SOFTWARE DEFECT PREDICTION USING MACHINE LEARNING TECHNIQUES

Mohammad Aquil, Md. Abdul Hai Al Hadi
(1835355050), (1835148050)
CSE-534

Abstract— **Software systems are any software product or applications that support business domains such as Manufacturing,Aviation, Health care, insurance and so on.Software quality is a means of measuring how software is designed and how well the software conforms to that design. Some of the variables that we are looking for software quality are Correctness, Product quality, Scalability, Completeness and Absence of bugs, However the quality standard that was used from one organization is different from other for this reason it is better to apply the software metrics to measure the quality of software. Software maintenance, especially defect prediction, plays an important role in enhancing software quality, resulting reduced cost and time for software testing with great improvement in software reliability and maintenance. Machine Learning approaches are good in solving problems that have less information. In most cases, the software domain problems characterize as a process of learning that depend on the various circumstances and changes accordingly when exposed to new data. This study used public available data sets of software modules and provides comparative performance analysis of different machine learning techniques for software defect prediction. The evaluation process showed that ML algorithms can be used effectively with high accuracy rate.**

I. INTRODUCTION

Nowadays, software systems have become increasingly complex and versatile. It is very important to continuously identify and correct software design defects. Thus, accurately predicting whether a software entity contains design defects can help improve the quality of the software systems. Software metric is measure of some property of a piece of software. It measured during the software development phases such as design or coding and used to evaluate the quality of software. A Software defect is a condition in a software product which does not meet a software requirement or end user. In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect unexpected results. Software defect prediction is the process of locating defective modules in software. For producing high quality software, the delivering final product should have as few defects as possible. For early detection of software defects could lead to reduced development costs and rework effort and more reliable software. Therefore the defect prediction is important to achieve software quality. Various techniques have been proposed to tackle Software Defect Prediction (SDP) problem. The most known techniques are Machine Learning (ML) techniques.

The ML techniques are used extensively in SDP to predict the buggy modules based on historical fault data, essential metrics and different software computing techniques. The machine learning techniques that can be used to detect bugs in software data sets can be classification and clustering. Classification is a data mining and machine learning approach, useful in software bug prediction. It involves categorization of software modules into defective or non-defective that is denoted by a set of software complexity metrics by utilizing a classification model that is derived from earlier development projects data [1].

Clustering is a kind of non-hierarchical method that moves data points among a set of clusters until similar item clusters are formed or a desired set is acquired. Clustering methods make assumptions about the data set. If that assumption holds, then it results into a good cluster. But it is a trivial task to satisfy all assumptions. The combination of different clustering methods and by varying input parameters may be beneficial.

This paper explores the different machine learning techniques for software bug detection and provides a comparative performance analysis between them. The rest of the paper is organized as follows: Section II provides a related work on the selected research topic; Section III discusses the different selected machine learning techniques, data pre-process and prediction accuracy indicators, experiment procedure and results; Section VI provides the discussion about comparative analysis of different methods; and Section V concludes the research.

II. RELATED WORK

Sharma and Jain [2] explored the WEKA approach for decision tree classification algorithms. They characterized specific approach for classification and developed method for WEKA in order to utilize the implementation of different datasets. The high rate of accuracy is presented and achieved by each decision tree. It correctly classify data into its related instances. The proposed approach can be used in banking, medical and various areas. Their proposed method is generic one not especially for software bug prediction. Various machine learning approaches such as Artificial Neural Network (ANN), Bayesian Belief Network (BBN), Decision Tree, clustering and SVM are some techniques which are generally used for fault prediction in software. Elish and Elish proposed a software prediction model by utilizing SVM approach. A comparative analysis was also performed for SVM against four NASA datasets including eight machine learning models.

Ghouti et al., [3] proposed a model based on Probabilistic Neural Network (PNN) and SVM for fault prediction and used PROMISE datasets for evaluation. This research work suggested that predictive performance of PNN is better than SVM for any size of datasets.

Ruchika Malhotra[4] in this paper they analyses and compares the statistical and six machine learning methods for fault prediction. These methods (Decision Tree, Artificial Neural Network, Cascade Correlation Network, Support Vector Machine, Group Method of Data Handling Method, and Gene Expression Programming) are empirically validated to find the relationship between the static code metrics and the fault proneness of a module. In this they compare the models predicted using the regression and the machine learning methods used two publicly available data sets AR1 and AR6.

Martin Shepperd et al.[5] studied on the publicly available NASA datasets have been extensively used as part of this research to classify software modules into defect prone and not defect-prone categories. In this regard, the Promise Data Repository 2 has served an important role in making software engineering data sets publicly available. For example, there are 96 software defect datasets available. Amongst these are 13 out of the 14 data sets that have been provided by NASA and which were also available for download from the NASA Metrics Data Program (MDP) website.

Saiqa Aleem et al.[6], in this study they were used around fifteen data sets (AR1, AR6, CM1, KC1, KC3 etc) with various machine learning methods. Measured the performance of each method and finally conclude that SVM, MLP and bagging had high accuracy and performances.

Most of the existed studies on software defect prediction are limited in performing comparative analysis of all the methods of machine learning. Some of them used few methods and provides the comparison between them and others just discussed or proposed a method based on existing machine learning techniques by extending them [5, 6].

III. MACHINE LEARNING IN SOFTWARE DEFECT PREDICTION

The field of machine learning has been growing rapidly, producing a variety of learning algorithms for different applications. The ultimate value of those algorithms is to a great extent judged by their success in solving real-world problems. Therefore, algorithm reproduction and application to new tasks are crucial to the progress of the field. However, various machine learning researchers currently publish for software fault prediction model development. Now, we categorizing successful software defect model into three that are based on classification, clustering and ensemble methods. In supervised learning algorithms such as ensemble classifier like Extra Tree Classifier, Xgboost, Passive Aggressive classifier, Gaussian Naive Bayes classifier, Light GBM, Maximum Vote Classifier and Linear SVC are compared. In case of unsupervised learning methods, clustering techniques such as mini-batch K-means algorithm, GMM and Stacking classifiers are compared against each other.
The brief description of each algorithm is as follows:

A. Random Forest (RF)

Random forest is a machine learning algorithm that reports the aggregated result of different classifiers. Random forest algorithm is used for both regression and classification. In Random forest, several classification trees are created and each tree is trained in a bootstrap sample of the original training data. Every tree in the RF will cast a vote for some input x. Then the output of the RF will be determined based on the majority voting of the trees. High dimensional data can be handled by the Random forest classifiers [2]. Random forest is an ensemble classifier technique that includes many decision trees. The output of random forest algorithm is the class that is the mode of the classes output by individual trees.

B. Maximum Voting Classifier

In Majority voting, several classifiers generate predictions for each instance in the testing data. The final prediction for each instance is the one which receives the maximum votes and it more than half of the votes. The pseudocode of the Majority Voting used in the proposed work is as follows:

- Apply 2 classifiers (RF and ET) on training data
- Compare the performance of 2 classifiers
- Performing majority voting for every observation

Majority Vote Based Ensemble Classifier method increases the accuracy by combining the advantages of each individual classifier.

C. Extra Tree Classifier (ET)

The Extra-Tree method (standing for extremely randomized trees) , with the main objective of further randomizing tree building in the context of numerical input features, where the choice of the optimal cut-point is responsible for a large proportion of the variance of the induced tree. With respect to random forests, the method drops the idea of using bootstrap copies of the learning sample, and instead of trying to find an optimal cut-point for each one of the K randomly chosen features at each node, it selects a cut-point at random. This idea is rather productive in the context of many problems characterized by a large number of numerical features varying more or less continuously: it leads often to increased accuracy thanks to its smoothing and at the same time significantly reduces computational burdens linked to the determination of optimal cut-points in standard trees and in random forests.

D. Passive Aggressive Classifier (PAC)

Passive: if correct classification, keep the model; Aggressive: if incorrect classification, update to adjust to this misclassified example. In passive, the information hidden in the example is not enough for updating; in aggressive, the information shows that at lest this time you are wrong, a better model should modify this mistake.

E. Xgboost

XGBoost was mainly designed for speed and performance using gradient-boosted decision trees. It represents a way for machine boosting, or in other words applying boosting to machines, initially done by Tianqi Chen [27] and further taken up by many developers. It is a tool that belongs to the Distributed Machine Learning Community (DMLC). XGBoost or eXtreme Gradient Boosting helps in exploiting every bit of memory and hardware resources for tree boosting algorithms. It gives the benefit of algorithm enhancement, tuning the model, and can also be deployed in computing environments. XGBoost can perform the three major gradient boosting techniques, that is Gradient Boosting, Regularized Boosting, and Stochastic Boosting. It also allows for the addition and tuning of regularization parameters, making it stand out from other libraries. The algorithm is highly effective in reducing the computing time and provides optimal use of memory resources. It is Sparse Aware, or can take care of missing values, supports parallel structure in tree construction, and has the unique quality to perform boosting on added data already on the trained model (Continued Training)

F. LightGBM

LightGBM is a gradient learning framework based on tree learning. The main difference between it and the Xgboost algorithm is that it uses a histogram-based algorithm to speed up the training process, reduce memory consumption, and adopt a leaf-wise leaf growth strategy with depth limitation [7]. The following describes the histogram algorithm and the leaf growth strategy with depth-limiting Leaf-wise.

G. Mini-batch K-means

The algorithm assumed that when K-Means algorithm modifies and optimizes the K-Means by using one-pass over the input data and produces as many centroids as it determines is optimal. Avoiding multiple passes over the input data can have major impacts on running time because just reading large data set can increase the cost in large-scale computations. The basic idea behind this algorithm is that incoming points are assigned to a nearby cluster or used as basis of new cluster. Decision is made according to how distance to nearby cluster compares to adaptive scale parameter,

H. Gaussian Mixture Model (GMM)

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMMs are commonly used as a parametric model of the probability distribution of continuous measurements or features in a biometric system, such as vocal -tract related spectral features in speaker recognition systems. GMM parameters are estimated from training data using the iterative Expectation-Maximizations (EM) algorithm or Maximum A Posteriori (MAP) estimation from a well trained prior model.

I. Stacking Classifier

Stacking is a popular ensembling method in machine learning (Wolpert, 1992) and has been successfully used in many applications including the top performing systems in the Netflix competition (Sill et al., 2009). The idea is to employ multiple learners and combine their predictions by training a meta-classifier to weight and combine multiple models using their confidence scores as features. By training on a set of supervised data that is disjoint from that used to train the individual models, it learns how to combine their results into an improved ensemble model. We employed 3 classifiers: KNN, Adaboost and Random Forest classifier to train and test on all slot types using an Logistic Regression.

J. Gaussian Naive Bayes (GNB)

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values. Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution. This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

K. Quadratic Discriminant Analysis (QDA)

Quadratic discriminant analysis (QDA) is a classical generative probabilistic method for classification problems. Conditional distributions of QDA are assumed to be multivariate Gaussian; posterior distributions are derived via Bayes theorem, and used to classify observations into different classes. Traditionally, parameters of QDA are estimated by maximizing the joint likelihood of observations and their associate class labels.

IV. DEFECT DETECTION FRAMEWORK

A. Framework Overview

A general pattern recognition approach is to perform preprocessing, feature extraction and classification . The steps are as follows:

- Step 1: Preprocessing
- Step 2: Feature Extraction
- Step 3: Classification

B. Preprocessing

Preprocessing shapes data into a form more usable by the classification engine. If the input data is an image, preprocessing may include translating or rotating an image to place it in a standard position and orientation or sharpening of the image to simplify the feature selection processing. In other applications where vectors or records of data are inputs, the preprocessing step may filter out inputs based on some a priori criteria or a statistical property of the overall dataset. Further, preprocessing may fill in missing data elements or normalizing numeric data.

The datasets from PROMISE data repository [39] were used in the experiments. Table 1 shows the information about datasets. The datasets were collected from real software projects by NASA and have many software modules. We used public domain datasets in the experiments as this is a benchmarking procedure of defect prediction research, making easier for other researcher to compare their techniques [13, 8]. Datasets used different programming languages and code metrics such as Halsteads complexity, code size and McCabes cyclomatic complexity etc. Experiments were performed by such a baseline.

The machine learning framework scikit-learn and Python Programming was used for data analysis and investigations.

C. Feature Extraction

Feature extraction transforms pre-processed data into a form usable by the pattern recognition engine. Creating a form that is optimized to a given machine-learning algorithm is fundamental to the application of such technology to software engineering data[8]. Pattern recognition algorithms are quite sensitive to the form of data provided them. We used Tree based algorithm for feature selection.

D. Classification

Given that the translation problem can be solved, many classification algorithms exist today that can be customized to the task of machining source code tokens, fragments or flows to defect classes. Different classifiers have different strengths and weakness that may fit a certain needs. All the above mentioned algorithms were used for classification and their performances were measured based on the accuracy matric:

- Accuracy: Under normal circumstances, for the two classification labels True and False, there are definitions as follows:
  Acc= (TP+TN)/(TP+TN+FP+FN)
  Accuracy (Prec) is used to represent the general characteristics of the classifier. Accuracy is the percentage of cases that are marked as positive and indeed positive.

V. RESULT

For comparative performance analysis of different machine learning methods, we selected 15 software bug datasets and applied above mentioned machine learning methods. The accuracy scores of each algorithm for each datasets are mentioned in the following tables:

TABLE I
PERFORMANCE OF SUPERVISED MACHINE LEARNING METHODS WITH CROSS VALIDATION TEST MODE BASED ON ACCURACY

Dataset	RF	ET	MVC	PAC	QDA	LGBM	GNB	XGB
AR1	87.63	91.79	92.63	89.29	91.79	92.3	77.56	92.56
AR6	85.67	82.74	86.78	72.73	87.69	72.73	84.35	84.56
CM1	86.24	85.66	86.78	76.87	84.73	81.82	81.32	85.02
JM1	80.84	79.14	80.83	71.15	79.93	80.80	80.44	78.1
KC1	85.38	83.2	85	76.45	81.25	81.52	82.34	82.87
KC2	81.27	78.78	83.05	62.32	82.29	66.04	82.85	79.24
KC3	83.41	80	89.96	65.02	86.43	89.13	86.24	90.36
MC1	97.69	97.99	97.69	92.64	97.18	97.49	97.91	97.84
MC2	72.71	72.7	72.74	67	75.21	70.59	72.7	71.49
MW1	97.56	97.99	97.64	92.64	97.18	97.49	92.91	97.84
PC1	92.97	93.43	92.97	78.02	90.27	92.79	89.19	92.26
PC2	97.46	97.86	97.86	93.3	97.86	97.33	91.5	97.45
PC3	87.46	87.84	87.65	52.90	47.34	85.19	20.09	87.01
PC4	89.20	90.37	89.04	76.89	50.59	85.28	86.01	90.75
PC5	97.18	96.95	97.24	95.67	95.53	96.97	97.14	96.64
Mean	88.18	87.76	89.19	77.53	83.02	85.83	81.5	88.26

TABLE II
PERFORMANCE OF UNSUPERVISED MACHINE LEARNING METHODS WITH CROSS VALIDATION TEST MODE BASED ON ACCURACY

Dataset	Minibatch K-means	GMM	KNN	SC
AR1	90.06	84.17	85.43	91.8
AR6	60.02	63.55	76.6	85.67
CM1	68.05	87.176	81.54	84.7
JM1	37.63	80.66	71.3	78.1
KC1	83.96	68.7	78.89	84.06
KC2	68.37	79.51	60.37	81.73
KC3	83.75	90.62	84.95	89.72
MC1	32.4	97.69	97.18	97.84
MC2	58.8	64.4	60.31	72.11
MW1	32.40	97.69	97.18	97.84
PC1	55.1	93.06	89.01	93.33
PC2	86.66	88.39	95.17	97.72
PC3	80.71	87.56	80.13	87.65
PC4	55.9	68.77	79.88	90.13
PC5	59.67	97	95.65	97.09
Mean	57.49	83.26	82.24	88.63

Table 1 and 2 shows stratified 10- fold cross validation test mode accuracy for selected datasets.

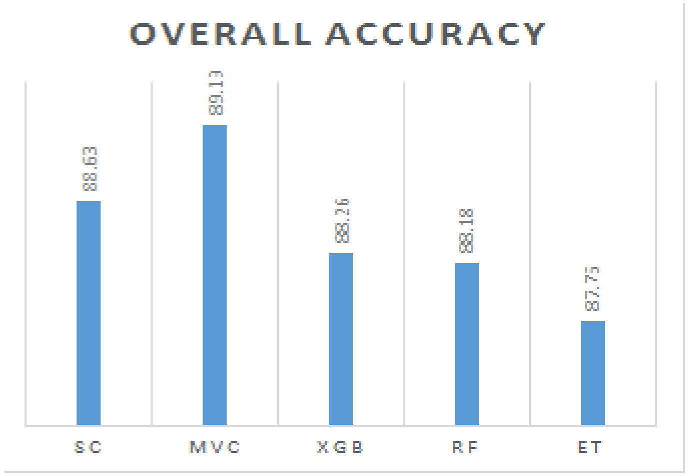

fig:Top 5 algorithms based on average accuracy

Above figure shows the top 5 highest performing algorithms based on their average performance

VI. CONCLUSIONS

This paper presents a survey of various machine learning techniques for software defect predication. From the survey, it can be observed that software defect is indeed a major issue in software engineering. Software defect module prediction using different machine learning techniques is to improve the quality of software development process. By using this technique, software manager effectively allocate resources. This study covered the different machine learning methods that can be used for a bugs prediction. The performance of different algorithms on various software datasets was analysed. Mostly Stacking Classifier, Voting classifier and Random forest techniques performed well on bugs datasets. In order to select the appropriate method for bugs prediction domain experts have to consider various factors such as the type of datasets, problem domain, uncertainty in datasets or the nature of project.

REFERENCES

[1] Sandeep D and R. S, ”Case Studies of Most Common and Severe Types of Software System Failure ” International Journal of Advanced Research in Computer Science and Software Engineering vol. 2, pp. 341-347 August 2012

[2] T. C. Sharma M. Jain (2013) WEKA approach for comparative study of classification algorithm, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 4, 7 pages.

[3] H. A. Al-Jamimi L. Ghouti (2011) Efficient prediction of software fault proneness modules using support vertor machines and probabilistic neural networks, 5th Malaysian Conference in Software Engineering (MySEC), IEEE Press, pp. 251-256.

[4] R. Malhotra, Comparative analysis of statistical and machine learning methods for predicting faulty modules,Applied Soft Computing, vol. 21pp. 286-297 2014.

[5] Saiqa A, Luiz F, and F. A, ”Benchmarking Machine Learning Techniques for Software Defect Detection,” International Journal of Software Engineering Applications, vol. 6, pp. 11-23, May 2015.

[6] M. SURENDRA and D. N. GEETHANJALI, ”Classification of Defects In Software Using Decision Tree Algorithm,” vol. 5, pp. 1332-1340, June 2013.

[7] Nguyen V, ”The Application of Machine Learning Methods in Software Verification and Validation ” MSc, Master of Science in Engineering, The University of Texas at Austin, Texas 2010.