IMPROVING SOFTWARE FAULT PREDICTION BY HANDLING NOISY DATA

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

HIMANSHI

(17535007)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247667 (INDIA)
MAY, 2019

DECLARATION

I declare that the work presented in this dissertation with title, "**IMPROVING SOFTWARE FAULT PREDICTION BY HANDLING NOISY DATA**", towards the fulfillment of the requirements for award of the degree of Master of Technology in Computer Science & Engineering, submitted to the Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India is an authentic record of my own work carried out during the period from May 2018 to May 2019 under the guidance of Dr. Sandeep Kumar, Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology, Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other institute.

(HIMANSHI)

CERTIFICATE

This is to certify that the statement made by the candidate in the declaration is correct to the best of my knowledge .

Date: ……..
Place: Roorkee

Signed: ……..
Dr. Sandeep Kumar
(Associate Professor)
Indian Institute of Technology
Roorkee

ABSTRACT

Software fault prediction is the procedure to foresee whether a module in software is faulty or not by utilizing the past information and some learning models. From the previous version of the software past information is collected. The performance of a classifier depends upon various factors and quality of dataset is one among them. Real world datasets often contains noise which degrades the classifier's performance. So, to remove the noise in dataset we propose a two stage pre-processing, which combines rough set theory followed by oversampling and denoising auto encoder to extract the noise robust version of original dataset. In first stage we collect the certain instances from dataset using rough set theory followed by oversampling for handling class imbalance. In second stage, we extract the robust to noise version of original dataset with the help of denoising auto encoder. The proposed approach is being evaluated on NASA MDP dataset and Eclipse dataset in order to show the effectiveness of proposed approach. Further this work tries to study various denoising techniques present in literature.

ACKNOWLEDGMENT

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

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

HIMANSHI

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software Fault Prediction is the mechanism to predict whether in a software the modules are going to be faulty or non faulty, before even applying the testing mechanism. In other words, Fault Prediction in Software is a way to find the fault proneness of the software module during the earlier stages of development life cycle process. This prediction has a great role to play in improving the quality of the software as well as reducing the time and efforts needed in the testing phase of the development life cycle of the software. This chapter describes the basic terminologies and brief about topics that are needed for understanding of this work.

1.1 Software Fault Prediction

Now days with the increase in technology there is great development happening in the field of software fault prediction. Software now a days are used widely in a variety of areas out of which many are used in real time applications which are associated with a lot of financial, economical and even human life risks. Thus there need of some mechanisms to reduce the rate of software failures to avoid the associated risks involved with them. Here comes Software Fault Prediction [16]. Software fault or default prediction is the prediction whether a module in software is faulty or not by using the previous data and some learning models. The previous data is available from the previous versions of the software. Thus software fault prediction makes use of the data of previous versions of the software to find out the probability of faults in the upcoming versions of that software based on some characteristics known as metrics, by applying some learning model. More testing efforts are made in the module which is predicted as faulty as compared to the one predicted as non faulty [17].

1.2 Software Fault Prediction Methodology

Software Fault prediction process involves two steps, i.e. training and testing. A prediction model is developed using the data and metrics form previous versions of the software during training phase and this model is used to predict the presence of faulty modules in the new versions of the software during testing phase [18]. The data from the previous version of the software (inter release experiments) or from some other software belonging to the related domain (cross project software fault prediction) is used to train the prediction model. A suitable classifier is trained using this data. This trained prediction model is now fed with some new data over which testing mechanism is to be

performed. In case of binary classification of the Software Fault Prediction, the prediction model gives output in the terms of module being faulty or non faulty.

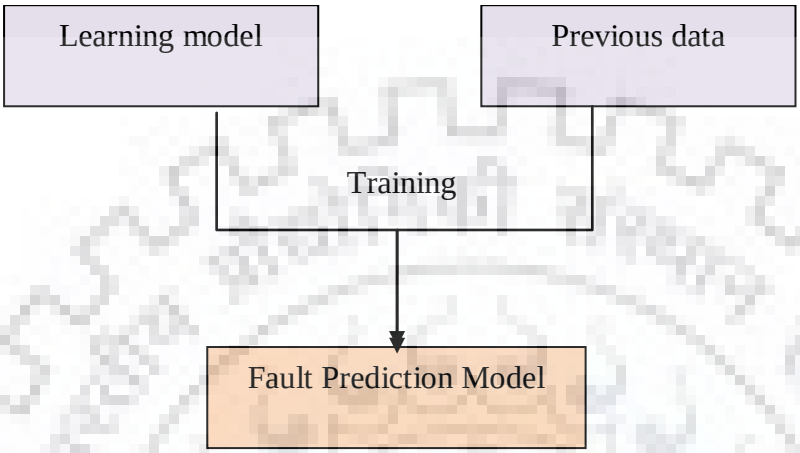

Figure 1.1 Training phase in Software fault Prediction

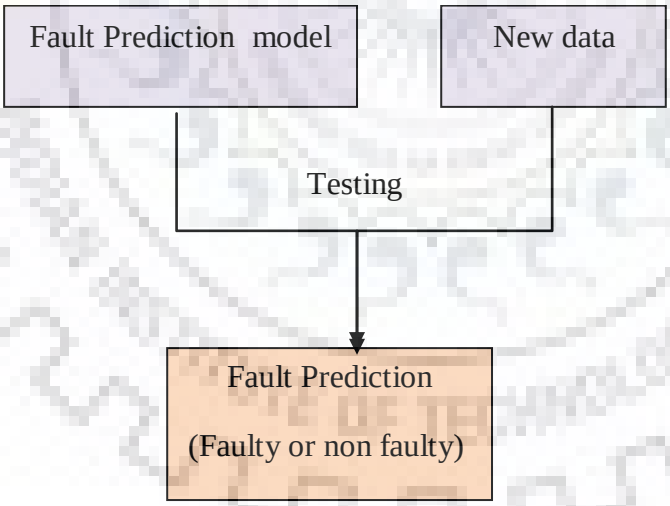

Figure 1.2 Testing phase in Software fault Prediction

1.3Noise in dataset

Real-world datasets can be affected by many reasons. Among them the presence of noise is mostly found. Noise can be introduced in either data collection or in data preparation processes. Noise can have two sources: implicit errors [2] which can be introduced by measuring tools eg.

sensors and random errors [2] which can be introduced while gathering the data by an expert. The performance of the classifiers depends heavily on the quality of the dataset provided to it and also on its robustness against noise. The noise can prove to be an unavoidable problem. If the performance of prediction model is to be maximized, then it has to be trained with a dataset which is not affected by noise. There can be two types of noise present in the dataset class noise and attribute noise [8].

Class noise [9] refers to the condition when for the same set of attributes we have contradictory class labels or mislabelled examples. Attribute noise [9] refers to the condition when there is erroneous values, missing values or do not care values.

1.4 Denoising Techniques

In the literature, there are two types of methods provided to reduce the amount of noise present in the dataset. First one is to use the Noise Filters and second one is to modify some pre-existing classification techniques to reduce the effect of noise on its decision making.

1.4.1 Noise Filters

Iterative filter [11]: In this approach, first the dataset is divided into n equal parts. Then a base classifier is built from each of partitions of data. Then testing is performed on the whole dataset. Removal of noisy instances is decided by majority or consensus voting. Finally, a proportion of noisy instances are removed whose label disagrees with the judgment of classifiers.

Multiple partitioning filters [11]: Combines m different base classifier built on different splits of training data. Multiple partitioning filter identifies an instance as noisy if it is misclassified by at least α models, where α is filtering level.

1.4.2 Denoising Auto encoder

An auto encoder is a type of artificial neural network used to learn efficient data coding in an unsupervised manner [3]. Auto encoder performs data compression algorithm which is learned automatically from the input features rather than done by the humans. Autoencoding includes compression and decompression functions which are data specific and are lossy [3]. Being data specific, autoencoders can only compress similar kind of data on which they have been trained. Auto encoders are widely used for denoising and data compression. In denoising auto encoder, some amount of noise is added to the training set and then the neural network is made to minimize the loss between the actual training set and noise induced training set. The auto encoder while performing compression and decompression is made to denoise the data.

1.5 Rough Set Theory

Rough set theory is a new mathematical approach to imperfect knowledge. Rough set theory provides efficient algorithms for finding hidden patterns in data, finds minimal sets of data (data reduction), evaluates significance of data, generates sets of decision rules from data and etc [19]. The Rough Set Theory has an advantage over other data analysis techniques i.e. it does not require any prior information about data − like probability in statistics. Some of the basic concepts of rough set theory include Information table, decision table, lower approximation, reduct, core and etc.

1.5.1 Information Table

Data represented as a table, in which each row can represents an event or simply an object. The columns represents attributes (a property or a variable) that can be measured for each instance of dataset. The attribute value can be human engineered or can be taken from some measuring device. This is called an information table.

EXAMPLE : An example of information system is given below having two instances x1,x2 and two attributes Age and Sex.

Objects	Age	Sex
x1	55	F
x2	78	M

Table 1.1 Information table

1.5.2 Decision Table

A decision system is any information system having additional decision attribute. The elements of decision table are called conditional attributes or simply conditions [19].

EXAMPLE :

Objects	Age	Sex	Walk
x1	5	F	No
x2	78	M	Yes

Table 1.2 Decision Table

1.5.3 Indiscernibility

Let I= (U,A) be an information system, where U is a non-empty, finite set of objects (the universe) and A is a non-empty, finite set of attributes.

With any $R \subseteq A$ there is an associated equivalence relation $IND(R)$

$$IND(R) = \{(x, y) \in U^2 | \forall_a \in R, a(x) = a(y)\} \quad (1.1)$$

1.5.4 Lower approximation

The lower approximation [19] of target set X is set of all observation which can be classified with certainty as member of X with respect to equivalence classes induced by R.

$$\underline{R}(X) = \{ x \mid [x_R] \subseteq X\} \quad (1.2)$$

Were $[x_R]$ is equivalence class of x induced by R

1.6 Organization of thesis

This report is divided into 6 chapters. First chapter concluded the preliminaries and basic concept knowledge that will be needed for understanding this thesis. Second chapter is literature review which includes papers from software fault prediction domain as well as from other domains using denoising techniques. This chapter highlights the research gaps found in literature and presents the tabular representation for the same. Third chapter contains detailed objective of this thesis including proposed architecture. Fourth chapter shows the experimental result of existing techniques as well as of proposed architecture. Fifth chapter concludes the entire work including the analysis of results and future work that can be done.

CHAPTER 2

LITERATURE REVIEW

Several works has been done till now in for removing noisy data from software prediction dataset using different techniques like noise filter, changing working of algorithm and for various types of noise. A lot of work and empirical analysis has been done in other fields apart from software fault prediction for removing noisy instances.

2.1 Removal of class label noise in software fault prediction

Khoshgoftaar [11] presented two noise filtering techniques namely iterative filter and multiple partitioning. In iterative filter, noisy instances were removed in multiple iterations using classification filter. Multiple partitioning filter was based on the fact that single base learner can become biased. To remove this condition, different base learner were used on same partitions of training set and majority voters of learned base learners were used to classify test instance as noisy or clean. Riaz et al [4] proposed two-stage data pre-processing technique, which used rough set theory as a pre-processing technique to eliminate noisy examples from the datasets. Both feature selection and rough set-based K nearest neighbour rule (KNN) noise filter was used before executing easy ensemble for classification. Noisy samples from both the minority and the majority class were removed from the boundary regions by checking K-nearest neighbour of that point and then deciding its class label. If the predicted label does not match with the actual label then that instance is removed.

2.2 Removal of class label noise in fields apart from software fault prediction

Gamberger et al [10] first introduced the concept of noise filtering scheme in medical domain. They introduced classification filter which proved to be a very basic yet efficient technique for noisy instances removal. In classification filter, one base learner is trained on k-1folds of dataset and kth fold is used for testing and instances identified as noisy in this dataset are removed. Seaz et al [20] implemented a noise filter which is combination of three techniques namely Ensemble learning, Iterative partitioning techniques and metric based partitioning. The first two strategies (use of multiple classifiers and iterative filtering) are used to get better judgment about noisy instances, whereas the last one helps to controls the amount of noisy instances used. a noise score which is calculated using KNN on entire set of noisy instances is used to classify instance as noisy or not.

2.3 Removal of attribute noise in other software fault prediction

Tong et al [5] proposed a two-phase Software Defect Prediction model, which is built by using stacked denoising auto encoders and ensemble learning. Author proposed the use of deep representation of software metrics extracted by stacked denoising auto encoders instead of using traditional software metrics. In second stage, a proposed two stage ensemble learning method is used in order to tackle the problem of over fitting in existing models. David et al [31] and Petric et al [30] questioned the quality of datasets of NASA MDP datasets as they are generated from closed source. Gray et al [29] found significant amount of erroneous data points which even contained impossible metrics values.

2.4 Removal of attribute noise in fields apart from software fault prediction

Vincent et al [21] used stacked denoising auto encoder in order to get noise free representation of mnist dataset. Their main aim was to undo the effect of noise in mnist dataset so that images can be robust to small amount of noise. Seaz at el [9] analyzed if the predictive power and robustness of a classifier can be increased by implementing it in a Multiple Classifier Systems in presence of noise. First noisy datasets have been created by introducing attribute, class level, Gaussian as well as uniform noise. Then different MCS techniques were have been compared against the single classification algorithm respectively. Algorithms tested in this paper include C4.5, SVM, KNN with different k value settings. The results obtained have shown that the improvement by MCSs is highly dependent upon individual components classifiers and their ability to be less affected by noise. Authors have shown that BagC4.5 works well with both attribute as well as class label noise. C4.5 in bagging setup has proven to be more robust in noisy environment than its single classifier C4.5.

2.5 Handling class imbalance in software fault prediction

Under-sampling is a method to solve the class imbalance which removes the instances from the majority class of the dataset. Oversampling is a method to solve the class imbalance problem which adds instances to the minority class by either duplicating minority class instances or adding fake data in the dataset. Random under-sampling and oversampling techniques have been widely used [27], [22]. However, they are affected by data distribution and the algorithm considered for the classification [27]. Barandela *et al.* [26] showed that in highly imbalanced datasets, instead of using under sampling of majority class oversampling of the minority class should be performed for better results. Wang and Yao [27] presented a comparative study between five class imbalance learners

including, SMOTE Boost (which is a combination of the random oversampling technique plus Adaptive Boost , Threshold Moving (which applies a cost-sensitive method directly on the data, Random Under-sampling of both classes, Random Under-sampling of the majority [27].

2.6 Tabular representation

S.NO	Paper	Key points	Technique/Tool used	Contribution	Negative points
1	[10]	Introduced classification filter	-	Noise instance removal should be performed before hypothesis formulation so that formulated hypothesis is not affected	Only single classifier is used
2	[11]	Introduced iterative filter and multiple partitioning filter	Weka	Multiple classifiers with majority voting is better than using single classifer for multiple iteration	No balance between amount of noisy instances removed and retained for training
3	[20]	Used noise scoring technique to remove noisy instances	KEEL data-mining software tool	Incorporated best part of three noise filtering technique into one	Used accuracy as a performance measure
4	[5]	Used deep representation, Combined stacked denoising auto encoder and ensemble learning	MATLAB 2018a, Weka	Usage of deep representation extracted by stacked denoising auto encoder in software fault prediction domain	Ensemble of three strong learners were used in order to deal with class imbalance
5	[9]	Ensemble of classifiers used for handling noise in dataset	KEEL data-mining software tool	Wide range of experiments being performed for different levels as well as type of noise	No comparison with other existing techniques, Less focus on future scope for better denoising technique
6	[4]	Rough set theory to delete noisy instances, Easy ensemble technique for handling imbalance in dataset	MATLAB 2018a,Weka	Algorithm for removing noisy instance using rough set theory is proposed	Experiments are not well defined, Experiments not performed on different level of noise setting
7	[21]	Used stacked denoising auto encoder to intermediate representation of images	-	Intermediate representation can be much more suited for supervised learning	-
[8]	[26]	Comparative study between under sampling and over sampling	-	In case of severe imbalance over sampling technique should be applied	No experiment on multi class datasets
[9]	[22]	Introduced SMOTE		To overcome over fitting caused by oversampling, synthetic examples of minority class are added in place o exact replicas	Does not take into consideration neighbouring examples from other classes

Table 2.1 A comparative study of related work

2.7 Research gaps

In the field of software fault prediction a lot of denoising techniques have been applied ranging from iterative filter [11], classification filter [10] and ensemble-partitioning filter [13] but all of those work are based on the class label noise and only few experiments have been performed for removal attribute noise. Hanon tong et al [5] used deep representation learned from the entire dataset which may have noise instances in it. David et al [31] and Petric et al [30] questioned the quality of datasets of NASA MDP datasets and has suggested different cleaning techniques should be applied before using these datasets. Gray et al [29] found significant amount of erroneous data points in NASA MDP dataset which even contained impossible metrics values. So it can be said that the dataset from which they are learning might be corrupted.

2.8 Motivation and objective

As mentioned in above paragraph only a few studies have been performed for removal of attribute noise in software fault prediction domain. As introduced by Riaz et al[4] rough set theory can be used to perform denoising in dataset as it is a tool which can be used to remove vagueness in the dataset. Rough set theory can be used to extract significant data from the entire dataset. As given by Haonan Tong et al [5], deep representation of dataset can perform better than the traditional software metrics in most of the noise cases. So, Objective of this thesis is to provide machine learning model with dataset from which it can learn most and reflect the same during testing. So, to extract the noise robust dataset from any software fault prediction dataset an approach involving usage of rough set theory followed by oversampling and denoising auto encoder has been proposed.

2.8.1 Problem Description

Implementing denoising auto encoder as denoising technique with rough set theory and oversampling for pre-processing in order to handle noise in dataset which can further be used in software fault prediction.

Following objectives will be explored

- Usage of rough set theory for the extraction of clean or confident instances from the dataset
- Comparison of usage of deep representation extracted after proposed method with the traditional datasets
- Comparison proposed approach with existing techniques

CHAPTER 3

PROPOSED WORK

3.1 Methodology

In this report, handling the noise in dataset with denoising auto encoder has been proposed for software fault prediction. Figure 3.1 shows the framework of our proposed approach. This approach consists of two parts 1) Lower approximation based instance selection 2) denoising auto encoder based dataset generation which takes minority class oversampled lower approximation of datasets as input. Then it corrupts attributes values on purpose and reconstructs the correct values from the corrupted attribute value by optimizing the loss between correct dataset and corrupted dataset.

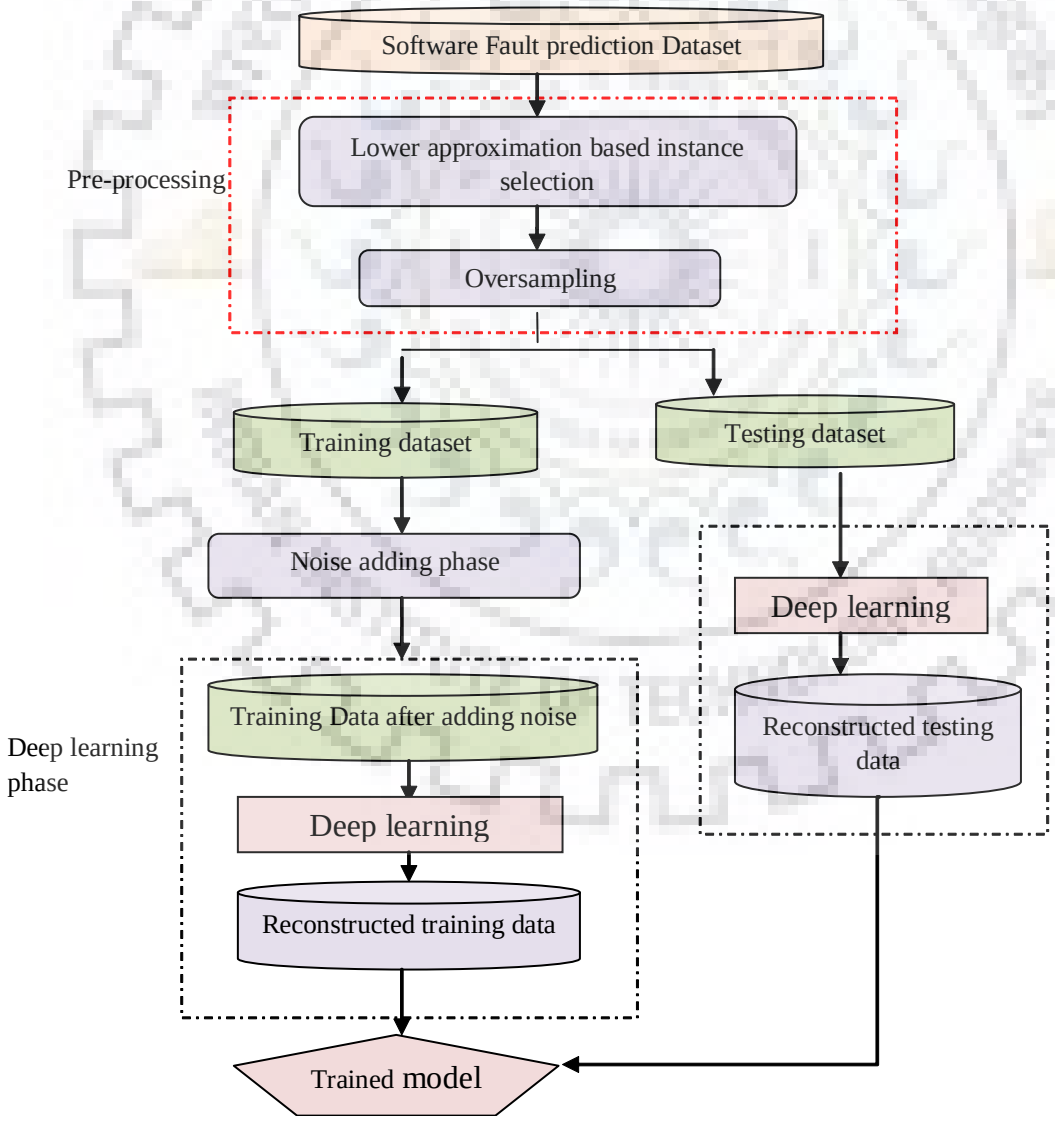

Figure 3.1: The framework of proposed approach for noise robust dataset generation

3.1.1 Lower approximation based instance selection

The main goal of the rough set theory is handling the uncertainty and overlapping dataset. Lower approximation is utilized to get the certain instances from the dataset. The lower approximation [19] of target set X is set of all observation which can be classified with certainty as member of X with respect to equivalence classes induced by R.

$$\underline{R}(X) = \{ x \mid [x_R] \subseteq X \} \tag{3.1}$$

were $[x_R]$ is equivalence class of x induced by R

On the basis of Rough set theory, two-ways decision can be made: immediate acceptance (lower approximation) and may acceptance (upper approximation). On the basis of these possibilities, we consider checking the possibility of correct and data free from noise in the lower approximation region. As all three of these datasets are highly imbalanced, we have used SMOTE [21] for oversampling as it out performances random oversampling. SMOTE [21] stands for Synthetic minority oversampling technique. It synthesizes new minority instances between existing minority instances. It synthesizes new minority instances with the help of K- nearest neighbour points of existing instances.

3.1.2 Denoising auto encoder based noise robust dataset generation

Denoising auto encoder is special type of auto encoder in which noise is deliberately added in order to corrupt attributes of datasets. And then by using encoding and decoding process of auto encoder, the robust or free from noise representation of dataset is generated. Suppose y is an output vector and $\bar{y}$ is the corrupted input to auto encoder, then the auto encoder learns the noise free representation of data by minimizing the reconstruction error between z and y which can be measured in many ways, depending on the distributional assumptions of the input.

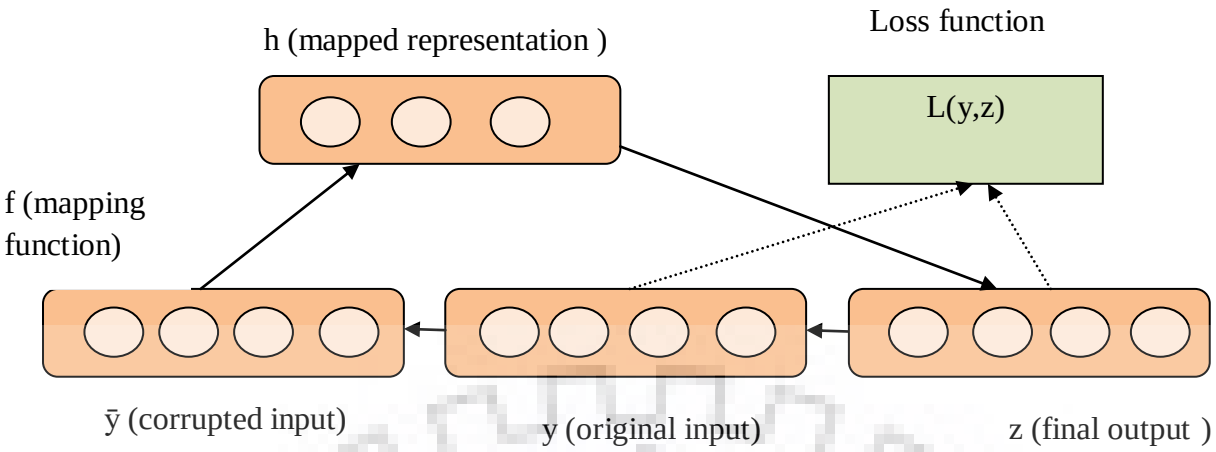

Figure 3.2: An example of denoising auto encoder, the original input y is corrupted to ȳ, ȳ is mapped to h and then representation z tries to reconstruct y. The reconstruction error is denoted by L(y, z).

Then to get the train dataset from denoising auto encoder, decoder is used to predict reconstructed training data from corrupted noisy data. In this experimentation for eclipse datasets, a 3 layered encoder and 2 layered decoder has been used. There are 50, 25, 20 neurons in the three encoding layers respectively. The decoder contains 25, 50 neurons in its two layers respectively.

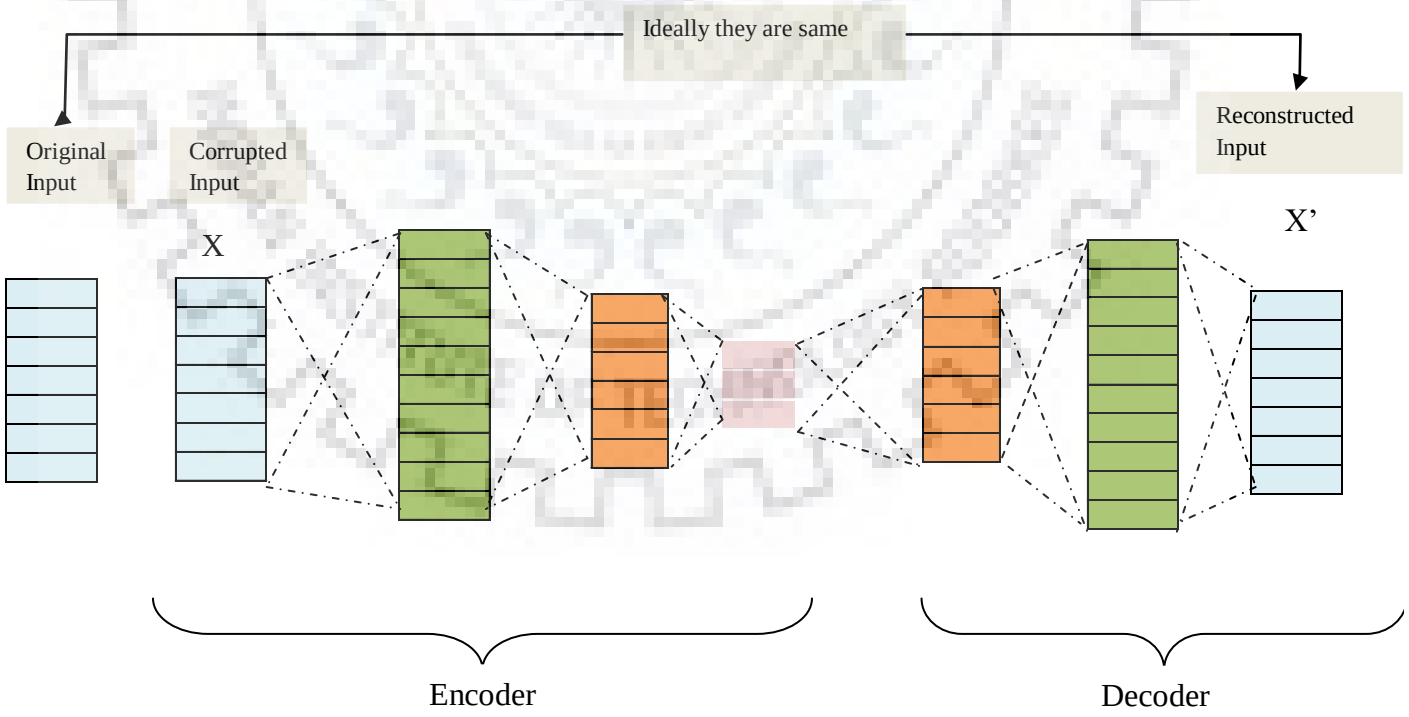

Figure 3.3 Representation of Denoising Auto encoder used

CHAPTER 4

EMPIRICAL STUDY OF EXISTING TECHNIQUES

4.1 Dataset

The software metrics and quality data used in this thesis are from publicly available ECLIPSE [7] program and NASA MDP [27]. All the datasets are obtained from the PROMISE software engineering repository [7]. All datasets are imbalanced. ECLIPSE 3 has the missing values. To obtain the noisy dataset and to study the effect of noise on different classifiers, different levels of attribute noise is being produced in each of these datasets. New datasets included 2%, 5%, 10%, and 20% of noisy instances in each of these datasets. Noise was introduced by adding guassian noise in selected row values in the range of that attribute. 5 fold cross validation is used and to get average values of performance evaluation metrics, each experiment has been performed 5 times.

S.NO	Dataset	%instances non faulty	% instances faulty
1	Eclipse 2.0 [7]	85.51	14.49
2	Eclipse 2.1 [7]	89.17	10.82
3	Eclipse 3 [7]	85.19	14.80
4	CM1 [27]	90.16	9.83
5	KC1 [27]	84.54	15.45
6	JM1 [27]	80.65	19.35

Table 4.1: Datasets used for experiments

4.2 Machine learning techniques used

Four machine learning techniques used are Naïve Bayes[25], logistic regression [5], decision tree[24], and k nearest neighbor[32]. Apart from that as proposed by Jose et al [9] bagging performs best in the case of attribute noise i.e. we have included bagged decision tree [24] classifier also for result comparison.

4.3 Performance evaluation measures used

Four different performance evaluation techniques has been used as discussed below:

Accuracy [14]: It denotes the percentage of correctly classified instances to the total number of instances.

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} * 100$$

Precision [14]: It denotes number of correctly classified faulty instances among the total number of instances classified as faulty.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Recall [15]: It denotes the number of correctly classified faulty instances amongst the total number of instances which are faulty.

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

F-measure [15]: It denotes harmonic mean of precision and recall values

$$F - measure = \frac{2 * \mathrm{precision} * \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Where TP denotes True Positive, FP denotes False Positive, TN denotes True Negative and FN denotes False Negative.

AUC (Area under the curve) [23]: It is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative').

4.4 Results and Discussion

Below are the comparative analysis of various classifiers along with the datasets and different amount of noise present in them. The datasets are partitioned into 5 equal parts called folds, each fold having almost equal number of faulty and non faulty instances. 4 out of 5 folds are used for training

and testing is done on the 5th fold. Each experiment is performed five times and average of performances evaluation metrics is taken. The performance of various classifiers with respect to different evaluation metrics are as follows:

	Accuracy					Precision				
Noise % / Classifier	0%	2%	5%	10%	20%	0%	2%	5%	10%	20%
Logistic regression	87.81	87.71	87.45	86.94	86.23	0.735	0.742	0.753	0.752	0.779
Naïve bayes	83.98	84.60	28.71	78.80	14.60	0.447	0.467	0.166	0.355	0.145
K nearest neighbor	88.57	88.07	88.07	88.23	87.86	0.690	0.662	0.667	0.681	0.649
Decision tree	84.32	83.79	84.14	84.01	83.47	0.460	0.444	0.453	0.449	0.432
Bagged decision tree	88.40	88.52	88.50	88.18	88.14	0.669	0.679	0.693	0.674	0.687

Table 4.2 Accuracy and precision for various machine learning techniques on Eclipse 2.0 dataset under different noise level

	Recall					F-measure				
Noise % / Classifier	0%	2%	5%	10%	20%	0%	2%	5%	10%	20%
Logistic regression	0.251	0.234	0.202	0.148	0.070	0.373	0.355	0.317	0.247	0.129
Naïve bayes	0.438	0.426	0.967	0.543	1.000	0.442	0.445	0.283	0.425	0.254
K nearest neighbor	0.385	0.361	0.358	0.354	0.354	0.494	0.467	0.465	0.466	0.457
Decision tree	0.460	0.458	0.451	0.446	0.437	0.460	0.450	0.452	0.447	0.434
Bagged decision tree	89.61	89.62	89.45	89.45	89.64	0.553	0.558	0.537	0.539	0.566

Table 4.3: Recall and F-measure for various machine learning techniques on Eclipse 2.0 dataset under different noise level

Accuracy						Precision				
Noise % / Classifier	0%	2%	5%	10%	20%	0%	2%	5%	10%	20%
Logistic regression	89.96	89.83	89.63	89.51	89.29	0.688	0.669	0.662	0.671	0.661
Naïve bayes	85.23	86.31	22.42	89.43	24.25	0.332	0.350	0.120	0.564	0.127
K nearest neighbor	89.47	89.41	89.38	89.36	89.30	0.534	0.527	0.524	0.524	0.516
Decision tree	85.21	84.78	84.82	84.99	85.62	0.312	0.300	0.298	0.305	0.321
Bagged decision tree	89.32	89.71	89.51	89.67	89.54	0.515	0.571	0.548	0.577	0.555

Table 4.4: Accuracy and precision for various machine learning techniques on Eclipse 2.1 dataset under different noise level

Recall						F-measure				
Noise % / Classifier	0%	2%	5%	10%	20%	0%	2%	5%	10%	20%
Logistic regression	0.134	0.120	0.088	0.063	0.023	0.224	0.203	0.155	0.116	0.044
Naïve bayes	0.359	0.295	0.945	0.100	0.935	0.344	0.317	0.212	0.170	0.221
K nearest neighbor	0.214	0.208	0.207	0.205	0.192	0.305	0.298	0.296	0.294	0.280
Decision tree	0.304	0.305	0.297	0.303	0.296	0.307	0.302	0.297	0.304	0.307
Bagged decision tree	0.212	0.198	0.185	0.177	0.182	0.306	0.291	0.274	0.265	0.275

Table 4.5: Recall and F-measure for various machine learning techniques on Eclipse 2.1 dataset under different noise level

Accuracy						Precision				
Noise % / Classifier	0%	2%	5%	10%	20%	0%	2%	5%	10%	20%
Logistic regression	87.21	87.11	86.38	86.00	85.68	0.743	0.749	0.718	0.714	0.787
Naïve bayes	83.37	84.37	85.45	67.94	73.26	0.428	0.457	0.536	0.407	0.573
K nearest neighbor	85.92	86.07	86.14	86.21	86.03	0.550	0.559	0.567	0.571	0.562
Decision tree	82.83	82.80	82.36	82.17	82.79	0.416	0.416	0.402	0.395	0.414
Bagged decision tree	87.21	87.12	87.15	86.90	87.03	0.628	0.624	0.630	0.621	0.632

Table 4.6: Accuracy and precision for various machine learning techniques on Eclipse 3 dataset under different noise level

Recall						F-measure				
Noise % / Classifier	0%	2%	5%	10%	20%	0%	2%	5%	10%	20%
Logistic regression	0.207	0.193	0.131	0.091	0.044	0.324	0.307	0.221	0.161	0.083
Naïve bayes	0.368	0.274	0.167	0.481	0.245	0.395	0.338	0.248	0.280	0.136
K nearest neighbor	0.266	0.278	0.269	0.275	0.255	0.358	0.371	0.364	0.370	0.350
Decision tree	0.399	0.402	0.393	0.387	0.387	0.407	0.409	0.397	0.391	0.400
Bagged decision tree	0.332	0.325	0.319	0.295	0.296	0.434	0.427	0.423	0.400	0.403

Table 4.7: Recall and F-measure for various machine learning techniques on Eclipse 3 dataset

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

1. Results suggests that the accuracy for the machine learning techniques used are high (>0.85 and generally >0.7)

2. As all three datasets are highly imbalanced in this case accuracy measure of classifier is misleading as it is getting over fit with majority class.
3. A more stable performance measure such as f1 score or AUC should be used for the classifier used in the case of imbalanced datasets.
4. Amount of noisy instances present in dataset affects classifiers differently.
5. Amount of noise added degrades performance of classifier.
6. Highest effect of noise is observed in naïve bayes
7. Generally, the machine learning procedures have low F-measure
8. For all the combinations of datasets, the f1 score of Bagged decision tree has turned out to be most stable in presence of noise
9. Other performance measures are also degraded by presence of noise.

CHAPTER 5

EXPERIMENTS AND RESULTS

5.1 Experiments and results on Intra release datasets

In the proposed approach datasets listed in chapter 4 were used to get noise free representation produced by denoising auto encoder after taking lower approximation and oversampling minority class of each datasets .The datasets is partitioned into 5 equal parts called folds, each fold having almost equal number of faulty and non faulty instances. 4 out of 5 olds are used for training and testing is done on the 5^{th} fold. Each experiment is performed five times and average of performances evaluation metrics is taken. The following are the values of different performance measures observed for different datasets.

Accuracy						*Precision*				
Noise % / *Classifier*	*0%*	*2%*	*5%*	*10%*	*20%*	*0%*	*2%*	*5%*	*10%*	*20%*
Logistic regression	73.79	73.89	74.21	73.68	73.73	0.736	0.738	0.737	0.732	0.728
Naïve bayes	72.19	73.67	72.26	73.20	70.11	0.793	0.747	0.796	0.749	0.815
K nearest neighbor	82.49	81.95	82.04	82.15	81.34	0.780	0.776	0.776	0.777	0.774
Decision tree	74.24	74.11	74.29	74.67	73.74	0.751	0.738	0.786	0.774	0.776
Bagged decision tree	84.79	85.14	84.53	84.64	83.61	0.842	0.846	0.849	0.844	0.824

Table 5.1 Accuracy and precision for various machine learning techniques with denoised dataset on Eclipse 2.0 dataset under different noise level

Recall						*F-measure*				
Noise % / *Classifier*	*0%*	*2%*	*5%*	*10%*	*20%*	*0%*	*2%*	*5%*	*10%*	*20%*
Logistic regression	0.741	0.742	0.754	0.747	0.760	0.739	0.740	0.745	0.739	0.743
Naïve bayes	0.601	0.717	0.600	0.701	0.521	0.683	0.731	0.683	0.723	0.635

K nearest neighbor	0.905	0.900	0.901	0.901	0.886	0.838	0.833	0.834	0.835	0.826
Decision tree	0.737	0.763	0.678	0.711	0.687	0.739	0.745	0.722	0.735	0.721
Bagged decision tree	84.79	85.14	84.53	84.64	83.61	0.842	0.846	0.849	0.844	0.824

Table 5.2 Recall and F-measure for various machine learning techniques with denoised dataset on Eclipse 2.0 dataset under different noise level

Accuracy						*Precision*				
Noise % / *Classifier*	*0%*	*2%*	*5%*	*10%*	*20%*	*0%*	*2%*	*5%*	*10%*	*20%*
Logistic regression	72.40	72.57	72.48	72.03	71.98	0.761	0.761	0.758	0.751	0.746
Naïve bayes	68.36	68.26	67.25	67.10	64.33	0.803	0.798	0.818	0.815	0.825
K nearest neighbor	81.53	81.79	81.56	80.89	80.37	0.762	0.765	0.763	0.759	0.755
Decision tree	74.05	74.03	74.22	74.25	74.24	0.767	0.761	0.766	0.760	0.761
Bagged decision tree	83.81	83.78	83.16	83.18	82.47	0.825	0.829	0.820	0.820	0.814

Table 5.3 Accuracy and precision for various machine learning techniques with denoised dataset on Eclipse 2.1 dataset under different noise level

Recall						*F-measure*				
Noise % / *Classifier*	*0%*	*2%*	*5%*	*10%*	*20%*	*0%*	*2%*	*5%*	*10%*	*20%*
Logistic regression	0.653	0.658	0.660	0.659	0.666	0.703	0.706	0.706	0.702	0.704
Naïve bayes	0.486	0.489	0.444	0.443	0.364	0.606	0.606	0.575	0.574	0.505

K nearest neighbor	0.916	0.919	0.916	0.905	0.899	0.832	0.835	0.832	0.826	0.821
Decision tree	0.691	0.702	0.702	0.712	0.708	0.727	0.730	0.731	0.734	0.733
Bagged decision tree	0.858	0.852	0.849	0.851	0.842	0.841	0.840	0.835	0.835	0.828

Table 5.4 Recall and F-measure for various machine learning techniques with denoised dataset on Eclipse 2.1 dataset under different noise level

Accuracy						*Precision*				
Noise % / *Classifier*	*0%*	*2%*	*5%*	*10%*	*20%*	*0%*	*2%*	*5%*	*10%*	*20%*
Logistic regression	74.40	74.27	74.07	73.71	72.62	0.754	0.753	0.750	0.743	0.722
Naïve bayes	68.49	68.66	68.46	68.75	68.85	0.774	0.776	0.773	0.779	0.769
K nearest neighbor	80.20	80.36	80.23	79.66	78.80	0.756	0.758	0.756	0.752	0.745
Decision tree-	70.76	71.01	70.99	71.20	70.30	0.814	0.798	0.797	0.810	0.811
Bagged decision tree-	82.91	82.66	82.74	82.30	81.86	0.830	0.823	0.826	0.823	0.817

Table 5.5 Accuracy and precision for various machine learning techniques with denoised dataset on Eclipse 3.0 dataset under different noise level

Recall						*F-measure*				
Noise % / *Classifier*	*0%*	*2%*	*5%*	*10%*	*20%*	*0%*	*2%*	*5%*	*10%*	*20%*
Logistic regression	0.724	0.722	0.722	0.724	0.736	0.739	0.737	0.736	0.734	0.729
Naïve bayes	0.523	0.525	0.522	0.523	0.539	0.624	0.626	0.623	0.626	0.634

K nearest neighbor	0.892	0.893	0.894	0.886	0.875	0.818	0.820	0.819	0.813	0.805
Decision tree	0.550	0.570	0.572	0.561	0.534	0.650	0.661	0.661	0.659	0.642
Bagged decision tree	0.827	0.832	0.829	0.824	0.821	0.829	0.828	0.828	0.823	0.819

Table 5.6 Recall and F-measure for various machine learning techniques with denoised dataset on Eclipse 3.0 dataset under different noise level

5.2 Experiments and results on inter release datasets

The earlier release of a dataset is used for training purpose to predict the fault proneness for the later release that is used as testing dataset. There are 3 pairs of training-testing datasets in our experiments. Table 5.7 provides details of the used datasets.

Serial number	Training dataset	Testing dataset
1	Eclipse 2.0	Eclipse 2.1
2	Eclipse 2.0	Eclipse 3.0
3	Eclipse 2.1	Eclipse 3.0

Table 5.7 Datasets used for inter release experiments

The following are the values of different performance measures observed for different datasets.

Accuracy			
Noise % / Classifier	0%	10%	20%
Logistic regression	88.64	87.20	86.80
Naïve bayes	83.92	76.97	70.34
K nearest neighbor	83.96	83.36	82.51
Decision tree	86.92	86.76	86.96

Table 5.8 Accuracy for various machine learning techniques with denoised dataset on Eclipse 2.0 dataset and tested on Eclipse 2.1 under different noise level

Accuracy			
Noise % / Classifier	0%	10%	20%
Logistic regression	86.92	84.42	83.06
Naïve bayes	82.55	76.21	73.61
K nearest neighbor	82.82	82.21	81.28
Decision tree	85.80	85.49	84.92

Table 5.9 Accuracy for various machine learning techniques with denoised dataset trained on Eclipse 2.0 and tested on Eclipse 3 dataset under different noise level

Accuracy			
Noise % / Classifier	0%	10%	20%
Logistic regression	86.09	84.37	83.45
Naïve bayes	83.34	82.08	72.65
K nearest neighbor	82.20	81.30	81.25
Decision tree	85.54	85.05	85.28

Table 5.10 Accuracy and precision for various machine learning techniques with denoised dataset trained on Eclipse 2.1 and tested on Eclipse 3 dataset under different noise level

5.3 Intermediate results for various datasets

This section includes the step by step results of techniques used in this work. Original dataset corresponds to traditional software fault prediction dataset. Dataset after lower approximation columns corresponds to results obtain after performing lower approximation on original dataset followed by column for Dataset obtained by applying denoising auto encoder on dataset obtained after lower approximation and SMOTE[22] .

Serial number	Dataset name	Original dataset	Dataset after lower approximation	Dataset after Proposed approach
1	CM1	0.5319	0.5089	**0.7884**
2	JM1	0.5514	0.5416	**0.8314**
3	KC1	0.6003	0.6257	**0.7457**

Table 5.11 AUC when logistic regression is applied with different datasets constructed during proposed approach

5.4 Experiments and results on existing techniques

Noise	Bagged Decision tree	Proposed approach
Eclipse 2.0	0.682	**0.851**
Eclipse 2.1	0.596	**0.838**
Eclipse 3.0	0.634	**0.829**

Table 5.12 Comparison of AUC of proposed approach with bagged decision tree on eclipse datasets with 20% of noise

Noise	Stacked denoising auto encoder with ensemble learning[5]	Proposed approach with logistic regression
Cm1	0.8373 [5]	**0.7884**

Jm1	0.7731 [5]	**0.8134**
Kc1	0.7426 [5]	**0.7457**

Table 5.13 Comparison of AUC of proposed approach with Software defect prediction using stacked denoising auto encoders and two-stage ensemble learning.

5.5 Results analysis

From the results obtained in above section it is observed that performance of every classifier when applied with traditional software dataset has increased with a huge margin when the classifier used the dataset generated by proposed approach. With the traditional dataset the AUC score was in the range (0.13-0.57) and with the dataset generate by proposed approach the AUC score elevated to range (0.6-0.91). Measures such as recall and f1-score were also increased for every classifier denoting the stable classification. It can also be observed that when in the presence of noise, classifiers such as naïve bayes and k nearest neighbour were highly affected even with the slightest (2%-10%) noise , whereas when the proposed approach is used for dataset generation even with the noise level (10%-20%) performance of classifiers were not affected. If we compare the proposed approach with approach used in [5], proposed approach has outperformed it for all datasets. When compared with other approaches such as bagged decision tree used for handling class imbalance and attribute noise, proposed approach has outperformed it for all the three Eclipse datasets [7].

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we have applied decision tree, k nearest neighbour, Logistic classifier and Naïve bayes, bagged decision for the purpose of software fault prediction. We have also applied different amount of attribute noise to study its effect on classifier's performance. The study was performed on the Eclipse dataset available in PROMISE software engineering repository [7] and NASA MDP datasets [28]. By taking lower approximation we eliminated the presence of uncertain instances from both classes i.e. faulty and non faulty of the dataset. So that auto encoder can learn from a representation which is free from outliers. The denoising auto encoder was applied to extract the attribute noise robust dataset. From the experimental analysis; it was observed that all of the studied learning models have benefitted from using dataset reconstructed by the use of lower approximation, SMOTE and denoising auto encoder. Performance of classifiers is not degraded by the presence of noise up to 20% in them by using dataset reconstructed by proposed approach. And further, performance of classifiers when using noise robust dataset remains the same for most of the combination of noise. Weak learners such as naive bayes and k nearest neighbor improved significantly when used with reconstructed datasets. Noise robust dataset have performed best under all the noise conditions as compared to when the classifiers use raw metrics. To state the validity of our approach, we compared our approach with existing techniques used in software fault prediction domain and got better results in all of the datasets.The future work can include extending this work to more datasets on which this work can be replicated to check the consistency of the results. Apart from five machine learning technique used in this work, some other advanced machine learning techniques can be used to see the effect of noise of these techniques. Some other advanced variations of SMOTE are also present in literature which can be used to increase the predictive power of proposed approach. Further, this work can also extend to incorporate the multiclass prediction problem in software fault prediction.

REFERENCES

[1] "Noise reduction," *Wikipedia*, 20-Feb-2019. [Online]. Available: https://en.wikipedia.org/wiki/Noise_reduction. [Accessed: 19-May-2019].

[2] "Soft Computing and Intelligent Information Systems," *Noisy Data in Data Mining | Soft Computing and Intelligent Information Systems*. [Online]. Available: https://sci2s.ugr.es/noisydata. [Accessed: 19-May-2019].

[3] "The Keras Blog, " *The Keras Blog ATOM*. [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html. [Accessed: 19-May-2019].

[4] S. Riaz, A. Arshad and L. Jiao, "Rough Noise-Filtered Easy Ensemble for Software Fault Prediction," in *IEEE Access*, vol. 6, pp. 46886-46899, 2018.

[5] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," Information and Software Technology, vol. 96, pp. 94–111, Apr. 2018.

[6] J. A. Sáez, J. Luengo, and F. Herrera, "Evaluating the classifier behavior with noisy data considering performance and robustness: The Equalized Loss of Accuracy measure," Neurocomputing, vol. 176, pp. 26–35, Feb. 2016.

[7] "Software Defect Prediction Data," *SEIP Lab*. [Online]. Available: http://www.seiplab.riteh.uniri.hr/?page_id=834&lang=en. [Accessed: 20-May-2019].

[8] X. Zhu and X. Wu, "Class Noise vs. Attribute Noise: A Quantitative Study," Artificial Intelligence Review, vol. 22, no. 3, pp. 177–210, Nov. 2004.

[9] J. A. Sáez, M. Galar, J. Luengo, and F. Herrera, "Tackling the problem of classification with noisy data using Multiple Classifier Systems: Analysis of the performance and robustness," Information Sciences, vol. 247, pp. 1–20, Oct. 2013.

[10] D. Gamberger, N. Lavrac, C. Groselj, "Experiments with noise filtering in a medical domain" *In Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99),*. San Francisco (USA, 1999), Ivan Bratko and Saso Dzeroski (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 143-151

[11] T. M. Khoshgoftaar and P. Rebours, "Noise elimination with partitioning filter for software quality estimation," International Journal of Computer Applications in Technology, vol. 27, no. 4, p. 246, 2006.

[12] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction, " in *Proceedings of the International Conference on Software Engineering*, 2013, pp. 392–401.

[13] T. M. Khoshgoftaar and P. Rebours, "Generating multiple noise elimination filters with the ensemble-partitioning filter," *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, Las Vegas, NV, 2004, pp. 369-375.

[14] "ISO 5725-1:1994," *ISO*, 24-Jul-2018. [Online]. Available: https://www.iso.org/standard/11833.html. [Accessed: 19-May-2019].

[15] D. L. Olson and D. Delen, "Data Mining Process, " *Advanced Data Mining Techniques*, pp. 9–35.

[16] C.Catal and B.Diri, "A fault detection strategy for software projects, " *Technical Journal*,2013, pp. 1-7.

[17] G. Abaei and A. Selamat, "A survey on software fault detection based on different prediction approaches," Vietnam Journal of Computer Science, vol. 1, no. 2, pp. 79–95, Nov. 2013.

[18] C.Catal, "Performance Evaluation Metrics for Software Fault Prediction Studies, " *Acta Polytechnica Hungarica*,2012,vol. 9,pp 9-22.

[19] Q. Zhang, Q. Xie, and G. Wang, "A survey on rough set theory and its applications," CAAI Transactions on Intelligence Technology, vol. 1, no. 4, pp. 323–333, Oct. 2016.

[20] J. A. Sáez, M. Galar, J. Luengo, and F. Herrera, "INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control," Information Fusion, vol. 27, pp. 19–32, Jan. 2016.

[21] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in Proceedings of the 25th international conference on Machine learning - ICML '08, 2008.

[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, Jun. 2002.

[23] T. Fawcett, "An introduction to ROC analysis," Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, Jun. 2006.

[24] J. R. Quinlan, "Induction of decision trees," Machine Learning, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[25] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press ,2000.

[26] R. Barandela, R. M. Valdovinos, J. S. Sánchez, F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?, " *In Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition*

(SSPR), Berlin: Springer,2004. pp. 806-814

[27] S. Wang and X. Yao, "Multiclass Imbalance Problems: Analysis and Potential Solutions," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 4, pp. 1119-1130, Aug. 2012.

[28] *PROMISE DATASETS PAGE*. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html. [Accessed: 19-May-2019].

[29] M. Shepperd, Q. Song, Z. Sun and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," in *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208-1215, Sept. 2013.

[30] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "The jinx on the NASA software defect data sets," in Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16, 2016

[31] D. Gray, D. Bowes, N. Davey, Y. Sun and B. Christianson, "The misuse of the NASA metrics data program data sets for automated software defect prediction," *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, Durham, 2011, pp. 96-103.