Software Reliability Prediction Using Various Ensemble Methods

Rohit Prasad, Abhishek Kumar, Perdun Kumar Mandal

Department of Computer Science and Engineering
Delhi Technological University
New Delhi, India

Abstract**—***Softwares covers an important aspect of human life today. A normal human being is surrounded by considerable amounts of softwares in his day-to-day life requiring it for all of his basic needs and activities pertaining to their survival. Moreover, our country has also been digitized by the Honourable Prime Minister of India through his Digital India Initiative famously known as "Digital India".Thus the demand for effective software systems has also increased. Most of the industries in various disciplines depend on computer software for their basic functioning. Thus the software companies need to deliver reliable and quality software on time. Reliability is treated as the key factor for the quality of a software system. A number of researches has been performed on software quality management reliability estimation. The objective of this paper is to use the ensemble methods and different machine learning techniques for software reliability prediction and evaluating them on the basis of selected performance criteria. In this paper Software Reliability modelling based on test data is done to estimate whether the current reliability level is up to mark of requirement for product. Software Reliability Modelling also helps to predict the reliability of different modules in the software.*

In this paper we used different ensemble methods on different machine learning techniques and studied performance using bagging, Adaboosting and Stacking. Using Ensembling a module in a software can be classified as having faults or not on the basis of certain attributes of software module and prediction of next failure in a software using the mean time between failure.

Keywords**—***Softwares, Digital India, Review and synopsis, Software Reliability Prediction, Ensemble methods, Machine Learning, Ada-boosting, Stacking, Bagging, faults, failure.*

1. INTRODUCTION

A software is something or we can say a product which is penetrating vigorously into our daily lives. Most of the day-to-day systems and activities are now driven by a lot of softwares. It has become a crucial part of critical and non-critical applications of many aspects of society, some of them of which probably are telecommunications, web teaching, home appliances, auditing, shopping, personal entertainment, airplanes, automobiles and so on. In particular, science and technology demand high-quality software for making improvements and breakthroughs and for delivering useful outputs. Most of the systems are now driven by software. Using software and its services has become a basic need for nearly 51% of the urban population of the country and will reach upto 66% of the total global population by 2023 as predicted by Cisco annual report(2018-2023)[1]. Also the size and complexity of software systems have grown drastically during the past few decades, and the trend will undeniably continue in the future. Previous researches show that size of the software for various systems and applications has been growing exponentially for the past 40 years[2]. Because of this accelerating dependency, software

failures can lead to serious, even fatal, consequences in safety-critical systems as well as in normal business. Previous software failures have diminished several high- visibility programs and have led to deprivation of businesses.

So evaluation, assurance and prediction of the quality of the software becomes an issue of critical concern. A software firm or a company must check the quality and performance of their software products before its final launch and delivery to end customers otherwise they may face several consequences or losses like the product may not survive in the market for long. In short, the software product should be reliable. Thus software quality assurance becomes an important feature to consider. Out of all software quality attributes such as reliability, availability, safety, security, and performance, software reliability is generally considered as the most important factor[3]–[6]. A software should be reliable for qualifying as a high quality software[7]. It quantifies software faults and failures, which can lead to serious consequences in safety-critical systems as well as in normal business[2].Therefore predicting software reliability is increasingly demanded in large, medium as well as small sized projects in order to achieve highly reliable and functional software systems.

Reliability of a software specifies the probability of failure free operation of the software for a given time duration in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of errors. Software Reliability is a dynamic system characteristic which can be characterized as a function of the number of software or system failures(presence of faults) that are encountered and reported during the functioning of the software within the given time period. A failure is an execution event where the software behaves unexpectedly. Failure is something that has to be executed, i.e., it is self dynamic in nature. Thus, more is the failure rate, less will be the reliability of the software. Not all software faults have equal probability of manifestation/execution. Removing software faults from those software faults which are rarely used makes little improvement in reliability. Failure of a software depends on

two factors: Number of faults being evaluated in software, and, the operational profile of execution of software.

In this research paper, we have evaluated the functioning of different machine learning techniques and algorithms for Software Reliability Prediction on three ensemble methods and successfully noted the effectiveness and accuracy and error in each of the ensemble methods and tried to find out the best fit model for reliability prediction with greater accuracy and least error rate. The remaining part of this paper is arranged as follows: Section II describes the ensemble techniques used such as bagging, boosting, stacking, etc. Section III describes the Machine Learning Techniques used and applied on the software prediction models like the ANN, KNN, Linear Regression, Decision Trees, etc. The list of tables and figures and dataset, results and the Conclusion isdescribed in the sections IV, V and VI respectively.

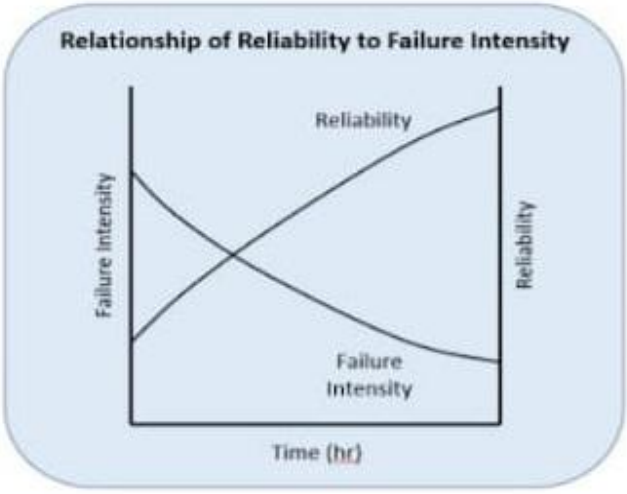

Figure 1. Reliability and Failure Intensity

1.1 Software Reliability Growth Model :-

Software Reliability Model is classified into two models, the static and the dynamic model. In static models, the modelling and analysis of the logic of the program is done on the

same code. In dynamic models, the debugging process is carried out temporarily and synchronously during the testing phase. A model describing the error detection in Software Reliability is known as Software Reliability Growth Model[8].

The Software Reliability Growth Model's Characteristics are as follows:-

- SRGM can be viewed as a product of a cumulative density function and a positive constant.

$$H(t) = a\left[1 - exp\left(-\int_0^1 f(x)dx\right)\right] = aG(t)$$

Here H(t) is a mean value function.

G(t) is Cumulative Density Function.

- The fault detection rate must be finite; G(t) must meet the condition that the corresponding failure rate function is finite.

$$h(t) = d(t)[a - H(t)] = ag(t)$$

where g(t)=dG(t)/dt is the probability density function(pdf) associated with G(t).

$$d(t) = ag(t)\frac{ag(t)}{a-aG(t)} = \frac{g(t)}{1-G(t)} = r(t) \text{.....(1)}$$

Where r(t) is the failure rate function associated with G(t). h(t) is intensity function. Since the fault detection rate must be finite, G(t) must meet the condition that the corresponding failure rate function is finite.

- We need to represent appropriately the right tail of g(t) behaviour to achieve a good reliability prediction. The right tail of SRGM associated with g(t) should be heavy. All these Characteristics should be satisfied for the Software Reliability Model[9].

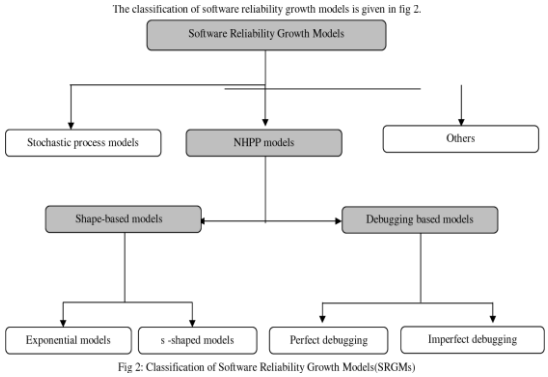

Figure 2: Classification of Software Reliability Growth Model[10]

1.2 Non-Homogeneous Poisson Process :-

Non-Homogeneous Poisson Process(NHPP) models are also termed as fault counting models and can be either finite failure or infinite failure models, depending on how they are specified. These models are useful for both the calendar time data as well as the execution time data[11].

Some Features of NHPP model:-

- A process which follows Poisson distribution with a time-dependent failure rate is called NHPP.
- Based on NHPP assumptions.
- Widely used by practitioners because they can be applied to hardware systems as well.
- Poisson distribution is suitable for modelling rare and random events such as failure of telephone calls.
- If NHPP can be assumed both for hardware and software systems, it becomes easier to find system reliability.

Basic Assumptions of NHPP models:-

- A software system is subject to failure at random times caused by software faults.
- There are no failures at time t=0.

- The probability that a failure will occur in a small interval 'h' is (t)h + 0(h), where (t) is instantaneous failure intensity.
- The probability that more than 1 failure will occur in a small interval 'h' is 0(h).This means that there is little chance of more than 1 failure occurring at small intervals of time.
- The expected number of failures during any time 't' to 't+t' is proportional to expected number of undetected faults at time 't'.
- The mean value function is also assumed to be bounded non-decreasing function of time in case of finite failure model such that:

$$\lim_{t\to\infty} u(t) = N < \infty$$

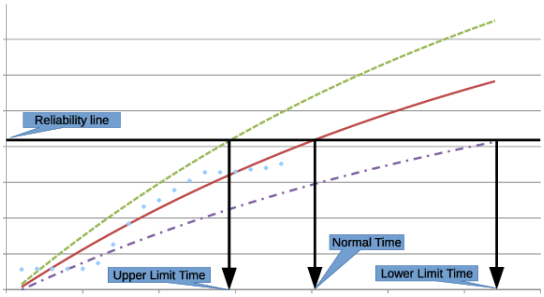

Figure 3: Time ranges based on NHPP model.

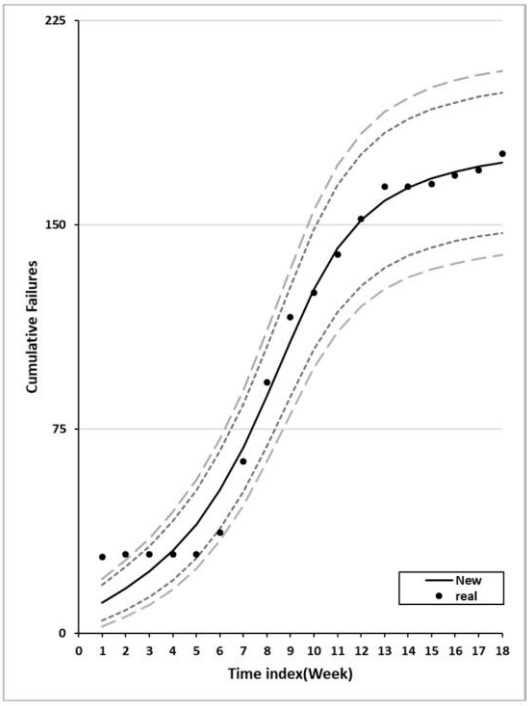

Figure 4: Time vs failure graph for NHPP model.

2. ENSEMBLE TECHNIQUE USED

2.1 Bootstrap Aggregating (Bagging)

It is a method to train a lot of decision trees on a data set to reduce the variance, increase stability and improve accuracy which in turn removes challenges of over-fitting.

Bootstrap aggregating, often abbreviated as bagging, involves having each model in the ensemble vote with equal weight. To promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set. As an example, the random forest algorithm combines random decision trees with bagging to achieve very high classification accuracy.

Example: We are given a training Data set T whose size is m, by using bagging it generates n new Training sets in which each has a size of mi by sampling from the trained data set D and replacement if m=mi then for large m Di have 63.2% of unique examples of D, the rest are duplicates. This type of sample is known as bootstrap sample.
Bagging can also degrade the performance of some stable algorithms such as k-nearest neighbours.

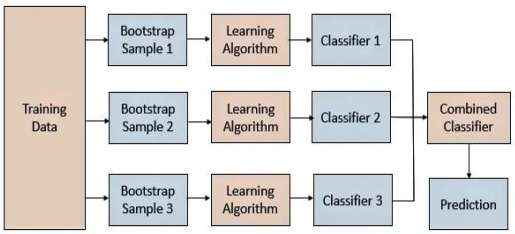

Figure 5. Bagging Model

2.2 Boosting

Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models' mis-classified. In some

cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to overfit the training data. By far, the most common implementation of boosting is Adaboost, although some newer algorithms are reported to achieve better results. In this work Adaboost algorithm is used to improve the accuracy of the weak classifiers. A boosting algorithm create an ensemble of classifiers, each one gives a weighted vote.

Boosting algorithm improves prediction power by training weak models, in which each weak model compensates for the weakness of its previous weak model. It is a generic model rather than a specific model.

Ada-boost was the first successful boosting algorithm developed for binary classification.

Algorithm:

i. Initialise the dataset and assign equal weight to each of the data points.

ii. Provide this as input to the model and identify the wrongly classified data points.

iii. Increase the weight of the wrongly classified data points.

iv. if(got required results)

    1. Goto step 5

v. else

    1. Goto step 2

vi. End

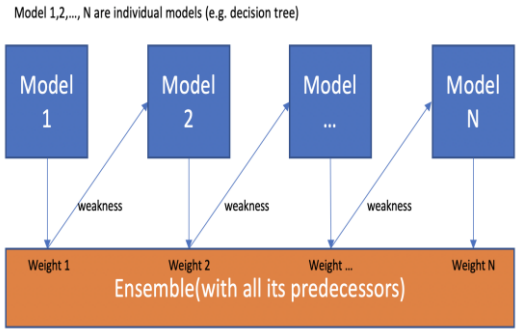

Figure 6. Boosting Model

2.3 Stacking

Stacking (sometimes called stacked generalization) involves training a learning algorithm to combine the predictions of several other learning algorithms. First, all the other algorithms are trained using the available data, then a combiner algorithm is trained to make a final prediction using all the predictions of the other algorithms as additional inputs. If an arbitrary combiner algorithm is used, then stacking can theoretically represent any of the ensemble techniques described in this article, although, in practice, a logistic regression model is often used as the combiner. The stacking model is represented as shown in Fig. 7 and Fig. 8.

Stacking is used to explore different models for a single problem. It is called Stacking because the final model is stacked on top of different models. This improves the overall performances as the final model is better than all other individual models.

How Stacking works:

i. The training data is divided into k-folds.

ii. In the k-1 part the base model is fitted and for the k-th part we make predictions.

iii. This is done for each part in training data.

iv. Base model is shaped into the whole train set and we calculate the test set performance.

v. Step 2,3,4 is repeated for every other model.

vi. The prediction of the initial set serve as a feature for 2nd level model.

vii. The prediction on the test set is made by using a 2nd level model.

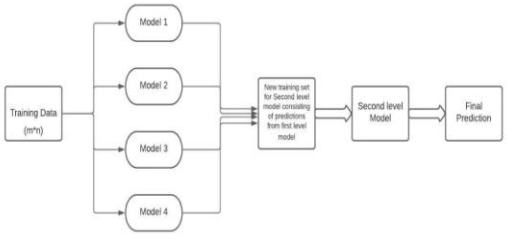

Figure 7. Stacking Model

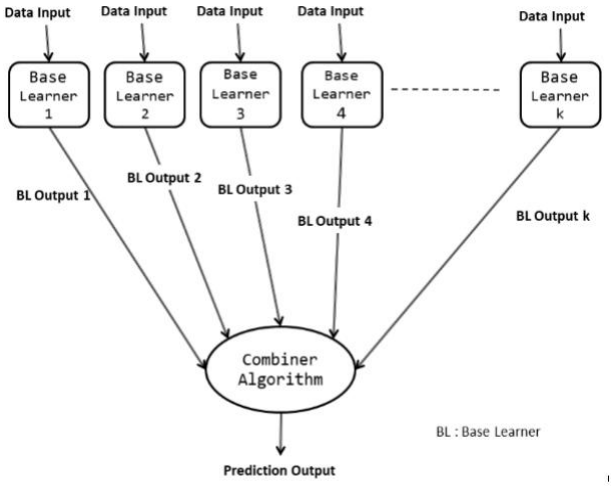

Figure 8. Stacking Model

2.4 Bayesian Parameter Averaging

- It is an estimate or new predicted observation which we get by averaging results of different models which we have used and in which each is weighted by its model probability.
- Conventionally after analysis we select the best model according to different parameters and then we learn about the parameters of the finalised model.
- But in this approach the uncertainty about the model-selection process is ignored. We can alternatively first learn the parameters of all models and then estimates can be combined by posterior probabilities of associate models.
- The above discussed approach is known as the Bayseian model averaging or the BMA.

Advantages of BMA:

- BMA reduces overconfidence as it does not ignore the uncertainties of the models.
- In case of logarithmic loss function or squared error loss the BMA predictions are most optimal.
- BMA is comparatively robust to model failure.

3. MACHINE LEARNING TECHNIQUES USED

3.1 Linear Regression

The linear model assumes the relation between features and target vector as approximately linear. Mathematically, it can be represented by equation:

$$y=mx + c + E$$

Here, y : target, x : data for a single feature, m and c are coefficients identified by fitting the model and E: error. The aim is to match the values of m and c to establish better relation between variable x and variable y.

Working:

We take a tuple from data set and enter the input to equation and predict the expected values. Now calculate the loss using a loss function. Also keep updating m(variance) and c(bias) to minimize the error. The loss function used is mean squared error.

Mathematically:

$$\sum_{i=1}^{n}\left(actual_{output} - predicted_{output}\right)^2$$

Equation of bias and variance are:

$$b_0 = \bar{y} - b_1\bar{x}$$

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\,(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

We update the values of m and c as:

$$m = m - \alpha\frac{d}{dm}(Error)$$

$$c = c - \alpha\frac{d}{dc}(Error)$$

When the model gets trained for minimum error, we fix bias and variance.

3.2 Logistic Regression

Logistic Regression produces results in a binary format which is used to predict the outcome of a categorical dependent variable. So the outcome should be discrete/categorical. The value of Y in logistic Regression is between 0 and 1, the linear line is clipped at 0 and 1.

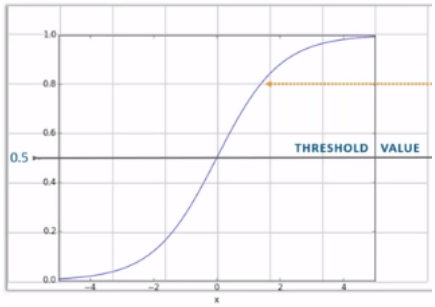

Figure 9: Logistic Regression Curve

Here the threshold value indicates the probability of winning or losing.

Logistic Regression Equation:

$$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \ldots\ldots + b_nx_n$$

To limit the value of Y to 0 and 1, we divide the above equation by (1-y) and take logarithmic of the equation:

$$log\left(\frac{y}{1-y}\right) = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \ldots\ldots + b_nx_n$$

Above one is the final Logistic Regression Equation.

3.3 Support Vector Machine

This model is generally used for performing classification tasks using a multidimensional space. In this, the example is represented as points in space that we map so that the points of different categories are as wide as possible.

3.4 Ridge Regression

It is one of the types of special linear regression models. Ridge regression contracts coefficients and reduces model complexity and multi-collinearity. In Ridge regression function is altered by adding a penalty value.

$$\sum_{i=1}^{M}(y_i - \bar{y})^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{p} w_j^2$$

Lower the value of, the model will resemble the Linear Regression Model.

3.5 Lasso Regression

It is a special type of linear regression model. It reduces over-fitting and also helps us in feature selection. The cost function for lasso regression is:

$$\sum_{i=1}^{M}(y_i - \bar{y})^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{p} |w_j|$$

For some t > 0, $\sum_{j=0}^{p}|w_j| < t$

3.6 Elastic Net Regression

It is combination of both Lasso and Ridge regression model into one model with two penalty factors , one is proportional to L1 norm and other is proportional to L2 norm.

3.7 ANN

From speech recognition to marketing and from healthcare to face recognition neural networks are widely used. Deep Learning uses Artificial Neural Networks which mimic the behaviour of the human brain to solve complex data driven problems. A neural Network works when some input data is given to it. The data is processed using layers of Preceptors to produce the output we require. Some real-world applications of ANN: self-driving cars are being perfected with the help of Neural Network. MuseNet, a Deep Neural Network that can generate 4-minute musical compositions with 10 different instruments.

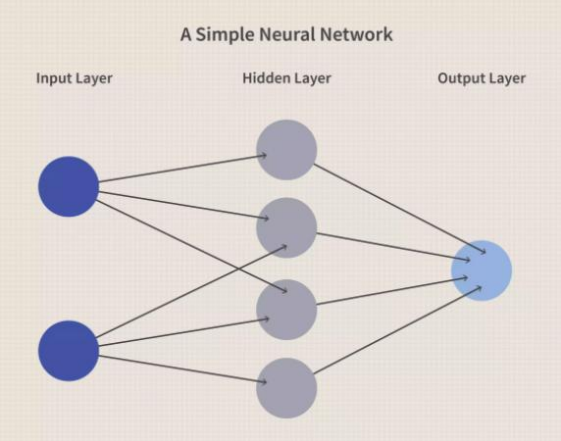

Figure 10:Simple Neural Network

3.8 KNN

K Nearest Neighbour is a simple algorithm that stores all the available cases and classifies the new data or case based on similarity measure.

Industrial application of KNN algorithm

1. Recommender system: In amazon when we search for a product in result it also shows relevant products.
2. Concept Search: Like an ocr and image search.

Example of KNN algorithm:

For predicting the T-shirt size for data H:161cm, W:61kg find out the Euclidean Distance from each Points, here value of k is 5 so find 5 nearest neighbours of the point, 4 points have size=M and 1 have size=L so the t-shirt size predicted is **M**.

Table 1. KNN example

S.NO	HEIGHT(CM)	WEIGHT(CM)	T Shirt Size	Euclidean Distance	Minimum Euclidean Distance
1.	158	58	M	4.242	
2.	158	59	M	3.605	
3	158	63	M	3.605	
4.	160	59	M	2.236	4
5.	160	60	M	1.414	1
6.	163	60	M	2.236	3
7.	163	61	M	2	2
8.	160	64	L	3.162	5
9.	163	64	L	3.605	
10.	165	61	L	4	
11.	165	62	L	4.123	
12.	165	65	L	5.656	
13.	168	62	L	7.071	
14.	168	63	L	7.280	
15.	168	66	L	8.602	
16.	170	63	L	9.219	
17.	170	64	L	9.486	
18.	170	68	L	11.401	

3.9 Naive Bayes Classifier

It is a classification technique based on Bayes theorem. Assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Some of the industrial use of Naive Bayes Classifier are :-

1.)News Categorization

2.) Spam Filtering

3.) Object and face recognition

4) Medical Diagnosis

5) Weather Prediction.

These are the steps involved in naive Bayes:

Step 1: Handling Data

Step 2: Summarizing Data

Step 3: Making a predictions

Step 4: Making all predictions

Step 5: Evaluate Accuracy

Step 6: Tying all together

3.10 Random Forest

Random forest developed by Leo Breiman[12] is a group of un-pruned classification or regression trees made from the random selection of samples of the training data. Random features are selected in the induction process. Prediction is made by aggregating (majority vote for classification or averaging for regression) the predictions of the ensemble.

It is a model which is used by operating Many Decision Trees during the training phase. For choosing the final decision the decision of majority decision Trees are chosen.

Why Random Forest is used:

i. No overfitting: because we use multiple trees it reduces the risk of overfitting upto a great extent.
ii. It runs very accurately on large databases.
iii. Random Forest can be used to Estimate missing data.

Applications of Random Forest:

i. Remote sensing

ii. Object detection

iii. Kinect: it is a gaming console that tracks the body movement and recreates it in a game.

Each tree is grown as described in[12]:

i. By Sampling N randomly, If the number of cases in the training set is N but with replacement, from the original data. This sample will be used as the training set for growing the tree.
ii. For M number of input variables, the variable m is selected such that m << M is specified at each node, m variables are selected at random out of the M and the best split on these m is used for splitting the node. During the forest growing, the value of m is held constant.
iii. Each tree is grown to the largest possible extent. No pruning is used.

Random forest generally exhibits a significant performance improvement as compared to single tree classifiers such as C4.5. The generalization error rate that it yields compares favourably to Ada-boost, however it is more robust to noise.

3.11 Decision Tree

Decision Tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. It is the graphical representation of all the possible solutions to a decision. The Decisions are

based on some conditions. Decisions made can be easily explained.

Example of decision tree:-

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Figure 11. Decision Tree example

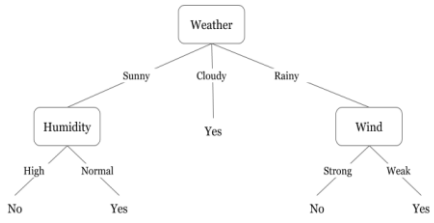

Figure 12: Decision tree

4. DATASETS USED

This section describes about the two dataset that is used in this proposed work. In this section, it is described about the two-benchmark datasets that have been used in this proposed work. Many standardized published works have been done on these datasets and statistical error measures are available.

For prediction of time of failure, dataset used: Musa, J.D: Software reliability data, IEEE Computer Society Repository(1979), Raj and Kiran: Software reliability data (2008), Iyer and Lee Software reliability dataset (1996).Failure data during system testing phase of various projects collected at Bell Telephone Laboratories, Cyber security and Information Systems Information Analysis Centre(CSIAC) by John D. Musa are considered.

For classification, the software reliability classification dataset used : Source : NASA, then the NASA Metrics Data Program, http://mdp.ivv.nasa.gov. Mike Chapman, Galaxy Global Corporation.

1. loc : McCabe's line count of code
2. v(g) : McCabe's "cyclomatic complexity"
3. ev(g) : McCabe's "essential complexity"
4. iv(g) : McCabe's "design complexity"
5. n : Halstead total operators + operands
6. v : Halstead "volume"
7. I : Halstead "program length"
8. d : Halstead "difficulty"
9. i : Halstead "intelligence"
10. e : Halstead "effort"
11. b : Halstead value
12. t : Halstead's time estimator
13. IOCode : Halstead's line count
14. IOComment : Halstead's count of lines of comments
15. IOBlank : Halstead's count of blank lines
16. IOCodeAndComment : IO lines

17. uniq_Op : unique operators
18. uniq_Opnd : unique opearnds
19. total_Op : total operators
20. total_Opnd : total operands
21. branchCount : of the flow graph
22. prediction : {false,true}module has one or more reported defects or not

5.RESULTS

In this work of Software Reliability Prediction we used a different Machine Learning Prediction Algorithm. While working on the actual data both Ridge regression and Bayesian Ridge regression perform better together on both the datasets. The performance was found to be better compared to the existing algorithms on the Normalized Root Mean Square Error values(NRMSE). After interpolating some points some models like K-nearest neighbours(KNN), Artificial Neural Network(ANN) perform better. As a whole we observed, logarithmic scaling was an important factor in the improvement of the model performance because of uneven distribution of data values and the time series datasets having peaks at certain points. As we observed that when the lag length was between 7 and 11 Ridge, Bayesian Ridge, Radial Basis Function Network regressors gave the best result. Random Forest, Artificial Neural Network(ANN) regressors give their best while increasing the lag length but on the actual dataset.

So for Prediction of time of failure we use a Dataset from IEEE computer Society Repository namely Musa, J.D: Software Reliability Data.

Different accuracies and confusion matrix obtained on the testing dataset, by several machine learning algorithms like ANN, KNN, Decision Tree algorithm for bagging method are shown in TABLE 2.

Table 2. Bagging Accuracy(in %) for different machine learning algorithms along with the confusion matrix.

Model	Accuracy	Confusion Matrix	
Random Forest	95.87	255	71
		16	1767
ANN	89.81	149	177
		38	1745
KNN	88.48	135	191
		52	1731
Decision Tree	96.68	280	46
		24	1759
SVM	88.90	97	229
		5	1778
Logistic Regression	86.25	74	252
		38	1745
Naïve Bayes	82.50	122	204
		165	1618

Stacking is used to explore different models for a single problem. It is called Stacking because the final model is stacked on top of different models. This improves the overall performance as the final model is better than all other individual models. The stacking accuracy for different machine learning algorithms under the stacking ensemble method are shown in TABLE 3.

Table 3. Stacking Accuracy(in %) for different machine learning algorithms for three values of k.

Model	K = 7	K = 9	K = 11
Random Forest	85.15	86.57	85.15
KNN	84.67	86.09	85.78
Decision Tree	84.36	83.57	84.04
SVM	84.83	85.15	83.72
Logistic Regression	84.04	85.93	85.3
Naïve Bayes	84.2	83.57	84.044

Reliability prediction for ada-boost ensemble method was performed using Logistic regression, Naive Bayes, and KNN machine learning algorithms for different values of k and the results obtained is shown in TABLE 4.

Table 4. Individual and Boosted Ada-Boost Accuracy(in %) for different machine learning algorithms.

Model	Logistic Regression	Naïve Bayes	KNN
K = 9	**Ind:** 86.30 **Boosted:** 86.15	**Ind:** 82.74 **Boosted:** 82.36	**Ind:** 89.76 **Boosted:** 91.84
K = 11	**Ind:** 86.58 **Boosted:** 86.30	**Ind:** 82.84 **Boosted:** 82.46	**Ind:** 89.33 **Boosted:** 92.65
K = 13	**Ind:** 86.44 **Boosted:** 86.15	**Ind:** 82.65 **Boosted:** 82.55	**Ind:** 89.38 **Boosted:** 92.70
K = 15	**Ind:** 86.44 **Boosted:** 86.20	**Ind:** 83.03 **Boosted:** 82.46	**Ind:** 88.90 **Boosted:** 91.75

There is no software that is purely perfect or error-free. It is not possible for a software to have 100% accuracy. There always are some errors that resides. So we have also calculated the mean square errors tested for bagging prediction and stacking prediction under various machine learning models which is shown below in TABLE 5.

Table 5. Mean Square Errors(in %) of different algorithms for bagging prediction and stacking prediction.

Model	Bagging Prediction Result	Stacking Prediction Result
Ridge	0.000015	0.001428
Lasso	0.000015	0.001440
Decision Tree	0.000005	0.000480
Random Forest	0.000005	0.001038
SVM	0.016254	1.660045
Linear Regression	0.000014	0.001445
Bayesian Ridge	0.000014	0.001430
Lasso Lars	0.000014	0.001437
Logistic Regression	0.015817	1.773583

6. CONCLUSION

Various machine learning Prediction & Classification algorithms have been used in ensemble learning for software reliability prediction. As we know that in the bagging approach of the classification the output is classified in different class and then the final output with maximum vote is chosen using voting method.

Boosting algorithm improves prediction power by training weak models, in which each weak model compensates for the weakness of its previous weak model. It is a generic model rather than a specific model. For Binary classification Ada-boost was proven to be the very first successful boosting algorithm.
It is a method to train a lot of decision trees on a data set to reduce the variance, increase stability and improve accuracy which in turn removes challenges of overfitting. This method is useful when a single Model is giving very high variance so to reduce the variance we run the same dataset on multiple models and take the vote count of output. The different model used in this technique were Naive-Bayes, KNN, Logistic Regression, ANN, Random forest.

We are getting very less error rate in prediction problem for prediction. In boosting model, weak machine learning model have been taken and then by Ada-Boost algorithm the accuracy of the model has been improved. In stacking we are combining different base learner algorithm and then predicting the dataset and finally we are using a combiner algorithm to predict the output. All the output value is stored in xlsx file and tableau software is used to visualize the data or the output that we got from the Ensemble method for software reliability prediction. In bagging method of classification, we can see that for k=7 the decision tree gives a highest performance among all other machine learning model. In boosting method, for different value of k the output of KNN model is improved and the accuracy value is increased by maximum 1%. For stacking for different value of k, i.e. 7, 9 or 11 KNN model gives highest accuracy among all.

On linearly interpolated data K-Nearest Neighbours(KNN) and Support Vector Machines(SVM) regressors were performing better which was observed to be the best performance among all the cases in certain lag lengths.

REFERENCES

[1] "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper - Cisco." https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed Nov. 30, 2020).

[2] M. R. Lyu, "Software reliability engineering: A roadmap," *FoSE 2007 Futur. Softw. Eng.*, pp. 153–170, 2007, doi: 10.1109/FOSE.2007.24.

[3] "Software Reliability Characteristics | Download Scientific Diagram." https://www.researchgate.net/figure/Software-Reliability-Characteristics_fig2_228872404/actions#reference (accessed Nov. 29, 2020).

[4] "Quality Assurance » Software Quality Attributes." http://www.qasigma.com/2008/12/software-quality-attributes.html (accessed Nov. 29, 2020).

[5] A. Amin, L. Grunske, and A. Colman, "An approach to software reliability prediction based on time series modeling," *J. Syst. Softw.*, vol. 86, no. 7, pp. 1923–1932, 2013, doi: 10.1016/j.jss.2013.03.045.

[6] M. Palviainen, A. Evesti, and E. Ovaska, "The reliability estimation, prediction and measuring of component-based software," *J. Syst. Softw.*, vol. 84, no. 6, 2011, doi: 10.1016/j.jss.2011.01.048.

[7] "What Makes Software High-Quality? - Shlomi Fish's Homesite."

https://www.shlomifish.org/philosophy/computers/high-quality-software/ (accessed Nov. 30, 2020).

[8] L. Shanmugam, "A Comparison of Parameter Best Estimation Method for Software Reliability Models," *Int. J. Softw. Eng. Appl.*, vol. 3, no. 5, pp. 91–102, 2012, doi: 10.5121/ijsea.2012.3508.

[9] R. Jiang, "Required characteristics for software reliability growth models," in *2009 WRI World Congress on Software Engineering, WCSE 2009*, 2009, vol. 4, doi: 10.1109/WCSE.2009.157.

[10] B. John, "A Brief Review of Software Reliability Prediction Models," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. V, no. IV, pp. 990–997, 2017, doi: 10.22214/ijraset.2017.4180.

[11] R. Lai and M. Garg, "A detailed study of NHPP software reliability models," *J. Softw.*, vol. 7, no. 6, pp. 1296–1306, 2012, doi: 10.4304/jsw.7.6.1296-1306.

[12] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, "Random Forests and Decision Trees," Int. J. Comput. Sci. Issues, vol. 9, no. 5, pp. 272–278, 2012.