPSEARCH, 56*(1), 242–260.
15. Chen, Y., Abraham, A., & Yang, J. (2005). Feature selection and intrusion detection using hybrid flexible neural tree. In *Advances in Neural Networks—ISNN 2005* (p. 980).
16. Chen, Y., Yang, B., & Meng, Q. (2012). Small-time scale network traffic prediction based on flexible neural tree. *Applied Soft Computing, 12*(1), 274–279.
17. Cieslak, D. A., Hoens, T. A., Chawla, N. V., & Kegelmeyer, W. P. (2012). Hellinger distance decision trees are robust and skew-insensitive. *Data Mining and Knowledge Discovery, 24*(1), 136–158.

18. Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers, 3,* 326–334.
19. Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning* (pp. 233–240). ACM.
20. Dewanji, A., Sengupta, D., & Chakraborty, A. K. (2011). A discrete time model for software reliability with application to a flight control software. *Applied Stochastic Models in Business and Industry, 27*(6), 723–731.
21. Dey, S., & Chakraborty, A. K. (2022). Estimating software reliability using size-biased concepts.
22. Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering, 25*(5), 675–689.
23. Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research, 61,* 863–905.
24. Foresti, G. L., & Dolso, T. (2004). An adaptive high-order neural tree for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34*(2), 988–996.
25. Gong, L., Jiang, S., Bo, L., Jiang, L., & Qian, J. (2019). A novel class imbalance learning approach for both within-project and cross-project defect prediction. *IEEE Transactions on Reliability, 69*(1), 40–54.
26. Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2009). Using the support vector machine as a classification method for software defect prediction with static code metrics. In *International Conference on Engineering Applications of Neural Networks* (pp. 223–234). Springer.
27. Guo, L., Ma, Y., Cukic, B., & Singh, H. (2004). Robust prediction of fault proneness by random forests. In *15th International Symposium on Software Reliability Engineering* (pp. 417–428). IEEE.
28. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359–366.
29. Jing, X. Y., Fei, W., Dong, X., & Xu, B. (2016). An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering, 43*(4), 321–339.
30. Khoshgoftaar, T. M., & Seliya, N. (2002). Tree-based software quality estimation models for fault prediction. In *Proceedings Eighth IEEE Symposium on Software Metrics* (pp. 203–214). IEEE.
31. Khoshgoftaar, T. M., Allen, E. B., Jones, W. D., & Hudepohl, J. I. (1999). Classification tree models of software quality over multiple releases. In *Proceedings 10th International Symposium on Software Reliability Engineering* (Cat. No. PR00443) (pp. 116–125). IEEE.
32. Kim, K. (2016). A hybrid classification algorithm by subspace partitioning through semi-supervised decision tree. *Pattern Recognition, 60,* 157–163.
33. Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology, 58,* 388–402.
34. Lee, D. S., & Srihari, S. N. (1995). A theory of classifier combination: The neural network approach. In *Proceedings of 3rd International Conference on Document Analysis and Recognition* (Vol. 1, pp. 42–45).
35. Liu, M., Miao, L., & Zhang, D. (2014). Two-stage cost-sensitive learning for software defect prediction. *IEEE Transactions on Reliability, 63*(2), 676–686.
36. Lopez, V., Fernandez, A., Garcıa, S., Palade, V., & Herrera, F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences, 250,* 113–141.
37. Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi disciplinary survey. *Data Mining and Knowledge Discovery, 2*(4), 345–389.
38. Pelayo, L., & Dick, S. (2012). Evaluating stratification alternatives to improve software defect prediction. *IEEE Transactions on Reliability, 61*(2), 516–525.

39. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Technical report). University of California, Institute for Cognitive Science, La Jolla, San Diego.
40. Ryu, D., Choi, O., & Baik, J. (2016). Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empirical Software Engineering, 21*(1), 43–71.
41. Sakar, A., & Mammone, R. J. (1993). Growing and pruning neural tree networks. *IEEE Transactions on Computers, 42*(3), 291–299.
42. Sethi, I. K. (1990). Entropy nets: From decision trees to neural networks. *Proceedings of the IEEE, 78*(10), 1605–1613.
43. Shatnawi, R. (2012). Improving software fault-prediction for imbalanced data. In *2012 International Conference on Innovations in Information Technology (IIT)* (pp. 54–59). IEEE.
44. Sirat, J., & Nadal, J. (1990). Neural trees: A new tool for classification. *Network Computation in Neural Systems, 1*(4), 423–438.
45. Subasi, A., Molah, E., Almkallawi, F., & Chaudhery, T. J. (2017). Intelligent phishing website detection using random forest classifier. In *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)* (pp. 1–5). IEEE.
46. Sun, Z., Song, Q., & Zhu, X. (2012). Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42*(6), 1806–1817.
47. Turhan, B. (2012). On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering, 17*(1), 62–74.
48. Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability, 62*(2), 434–443.
49. Zheng, J. (2010). Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications, 37*(6), 4537–4543.
50. Zhou, Z. H., Wu, J., & Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial Intelligence, 137*(1–2), 239–263.
51. Zimmermann, T., Nagappan, N., & Zeller, A. (2008). Predicting bugs from history. In *Software evolution* (pp. 69–88). Springer.