Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Software defect prediction using ensemble learning on selected features

Issam H. Laradji, Mohammad Alshayeb *, Lahouari Ghouti

Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

ARTICLE INFO

Article history:
Received 4 February 2014
Received in revised form 22 May 2014
Accepted 8 July 2014
Available online 24 July 2014

Keywords:
Defect prediction
Ensemble learning
Software quality
Feature selection
Data imbalance
Feature redundancy/correlation

ABSTRACT

Context: Several issues hinder software defect data including redundancy, correlation, feature irrelevance and missing samples. It is also hard to ensure balanced distribution between data pertaining to defective and non-defective software. In most experimental cases, data related to the latter software class is dominantly present in the dataset.
Objective: The objectives of this paper are to demonstrate the positive effects of combining feature selection and ensemble learning on the performance of defect classification. Along with efficient feature selection, a new two-variant (with and without feature selection) ensemble learning algorithm is proposed to provide robustness to both data imbalance and feature redundancy.
Method: We carefully combine selected ensemble learning models with efficient feature selection to address these issues and mitigate their effects on the defect classification performance.
Results: Forward selection showed that only few features contribute to high area under the receiver-operating curve (AUC). On the tested datasets, greedy forward selection (GFS) method outperformed other feature selection techniques such as Pearson's correlation. This suggests that features are highly unstable. However, ensemble learners like random forests and the proposed algorithm, average probability ensemble (APE), are not as affected by poor features as in the case of weighted support vector machines (W-SVMs). Moreover, the APE model combined with greedy forward selection (enhanced APE) achieved AUC values of approximately 1.0 for the NASA datasets: PC2, PC4, and MC1.
Conclusion: This paper shows that features of a software dataset must be carefully selected for accurate classification of defective components. Furthermore, tackling the software data issues, mentioned above, with the proposed combined learning model resulted in remarkable classification performance paving the way for successful quality control.

1. Introduction

With the increasing impact of software applications on day-to-day businesses and activities, software attribute prediction such as effort estimation [1,2], maintainability [3,4], defect [5,6] and quality [7,8] classification are gaining growing interest from both academic and industry communities. Defective software components have devastating consequences on increased development and maintenance costs and declining customer satisfaction [9]. This safety and reliability fact calls for the adoption of rigorous software quality control processes. However, the scarcity of human and financial resources dictates the need for cost-efficient approaches to detect and repair defective components.

Research on software defect prediction emphasized the success of many algorithms including decision trees [10,11], Bayesian methods [12,13], and artificial neural networks multilayer perceptrons (ANN-MLPs) [14]. However, these methods are sub-optimal in the case of skewed and redundant defect datasets [15]. The prediction performance of these methods gets worse when the defect datasets contain incomplete or irrelevant features [16]. Classifiers such as support vector machines (SVMs) [17] and ANN-MLPs [18], biased towards the dominant class, tend to ignore the minority class which results in high false negative rates [19]. It is noteworthy that ensemble learning models are very adequate to address the data issues mentioned earlier. For instance, random forests [20] outperforms the aforementioned algorithms in detecting defective modules even though they are not tuned to directly address imbalanced data [21]. In addition, the voting mechanism in ensemble learning mitigates any residual effect attributed to feature irrelevance and redundancy. This mitigation is carried out

* Corresponding author.
E-mail addresses: g200790850@kfupm.edu.sa (I.H. Laradji), alshayeb@kfupm.edu.sa (M. Alshayeb), lahouari@kfupm.edu.sa (L. Ghouti).

http://dx.doi.org/10.1016/j.infsof.2014.07.005
0950-5849/© 2014 Elsevier B.V. All rights reserved.

by assigning higher weights to individual classifiers that perform well on the tested datasets. The robustness to irrelevant and redundant features certainly enhances the prediction performance. In fact, voting, in its simple form of averaging, ensures the mitigation of the noise effects, which boots the overall prediction performance.

In this paper, we propose a software defect classification method using an average probability ensemble (APE) learning module. The proposed APE system incorporates seven classifiers: random forests (RF), gradient boosting (GB), stochastic gradient descent (SGD), weighted SVMs (W-SVMs), logistic regression (LR), multinomial naive Bayes (MNB) and Bernoulli naive Bayes (BNB). The base classifiers have been selected after extensive simulation validation as indicated in the sequel of the paper. It is worth mentioning that some of these classifiers, such as the RF and W-SVMs models, are considered "de-facto" classifiers in the literature [22]. In addition, the variation in classification capabilities of the base classifiers enable them to capture different statistical characteristics of the underlying data, which makes their combination an added value for the proposed ensemble learning model.

To further improve the classification performance of the proposed ensemble classifier, efficient feature selection is combined with the proposed ensemble model yielding an enhanced ensemble classifier. This enhancement resulted in efficient handling of redundant and irrelevant features in software defect datasets.

Therefore, the objectives of the paper are to demonstrate the positive effect of feature selection on the performance of defect classification and to propose a two-variant ensemble learning algorithm which is robust to both data imbalance and feature redundancy. In addition, the proposed two-variant ensemble algorithm has exhibited stronger robustness to redundant and irrelevant features, which constitutes a major contribution attributed to this paper.

The paper is organized as follows: Section 2 summarizes the related work. Section 3 gives a detailed description of the proposed learning model for software defect prediction. In Section 4, we describe the software defect datasets, the experimental setup and results. Detailed analysis and discussion on the reported resulted are given in Section 5. In Section 6, we present the threats to validity and finally, conclusions, along with suggestions for future work, are given in Section 7.

2. Literature review

A detailed account of current related work is given below. First, general prediction techniques are reviewed followed by a summary of sampling techniques that are used to properly handle data imbalance. Then, an overview of commonly used cost-effective classification techniques is given. The use of ensemble learning models to remedy data imbalance is discussed afterwards. Finally, approaches for feature selection are outlined before the section concludes with a review of other defect classification methods.

2.1. General techniques

Decision trees [10,11], Bayesian methods [12], and ANNs [14] have paved the way for machine learning-based methods in the field of defect classification. These methods use software metrics to properly classify defective software modules. However, it should be noted that these methods often ignore the skewness and other statistical characteristics of the defect datasets. Omitting such characteristics substantially affects, in a negative way, the classification performance [19]. Conventional methods, such as SVMs [17] and Bayesian networks [12], generate models that tend to ignore the minority class (defective modules usually) [19]. For instance, for a given dataset with 0.5% defective components, a classification accuracy of 99.5% can be achieved by simply classifying all components as non-defective. However, the AUC measure of the receiver operating characteristics (ROC) curve is 0.5 indicating that the classifier is simply tossing the coin to classify the dataset.

2.2. Sampling techniques

Oversampling and under sampling are two well-known standard techniques commonly used to deal with imbalanced datasets where the majority class is highly over-represented compared to the minority class [21]. The former technique adds data duplicates or synthetic samples to the minority class and data samples are removed from the majority class in the latter sampling technique. Moreover, results reported by Seiffert et al. [23] clearly indicate that the use of data sampling techniques improves the classification performance in the case of software defect prediction applications. However, it was reported that ensemble learning models, combined with boosting, always outperform data sampling-based defect classifiers in terms of classification accuracy [23].

In their experimental evaluation, Seiffert et al. [23] assessed the performance of 50,000 classification models using 15 datasets with five different sampling algorithms. While the best data sampling technique yielded an AUC of 0.744, boosting-based ensemble learning models achieved an AUC measure of 0.798. Pelayo and Dick [24] investigated the use of two data stratification approaches: (1) under-sampling and (2) over-sampling methods for software defect prediction. Data pertaining to the minority class was over-sampled to generate synthetic samples based on the synthetic minority oversampling technique (SMOTE) method [25]. Fewer samples were selected from the majority class by random under-sampling. Their approach resulted in approximately 23% mean classification accuracy improvement. It is noteworthy that over-sampling and under-sampling may lead to over-fitting and removal of relevant samples, respectively.

2.3. Cost-sensitive methods

Although sampling techniques tend to balance the data distribution properly, misclassifying different software defect classes might have aggravated costs [9]. Errors occurring in data classification can be cast into two types, namely, "Type-I" and "Type-II" [26]. The former quantifies the misclassification rates of defect-free software components and the latter is concerned with the misclassification of defective ones. Needless to mention that Type-II errors are more costly and, therefore, should be carefully looked at. This consideration stems from the devastating costs that could result from accepting a defective software component as defect-free. Given these facts, Khoshgoftaar et al. [26] proposed a cost-sensitive boosting technique that combines boosting ensemble learning algorithm and cost-sensitivity feature. Cost-sensitivity allows for the incorporation of a cost matrix that measures the penalty that is incurred by misclassifying data samples. In addition, the cost-sensitivity is reflected on the weight update of the classifier, which takes place more aggressively when Type-II errors are detected. In this way, higher penalty costs are assigned to the misclassification of defective software components.

Using the C4.5 decision tree as a base classifier, Quinlan improved the classification performance compared to the original boosting technique [27]. Another study is attributed to Zheng [28] where three types of cost-sensitive prediction models were compared. In all three models, boosted ANN techniques were used consisting of 10 basic back-propagation ANN blocks with 11 hidden neurons each. Unlike fixed weight update schemes, the smallest expected cost of misclassification (ECM) was achieved using the threshold moving technique for the weight updates. The threshold moving approach was found more tolerant to cost estimation. Moreover, the cost matrix was provided as an estimate of the cost of Type-I and Type-II errors. In summary, one main issue arises in cost-sensitive classification concerning the proper definition of the cost matrix. In fact, defining the cost matrix requires expert domain knowledge especially in the case of life-critical applications.

2.4. Ensemble techniques

Ensemble learning techniques have been very successful in handling small-sized and imbalanced datasets although not specifically designed to address the data imbalance problem [29]. Successful applications of ensemble learning encompass a wide range of problems including prediction of functional roles in Bioinformatics [30], detection of concept [31], and detection of live tumor [32].

Wang et al. [33] proposed a new ensemble learning approach known as sampling based online bagging. In their empirical study, sampling based online bagging achieved balanced performance with positive and negative samples but unstable performance when the class distributions change over time. However, Wang and Minku proposed Undersampling-based online bagging that is robust against samples with changing class distributions.

Several solutions based on ensemble learning have been proposed to address the software defect classification problem. An early extension of the bagging ensemble learning algorithm, known as roughly balanced bagging (RBBAG) algorithm, was proposed by Seliya et al. [34]. Simulation results clearly indicated the effectiveness of the RBBAG method over individual classifiers. Higher classification performance, measured by the geometric mean (GM), was achieved by the RBBAG method given its ability to properly handle data imbalance present in the test datasets. Seliya et al. evaluated two different base classifiers empirically: (1) the naive Bayes and (2) the C4.5 decision tree. RBBAG model based on the former classifier outperformed that based on the latter classifier in terms of achieved GM measure. However, the performance of the RBBAG method was not evaluated against varying degrees of data skewness as suggested by Seliya et al.

The problem of data skewness in software defect classification was tackled by Sun et al. [35]. Sun et al. divided non-defective components into equal-sized bins where the bin size is set to the number of defective modules. Then, each bin is assigned to a new class label, which results into a multi-classification problem. Three different coding schemes (one-against-one, one-against-all, and random correction code) were evaluated under the new classification formulation. Each of these coding schemes was integrated with three different ensemble learning strategies including bagging, boosting, and random forests. Using the AUC performance measure, one-against-one coding scheme outperformed its counterparts. However, the proposed solutions do not address the problem of assigning data samples to the bins during the data balancing stage. Wang et al. provided a detailed review of ensemble learning applications in software defect classification. The models included bagging, boosting, random trees, random forests, random subspace, stacking, and voting [36]. To contrast the superior performance of these models, classification results attained by a single classifier based on a naive Bayes were reported. As expected, Wang et al. clearly demonstrated how the reviewed ensemble models outperformed their single classifier counterpart using several public-domain software defect datasets. In their comparative analysis, Wang et al. found reported a noticeable classification superiority attributed to random forests. Using other software datasets (CM1, KC1-KC4, MW1, JM1, PC1-PC4, AR), Lessmann et al. [37] investigated the classification performance of several ensemble learning models including random forests and logistic model tree [38]. Unlike Wang et al. [36], Lessmann et al. benchmarked these ensemble models against several base classifiers representing statistical, neural, clustering and SVM models. In most instances, the ensemble model based on random forests attained the highest classification performance in terms of AUC measure. The results reported in [36,37] reveal an interesting finding: Ensemble learning models based on random forests achieve the best classification performance regardless of the software defect dataset used.

Finally, a new two-variant ensemble learning model is proposed in this paper. The two variants of the proposed model are solely motivated by the promising results reported in [36,37]. However, rather than using random forests only, a combination of base classifiers is suggested. In addition, these base classifiers are selected given their proven performance provided proper parameter tuning is ensured [22].

2.5. Feature selection

Feature selection, an important preprocessing step in machine learning, has been widely used in a plethora of domains ranging from text classification [39] to bioinformatics and biological networks [40]. It is a well-known fact that different metrics might correlate with defect-proneness in software systems. Therefore, removing uncorrelated metrics could enhance classification performance.

Using BN networks, Okutan and Yildiz [41] selected informative metrics/features by ranking them based on their correlation degree to defect-proneness. These informative metrics include probabilistic influential relationships among software metrics and defect. Okutan and Yildiz adopted the number of developers (NOD) and lack of coding quality (LOCQ) as new metrics. Reported results suggested that response for class (RFC), lines of code (LOC) and LOCQ are the most effective metrics for detecting a defective software module. On the other hand, the coupling between objects (CBO), weighted method per class (WMC) and lack of cohesion of methods (LCOM) are less effective for defect classification. Furthermore, a positive correlation between the number of developers and defect probability was identified.

Khoshgoftaar et al. [16] examined seven filter-based feature ranking techniques for comparison using 16 software datasets. These features include chi-squared (CS), information gain (IG), gain ratio (GR), symmetrical uncertainty (SU) and ReliefF with two variants (RF and RFW). In their approach, Khoshgoftaar et al. explored the signal-to-noise ratio (SNR) which is rarely employed as software metric. The proposed ensemble learning architecture considered various basic classifiers including NB, ANN, K-nearest-neighbor (KNN), SVM, and LR models. Throughout the evaluation experiments, the AUC measure was considered as the performance criteria. The IG and SNR features yielded the best average defect classification performance across all investigated datasets. Classifiers using CS and SU features performed slightly worse than the classifiers using IG and SNR features. Given the reported results, Khoshgoftaar et al. recommended IG and SNR as basic features for software quality datasets. Although feature selection improves the defect classification performance, it should be kept in mind that it does neither address the imbalanced data problem nor does it consider the cost of misclassifying different classes.

2.6. Other techniques

The problem of software defect classification has also been tackled by approaches that are quite distinct from the techniques described above. Yu and Mishra [42] considered multiple software defect datasets to efficiently classify defected modules of the investigated software project. Simulation results reported a performance improvement by learning models trained using more than one project dataset while keeping the recall rate unaffected. Furthermore, Yu and Mishra noticed that the forward assessment technique yielded worse classification results than self-assessment although better overall classification accuracy was achieved using their combination.

An improved unsupervised learning technique for defect prediction was proposed by Bishnu and Bhattacherjee [43]. The proposed learning scheme, called quad tree-based K-means, is an unsupervised learning technique inspired by the standard K-means clustering algorithm. Classification performance, similar to that attained by supervised learning techniques, makes quad-trees very promising in addressing the software defect classification problem. However, these techniques are not specifically tailored to handle the skewness nature of software defect datasets, which still leaves room for possible improvements.

2.7. Data imbalance

Usually, the classification algorithms described above attain acceptable classification accuracy in the case of properly distributed data. The data distribution quality is measured using the class balance ratio defined below [21,44]:

$$r_k = \frac{\sum_{i=1}^{C_k} s_i^{(k)}}{\sum_{i=1}^{N} s_i} \tag{1}$$

where r_k is the imbalance ratio for class k ($k = 1, 2, \ldots, K$), C_k is the number of samples pertaining to class k and N is the total number of samples in the dataset. In this case, each class k has C_k samples. Data imbalance is remedied using data-level or algorithmic processing [21]. In the former processing, the original data samples are statistically redistributed using up-sampling or down-sampling depending on the imbalance ratios [21]. In the algorithmic approach, base classifiers are reformulated to take into account the imbalance ratio via weight assignment. Imbalanced datasets are often encountered in software engineering research given the difficulty of producing software metric data that represent well defective software in quality and quantity [45]. On the other hand, commonly used metrics such as accuracy and AUC fail to properly capture the classifier behavior in the presence of imbalanced data [21]. The geometric mean (G-mean) measure yields more accurate classification measures when the underlying data is impacted with imbalance [46]. In this paper, the G-mean measure is used to assess the performance of the proposed classifiers. A detailed account of this measure in given in Section 5.4.[1] To highlight the relevance of G-mean measure in imbalanced software datasets, a hypothetic example is considered. An overall accuracy of 99% can be attained by a binary classifier that classifies all data samples as majority class. This would wrongly indicate an excellent performance while it is not the case. However, the G-mean measure would score 0 to indicate that the classifier has performed extremely poorly with respect to the samples pertaining to the minority class.

3. Research approach

In this section, we describe the research objectives, hypothesis and methodology.

3.1. Objectives and hypothesis

The paper objectives are to demonstrate the positive effect of feature selection on the performance of defect classification and to propose an ensemble learning algorithm which is robust to imbalanced data and redundant features. Therefore, we propose a novel ensemble learning algorithm to address the issue of software defect classification problem using. The proposed algorithm is not only robust to imbalanced data but it also handles efficiently redundant features inherent in software quality datasets.

To achieve the above-mentioned objectives, we define the two following hypotheses:

Hypothesis 1. Selecting different feature subsets yields highly inconsistent performance in defect classification problems. This performance inconsistency is mainly attributed to the presence of poor and redundant features.

Hypothesis 2. In the presence of software defect datasets impaired with data imbalance and feature redundancy, the proposed ensemble learning, APE algorithm, outperforms existing single and ensemble classifiers in terms of classification performance.

3.2. Outline of experiment setup

To assess the classification performance of the proposed models and assert the validity of the paper, all computer simulations are performed using a stratified 10-fold cross-validation with proper data split (training and testing). A 10-fold cross-validation guards against biasedness towards testing hypotheses suggested by the data (i.e., Type III errors [22]) especially in the case of software defect datasets which are costly and time-consuming to collect. On the other hand, stratified sampling ensures that each fold contains roughly the same proportions of the defect-free and defective classes. In this way, the class imbalance ratio is properly maintained.

The following steps are adopted to assert the first hypothesis:

1. Split the dataset into training and testing subsets (90% and 10%, respectively).[2]
2. Define an empty feature list F_i*ist*.
3. Select the next best feature from the training set using the adopted feature selection criterion and append it to F_i*ist*.
4. Train the ensemble classifier on the training set with the selected features.
5. Predict defective components of the testing set using the trained classifier.
6. Save the performance result.
7. Repeat step 2 for the remaining features.
8. Report the performance results.

Moreover, the reported performance results would flag any inconsistency effect occurring during the feature selection process. Therefore, the first object of the paper will be achieved.

The evaluation of the second hypothesis is implemented through these steps:

1. Split the dataset into training and testing subsets (90% and 10%, respectively).[3]
2. Train the proposed APE model using the training set.
3. Evaluate the trained APE model using the testing set.
4. Store the classification performance as p_1.

[1] To our best knowledge, the use of G-mean measures for defect software datasets is first proposed in this paper.

[2] The same experimental setup is adopted where a stratified 10-fold cross-validation is performed throughout all computer simulations.

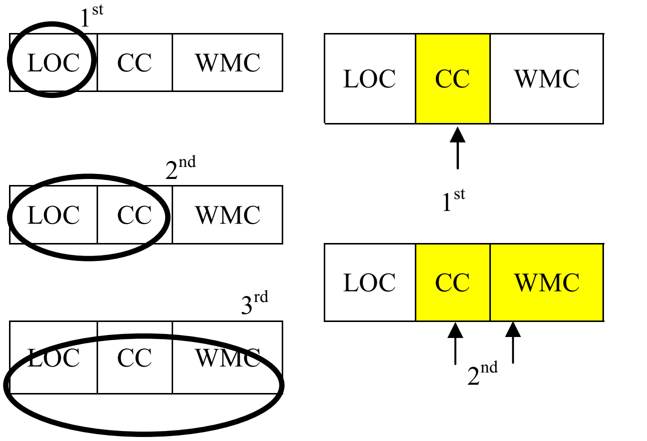
[3] As mentioned earlier, stratified 10-fold cross-validations guards against biasedness towards hypotheses suggested by the data and ensures maintaining proper class imbalance ratio across all folds.

5. Select the best features from the training set using the greedy-forward selection method.
6. Append the selected feature to the list F_i.
7. Re-train the APE model using the list F_i.
8. Repeat steps 5–7 using the testing set.
9. Store the classification performance as p_2.

The second objective of the paper is achieved when the APE model yields p_2 higher than p_1 and outperforms other classifiers. In this case, the APE model exhibits a behavior that is robust to both data imbalance and feature redundancy.

3.3. Feature subset selection

In software defect datasets, features represent software metrics extracted from the source code. However, some features are redundant and/or irrelevant which calls for the removal of the latter. In fact, it is expected that such preprocessing step will substantially improve the classification performance. Forward selection is commonly used technique to select good features. Fig. 1 illustrates the process of forward selection using a feature set consisting of cyclomatic complexity (CC), weighted methods per class (WMC) and LOC software metrics. Using an initially empty set, forward selection selects the first feature from the full feature set. In the case of Fig. 1a, the feature subset contains only the LOC metric at the first iteration. Then, defect classification is carried out and its performance is evaluated. Afterwards, a second feature is selected and the subset contains LOC and CC metrics. Using this augmented subset, the classification performance is evaluated as well. The same process is repeated gradually until all the features are considered. Finally, the feature subset achieving the highest accuracy is retained. However, it is obvious that forward selection becomes time-consuming when dealing with a large set of features. Efficient feature selection is highly desirable. Greedy forward selection (GFS) is an efficient, yet simple, feature selection technique. Unlike other time-consuming schemes (forward and backward selection [22]), the GFS selects only those features that contribute positively to the improvement classification performance. For instance, GFS selects first the feature that attains the highest classification performance. Then, the best feature given the previously selected features is appended to the retained feature subset. This process is demonstrated in Fig. 1b. On the other hand, correlation-based selection (CBS) technique obtains a list of features that are sorted based on their degree of correlation to the module class. The first selected feature is that is most class-discriminative. The correlation metric is calculated using two approaches:



(a) Forward Selection (b) Greedy Forward Selection

Fig. 1. Forward selection.

3.3.1. Pearson's correlation

Evaluates how accurately a feature can predict the class of the software module when other features are discarded. Each feature is ranked based on the attained correlation score p:

$$p = \frac{cov(X_{i,} Y)}{\sqrt{var(X_i) \cdot var(Y)}} \tag{2}$$

where $var(X_i)$ and $cov(X_{i,}, Y)$ represent the variance of feature X_i and the covariance between a feature X_i and the target class Y, respectively.

3.3.2. Fisher's criterion

Provides a measure of the F-score

$$F(X_i) = \frac{\left(\widehat{X}_i^1 - \widehat{X}_i^2\right)^2}{\left(S_i^1\right)^2 - \left(S_i^2\right)^2} \tag{3}$$

where $\widehat{X}_i$ and, S_i are the average and the variance for feature X_i, respectively.

The Pearson's correlation ranges from +1 and −1 where 1, 0 and −1 indicate positive, no, and negative correlation, respectively. Fisher's criterion allows the selection of the most discriminant features from a dataset [47].

3.4. Ensemble learning

In this paper, an ensemble of classifiers is proposed to address the problem of robust software defect classification. This model is based on the average probability ensemble (APE) approach. Ensemble learning is the process of grouping learning models generated from a set of base classifiers. Such models are expected to exhibit robustness against data imbalance and feature redundancy that usually hinder software defect datasets. This robustness is guaranteed by averaging the classification performance of multiple classifiers. This averaging leads to the elimination of uncorrelated errors and, thus, enhancing overall classification performance [48]. Unlike voting schemes, model classifiers with outputs viewed as probability outcomes are in conformity with the AUC measure, which quantifies the degree of confidence associated with the selected class. In addition, voting schemes require a hard threshold to assign decisions to selected classes. This threshold may be looked at as a quantization step that will definitely introduce errors in the decisions made [22]. Therefore, our selection of average probability ensemble serves two purposes: (1) conformity with the AUC measure and (2) obsoletes the need for threshold selection.

In the proposed APE model, each classifier predicts the probability that a software component belongs to the defective class. Then, the output probabilities are averaged as the final probability estimation. An outline of the proposed framework is shown in Fig. 2 where seven base classifiers are combined to form the APE model. These classifiers include random forests, gradient boosting, stochastic gradient descent, W-SVMs, logistic regression, multinomial naive Bayes, and Bernoulli naive Bayes. In this paper, the selection of the base classifier types is solely motivated by their wide acceptance in the software engineering community specifically and the machine learning experts at large [37]. Lessmann et al. provide a survey study where several defect classification methods are evaluated [37]. These classification methods include statistical, clustering, neural and SVM and decision tree models. Each of these base classifiers, used in our proposed ensemble learning model, is introduced below. Finally, the selection of seven base classifiers is empirically justified in the sequel of the paper.

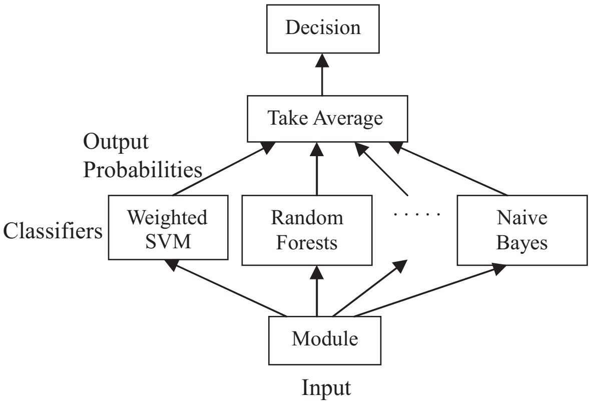


Fig. 2. The proposed framework for the average probability ensemble.

3.4.1. Random forests

Random forests consist of several unpruned classification or regression trees. Using random feature selection, these trees are induced from bootstrap samples of the training data [20]. In classification problems, each data sample is fed down each of the trees in the random forest. Then, the latter outputs as its decision class the class that received most of the votes made by the individual trees. Breiman [20] showed that error rates in random forests depend on the strength of each individual tree and the correlation between any two trees in the forest. However, results extracted from random forests are difficult to interpret. A typical random forest is shown in Fig. 3. In such settings, each individual tree handles a small subset of features selected randomly. Then, each tree is optimized using this subset.

3.4.2. Gradient boosting

Friedman proposed gradient boosting to solve regression problems using a prediction model consisting of an ensemble of weak predictors [49]. These predictors are typically decision trees. Given a set of decision trees, {DT$_1$, DT$_2$, … , DT$_i$, … , DT$_n$}, the gradient boosting algorithm produces a weighted summation of the output decisions of each individual tree as follows:

$$f(x) = w_0 + w_1h_1(x) + w_2h_2(x) + \cdots + w_nh_n(x) \tag{4}$$

where $h_i(x)$ is the output decision of the ith individual tree. The weights, w_i, applied on the individual decisions are optimized by minimizing a differentiable loss function [49]. Finally, classification problems can also be solved using the gradient boosting method by transforming them into a regression with an appropriate loss function [49].

3.4.3. Stochastic gradient descent (SGD)

Classification and regression problems involving large datasets are efficiently solved using second order stochastic gradient and averaged stochastic gradient techniques [50]. In the stochastic gradient descent, cost functions are minimized using the stochastic gradient descent (SGD) algorithm:

$$w_{k+1} = w_k - \mu\nabla_w Q(x_k, w_k) \tag{5}$$

where μ is the learning rate of the SGD algorithm and $Q(x_k, w_k)$ is the instantaneous approximation of the loss function $Q(x, w)$ using the input vector x_k at instant k. In this way, the model parameters or features, w_k, are updated incrementally using each input vector. When μ is sufficiently small, the SGD algorithm achieves linear convergence [50]. Finally, the SGD algorithm, also known as the second-order gradient descent (2GD), represents a variant of the well-known Newton algorithm [50]. Fig. 4 illustrates the effects of

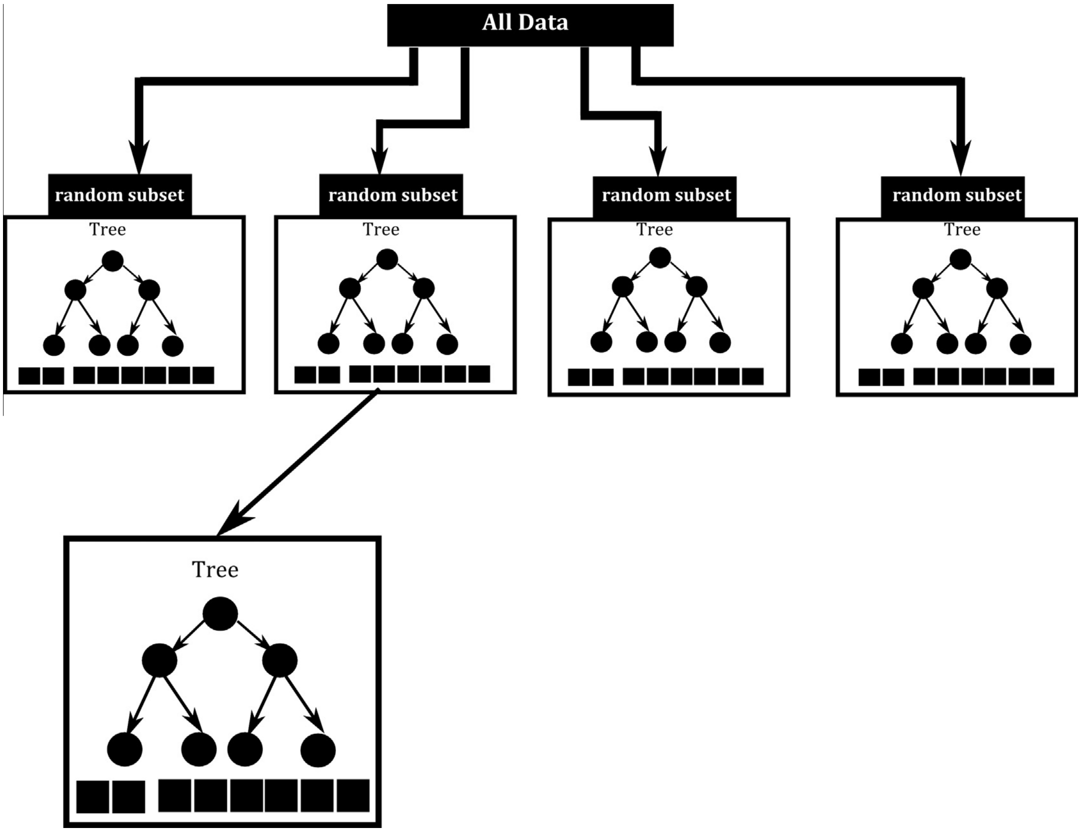


Fig. 3. Typical random forest with many individual decision trees (for regression or classification).

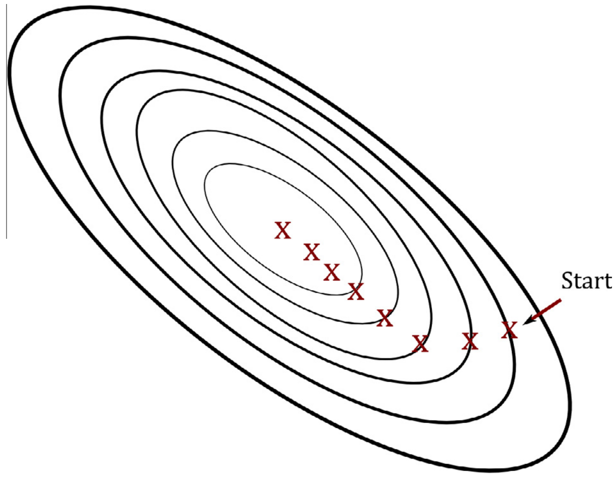


Fig. 4. Illustration of incremental update in the SGD algorithm.

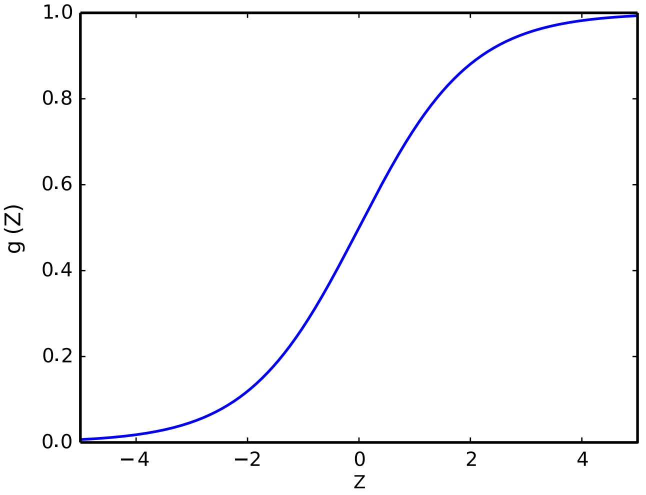


Fig. 5. Logistic (sigmoid) function.

instantaneous update of the model parameters, w_k, on the error curve. Steepest descent is ensured using a small learning rate at the expense of resulting in a local minima [50].

3.4.4. Logistic regression

Logistic regression provides a very powerful discriminative model based on the well-known logistic (sigmoid) function [22]:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

The logistic function, shown in Fig. 5, has very attractive properties including continuous differentiability and linear relation between the function and its derivatives (of any order) [22]. The logistic regression has been successfully applied in classification problems. Given two classes, labeled $Y = 0$ and $Y = 1$, and N n-dimensional features $\{x_1, x_2, \ldots, x_i, \ldots, x_N\}$ where each feature sample is treated as a random vector consisting of discrete random variable, the logistic regression yields a generative model[4] that learns $p(Y|x)$ using a direct application of Bayes rule as follows [22]:

$$p(Y|x) = \frac{p(x|Y)p(Y)}{p(x)} = \frac{p(x|Y)p(Y)}{\sum_y p(x|Y = y)} \tag{7}$$

Therefore, the logistic regression technique splits the model learning into two steps requiring the knowledge of $p(X|Y)$ and $p(Y)$. Then, the following probabilities are assigned:

$$p(Y = 1|y) = \frac{1}{1 - e^{-y}} \tag{8}$$

$$p(Y = 0|y) = 1 - p(Y = 1|y) = \frac{e^{-y}}{1 - e^{-y}} \tag{9}$$

where y is the model output given by:

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \tag{10}$$

Finally, the minimization of the cost function in logistic regression models is very similar to that in SGD algorithms. However, this paper applies a batch learning approach in the implementation of the logistic regression classifier.

3.4.5. Multinomial naive Bayes

The standard naive Bayes classifier provides simplified generative models given the assumption of statistical independence between the model inputs and outputs. Using this assumption, Eq. (7) becomes:

$$p(Y|x) = \frac{p(Y)\prod_{i=1}^{N} p(x_i|Y)}{\sum_y \prod_{i=1}^{N} p(x_i|Y)} \tag{11}$$

Maximum likelihood estimates can be obtained for the quantities defined in Eq. (11) [22]. In multinomial naive Bayes classifiers, the data samples follow a multinomial distribution. This model is commonly used in text classification problems.

3.4.6. Bernoulli naive Bayes

When the underlying data follows a multivariate Bernoulli distribution, Bernoulli naive Bayes classifiers are preferred. In this case, the feature samples are assumed to be binary-valued (Bernoulli or Boolean) variables. Given a sample, x_i, the decision rule for the Bernoulli naive Bayes classifier is defined as:

$$p(x_i|Y) = p(i|Y)x_i \times (1 - p(i|Y))(1 - x_i) \tag{12}$$

Unlike multinomial naive Bayes classifiers, Bernouilli-based ones explicitly penalize the non-occurrence of a feature i that is an indicator for class Y.

3.4.7. Regular and weighted support vector machines

Using statistical learning theory, Vapnik et al. proposed support vector machines (SVMs) [51]. SVMs have found successful application in many fields including bioinformatics, text mining, image recognition, system identification and leak detection [52,53]. SVMs represent a discriminative classifier, defined by a separating hyperplane, finds an optimal hyperplane that separates samples pertaining to two different labels (binary classification) [22,54]. Fig. 6 illustrates the discriminating capability of SVMs in binary classification problems. Data samples featured on the hyperplanes represent the support vectors. SVMs generate optimal hyperplanes via the solution of a Lagrangian optimization problem. This optimization is usually formulated in the dual space and takes into account kernels. Several issues company the use of SVMs including its extension to multi-class problems and regression applications [22,54]. Moreover, finding optimal settings for SVMs is time-consuming unlike other models such as neural networks. To tackle these issues, several solutions are proposed including the use of kernels, derivative-free cost functions and grid search optimization. Fig. 7 shows the positive effect of kernels on the classification performance of SVMs. The use of sophisticated kernels (Fig. 7 right) boosts the discriminating capability of SVMs.

[4] Unlike generative models, p(Y|x) is estimated directly by discriminative models [22].

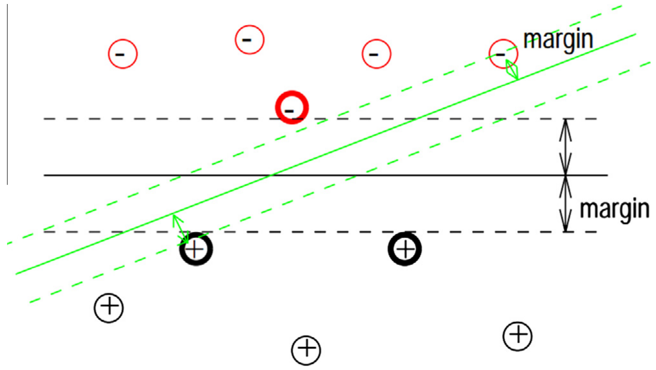


Fig. 6. Hyperplanes in SVMs.

Finally, many issues have attracted a lot of attention in the literature including the design of new kernels tailored to a particular application, improving the generalization ability and improving the training speed in the case of large datasets [22,54].

Given N m-dimensional feature vectors $x^{(i)} = \{x_1, x_2, \ldots, x_i, \ldots, x_m\}$ $(i = 1, 2, \ldots, N)$ and their associated class label $y^{(i)} \in \{-1, +1\}$, the two-class SVM model is defined as:

$$\min \quad \mathrm{E}(\mathbf{w}, b, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \Phi(\xi) \tag{13}$$

$$\text{subject to}: \begin{cases} y_i(\langle \mathbf{w}, \mathbf{z}^{(i)} \rangle + b) \geq 1 - \xi^{(i)} & \forall i = 1, 2, \ldots, N \\ \xi^{(i)} \geq 0 & \forall i = 1, 2, \ldots, N \end{cases}$$

where $\boldsymbol{z}^{(i)} = \varphi(\boldsymbol{x}^{(i)})$ defines the non-linear mapping, $\varphi(.)$, applied on the ith feature vector $\boldsymbol{x}^{(i)}$. The SVM hyperplane is expressed by $\boldsymbol{w}$ and b. The slack variables, $\xi^{(i)}$, allow the training samples to be misclassified or located inside the margin [51]. Eq. (13) penalizes the solutions impacted with many training error using the term $\varphi(.)$. A commonly used penalty term is given by [51][5]:

$$\Phi(\xi) = \sum_{i=1}^{N} \xi^{(i)} \tag{14}$$

Eq. (13) forms the basis for weighted SVMS (W-SVMs) which assigns more weights to the training samples pertaining to the minority class. Given the imbalanced nature of the software defect datasets, the classification models based on W-SVMs is considered for benchmarking purposes in this paper. The proper handling of data imbalance in W-SVMs is illustrated in. As shown in Fig. 8, the decision boundary defined by W-SVMs (lower line) results in fewer misclassified defect samples.

4. Experimental design and setup

In this section, we present the details of the experimental design and setup.

4.1. Experimental setup

In this paper, all computer experiments were performed in a Windows 7-based computer with 3.6 GHZ quad-core CPU and 32 GB of RAM. Performance evaluation was carried out by running a 10-fold cross-validation. The performance of each classifier is evaluated against each software defect dataset. The experimental design considers several performance factors including the performance of the GFS selection method, the correlation-based method with its two variants, Fisher's criterion, Pearson's correlation, and the proposed APE ensemble learning model. All evaluated algorithms were implemented using Python language. Classification performance is measured using the AUC measure. ROC curves are popular metrics to evaluate classification algorithms against imbalanced datasets [35]. The AUC measure determines the extent of the area below a ROC curve and is computed as follows:

$$AUC = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} 1_{pi>pj} \tag{15}$$

where the index i loops over the correctly predicted positive class samples and j loops over the correctly predicted negative class samples. p_i, p_j are the predicted probabilities to the data sample i and j respectively. Finally, $1_{pi>pj}$ returns 1 if and only if $p_i > p_j$, and 0 otherwise.

4.2. Experimental design

The proposed APE ensemble learning model is benchmarked against the basic classifiers W-SVMs and random forests. Moreover, some feature selection techniques are assessed to verify Hypothesis 1. More specifically, the selection criterion, considered in the performance evaluation, includes Pearson's correlation, Fisher's criterion, and the GFS approach. Finally, the classification performance of the proposed APE model is evaluated using six publicly available software defect datasets:

1. Ant-1.7.
2. Camel-1.6.
3. KC3 datasets.
4. MC1.
5. PC2.
6. PC4.

More details on the first three datasets are available in [55]. The last three datasets are accessible online.[6] Although there have been questions raised about the quality of these datasets [45], yet these datasets have been widely used by researchers [56]. Table 1 provides a summary of these datasets. It is noteworthy to mention that these datasets consist of commonly used software metrics including LOC, McCabe's CC, Halstead's length, Halstead's volume, Halstead's difficulty, total number of operands (TNO), and branch count (BC). A comprehensive list of the metrics included in the test software defect datasets is given Table 2. On average, each of the six datasets contains 30 metrics. Data imbalance is consistently present in all datasets with a ratio of defective to total samples ranging 1.01–22.28.

5. Analysis and discussion

This section presents the analysis of the experiment and the discussion.

5.1. Performance evaluation of feature selection methods

Using the AUC measure, the effect of three popular feature selection techniques on the defect classification performance is reported in Fig. 9. As mentioned earlier, results pertaining to two basic classifiers (W-SVMs and random forests) are also reported for benchmarking purposes. The W-SVMs classifier assigns weights that are inversely proportional to the class occurrence in the dataset [57]. More specifically, the minority class (i.e., defective modules) receives more weight, which allows for potentially better model fitting in the case of imbalanced datasets.

[5] Any feasible solution (w, b, ξ) assign $\xi \geqslant 1$ to misclassified examples. However, it is very hard to base the penalty term $\Phi(.)$ on these variables [46].

[6] Available at: http://nasa-softwaredefectdatasets.wikispaces.com/.

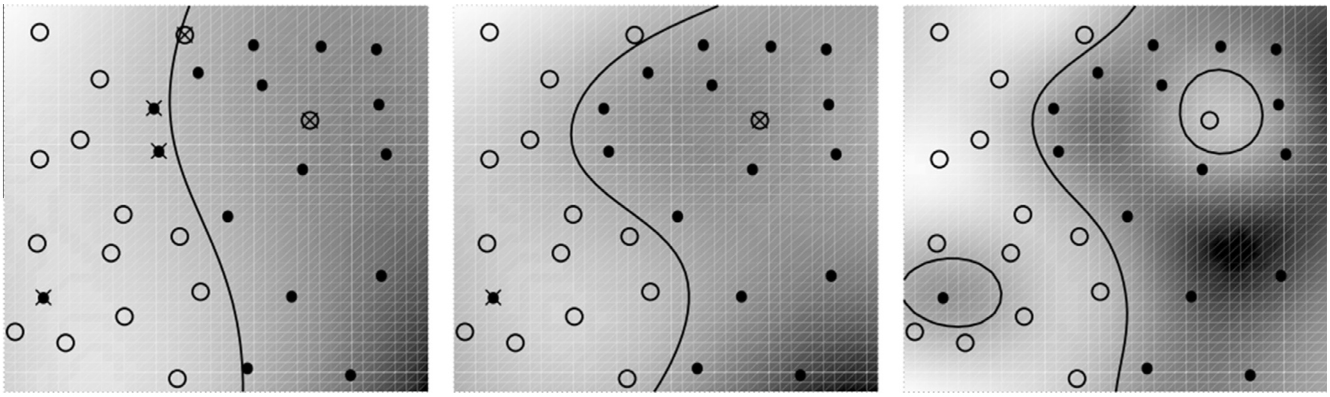


Fig. 7. Effect of kernels on SVMs.

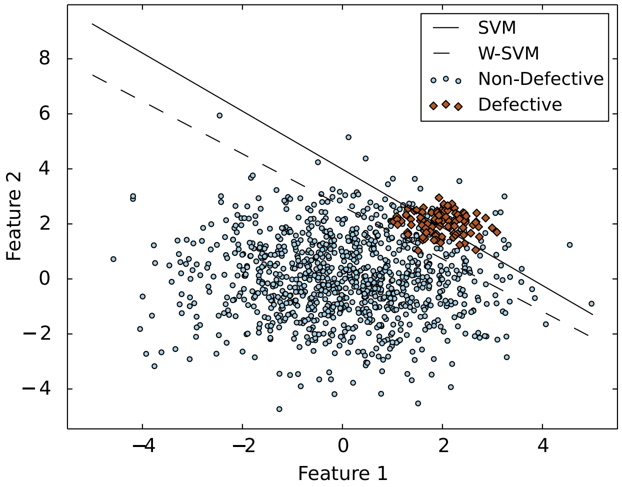


Fig. 8. Decision boundaries generated by SVMs and W-SVMs. W-SVMs correctly classified more defective components than SVMs.

Table 1
Defect datasets information.

Dataset	Language	# Components	# Defective components
Ant-1.7	Java	745	166
Camel-1.6	Java	965	188
KC3	Java	200	36
MC1	C++	1988	46
PC2	C	1585	16
PC4	C	1287	177

Fig. 9 reveals an interesting trend in classification performance achieved by the FGS selection technique. In fact, for all considered datasets, FGS does not only outperform Fisher's criterion and Pearson's correlation method and attains the highest AUC values but also it requires the least number of features to achieve optimal classification accuracy. Another interesting fact stems from the results shown in Fig. 9a and e regarding the degradation in classification performance due to the use of all available features. This degradation in performance manifests itself as a result to the presence of redundant and/or irrelevant elements in the selected feature set. More specifically, using all feature elements, W-SVMs achieved 0.76 and 0.04 AUC on ant-1.7 and PC2 datasets, respectively. However, using a small, but more meaningful, feature subset, W-SVMs model was able to attain AUCs scores of 0.87 and 0.92, respectively. On the other hand, the Pearson's correlation approach picks many poor features, which affects negatively the classification performance as evidenced by Fig. 9c. In this case, the attained AUC decreased to less than 0.1 when 21, 24, 36, 39 features were included in the feature set.

On the remaining datasets, similar performance was observed with Pearson's correlation technique. Unlike the latter technique, Fisher's criterion was able to pick better features as indicated by the AUC values shown in Fig. 9c–f. However, this performance does not compete with that achieved by the GFS method. On the other hand, a careful inspection of the feature list produced by the GFS method indicate that the cyclomatic complexity, lines of code, lines of comments, condition count, parameter count, and Halstead operators metrics were the most meaningful features that helped boost the classification performance. The GFS feature selection method is adopted as a preprocessing step when proposing an enhanced version of the APE ensemble learning method as detailed in Section 5.3.

Finally, the performance results, summarized in Fig. 9, confirm the assertion given by hypothesis1. As expected, an informed feature selection, the GFS method in our case, has not only contributed significantly in improving the classification performance but also reduced the number of features (software metrics) required to achieve such performance. Therefore, we accept Hypothesis 1.

5.2. Performance evaluation of proposed APE model

As stated earlier, the proposed APE ensemble learning model consists of seven base classifiers whose output probabilities are averaged as the final classification decision. Using the software defect datasets described above, the performance of the APE model is benchmarked against that yielded by the W-SVMs and random forests classifiers on an equal-foot standing in terms of preprocessing and feature selection. The AUC measures attained by the three classifiers are given in Table 3. As asserted in the second hypothesis given in this paper, the APE model has achieved the highest AUC measure of 0.82 followed by W-SVMs and random forests with AUC scores of 0.79 and 0.77, respectively. Interestingly enough, the proposed APE model has outperformed W-SVMs, which specifically designed for imbalanced data. This clearly indicates that the averaging of decision probabilities has enabled the APE model to adjust its decision boundaries to accommodate better the minority class samples. This averaging is the result of each base classifier in the APE model trying to compensate for the errors induced by the other base classifiers. In addition to the robustness to data imbalance, the APE will be further enhanced for robustness against the presence of redundant and irrelevant features. This suggests that such Ensembles are robust for software defect classification.

The highest classification was attained by the proposed APE model against the PC2 dataset where an AUC measure of 0.91

Table 2
Metrics used in the study.

Metric	Code[a]	Metric	Code
Decision count	M_1	Node count	M_27
Cyclomatic complexity	M_2	Response for class	M_28
Global data density	M_3	Measure of functional abstraction	M_29
Halstead difficulty	M_4	Edge count	M_30
Halstead content	M_5	Decision density	M_31
Maintenance severity	M_6	Design density	M_32
Coupling between objects	M_7	Parameter count	M_33
Number of unique operands	M_8	Number of public methods	M_34
Afferent couplings	M_9	Lines of commented lines	M_35
Essential density	M_10	Lines of code	M_36
Global data complexity	M_11	Halstead programming time	M_37
Efferent couplings	M_12	Halstead level	M_38
Lack of cohesion of methods	M_13	Design complexity	M_39
Condition count	M_14	Call pairs	M_40
Lines of blank lines	M_15	Lines of executable code	M_41
Average method complexity	M_16	Halstead volume	M_42
Number of children	M_17	Lines of code & comments	M_43
Essential complexity	M_18	Halstead error estimate	M_44
Measure of aggregation	M_19	Halstead effort	M_45
Percentage of comments	M_20	Cyclomatic density	M_46
Depth of inheritance tree	M_21	Halstead length	M_47
Coupling between modules	M_22	Inheritance coupling	M_48
Multiple condition count	M_23	Cohesion among methods	M_49
Number of unique operators	M_24	Data access metric	M_50
Weighted methods for class	M_25	Maximum cyclomatic complexity	M_51
Branch count	M_26		

[a] An abbreviated code is assigned to each metric considered for clarity purposes only.

was achieved. At the same time, W-SVMs and random forests could only achieve AUC scores of 0.14 and 0.72, respectively. It is very surprising that the W-SVMs model has performed so poorly in the case of PC2 dataset. This drastic drop in classification performance may require further investigation of the decision boundaries of W-SVMs and the nature of PC2 dataset and its underlying features.

Another interesting fact is revealed from the performance of the proposed APE model, which includes W-SVMs as one of its base classifiers. More specifically, the APE model achieved an AUC measure reaching 0.91 using the PC2 dataset although the W-SVMs model, when used alone, achieved a low AUC measure of 0.14. It is clear that the averaging power of the APE model is not affected at all by the individual, although poor, performance of one of its base classifiers.

Finally, the performance results, summarized in Table 3 confirm the assertion given by the second hypothesis of this paper. As claimed, ensemble learning outperforms single classifiers in terms of classification accuracy especially when the base classifiers are selected based on their known base performance. In addition, the overall performance of ensemble learning is not hindered by its base low-performing classifiers. Therefore, we accept Hypothesis 2.

5.3. Effects of number and type of base classifiers on APE classification performance

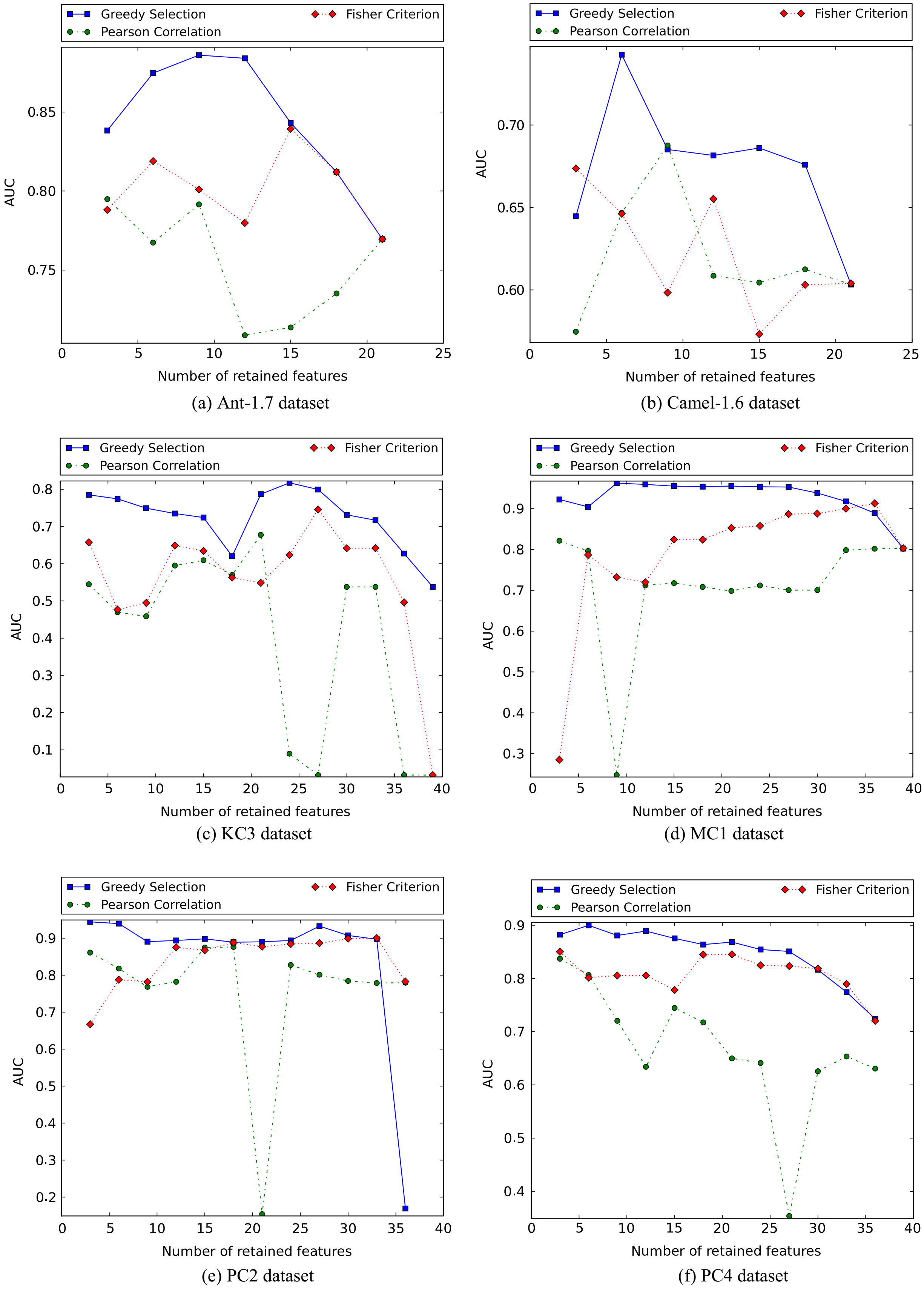
As mentioned earlier, the proposed APE model uses seven different base classifiers. The APE model may consist of a single base classifier with different settings (i.e., homogeneous APE) or different base classifiers (i.e., heterogeneous APE). Using KC3 and camel datasets, the effect of the number of base classifiers in the homogenous APE model is portrayed in Fig. 10.

For both datasets, the performance of the APE model does not improve beyond seven classifiers. There is even a slight drop in performance when using nine classifiers. Similar trend is observed using the remaining datasets. Therefore, the selected model consists of seven base classifiers based on an optimized SVM model. A similar assessment is carried out to find the "optimal" number of base classifiers in the heterogeneous APE model. Fig. 11 provides a summary of this assessment using PC2 and PC4 datasets. It is clearly visible that adding more classifiers in the case of PC4 dataset does not improve the classification performance but only degrades it beyond 10 classifiers. This drop in performance clearly indicates the presence of over fitting which is due to base learners biased towards training data samples. On the other hand, the classification does not improve much beyond 10 classifiers in the case of PC2 dataset. The use of seven base classifiers is suggested by this performance trend. It should be noted that similar behavior is observed with respect to the remaining datasets.

5.4. Effect of feature selection on APE classification performance

The promising classification results attributed to feature selection, depicted in Fig. 11, call for considering the incorporation of a feature selection module into the proposed APE model. This extended model, called enhanced APE, selects only defect metrics that are expected to boost the classification performance. Furthermore, feature selection is carried out using the FGS method, which outperformed other feature selection methods as reported in Fig. 11. To highlight the performance enhancement attributed to the FGS method, classification performance to the APE model and its enhanced version are summarized in Table 4. The enhanced model has outperformed the basic one in the case of all software defect datasets. This classification improvement is attributed to the FGS method, which considers only relevant and meaningful software metrics in the classification process. Irrelevant and redundant software metrics are, therefore, discarded.

Table 4 reveals another interesting fact regarding the nature of the software metrics pertaining to each dataset considered. Some datasets contain more irrelevant and/or redundant software metrics, which is reflected by more improvement in the classification performance. The MC1 defect dataset contains such type of software metrics. For the remaining datasets, the classification performance was enhanced in acceptable ranges.



(a) Ant-1.7 dataset

(b) Camel-1.6 dataset

(c) KC3 dataset

(d) MC1 dataset

(e) PC2 dataset

(f) PC4 dataset

Fig. 9. AUC performance measures of feature selection techniques (FGS, Fisher's criterion and Pearson's correlation).

Table 3
Classification results of W-SVMs, random forests and APE models.

Datasets	W-SVMs	Random forests	APE
ant-1.7	0.77	0.79	**0.82**
camel-1.6	0.59	0.67	**0.73**
KC3	0.52	0.68	**0.75**
MC1	0.80	0.83	**0.87**
PC2	0.14	0.72	**0.91**
PC4	0.65	0.88	**0.90**

The effect of feature selection on the performance of existing algorithms such as W-SVMs and random forests is assessed below. Table 5 summarizes the performance of the proposed enhanced algorithm along with enhanced versions of the classifiers based on W-SVMs and random forests. As formulated in the first hypothesis, feature selection enhances the classification of the proposed and existing classifiers as well. In fact, the enhanced versions of W-SVMs and random forest classifiers exhibited the expected behavior. Moreover, the enhanced version of the proposed model, enhanced APE, outperforms the enhanced version of both classifiers.

The geometric mean (G-mean) represents a commonly used performance measure in classification problems involving imbalanced datasets [46]. This measure indicates the classifier performance on the majority and minority classes. This measure involves the sensitivity and specificity measures. These two measures are defined as follows:

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{16}$$

$$\text{specificity} = 1 - \frac{FP}{TN + FN} \tag{17}$$

where TP, FP, TN and FN define the true, false positives, true and false negatives, respectively. Using the sensitivity and specificity, the G-mean takes into account the accuracy on both positive and negative samples [46]:

$$G\text{-mean} = \sqrt{\text{specificity} \times \text{sensitivity}} \tag{18}$$

Finally, Table 6 provides a summary of the classification performance of the proposed two-variant ensemble learning algorithm along with a comparison with simple and enhanced versions of existing algorithms (W-SVMs and random forests) using the

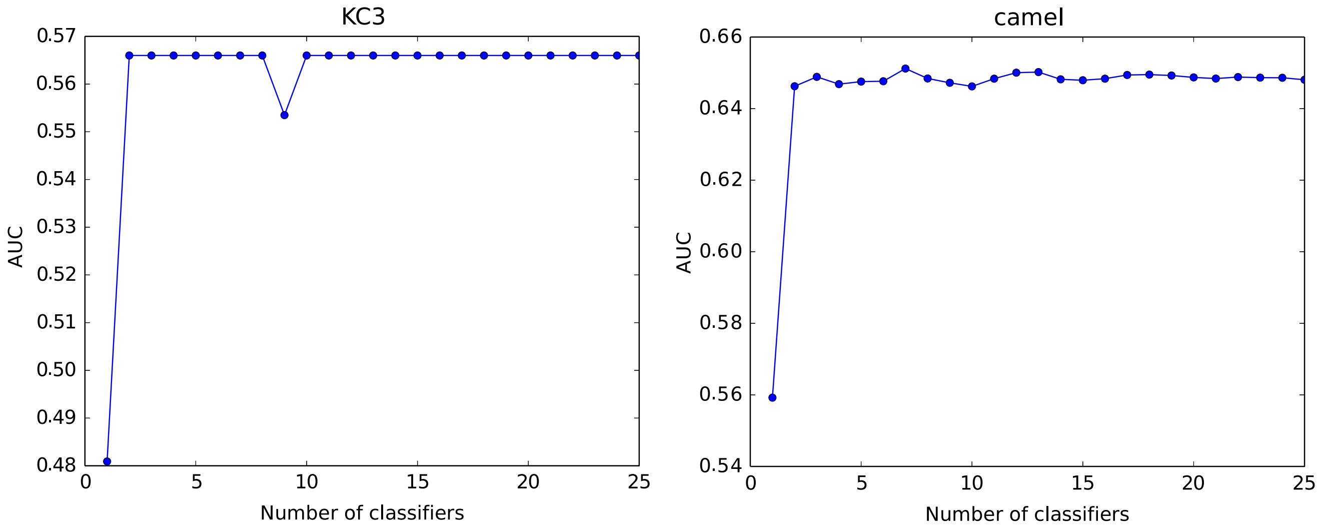


Fig. 10. Effect of number of base classifiers on homogeneous APE classification performance.

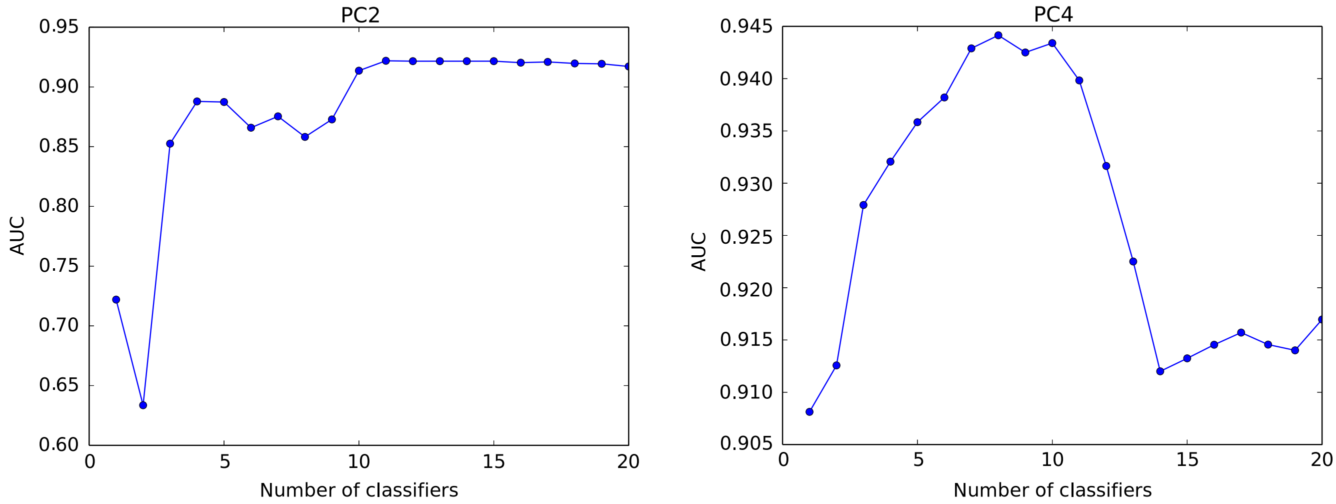


Fig. 11. Effect of number of base classifier on heterogeneous APE classification performance.

Table 4
Classification results of APE and enhanced APE models.

Datasets	APE	Enhanced APE
ant-1.7	0.82	**0.86**
camel-1.6	0.73	**0.80**
KC3	0.75	**0.86**
MC1	0.87	**0.98**
PC2	0.91	**0.95**
PC4	0.90	**0.96**

Table 5
Classification results of enhanced APE model versus enhanced versions of W-SVMs and random forests (AUC measure).

Datasets	Enhanced APE	Enhanced W-SVMs	Enhanced random forests
ant-1.7	**0.86**	0.80	0.82
camel-1.6	**0.80**	0.69	0.72
KC3	**0.86**	0.84	0.82
MC1	**0.98**	0.95	0.92
PC2	**0.95**	0.94	0.85
PC4	**0.96**	0.89	0.93

G-mean measure. It is noteworthy that the effect of data imbalance on classification performance is properly captured by the G-mean measured as clearly indicated Table 6. More specifically, a zero G-mean measure indicates that the classifier has performed very poorly with respect to the samples pertaining to the minority class. This is the case for the W-SVMs classifier with the KC3, PC2 and ant-1.7 datasets. Also, the classifier based on random forests yielded a similar behavior with the PC2. The non-enhanced version of the proposed model, simple APE, yielded a poor performance also against the PC2 dataset. However, the enhanced version did not only outperform the other classifiers but also attained the highest classification performance approach a G-mean measure of one. To the best knowledge of the authors, this is highest classification performance reported in the literature using the software defect datasets considered and the G-mean measure. The drastic performance improvement attributed to the feature selection process where only meaningful software metrics are retained reveals an interesting finding that is worth the analysis carried out in Section 5.5.

5.5. GFS-based selected features

Based on the results reported in Table 6, it is very legitimate to investigate the quality of the retained features using the GFS technique. Tables 7 and 8 provide a summary of the retained features using the GFS technique using AUC and G-Mean measures, respectively. Many insights are obtained by careful inspections of Tables 7 and 8. A simple comparison between the number of metrics retained using AUC and G-Mean measures reveals the efficiency of the latter in picking up only a small subset of the initial metrics. Also, the number of selected metrics differ from one algorithm to another with slight higher number in the case of the enhanced APE algorithm. On the other, all algorithms agree on the “most discriminating” metrics across the datasets considered. For instance, this fact has not been revealed in the benchmark study of Lessmann et al. [37]. To the best knowledge of the authors, the nature of retained software metrics has not been investigated in the literature. In fact, Tables 7 and 8 do not only reveal the relationship between software metrics and the presence of defects but also confine this relationship to a small subset of software metrics. This restricted relationship is clearly disclosed by the G-Mean measure. Another important contribution of the work presented in this paper consists in guarding the defect classifier from ignoring samples pertaining to the defect class. More specifically, Table 6 clearly identifies this trend by scoring a zero G-Mean for the classifier that has the tendency to simply ignore the minority class samples. The inaccuracy of the conclusions drawn by Lessmann et al. [37].

5.6. Comments on software metrics for defect classification

The results, presented and discussed above, have shed the light of several issues that concerned software engineering practitioners. The search for adequate and relevant software metrics that are strongly related to software quality has been the focus of active research [56]. However, the questionable quality of available software defect datasets have only added to this task complexity [45]. Contrary to the conclusions drawn by Lessmann et al. [37], the selection of a small subset of software metrics should not over shadow the emphasis on the classifier design and requirements. Moreover, widely accepted performance measures such as the AUC should be used with caution since these measures reveal only the classifier overall performance. Further, the AUC measure could be misleading in case of highly imbalanced samples as it is the case with software defect datasets. This paper combines the GFS technique and G-Mean measure to strike a balance between efficient feature selecting and accurate classification measure.

6. Threats to validity

There are a number of threats that may have an impact on the results of this study. Below we discuss the internal and external threats to validity.

An internal threat is related to the selection of metrics used as predictors. Many metrics are defined in literature. Other metrics might be better indicator to defects; however, we used metrics that were available from the selected datasets.

The research conclusions presented in this paper are based on the selected systems. These systems may not be characteristic of all industrial domains It was also reported that NASA datasets have questionable quality [45]. These systems may not also be good representatives in terms of the numbers and sizes of classes. Although this may be looked at as an external threat to the accuracy of the

Table 6
Classification results of enhanced APE model versus enhanced versions of W-SVMs and random forests (G-mean measure).

Datasets	W-SVMs	Random forests	APE	Enhanced W-SVMs	Enhanced random forests	Enhanced APE
ant-1.7	0	0.62	**0.62**	0.68	0.67	**0.84**
camel-1.6	0.32	0.35	**0.25**	0.78	0.49	**0.79**
KC3	0	0.23	**0.40**	0.71	0.56	**0.83**
MC1	0.323	0.40	**0.10**	0.77	0.63	**0.90**
PC2	0	0	**0**	0.90	0.17	**0.95**
PC4	0.05	0.50	**0.50**	0.85	0.72	**0.95**
camel-1.6	0.32	0.35	**0.25**	0.78	0.49	**0.79**

Table 7
AUC-based retained features.

Dataset	Algorithm		
	Enhanced W-SVMs	Enhanced random forests	Enhanced APE
ant-1.7	M_49 M_16 M_13 M_16	M_25 M_28 M_49 M_17 M_36 M_22 M_21 M_35 M_16	M_28 M_17 M_48 M_13 M_50 M_19 M_51 M_29 M_36 M_12 M_22 M_49 M_35 M_25 M_34 M_21 M_9 M_16
camel-1.6	M_13 M_35 M_51 M_22 M_17 M_21 M_16 M_28 M_12 M_16	M_9 M_16 M_49 M_21 M_29 M_13 M_34 M_12 M_19 M_28 M_22	M_9 M_13 M_51 M_21 M_29 M_34 M_17 M_22 M_16
KC3	M_2 M_43 M_18 M_33 M_2 M_6 M_11 M_32 M_36	M_43 M_24 M_44 M_10 M_23 M_36	M_2 M_43 M_18 M_33 M_23 M_36
MC1	M_36 M_20 M_47 M_4 M_26 M_40 M_36 M_43M_30 M_27	M_36 M_20 M_15 M_46 M_30 M_1 M_18 M_8 M_27 M_33 M_24 M_40 M_31 M_41M_14 M_6 M_2 M_36 M_35	M_20 M_2 M_31 M_14 M_39 M_46 M_43 M_40 M_1 M_18 M_44 M_11 M_3 M_38 M_33 M_8 M_2 M_36 M_15 M_26 M_24 M_5
PC2	M_37 M_24 M_44 M_41 M_24 M_8 M_1 M_23 M_26 M_20	M_38 M_20 M_8 M_26 M_45 M_5 M_2 M_36 M_40 M_1 M_37 M_33	M_45 M_37 M_24 M_20 M_43M_10 M_36
PC4	M_43M_2 M_15 M_6 M_31 M_14 M_10 M_26 M_2 M_46 M_38 M_18 M_31 M_36	M_43M_14 M_2 M_15 M_36 M_20 M_39 M_33 M_38 M_35 M_42 M_24	M_43M_15 M_24 M_31 M_18 M_40 M_33 M_5 M_35 M_44 M_41 M_36 M_39 M_26 M_14

Table 8
G-Mean-based retained features.

Dataset	Algorithm		
	Enhanced W-SVMs	Enhanced random forests	Enhanced APE
ant-1.7	M_35 M_51 M_19 M_16	M_35 M_16	M_28 M_19 M_51 M_29 M_13 M_48 M_21 M_17 M_16
camel-1.6	M_17 M_9 M_50 M_29 M_16 M_19 M_13 M_49 M_22 M_48 M_51	M_16 M_17 M_21 M_48 M_12 M_22	M_29 M_34 M_21 M_17 M_48 M_28 M_25 M_19 M_9 M_13 M_12 M_50 M_16 M_36
KC3	M_15 M_18 M_43 M_31 M_46 M_31 M_10 M_3 M_44 M_38 M_6 M_2 M_20 M_2 M_35 M_36	M_44 M_43 M_41 M_36	M_8 M_43 M_20 M_5 M_36
MC1	M_15 M_43 M_31 M_36	M_20 M_36 M_26 M_24 M_23 M_47M_2 M_14 M_42 M_33 M_1 M_6 M_10 M_36	M_36 M_46M_26 M_41 M_31 M_14 M_43 M_6 M_39 M_2 M_15 M_36
PC2	M_35 M_18 M_36	M_20 M_38 M_36	M_45 M_37 M_8 M_41 M_1 M_36
PC4	M_43M_2 M_31 M_6 M_10 M_46 M_44 M_36	M_43 M_15 M_31 M_24 M_36	M_43M_2 M_15 M_31 M_10 M_42 M_20 M_40 M_33 M_36

models and hence the validity of the results, this practice is common among the software engineering research community. Another external threat is related to the data analysis process as we wrote our own code, which might be prone to errors in developing the algorithms. To mitigate the risks of coding errors, benchmark datasets are used to assess the correctness of the main code blocks. These datasets are available at the hosting site of *scikit-learn* the Python-based machine-learning library.[7]

7. Conclusion

In this paper, we investigated several feature selection techniques for software defect prediction 3and observed that selecting few quality features makes for much higher AUC than otherwise. We also presented the efficacy of ensemble learning against imbalanced datasets with redundant features. A two-variant ensemble learning classifier is proposed. Experimentation, conducted on six datasets, has shown that greedy forward selection outperformed correlation-based forward selection substantially. Furthermore, we demonstrated the effectiveness of using average probability

[7] Various benchmark datasets with test cases are available at: http://scikit-learn.org/stable/.

ensemble (APE) that consists of seven carefully devised learners, which attained improved results over conventional methods such as weighted SVMs and random forests. Finally, the enhanced version of the proposed model, APE combined with greedy forward selection, attained even higher AUC measures for each dataset, which were close to 1.0 in the case of PC2, PC4 and MC1 datasets.

For future work, we intend to apply other feature selection techniques to further verify the claim that many features, present in publicly-available software defect datasets, are irrelevant and/or redundant. In fact, Principal Component Analysis (PCA) at small scale have shown that only few features capture the whole variance of the KC1 NASA dataset. Future work would also include the exploration of other ensemble learners to compare with against our method.

Acknowledgment

The authors acknowledge the support of King Fahd University of Petroleum and Minerals in the development of this work.

References

[1] H. Lim, A.L. Goel, Software Effort Prediction, Wiley Encyclopedia of Computer Science and Engineering, 2008.
[2] M. Jorgensen, Experience with the accuracy of software maintenance task effort prediction models, IEEE Trans. Softw. Eng. 21 (1995) 674–681.
[3] M. Riaz, E. Mendes, E. Tempero, A systematic review of software maintainability prediction and metrics, in: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 367–377.
[4] Y. Zhou, H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, J. Syst. Softw. 80 (2007) 1349–1361.
[5] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inform. Softw. Technol. 54 (2012) 248–256.
[6] A. Tosun, A. Bener, B. Turhan, T. Menzies, Practical considerations in deploying statistical methods for defect prediction: a case study within the Turkish telecommunications industry, Inform. Softw. Technol. 52 (2010) 1242–1257.
[7] X. Yuan, T.M. Khoshgoftaar, E.B. Allen, K. Ganesan, An application of fuzzy clustering to software quality prediction, in: Proceedings, 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 85–90.
[8] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Trans. Softw. Eng. 25 (1999) 675–689.
[9] A. Koru, H. Liu, Building effective defect-prediction models in practice, IEEE Softw. 22 (2005) 23–29.
[10] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (2007) 2–13.
[11] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect-proneness prediction framework, IEEE Trans. Softw. Eng. 37 (2011) 356–370.
[12] N. Bouguila, J.H. Wang, A. Ben Hamza, A Bayesian approach for software quality prediction, in: 4th International IEEE Conference Intelligent Systems, 2008, IS'08, vol. 2, 2008, pp. 11–49.
[13] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, et al., Predicting software defects in varying development lifecycles using Bayesian nets, Inform. Softw. Technol. 49 (2007) 32–43.
[14] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, S.J. Aud, Application of neural networks to software quality modeling of a very large telecommunications system, IEEE Trans. Neural Netw. 8 (1997) 902–909.
[15] C. Andersson, A replicated empirical study of a selection method for software reliability growth models, Empirical Softw. Eng. 12 (2007) 161–182.
[16] Taghi M. Khoshgoftaar, K. Gao, A. Napolitano, An empirical study of feature ranking techniques for software quality prediction, Int. J. Softw. Eng. Knowl. Eng. 22 (2012) 161–183.
[17] R. Akbani, S. Kwek, N. Japkowicz, Applying support vector machines to imbalanced datasets, in: Machine Learning: ECML 2004, Springer, 2004, pp. 39–50.
[18] N. Japkowicz, S. Stephen, The class imbalance problem: a systematic study, Intell. Data Anal. 6 (2002) 429–449.
[19] Y. Sun, M.S. Kamel, A.K. Wong, Y. Wang, Cost-sensitive boosting for classification of imbalanced data, Pattern Recogn. 40 (2007) 3358–3378.
[20] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[21] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (2009) 1263–1284.
[22] R. Duda, P. Hart, D. Stork, Pattern Classification, John Wiley & Sons, 2004.
[23] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, Improving software-quality predictions with data sampling and boosting, IEEE Trans. Syst., Man Cybernet., Part A: Syst. Hum. 39 (2009) 1283–1294.
[24] L. Pelayo, S. Dick, Evaluating stratification alternatives to improve software defect prediction, IEEE Trans. Reliab. 61 (2012) 516–525.
[25] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, 2011. arXiv:1106.1813.
[26] T.M. Khoshgoftaar, E. Geleyn, L. Nguyen, L. Bullard, Cost-sensitive boosting in software quality modeling, in: Proceedings, 7th IEEE International Symposium on High Assurance Systems Engineering, 2002, pp. 51–60.
[27] J.R. Quinlan, Bagging, boosting, and C4. 5, in: AAAI/IAAI, vol. 1, 1996, pp. 725–730.
[28] J. Zheng, Cost-sensitive boosting neural networks for software defect prediction, Expert Syst. Appl. 37 (2010) 4537–4543.
[29] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Trans. Syst., Man, Cybernet., Part C: Appl. Rev. 42 (2012) 463–484.
[30] C.-R. Peng, W.-C. Lu, B. Niu, Y.-J. Li, L.-L. Hu, Prediction of the functional roles of small molecules in lipid metabolism based on ensemble learning, Protein Peptide Lett. 19 (2012) 108–112.
[31] S. Tang, Y.-T. Zheng, Y. Wang, T.-S. Chua, Sparse ensemble learning for concept detection, IEEE Trans. Multimedia 14 (2012) 43–54.
[32] A. Miyamoto, J. Miyakoshi, K. Matsuzaki, T. Irie, False-positive reduction of liver tumor detection using ensemble learning method, in: SPIE Medical Imaging, 2013, pp. 86693B–86693B.
[33] S. Wang, L.L. Minku, X. Yao, Online class imbalance learning and its applications in fault detection, Int. J. Comput. Intell. Appl. 12 (2013).
[34] N. Seliya, T.M. Khoshgoftaar, J. Van Hulse, Predicting faults in high assurance software, in: IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE), 2010, pp. 26–34.
[35] Z. Sun, Q. Song, X. Zhu, Using coding-based ensemble learning to improve software defect prediction, IEEE Trans. Syst., Man, Cybernet., Part C: Appl. Rev. 42 (2012) 1806–1817.
[36] T. Wang, W. Li, H. Shi, Z. Liu, Software defect prediction based on classifiers ensemble, J. Inform. Comput. Sci. 8 (2011) 4241–4254.
[37] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (2008) 485–496.
[38] N. Landwehr, M. Hall, F. Eibe, Logistic model trees, Mach. Learn. 59 (2005) 161–205.
[39] G. Forman, An extensive empirical study of feature selection metrics for text classification, J. Mach. Learn. Res. 3 (2003) 1289–1305.
[40] Y. Saeys, I. Inza, P. Larrañaga, A review of feature selection techniques in bioinformatics, Bioinformatics 23 (2007) 2507–2517.
[41] A. Okutan, O.T. Yıldız, Software defect prediction using Bayesian networks, Empirical Softw. Eng. 19 (2012) 154–181.
[42] L. Yu, A. Mishra, Experience in predicting fault-prone software modules using complexity metrics, Qual. Technol. Quant. Manage. 9 (4) (2012) 421–433.
[43] P.S. Bishnu, V. Bhattacherjee, Software fault prediction using quad tree-based K-means clustering algorithm, IEEE Trans. Knowl. Data Eng. 24 (2012) 1146–1150.
[44] G. Wu, E. Chang, Adaptive feature-space conformal transformation for imbalanced-data learning, in: International Conference on Machine Learning (ICML 2003), 2003.
[45] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, The misuse of the NASA metrics data program data sets for automated software defect prediction, in: Evaluation & Assessment in Software Engineering EASE 25, 2011, pp. 12–25.
[46] G. Nguyen, A. Bouzerdoum, S. Phung, Learning pattern classification tasks with imbalanced data sets, Pattern Recogn. (2009) 193–208.
[47] D. Zhong, J. Han, X. Zhang, Y. Liu, Neighborhood discriminant embedding in face recognition, Opt. Eng. 49 (2010). 077203-077203-7.
[48] T.G. Dietterichl, Ensemble learning, in: The Handbook of Brain Theory and Neural Networks, 2002, pp. 405–408.
[49] J. Friedman, Stochastic gradient boosting, Comput. Stat. Data Anal. 38 (2002) 367–378.
[50] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, 2010, pp. 177–186.
[51] V. Vapnik, The Nature of Statistical Learning Theory, Springer, 2010.
[52] F. Markowetz, Support Vector Machines in Bioinformatics, Master's thesis, University of Heidelberg, 2001.
[53] H. Chen, H. Ye, L. Chen, H. Su, Application of support vector machine learning to leak detection and location in pipelines, in: Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference, 2004, IMTC 04, vol. 3, 2004, pp. 2273–2277.
[54] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2007.
[55] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The PROMISE repository of empirical software engineering data, 2012.
[56] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (2012) 1276–1304.
[57] J.A. Suykens, J. De Brabanter, L. Lukas, J. Vandewalle, Weighted least squares support vector machines: robustness and sparse approximation, Neurocomputing 48 (2002) 85–105.