Dissertation Report

on

Analysis of Ensemble Models for Aging Related Bug Prediction in Software Systems

Submitted By:

Shubham Sharma

Enrollment No.: 16535040

Submitted in the partial fulfillment of the

requirements for the award of the degree of

of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Under the guidance of

Dr. Sandeep Kumar

Assistant Professor

Department of Computer Science and Engineering

Indian Institute of Technology

Roorkee

May, 2018

Declaration

I declare that the work presented in this dissertation with title **"Analysis of Ensemble Methods for Age Related Bug Prediction in Software Systems"** towards fulfillment of the requirement for the award of the degree of **Master of Technology** in **Computer Science & Engineering** submitted in the **Department of Computer Science & Engineering, Indian Institute of Technology Roorkee, India** is an authentic record of my own work carried out during the period of **May 2017 to May 2018** under the supervision of **Dr. Sandeep Kumar**, Assistant Professor, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India. The content of this dissertation has not been submitted by me for the award of any other degree of this or any other institute.

Date: ....
Place: ROORKEE

SHUBHAM SHARMA
(16535040)
M.TECH (CSE)

Certificate

This is to certify that the statement made by the candidate in the above declaration is correct to the best of my Knowledge and belief.

Date: ....
Place: ....

Sign: ....
Dr. Sandeep Kumar
(Assistant Professor)
Indian Institute of Technology
Roorkee

Abstract

With the evolution of the software industry, the growing software complexity led to the increase in the number of software faults. According to the study, the software faults are responsible for many unplanned system outages and affects the reputation of the company. Many techniques are proposed in order to avoid the software failures but still software failures are common. Software fault prediction is the process, which predicts about the fault proneness of the software module. This process is based on some previous data (if available) and certain learning models. The prediction of the accurate location of faults can boost the testing process and allows the developers to focus on the critical modules that may account for the maximum number of faults. Many software faults and failures are outcomes of a phenomenon, called software aging. In this work, we have presented the use of various ensemble models for development of approach to predict the Aging Related Bugs (ARB). A comparative analysis of different ensemble techniques, bagging, boosting and stacking have been presented. The experimental study has been performed on the LINUX and MYSQL bug datasets collected from Software Aging and Rejuvenation Repository.

Acknowledgement

Dedicated to my family and friends, for standing by me through thick and thin, without whom i would not have gotten this far. I would like to express my sincere gratitude to my advisor **Dr. Sandeep Kumar** for the continuous support of my study and research, for his patience, motivation, enthusiasm and immense knowledge. His guidance helped me in all time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my study. I would like to express my sincere appreciation and gratitude towards **Mr. Kanishk Lohumi** for his encouragement, consistent support and invaluable suggestions at the time I needed the most.

I am also grateful to the Department of Computer Science and Engineering, IIT Roorkee for providing valuable resources to aid my research.

SHUBHAM SHARMA

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software engineering is a tedious process composed of developers and testers along with stringent deadlines and constrained by limited budget [2]. Delivering precisely, as required by customers involves minimizing the defects in the programs. Hence, it becomes crucial to impose the high quality measures in the early phase of software development life cycle. In Software industry, we routinely come across various software projects containing inconsistencies and errors which proves to be very ineffective and led to exceeding of proposed cost and time.Software fault prediction process is very important in software development and the accurate prediction of faults and the recognition of the area which is most prone to fault occurrence can directly help in reducing the development cost, testing efforts and improves the overall quality of the software. Software a fault is the main concern to be dealt with that affects overall software reliability and correctness. The accurate predictions of the faults empower the software developers to evaluate the overall reliability of the software during the development process. Moreover, the prediction of the accurate location of faults can boost the testing process and allows the developers to focus on the critical modules that may account for the maximum number of faults.

As our reliance on the software systems are continuously increasing in order to manage the ordinary and critical tasks in our life, the complexity of the software systems are growing so as to manage the ever changing requirements and parallely maintaining the performance and user friendliness. As the complexity of software is growing, their management is getting difficult. Management of these requires more skilled administrators and developers which accounts for a significant amount of the total cost.

As the complexity of the software system are growing continuously, the rate of software failure is also increasing resulting in undesirable behaviour of the system along with poor services and sometimes complete outages. Dealing with software faults is very important task. Faulty modules present in software deteriorates the quality of the software and also increases the overall cost of the software system [3]

It is evident from the various studies that most of the software faults are more responsible for software outages as compared to hardware faults [4, 5]. The presence of sophisticated developing, testing and testing tools have made it easier to avoid software faults, still the occurrence of software faults are significant and affects both quality and performance of the system.

Various studies [6, 7] indicated that in many cases software aging phenomenon is responsible for many software failures. Software Aging describes many potential state of system such as accumulation of errors, in long running applications, such as web applications, leading to hang or crash of application/software [8]. Software aging could also lead to gradual performance degradation, and also related to memory leaks, memory bloating, data corruption, unreleased file-locks, unterminated threads and overruns. Software aging is evident in any type of software such as web-servers, spacecraft systems [9], and military systems [10] and led to loss of lives as a consequence in many instances [10].

It is very necessary to identify the aging phenomenon in the early stage of the software development, in order to prevent the damage and extra cost needed to cope up the software aging effect at the later stages of software development. The sooner these age related bugs are predicted, the lower it will cost to remove or modify these bugs. By predicting these bugs, we can make software testers to focus on these bugs by giving these bugs priority. The objective of the prediction of these aging related bugs is to make them available to the tester, even before the software testing phase takes place.

Software rejuvenation is the mechanism used to counter the effects of the software aging. It is very effective in nullifying the effect ARBs (Aging Related Bugs) during runtime phase [11, 12] but more cost and effects can be saved by using these policies in advance [13]. Software Rejuvenation techniques are broadly classified as Time-based and Prediction-based rejuvenation.

Rejuvenation policy is applied after regular time intervals, in case of time-based approach. Predictive rejuvenation approach is based on continuous monitoring of system metrics and rejuvenation is applied if the aging effect leads to hanging or crash in a system. This approach has an advantage over time-based approach, as we have to apply the rejuvenation polices only when needed in case of the predictive approach, while we regularly apply rejuvenation, irrespective of whether needed or not.

We may take necessary preventive measures, bur our ability to understand the software and predict its future is uncertain and we can approximate it to certain level only. Over the period of time, we will somehow make certain assumptions that will be inconsistent with our initial assumptions. Documentations even with much precision, cannot be perfect. Reviewers may find out the issues missed by the designer, but there will surely be the issues which will be missed by the reviewers too. Preventive measures are effective but cannot eliminate the aging completely [14].

In this work, we have presented the ensemble model to predict the ARBs (Aging Related Bugs) which are responsible for the software aging phenomenon and its evaluation on the previous dataset available from software aging and rejuvenation repository. Preliminarily, we evaluated various machine learning techniques to predict the bugs and compare the performance of these basic machine learning techniques. Then further we created ensemble models using bagging, boosting and stacking on the best performing ML techniques. We evaluated and compared these ensembles among themselves and with the basic ML techniques. Ensemble methods can boost the performance of various machine learning techniques.

The paper is organized as follows: Section 2 presents the background, while section 3 discusses the related works done regarding software aging. Section 3 describes the ensemble based learning approach in detail. Section 4 describes the experimental set up to predict the accuracy. Further, results are discussed in Section 5 and Section 6 concludes the paper.

Chapter 2

Background

2.1 Software fault prediction process

Fault prediction process basically involves two steps: machine training and prediction of fault.During training phase, a fault prediction model is developed using the previous metrics and data available from previous versions of software by using certain kind of machine learning models.In the testing phase, the fault prediction model is used to predict the fault proneness of modules in the software being developed.

The prediction of the accurate location of faults can make the developers to focus on the critical modules that may account for the maximum number of faults. Software fault prediction techniques help in recognizing the faulty modules even before the testing phase (during early phases of software development cycle).

Several software defect prediction models, based on various machine learning models have been proposed [15, 16], But no learning model can accurately and consistently have been able to predict the faulty modules. Research are still going on to find the best suitable learning model or software metric which can accurately predict the faulty modules for all type of datasets. For varying dataset, the different learning models behaves differently and give varying efficiency. Thus, no learning model can said to be reliable for every form of dataset.

2.2 Software Aging

Programs, like people, get old. We cant prevent aging, but we can understand its causes, take steps to limit its effects, temporarily reverse some of the damage it has caused, and prepare for the day when the software is no longer viable [17].

Software aging refers to the inevitable process which constitutes all softwares tendency to fail, or cause a system failure after running continuously for a certain time. With the passage of time, software becomes less immune and less adaptable and stops functioning eventually, and requires rebooting or reinstalling, but it cannot work as long time fix. Software rejuvenation is the proper process, a proactive fault management method to deal with the software aging incident [17].

2.2.1 Causes of Software Aging

There are two causes for the software aging:

1. Products owners failure to modify itself to meet the changing needs.
2. The changes that are made in order to maintain the software efficiency, somehow damages the software design.

Lack of Movement

Due to changing and ever evolving technology, the software needs to be agile and must be able to cope with the changing needs of the users. Failing to do so, will lead to the dismay at user experience and loss of customers. Unless the software is frequently updated, its users may feel dissatisfied and may move to the newer products, as soon as the benefits of the other product feels to be significant in comparison to the cost of retraining and converting to the newer products [17].

Ignorant Changes

It is necessary to change and upgrade the software to prevent aging, but changing the software without proper care may cause different form of aging. It is necessary to understand the original design concept before making any changes to the persisting software. Understanding the concept of design states understanding the interfaces used within and between the system and the execution environment in which it operates [17].

Changes made by the operator who are unaware about the design concept, almost always degrades the structure of the program. Under such conditions, the changes are not at par with the original design and concept and will invalidate or vary from the original concept. Sometimes the damage due to these ignorant changes are small but often these are quite severe.

In order to understand the system, after these changes, one must have knowledge about both the original design concept and new additional exceptions introduced to the design rules. The original designer of the software are unable to understand the system, after many such changes are introduced in he system. And the people who made these ignorant and inconsistent changes, never really understood the software. So, overall no one understands the product. Software which is updated this way, repeatedly, becomes very expensive to update. Changes to such software take longer time and may introduce new bugs.

2.2.2 Cost of Software Aging

Aging in software corresponds to the human aging. It mirrors to the various problems faced due to old age in humans like:

1. It becomes very difficult for the owner of the product to keep up with the needs of the consumers and market and might lose some of their customers to newer products with better performances.

2. Due to continuously deteriorating structure, the space and time performance of the product degrades significantly.

3. Aging software might turn buggy, due to introduction of errors in the software while making changes in the system to cope up with the aging effect.

Each of the above factors turns out to be a real cost to the owner [14].

Inability to keep up

As a result of the software aging, with time, its size increases. This weight gain or size gain is due to the fact thatany new feature added to the software by adding additional code. It is difficult to modify the existing code in order to add new functionalities or handle new situations, as the code is not well-understandable and also not well-documented. As the program size increases, by a magnitude of two or more, over the period of time, it become difficult to adapt changes and it is very troublesome to keep up the pace with changing needs. It is not practically feasible to fully verify whether a software is bug free or not [14].

Reduced Performance

As the program size grows, it requires more memory usage and have high resource demands and led to more delays due to increased swapping from the mass storage. More the size of programs, more CPU cycles and more memory space are required for processing. The program respond slowly and must have an upgradation to give acceptable response. Poor design, as a result of changes made, for several years also decreases the performance. Thus a requirement for a newer product with better efficiency arises [14].

Decreasing Reliability

Due to the introduction of various errors during the changes made in the software over a period of time, makes the product buggy and less-reliable [14].

Many of the research till now have focused on the challenges regarding the software's first release. Many research focused on the management issues (i.e. configuration management and control). However, software aging requires much more than the patient management. An efficient engineering is required.

Reducing the cost of aging software

1. Design with change adaptiveness
2. Second opinions reviews

Why Software Aging is Inevitable?

We may take necessary preventive measures, our ability to understand the software and predict its future is uncertain and we can approximate it to certain level only. Over the period of time, we will somehow make certain assumptions that will be inconsistent with our initial assumptions. Documentations even with much precision, cannot be perfect. Reviewers may found out the issues missed by the designer, but there will surely be the issues which will be missed by the reviewers too. Preventive measures are effective but cannot eliminate the aging completely [14].

Chapter 3

Literature Review

In [18], a study has been made to analyze the Service Level Objective (SLO) violations. It proposed an off-line framework that performs analysis of these SLO violations. It uses TANs (Tree Augmented Nave Bayesian Networks) to find out the most correlated resources which are responsible for performance behavior change.

Andrzejak et al. [19] presented an empirical study for prediction of software aging using SVM, Nave Bayes and decision trees techniques. However, the testing was limited and the authors didnt explore their approach for the cases involving multiple resources and in dynamic environment. In [20], a study regarding critical events in large clusters was made and a framework was presented to predict such events. It stated that prediction techniques differ based on elements to be predicted. Some other works such as [21, 22] made a study on workload and resource exhaustion in a system suffering from aging using the various predictive machine learning algorithms.

A study on M5P machine learning algorithm mentioning the dynamic nature of software aging was presented in [23]. The prediction model used M5P algorithm to test a J2EE web application and evaluated the prediction accuracy for the same, with all types of datasets, specifically complex and smaller ones. It also presented an approach to determine the root cause for the failure by using M5P machine learning algorithm. It provided a model, which had significant improvement in prediction regarding the dynamic scenarios of software aging.

A study [24] proposed a method to derive an accurate and optimized schedule for rejuvenation of a web server (Apache) by using Radial Basis Function (RBF) based

Feed Forward Neural Network, a variant of Artificial Neural Networks (ANN). Aging indicators are obtained through experimental setup involving Apache web server and clients, which acts as input to the neural network model. This method is better than existing ones because usage of RBF leads to better accuracy and speed in convergence.

In another work, Cotroneo et al. [25] tried to find the aging prone source code files and predicted the aging related bugs using predictor variables. Furthermore, Qin et al. [26] presented a TLAP (Transfer learning based aging bug prediction) approach which provides cross project prediction using transfer learning among different projects. It also presented a class imbalance mitigation strategy to counter the effects of class imbalance. It used the cross project approach, where the dataset of multiple project was used to develop and train a prediction model that can be used to test any of these projects. TLAP model showed significantly improved prediction as compared to other prediction models.

In the work [27], a study on the application of static source code metrics and machine learning techniques to predict aging related bugs was presented. A a series of experiments on publicly available dataset from two large open-source software systems: Linux and MySQL. Class imbalance and high dimensionality are the two main technical challenges in building effective predictors for aging related bugs. An investigation of five different feature selection techniques (OneR, Information Gain, Gain Ratio, RELEIF and Symmetric Uncertainty) for dimensionality reduction and SMOTE method to counter the effect of class imbalance was presented in this machine learning based research work. The author used Extreme Learning Machines (ELM) with three different kernels (linear, polynomial and RBF) and presented experimental results which demonstrated the effectiveness of the approach.

In Cotroneo et al. [17], various works in the field of software aging and rejuvenations were surveyed and it highlighted the aspects that requires special attention and the important trends observed in various experiments performed in software aging and rejuvenation field. It also highlighted the various aging indicators which should be focused during prediction. It also provided an insight into the software aging and rejuvenation field and tried to provide a direction for further study.

It is evident from the works such as Misrili et.al. [28], Wang et.al. [29], Khoshgoftaar et.al. [30] and Aljamann et.al. [31] that ensemble methods helps in achieving better prediction accuracy in comparison with individual machine learning techniques. But, all these works are used in the field of software fault prediction in order to predict the software modules to be faulty or non-faulty. But as per the study by Cotroneo et.al [17], the ensembles methods are rarely used in the field of software aging to find the aging related bugs in the software system and is not fully explored. Thus, it is required to construct and evaluate ensemble methods for finding the aging related bugs.

In work [14] have explored the use of few ensemble methods for a learning technique for ARB prediction for imbalanced dataset. It used adaboost algorithm[32] along with other machine learning techniques and techniques for imbalanced datasets, but still the use of ensemble learning based model is very rare. There is a need to study the behaviour of various ensemble methods for different learning models for ARB prediction.

This work presents the ensemble methods for predicting the aging related bugs in the software system.

Chapter 4

Experimental Setup

4.1 Prediction Process using Ensemble Learning

"The aim of this work is to model an approach based on ensemble learning to predict the location of Age related bugs, which improve upon the previous work done."

In this section, we present an ensemble learning based approach, which aims to find the aging related bugs using ensemble models like bagging boosting and stacking. It explores the dependence of bugs present and software metrics used.

Generally, the dataset related to software aging suffers from class imbalance problem, as the number of bugs belonging to ARB class is very less in comparison to other class. So, we perform the class imbalance mitigation procedure to counter the effect of class imbalance problem [33].

After the class imbalance mitigation procedure, We used the 5- fold split and perform the classification procedure. We split the data into five parts according to bug proportion and then assign four parts for training the model, while one part is used for validating the model. Each case of experiment is done five times and the mean value of output is used as the final output, in order to avoid any occasional error.

IEEE Software Aging and Rejuvenation Bug repositories are used to collect the data [34].

Our main target for prediction is to do prediction at the level of a file with reference to its proneness for ARB. We have to predict whether the file is ARB (Age related bug) prone or ARB-free. Samples are classified into two classes. We are concerned with the presence of ARBs in a file, not their count. Even if file contains one ARB, it is considered as ARB prone, else it is considered as ARB free. Various machine learning approaches are used for the prediction of bugs in the dataset. Some of these techniques are Nave Bayes, SVM, Logistic Regression, KNN, MLP and Decision Tree, MLP. In this section, we have discussed these techniques in chapter 5. We have applied three ensemble methods on each of the machine learning techniques mentioned above to predict the ARB prone files. The ensemble methods used to predict the aging related bugs are Bagging, Boosting and Stacking. In this section, we have discussed these methods in chapter 6.

The flowchart depicting the above approach is given in fig 1.

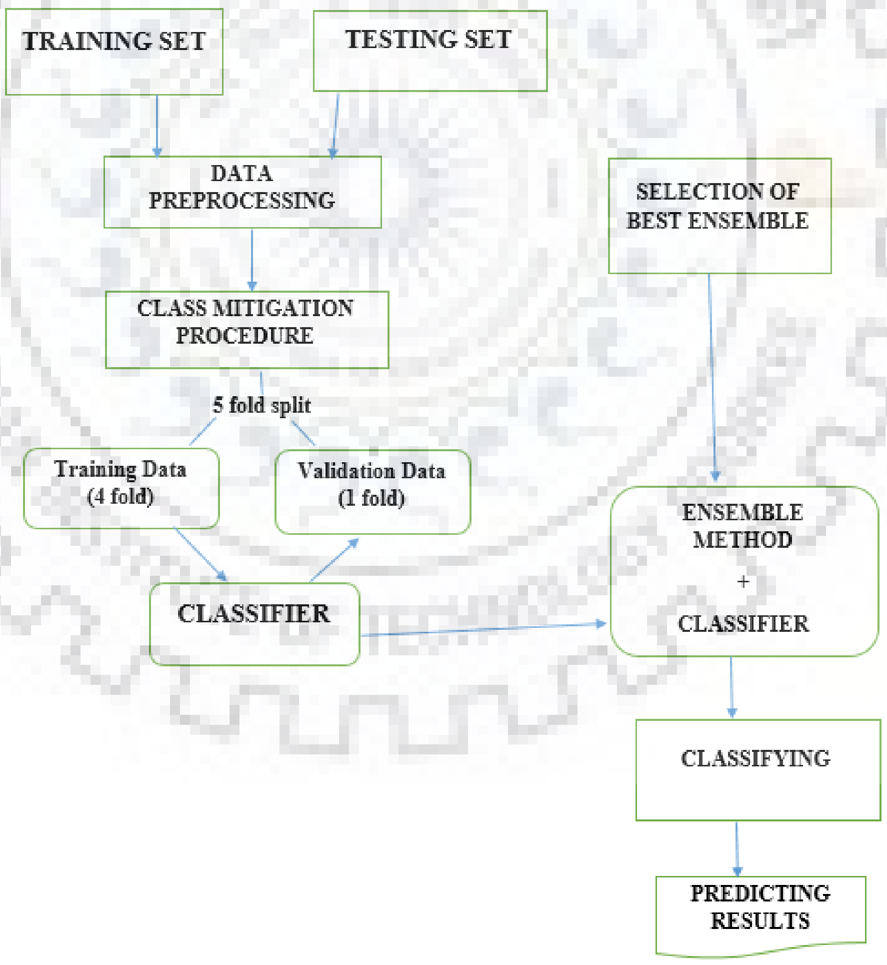


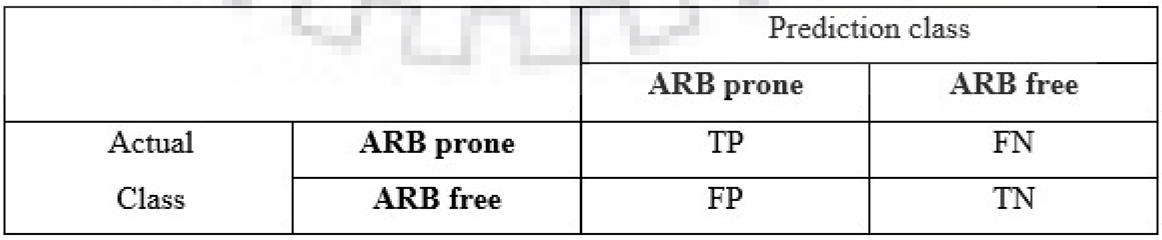
FIGURE 4.1: Prediction Process using Ensemble learning

This work comprises the following steps:

1. Data preprocessing
2. Class Imbalance Mitigation Procedure
3. Empirical analysis of common machine learning techniques in the field of software fault prediction by executing them on Aging related bug prediction dataset.
4. Selection of top three classifiers.
5. Empirical analysis of the various ensemble methods used in the software fault prediction field.
6. Selection of the best ensemble model.
7. Performing ensemble learning with the various selected ensemble model for the Aging related bug prediction.
8. Comparison of various techniques and ensemble models used for prediction in this work.

4.2 Evaluation Metrics

There are several performance measures to evaluate the performance of the classifier and some of the most common performance measures are: Accuracy, Recall, F-measure, and Precision.



		Prediction class	
		ARB prone	**ARB free**
Actual Class	**ARB prone**	TP	FN
	ARB free	FP	TN

Figure 4.2: Confusion Matrix

We have used confusion matrix to define the above mentioned performance measures.
TP: True Positive,
TN: True Negative
FP: False Positive,
FN: False Negative.

Accuracy - Accuracy performance measure calculates the number of data points correctly predicted. It is calculated by the equation-4.1;

$$Accuracy = \frac{(TP + TN) * 100}{TP + FP + TN + FN} \tag{4.1}$$

Recall (True positive rate (TPR)) - It calculates the fraction of positive class data points predicted as positive. The equation-4.2 is used to calculate it;

$$Recall(TPR) = \frac{TP}{TP + FN} \tag{4.2}$$

Precision - Precision is the fraction of predicted positive class that is actually positive class. It is calculated by equation-4.3:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

F-Measure - F-measure depends on recall and precision and is defined by equation-4.4:

$$F - Measure = \frac{2 * Recall * Precision}{Recall + Precision} \tag{4.4}$$

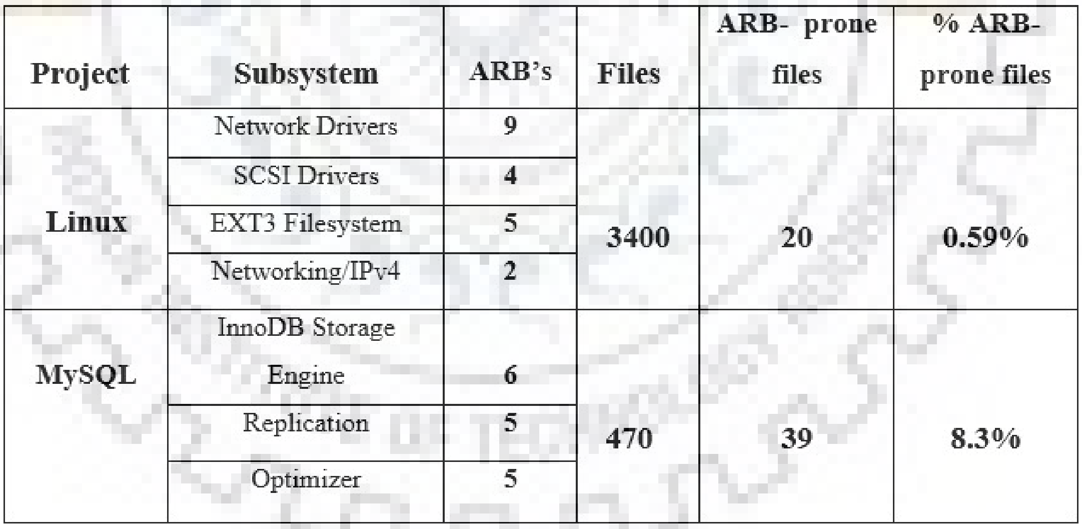
4.3 Class Imbalance Mitigation Procedure

Class imbalance is one of the major challenges and it represents the cases where the examples in one class is very less as compared to other classes. The class with higher size of data is majority class, while the class with smaller size is considered as minority class.

Minority class is unable to draw attention of classifiers in comparison to the majority classes.
Let us assume that D is a dataset with N samples and out of it a part of dataset N_{min}, includes the examples belonging to the minority class (ARB prone class in our case) with size N_{min}, and another part of dataset, D_{max} contains the N_{max} examples belonging to the majority class (ARB free classes).

We calculate $R = N_{max} / N_{min}$, termed as size ratio to calculate the relation between the size of minority and majority class. Consider that O_{min} and O_{max} are the sizes of the parts of datasets after the mitigation process. During mitigation, we keep the majority class as it is, such that size of O_{max} equals N_{max} while we change the minority class by enlarging it such that size of $O_{min} = R * N_{min}$. Thus, both the majority and minority classes are now equally attended and considered by the classifier [26]. For this experimentation, we have used class imbalance mitigation procedure for handling the class imbalance problem.

4.4 Dataset and Tools Used



Project	Subsystem	ARB's	Files	ARB- prone files	% ARB-prone files
Linux	Network Drivers	9	3400	20	0.59%
	SCSI Drivers	4			
	EXT3 Filesystem	5			
	Networking/IPv4	2			
MySQL	InnoDB Storage Engine	6	470	39	8.3%
	Replication	5			
	Optimizer	5			

FIGURE 4.3: Dataset Used

An overview of software aging datasets are shown in Figure 4.2[25].

We have used two real software system datasets: Linux and MySQL as described in in the Table [25]. We have used aging-related metrics [7] along with Halstead and McCabe complexity metrics. Tai et al. [9] provide the details about all these metrics.

Tools Used: All the implementation in this work have been done in Python programming language version 3.5.0.

Chapter 5

Empirical Analysis of Learning Techniques for ARB prediction

Machine Learning methods are generally used in many applications to solve different classification problems.Different machine learning methods and their learned classifiers may have different assumptions about the data, thus the various Machine Learning algorithms or other learning models, behaves drastically different with different datasets, because of the different properties of different samples and different assumptions and reaches to different performance with different dataset. And an algorithm which performs efficiently with one dataset may have varying result with another.

5.1 Prediction Techniques Used

In the field of software fault prediction, learning models plays an important role and various machine learning techniques are available to train the machine with previously available dataset, which are later used for prediction.

In our work, we have implemented the following machine learning methods:

1. **Naive Bayes**
   Naive Bayes is one of the most widely used machine learning method for classification.Nave Bayes Classification is a classification technique, which works

on the independence between the predictor variables. It uses Bayes theorem, which finds conditional probabilities among variables. A Nave Bayes classifier considers that the various features and checks their contribution independently to the probability of occurrence of any event. [35, 36].

Nave Bayesian model is very useful in case of very large datasets and is very easy to build. It can outperform all other classification methods, even those which are highly sophisticated. Nave Bayesian is known for its simplicity and easy implementation.

The posterior probability P(c/x), provided P(c) and P(x) are given, using Bayes theorem, is mathematically written as

$$P(C_j|M) = \frac{[\prod_{i=1}^{m} P(M_i|C_j)] * P(C_j)}{P(M)} \tag{5.1}$$

where m represents the number of features used for classification, while M represents all the features after class mitigation procedure, such that M_i represents the ith feature, and C represents the various classes used for classification such C_j represents the jth class.The sample belongs to the class having maximum value of P(C_j—M), using the above classifier.

2. **Logistic Regression**

It is a statistical model which uses fitting of logistic curve to the dataset. It is generally preferred when the outcome variable or target variable is dichotomous. It is used for estimation of discrete values based on the independent variables given.It assigns the predictor variable with the value between 0 and 1 using sigmoid function, thus avoiding too small or too large values.

It gives output in terms of probability and due to this the adjustment of classification threshold is adjustable in case of logistic regression. It uses statistical analysis and generally can be used for both two class and multi class problem.

Mathematically, Logistic Regression can be defined as follows:

$$P(M) = \frac{1}{e^{-(a_0+a_1M_1+....+a_mM_m)}} \tag{5.2}$$

where m is the number of features used in the classification process, and M_i (for i=1 to m) represents the ith feature, while a_i (for i=1 to m) represents the coefficient of the respective feature.The prediction value P(M) decides the class to which the sample belongs. For two class problem, if value of P(M) is greater than 0.5, it is ARB prone, else it is ARB free [36].

3. **Decision Tree**

   In this classification, we divide the dataset or sample space into two or more homogeneous data sets (or sub-sample sets) based on most significant classifier / differentiator provided implicitly in input variables [12].It works well with both the discrete, categorical and continuous input and output variables.

   In Decision Tree classifier, we use weighted tree concept, where the internal nodes are marked with featured used for classification and edges of the tree created are marked as trial with dataset weight. Tree leaves are named by categorization.By this way, whole document can be categorized from root till the leaf node is reached, moving through the branches. Learning in decision tree adopts a decision tree classifier, which maps information of an item to conclusions of that items expected value [35, 36].

4. **Multilayer Perceptron (MLP)**: Multilayer Perceptron (MLP) or neural network (NN) is a machine learning algorithm that works on the principle of biological neural network (Kotsiantis, 2007). It consist of series of processing layers interconnected, with each connection possessing some weight. During training, based on the knowledge of domain, it develops a representation that maps input space to output space. MLP uses supervised learning technique called back-propagation to train the network.

   The working of MLP can be described as follows:
   For each iteration, the training data are iteratively fed into the neural network and the output obtained is compared with the desired output and error is calculated, which is used to update the hidden layer weights and re-feed the network. The updation is done ensuring that error decreases after each iteration and the output obtained is closer to the desired output.[35, 36].

5. **K Nearest Neighbour (KNN)**: It is a classification algorithm, which is non-parametric in nature. It selects an unlabelled dataset point and assigns it to the nearest set of already classified set of points by majority vote of its nearest neighbours It is well suited for multi-modal classes, but can perform on two class problem too. It needs many iterations and is lazy in nature.

The rule used for classification is independent of joint distribution of sample points and classification, but efficiency is dependent on selecting a good value of k, where k refers to the number of nearest neighbour points used for voting [36].

6. **Support Vector Machines**: Support vector machines (SVMs) are linear classifiers which is generally used for two class problem. Its working is based on margin maximization principle. It classifies the data optimally into two categories by constructing hyperplanes in higher dimensional space. The idea of SVM is to select the hyperplane that separates the two classes with maximum margin from the hyperplane (Adankon et.al, 2009).
   We plot each sample as a point in n-dimensional space, and the values of features represents the particular coordinate. These co-ordinates are called as Support Vectors [36].

5.2 Empirical Study

5.2.1 Experimental Design

We have evaluated the dataset with six prediction models using six different learning techniques discussed above. The performance was measured for three combination of dataset and the evaluation metrics are the average output of all the iteration for each combination of the dataset.

We have used 5-fold cross validation to build and validate the prediction models. Out of the dataset, 80 percent of the total dataset is used for the training of the data, while 20 percent of the data is used for the testing purpose. Each time data is shuffled randomly and then it is divided into the training and testing data.

The experiments were conducted for 5 iterations and the value used are the average result of all the five iteration.

5.2.2 Results and Discussion

Below are the comparative analysis of various classifiers along with the dataset and their combinations. The performance of various classifiers with respect to various classifiers are as follows:

Technique	**Accuracy**	**Recall**	**Precision**	**F-measure**
SVM	**0.9891**	**0.9892**	0.8461	0.8523
DT	0.9597	0.9654	**0.9545**	**0.9484**
NB	0.6894	0.8496	0.4620	0.5994
MLP	0.7173	0.6476	0.9262	0.7849
KNN	0.9193	0.8545	0.9509	0.8882
LR	0.6296	0.5789	0.9023	0.7823

TABLE 5.1: Empirical Study of Machine Learning Techniques

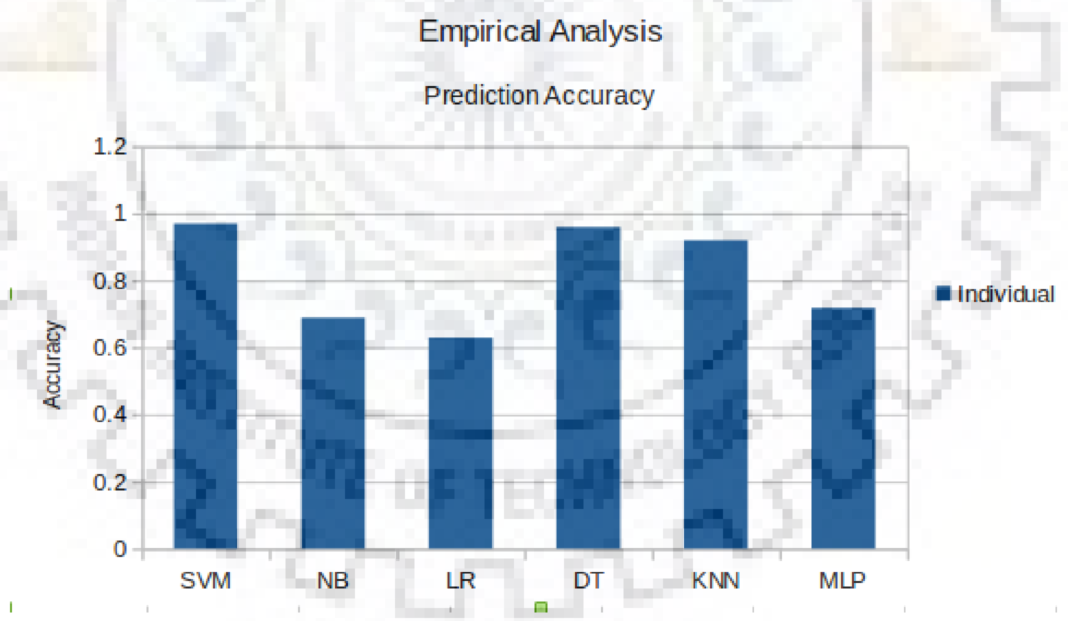


FIGURE 5.1: Empirical Analysis of Individual ML techniques

From the analysis of the above results following conclusions can be drawn:

1. Results suggest that the accuracy for the machine learning techniques used are high (> 0.9 for many cases and generally > 0.7).

2. Recall (True Positive Rate) for the classifier used is generally varying for different classifier.
3. The classification accuracy is quite high for most of the classifier.
4. The process is able to classify the imbalanced dataset correctly after the mitigation procedure used.
5. Generally, the machine learning procedures have low F-measure.
6. For all the combination of dataset, the classification accuracy of SVM is very high (>0.95 in most of the cases) and it predicts best out of all the classification techniques used.
7. SVM performed best with prediction accuracy of more than 0.95 for all the three combination of dataset.
8. Decision tree also have high performanace with prediction accuracy more than 0.9 for all the dataset.
9. Multi layer perceptron,Naive Bayes and Logistic also have good prediction accuracy with the range from 0.7 to 0.8 in every iteration.
10. Random Forest have lesser prediction accuracy while Logistic Regression have wide variation in accuracy in comparison to the other classifiers used in the empirical analysis.

Chapter 6

Ensemble Model Based ARB Prediction

6.1 Introduction and Motivation

Often, it is observed that a group of people takes better decision as compared to the individuals, as the individuals generally have their own bias. The same phenomenon is true for software bug prediction techniques. An ensemble model is a prediction approach which combines the output of different machine learning techniques so as to improve the overall prediction efficiency. In doing so, ensemble method utilizes several learning techniques and tries to achieve improved results. It might be possible that a group of technique used in ensemble makes similar assumptions and hence resulting in same error. We must select disparate techniques to ensure that they do not make same error.

The idea of ensemble method is as follows:

Consider a supervised learning problem, where training examples are given in the form of S = (x1 , y1),(x2 , y2),... ,(xn , yn) for some learning function Y = F(x). Here, x1 , x2 , ... ,xn are the features or the attributes having the values of discrete or real types; y1 , y2 , ... ,yn are the corresponding output values of discrete types (for classification) or continuous types (for regression). For the given training set S, each technique outputs an estimated value of y. Now for the set of unknown test examples T, learning techniques estimate the corresponding y values. Let call them h1 , h2 , ... ,hL , for L learning techniques. The ensemble method combines these outputs in the form of P = F(h1,h2 ...hL) , where F depends on the used combination rule and P is the final prediction over T.

Ensemble method combines the outputs of several base learners. Base learners can be created by re-sampling, manipulation, or randomizing the training dataset and the parameters of the learners. Broadly, ensemble method can be classified into two types, homogeneous ensemble method and heterogeneous ensemble method. Homogeneous ensemble method uses the same base learner on different subsets of training dataset. The examples of such methods are bagging, boosting, etc.[37]. Heterogeneous ensemble method uses different learning techniques. In heterogeneous ensemble method, ensemble process is two-fold. In first fold, different base learners are generated by applying learning techniques over the training dataset. In second fold, generated models are combined for the ensemble [38] .

Further, based on the combination rule, ensemble method can be classified into the linear ensemble method and non-linear ensemble method. In the linear ensemble method, outputs of learning techniques are combined in a linear fashion such as weighted average, simple averaging etc. [39] . In the non-linear ensemble method [40], some non-linear techniques such as decision tree, neural network, support vector machine are used to combine the outputs of learning techniques .

The building up of an Ensemble possesses three important phases:

6.1.1 Base Learners Used

The selection of base learners is very crucial in building of an ensemble method. For an ensemble method to be efficient, base learner needs to be precise and accurate and must be diverse enough. If the prediction for an unknown example is better than random guessing, then the learner can be considered as accurate. And two learners producing different error rate for a set of unseen test cases can be considered as diverse, in generating an ensemble method.

In order to assure that the base learners used are accurate and diverse enough for building an ensemble, we performed the experimental study as mentioned in section 5.2. It was aimed to evaluate the prediction performance of six machine learning technique, Support vector machine (SVM), K Nearest Neighbor (KNN), Naive Bayes (NB), Decision Tree (DT), Multi-Layer Perceptron (MLP) and Logistic Regression (LR) for the prediction of Age Related Bugs. We have evaluated Accuracy, Recall, Precision and F-Measure as performance measure.

6.1.2 Generation of Ensemble

We generate the ensemble methods by training the different base learners over bootstrapped subsets of training samples. We have used Bagging, Boosting and Stacking technique as described in section 6.1. The training of learning techniques may be over-fitting subsets of training samples used, thus creating performance error when used with validation data.

Ensemble methods uses meta-algorithms which is used to combine base learners and create a predictive model that can reduce variance (in case of Bagging), bias (in Boosting) or to improve the prediction performance (Stacking). As a result the overall performance of techniques improves when used with ensemble learning.

6.1.3 Ensemble Integration

Using the output of the base learners used and the combination rules used or meta classifier, the final prediction of the ensemble is used. Bagging uses majority Voting, Boosting uses weighted majority vote for final prediction, while Stacking feeds the result to train the meta-classifier which in turn makes the final prediction.

Bagging and boosting are homogeneous, the classifiers or base learners used are same trained with different subset of training samples of equal size. Stacking is heterogeneous with different base learners used at level 0 and a single classifier used at level 1 which gives final output.

Many variations of stacking can be used and evaluated to check the prediction performance over the aging dataset and compare the various ensemble models.

6.2 Motivation and Related Works

It is evident from the works such as Misrili et.al. [28], Wang et.al. [29], Khoshgoftaar et.al. [30] and Aljamann et.al. [31] that ensemble methods helps in achieving better prediction accuracy in comparison with individual machine learning techniques. But, all these works are used in the field of software fault prediction in order to predict the software modules to be faulty or non- faulty.

But as per the study by Cotroneo et.al [17], the ensembles methods are rarely used in the field of software aging to find the aging related bugs in the software system and is not fully explored. Thus, it is required to construct and evaluate ensemble methods for finding the aging related bugs.

In work [14] have explored the use of few ensemble methods for a learning technique for ARB prediction for imbalanced dataset. It used adaboost algorithm[32] along with other machine learning techniques and techniques for imbalanced datasets, but still the use of ensemble learning based model is very rare. There is a need to study the behaviour of various ensemble methods for different learning models for ARB prediction.

In this work, we have build and evaluated the ensemble learning models for predicting the aging related bugs in the software system. We have also proposed two stacking based ensemble models for the prediction of ARBs.

6.3 Background

We have applied three ensemble methods on each of the machine learning techniques discussed in section 4.1 to predict the ARB prone files. The ensemble methods used to predict the aging related bugs are Bagging, Boosting and Stacking. In this section, we have discussed these methods in brief.

6.3.1 Bagging

Bagging uses Bootstrap Aggregating algorithm for classification process [41]. In bagging, various classifiers used as members of ensemble are constructed from different training datasets, and the combined prediction is usually the uniform averaging or voting over all the members. Each classifier is trained on sample training example take from training dataset with replacement such that size of each sample equals the size of actual training dataset.

6.3.2 Boosting

Boosting is one of the popular ensemble technique, which generally uses AdaBoost algorithm [32]. It uses sequential training of models and performs training using multiple iterations, with a new model used for training in each iteration. It constructs an ensemble by using different example weights at different iterations. The prediction output is combined using voting on output of each classifier.In Boosting,examples that are incorrectly predicted by previous classifiers in the series are chosen more often than examples that were correctly predicted. Thus Boosting attempts to produce new classifiers that are better able to predict examples for which the current ensemble's performance is poor.

6.3.3 Stacking

Stacking is an ensemble technique which needs two level of models to construct an ensemble [1]. These are base models, also called as level-0, and meta-model or level-1 model. The level-0 models uses bootstrap samples of the training dataset, and the output of these base models are fed as input to a level-1 model. The meta-model aims to classify the target correctly by combining the outputs of base models and correcting the mistakes made by base models.

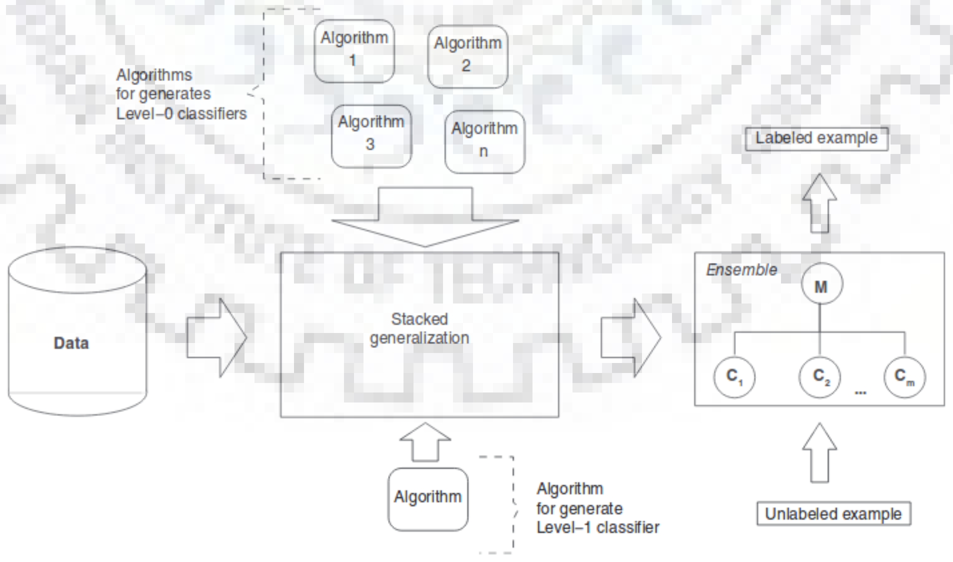


FIGURE 6.1: Overview of Stacking Procedure [1]

As initially noted by [1] some issues of Stacking are considered black art, such as the selection of base classifiers, the type of meta-data and the classifier to be used in level-1. Ting and Witten[42] conclude that multi-response linear regression (MLR) is the most appropriate algorithm for generating the meta-level model, atleast when using class probabilities as meta-level data. However their results didn't match their claim fully. However, after more than two decades since the pub- lication of Wolperts paper, the use of Stacking in real applications remains relatively rare, possibly due to what Wolpert called black art. In other words, there are several issues that could be considered when using Stacking, such as the following:

- The algorithms that are used to create the base-level classifiers and their learning parameters
- The number of base classifiers
- The algorithm used to generate the meta-classifier and its learning parameters
- The type of attributes that should be used to create the meta-data.

6.4 Proposed Model

Stacking produces excellent results in classification problems, but still has been rarely used ensemble model due to certain issues called black art by Wolpert[1]. Various variants of stacking has been proposed, but still selection of Level 1 classifier and Level 0 classifier is a black art, with no specific guidelines. As a result, Stacking and its variants are not fully explored in the area of software fault prediction.

We have proposed two stacking based ensemble models to predict the age related bugs.

6.4.1 Meta Classifier based Stacking Ensemble Model (MC-SEM)

The proposed model is a variant of generalized stacking algorithm[1].

For the selection of meta-classifier, MCSEM uses the best performing base learners. The selection of meta-classifier is based on the performance of base learners used. The best performing base learner is used as meta-classifier.

The working algorithm for MCSEM is as follows:

1. Set up the ensemble.
    - Specify a list of L base algorithms (Machine Learning Techniques) (with a specific set of model parameters).
    - Find the best performing base algorithm out of L classification algorithm.
    - Use it as metalearning algorithm.
2. Train the ensemble.
    - Train each of the L base algorithms on the training set.
    - Perform k-fold cross-validation on each of these learners and collect the cross-validated predicted values from each of the L algorithms.
    - The N cross-validated predicted values from each of the L algorithms can be combined to form a new N x L matrix. This matrix, along wtih the original response vector, is called the level-1 data. (N = number of rows in the training set.)
    - Train the metalearning algorithm on the level-one data. The ensemble model consists of the L base learning models and the metalearning model, which can then be used to generate predictions on a test set.
3. Predict on new data.
    - generate ensemble predictions, first generate predictions from the base learners.
    - Feed those predictions into the metalearner to generate the ensemble prediction.

In this model, we have used all six machine learning techniques namely SVM, MLP, Naive Bayes, Decision Tree, KNN and Logistic Regression as base learners.

After the empirical analysis of all these techniques defined in section 5, we select the best techniques as meta-classifier for the stacking. For our work, SVM performs best out of all learning techniques and used as meta-classifier for stacking in this proposed model. We have used 5-fold cross validation for the model (k=5). We have evaluated the accuracy, recall, precision and f-measure for this ensemble and compare with the other ensembles and individual learning techniques.

6.4.2 Hierarchical Stacking Ensemble Model (HSEM)

The proposed model is a variant of generalized stacking algorithm[1]. Here we have used six different stacking classifiers and majority voting of these six stacking classifiers will predict the final output.

The working algorithm for HSEM is as follows:

1. Specify a list of L base algorithms
2. for i=1..L
    - Make L base algorithms as classifier,
        - Set up the ensemble.
        - Select the i_{th} base algorithm as metalearning algorithm.
    - Train the ensemble.
        - Train each of the L base algorithms on the training set.
        - Perform k-fold cross-validation on each of these learners and collect the cross-validated predicted values from each of the L algorithms.
        - The N cross-validated predicted values from each of the L algorithms can be combined to form a new N x L matrix. This matrix, along wtih the original response vector, is called the level-1 data. (N = number of rows in the training set.)
        - Train the metalearning algorithm on the level-one data. The ensemble model consists of the L base learning models and the metalearning model, which can then be used to generate predictions on a test set.
    - Predict on new data.

  – generate ensemble predictions, first generate predictions from the base learners.
  – Feed those predictions into the metalearner to generate the ensemble prediction and store the prediction into PR$_i$.

3. for i=1.. L, use majority voting on PR$_i$ to make final prediction.

In this model, we have used all six machine learning techniques namely SVM, MLP, Naive Bayes, Decision Tree, KNN and Logistic Regression as base learners.

It is a three level model. We have build six stacking ensemble with all six learning techniques as base learners and one of these base learners are is used as meta classifier. We combine the prediction of all the six stacking ensembles to form the final prediction.

We have used 5-fold cross validation (k=5) for our work.

We have evaluated the accuracy, recall, precision and f-measure for this ensemble and compare with the other ensembles and individual learning techniques.

6.5 Experimental Results of Existing ensemble models

We have evaluated the various ensemble models with the six machine learning techniques defined in section 5.2 for our work. These are Bagging, Boosting, Stacking, Voting Classifier and few other variants of stacking on the dataset described in section 4.4.

We have compared the prediction performance of individual machine learning techniques used in section 5.2, with the different ensemble models evaluated in this section. We have also compared the performance of various ensemble models used with each other and previous works.

Table 6.1 shows the prediction performance of Bagging ensemble model with all the learning techniques.

Technique	Accuracy	Recall	Precision	F-measure
SVM	**0.9905**	**0.9952**	0.9937	0.9961
NB	0.7364	0.8550	**0.5570**	**0.6731**
DT	0.9896	0.9922	0.9845	0.9960
MLP	0.7476	0.6368	0.9112	0.7758
KNN	0.9712	0.9556	0.9862	0.9706
LR	0.6746	0.5985	0.9512	0.7121

Table 6.1: Empirical Study of Machine Learning Techniques using Bagging

Table 6.2 and Table 6.3 shows the prediction performance of Boosting ensemble model and Voting Classifier with all the learning techniques respectively.

Technique	Accuracy	Recall	Precision	F-measure
SVM	**0.5661**	**0.5662**	0.6137	0.5561
DT	0.9757	0.9632	**0.9889**	**0.9758**
NB	0.7990	0.7850	0.7870	0.7731
MLP	0.7194	0.6884	0.9084	0.7842
KNN	0.9244	0.8448	0.9459	0.8912
LR	0.5646	0.5385	0.8852	0.5721

Table 6.2: Empirical Study of Machine Learning Techniques using Adaboost

Technique	Accuracy	Recall	Precision	F-measure
Voting Classifier	**0.9816**	**0.9724**	0.9888	0.9881

Table 6.3: Voting Classifier

Technique	Meta-classifier	Accuracy	Recall	Precision	F-measure
SVM+KNN+DT	LR+PD	0.9811	0.9778	0.9685	0.9888
Stack of ALL	LR+PD	0.9905	0.9823	0.9785	0.9921

Table 6.4: Performnace of Stacking Classifier

From the experiments performed, following observations are drawn:

1. The ensemble learning methods improve the prediction accuracy of the machine learning algorithms mentioned in section 5.2 in almost all the cases for ARB prediction as shown in Table 6.1 and 6.2.

2. In the case of using bagging based ensemble, SVM learning model shows highest accuracy, recall, precision and F-measure and Logistic Regression performed worst in terms of accuracy and recall, while Naive Bayes performed worst in terms of precision and F-measure as given in Table 6.1.

3. In case of Boosting based ensemble model, Decision Tree outperformed every other learning models in terms of accuracy, recall, precision and F-measure. SVM performed worst in every performance parameter as given in Table 6.2.

4. The performance for the strong learners improved little or remained same, but the performance of the weak learners improved significantly with ensemble methods.

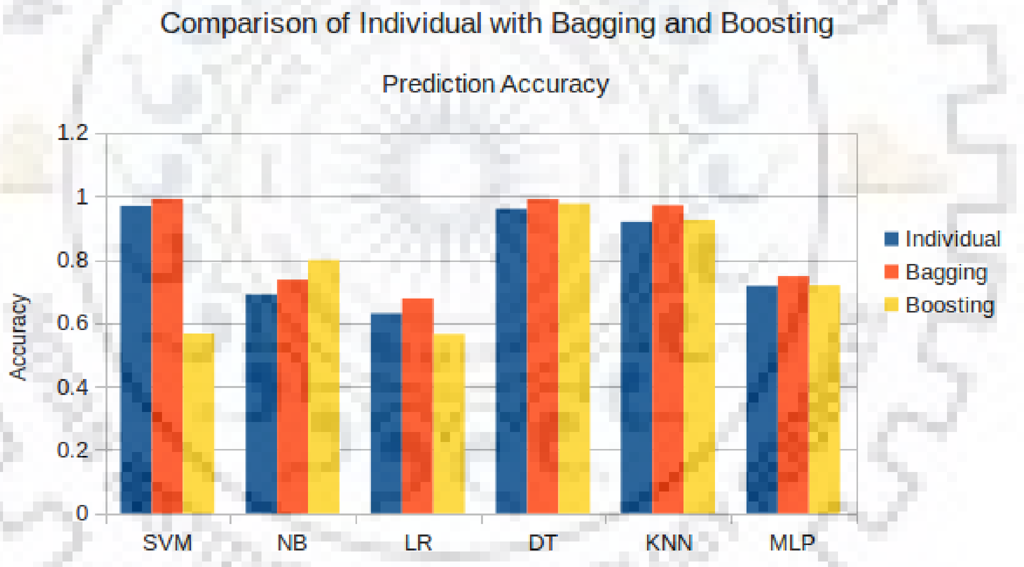


FIGURE 6.2: Comparison of Individual techniques with Bagging and Boosting.

5. In stacking, group of KNN, Decision Tree and SVM performed best in terms of accuracy, recall, precision and f-measure.

6. Voting classifier performs better than the individual machine learning techniques.

7. Various Stacking variants ensemble models used outperformed the other ensemble methods such as bagging and boosting.All of them are giving very high prediction accuracy (>0.98).

6.6 Experimental Results and Comparative Analysis of Proposed Model

We have also evaluated the proposed variants of stacking described above.
MCSEM-Using all techniques as base learner with best technique out of these as meta-classifier.
HSEM- Using the mean of all the six combination of meta-classifier as final prediction.

Technique	Meta-classifier	Accuracy	Recall	Precision	F-measure
MCSEM	SVM	**0.9971**	**0.9988**	**0.9862**	0.9748
HSEM		0.9938	0.9958	0.9782	0.9882

Table 6.5: Comparison of Proposed Stacking based ensemble model

MCSEM has very high accuracy of 0.9970 and outperformed every other ensemble model discussed . HSEM too has very high accuracy of 0.9938.

Technique	Meta-classifier	Accuracy	Recall	Precision	F-measure
Stack of ALL	LR	0.9905	0.9785	0.9923	0.9888
Stack of ALL	SVM	**0.9971**	**0.9988**	**0.9862**	**0.9748**
Stack of ALL	MLP	0.9815	0.9885	0.9778	0.9665
Stack of ALL	KNN	0.9920	0.9923	0.9876	0.9764
Stack of ALL	DT	***0.9953***	0.9753	0.9685	0.9888
Stack of ALL	NB	0.9823	0.9485	0.9588	0.9734

Table 6.6: Comparison of various stacking Classifier used in HSEM

We have performed the comparative study of the two proposed ensemble model namely Meta-Classifier based Stacking Ensemble Model (MCSEM) and Hierarchical Stacking Ensemble Model (HSEM) with the best performing base learning techniques described in section 5.2 as well as with the other existing ensemble models as presented in Table 6.7

Technique	**Accuracy**	**Recall**	**Precision**	**F-measure**
SVM	0.9891	0.9892	0.8461	0.8523
Bagging	0.9905	0.9952	0.9937	0.9961
Boosting	0.9757	0.9632	0.9889	0.9758
Stacking	0.9905	0.9823	0.9785	0.9921
MCSEM	**0.9971**	**0.9988**	**0.9862**	**0.9748**
HSEM	0.9938	0.9958	0.9782	0.9882

TABLE 6.7: Comparison of proposed models and other existing approaches

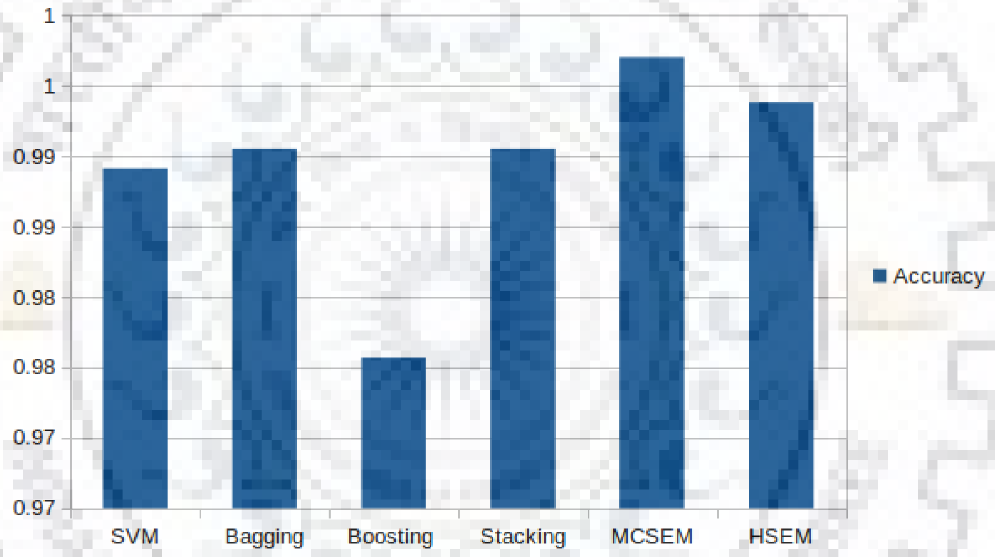


FIGURE 6.3: Comparison of proposed models and other existing approaches

It is evident from the above table and figure that MCSEM model has performed best among all the ensemble models evaluated and also outperformed even the best performing learning techniques.

HSEM also performed very well in prediction of ARBs and have better accuracy than most of the ensemble methods used and all the individual learning techniques.

We have also compared the prediction performance of MCSEM and HSEM with the earlier works performed in the area of software aging. The comparative study of these work with MCSEM and HSEM is shown in Table 6.8.

Attributes	**Work1**[14]	**Work2**[27]	**Work3**[27]	**MCSEM**	**HSEM**
Accuracy	0.9154	0.9136	0.9595	**0.9970**	0.9938
Is dataset Imbalanced?	Yes	Yes	No	**No**	No
Ensemble Methods used	Yes	No	No	**Yes**	Yes

TABLE 6.8: Comparative analysis of proposed model with other works

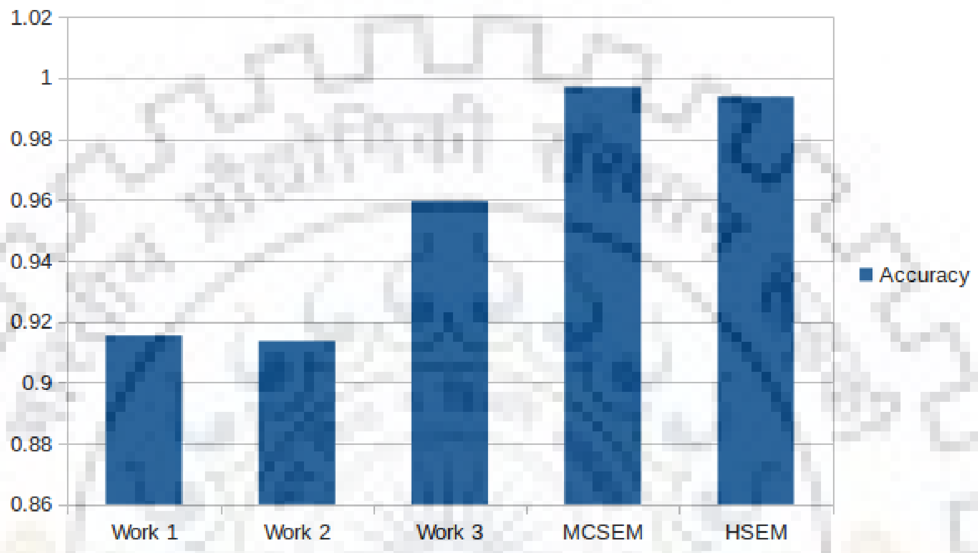


FIGURE 6.4: Comparative analysis of proposed model with other works

The proposed model MCSEM and HSEM has significant improvement over the other works in the area of age related bug prediction and have performed better than all other existing ensembles. as shown in Figure 6.3 and 6.4.

Chapter 7

Conclusion

In this thesis, we have applied bagging, boosting and stacking ensemble models on SVM, KNN, Decision Tree, Nave Bayes and Logistic Regression for the prediction of ARBs. We have also evaluated the individual machine learning techniques and performed the empirical analysis. The study was performed on the LINUX and MYSQL bug datasets available in IEEE software aging and rejuvenation repository. From the experimental analysis, it was observed that out of the studied learning models, SVM performed best with Bagging and Decision Tree performs best with Boosting. Whereas, Logistic Regression and SVM shows worst performance in case of Bagging and Boosting respectively, in terms of prediction accuracy, Recall, Precision and F-measure. Using stacking, the combined stack of SVM, KNN and Decision Tree outperformed all other learning models. The ensemble methods used performed significantly better than the individual base learners in most of the cases, while in some cases, the performance of ensemble methods is comparable with that of participating base learning techniques. Weak learners such as Logistic regression, MLP and Naive Bayes improved significantly, when used with ensembles. Various variants of stacking are created and they have shown comparable or improved results over other ensemble methods such as bagging and boosting, and have significant improvement over the individual techniques.

MCSEM model has performed best among all the ensemble models evaluated and also outperformed even the best performing learning techniques.HSEM also performed very well in prediction of ARBs and have better accuracy than most of the ensemble methods used and all the individual learning techniques.MCSEM provides the best result with accuracy of 0.9970. HSEM too has very high accuracy of 0.9938.

The proposed model MCSEM and HSEM has significant improvement over the other works in the area of age related bug prediction and have performed better than all other existing ensemble.

References

[1] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[2] Evren Ceylan, F Onur Kutlubay, and Ayse B Bener. Software defect identification using machine learning techniques. In *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on*, pages 240–247. IEEE, 2006.

[3] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Ayşe Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, 2010.

[4] David Oppenheimer, Archana Ganapathi, and David A Patterson. Why do internet services fail, and what can be done about it? In *USENIX symposium on internet technologies and systems*, volume 67. Seattle, WA, 2003.

[5] Soila Pertet and Priya Narasimhan. Causes of failure in web applications (cmu-pdl-05-109). *Parallel Data Laboratory*, page 48, 2005.

[6] Kishor S Trivedi, Kalyanaraman Vaidyanathan, and Katerina Goseva-Popstojanova. Modeling and analysis of software aging and rejuvenation. In *Simulation Symposium, 2000.(SS 2000) Proceedings. 33rd Annual*, pages 270–279. IEEE, 2000.

[7] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 381–390. IEEE, 1995.

[8] Michael Grottke, Rivalino Matias, and Kishor S Trivedi. The fundamentals of software aging. In *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pages 1–6. Ieee, 2008.

[9] Ann T Tai, Savio N Chau, Leon Alkalaj, and Herbert Hecht. On-board preventive maintenance: Analysis of effectiveness and optimal duty period. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on*, pages 40–47. IEEE, 1997.

[10] Eliot Marshall. Fatal error: how patriot overlooked a scud. *Science*, 255 (5050):1347–1348, 1992.

[11] Kalyanaraman Vaidyanathan and Kishor S Trivedi. A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, 2(2):124–137, 2005.

[12] Long Zhao, QinBao Song, and Lei Zhu. Common software-aging-related faults in fault-tolerant systems. In *Computational Intelligence for Modelling Control & Automation, 2008 International Conference on*, pages 327–331. IEEE, 2008.

[13] Rivalino Matias and JF Paulo Filho. An experimental study on software aging and rejuvenation in web servers. In *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, volume 1, pages 189–196. IEEE, 2006.

[14] A Ahmed. Predicting software aging related bugs from imbalanced datasets by using data mining techniques. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 18:27–35, 2016.

[15] Yasutaka Kamei and Emad Shihab. Defect prediction: Accomplishments and future challenges. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 5, pages 33–45. IEEE, 2016.

[16] Ishrat Un Nisa and Syed Nadeem Ahsan. Fault prediction model for software using soft computing techniques. In *Open Source Systems & Technologies (ICOSST), 2015 International Conference on*, pages 78–83. IEEE, 2015.

[17] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. Software aging and rejuvenation: Where we are and where we are

going. In *Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on*, pages 1–6. IEEE, 2011.

[18] Ira Cohen, Jeffrey S Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, volume 4, pages 16–16, 2004.

[19] Artur Andrzejak and Luis Silva. Using machine learning for non-intrusive modeling and prediction of software aging. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 25–32. IEEE, 2008.

[20] Ramendra K Sahoo, Adam J Oliner, Irina Rish, Manish Gupta, José E Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435. ACM, 2003.

[21] Kalyanaraman Vaidyanathan and Kishor S Trivedi. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, pages 84–93. IEEE, 1999.

[22] Lei Li, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. An approach for estimation of software aging in a web server. In *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*, pages 91–100. IEEE, 2002.

[23] Javier Alonso, Jordi Torres, Josep Ll Berral, and Ricard Gavalda. Adaptive on-line software aging prediction based on machine learning. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 507–516. IEEE, 2010.

[24] G Sumathi and R Raju. Software aging analysis of web server using neural networks. *arXiv preprint arXiv:1206.1534*, 2012.

[25] Domenico Cotroneo, Roberto Natella, and Roberto Pietrantuono. Predicting aging-related bugs using software complexity metrics. *Performance Evaluation*, 70(3):163–178, 2013.

[26] Fangyun Qin, Zheng Zheng, Chenggang Bai, Yu Qiao, Zhenyu Zhang, and Cheng Chen. Cross-project aging related bug prediction. In *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on*, pages 43–48. IEEE, 2015.

[27] Lov Kumar and Ashish Sureka. Aging related bug prediction using extreme learning machines. *IEEE India Council International Conferece (INDICON)*, 2017.

[28] Ayşe Tosun Mısırlı, Ayşe Başar Bener, and Burak Turhan. An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, 19(3):515–536, 2011.

[29] Tao Wang, Weihua Li, Haobin Shi, and Zun Liu. Software defect prediction based on classifiers ensemble. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, 8(16):4241–4254, 2011.

[30] Taghi M Khoshgoftaar, Erik Geleyn, and Laruent Nguyen. Empirical case studies of combining software quality classification models. In *Quality Software, 2003. Proceedings. Third International Conference on*, pages 40–49. IEEE, 2003.

[31] Hamoud I Aljamaan and Mahmoud O Elish. An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software. In *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, pages 187–194. IEEE, 2009.

[32] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy, 1996.

[33] Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2):434–443, 2013.

[34] Domenico Cotroneo, Antonio Ken Iannillo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. The software aging and rejuvenation repository. In *Software Aging and Rejuvenation (WoSAR), 2015 IEEE 7th Intl. Workshop on*. IEEE, 2015.

[35] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, pages 1310–1315. IEEE, 2016.

[36] BK Bhavitha, Anisha P Rodrigues, and Niranjan N Chiplunkar. Comparative study of machine learning techniques in sentimental analysis. In *Inventive Communication and Computational Technologies (ICICCT), 2017 International Conference on*, pages 216–221. IEEE, 2017.

[37] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *AAAI/IAAI, Vol. 1*, pages 725–730, 1996.

[38] Joao Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *ACM Computing Surveys (CSUR)*, 45(1):10, 2012.

[39] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[40] Tin Kam Ho. Multiple classifier combination: Lessons and next steps. In *Hybrid methods in pattern recognition*, pages 171–198. World Scientific, 2002.

[41] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[42] Kai Ming Ting and Ian H Witten. Stacking bagged and dagged models. 1997.