ABSTRACT

Software reliability is an indispensable part of software quality. It is one of the most valuable aspect for evaluating quality of software product. Software industry endures various challenges in developing highly reliable software. Here we have used machine learning techniques & ensemble approach for study of software reliability prediction and compared their performance with existing algorithms on various statistical measures. Software reliability prediction problem is about detection of accurate next failure time using various machine learning regressors and scaling techniques.

In this proposed work of prediction various regression models are trained and final value is predicted, and different error measures like Normalized Root Mean Square Error (NRMSE), Sum of Squared Error (SSE) & Mean Squared Error (MSE) are computed to find the performance of different models. In our work, the performance comparison with various models on benchmark datasets on statistical measures are done.

Contents

Chapter 1

INTRODUCTION

Software reliability is defined as “The ability of the software to perform its required function under stated conditions for a stated period of time”. With the rapid growth and increasing complexity of the software, reliability of the software is often hard to achieve. Reliability is one of the important and crucial aspects of the software quality.

According to ANSI, software reliability is ‘‘The probability of failure free operation of a computer program for a specified period in a specified environment’’ (Quyoum et al. 2010). In this work, the model has been used for predicting and estimating the number of errors in the software. The primary goal of software reliability modelling is to find out the probability of a system failing in given time interval or the expected time span between successive failures.

Machine Learning (ML) techniques have proved to be successful in predicting better results that statistical methods and can be used for prediction of software failures more accurately and precisely. ML is an approach which is focused on learning automatically and allows computers to evolve and predict the system behaviour based on past and the present failure data. Thus, it is quite natural to know that which method tends to work well for a given failure dataset and up to what extent quantitatively (Aggarwal et al. 2006; Goel and Singh et al. 2009).

In this work, ensemble learning approach is used to predict and evaluate the software models based on their performance. Various ML algorithms are used to predict the time between successive software failures. In prediction work, different models are trained on the input data and models predict the values based on the test data. The main challenge in the prediction problem is to use different error measuring techniques to know the performance of the models. Using a single model for predicting the data may not result in better prediction all the time. So different regression models are used to determine the next failure time and error measures of different models are compared. Also, by combining various models, the objective is to propose a model to improve performance on multiple datasets.

Chapter 2

Work Objective

Business applications which are critical in nature require reliable software, but developing such software is a key challenge which the software industry faces today. With the increasing complexity of the software these days, the reliability of the software is often hard to achieve. Ensemble method has been used for predicting, estimating number of errors remaining in the software.

The primary goal of software reliability modelling is to find out the probability of a system failing in given time interval or the expected time span between successive failures. In this work, ML techniques used for predicting software reliability prediction are Random Forest, Decision Tree, Ridge Regression, Bayesian Ridge Regression, KNeighbors Regression, Lasso Regression, ElasticNet Regression, Artificial Neural Network (ANN), Radial Basis Function Network (RBFN), Support Vector Regressor (SVR). The mentioned machine learning methodologies are used on different datasets.

For predicting software reliability, the dataset of successive failures of the software is used and different above-mentioned ML techniques are used for prediction of the failure time of the software reliability dataset. The value is predicted based on the Time Between Successive Failures (TBSF) using a single feature dataset representing the time between successive failures in chronological order. Before predicting the result, the time series data is pre-processed using Logarithmic Scaling approach and the required data is extracted accordingly. On the basis of lag length, data-window of various length in the dataset is created. Various ML techniques and algorithms are applied on the lagged dataset for getting different statistical error measures and the next failure time by reverse Logarithmic Scaling. In prediction, a set of hypotheses are combined to give better accuracy and improved results.

Chapter 3

Literature Survey

Several ML techniques have been proposed and applied in the literature for software reliability modelling. Some of the techniques are Generic Programming, Gene Expression Programming, Artificial Neural Network, Decision Trees, Support Vector Machines, Feed Forward Neural Network, Fuzzy models, Generalized Neural Network etc (Malhotra et al. 2011; Xingguo and Yanhua 2007; Hua Jung 2010; Karunanithi et al. 1992; Singh and Kumar 2010d; Eduardo et al. 2010). Ho et al. (2003) carried out a comprehensive study of connectionist models and their applicability to software reliability prediction and inferred that these are better as compared to traditional modes. Su and Huang (2006) have applied neural network for predicting software reliability.

3.1 The Connectionist Model

The artificial neural networks, also known as connectionist models, represent an innovative approach that is rooted in many disciplines. It essentially tries to develop the underlying mapping function of the process by “learning” based on a series of input patterns.

However, despite its highly successful applications in such diverse fields as modelling, process identification and control, pattern recognition, speech and signal processing, classification, etc., its use in reliability data analysis and prediction is not widespread. A relatively novel technique in software reliability growth prediction(Karunanithi et al. 1992), have demonstrated that neural networks models performed better than parametric models and that the types of network architectures can significantly influence predictive performance. In recent years, there have been some revived interests in exploring the use of neural models for software quality and reliability.

For an input stream $\{X_t\}$ consisting of history of the random variable being studied, under the assumption of no external variables, the general neural network structure is essentially a nonlinear model

$$Y_{t+1} = F(X_t, X_{t-1}, X_{t-2}\ldots\ldots\ldots\ldots\ldots\ldots X_{t-p}) \qquad [3.1.1]$$

where Y_{t+1} denotes the output target, F is the underlying mapping function, and there is *p+1* input neuron. From a statistical perspective, in

time series forecasting, the stochastic behaviour of the output is given by the conditional expectation

$$E[Y_{t+1} \mid X_t, X_{t-1}, X_{t-2}\ldots\ldots\ldots\ldots\ldots X_{t-p}], \qquad [3.1.2]$$

which is the minimum mean square error predictor. One of the most popular neural network models is the multilayer feedforward network. Suppose that there are k input neurons connected to a single neuron i. A simple form of this dynamic feedforward neural model has the following relationship:

$$u_j(t+1) = F\{\sum_{i=1}^{n} w_{ji} G[u_i(t)]\} \qquad [3.1.3]$$

where $u_j(t)$ is the activation of neuron *j* at time *t*; *G* is the threshold function; w_{ji} is the connection weight from the input neuron *i* to neuron *j* in the next layer; *F* defines the type of transfer function under consideration. We can easily extend this concept to a multilayer network structure consisting of many neurons in the hidden layers. It can also be shown that for any feedforward network with an input vector *X* of *n* neurons, i.e., $\{x_1, x_2, \ldots, x_n\}$, and a hidden layer of k neurons, there exists a network output function *g(X, Θ)* that can provide an accurate approximation to any function of input vector scaled between 0 and 1. Under the assumption of linear threshold function, the generated output signal will be

$$g(X,\Theta) = F\{\sum_{j=0}^{k} v_j f(\sum_{i=0}^{n} w_{ji} x_i)\} \qquad [3.1.4]$$

where Θ represents the overall weight vector, and v_j denotes the connecting weights between the output neuron and neuron *j* in the hidden layer.

3.2 Time Series Approach in ANN

Time series approach in generally used for predicting future time between software failures (Bisi-Goyal et al. 2017). Predicting next step time between failures using time series approach has only one dependent variable y_i as output and several explanatory variables (y_{i-1}, $y_{i-2, \ldots,}$ $y_{i-p+1,}$ y_{i-p}) as input. The time series model can be represented as follows:

$$y_i = f(y') \qquad [3.2.1]$$

Where y' is a vector of variables (y_{i-1}, $y_{i-2, \ldots,}$ $y_{i-p+1,}$ y_{i-p}) and *p* is the lag value. The function *f* is determined by training and architecture of ANN. This function in case of ANN is not explicitly available to users but

inherently built in the architecture of ANN. A feed forward network with single hidden layer, as shown in Fig 1, is considered. A vector of previous time between failure data (y_{i-1}, $y_{i-2, \ldots}$, y_{i-p+1}, y_{i-p}) are input and y_i is the output of ANN.

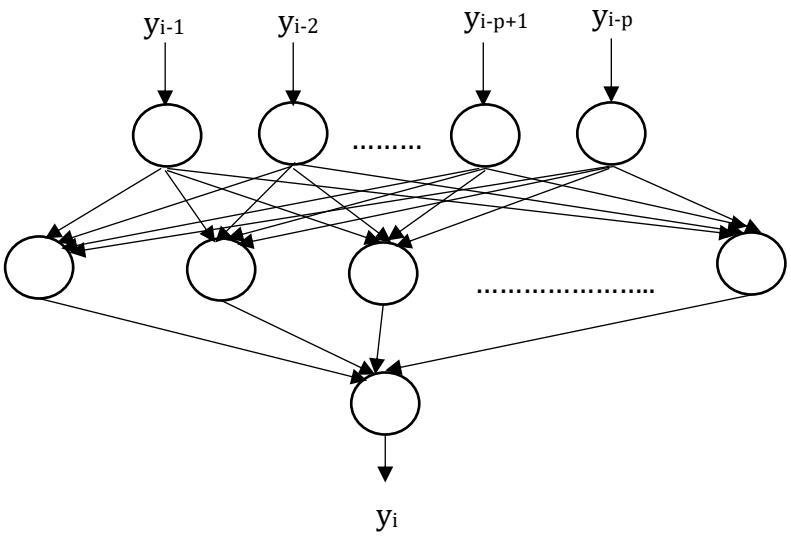

Fig 1. *Feed Forward artificial neural network*

This function in case of ANN is not explicitly available to users but inherently built in the architecture of ANN.

3.3 ANN Model

In the ANN model (Bisi and Goyal et al. 2017), y_i, which is the time between i^{th} and $(i-1)^{th}$ failure, is predicted using ANN to which (y_{i-1}, $y_{i-2, \ldots}$, y_{i-p+1}, y_{i-p}) are applied as input. Here, p is the lag value, which denotes the number of input variables. In general, ANN scales its input in the interval [0,1] using linear scaling such as dividing the values with maximum expected value of corresponding input variable. ANN shows better training and prediction capability when data points are evenly distributed in range [0,1] (Karunanithi et al. 1992). However, during the initial phase of testing, time between failures is expected to be small. As testing progresses, software becomes more reliable and time between failures becomes large.

In case of linear scaling, whenever time between failures values are divided by the maximum expected time between failures value, most of data points will be located towards lower region near 0 during training and near 1 during the prediction process. With application of the logarithmic scaling, the scaled data points will have better distribution in

the range [0,1]. Therefore, logarithmic function is used to scale the time between successive failures data in this work.

The predictive capability of ANN for software cumulative failures data has been found to be improved when data are scaled using appropriate logarithmic function (Viswanath et al. 2006), which scales the input data in range [0,1]. To verify the above, experiments are conducted, and accuracy of results compared, obtained with and without scaling of data. It is found that ANN provides better accuracy when logarithmic scaling is used. The scaled values are taken as input as well as output of ANN. The logarithmic scaling function is discussed in section 5.1.11.

In ANN, appropriate architecture is required to provide good prediction accuracy. But this is a difficult task and needs optimization of many components like number of hidden layers, number of hidden neurons in each layer and input delay in the ANN. The architecture of the ANN used for prediction consists of an input layer, one hidden layer and an output layer.

There is no formal theory for determining optimal ANN architecture. In order to investigate whether error measure, which is normalized root mean squared error (NRMSE). It would decrease further by taking higher number of hidden neurons. The number of hidden neurons is varied from 2 to 7. It is found that beyond 7 number of hidden neurons, there is no improvement on the prediction accuracy for the proposed approach. In some case it made ANN to remember the training data and provided worse results.

The predictive performance of ANN may be seriously affected if the time lag is either too little or too large. A lot of experiments are carried out using the different benchmark dataset as discussed in Section 4 by varying lag from 1 to 7. In the proposed ANN and ANN-PSO approaches, for lag value higher than 5, the structural complexity of ANN arises, which leads to increase in the convergence speed of ANN training. Due to which prediction capability of ANN decreases. Therefore, lag length is varied from 1 to 5. All datasets are partitioned into two parts containing 80% and 20% of data for training and testing respectively. Training epochs is set to be 1000 and 0.01 learning rate is used.

3.4 ANN-PSO Model

The ANN model described above is trained using backpropagation algorithm (Bisi and Goyal et al. 2017). The value of scaling parameter is obtained by assuming the maximum time between failures value in advance. Wrong assumptions may affect the predictive capability of proposed model. The maximum encoded value is fixed as 0.9 to determine the scaling parameter of ANN model developed. It is observed from set up that scaling parameter has a major impact on NRMSE for all datasets. So, it is not advisable to fix this value for multiple datasets. This scaling parameter needs to be optimally fixed for different datasets. In this new model another ANN model is proposed with extra one hidden layer. The additional layer is added between input and 1st hidden layer which converts the time between failures in the range [0,1]. Logarithmic scaling function described in section 5.1.11 is used as the activation function in the additional input layer. But the notations are different. Suppose, the input is y_{i-1}. It is converted into range [0,1] using the following instruction:

$$y'_{i-1} = \ln(1 + (W_{111} * y_{i-1}) + B_{11}) \qquad [3.4.1]$$

Where, y'_{i-1} is the output of 1st neuron in additional layer which is the scaled value of y_{i-1}. W_{111} is the scaling parameter which is determined during ANN training . B_{11} is the bias of 1st neuron in additional layer. PSO approach is used to train the ANN and to optimize the weights and biases of ANN. If the output of a neuron in additional layer is less than 0 and greater than one, the solutions are rejected and new solutions are generated to scale output values in the range [0,1]. Therefore, the search space of the solution lies in the above range in the proposed approach. In a similar manner, output of each neuron in additional layer is calculated.

Each particle consists of weights and biases of ANN. The number of weights and biases of ANN depends upon the lag value for the proposed approach. If lag value is 2, then the number of weights in ANN is 12 and number of biases is 7. So, each particle is initialized randomly. Then, pbest which is the particle's best position and gbest which is all particles best position in the swarm are calculated. NRMSE is considered as the fitness function used in the Particle Swarm Optimization (PSO) approach. The position and velocity of the particle is changed depending upon pbest and gbest value. Velocity of individual particle is obtained using the following function:

$$v_i^t = wv_i^{t-1} + n_1 r_1 (p_i^t - x_i^t) + n_2 r_2 \left(p_g^t - x_i^t\right) \qquad [3.4.2]$$

Where, v_i is the i^{th} particle's current velocity, w is the inertia constant, n_1 and n_2 are the acceleration constant, r_1 and r_2 are two uniformly distributed random numbers, p_i is i^{th} particle's pbest value and p_g is the gbest value in the entire swarm in a particular iteration. Position of individual particle is obtained using the following function:

$$x_i^t = x_i^{t-1} + v_i^t \qquad [3.4.3]$$

Performance is influenced by the combination of parameters of PSO in general. For each dataset, simulations are carried out using several combinations of PSO parameters. The proposed ANN-PSO model is applied on the benchmark datasets.

3.5 Statistical Efficiency Measures

To validate the proposed reliability prediction models have used several efficiency measures (Jaiswal-Malhotra et al. 2016). The precise view of overall prediction methodology adopted is illustrated through a flow chart, Fig 2. To conduct this experiment, foremost requisite is the selection of independent and dependent variables. The dependent variable which has been used in this research paper(Jaiswal-Malhotra et al. 2016) was failure rate and the independent variable used was time interval in weeks.

For the corresponding failures, testing time in weeks is chosen to be independent variable. Su and Huang (2006) had applied neural network for predicting software reliability. Pai and Hong (2006) performed experiments using SVMs for forecasting software reliability. Kumar and Singh (2012) discusses about the usage of machine learning techniques like cascade correlation neural network, decision trees and fuzzy inference system to predict the reliability of software products.

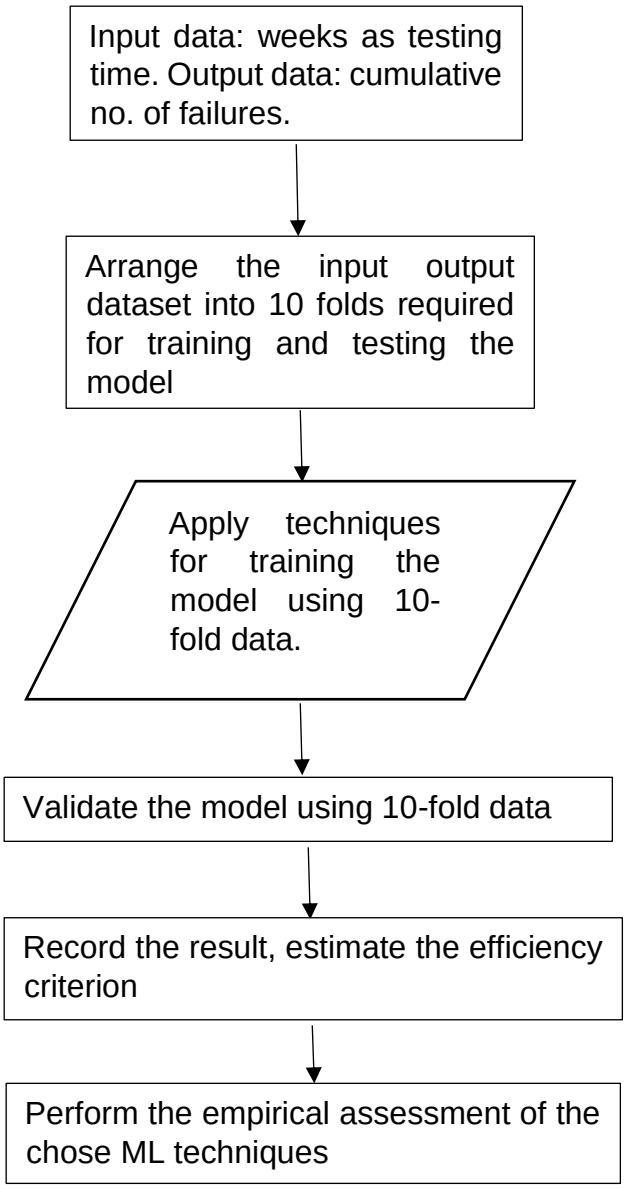

Fig. 2. *Flowchart of software reliability prediction methodology adopted.*

Chapter 4

Background 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).

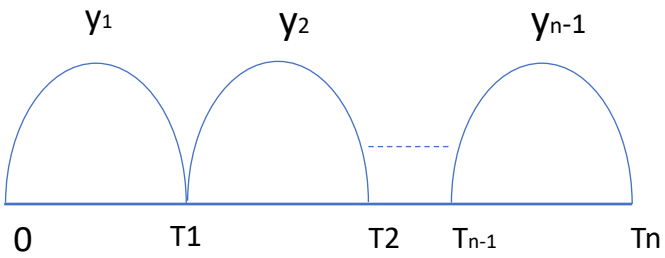

Fig 3. *Software failure process*

A typical software failure process is shown in Fig 3, Where T_i is the time instant of i^{th} failure. The datasets provide time between failures for all observed failures. Each observation of the dataset is represented by (i, y_i) where *i* is the failure number and y_i is time between $(i-1)^{th}$ and i^{th} failures. The mathematical equation of y_i generated from the software failure is:

$$y_i = T_i - T_{i-1} \qquad [4.1]$$

Chapter 5

MACHINE LEARNING AND SCALING TECHNIQUES, ERROR MEASURES AND ENSEMBLING

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent algorithms alone. Machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives. Supervised learning algorithms are most commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a problem. Even if the hypotheses space contains hypotheses that are very well-suited for a problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a better hypothesis. The term ensemble is usually reserved for methods that generate multiple hypotheses using the same base learner. The broader term of multiple classifier systems also covers hybridization of hypotheses that are not induced by the same base learner.

Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of to compensate for poor learning algorithms by performing a lot of extra computation. Fast algorithms such as decision trees are commonly used in ensemble methods (for example, random forests), although slower algorithms can benefit from ensemble techniques as well. An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble, therefore, represents a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. This flexibility can, in theory, enable them to over-fit the training data more than a single model would, but in practice, some ensemble techniques tend to reduce problems related to over-fitting of the training data.

Empirically, ensembles tend to yield better results when there is a significant diversity among the models. Many ensemble methods, therefore, seek to promote diversity among the models they combine. Although perhaps non-intuitive, more random algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms (like entropy-reducing decision trees). Using a variety of strong learning algorithms, however, has been shown to be

more effective than using techniques that attempt to *dumb-down* the models to promote diversity.

While the number of component models of an ensemble has a great impact on the accuracy of prediction, there is a limited number of studies addressing this problem. *A* prior determining of ensemble size and the volume and velocity of big data streams make this even more crucial for online ensemble models. Mostly statistical tests were used for determining the proper number of components. More recently, a theoretical framework suggested that there is an ideal number of component model for an ensemble such that having than this number of models would deteriorate the accuracy. It is called "the law of diminishing returns in ensemble construction." Their theoretical framework shows that using the same number of independent component model gives the highest accuracy.

5.1 Machine Learning and Scaling Techniques

There are different regression techniques for prediction like Ridge Regression, Bayesian Ridge Regression, Random Forest, Decision Tree, SVR, K Nearest Neighbors, ElasticNet Regression, Lasso Regression, ANN & RBFN. For Scaling and pre-processing of the data Logarithmic Scaling approach is used.

5.1.1 Ridge Regression

Ridge Regression is a technique for analysing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large, so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable

5.1.2 Bayesian Ridge Regression

In statistics, Bayesian Ridge Regression is an approach similar to Ridge Regression in which the statistical analysis is undertaken within the context of Bayesian Interface. When the regression model has errors that have a normal distribution, and if a form of prior distribution is assumed, explicit results are available for the posterior probability distributions of the model’s parameter.

5.1.3 Random Forest (RF)

Random Forests are an ensemble learning method for regression and other tasks. They operate by constructing multitude of decision trees at

training time and outputting the class that is the mode of the classes or mean prediction of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

5.1.4 Decision Tree (DT)

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.

5.1.5 Support Vector Regression (SVR)

Support Vector Regression is a regression method, which maintains all the main features similar to Support Vector Machine (SVM) that characterize the algorithm. A margin of tolerance is set in approximation to the SVM which would have already requested from the problem. The main idea always remains same which is to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

5.1.6. K Nearest Neighbors Regression (KNN)

K nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure. KNN has been used in statistical estimation and pattern recognition. A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance function as KNN classification.

5.1.7 ElasticNet Regression

In statistics and in the fitting of linear or logistic regression models, the elastic net is a regularized regression method that linearly combines the L_1 and L_2 penalties of the lasso and ridge methods.

5.1.8 Lasso Regression

Lasso is a regression analysis method that performs both variable selection and regularization to enhance the prediction accuracy and interpretability of the statistical model it produces.

5.1.9 Artificial Neural Network (ANN)

ANN or connectionist systems are computing systems vaguely inspired by the biological neural networks that are found in the brain of animal. The Neural network itself isn't an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems learn to performs tasks by considering examples, generally without being programmed with any task-specific rules. ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and the signal additional artificial neurons connected to it.

5.1.10 Radial Basis Function Network (RBFN)

In the field of mathematical modelling, a Radial Basis Function Network is an ANN that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. RBFN have many uses, including function approximation, time series prediction, system control.

5.1.11 Logarithmic Scaling

Linear scaling of data points decreases the correlation between the points and hence decrease the performance of evaluation. So, in-order to improve the performance of the model Logarithmic scaling method is used between the successive failures and given by:

$$y^* = \ln(1 + By) \qquad [5.1.11.1]$$

Where, y^* is the scaled value of time between successive failures, y is the time between successive failures and B is the Scaling constant which is given by:

$$B = \frac{e^{y^*max} - 1}{y_{max}} \qquad [5.1.11.2]$$

After prediction reverse scaling is done to get the actual time between failures and is given by:

$$y = \frac{e^{y^*} - 1}{B} \qquad [5.1.11.3]$$

Where y^* is the simulated output of prediction and B is the scaling constant obtained using equation [5.1.11.2].

5.2 Lagging

The bench mark datasets that has been used for this proposed work are single featured datasets. So, for single instance of the prediction of error; Lag method is used to generate a window of values and combining the set of values for the machine learning prediction problem. The lagging Algorithm in this work is as follows:

Algorithm 1 Lagging

- *Input: Dataset, lag sequence(k, e.g.: k=3 or 4 or 5 etc)*
- *Algorithm:*
    1. *Length = length of the input dataset*
    2. *for i in 0 to Length-k+1:*
    3. *Store data from i to i+k index of the dataset in the list z*
    4. *return z*
- *Output: array with lagged value from the original dataset in this case z list*

The lag length in this proposed is varied from 2 to 25 depending upon the model which is used for prediction. After lagging, the single featured dataset is segmented into different lag sequence according to the defined lag length.

5.3 Linear Interpolation

Linear Interpolation is a method of creating virtual point in the dataset. If the dataset size is small; then it is very difficult to get good accuracy after train-test split. So, in order to improve the accuracy of the model this method is used to increase the size of the training data. After increasing the dataset size, it is very efficient to train the model and test the model. Due to this, the performance of the model increases. As the benchmark dataset used in this proposed is single featured failure time data; So, linear interpolation is easy to use. The Linear Interpolation algorithm works as follows:

Algorithm 2 Linear Interpolation

- *Input: Dataset*
- *Algorithm:*
    1. *train = train_set(Dataset)*
    2. *data_array = [ ]*
    3. *data_array.append(train [0])*
    4. *for index in range(1, length(train))*
    5. *data_array.append((train [index-1]+train[index])/2)*
    6. *data_array.append(train[index])*
- *Output: array with linearly interpolated value i.e. data_array*

This algorithm helps to create virtual point in the train data by using the linear interpolation method.

5.4 Error Measures

Various statistical error measures used in this work to evaluate the regression models and compare the performance in benchmark dataset are:

- NRMSE: Normalized Root Mean Squared Error

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - y_i')^2}{\sum_{i=1}^{n} y_i^2}} \quad [5.4.1]$$

- SSE: Sum of Squared Error

$$SSE = \sum_{i=1}^{n}(y_i - y_i')^2 \quad [5.4.2]$$

- RE: Relative Error

$$RE = \left|\frac{y_i' - y_i}{y_i}\right| * 100 \quad [5.4.3]$$

Where, n is the number of prediction data points; y_i is the actual time between failures and y' is the estimated time between i^{th} and $(i-1)^{th}$ failures. These are the standard error measures which are used in this proposed work to evaluate the performance of the model.

5.5 Proposed Model

The Proposed Model in Software Reliability Prediction 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 model can theoretically represent any of the ensemble technique. The Proposed Model is represented as shown in Fig. 4.

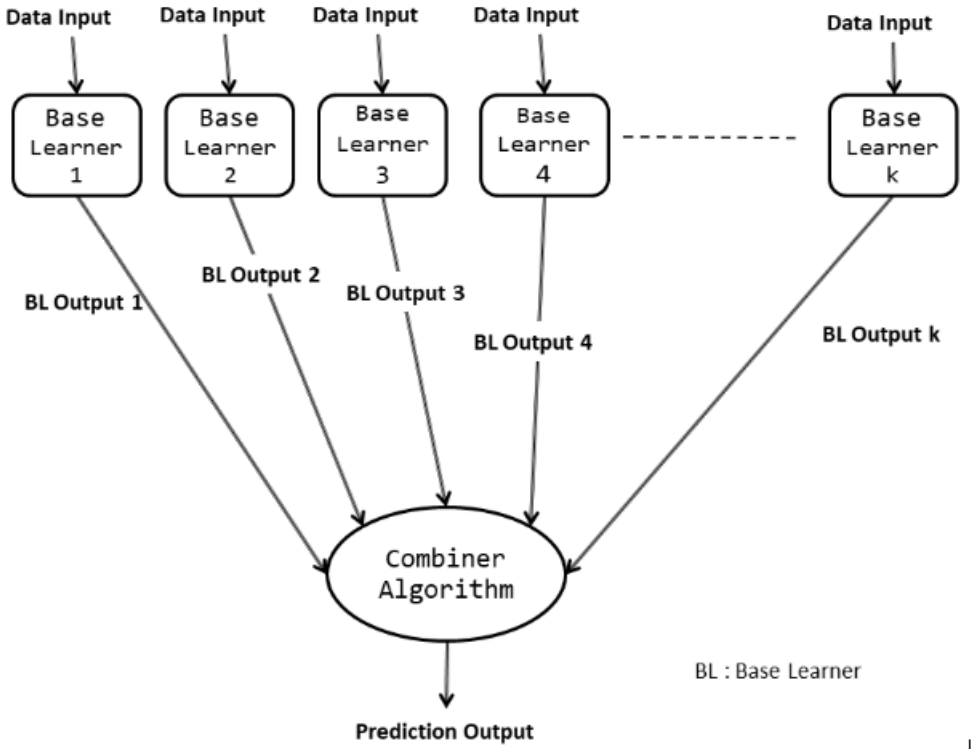

Fig.4 . *The Proposed Model*

In the Fig.4. it is explained how different based learner algorithms are combined to give the final output and better accuracy. The Proposed Model typically yields performance better than any single one of the trained models. It has been successfully used on both supervised learning tasks and unsupervised learning.

The algorithm of the Proposed Model is given below:

Algorithm 4 Proposed Model

- *Input: Dataset*
- *Method:*
  1. *BaseLearners = [SVR, ANN, Bayesian Ridge, KNN……]*
  2. *Combiner = [Ridge]*
  3. *X_train, X_test, Y_train, Y_test = test_train_split(Dataset)*
  4. *Model = [ ]*
  5. *For function in BaseLearners:*
  6. *Model.append(function.train(X_train, Y_train)*
  7. *Train_set = [ ]*
  8. *For i in range(length(X_train)):*
  9. *Train = [ ]*
  10. *For m in Model:*
  11. *Train.append(m.predict(X_train[i]))*
  12. *Combiner.train(Train_set, Y_train)*
- *Prediction:*
  1. *Train_set_x = [ ]*
  2. *For m in model:*
  3. *Train_set_x.append(m.predict(x))*
  4. *Output = Combiner.predict(Train_set_x)*
- *Output: Individual Accuracy array of each method, Stacking accuracy*

This is the ensembling algorithm that we have used in this work to improve the accuracy by combining different other algorithms.

Chapter 6

RESULT ANALYSIS

In this work of predicting Time Between Successive Failures, the two benchmark datasets are passed to various algorithms. The statistical error measures of the model's performance are plotted. We have used Python as basic programming language to implement these algorithms.

To find the performance of the models on various regressors, the length of the sliding window is varied from length 2 to 25 and the observations are recorded. We also observed that on increasing the length of sliding window, the size of the train set decreases. Hence, creation of virtual data points seemed a way of increasing the dataset size. As this is a time series data having positive and negative peaks; Linear Interpolation was used to populate the train set. As observed, certain models improved their performance drastically after interpolating the points as the correlation between data points were established and variance between them decreased. But for certain models like Ridge and Bayesian Ridge regressors, it had an inverse impact. Since, the bias of the non-CART (Classification and Regression Trees) models increases as we decrease the variance during the bias-variance trade-off, the performance of the model decreases significantly.

6.1 Musa Dataset Results

The plots of NRMSE values with variation on lag length of different regressors are given below. The plots also include performance with and without interpolation. The plots show that regressors like Ridge, Bayesian Ridge, SVR, Random Forest performed better on this dataset. While regressors like ElasticNet and Lasso regression show the worst error values. The above mentioned two regressors' error values don't vary much with the change of lag window size. KNN and ANN regressors performed better with the inclusion of the value from interpolation.

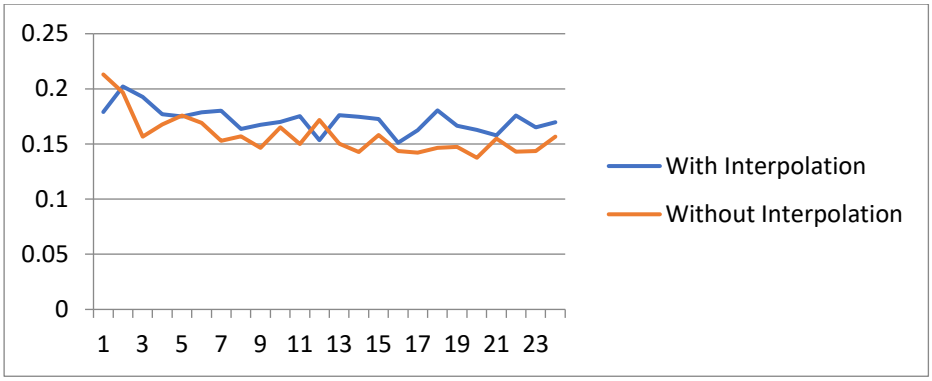

Fig 5. *NRMSE performance of different lag length in Random Forest*

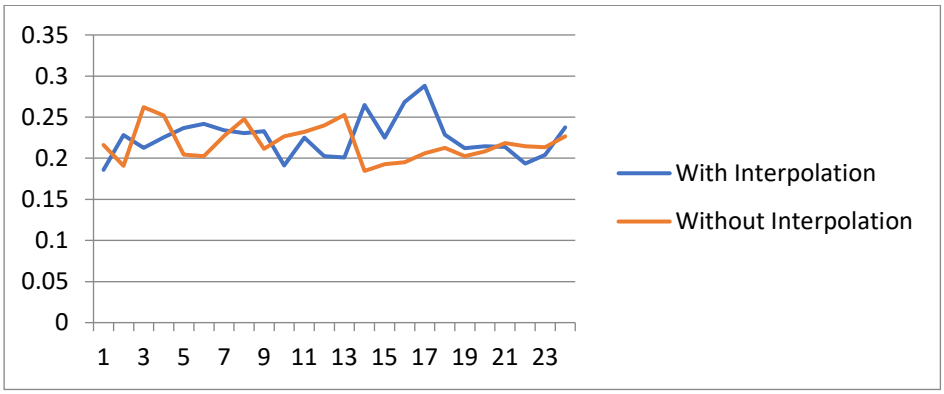

Fig 6. *NRMSE performance of different lag length in Decision Tree*

There is a non-predictable dependency between the plots of NRMSE values vs the lag length of datasets for the regressors. Linear Interpolation of the train sets affects the NRMSE performance for certain lag lengths by improving it but in most of the cases of regressors like DT, RF,RBFN,SVR the performance deteriorates as seen from the plots.

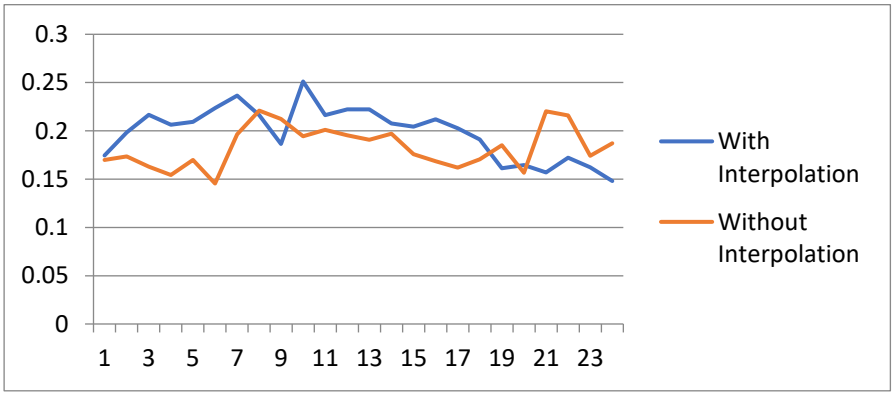

Fig 7. *NRMSE performance of different lag length in RBFN*

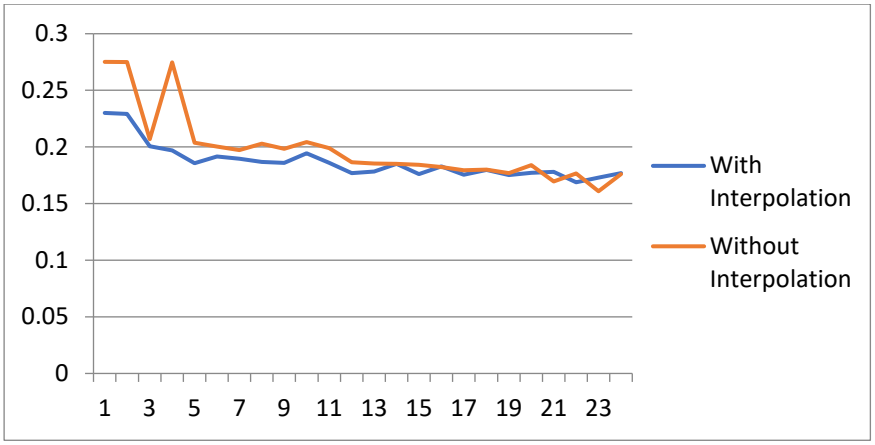

Fig 8. *NRMSE performance of different lag length in ANN*

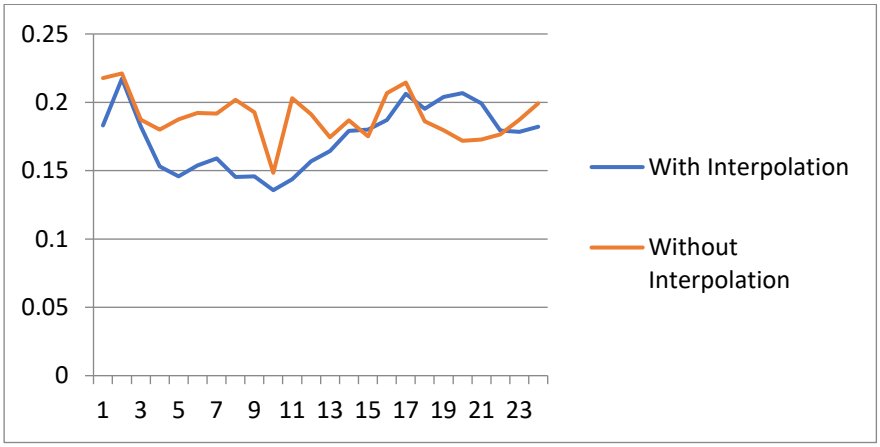

Fig 9. *NRMSE performance of different lag length in KNeighbors*

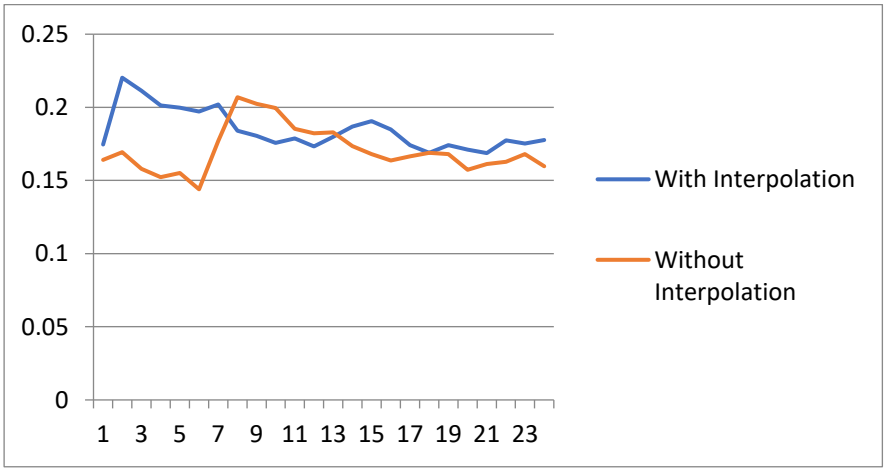

Fig 10. *NRMSE performance of different lag length in SVR*

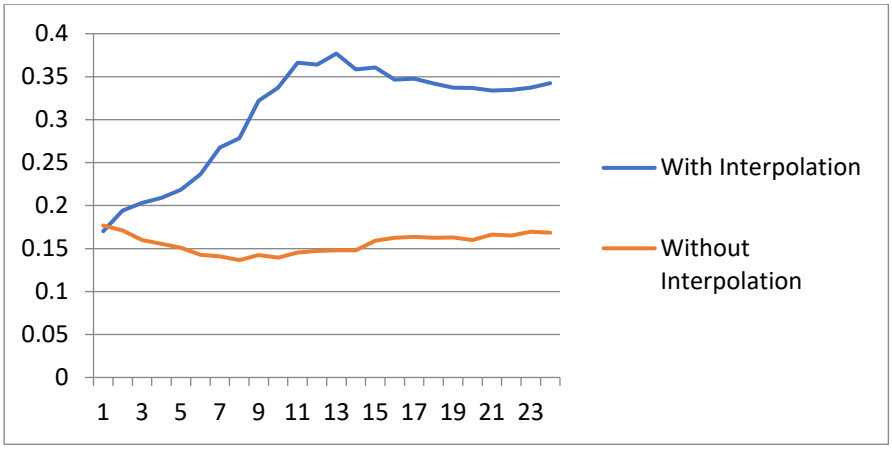

Fig 11. *NRMSE performance of different lag length in Ridge*

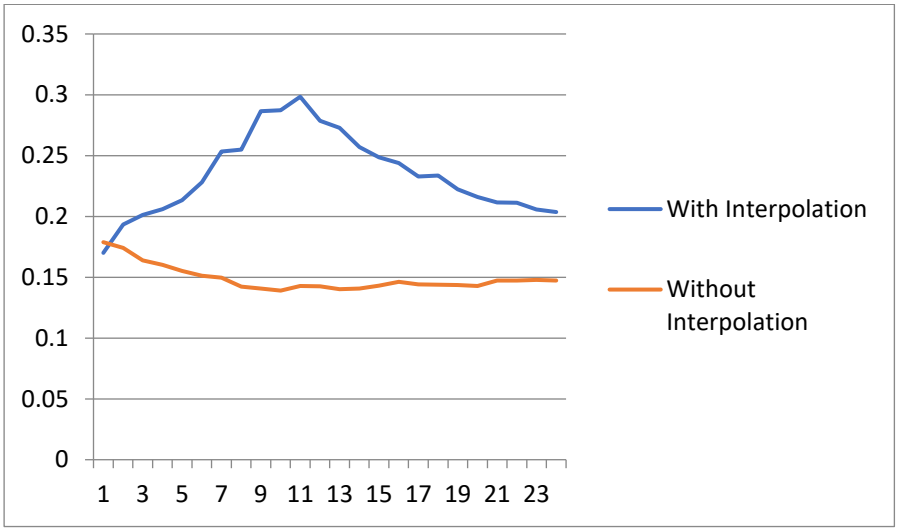

Fig 12. *NRMSE performance of different lag length in Bayesian Ridge*

6.2 Iyer Lee Dataset Results

The plots of NRMSE values with variation on lag length of different regressors are given below. The plots also include performance with and without interpolation. The plots show that regressors like RBFNN, ANN and SVR performed better on this dataset. While regressors like ElasticNet and Lasso regression show the worst error values. The above mentioned two regressors' error values don't vary much with the change of lag window size. KNN, ANN, Random Forest, Decision Tree regressors performed better with the inclusion of the value from interpolation.

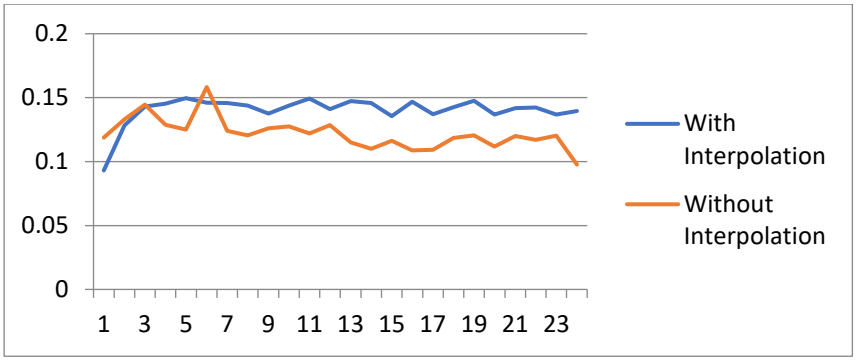

Fig 13. *NRMSE performance of different lag length in Random Forest*

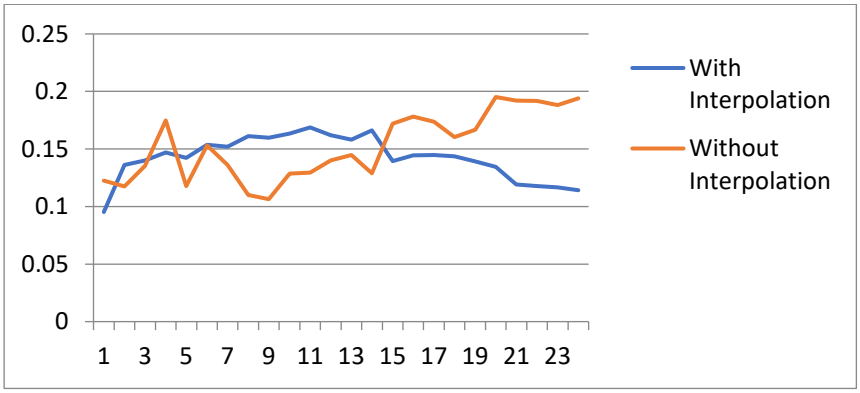

Fig 14. *NRMSE performance of different lag length in KNeighbors*

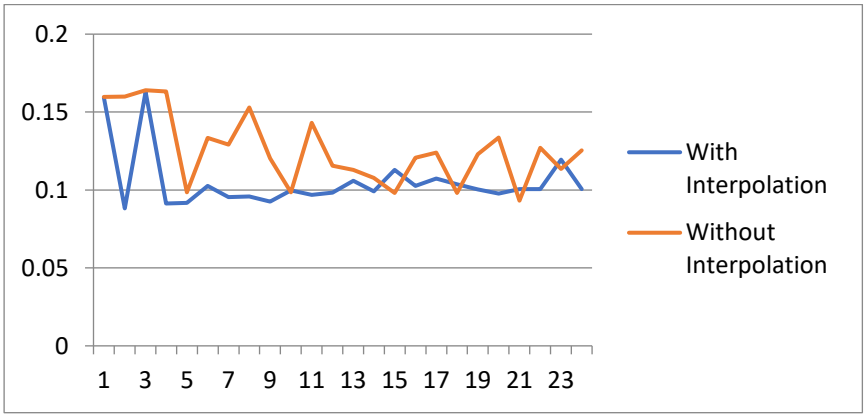

Fig 15. *NRMSE performance of different lag length in ANN*

As acknowledged from the graphs, KNN, ANN, DT regressors have random plot performance. The interpolation of data in the train set also has a random effect on the regressors. It increases the NRMSE values for certain lag lengths while decreases for some cases. For RF regressor, the interpolated data doesn't perform well against the actual data.

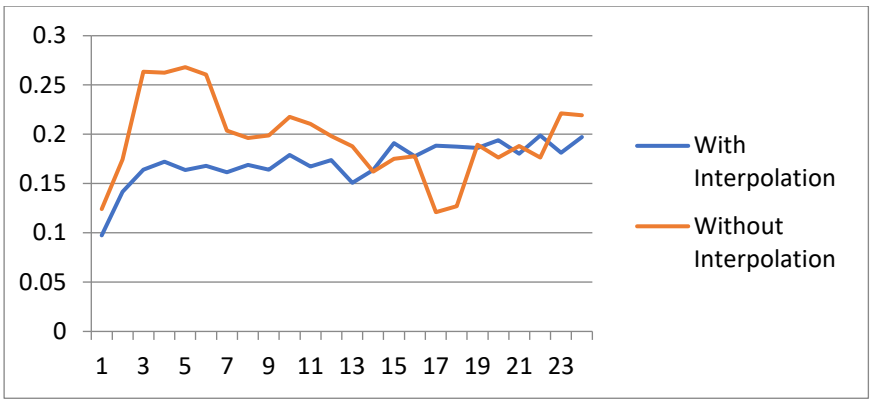

Fig 16. *NRMSE performance of different lag length in Decision Tree*

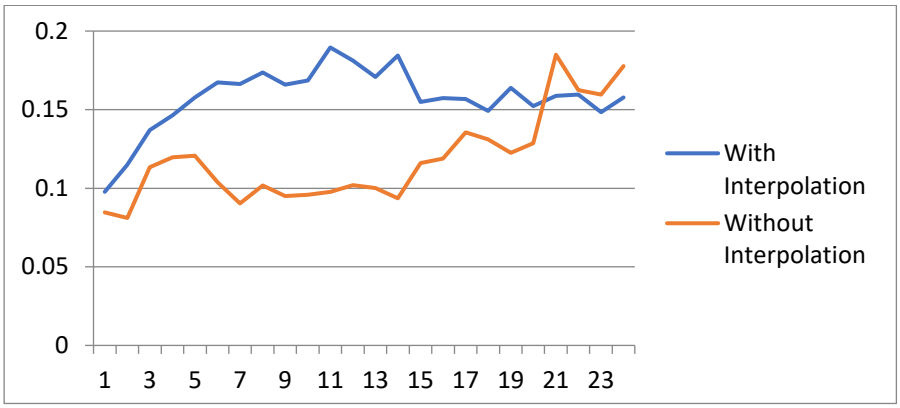

Fig 17. *NRMSE performance of different lag length in RBFN*

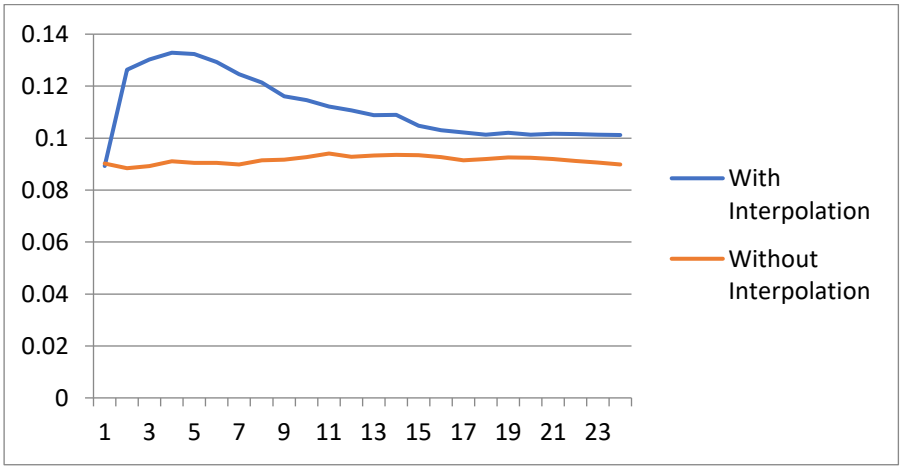

Fig 18. *NRMSE performance of different lag length in SVR*

The interpolation of datapoints have an inverse impact on the NRMSE performance initially for the SVR regressor. As the lag length increases, the values converge but actual data outperforms the interpolated data.

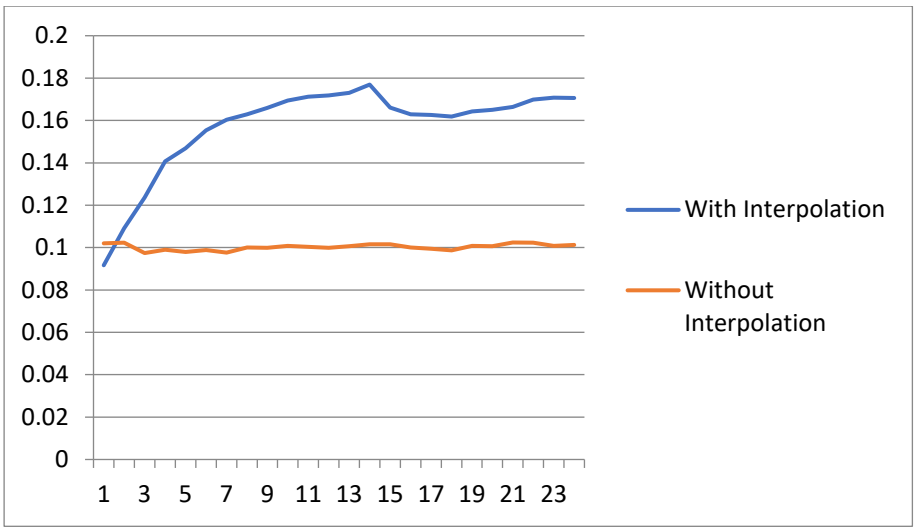

Fig 19. *NRMSE performance of different lag length in Ridge*

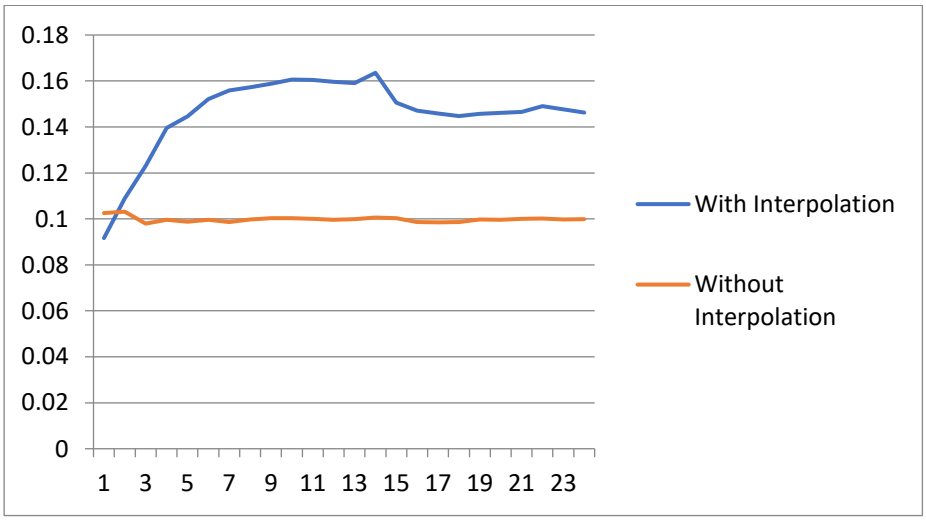

Fig 20. *NRMSE performance of different lag length in Bayesian Ridge*

For Ridge and Bayesian Ridge regressors in this dataset, actual data has linear NRMSE values whereas the interpolated train set model behaves like a quadratic plot with the increase of lag length.

6.3 Overall Performance

The plot of top two performers varying on the lag length, with and without interpolation, and NRMSE values as shown in Fig 19 and 20.

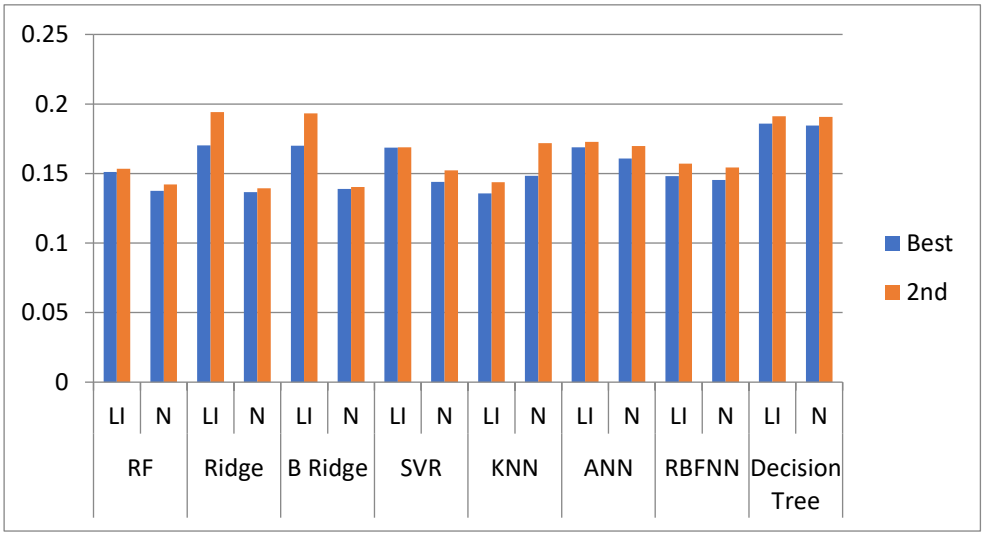

Fig 21. *Top two performers of each regressor in Musa Dataset*

For the Musa Dataset; Ridge, Bayesian Ridge and RF regressors are the best performers when measuring on the actual dataset. KNN outperforms others with the interpolated dataset. DT, RF, Ridge and Bayesian Ridge with interpolation data are the worst performers for this data.

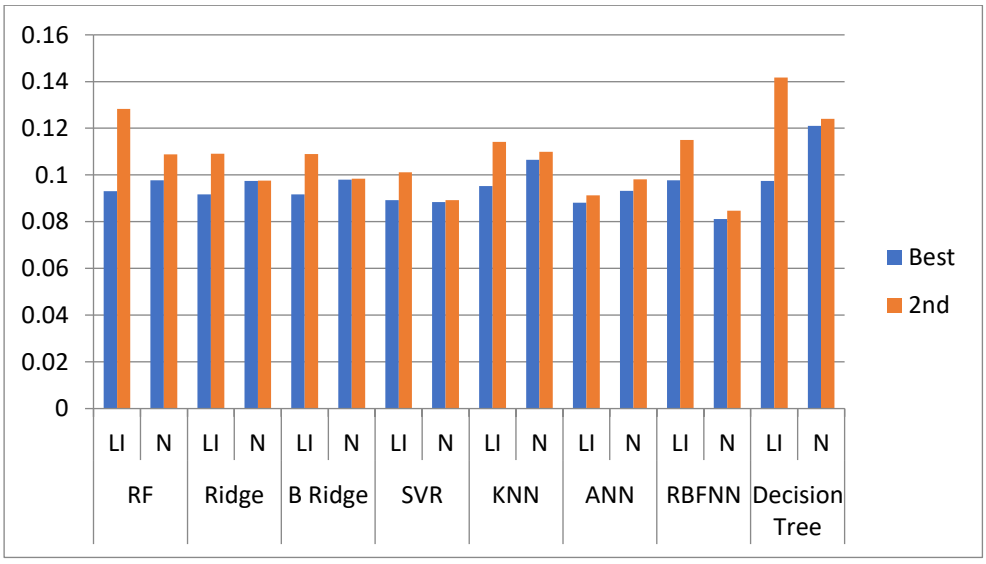

Fig 22. *Top two performers of each regressor in Iyer Lee Dataset*

RBFN and SVR are the better performers on the Iyer Lee dataset while measuring on the actual dataset. Other regressors have similar performance on this dataset with minimum variation. DT is the worst performer among all.

Table 1. *NRMSE performance comparison*

Lag Length	ANN		ANN PSO		Ridge	
	Musa	**Iyer Lee**	**Musa**	**Iyer Lee**	**Musa**	**Iyer Lee**
2	0.1593	0.5129	0.1686	0.3908	0.176869	0.091658
3	0.1415	0.4939	0.1628	0.3765	0.171176	0.109134
4	0.053	0.541	0.1451	0.3245	0.159977	0.12362
5	0.1528	0.5321	0.1283	0.3098	0.155523	0.140684
6	0.1634	0.5642	0.1529	0.3472	0.151145	0.146877
7	0.1682	0.5871	0.1782	0.3981	0.142872	0.155379

Table 2. *Performance comparison with existing algorithms performance*

Statistical Models	Musa	Iyer Lee
Jelinski Moranda	0.1444	-
Geometric	0.1419	3.5375
Musa Basic	0.1413	-
Musa Okumoto	0.1409	3.3255
ANN	0.1415	0.4939
ANN-PSO	0.1283	0.3098
KNN	0.1357	0.09525
Ridge	0.1366	0.09165

The performance comparison with existing statistical models and reference model (Bisi-Goyal et al. 2017) is given in the Table 1 and Table 2. For Iyer Lee dataset, Ridge Regressors have outperformed existing algorithms whereas ANN-PSO still remains the best performer for Musa dataset.

Chapter 7

Conclusions

Different machine learning prediction algorithms are used in this work of software reliability prediction. As seen from the results above the Ridge regressor along with the Bayesian Ridge regressor performs better on both the datasets while working on actual data. The performance is better compared with existing algorithms on the NRMSE values. Certain models perform better while interpolating points like KNN, ANN.

As a whole, logarithmic scaling was an important factor in the improvement of the model performance because of uneven distribution data values and the time series datasets having peaks at certain points. As observed Ridge, Bayesian Ridge, RBFN regressors perform their best while the lag length is around 7 to 11. RF, ANN regressors give their best while increasing the lag length but on the actual dataset. KNN and SVR regressors were performing better on the linearly interpolated data which was the best performance among all the cases in certain lag lengths.

Acronyms

- **TBSF:** Time Between Successive Failures
- **ANN:** Artificial Neural Network
- **KNN:** K-Nearest Neighbours
- **ML:** Machine Learning
- **SVM:** Support Vector Machine
- **SVR:** Support Vector Regression
- **IEEE:** Institute of Electrical & Electronics Engineers
- **ANSI:** American National Standard Institute
- **RBFN:** Radial Basis Function Network
- **PSO:** Particle Swarm Optimization
- **NRMSE:** Normalized Root Mean Squared Error
- **SSE:** Sum of Squared Error
- **RE:** Relative Error
- **RF:** Random Forest
- **DT:** Decision Tree

Bibliography

[1] Bisi, M., & Goyal, N. K. (2017). *Artificial Neural Network Applications for Software Reliability Prediction*. John Wiley & Sons.

[2] Aljahdali, S. H., & Buragga, K. A. (2008). Employing four ANNs paradigms for software reliability prediction: an analytical study. *ICGST International Journal on Artificial Inteligence and Machine Learning*, *8*(2), 1-8.

[3] Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, *18*(7), 563-574.

[4] Quyoum, A., Dar, M. D., & Quadri, S. M. K. (2010). Improving software reliability using software engineering approach-A review. *International Journal of Computer Applications*, *10*(5), 41-47.

[5] Aggarwal, K. K., Singh, Y., Kaur, A., & Malhotra, R. (2006). Investigating the effect of coupling metrics on fault proneness in object-oriented systems. *Software Quality Professional*, *8*(4), 4.

[6] Goel, B., & Singh, Y. (2009). An empirical analysis of metrics to predict the maintainability for real-time object-oriented software. *Software Quality Professional*, *11*(3), 35.

[7] Malhotra, R., Kaur, A., & Singh, Y. (2010). Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. *International Journal of System Assurance Engineering and Management*, *1*(3), 269-281.

[8] Li, X., Li, X., & Shu, Y. (2007, September). An early prediction method of software reliability based on support vector machine. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing* (pp. 6075-6078). IEEE.

[9] Lo, J. H. (2010, May). Predicting software reliability with support vector machines. In *2010 Second International Conference on Computer Research and Development* (pp. 765-769). IEEE.

[10] Singh, Y., & Kumar, P. (2010, December). Prediction of software reliability using feed forward neural networks. In *2010 International Conference on Computational Intelligence and Software Engineering* (pp. 1-5). IEEE.

[11] Costa, E. O., Pozo, A. T. R., & Vergilio, S. R. (2010). A genetic programming approach for software reliability modeling. *IEEE transactions on reliability*, *59*(1), 222-230.

[12] Ho, S. L., Xie, M., & Goh, T. N. (2003). A study of the connectionist models for software reliability prediction. *Computers & Mathematics with Applications*, *46*(7), 1037-1045.

[13] Su, Y. S., & Huang, C. Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software*, *80*(4), 606-615.

[14] Pai, P. F., & Hong, W. C. (2006). Software reliability forecasting by support vector machines with simulated annealing algorithms. *Journal of Systems and Software*, *79*(6), 747-755.

[15] Kumar, P., & Singh, Y. (2012). An empirical study of software reliability prediction using machine learning techniques. *International Journal of System Assurance Engineering and Management*, *3*(3), 194-208.

[16] Musa, J. D., Iannino, A., & Okumoto, K. (1990). Software reliability. *Advances in computers*, *30*, 85-170.

[17] Vishwanath, S. P. (2006). *Software reliability prediction using neural networks* (Doctoral dissertation, IIT Kharagpur).

[18] Jaiswal, A., & Malhotra, R. (2018). Software reliability prediction using machine learning techniques. *International Journal of System Assurance Engineering and Management*, *9*(1), 230-244.