Computing (2017) 99:255–285
DOI 10.1007/s00607-016-0489-6

A decision tree logic based recommendation system to select software fault prediction techniques

Santosh S. Rathore[1] · **Sandeep Kumar**[1]

Received: 22 January 2015 / Accepted: 24 February 2016 / Published online: 21 March 2016

Abstract Identifying a reliable fault prediction technique is the key requirement for building effective fault prediction model. It has been found that the performance of fault prediction techniques is highly dependent on the characteristics of the fault dataset. To mitigate this issue, researchers have evaluated and compared a plethora of fault prediction techniques by varying the context in terms of domain information, characteristics of input data, complexity, etc. However, the lack of an accepted benchmark makes it difficult to select fault prediction technique for a particular context of prediction. In this paper, we present a recommendation system that facilitates the selection of appropriate technique(s) to build fault prediction model. First, we have reviewed the literature to elicit the various characteristics of the fault dataset and the appropriateness of the machine learning and statistical techniques for the identified characteristics. Subsequently, we have formalized our findings and built a recommendation system that helps in the selection of fault prediction techniques. We performed an initial appraisal of our presented system and found that proposed recommendation system provides useful hints in the selection of the fault prediction techniques.

Keywords Recommendation system · Software fault prediction · Decision tree · Software fault prediction techniques

Electronic supplementary material The online version of this article (doi:10.1007/s00607-016-0489-6) contains supplementary material, which is available to authorized users.

✉ Sandeep Kumar
sandeepkumargarg@gmail.com

Santosh S. Rathore
sunnydec@iitr.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

Mathematics Subject Classification 68N30 Mathematical aspects of software engineering (specification, verification, metrics, requirements, etc.)

1 Introduction

Software fault prediction (SFP), in principle, can play a vital role to ameliorate and streamline the software quality assurance process. It may help in reducing unnecessary fault finding efforts during the development of the software system. Typically, a fault prediction system uses software metrics and fault dataset (collected from the previous releases or similar projects) to train a fault-prediction model. Subsequently, it uses this model to predict the faults in the current release of the software system. It helps in locating the software modules that are more prone to faults. This could be helpful when the project resources are limited or the system is too large and could not be tested exhaustively.

Various factors affect the performance of software fault prediction such as, software metrics, learning algorithms and others. One of the concerns with software fault prediction is the evolution of code bases [1]. Suppose, we built a fault prediction model based on some set of metrics and used it to predict the faults in the given software system and fixed the faults. Now, software system has evolved to accommodate the changes, but there may be the case when the values of used set of metrics did not change. In that case, if we reuse the built fault prediction model, it will re-raise the same code area as fault prone. This is a general problem of fault prediction models, if we use the code metrics [2]. To handle this issue, some researchers have proposed different set of metrics, such as software change metrics, file status metrics, etc [1,2]. These metrics analyze the change history of the software to determine the modules that are more likely to have faults when software evolved.

For developing a software fault prediction model, choosing a better learning algorithm is shown to be equally important as selection of software metrics and other parameters [3–5]. However, the selection of correct fault prediction technique is a challenging issue. Numerous variables and factors influence this selection process. Previously, a myriad of different machine learning and statistical methods have been proposed and validated by various researchers to facilitate fault prediction process. It includes the techniques based on Naive Bayes [6], Logistic Regression [7], Support Vector Machine [8], Neural Network [9], Ensemble classifiers [10], etc. There are variations upon the use of these techniques to build fault prediction model [11].

Many authors have performed studies comparing an extensive set of techniques for their fault prediction capabilities, and have reported the usability of the techniques in various contexts [12–16]. The aim of these studies was to choose the best predictor among many other available techniques for improved fault prediction. Catal et al. [11] have compared various machine-learning techniques for software fault prediction problem and found that characteristics of the dataset have a strong influence over the performance of the fault prediction techniques. The technique found best for one type of dataset might be performing poorly for another type of dataset. In another study, Menzies et al. [3] have assessed various classification techniques for software defects prediction and found that for different types of datasets, the performance of the

prediction techniques also varies. These studies show there is no single best technique that can be used with any type of dataset for fault prediction. Moreover, we found that there is a contradictory view over the usefulness of the fault prediction techniques. Some studies established the usability of some techniques for fault prediction, while other studies prompting questions about the same set of techniques. One possible reason for this unpredictable performance of the fault prediction techniques is that most of the studies have used the fault prediction techniques as a black box without analyzing the domain of the dataset. The fault prediction model may perform best when right technique is selected for the right set of dataset.

What sorely missing is a clear understanding that up to what extent data influence the performance of classifiers and how to make choice of the prediction techniques for improved fault prediction process. This raises the need of a baseline framework or recommendation system, which based on the characteristics of the software fault dataset, can suggest technique(s) for building fault prediction model. To the best of our knowledge, no such system has been reported in the literature. We aim to provide such a recommendation system by gathering an extensive knowledge about the fault dataset characteristics and by developing the correlation-ship between the fault prediction techniques and the identified fault data characteristics. As an effort, this paper presents a decision tree logic (DTL) based recommendation system that may be helpful to the practitioners or the researchers in the selection of the most appropriate fault prediction technique for a given target system.

In a software product, if we are predicting faults across the different releases, then a technique valid for one release should be consistently used for other releases also. But, if there are significant changes in a release of the software system that possess different fault dataset characteristics, then the technique used for fault prediction should also change. This situation is also taken care of by the proposed recommendation system. The proposed system initially identified the fault dataset characteristics of the given software and the based on these identified characteristics the recommendation regarding the suitability of fault prediction technique is given. Hence, if there is no significant change in the fault characteristics for a given release then there will not be change in the recommendation regarding fault prediction techniques. But, in the case, there is a significant change in the fault datasets characteristics then the recommendation regarding the fault prediction techniques is changed.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the taxonomical classification of software fault prediction techniques. Technique recommendation system is presented in Sect. 4. It includes architecture of the recommendation system, system workflow, some sample rules and implementation of recommendation system. Section 5 specifies the details of the datasets used and evaluation of recommendation system. Some case studies to validate the recommendations provided by the proposed system are presented in Sect. 6. A comparative analysis of the proposed work with other similar existing works is given in Sect. 7. The paper has been concluded in Sect. 8.

2 Related work

An important issue associated with the software fault prediction is the problem of selecting appropriate fault prediction techniques. Simply put, not all techniques can provide accurate prediction across different datasets and using them for prediction without analyzing the datasets domain may lead to lower prediction performance. Some efforts have been reported to solve this problem in order to provide guidance regarding the fault prediction technique selection.

Challagulla et al. [12] performed an empirical study using various machine learning techniques for software fault prediction. The study was performed over four software projects taken from the NASA data repository. Results found that not a single machine learning technique is consistent in predicting faults with higher accuracy across different datasets. Further, results suggested that the best choice of a fault prediction technique depends on the dataset available at a particular moment. Dejaeger et al. [17] investigated the performance of Bayesian Network Classifiers for software fault prediction. They have used fifteen different Bayesian Network based classifiers and compared them with other popular machine learning techniques. Results found that Naive Bayes and Random Forest are the most accurate predictors of software faults among the techniques considered and the performance of best technique depends on the development context. Ma et al. [10] presented a statistical framework for fault proneness prediction. The study compared the performance of Random Forest with ten other classifiers over five NASA datasets. Results showed that Random Forest performed well for large and diverse datasets, while Logistic Regression and Discriminant Analysis have produced a good performance for small datasets.

Stefan Lessmann et al. [15] performed a comparative analysis of twenty-two classifiers for software defect prediction. The experiments were performed over the ten datasets collected from NASA data repository. Additionally, they have applied hypothesis-testing methods to investigate the statistical significance of performance difference among the different used classifiers. Results found that there is no significant performance difference among the top seventeen used classifiers. Furthermore, they have investigated whether certain type of classifiers have produced significant accuracy over some datasets and then can be used, if similar type of software projects are encountered in future. However, no recommendation system or framework has been provided to guide the selection of fault prediction techniques. In another study, Zhongbin et al. [14] investigated the performance of four classification algorithms, three data coding schemes, and six conventional imbalance data handling methods over fourteen NASA datasets for handling imbalanced data problem. Results showed that C4.5, Ripper, and Random Forest performed better for imbalanced data, while Naive Bayes produced an average performance. Kanmani et al. [18] investigated the effectiveness of Probabilistic Neural Network (PNN), Back propagation Neural Network (BPN) and Discrimination Analysis over a dataset collected from student projects. The experiment was carried out on a small system and found that overall PNN based prediction models performed well in comparison to other used techniques.

Zimmermann et al. [19] reported a study for cross-project defect prediction over twelve software project datasets collected from the real-world applications. Additionally, they have identified various factors that do influence the success of the

cross-project defect prediction. Their study provided some suggestions regarding the dataset selection when performing cross-project prediction. However, no information was provided regarding the selection of the fault prediction techniques for the given datasets. Pickard et al. [20] performed an investigation of three statistical-based data analysis techniques for software fault prediction. They considered skewness, unstable variance, and outliers characteristics of the considered fault dataset and investigated the performance of the used analysis techniques with respect to these characteristics. Results found that no single analysis technique produced best results. Further, they suggested that it is important to identify suitable fault prediction technique to be used in a given environment for better prediction results.

These studies provide some guidance regarding the use of fault prediction techniques. However, they did not discuss the effect of dataset characteristics on the performance of classifiers comprehensively. Furthermore, they have not proposed or discussed any framework that helps the naive users for appropriate fault prediction technique selection. To fulfill this gap, our work aims to present a recommendation system that takes input of some parameter values regarding the fault datasets characteristics from the user and based on the given values, it suggests the best-suited techniques to build fault prediction model.

3 Classification of software fault prediction techniques

Various researchers have used different set of techniques to develop software fault prediction models. It includes several statistical techniques such as Logistic Regression, Discriminant Analysis, etc. and different machine learning techniques such as Decision Trees, Neural Network, Support Vector Machines and some ensemble techniques like Random Forest, Bagging, etc. [6–10]. In their paper, Dejaeger et al. [17] proposed a taxonomic classification of software fault prediction techniques. Among others, the works on classification of fault prediction techniques available are Stefan et al. [15], Witten et al. [21]. Encouraged from these works, we have proposed a theoretically complete classification of software fault prediction techniques as shown in Fig. 1. We further used it for recommendation system development.

Tree based classifiers are supervised learning methods used for classification. They create a tree type of structure that classify the value of target variable in one of the given classes by learning some decision rules deduced from the data features [22]. Feature

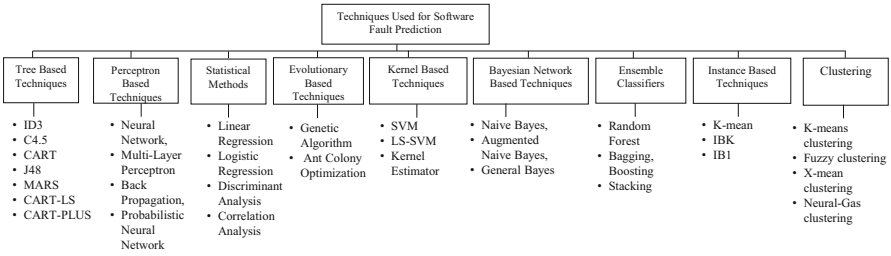

Fig. 1 Classification of software fault prediction techniques

that best splits the training data will serve as the root node of the tree. Perceptron based techniques consists of a series of processing elements interconnected through the connection weights in the form of layers. During the training phase, based on the domain knowledge, they develop an internal representation that maps the input stimulus space to the output response space [23]. Statistical based techniques formulate and use some statistical formula to determine the relationship between the software module properties and the fault proneness [23]. Evolutionary based techniques generally use some search methods to mimic the process of natural selection. They generate a population of solutions and then evolve them towards the best solution. The algorithm usually starts from a population of randomly generated solutions and at each iteration the fitness of generated solution is evaluated and modified to find out the optimal solution [24]. A Bayesian Network shows a joint probability distribution over a set of discrete or continuous variables. It is a graphical model that contains a set of nodes representing various variables and directed arcs showing the existence of dependencies between variables [17]. Ensemble classifiers are the learning methods that use multiple learning algorithms, instead of one, to obtain the better prediction results. Each of the used learning algorithms solves the same original task and then results are combined to obtain a better global model with more accuracy and reliability compared to the one obtained by any single learning method [25]. Instance based classifiers are the learning methods that compare unseen instances with the instances given in the data. Based on the stored knowledge, it classifies the instances into one of the given classes. They do not generate any explicit generalization of the data points. Generally, they are known as lazy classifiers [23]. Clustering based techniques are unsupervised learning techniques that group a set of objects in such way that instances of similar properties are kept in the same group. Clustering analysis uses an iterative task that learns from the training data and updates its knowledge to find out the optimal set of groups [26].

4 Proposed recommendation system

The proposed fault prediction technique recommendation system is based on the concept of decision tree logic. It intended to help the practitioners or the researchers in selecting the most appropriate fault prediction technique to use for building any software fault prediction model. The system inquires about some parameters related to the fault dataset characteristics to the users and based on the provided answers it recommends the most suitable technique that can be used for building fault prediction model for the given fault dataset.

4.1 Architecture of the proposed recommendation system

The basic modules of the recommendation system are shown in Fig. 2. The presented recommendation system has four parts: Rule database, knowledge base and decision tree learner, interface module and recommendation display module.

Rule database consist the set of rules formed by collecting the information from various sources. The knowledge base is composed of inference rules, facts, and rela-

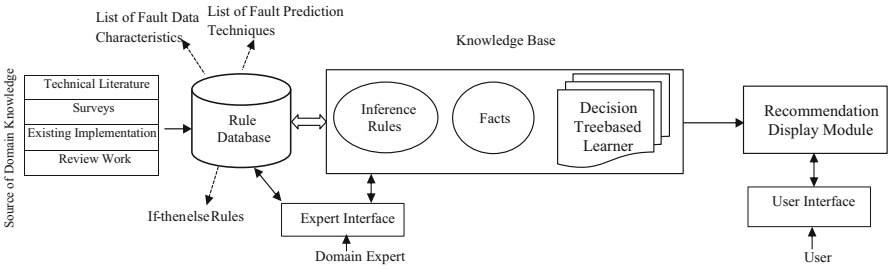

Fig. 2 Architecture of recommendation system

tionships used by the recommendation system. It stores the inference rules in the form of "F...THEN...ELSE". The rules are formed by varying the values of the fault dataset characteristics. The knowledge base also consists of a decision tree trained from the available rules. The main function of the decision tree is to provide an appropriate and convenient way to capture and store all the information available in the knowledge base. The user can interact with the system using the user interface module. This allows user to select the values of various parameters regarding the fault dataset that helps in selection of fault prediction techniques. The expert interface allows the expert to interact with the system. The expert has the right to review, delete, or modify any parameter from the database. The expert interface module and user interface module help recommendation system to learn about the rules and users when any modification is needed. The system takes fault data characteristics as input from the user and the recommendations are generated as output using the decision tree.

4.2 System work flow

It can be seen from Fig. 2 that one of the major task for the development of the proposed recommendation system is to develop a decision tree (Sect. 4.2.2). However, prior to that various characteristics of fault datasets to be considered for decision making also need to be finalized (Sect. 4.2.1).

4.2.1 Dataset characteristics

We have surveyed the literature and identified following ten characteristics of the fault dataset that most influence the performance of the classifiers during software fault prediction:

1. **Noise** According to Manago and Kodratoff [27] "noise is present when a knowledge base does not truly reflect the environment we want to learn from". It reflects the "lack of information or unreliable information". The presence of noise reduces the efficiency of the classifiers and the learning algorithm cannot find a function that exactly matches the training examples.
2. **High dimensionality of input space** High dimensionality of data refers to having too many features (software metrics) in the input/training dataset. The high dimension of input feature vectors may cause a problem for learning algorithm [28,29].

3. **Heterogeneity of the data** If the feature vectors contain features of many different natures such as discrete, discrete ordered, counts, continuous values etc., then data is called as heterogeneous data. Some algorithms are easy to apply for this type of data, while many other algorithms require that the input features be numerical and scaled to similar ranges [30].
4. **Redundancy in the data** Redundant instances occur when the same feature describes multiple modules with the same class label [30]. Some algorithms will perform poorly in the presence of redundant data points because of numerical instabilities.
5. **Outlier** Outliers are the data points that do not meet with the general behavior of the data. Such data points, which are different from the remaining data points, are called outlier [31]. They are also referred to as abnormalities, discordant, deviants, or anomalies in the datasets.
6. **Missing value** Missing value is the value that left blank in the dataset [30]. Some of the prediction techniques can automatically deal with the missing values and no special care is required. Whereas, some techniques others require extra care for this.
7. **Amount of training data** Amount of training data available to train learning algorithm plays an important role in the classifier performance. If the training set is small, high bias/low variance classifiers have an advantage over low bias/high variance classifiers, since the later will overfit [12].
8. **Class imbalance** Class imbalance represents a situation where certain type(s) of instances (called as minor class) are rarely present in the dataset compared to the other types of instances (called as major class). It is a common issue in prediction, where the instances of major class dominate the data sample as opposed to the instances of the minor class. In such cases, learning of the classifiers may be biased towards the instances of major class. Moreover, classifiers can produce poorer results for the minor class instances [32].
9. **Learning function** Type of interaction between the features (software metrics for the fault dataset) may influence the classifiers performance. If each of the features makes an independent contribution to the output, then algorithms based on linear functions generally perform well. However, if there are complex interactions between features, then algorithms based on non-linear function work better [8].
10. **Type of dependent variable** Dependent variable represents the type of output value or response of a classifier. It can of categorical or continuous types. Some algorithms are able to handle both types of variable, whereas, others require specific type of the dependent variable for prediction.

We have gathered evidences from existing empirical and theoretical studies [31, 33–38] to decide the possible set of values for the various fault data characteristics mentioned above. Based on the observations obtained from these studies, we have decided the values of different fault dataset characteristics. Following are the range of the values that are used to decide that when a characteristic can be marked as 'yes' and when it should be marked as 'no'.

1. Noise: yes ($\geq$40 %), no ($<$40 %) [35]
2. Missing value: yes ($\geq$30 %), no ($<$30 %) [33]

3. Imbalanced data: yes (>20 %), no (≤20 %) [34,36]
4. High data dimension: yes (number of features (n) > 20), no ($n \leq 20$) [37]
5. Outlier: yes (>5 %), no (≤5 %) [31]
6. Amount of training data: small (number of examples (n) ≤ 500), moderate ($500 < n < 1000$), Large ($n \geq 1000$) [37]
7. Data redundancy: yes (if two features has correlation ≥80 %), no (otherwise) [38]
8. Heterogeneity of data: yes (if features follow different distribution), no (otherwise) [37]
9. Learning function: (a) linear, (b) non-linear [8]
10. Dependent variable type: (a) continuous, (b) categorical [37].

For gathering the values of these characteristics, user needs to get an understanding of fault dataset beforehand. Subsequently, based on the knowledge of the dataset, value of each characteristic can be marked as 'yes' or 'no'.

4.2.2 Generation of decision tree

A decision tree logic based recommendation system comprises a set of if-then-else rules, a collection of facts, and an interpreter to analyze the rules given the facts. These if-then-else rules are used to devise the conditional statements that comprise the complete knowledge base [60].

The main advantage of using decision tree based approach is its comprehensibility. Moreover, it performs well even if its assumptions are somewhat violated by the true model from which the data were generated [22,23]. It handles overfit problem in an easy and well-mannered way. Decision tree uses a pruning method to avoid the overfit problem and gracefully adopt for the available training data [22]. Other advantage of building a model using a decision tree is that it is easy to develop and is fast in classifying unseen examples [61]. In addition, decision tree can be used to generate a set of rules that are easy to understand while providing accuracy comparable to other techniques. The use of rules with the extension of decision trees is easy to understand, easy to generate and they can classify new examples efficiently [60]. They improve the performance of a recommendation system by introducing domain knowledge with the decision tree [62]. In addition, decision tree is capable to learn the domain even if the partial information about the fault dataset characteristics is available [23].

4.2.2.1 Designing rules for decision tree development For the development of the decision tree for the proposed recommendation system, initially some significant if-then-else rules need to be designed. To design the rules, we performed an extensive study of the literature [23,24,40,41,43–45,50,58] and gather information about the influence of various characteristics of the fault dataset (Sect. 4.2.1) on the various considered fault prediction techniques (Sect. 3). The output of this study is given in Table 1. The last column of this table shows the studies based upon which the suitability of fault prediction techniques (first column of the table) for the given set of fault dataset characteristics (column 2 to 11 of the table) is established. The values corresponding to dataset characteristics stored in the table are showing the scenarios in which a particular technique could be applicable. The entries marked = yes in the Table 1 indicate that corresponding classifier can handle both the presence and the absence of the particular

Table 1 Knowledge database regarding the appropriateness of fault prediction techniques for given characteristics

Techniques/ characteristics	Noise	Missing values	Imbalanced data	High data dimensionality	Data redundancy	Outlier	Amount of training data requires	Heterogeneity of data	Linearities/ non-linearities	Dependent variable (categorical or continuous)	Papers based on which decision taken
Values of fault dataset characteristics for which given techniques are appropriate											
Tree based (decision tree, C4.5 and J48)	Yes	Yes	No	No	Yes	Yes	Large	Can handle	Non-linear	Categorical	[22,23,26]
Continuous tree based	Yes	Yes	No	No	Yes	Yes	Large	Can handle	Non-linear	Continuous	[39]
Perceptron based (Neural Network)	Yes	Yes	Yes	Yes	No	No	Small/large	Sensitive	Non-linear	Both	[23,40,41]
Back propagation neural network based	Yes	Yes	Yes	Yes	Yes	No	Small/large	Sensitive	Non-linear	Both	[42]
Probabilistic Neural Network based	Yes	Yes	Yes	Yes	Yes	Yes	Small/large	Sensitive	Non-linear	Both	[43]
Evolutionary based (genetic programming)	Yes	Yes	No	No	Yes	Yes	Small/large	Can handle	Don't care	Both	[24,44–47]
Multi objective evolutionary base	Yes	Yes	Yes	No	Yes	Yes	Small/large	Can handle	Don't care	Both	[48]

Table 1 continued

Techniques/ characteristics	Noise	Missing values	Imbalanced data	High data dimensionality	Data redundancy	Outlier	Amount of training data requires	Heterogeneity of data	Linearities/ non-linearities	Dependent variable (categorical or continuous)	Papers based on which decision taken
Kernel based (SVM)	No	Yes	Yes	No	No	No	Small	Sensitive	Don't care	Both	[8,49]
Statistical methods (linear regression, logistic regression)	Yes	Yes	Yes	Yes	No	No	Moderate/ large	Sensitive	Logistic can be nonlinear	Linear can handle both logistic handles categorical	[50–52]
Bayesian Network (Naïve Bayes)	Yes	Yes	No	Yes	No	No	Small	Can handle	Linear	Categorical	[53,54]
Ensemble classifiers (Random Forest, Bagging, etc.)	Yes	Yes	Yes	Yes	Yes	Yes	Large	Can handle	Don't care	Both	[10,14,25, 54]
Instance based (KNN, IBK)	No	Yes	Yes	No	Yes	No	Small/large	Sensitive	Non-linear	Both	[45,55]
Clustering	Yes	Yes	No	Yes	No	No	No training data requires	Sensitive	Don't care	Not applicable	[26,56,57]
Multi-objective clustering	Yes	Yes	No	Yes	No	No	No Training data requires	Can handle	Don't care	Not applicable	[58,59]

characteristic. While, entries marked = no indicate that corresponding classifier cannot handle the presence of the particular characteristic. For example, in case of tree-based classifiers, the presence of noise does not affect the classifier performance [23]. Therefore, it is marked as 'yes'. While, for imbalanced data points, performance of tree-based classifiers degrades, therefore, it is marked as 'no' [22]. The rules are developed based upon this study. As it is clear from Table 1, the rules will be dealing with all ten characteristics of the datasets. Out of these ten characteristics, eight can have two possible values and remaining two can have three possible values. Initially, we have designed 246 rules representing all the values of involved characteristics. These are further used to train the decision tree. The motive to use decision tree is that in some cases it is possible that values of all fault data characteristics are not available. Since, decision tree has the capability of inferring the decision from the incomplete knowledge also. Therefore, it helps in making recommendation of the fault prediction techniques, even if partial domain information is available. It can be seen from the Table 1 that for ten fault dataset characteristics, total possible number of cases are 2304. Therefore, performing the manual analysis and taking the manual decision can be highly speculative. Using decision tree based recommendation system can be very helpful in the selection of fault prediction techniques and also it reduces the work of manual selection. Some of the sample rules are given in Table 2. Each rule is a composition of all ten-fault dataset's characteristics. The rationale underlying these rules is also discussed.

Dataset characteristics corresponding to rule 1 specify that noise, outliers, data redundancy, and missing values are present in the dataset. But, dataset does not have imbalanced data points and does not have high dimension. Also, the given dataset is heterogeneous. The learning function of the classifier is non-linear and dependent variable is of categorical type. In this case, tree-based classifiers are best suited for prediction [22,23,26]. If the dependent variable is of continuous type, but all other characteristics are same as in rule 1 then, continuous tree based techniques are best suited for prediction (rule 2) [39].

If amount of training data is small and learning function is of linear or non-linear, but rest of the parameters are same as is rule 1, then according to rule 6, evolutionary based classifiers are best suited for prediction [24,44,45]. However, these types of classifiers are complex in nature and require higher efforts to optimize the classifier parameters. If a situation occurs, where evolutionary based classifiers and tree based classifiers both can be used, then choose tree based classifiers over evolutionary classifiers because it requires lesser efforts for optimizing classifier parameters and for model building [22, 46,47]. If dataset has imbalanced data points, and all other parameter values are same as in rule 6, then multi-objective evolutionary based classifiers are best suited for prediction (rule 7) [48].

Rule 3 shows that noise is present in the dataset. Missing values and imbalanced data points are also present in dataset. Dataset has high dimension. But, data redundancy and outliers are not present in the dataset. Amount of training data requires is small. Heterogeneous data is not present and learning function is non-linear. Dependent variable is of categorical type. In this case, perceptron based classifiers are suited for prediction [23,40,41]. If dataset has redundant values and all other parameters are same as in rule 3, back propagation neural network is suited for prediction (rule 4) [42].

Table 2 Sample rules for recommendation system

1. If ((Noise == yes) ∧ (Missing value == yes) ∧(Imbalanced Data == no) ∧(High Data Dimension == no) ∧(Data Redundancy == yes) ∧(Outlier == yes) ∧(Amount of Training Data == large) ∧(Heterogeneous Data == yes) ∧(Function == non-linear) ∧(Dependent Variable == Categorical)) then Tree Based classifiers	8. If ((Noise == no) ∧(Missing value == yes) ∧(Imbalanced Data == yes) ∧ (High Dat∧Dimension == no) ∧(Data Redundancy == no) ∧ (Outlier == no) ∧ (Amount of Training Data == small) ∧ (Heterogeneous Data == no) ∧ (Function == don't care) ∧ (Dependent Variable == Categorical)) then SVM
2. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == no) ∧ (High Data Dimension == no) ∧ (Dafa Redundancy == yes) ∧ (Outlier == yes) ∧ (Amount of Training Data == large) ∧ (Heterogeneous Data == yes) ∧ (Function == non-linear) ∧ (Dependent Variable == Continuous)) then Continuous Tree Based classifiers	9. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == no) ∧ (Outlier == no) ∧ (Amount of Training Data == Moderate) ∧ (Heterogeneous Data == no) ∧ (Function == linear) ∧ ((Dependent Variable == Continuous)) then Linear Regression
3. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == no) ∧ (Outlier == no) A(Amount of Training Data == small) ∧ (Heterogeneous Data == no) ∧ (Function == non-linear) ∧ (Dependent Variable == Continuous)) then Perceptron Based Techniques	10. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == no) ∧ (Outlier == no) ∧ (Amount of Training Data == Moderate) ∧ (Heterogeneous Data == no) ∧ (Function == non-linear) ∧ ((Dependent Variable == Categorical)) then Logistic Regression
4. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == yes) ∧ (Outlier == no) ∧ (Amount of Training Data == small) ∧ (Heterogeneous Data == no) ∧ (Function == non-linear) ∧ (Dependent Variable =Continuous))) then Back Propagation Neural Network	11. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == no) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == no) ∧ (Outlier == no) ∧ (Amount of Training Data == small) ∧ (Heterogeneous Data == yes) ∧ (Data Redundancy == no) ∧ (Function == linear) ∧ (Dependent Variable == Categorical)) then Bayesian Classifiers
5. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == yes) ∧ (Outlier == yes) ∧ (Amount of Training Data == small) ∧ (Heterogeneous Data == no) ∧ (Function == non-linear) ∧ (Dependent Variable=Continuous))) then Probabilistic Neural Network	12. If ((Noise == no) ∧ (Missing value == yes) ∧ (Imbalanced Data == no) ∧ (High Data Dimension == yes) ∧ (V Data Redundancy == no) ∧ (Outlier == yes) ∧ (Amount of Training Data == Large) ∧ (Heterogeneous Data == no) ∧ (Function == don't care) ∧ ((Dependent Variable == Continuous))) then Ensemble Classifiers
6. If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == no) ∧ (High Data Dimension == no) ∧ (Data Redundancy == yes) ∧ (Outlier == yes) ∧ (Amount of Training Data == small) ∧ (Heterogeneous Data == yes) ∧ f Function == don't care) A ((Dependent Variable == Categorical)) then Evolutionary Based Techniques	13: If ((Noise == no) ∧ (Missing value == yes) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == no) ∧ (Data Redundancy == yes) ∧ (Outlier == no) ∧ (Amount of Training Data == small) ∧ (Heterogeneous Data == no) ∧ (Function == non-linear) ((Dependent Variable == Categorical)) then Instance Based Classifiers

Table 2 continued

7. If ((Noise == no) ∧ (Missing value == no) ∧ (Imbalanced Data == yes) ∧ (High Data Dimension == no) ∧ (Data Redundancy == yes) ∧ (Outlier == yes) ∧ (Amount of Training Data == large) ∧ (Heterogeneous Data == yes) ∧ (Function = =don't care) ∧ ((Dependent Variable == Categorical)) then Multi-objective Evolutionary Based Techniques	14: If ((Noise == yes) ∧ (Missing value == yes) ∧ (Imbalanced Data == no) ∧ (High Data Dimension == yes) ∧ (Data Redundancy == no) ∧ (Outlier == no) ∧ (Heterogeneous Data == no) ∧ (Function == non-linear)) then Clustering
15. If ((Noise == yes Mjwe == "$_{0y}$) ∧ (Missing value == yes) f ∧ (Data Redundancy == no) ∧ (Outlier == no) ∧ (Heterogeneous objective Clustering (Imbalanced Data == no) ∧ (High Data Dimension == yes) s Data == no) ∧ (Function == non-linear))then Multi	

In the case, dataset has both redundant values and outliers, and all other parameters are same as in rule 3, then probabilistic neural network is suited for prediction (rule 5) [43].

In rule 8 dataset has missing values and high dimension data. Noise is not present in dataset. Learning function can be linear or non-linear and dependent variable is of categorical type. Amount of training data requires is small and data redundancy is not there. In this case, kernel based classifiers are best suited for prediction [8,49]. If data redundancy is present and learning function is non-linear, but all other parameters are same as in rule 8, then according to rule 13, instance based classifiers are suited for prediction [45,56]. When dataset has noise, high data dimensionality, heterogeneous data, and linear learning function, then according to rule 11, bayesian based classifiers are best suited for prediction [53,54].

In Rule 9 noise is present in the dataset. Dataset has missing values and high dimensional data. Amount of training data requires is moderate, learning function is linear and dependent variable is continuous. Here, linear regression model is best suited for prediction [51]. If learning function is non-linear and dependent variable is categorical and all other parameters are same as in rule 9, the logistic regression is best suited for prediction (rule 10) [50,52].

Rule 12 is for ensemble based classifiers. It is a most powerful classifier. Dataset has noise, imbalanced data points, missing values, outlier, heterogeneous and high dimensional data, the dependent variable is continuous type. Amount of training data requires is large. In such case, ensemble based classifiers are best suited for prediction [10,14,25,55].

Rule 14 shows that no training data is available for building the prediction model. Dataset has noise, missing value, and high dimensionality, but outlier values and imbalanced data points are not present in the dataset. In such scenario, clustering based unsupervised techniques are best suited for prediction [26,57,59]. If dataset has heterogeneity and all other values are same as in rule 14. Multi-objective clustering is best suited for prediction (rule 15) [58,63].

4.2.2.2 Developing decision tree The decision rules designed and discussed in Sect. 4.2.2.1 are further used to develop the decision tree. We used J48 classifier to design and develop our recommendation system. J48 classifier is an open source Java implementation of the C4.5 algorithm in the weka data-mining tool [64]. C4.5 is a decision tree based algorithm and is an extension of ID3 algorithm [22]. It uses a tree based non-parametric supervised learning method for classification. It creates a tree type of structure that predicts the value of a target variable, by learning simple decision rules inferred from the data features (attributes). The feature that best splits the training data would be the root node of the tree. The same process is then repeated for each partition of the divided data, creating sub-trees until the training data is not classified into one of the given class [22]. To select a feature that best classify the given examples, generally some statistical methods are used. The most commonly used method is based upon the information gain. It is used to measure how well a feature is separating the given set of examples. It is defined as follows [65],

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \epsilon \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \tag{1}$$

Where, Values(A) is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v (i.e., $S_v = \{s \epsilon S | A(s) = v\}$). S is the set of all training examples.

Entropy is calculated as,

$$\text{Entropy}(S) = \sum_{n=1}^{c} -p_i \log_2 p_i \tag{2}$$

where p_i is the proportion of S belonging to class i.

At each step, it searches for complete hypothesis space and calculate the information gain of all the remaining features to select a candidate feature while growing the tree.

Figure 3 shows the generated decision tree. The mapping of abbreviations used in the decision tree to corresponding names is given in Table 3. In the generated decision tree, ATDR (amount of training data required) is serving as the root node. The input set of examples is classified into four classes based on the four values of ATDR. Class is further classified into subclasses in the similar way. The path from root node to leaf node shows a branch of the tree. The value attached to a leaf node represents the classification performance of the particular branch. For example, value (50.0/4.0) reveals that this branch of the tree classifies four instances incorrectly out of 50 seen instances.

To support the concept and architecture of the presented recommendation system, we have developed a prototype of the presented recommendation system. The prototype has been implemented using the Java programming language. The system asks questions to the users regarding the parameter values of fault data characteristics in a GUI interface. User can select any one of the possible value and then system automatically suggests the best-suited fault prediction technique for building fault prediction model.

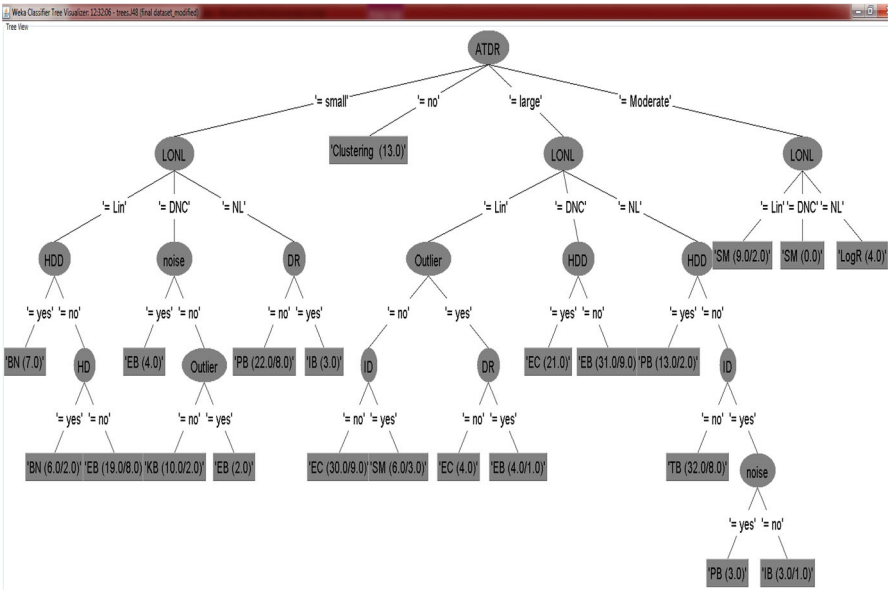

Fig. 3 Generated decision tree for techniques selection

Table 3 Mapping of the abbreviated names

Characteristics name	Techniques name
ATDR = amount of training data required	TB = tree based classifiers
LONL = linear or non-linear	PB = perceptron based classifiers
HDD = high data dimension	IB = instance based classifiers
DH = heterogeneous data	EC = ensemble classifiers
DR = data redundancy	EB = evolutionary based classifiers
DV = dependent variable	KB = kernel based classifiers
ID = imbalanced data	BN = Bayesian Network
MV = missing value	LogR = logistic regression
NL = non-linear	SM = statistical methods
DNC = dont care	

5 Experimental setup

The goal of this experimental study is to evaluate the performance of the presented decision tree logic based recommendation system. The details of the experiments are given in the coming subsections.

Table 4 Detail of the dataset

Dataset information	
Number of examples	246
Number of features	10
Number of classes in dependent variable	11
Type of dependent variable	Categorical

5.1 Detail of the dataset

To evaluate the presented recommendation system, we have created experimental dataset based on the information given in Table 1. The base of the experimental dataset lies in the thirty-three papers (referred in the Table 1) used for the formation of the rules for the selection of fault prediction techniques. Based on the results reported in these papers, the suitability of the different fault prediction techniques for the different values of fault datasets characteristics has been established. The dependent variable in our analysis is the fault prediction techniques (Sect. 3). The independent variables (features) are the ten characteristics of the fault datasets that we have considered in our study. The detail of the experimental dataset is given in Table 4.

The descriptive statistic of the dataset is given in Table 5. Table 5a is showing the frequency distribution of independent variables. Table presents possible class values with their occurrences for each of the characteristic considered for the fault dataset. First column of the table shows the possible class values for each characteristic. Second column shows the frequency of occurrence of each characteristic in the experimental dataset. Last column shows the percentage of the dataset instances in which the corresponding class value occurs. Table 5b is showing the frequency distribution of dependent variable. Table describes the frequency of occurrences of different fault prediction techniques considered in the study and the percentage of the dataset instances in which the corresponding techniques occurs.

5.2 Evaluation

In their work, Marcos et al. [62] have presented some measures to evaluate the rule-based systems. They have used majority error and error rate measures to evaluate the system. In another study, Mieczyslaw et al. [66] have used completeness, consistency, adequacy and reliability methods to validate the rule-based system. Encouraged from these above studies, we have used following evaluation parameters, as given in Table 6:

We have performed two sets of experiments to evaluate the performance and effectiveness of the recommendation system. They are given below:

Experiment 1 To measure the error rates and overall accuracy, using 10-fold cross validation approach.

In the first set of experiments, we use 10-fold cross validation scheme and evaluate the model. Cross validation scheme randomly divides the data into the ten parts. Nine

Table 5 Frequency distribution tables for the experimental dataset

Class values	Frequency	Percent ratio	Class values	Frequency	Percent ratio
(a) Table for independent variables					
Noise			*Missing value*		
No	135	54.9	No	127	51.6
Yes	111	45.1	Yes	119	48.4
Imbalanced data			*High data dimensional*		
No	155	63.0	No	176	71.5
Yes	91	37.0	Yes	70	28.5
Data redundancy			*Outlier*		
No	171	69.5	No	181	73.6
Yes	75	30.5	Yes	65	26.4
Amount of training data required			*Linear or non-linear*		
Large	147	59.8	DNC	74	30.1
Moderate	13	5.3	Lin	91	37.0
No	13	5.3	NL	81	32.9
Small	73	29.7			
Heterogeneous data			*Dependent variable*		
No	165	67.1	No	153	62.2
Yes	81	32.9	Yes	93	37.8

Techniques	Frequency	Percent ratio
(b) Table for dependent variable		
BN	13	5.3
Clustering	13	5.3
EB	50	20.3
EC	50	20.3
IB	16	6.5
KB	14	5.7
LogR	8	3.2
LR	11	4.48
PB	33	13.4
SM	14	5.7
TB	24	9.8
Total	246	100

parts are used to train the model and rest one part is used for testing the model. This process is repeated for 10 times and then results are averaged over the rounds.

Experiment 2 To measure the error rates and overall accuracy, using separate training and testing data.

Table 6 Performance measures

Performance measures	Definition
MAE and RMSE	MAE and RMSE measure the magnitude of the error in a set of prediction [67]
Accuracy	(TN + TP)/(TN + TP + FN + FP) [1]
Precision	TP/(TP + FP) [1]
Recall	TP/(TP + FN) [1]
False positive rate	FP/(TN + FP) [1]
F-measure	2 (precision * recall)/(precision + recall) [1]
ROC	It visualizes a trade-off between the correctly predicted faulty modules to the incorrectly predicted non-faulty modules [16]

TP true positive, *TN* true negative, *FP* false positive, *FN* false negative

Table 7 Results of the experiment

Parameters	10-fold cross validation	Separate testing data
Mean absolute error (MAE)	0.0393	0.0464
Root mean squared error (RMSE)	0.1555	0.164
Accuracy	84.84 %	84.20 %
False positive rate	2.5 %	2.8 %
Precision	83.4 %	81 %
Recall	84.8 %	84.3 %
F-measure	84 %	82.40 %
ROC value	95.40 %	95.90 %

In the second set of experiment, we use separate training and testing scheme. We randomly partition data in 30–70 ratios. 70 % of the data is used for training the model and remaining 30 % of the data is used for testing purpose. We use *RemovePercentage* filter available in weka data mining tool to partition the training and testing data.

The results of the above said experiments are given in Table 7. Table shows the results of the various evaluation parameters discussed above. For 10-fold cross validation, the accuracy value is 84.8 % with the 84 and 95.4 % of f-measure and ROC values respectively. The MAE value is 0.039, RMSE value is 0.155 and the false positive rate is 2.5 %. This shows that generated recommendation system has achieved high accuracy with lower misclassification errors. In Addition, higher value of ROC curve proving the effectiveness of the recommendation system. In second set of experiments, the proposed recommendation system has achieved an accuracy of 84.2 %, f-measure of 82.4 and 95.9 % of ROC curve value. The error rate is 0.0464 (MAE) and 0.164 (RMSE). The false positive rate is 2.8 %. The results of separate training and testing data scheme are comparable to the 10-fold cross validation results. These resulting statistics confirmed the effectiveness and reliability of the proposed recommendation system.

Table 8 Case studies

Case study	Reported works	Aim of study	Used datasets	Techniques used	Considered fault data characteristics	Techniques recommended by experimental study	Technique recommended by our proposed recommendation system
1.	Catal et al. [11]	Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction	KC1, KC2, PC1, CM1 and JM1	Random Forest, Naive Bayes, Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, and AIRS2	Noisy data, two training datasets are large and three are of small size, high dimension data, binary dependent variable and imbalanced data	JM1 and KC1: Random Forest (an Ensemble based classifier), PC1, CM1 and KC2: Naive Bayes (Bayesian base classifier)	For JM1 And KC1: Ensemble based classifiers or perceptron based classifiers. PC1, CM1 and KC2: Bayesian Network or evolutionary based classifiers
2.	Fei Xing et al. [49]	Investigate the effectiveness of SVM classifier for fault proneness	Medical imaging system	SVM, discriminant analysis, classification tree, Bayesian Network	Small amount of training data, non-liner learning function, binary dependent variable, no high dimensional data	SVM (kernel based classifier)	Kernel based classifiers or evolutionary based classifiers
3.	Arisho lm et al. [68]	Comparison of many machine learning techniques	A large Java legacy system	C 4.5, PART, logistic regression, RIPPER, back propagation NN and SVM	Large amount of training data, binary dependent variable, Imbalanced data	C 4.5 (tree based classifier) and Neural Network (perceptron based classifier)	Tree Based Classifier and Perceptron Based Classifier
4.	Elish et al. [69]	Defect prediction using support sector machine	CM1, PC1, KC1 and KC3	SVM, logistic regression, KNN, Multi-layer perceptrons, Bayesian Belief Network, Naive Bayes, Random Forest and decision tree	Small amount of training data, binary dependent variable, imbalanced data, non-linear learning function	SVM (kernel based classifier)	Kernel based classifiers

Table 8 continued

Case study	Reported works	Aim of study	Used datasets	Techniques used	Considered fault data characteristics	Techniques recommended by experimental study	Technique recommended by our proposed recommendation system
5.	Khoshgoftaar et al. [67]	Fault prediction using tree based approaches	A telecommunication system	Cart-ls,S-plus, and Cart-lad	Large amount of training data, Outlier, binary dependent variable, Imbalanced data	Cart-LS and S-plus (Tree based classifiers)	Tree Based Classifiers or Perceptron Based classifier or Ensemble Based classifiers
6.	Stefan et al. [15]	Benchmarking classification models for software defect prediction	CM1, KC1, KC3, KC4, MW1, JM1, PC1, PC2, PC3 and PC4	22 classification algorithms grouped into 5 categories: statistical classifiers, Nearest Neighbor, Support Vector Machine, tree-based classifiers and Ensemble classifiers	Noisy data, five datasets of large size, one dataset of moderate size and two datasets of small size, high dimensional data, binary dependent variable and imbalanced data	For KC1: Statistical Classifiers, Ensemble Classifiers. For PC1, PC2: Neural Network For MW1: statistical and Ensemble. For JM1: Ensemble classifier. For PC3: SVM. For PC4: Ensemble classifier	For KC1: Ensemble Classifiers. For JM1: Ensemble Classifier or evolutionary classifier. For PC1: statistical. For PC2: perceptron based or Ensemble classifier. For PC3: statistical or Ensemble classifier . For PC4: statistical or Ensemble classifier For MW1: perceptron or instance based classifiers

Table 8 continued

Case study	Reported works	Aim of study	Used datasets	Techniques used	Considered fault data characteristics	Techniques recommended by experimental study	Technique recommended by our proposed recommendation system
7.	Ma et al. [61]	Providing a statistical framework for the prediction of fault-proneness	KC1, KC2, JM1, PC1, CM1	Logistic regression, discriminant analysis, classification tree, boosting, J48, IBK, Naïve Bayes, RuleSet, VF1 and Random Forest, Kernel Density, Voted Perceptron, Hyper Pipes, Decision Stump, Kstar, Rocky	Noisy, two datasets of large size, one of moderate size and one of small size, high dimensional data, binary dependent variable and imbalanced data	JM1, PC1, KC1, KC2: Random Forest (Ensemble based classifier). CM1: logistic regression (statistical method)	JM1: Ensemble classifier or Evolutionary Classifier. CM1: perceptron classifier. PC1: statistical method. KC1 and KC2: Ensemble classifier

6 Case study

In this section, we present some case studies to establish the applicability of the proposed recommendation system. We have applied the proposed recommendation system on the datasets used in some recent experimental studies related to the comparison of fault prediction techniques and compared the recommendations of the proposed recommendation system with the results from the corresponding experimental studies. We have performed extensive study and considered only those studies that are comparing more than one fault prediction techniques. Studies considering only one technique for analysis were not considered. We observed that some of the information about the datasets has not been explicitly reported in the paper by the authors. So, we have consulted the corresponding sources and gathered the information about the dataset characteristics from there. Since, some of the authors used commercial or proprietary datasets; therefore, it is not possible to scrutinize those datasets. Hence, for these types of datasets, we have considered only those characteristics that were listed explicitly by the authors. These characteristics were used to get recommendations regarding the most appropriate fault prediction technique using the developed decision tree. The details of all the case studies are available in the form of supplementary material. The summarized results of the case studies are given in Table 8. Table 8 summarizes the experimental studies comparing various fault prediction techniques on some given datasets and shows the recommendations made by these works after experimental analysis regarding software fault prediction technique to use for a given dataset. We have used the same datasets for making recommendations as used by these works in their experimental investigations and have used these recommendations as the baseline results to validate the recommendations suggested by our presented system.

For each of the case study, the inputs to the recommendation system are the values of the dataset characteristics. Next, recommendation system processes the input values based on the information stored in the generated decision tree. Afterwards, it suggests the best possible fault prediction technique for the given dataset.

- **Case Study 1 (Catal et al.** [11]): In this work, the results of the used fault prediction techniques have been evaluated using AUC and accuracy parameters. The Random Forest technique performed best for with AUC values between 0.79 and 0.84 and accuracy value between 85 and 93 %, respectively. While, the Immunos2 technique performed worst with AUC values between 0.50 and 0.70 and accuracy value between 51 and 72 %, respectively. This study found that with the datasets that are imbalanced, have noise, are with the dependent variable of binary type, have high dimensional data, and are large in size, the Random Forest classifier produced the best results. Our recommendation system also suggests the use of Ensemble based classifiers or Perceptron based classifiers in these cases. For the small dataset having noise, with binary dependent variable, high data dimensions and imbalanced data, Naive Bayes classifier produced the best results. Our recommendation system also suggests the use of Bayesian based classifiers or Evaluation classifiers. The recommendations are in line with the results found by this reported study.

- **Case Study 2 (Fei et al.** [49]): In this work, the results of the fault prediction techniques have been evaluated using type-I and type-II error rate measures. The SVM technique performed best with type-I error of 2.3 % and type-II error of 6.4 %, respectively. Classification Tree technique performed worst with the type-I error of 9.5 % and type-II error of 8.8 %, respectively. The results of the study found that SVM classifier based fault prediction model produced the best result. Our recommendation system also suggests the use of kernel based classifier (SVM) or Evaluation based classifier in theses case, which is in alignment to the results reported by this study.
- **Case Study 3 (Erik et al.** [68]): In this work, the results of the experiments have been evaluated using accuracy, precision, and recall measures. The C4.5 classifier performed best with the accuracy values between 86 and 97 %, precision values between 0.07 and 0.42, and recall values between 0.22 and 0.83, respectively. Whereas, the PART classifier performed worst with the accuracy values between 77 and 96 %, precision values between 0.04 and 0.148, and recall values between 0.505 and 0.775, respectively. This study found that for the dataset of large size, with dependent variable of binary type and balanced data type, C 4.5 classifier produced best result. Our recommendation system also suggests the use of Tree based classifier in this case. For large size dataset with binary dependent variable and imbalanced fault data, the Neural Network produced the best results. Our recommendation system also suggests the use of Perceptron based classifier for this case.
- **Case Study 4 (Elish et al.** [69]): In this work, the results of the experiments have been evaluated using accuracy, precision, recall and f-measure parameters. The SVM classifier performed best with the accuracy value of 90 %, precision value of 0.94, recall value of 0.99, and f-measure value of 0.95, respectively. While, the Bayesian Network performed worst with the accuracy value of 83 %, precision value of 0.90, recall value of 0.92, and f-measure value of 0.85, respectively. The results of this study indicated that the performance of SVM classifier is better than other considered fault prediction techniques for all the used datasets. Our recommendation system also recommends the use of similar type of classifier for building fault prediction model.
- **Case Study 5 (Khoshgoftaar et al.** [67]): In this work, average absolute error (AAE) and average relative error (ARE) have been used to evaluate the performance of the fault prediction techniques. Cart-lad technique performed best with the AAE value of 1.13 and ARE value of 0.39, respectively. Whereas, Cart-ls performed worst with the AAE value of 1.28 and ARE value of 0.68, respectively. The results of the study found that Cart-ls based fault prediction models produced the best results. Our recommendation system also suggests the use of Tree based classifier or Perceptron based classifier in these cases. The recommendations are in line with the results found in this study.
- **Case Study 6 (Stefan et al.** [15]): In this work, AUC measure is used to evaluate the results of fault prediction. The results found that Random Forest performed best with the AUC value of 0.97 and Radical Basis Function Network performed worst with the AUC value of 0.60, respectively over various datasets. The results found that statistical techniques performed best for KC1 and MW1 data. Neural

network performed best for PC1 and PC2. SVM performed best for KC3, CM1 and PC3 and Ensemble classifiers performed best for KC1, KC4, MW1, PC4 and JM1 data. The fault prediction techniques suggested by our recommendation system are also in line with the results reported by this study.

- **Case Study 7 (Yan Ma et al.** [61]): In this work, the results of the considered techniques have been evaluated using several confusion matrix parameters. The Random Forest performed best with the accuracy value of 85 %, precision value of 0.40, recall value of 0.50 and the f-measure value of 0.48, respectively. While, the Voted Perceptron performed worst with the accuracy of 56 %, precision value of 0.24, recall value of 0.33 and, f-measure value of 0.24, respectively. The results of this study found that Random forest has performed better compared to all other used techniques for almost all the used datasets, except for CM1 data, where Logistic Regression produced the best results. Our recommendation system also suggests the use of ensemble based and statistical classifiers for most of the datasets, which is in line with the results found in this study.

From Table 8, it is clear that the recommendations made by our proposed recommendation system regarding the selection of fault prediction techniques are in line with the results reported by the authors. This confirmed the effectiveness of our proposed recommendation system. We are able to recommend the best-suited fault prediction technique accurately without performing any experimental study. From the analysis of these works, we found that using best fault prediction techniques increases the values of precision and recall by 20–25 % (approx.) compared to that of using any other relatively poorly performing fault prediction techniques.

From these case studies, it was observed that for the large datasets such as JM1, PC1, and KC1ensemble-based classifiers performed relatively well. The reason of this is that ensemble methods generally require large and diverse dataset to train themselves [9]. For the small datasets such as PC1, Naive Bayes performed relatively well. The reason of this is that Naive Bayes assumes that the value of a particular feature is independent of the value of any other feature, given the class variable and it requires a small amount of training data to estimate the parameters necessary for classification [17]. It is also observed from these case studies that all the classifiers handled missing values from the dataset efficiently.

7 Comparative evaluation

In this section, we present a comparative evaluation of the proposed recommendation system. To the best of our knowledge, no work in the literature has been reported on the development of recommendation system for the selection of software fault prediction technique in a given environment. Some of the works such as these [11,12,15,61,70–72], who have presented an experimental analysis of various fault prediction techniques over several software fault datasets and some concluding remarks on the suitability of fault prediction techniques are available. Table 9 presents the comparative analysis of the proposed recommendation system with these works.

Catal et al. [11] performed a study to investigate the effect of dataset size, metric set, and the feature selection techniques on software fault prediction. Additionally, they

Table 9 Summary of the comparative analysis

Factors considered	[11]	[15]	[12]	[61]	[70]	[71]	[72]	Proposed recommendation system
Noise	Yes	No	No	No	Yes	No	Yes	Yes
Missing Values	No	No	No	Yes	Yes	No	No	Yes
Outlier	Yes	No	Yes	No	No	No	No	Yes
HDD	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Heterogeneity of Data	No	No	Yes	No	No	No	Yes	Yes
ATDR	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Data Redundancy	No	No	Yes	No	No	Yes	Yes	Yes
Imbalanced Data	Yes	Yes	Yes	Yes	Yes	No	No	Yes
Learning Function	No	No	No	No	No	Yes	Yes	Yes
Dependent Variable	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Classification of fault prediction techniques	No	Yes	No	No	Yes, but limited	Yes	Yes, but limited	Yes
Providing suggestions for the selection of fault prediction techniques	Yes, but limited	Yes, but limited	Yes	Yes, but limited	Yes, but limited	Yes, but limited	Yes, but limited	Yes
Formal rules for effective decision making	No	No	No	No	No	No	No	Yes
Design of recommendation system	No	No	No	No	No	No	No	Yes

have evaluated nine classification algorithms for software fault prediction over five NASA datasets using various software metrics. The study considered only a limited number of fault dataset characteristics. Some important characteristics such as missing values, heterogeneity in the dataset, etc. have not been considered. Further, no discussion on the development of formal rules for the selection of software fault prediction technique can be found. Lessmann et al. [15] presented a framework for comparative analysis of several software fault prediction techniques. The comparative study has been performed over ten NASA datasets using twenty-two classifiers. Further, some suggestions regarding the selection of fault prediction techniques have been provided. The presented framework considered only few characteristics of fault dataset such as size of dataset, presence of noise and imbalance dataset. Many important characteristics affecting the selection of fault prediction technique have not considered. Furthermore, no recommendation system has been provided. Venkata et al. [12] evaluated different fault prediction techniques over four different software fault datasets. This work discussed some of the fault dataset characteristics such as multicollinearity, number of features (software metrics) in the dataset, and size of fault dataset. However, no discussion about the development of formal rules or recommendation system for the selection of fault prediction techniques can be found. Ma et al. [61] proposed a framework for software fault prediction using modified random forest algorithm. This study also compared the performance of presented modified random forest algorithm with several other software fault prediction techniques. This study primarily focused on evaluating the performance of different fault prediction algorithms. A little discussion has been found regarding the fault dataset characteristics. However, no taxonomical classification of fault prediction techniques or any type of recommendation system has been provided.

Hall et al. [70] presented a review on fault prediction performance in software engineering. The study primarily focused on investigating the influence of various factors on software fault prediction performance. A little discussion has been provided regarding the selection of fault prediction techniques for the given fault dataset. However, no description on the formal rules for the selection of fault prediction techniques or any recommendation system has been presented. Shihab [71] provided the details of the various factors related to software fault prediction (SFP) and presented a review of the works on SFP. In addition, a classification of the fault prediction techniques is presented. The study provided a little discussion of fault dataset characteristics. However, no detail about the formal rules or recommendation system that can be used for the selection of software fault prediction technique can be found. Nam [72] presented a survey of the works related to software defect prediction (SDP). The work discussed some characteristics of fault dataset such as dependent variable, noise, high data dimensionality and others. Many other important characteristics that influenced the selection of fault prediction techniques have not been considered. Further, no recommendation system has been provided.

8 Conclusions

In this paper, we present a decision tree logic based recommendation system that assists in the selection of a suitable technique to build fault prediction model. Here, initially we have presented a classification of the fault prediction techniques. Next, we have identified ten characteristics of the fault dataset that mostly influences the performance of the classification techniques used in the fault prediction. Then, an extensive study and analysis was performed to determine the influence of the selected fault-dataset characteristics on the suitability of the various classification techniques for fault prediction. This study was used to devise the rules and further for the development of a decision tree that takes values of dataset characteristics as input and generates the recommendations on the suitable fault prediction classifiers to be used for the given dataset. We have presented some case studies to confirm the usability and the effectiveness of the proposed recommendation system. The work has been compared with the existing similar works and the performance of the presented recommendation system has been evaluated. The system can find good use among the researchers and practitioners for getting the recommendations regarding the selection of suitable fault prediction techniques. In support of the presented architecture, a prototype system has also been implemented.

Acknowledgements The authors would like to thank the editor of the journal and the anonymous reviewers for their valuable comments, guidance, and suggestions that have really improved the quality of the paper and have led to the paper in its current form. Further, we would like to thank the Ministry of Human Resource Development (MHRD), India for providing institute assistantship.

References

1. Sheskin DJ (2003) Handbook of parametric and nonparametric statistical procedures. CRC Press, Boca Raton
2. Zimmermann T, Nagappan N, Zeller A (2008) Predicting bugs from history. Softw Evol J. Springer, Berlin, pp 69–88
3. Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A (2010) Defect prediction from static code features: current results, limitations, new approaches. Autom Softw Eng J
4. Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689
5. Briand LC, Daly JW, Wust J (1998) A unified framework for cohesion measurement in object-oriented systems. Empir Softw Eng 3(1):65–117
6. Alshayeb M, Li W (2003) An empirical validation of object-oriented metrics in two different iterative software processes. IEEE Trans Softw Eng 29(11):1043–1049
7. Li W, Henry S (1993) Object-oriented metrics that predict maintainability. J Syst Softw 23(2):111–122
8. Xing F, Guo P, Lyu MR (2005) A novel method for early software quality prediction based on support vector machine. In: Proceeding of 16th IEEE international symposium on software reliability engineering, pp 10–19
9. Khoshgoftaar TM, Ganesan K, Allen EB, Ross FD, Munikoti R, Goel N, Nandi A (1997) Predicting fault-prone modules with case-based reasoning. In: Proceedings of 8th international symposium on software reliability engineering, pp 27–35
10. Guo L, Ma Y, Cukic B, Singh H (2004) Robust prediction of fault-proneness by random forests. In: Proceeding of 15th international symposium on software reliability engineering, pp 417–428
11. Catal C, Diri B (2009) Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Inf Sci J 179(8):1040–1058

12. Challagulla UV, Bastani FB, Yen IL (2006) A unified framework for defect data analysis using the mbr technique. In: Proceeding of 18th IEEE international conference on tools with artificial intelligence, pp 39–46
13. Jiang Y, Cukic B, Ma Y (2008) Techniques for evaluating fault prediction models. Empir Softw Eng 13(5):561–595
14. Sun Z, Song Q, Zhu X (2012) Using coding-based ensemble learning to improve software defect prediction. IEEE Trans Syst Man Cybern Part C Appl Rev 42(6):1806–1817
15. Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans Softw Eng 34(4):485–496
16. Vandecruys O, Martens D, Baesens B, Mues C, De Backer M, Haesen R (2008) Mining software repositories for comprehensible software fault prediction models. J Syst Softw 81(5):823–839
17. Dejaeger K, Verbraken T, Baesens B (2013) Toward comprehensible software fault prediction models using bayesian network classifiers. IEEE Trans Softw Eng 39(2):237–257
18. Kanmani S, Uthariaraj VR, Sankaranarayanan V, Thambidurai P (2007) Object-oriented software fault prediction using neural networks. Inf Softw Technol 49(5):483–492
19. Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th joint meeting of the ESEC and FSE, pp 91–100
20. Pickard L, Kitchenham B, Linkman S (1999) An investigation of analysis techniques for software datasets. In: Proceedings of 6th international software metrics symposium, pp 130–142
21. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Burlington
22. Martinez J, Fuentes O (2005) Using c4.5 as variable selection criterion in classification tasks. In: Proceedings of the 9th international conference on artificial intelligence and soft computings. Benidrom, Spain
23. Kotsiantis SB (2007) Supervised machine learning: a review of classification techniques. In: Proceedings of emerging artificial intelligence applications in computer engineering, pp 3–24
24. Fitzpatrick JM, Grefenstette JJ (1988) Genetic algorithms in noisy environments. Mach Learn 3(2–3):101–120
25. Rokach L (2005) Ensemble methods for classifiers. In: Data mining and knowledge discovery handbook. Springer, Berlin, pp 957–980
26. Xuan L, Zhigang C, Fan Y (2013) Exploring of clustering algorithm on class-imbalanced data. In: Proceeding of 8th international conference on computer science and education. IEEE, New York, pp 89–93
27. Manago M, Kodratoff Y (1987) Noise and knowledge acquisition. In: IJCAI, pp 348–354
28. Gao K, Khoshgoftaar TM, Wang H, Seliya N (2011) Choosing software metrics for defect prediction: an investigation on feature selection techniques. Softw Pract Exp 41(5):579–606
29. Rodriguez D, Ruiz R, Cuadrado-Gallego J, Aguilar-Ruiz J, Garre M (2007) Attribute selection in software engineering datasets for detecting fault modules. In: Proceedings of 33rd EUROMICRO conference on software engineering and advanced applications, pp 418–423
30. Graves TL, Karr AF, Marron JS, Siy H (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661
31. Charu C (2013) Aggarwal. Outlier analysis. Springer Science and Business Media, Berlin
32. Moreno-Torres JG, Raeder T, Alaiz-Rodriguez R, Chawla NV, Herrera F (2012) A unifying view on dataset shift in classification. Pattern Recognit 45(1):521–530
33. Calikli G, Bener A (2013) An algorithmic approach to missing data problem in modeling human aspects in software development. In: Proceedings of 9th international conference on predictive models in software engineering. ACM, New York
34. Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalanced data. In: Proceeding of international conference on software engineering
35. Kim S, Zhang H, Wu R, Gong L (2011) Dealing with noise in defect prediction. In: 33rd international conference on software engineering, pp 481–490
36. Grbac T, Mausa G, Basic BD (2013) Stability of software defect prediction in relation to levels of data imbalance. In: SQAMIA, pp 1–10
37. Vu B, Challagulla FB, Bastani IL, Paul RA (2008) Empirical assessment of machine learning based software defect prediction techniques. Int J Artif Intell Tools 17(02):389–400

38. Succi G, Pedrycz W, Djokic S, Zuliani P, Russo B (2005) An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. Empir Softw Eng 10(1):81–104
39. Fayyad UM, Irani KB (1992) On the handling of continuous-valued attributes in decision tree generation. Mach Learn 8(1):87–102
40. Murphey YL, Guo H, Feldkamp LA (2004) Neural learning from unbalanced data. Appl Intell 21(2):117–128
41. Smith MR, Martinez T (2011) Improving classification accuracy by identifying and removing instances that should be misclassified. In: Proceeding of 2011 international joint conference on neural networks, pp 2690–2697
42. Sharpe PK, Solly RJ (1995) Dealing with missing values in neural network-based diagnostic systems. Neural Comput Appl 3(2):73–77
43. Venkatesh S, Gopal S (2011) Robust heteroscedastic probabilistic neural network for multiple source partial discharge pattern recognition-significance of outliers on classification capability. Exp Syst Appl 38(9):11501–11514
44. Haupt RL, Haupt SE (2004) Practical genetic algorithms. Wiley, New York
45. Allison PD (2001) Missing data, vol 136. Sage Publications, Chennai
46. Smith SF (1980) A learning system based on genetic adaptive algorithms. PhD thesis
47. Afzal W, Torkar R, Feldt R (2008) Prediction of fault count data using genetic programming. In: Proceeding of international multitopic conference, pp 349–356
48. Fonseca CM, Fleming PJ (1993) Multiobjective genetic algorithms. In: IEE colloquium on genetic algorithms for control systems engineering. IET, Thiruvananthapuram, pp 1–6
49. Li F, Li H (2012) Svm classification for large data sets by support vector estimating and selecting. In: Recent advances in computer science and information engineering. Springer, Berlin, pp 775–781
50. Hastie T, Tibshirani R, Friedman J, Franklin J (2005) The elements of statistical learning: data mining, inference and prediction. Math Intell 27(2):83–85
51. Debruyne M (2009) An outlier map for support vector machine classification. Ann Appl Stat 1566–1580
52. Khoshgoftaar TM, Seliya N (2003) Fault prediction modeling for software quality estimation: comparing commonly used techniques. Empir Softw Eng 8(3):255–283
53. Mauvsa G, Grbac TG, Bavsic BD (2012) Multivariate logistic regression prediction of fault-proneness in software modules. In: Proceedings of the 35th international convention, pp 698–703
54. Ratanamahatana CA, Gunopulos D (2002) Scaling up the naive bayesian classifier: using decision trees for feature selection
55. Briand L, Devanbu P, Melo W (1997) An investigation into coupling measures for c++. In: Proceedings of 19th international conference on software engineering, pp 412–421
56. Ghimire B, Rogan J, Galiano VR, Panday P, Neeti N (2012) An evaluation of bagging, boosting, and random forests for land-cover classification in cape cod, massachusetts, usa. GISci Remote Sens 49(5):623–643
57. Hinneburg A, Aggarwal CC, Keim DA (2000) What is the nearest neighbor in high dimensional spaces? In: Proceedings of the 26th international conference on very large data bases, VLDB '00, pp 506–515
58. Jiamthapthaksin R, Eick CF, Vilalta R (2009) A framework for multi-objective clustering and its application to colocation mining. In: Advanced data mining and applications. Springer, Berlin, pp 188–199
59. Acuna E, Rodriguez C (2004) The treatment of missing values and its effect on classifier accuracy. In: Classification, clustering, and data mining applications. Springer, Berlin, pp 639–647
60. Amatriain X, Jaimes A, Oliver N, Pujol JM (2011) Data mining methods for recommender systems. In: Recommender systems handbook. Springer, Berlin, pp 39–71
61. Ma Y, Guo L, Cukic B (2006) A statistical framework for the prediction of fault-proneness. In: Advances in machine learning application in software engineering. Idea Group Inc, Calgary, pp 237–265
62. Karimi K, Hamilton HJ (2002) Timesleuth: a tool for discovering causal and temporal rules. In: Proceedings of 14th IEEE international conference on tools with artificial intelligence. IEEE, New York, pp 375–380
63. Liu H, Yu L (2005) Toward integrating feature selection algorithms for classification and clustering. IEEE Trans Knowl Data Eng 17(4):491–502
64. Law HCM, Topchy AP, Jain AK (2004) Multiobjective data clustering. In: Proceedings of IEEE computer society conference on computer vision and pattern recognition, vol 2, pp II-424
65. Mitchell TM (1997) Machine learning, vol 1. McGraw-Hill, USA

66. Owoc ML, Galant V (1999) Validation of rule-based systems generated by classification algorithms. In: Evolution and challenges in system development. Springer, Berlin, pp 459–467
67. Khoshgoftaar TM, Seliya N (2002) Tree-based software quality estimation models for fault prediction. In: Proceedings of the eighth IEEE symposium on software metrics. IEEE, New York, pp 203–214
68. Arisholm E, Briand LC, Fuglerud M (2007) Data mining techniques for building fault-proneness models in telecom java software. In: Proceeding of 18th IEEE international symposium on software reliability, pp 215–224
69. Elish OK, Elish MO (2008) Predicting defect-prone software modules using support vector machines. J Syst Softw 81(5):649–660
70. Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic literature review on fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276–1304
71. Shihab E (2012) An exploration of challenges limiting pragmatic software defect prediction. PhD thesis, Queens University
72. Nam J (2014) Survey on software defect prediction. PhD Thesis, Hong Kong University of Science and Technology