PROGRESS REPORT

ON

Software Fault Prediction

SUBMITTED BY

Ayush Kumar - 2020UIT3072

Shreyansh Swaroop – 2020UIT3079

Anupam Adarsh – 2020UIT3136

UNDER THE GUIDANCE OF

Dr. Ankita Bansal

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

TABLE OF CONTENTS

INTRODUCTION

Software

Software is a collection of instructions, information, or computer programmes needed to run a computer and carry out particular activities. Software instructs a computer on how to run. Most computers would be worthless without software. As an illustration, a web browser is a piece of software that enables people to access the internet. The software programme known as an operating system (OS), acts as the conduit between other programmes and the hardware of a computer or mobile device. Software is therefore crucial in today's computer-driven environment.

Software Fault Prediction

One crucial step in the software development process is the accurate identification of software defects. A crucial step in the software development process is defect identification. Reducing software defects is very desired for a software development project. The primary objective of creating program patterns is to locate software defects.

Finding defects in a vast and sophisticated software system is the major challenge of software fault detection.

The main goal of software fault prediction is to use the underlying properties of the source code of a software project to predict faults before the actual testing process begins. This will help in prioritizingthe work in the testing process.

MOTIVATION

Software fault prediction is a crucial step in the process of developing new software. A computer is guided in how to operate by software, which plays a crucial role. The majority of computers are unusable without software. On a computer or mobile device, for instance, an operating system (OS) is a software program that acts as the interface between other program and the hardware. In our computer-driven world of today, software is therefore crucial. Consequently, finding defects is a crucial step in the software development process. Reducing software defects is very desired for a project developing software. Finding software defects is the fundamental purpose of creating program patterns. For the purpose of identifying dataset defects, we applied machine learning models together with several feature extraction strategies.

APPLICATION

The application of the project is primarily focused on improving the software development process, reducing costs, improving software quality, maintenance processes and ultimately delivering more reliable and secure software products to end-users. Here are some key applications of software fault prediction:

- **Early Defect Detection:** The primary application is to detect potential software faults, defects, or bugs early in the development process. By identifying these issues before the software is deployed, teams can proactively address them, reducing the likelihood of critical errors reaching production.

- **Resource Allocation**: Software fault prediction helps in optimizing resource allocation during development and testing phases. Teams can prioritize their efforts on the most error-prone parts of the code, allowing for more efficient use of time and resources.

- **Quality Assurance:** It aids in improving software quality by preventing defects from propagating into the final product. This leads to a reduction in post-release maintenance and bug-fixing efforts.

- **Security Enhancement:** Identifying potential security vulnerabilities early is crucial for building secure software. Software fault prediction can help in identifying code patterns that might lead to security issues, allowing for timely security enhancements.

- **Maintenance Cost Reduction:** Predicting and addressing software faults before deployment can significantly reduce the long-term maintenance costs associated with software projects. It minimizes the need for ongoing bug fixes and updates.

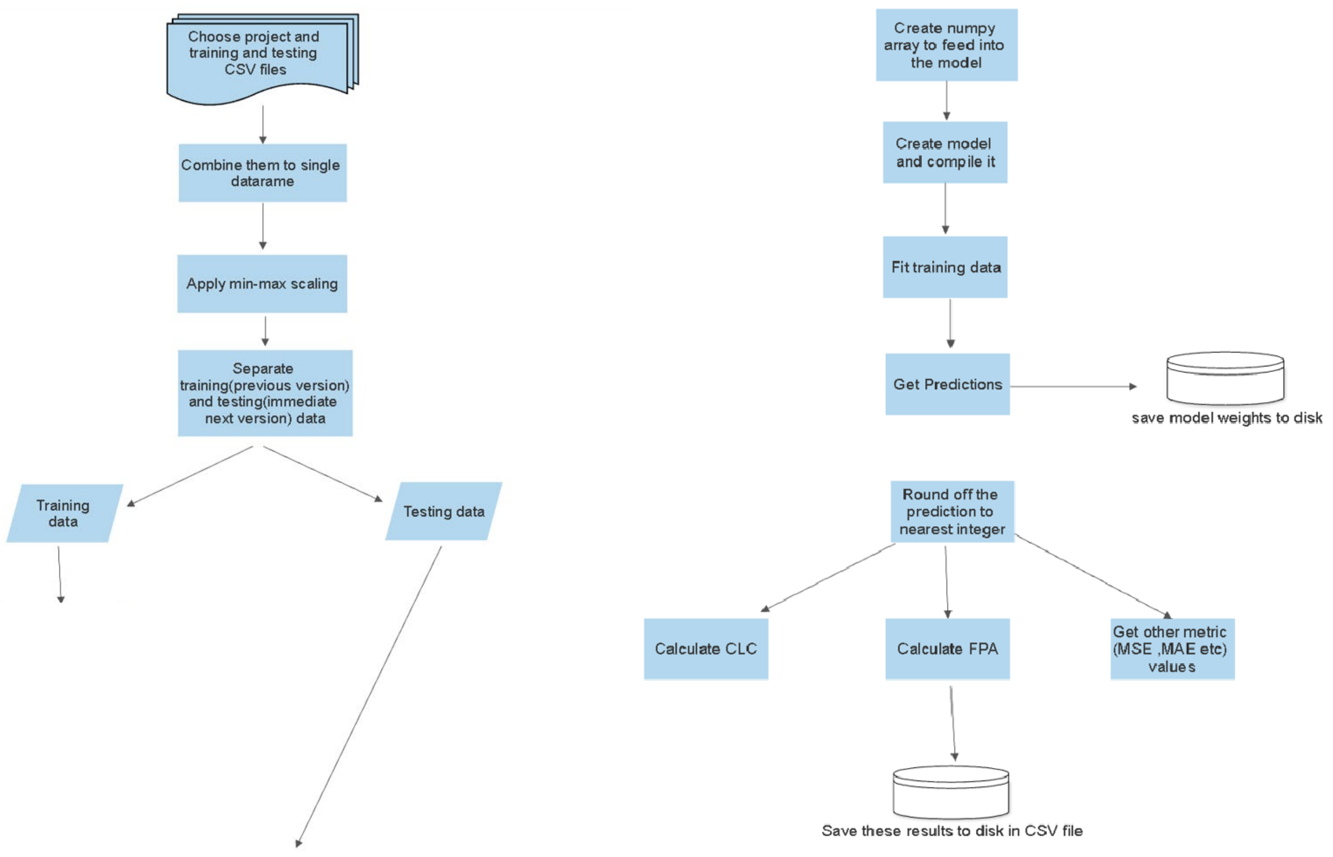
- **Risk Management:** By identifying areas of the codebase that are prone to faults, software fault prediction contributes to risk management. Teams can focus on mitigating high-risk areas, reducing the likelihood of critical software failures.

METHODOLOGY

Regression Using Machine Learning:

Exploratory data analysis has been done on the two versions of the project which were taken for training and testing the model. The analysis included checking the presence of null data points, making sure all the features had unique values for the datasets, checking the datatypes of the features and visualizing the scatter plot and frequency distribution of the data for each feature. As per the analysis done, no data cleaning or feature filteration was needed. A specific version ant-1.3 was taken to train the data and ant-1.4 version was taken for testing the model built. Min Max scaling was applied and model performance were checked without and with two different feature reduction techniques: SVD and PCA with each taking two different number of components hyperparameter values. The model were developed using tensorflow keras in python. We obtained the training and testing time of the model using the time() function from time package. The predictions obtained were round off to the nearest integer using np.rint() from numpy package because the target value to be predicted was a strict integer. Following the decision in the paper Ridge and Lasso Regression Models for Cross Version Fault Prediction by Xiaoxing Yang and Wushao Wen we have considered FPA (Fault-percentile-average) and CLC (Cumulative lift chart (also the area under CLC)) as our main model evaluation metrics. At the end of each experiment, we have automated the saving of the model weights and results in csv.

The detailed flow chart of the experiments done for regression using machine learning is given:



Choose project and training and testing CSV files
Combine them to single datarame
Apply min-max scaling
Separate training(previous version) and testing(immediate next version) data
Training data
Testing data
Create numpy array to feed into the model
Create model and compile it
Fit training data
Get Predictions
save model weights to disk
Round off the prediction to nearest integer
Calculate CLC
Calculate FPA
Get other metric (MSE ,MAE etc) values
Save these results to disk in CSV file

WORK DONE

Dataset Availability

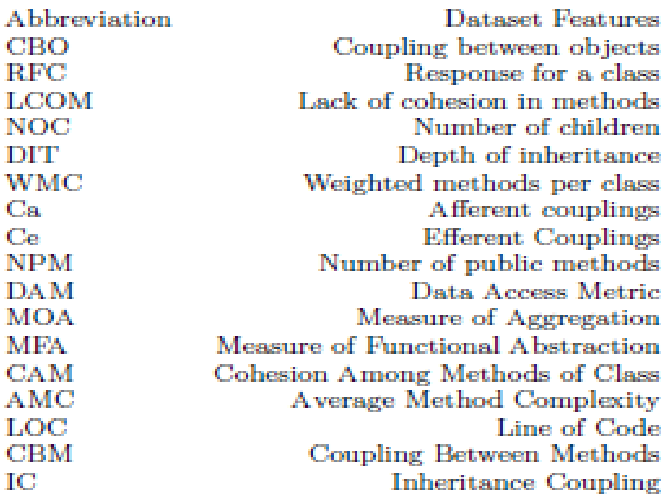
The datasets for training and testing the models are taken from the PROMISE repository which consists of 41 different versions of projects. The datasets are taken from 11 open-source projects. Data is collected from three or more versions from each of the open-source projects.

We used former version of a project as the training data of our models and tested the models predictions on the latter version for the machine learning regression models developed.

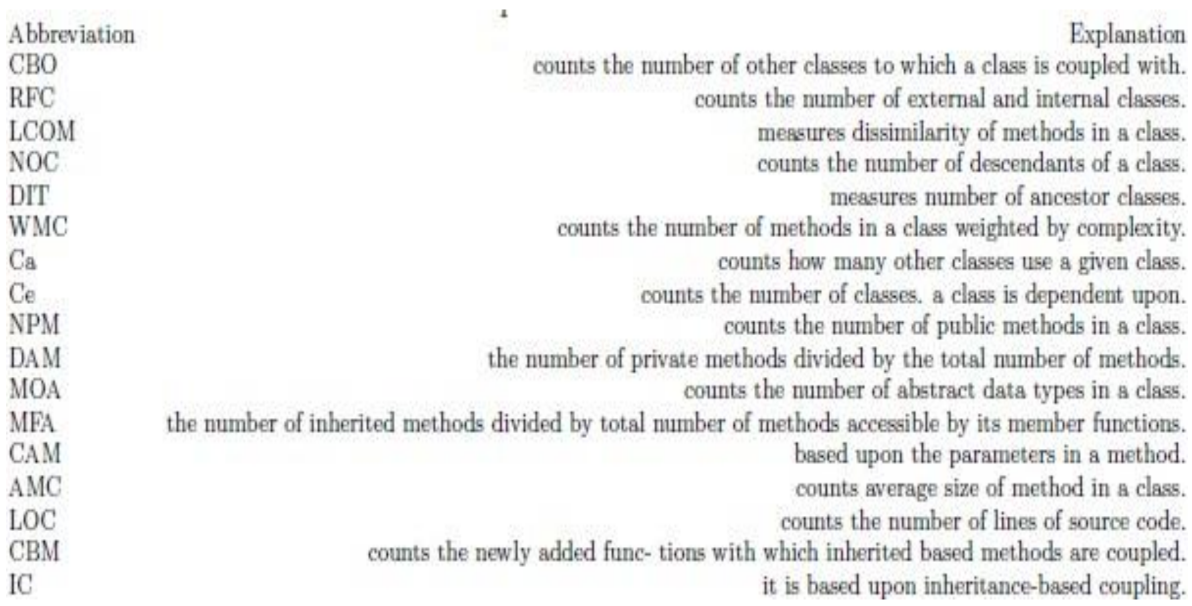
For the machine learning regression problem, we took that the datasets which had the target value as the number of faults corresponding to each data point.

Description of Dataset

The following matrices are proposed by Chidamber and Kemerer, Henderson-Sellers, Martins, QMOOD and Tang et al. The features in the datasets taken are:



Abbreviation	Dataset Features
CBO	Coupling between objects
RFC	Response for a class
LCOM	Lack of cohesion in methods
NOC	Number of children
DIT	Depth of inheritance
WMC	Weighted methods per class
Ca	Afferent couplings
Ce	Efferent Couplings
NPM	Number of public methods
DAM	Data Access Metric
MOA	Measure of Aggregation
MFA	Measure of Functional Abstraction
CAM	Cohesion Among Methods of Class
AMC	Average Method Complexity
LOC	Line of Code
CBM	Coupling Between Methods
IC	Inheritance Coupling



Abbreviation	Explanation
CBO	counts the number of other classes to which a class is coupled with.
RFC	counts the number of external and internal classes.
LCOM	measures dissimilarity of methods in a class.
NOC	counts the number of descendants of a class.
DIT	measures number of ancestor classes.
WMC	counts the number of methods in a class weighted by complexity.
Ca	counts how many other classes use a given class.
Ce	counts the number of classes. a class is dependent upon.
NPM	counts the number of public methods in a class.
DAM	the number of private methods divided by the total number of methods.
MOA	counts the number of abstract data types in a class.
MFA	the number of inherited methods divided by total number of methods accessible by its member functions.
CAM	based upon the parameters in a method.
AMC	counts average size of method in a class.
LOC	counts the number of lines of source code.
CBM	counts the newly added func- tions with which inherited based methods are coupled.
IC	it is based upon inheritance-based coupling.

Cross Version Fault Prediction

Metrics are extracted from a software project's specific version's source code and immediate next version's source code. Now specific version's data collected are collectively taken as the training data for training the model and the immediate next version's data collected as the collective testing data.

Machine Learning Regression

Machine learning models are built to predict the number of faults corresponding to a datapoint in the testing dataset. Since the predictions are strictly greater than zero, we have used ReLU activation function after the output layer in the model. Different concepts used in the feature engineering in these experiments are:

- **Min Max Scaling**

Min Max scaling is done to rescale a feature or observation value to a distribution value between 0 and 1. The formula followed is:

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

- **SVD (Singular Value Decomposition)**

The singular value decomposition (SVD) is a factorization of a real or complex matrix that generalizes the eigen decomposition of a square normal matrix to any matrix via an extension of the polar decomposition.

- **PCA (Principal Component Analysis)**

Principal Component Analysis (PCA) is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity.

- **FPA (Fault-Percentile-Average)**

Considering m modules f1, f2, . . ., fm listed in increasing order of predicted defect number, si as the actual defect number in the module fi, and s = s1 + s2 + ... + sm as the total number of defects in all the modules, the proportion of actual defects in the top t predicted modules (i.e., top t modules predicted to have most defects) to the whole defects $\frac{1}{s}\sum_{i=m-t+1}^{m} s_i$. Then FPA is defined as follows:

$$\frac{1}{m}\sum_{t=1}^{m}\frac{1}{s}\sum_{i=m-t+1}^{m} s_i$$

- **CLC (Cumulative Lift Chart)**

CLC (also the area under CLC) uses percentages of modules as x-axis and percentages of defects as y-axis. The area under CLC is always used for comparison, and the area is simply denoted as CLC in this paper. Considering m modules f1 , f2 , ..., fm , listed in increasing order of predicted defect number, si as the actual defect number in the module fi, and s = s1 + s2 + ... + sm as the total number of defects in all the modules, the area under the curve should be computed as the sum of areas of trapezoid composed of two adjacent points and the axes in the CLC. UsingCLC to denote the area under the curve in rest of the paper, it can be computed as follows:

$$CLC = \sum_{t=1}^{m} trapezoid_t$$

$$= (\frac{1}{m})(\frac{1}{2})((0 + \frac{s_m}{s} + ...$$

$$+(\frac{s_m + ... + s_2}{s} + \frac{s_m + ... + s_1}{s}))$$

Experimental Results

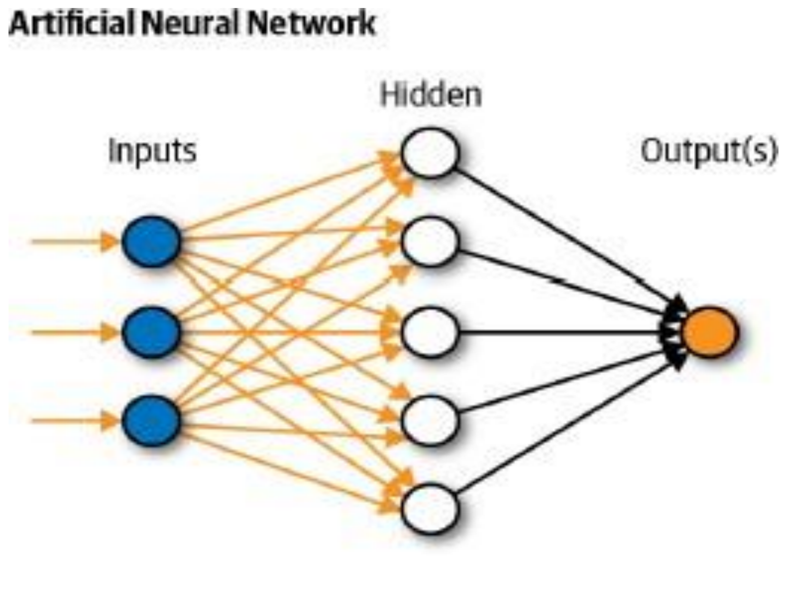
We performed 5 experiments without any feature selection and with two different feature selection techniques:

- Singular Value Decomposition (SVD)
- Principal Component Analysis (PCA)

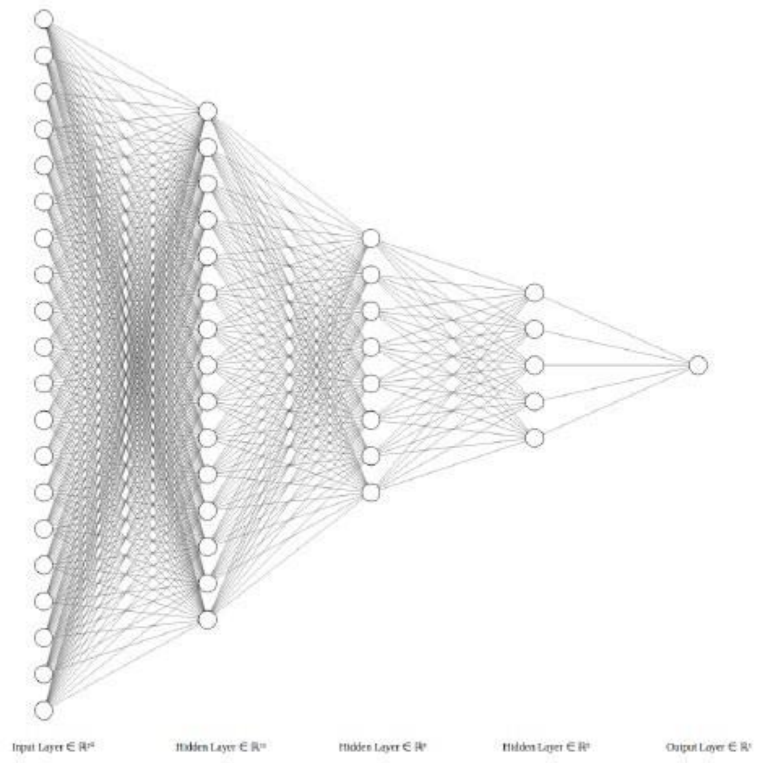
and each of SVD and PCA with two different number of components: 10 and 15. Min Max scaling pre- processing was performed on the datasets before running the model. The predictions obtained by the model were rounded off to nearest integers. We used MSE as the loss function for the model.

Artificial Neural Network (ANN)

Artificial Neural Networks are a special type of machine learning algorithms that are modelled after the human brain. That is, just like how the neurons in our nervous system are able to learn from the past data, similarly, the ANN is able to learn from the data and provide responses in the form of predictions or classifications. In a neural network, there are three essential layers –
Input Layer - It is the first layer of an ANN that receives the input information in the form of various texts, numbers, audio files, image pixels, etc.
Hidden Layer - In the middle of the ANN model are the hidden layers. There can be a single hidden layer, as in the case of a perceptron or multiple hidden layers. These hidden layers perform various types of mathematical computation on the input data and recognize the patterns that are part of.
Outer Layer - In the output layer, we obtain the result that we obtain through rigorous computations performed by the middle layer.



We have developed single architecture of ANN. The schematics of the ANN model designed is visualized as follows:


Input Layer $\in \mathbb{R}^{20}$
Hidden Layer $\in \mathbb{R}^{15}$
Hidden Layer $\in \mathbb{R}^{8}$
Hidden Layer $\in \mathbb{R}^{5}$
Output Layer $\in \mathbb{R}^{1}$

The performance results are as follows:

Observing the results, we can see that the FPA and CLC is obtained to be NAN for two of the experiments. This situation is encountered when all the predictions i.e the number of bugs in the testing data come out to be zero after rounding off. This is an unwanted result. We also see that the best FPA and CLC are obtained when no feature selection technique is applied.

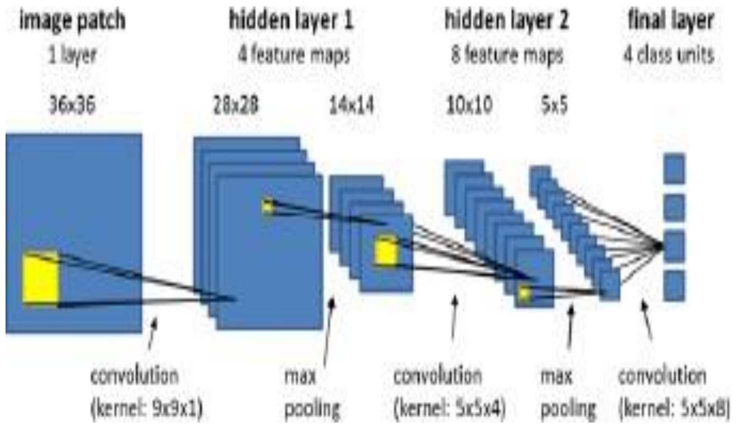
Feature Selection	No. of components	FPA	CLC	Train Loss at last epoch(MSE)	Test Loss(MSE)
N/A	**N/A**	**0.47545615**	**0.48384327**	**0.555**	**1.1668**
SVD	10	0.47456595	0.48294193	0.385	0.907
SVD	15	NAN	NAN	3.5	0.3652
PCA	10	NAN	NAN	3.5	0.3652
PCA	15	0.4627341	0.47109878	0.2115	0.8144

WORK TO BE DONE

Different classical machine learning techniques could be applied on the same data and checked for better results.

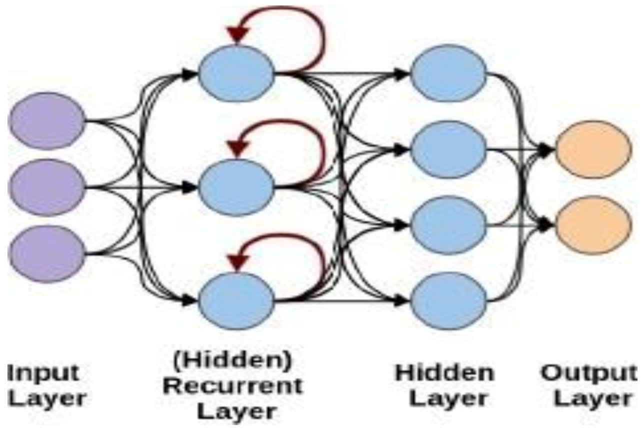
Convolutional Neural Network (CNN)

A convolutional neural network is a class of neural network, most commonly applied to analyze visual imagery. A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions.



Recurrent Neural Network (RNN)

A recurrent neural network is a class of neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. It is derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior.



REFERENCES

[1] C.-P. C. Ko-Li Cheng and C.-P. Chu, *"Software fault prediction using program patterns"* IEEE 2nd International Conference on Software Engineering and Service Science 2011.

[2] W. W. Xiaoxing Yang, *Ridge and Lasso Regression Models for Cross-Verison Fault Prediction*, vol. 67 No. 3, 2018.

[3] J. Schmidhuber, *"Machine learning in neural networks: An overview*", 2014.

[4] X. L. Hamza Turabieha, Majdi Mafarja*, "Iterated feature selection algorithms with neural network for software fault prediction",* 2019.

[5] S. W. N. T. T. V. G. Hoa Khanh Dam, Trang Pham and A. Ghose, *"A tree-based model for software fault prediction",* 2018.