IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 22, Issue 5, Ser. III (Sep. – Oct. 2020), PP 48-58
www.iosrjournals.org

Performance comparison of Machine learning classifiers in Software Defects Prediction

Naeem Hussain[1], Li Bixin[2], YuanXiaodong[3], IsrarHussain[4]
[1](School of Computer Science and Engineering, Southeast University, Nanjing China)
[2](School of Computer Science and Engineering, Southeast University, Nanjing China)
[3](School of Computer Science and Engineering, Southeast University, Nanjing China)
[4](College of Electronic Information Engineering, Shenzhen University, Shenzhen China)

Abstract:
Background: *In software development life cycle, software testing is the main stage which can minimize the defects of software. A domain which has receiving much attention of software researchers since past couple of years is software defects prediction (SDP). Its aim to minimize the cost, time and improve the efficiency of software. The main aim of this research is to show a comparative analysis of software defect prediction based on support vector machine SVM and extreme learning machine ELM. In this domain defect prediction models were created using three different prediction techniques based on test data and training data. i.e. cross-validation prediction, cross-version prediction and cross-project prediction. In this study we used cross version prediction approach, data from old version of a software is used as training data to develop the prediction model and the model is evaluated from same project of current version.*
Materials and Methods: *In our studies, we consider three different versions of eclipse version control system then we had split the data into training and tested sets. We choose different object oriented metrics and algorithm to build our model, aiming to predict software defects in different versions. For training purpose of our model we used SVM and ELM. To validate our prediction models, we can calculate the performance of prediction model using some popular used measurement scales such as accuracy, precision, recall, AUC (Area under ROC curve).*
Results: *By comparing the file based results of SVM and ELM we can find the average accuracy values and AUC. This means the extreme learning machine has the highest AUC value, but the value of accuracy is also close to SVM. And SVM have similar accuracy, and very close AUC value. Then we can see how these models perform in package based prediction. By comparing the data in package based prediction of SVM and ELM, the accuracy and AUC values shows thatSVM has best accuracy, but the value of AUC decreases apparently. So we can conclude that SVM has best prediction results in file based defects.The results demonstrate that support vector machine is best fit for the cross version defect prediction.*
Conclusion: *Software testing has become more and more important in software reliability since last couple of years. But on software testing we are wasting much time, resource and money. Software defect prediction can help to improve the efficiency of software testing and guide the direct resource allocation. In this study, we discussed the key techniques including software metrics, classifiers, and defect prediction models and its evaluation.Python language is most widely use language especially in data science. The significant factor giving the push for Python is the variety of data science/data analytics libraries made available for the aspirants. Pandas, NumPy, SciPy, and Scikit-Learn, are some of the libraries well known in the data science community. Python does not stop with that as libraries have been growing over time. When it comes to data science, machine learning is one of the significant elements used to maximize value from data.*
Key Words: *Software defects prediction, machine learning, software metrics, software quality prediction.*

Date of Submission: 06-11-2020 Date of Acceptance: 19-11-2020

I. Introduction

Defects are essential parameters of a software system. They come from software development life cycle. Defects in software can create unintended behaviors after the software project is implemented, thus causing tremendous economic loss to business in worst case. Software defects are very painful problem for developers, researchers as well as for end-users. The software project / product which is performing well at the moment may also have defects that are not caused now or that are not critical at this stage. Software defects are programming error that caused different actions compared to different expectations. Most of the defects are from the software development life cycle. Therefore, softwarecompanies used to use a resource called software quality assurance (SQA) team the job of this team is to test and inspect the code to detect defects before

deployment phase. At this stage, we are attempting to identify the software defects to achieve its necessary standards. In the whole software development life cycle more than 50 percent of time is spending on maintaining software quality and its reliability. Findings defects in software is not an easy task. Cost plenty of money to find and resolve the defects. To save the cost, time and testing resource companies want to identity potential defective modules as before deployment. Software defect prediction is one of the best approaches [1], [2],[3]. Researchers used to build software defect prediction models using machine learning techniques to identify defective modules in new projects. In addition, predicting number of defects for program, modules can assist in sorting program modules and then allocate more testing resources to these modules, which may contain more defects. In this way, allocation of testing resources can be further optimized.

In previous studies, there is no specific machine learning techniques which we will use to predict software defects prediction in file level and package level.To identify the best prediction techniques, we first sort out the best techniques from previous studies which machine learning techniques are widely used for defect prediction then we conduct comparison study of two different classifiers to identify the defects in file based and in package based.

In this study, we used the dataset which comes from the development of the Eclipse program repository, version control system (VCS) and bug tracking for version 2.0, 2.1 and 3.0.And there are 9 versions in total, use different object-oriented metrics to build our model and for training purpose of model, we use two classifiers SVM and ELM. Prediction models are designed with training data, and are tested on test data for its accuracy. Models of defect prediction are developed in the literature using three predictive techniques, which differ based on the training data and test data. The prediction techniques are(cross validation prediction, cross version prediction and cross project prediction). In this study, we used cross version prediction. In this approach, data from old version of a software is used as training data to develop the prediction model and model is evaluated from same project of current versions.

II. Material and Methods

Software defect prediction, in short SDP is popular technique based on machine learning models. Most research on identification of software defects have used machine learning techniques[4], [5], [6], [7], [8]. The very first stage to develop a prediction model is to create instances from software records, such as version control systems in short VCS, defect tracking systems, email records, etc. Instance can describe a process a software component (or package), a source code file, a class, a function (or method), and/or a change of code as per the specificity of predicting the defects. An instances has many metrics or attributes extracted from the repositories of the software and is marked with full of defects / clean or number of defects. For example, instances created from software repositories are marked with defect or clean or defect percentage [9].

After creating instances with metrics and labels which are popular in machine learning, we can introduce preprocessing steps. Preprocessing techniques used in prediction of defects involve character identification, data duplication and low noise[5], [10],[11]. We can train a prediction model with the final set of training instances. The prediction model will assess whether there is a defect in new sample or not.

Software metrics

Software metrics is a standard of measure that contains many activities which involves some degree of measurement. In term of software defect prediction, machine learning techniques have known to be helpful.The main purpose of using software metrics, by constructing defect prediction models using metrics selection techniques, software practitioners aim to focus on improving software quality.

To predict a software system defectiveness, we have to be able to calculate it. A software metric can be taken as a function of a set of software certain property that can be used to predict defects. There are numerous different types of metrics studied so far and we explain these metrics in detail below.

Static code metrics

Static code metrics are determined directly from source code and provide a high-level understanding and scale of the source code. For example, the number of branches or Boolean statements or the size of the source code itself is important while calculating these type of metrics. Menzies et al.[12]are classifying static code metrics as line of code (LOC) metrics, McCabe metrics, Halstead metrics[13]. (See fig 2.1)

Compared to other type of metrics, static code metrics are easy to understand and help us to see how much complex a software system is. For the purpose of defect prediction, these metrics are just inputs to the defect prediction algorithms.

Figure 2.1. McCabe, LOC, and Halstead metrics studied by Menzies et al. (2007)

McCabe	Locs	Halstead
v(G) Cyclomatic complexity, iv(G) Design complexity, ev(G) Essential complexity	LOC_T, LOC_CODE_COMMENT, LOC _COMMENT, LOC_EXE, NO_OF_LINES	N_OPERATORS, N_OPERANDS, N_UNIQUE_OPERATORS, N_UNIQUE_OPERANDS, LEN, VOL, LEVEL, EFFORT, DIFFICULTY

Object-oriented metrics

As the size and complexity of software grow and develop normally, developers come up with object oriented programming tools in 1990s.With the creation of OO design methodology, CK suggested new metrics[14],to access the quality of software systems that are built using OO programming languages. Similar to the traditional metrics explained above, these metrics model both inner module complexity and the interactions among the classes of the system and are extensively used in the defect prediction literature[15],These metrics are:

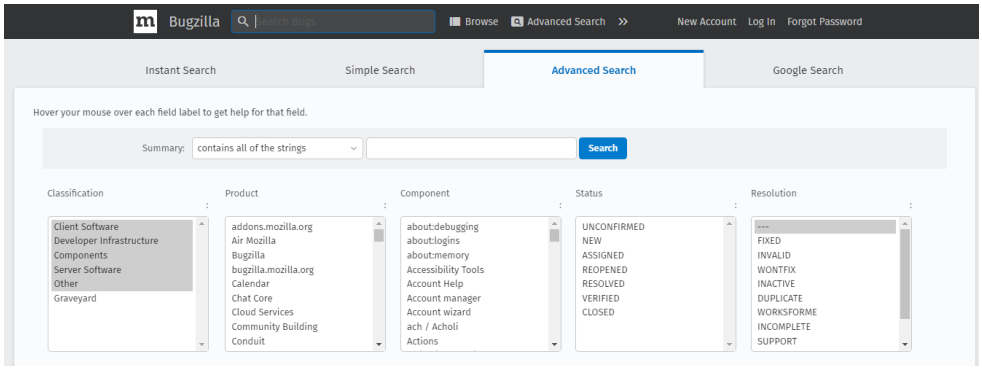
- WMC: The number of methods and operators defined in each class excluding inherited ones.
- DIT: Number of class ancestors.
- RFC: The number of functions (or methods) executed when a method or operator of that class is invoked. It is expected that when the value of RFC metric is higher, the class is more complex.
- NOC:Total of survivors of the main class, except categories of grandchildren.
- CBO: The number of combined groups in a software system. If a class is using operators or methods of that class, it is combined with another class.
- LCOM: The amount of procedures in a class using a common variable with other classes except the number of methods using variables which are not common with any other class. If LCOM is negative the setting is zero.
- LOC:The number of usable source code lines within a class. Remember that remarks are removed and blank lines.

Methodology for software defect prediction

In this section, we briefly explained the methods and techniques considered in our study for software defect prediction.

Bug tracking system

The data used in this study regarding bugs are the output of BUGZILLA, an open source project. Bugzilla is developed by the bug tracking system of Mozilla, which includes updating defect details, monitoring defects, finding defects and debating answers for developers and ultimately solving the problems. Bugzilla monitors eclipse defects discovered during development phase. The data of defects included in this research come from components JDT, PDE and platform, when Eclipse version 2.0, 2.1, 3.0. It has 4920 defects.



Bugs search system of bugzilla

Dataset obtain rules and algorithm

In this section, we want to acquire data sources from version control system and also from bug tracking system for defects in this phase. The data sources are acquires in three stages.

i. Collecting the change log details from certain version of software requires collecting all the defect IDs throughout construction that relate to a certain version of the software.
ii. We can find the version detail based on acquired defect ID similar location in version control system. Reporting any defect contains version of this field knowledge, involves version in which the defect was

identified. But in the life cycle of software, these specifics will change, we can get more than one version of this information in the final.

iii. Calculate software metrics for packages and files which exist defects. We are using the Eclipse plugin metrics 1.3.6. for this operation.

Brain Henderson-Sellers is proposing the set of metrics chosen in this study. It is the set of OO-metrics, including complexity metrics. For this study, we are using fourteen separate software metrics which are shown in below table.

Metrics set

Name	Detail	Name	Detail
fout	Functions_called	nom	Number_of_Methods
mloc	Method_line_of_code	nsm	Number_of_Static_Methods
nbd	Nested_Block_Dept	noc	Number_of_contained_classes
par	Number_of_Parameters	tloc	Total_line_of_codes
vg	McCabe Cyclomatic Complexity	noi	Number_of_interface
nof	Number_of_Attributes	nocu	Number_of_files

Dataset analysis and preprocessing

In this section, we need to examine the characters of metrics before the experiment of data sets. Empirical Software Engineering specifies that data used during predicting software defects typically has a certain amount of expertise. We need pre-processing to monitor the side effects of the quality of the experience.

Analysis dataset and defects distribution

The dataset can be broken down into two categories, dataset dependent on files and dataset based on packages. Each package has similar data files. Each file includes the defects of pre-release amount and defects of post release amount and also the value of metrics. In this research, the set of metrics used includes software complexity metrics, software size metrics and OO metrics as well.

There are total of 6729 file-based records in version 2.0, and package based contains 377 records. The below table contains lists the defects information in all three versions. Where if the simple size is between 10-100, so the validity of the findings will therefore be severely limited, this time usually on a small sample. The chosen classifier worked for better or using the bionic algorithm to provide large scale data. The sample size as used here, if the file level and amount of sample in package level are between 6000 and 11000 but the experimental findings have validity from the data point of view of the scale.

Dataset size

Release_Version	Files No.	Col_Nr	Failure_Prone	Package No.	Col_Nr	Failure_Prone
2.0	6729	202	.145	377	211	.504
2.1	7888	202	.108	434	211	.447
3.0	10593	202	.148	661	211	.473

Data preprocessing

Data preprocessing in machine learning is a key step that helps to improve data quality in order to facilitate the retrieval of useful insights from the data. In ML data preprocessing refers to the techniques of cleaning the raw data to make it feasible for models of constructing and machine learning model. The software defect prediction research reveals, that some defect system output is closely linked to data features. Heavily processed data sets often exhibit non-normal distribution, similarity and high redundancy and also unequal distribution of sample properties.

Using all these data greatly impacts assembly precision performance. Effective data pre-processing will also increase the efficiency and accuracy of the scale of data classification. Preprocessing of data consists in the following ways: *data cleaning, normalization, noise reduction, attribute selection.*

Defects prediction methods and steps

In this study we can discuss four aspects of defects classification prediction proposed which are collect the dataset, analyse and pre-process the dataset, develop prediction model by using different machine learning classifiers and evaluate the performance of the model. The steps are under below:

i. Obtain the dataset

ii. Split the dataset into two parts (file based and package based)
iii. The report obtains statistical records for file level data involving pre-release, amount of post release defects, file name and assembly that comprise the file.
iv. The report obtains statistical records for package level data involving pre-release, amount of post release defects, package name and assembly that comprise the package.
v. Use Eclipse metrics plugin to measure the software metrics per each file and package level.
vi. Data clean for both file and package based level.
vii. A clean file based data and package based data, create model to predict the defects in files / package of future version.
viii. Bugzilla handles the defects, so we can get details about the defects.
ix. At the last the predicted results and performance evaluation of the model.
x. Compare the performance of the models.

Prediction of support vector machine

The SVM develop in the linear classification. The low-dimensional issues which cannot be solved can be transformed into higher dimension and then linearly divisible. Its kernel function can map from small to large dimensions. If there is one sample fully divided by a linear function, we can then say this data is linear separable, otherwise it cannot be differentiated linearly. And space dimension, this linear function is a three-dimensional point space, his linearly function is s line. Where the dimension is infinite then the linear function is hyper plane.

At the beginning we define hyper plane as shown is figure 3.1.

Hyperplane: $g(x)=\{x|<w,x>+b=0\}, \langle w,x\rangle$ is the vector product, ω is normal vector, and b means offset. We get ω and b by classes which have labels. If the data is located in the position direction of this hyper plane, we regard it as a positive, otherwise it is negative. The data which are close to hyper plane are supported vectors. They define the position of hyper plane. Typically the problem of classification gets easier when we map the medium dimensional space to space of large dimensions. In this given situation, we need to construct the optimal hyper plane. The basic input model of support vector machine is $\{x[i]\} \in R_n$. It contains two types of points. If $x[i]$ belongs to class 1, then the $y[i]=1$, if $x[i]$ belongs to class two, then $y[i]=-1$. For the training data set $\{x[i], y[i], i=1,2,3,...,n\}$. The distance from sample data points to certain hyper plane is $\delta_i = y_i(\omega x_i + b)$, an optimal hyper plane means maximal intervals. The selection of hyper plane is depend on $\langle w,x\rangle$, we must determine the vector objective, and if the computing dimensions are very large complexity is high too. Kernel selection function is a solution to this problem. A healthy kernel output should be the same even if it in lower space measurements. i.e. we have expression profile $p=(\rho_1,\rho_{2,...},\rho_g) \in \square^g$ and $q=(q_1,q_{2,...},q_g) \in \square^g$.We can map into space of great dimensions, by linear kernel function $K(p,q)=\langle p,q\rangle$, polynomial kernel function $K(p,q)=(\gamma\langle p,q\rangle + C_0)^d$ and the kernel function based on radial $K(p,q)=\exp(-\gamma \square p-q \square^2)$.

There are plenty of instructions for choosing kernel functions until now. Normally the radial-based kernel functions are commonly used.

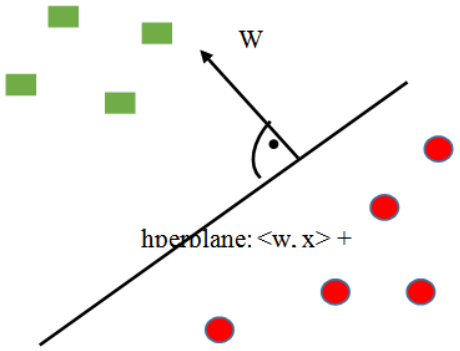


Figure 3.1 Hyper plane

Table 3.2 c-SVM model performance

Nr of support vector	1987
SVM type	C-classification
SVM-kernel	Radial
Cost	1
gamma	0.03125

Table 3.3 nu-SVM model performance

Nr of support vector	1274
SVM type	nu-classification
SVM-kernel	Radial
nu	0.01
gamma	0.03125

Apart from kernel functions, another big issue is when using kernel function to map high dimensional space, the problem is still what we should do in this case, not linearly differenced. But the presence of noise, we use slacks variables to solve the issue, the noise could be the data which can be separated. Data on sample requirements can be represented in exp 3.1 and n is the sample data amount.

$$y[i]\left(w \bullet x[i]+b\right) \geq 1\left(i=1,2,...n\right) \qquad 3.1$$

Which is the distance from the data to the hyper plane should be 1 max. The requirements can be redefined in exp 3.2 if we permit noise to happen, we can attach a slack variable to 1.

$$y[i]\left(w \bullet x[i]+b\right) \geq 1-\varsigma_i \left(i=1,2,...n\right) \varsigma_i \geq 0 \qquad 3.2$$

But this poses another problem and some of the outer ones still follow the requirements for classification. We abandon the accuracy of those data penetrates. This is setback for our ranking of classifier. To give up the data points, on the other hand also implies that the hyper plane goes to this position so we have capacity. We must of course evaluate the cost and the disadvantages. We use the punishment factor C as a guidance. We are trying to get the more distant hyper plane and even a lower one value C.

Condemnation factor C is not a variable. When we fix this optimization technique, C is a predefined value. We get the predefined value for the rating. And test the results, then turn to another C if it's not good and replicate until the right one in found. We see that function of kernel and the slack variables are built to solve the issue, the linearly inseparable issue one mapped into high dimensions space. One fixes outlier data points. No matter how we navigate the classification plane, there is still going to be a lot of outlier. To solve the issue, we can use kernel function to map them into space of high dimensions. If they are not yet clearly distinct but stronger than in the original space, then we add slack variable to fix the issue. Based on the difference of defection methods, different SVM have different classification algorithm. There are C-SVM and nu-SVM, epsilon-SVM which provides support for regression. In this case we can use, c-SVM and nu-SVM.

c-SVM uses radial based kernel functions to mapped from low dimension space into high dimension space, see table 3.2. The penalty factor is 1, obtained by learning a collection of sample data. Parameter: Sparsely described in which manner it accepts sample data outliers. The smaller the spare parameter indicates we pay greater attention to outliers. We can see the result of evaluation of model in below tables. All kernel functions except linear kernel functions need Gamma; the default value is dimensions 1 / data. We need 1987 support vector to determine the hyper plane. The higher the support factor, the more calculation will be complex.

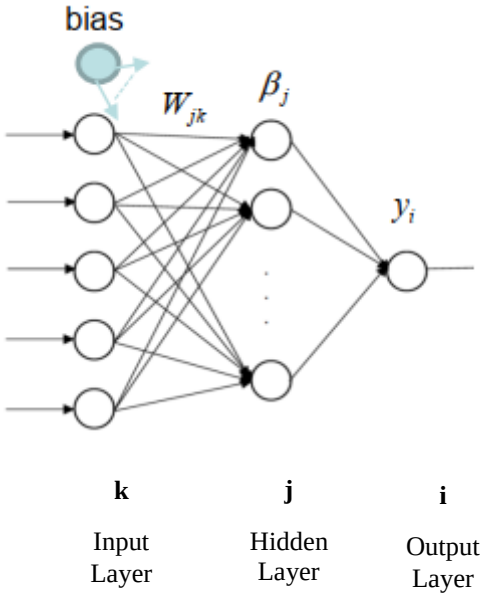
Prediction of extreme learning machine

Extreme learning machine is a basic single-layer feed-forward neural network (SLFN) is a fast learning algorithm proposed by Professor Huang Guang bin. Theoretically the ELM algorithm aims to provide good output at an extremely high learning level.

The ELM does not use a gradient-based strategy as opposed to conventional feedforward network learning algorithms such as back-propagation algorithms. All parameters are tuned in once with this system. This algorithm does not iterative instruction. The characteristic of this can be claimed that the input weights and the bias of the hidden layer can be randomly chosen without any change in the process of deciding network parameters, which increases the training speed and reduces the adjustment time of parameters.

At first create the random weights matrix and bias for input layer. The weight matrix and bias size is (j x k) and (1 x k) where j is the number of hidden nodes and k is the number of entry nodes.

$$W = \begin{bmatrix} rand & rand \\ rand & rand \end{bmatrix}$$



We need to look at how extreme learning machine (ELM) production is measured before going into depth.

$$f_L(x) = \sum_{i=1}^{L} \beta_i g_i(x) = \sum_{i=1}^{L} \beta_i g(w_i * x_j + b_i),\ j = 1,..,N$$

Where:

L is the quantity of input layer, N is the number of measures of training, w is the weight vector between the input and hidden layer, g is a function of stimulation, b is a vector of vias and x is a state vector.

There are many advantages that ELM holds, ELM learning speed can be thousand times higher than the traditional learning algorithm for back propagation, it gets the smallest training defect as well as the lowest weight rate, and it also achieves a good output in generalization compared to the other learning algorithm. At start there are seven filter based selection techniques were applied. The top $\lceil \log_2 n \rceil$ features were selected out of all the n ranked features. The filter based classifiers are based on the concept of the unique requirements of an individual aspect that can be calculated along precision measurements, auc and f-measurement[16], [17], [18].
Five subset collection strategies based on the wrapper features were also added. The wrapper based strategies are based on the premise that an attribute can work significantly well in a category, despite not having comparable results when separately examined. The approaches often involve a learning algorithm to find a set of characteristic that are useful in constructing the predict prediction model.

III. Results

In this section,we can examine the model's performance in general. Distribution of data defects and metrics by evaluating the data distribution of defects. We realize they are not equilibrium and they are similar to rules 20-80. The potential reason is that modules with more defects are always the main models and are used more often and in a more profound manner. Only part of the metrics have a major relationship with data of defects but they are not the same across different versions and different variability.

The impact of various metrics on the results of prediction by adjusting the metrics sets, we can find that the more metrics we have the more reliable the outcome is. The metrics sets replicate the software characters. Therefore the more metrics it includes the more reliable it is. But not all predictive models can manage the metrics well, so if you use more metrics there's no big improvement. We choose three different versions of eclipse (09 versions in total). The data can be downloaded from the development of the Eclipse program repository, version control system (VCS) and bug tracking for version 2.0, 2.1 and 3.0. We got the comparison results of two widely used machine learning techniques. Detail result of both classifiers are mentioned below.

Support vector machine

The detailed result of each dataset are shown below. And also line chart for file based and package based prediction SVM model for all three versions separately.By comparing both tables (3.1 and 3.2). We will note that findings from file-based predictions have a higher accuracy. But the outcome of package based prediction has an AUC higher.

Table 3.4 Result of file based prediction SVM Model

TRAINING	TESTING	ACCURACY	RECALL	PRECISION	AUC
File 2.0	File 2.0	0.8917	0.3187	0.8289	0.6486
	File 2.1	0.9105	0.1216	0.3642	0.5975
	File 3.0	0.8653	0.2098	0.6214	0.5863
File 2.1	File 2.0	0.8761	0.1498	0.7275	0.5688
	File 2.1	0.9008	0.1779	0.9185	0.5764
	File 3.0	0.8607	0.1128	0.7475	0.5548
File 3.0	File 2.0	0.8674	0.2766	0.6017	0.6028
	File 2.1	0.8915	0.2263	0.5174	0.5953
	File 3.0	0.8479	0.4149	0.4876	0.6690

Chart 3.4 Line charts of file based prediction SVM Model

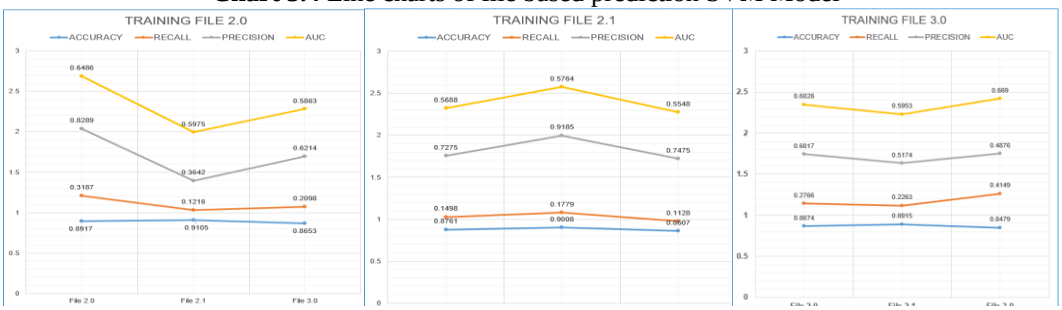
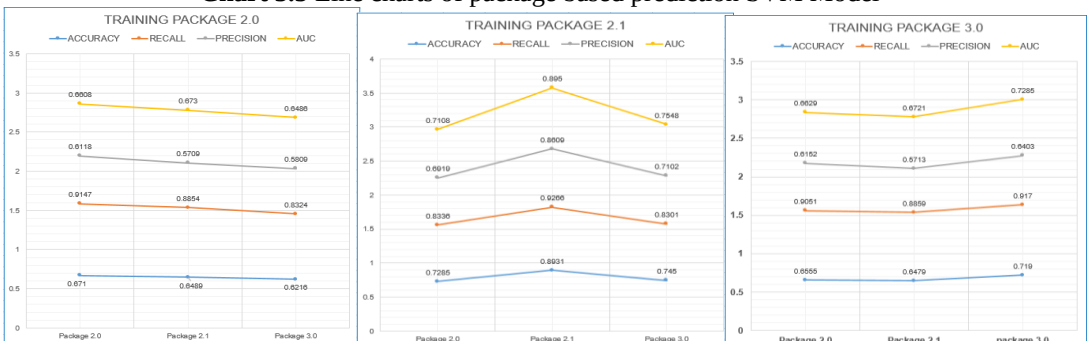


Table 3.5 Result of package based prediction SVM Model

TRAINING	TESTING	ACCURACY	RECALL	PRECISION	AUC
Package 2.0	Package 2.0	0.6710	0.9147	0.6118	0.6608
	Package 2.1	0.6489	0.8854	0.5709	0.6730
	Package 3.0	0.6216	0.8324	0.5809	0.6486
Package 2.1	Package 2.0	0.7285	0.8336	0.6919	0.7108
	Package 2.1	0.8931	0.9266	0.8609	0.8950
	Package 3.0	0.7450	0.8301	0.7102	0.7548
Package 3.0	Package 2.0	0.6555	0.9051	0.6152	0.6629
	Package 2.1	0.6479	0.8859	0.5713	0.6721
	Package 3.0	0.7190	0.9170	0.6403	0.7285

Chart 3.5 Line charts of package based prediction SVM Model



Extreme learning machine

In this section, the detailed result of extreme learning machine of each dataset are shown below. And also line chart for file based prediction SVM model for all three versions separately. By comparing both tables (3.4 and 3.5). We will note that findings from file-based predictions have a higher accuracy. But the outcome of package based prediction has an AUC higher.

Table 3.6 Result of file based prediction ELM Model

TRAINING	TESTING	ACCURACY	RECALL	PRECISION	AUC
File 2.0	File 2.0	0.8453	0.3687	0.6926	0.7051
	File 2.1	0.8502	0.3356	0.5421	0.6716
	File 3.0	0.8469	0.3589	0.4497	0.6545
File 2.1	File 2.0	0.8419	0.2268	0.4492	0.6398
	File 2.1	0.8367	0.2678	0.4637	0.6137
	File 3.0	0.8096	0.2477	0.5943	0.6272
File 3.0	File 2.0	0.8262	0.2667	0.5017	0.6786
	File 2.1	0.8526	0.2133	0.4162	0.6538
	File 3.0	0.8276	0.4138	0.4865	0.6691

Chart 3.6 Line charts of file based prediction ELM Model

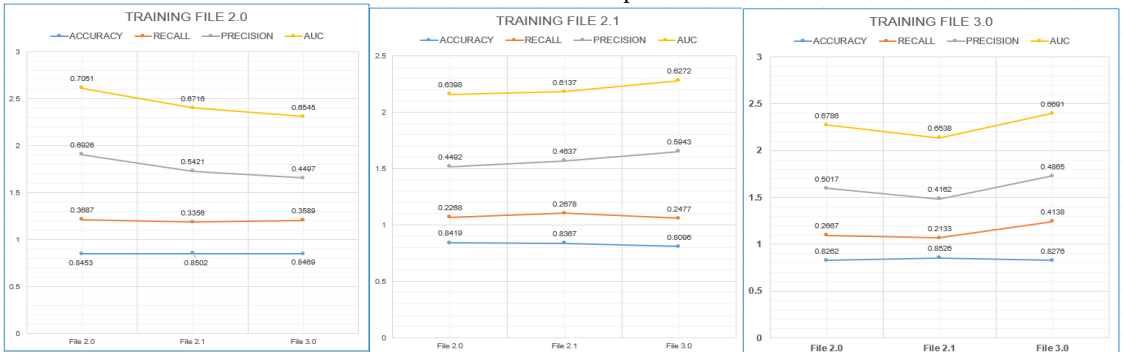
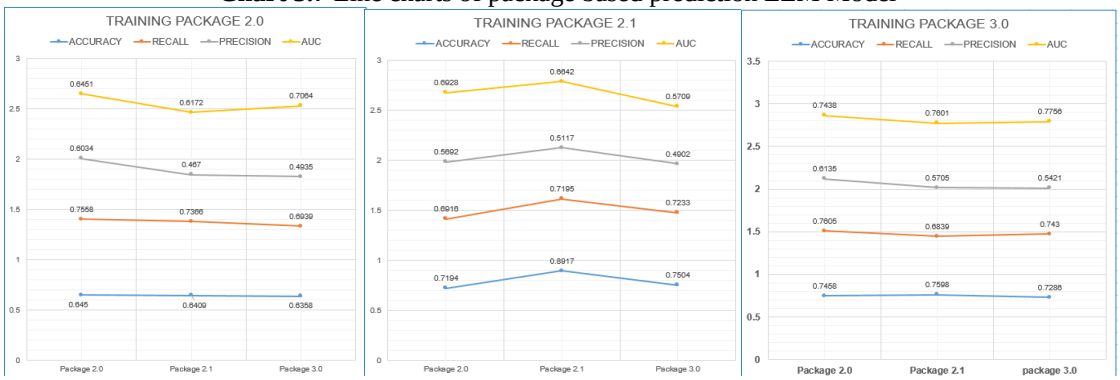


Table 3.7 Result of package based prediction ELM Model

TRAINING	TESTING	ACCURACY	RECALL	PRECISION	AUC
Package 2.0	Package 2.0	0.6450	0.7558	0.6034	0.6451
	Package 2.1	0.6409	0.7366	0.4670	0.6172
	Package 3.0	0.6358	0.6939	0.4935	0.7064
Package 2.1	Package 2.0	0.7194	0.6916	0.5692	0.6928
	Package 2.1	0.8917	0.7195	0.5117	0.6642
	Package 3.0	0.7504	0.7233	0.4902	0.5709
Package 3.0	Package 2.0	0.7458	0.7605	0.6135	0.7438
	Package 2.1	0.7598	0.6839	0.5705	0.7601
	Package 3.0	0.7286	0.7430	0.5421	0.7756

Chart 3.7 Line charts of package based prediction ELM Model



By comparing the file based tables of SVM and ELM we can find the average accuracy values and AUC. This means the extreme learning machine has the highest AUC value, but the value of accuracy is also close to SVM. And SVM has similar accuracy, and very close AUC value. Then we can see how these models perform in package based prediction. By comparing the data in package based prediction of SVM, ELM, the accuracy and AUC values shows thatSVM has best accuracy, but the value of AUC decreases apparently. So we can conclude that SVM has best prediction results in file based defects.

IV. Discussion

In this section, we can compare the results of two widely used classifiers which are used for software defects prediction. We have made a comparative study of SVM and ELM using different metrics, classifiers and techniques. We get the datasets form eclipse version control system. Our data consist of different versions, we split the data into test based and training based. After preprocess the data with the help of python language and by using different metrics we build our model thenwe trained our model using different machine learning techniques and algorithms. Our study also shows that these two classifiers are best for software defects prediction among all classifiers. As we mentioned the file based and package results of our models in above tables separately of all different versions of dataset. Our methods perform well the results as compare to previous proposed methods[19], [20].

In this study, based on the understanding and results of the mechanism and practical values of prediction of software defects, there is stills some possibility to enhance the research. In this study we just used two machine learning classifiers which are suitable for predicting work and we also used some limited metrics. We will use more machine learning techniques in the future to predict the number of defects, the type of defects and the solution for removing them.At this stage we only apply this model to Eclipse datasets but in future we will apply this model to different open source repositories datasets and as well as will test this model in cross platforms also.

V. Conclusion

Software testing has become more and more important in software reliability since last couple of years. Yet we spent a great deal of time, energy and money on software testing. Prediction of software defects may help increase the effectiveness of software testing and guide the direct allocation of resources. In this study, we have been discussed past related work and current situation of software defects prediction. We also addressed basic features, such as software metrics, classifiers and defect prediction models, and assessment.

We get the data for this research comes from Eclipse software development repository and VCS. And by using Eclipse plugin to calculate the software metrics in file based and as well as package based. By using Python we developed defect prediction models with the help of different algorithms. And then use these models to predict defects for file and packages. At the last, we discussed the performance of all these models. SVM has the best accuracy values among these models. So we can say that SVM is the best fit for software defect prediction. This will provide software testing resource guidance for predicting software predicts.

References

[1]. Y. Kamei and E. Shihab, "Defect Prediction: Accomplishments and Future Challenges," *2016 IEEE 23rd Int. Conf. Softw. Anal. Evol. Reengineering*, vol. 5, no. 1, pp. 33–45, 2016, doi: 10.1109/saner.2016.56.

[2]. M. D. Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," *2010 7th IEEE Work. Conf. Min. Softw. Repos. (MSR 2010)*, pp. 31–41, 2010, doi: 10.1109/MSR.2010.5463279.

[3]. F. Wu *et al.*, "Cross-Project and Within-Project Semisupervised Software Defect Prediction: A Unified Approach," *IEEE Trans. Reliab.*, vol. 67, no. 2, pp. 581–597, 2018, doi: 10.1109/TR.2018.2804922.

[4]. M. D. Ambros, M. Lanza, and R. Robbes, *Evaluating defect prediction approaches : a benchmark and an extensive comparison*. 2012.

[5]. N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models."
[6]. R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput. J.*, vol. 27, pp. 504–518, 2015, doi: 10.1016/j.asoc.2014.11.023.
[7]. R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with Android software," *Appl. Soft Comput. J.*, vol. 49, pp. 1034–1050, 2016, doi: 10.1016/j.asoc.2016.04.032.
[8]. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, 2012, doi: 10.1109/TSE.2011.103.
[9]. J. Nam, *Survey on Software Defect Prediction*. .
[10]. A. E. Hassan, "Predicting Faults Using the Complexity of Code Changes," *2009 IEEE 31st Int. Conf. Softw. Eng.*, pp. 78–88, 2009, doi: 10.1109/ICSE.2009.5070510.
[11]. M. Tang, M. Kao, and M. Chen, "An Empirical Study on Object-Oriented Metrics."
[12]. T. Menzies, J. Greenwald, and A. Frank, "to Learn Defect Predictors," vol. 33, no. 1, pp. 2–13, 2007.
[13]. V. R. Basili, L. Briand, W. L. Melo, V. Basili, and W. Melo, "A VALIDATION OF OBJECT-ORIENTED DESIGN METRICS AS QUALITY INDICATORS*," 1995.
[14]. S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," vol. 20, no. 6, pp. 476–493, 1994.
[15]. V. R. Basili, L. C. Briand, W. L. Melo, and I. C. Society, "A Validation of Object-Oriented Design Metrics as Quality Indicators," vol. 22, no. 10, 1996.
[16]. "Are Change Metrics Good Predictors for an Evolving Software Product Line.pdf." .
[17]. R. Ferenc, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," vol. 31, no. 10, pp. 897–910, 2005.
[18]. R. Malhotra, "Ac ce p te d us cr t," *Appl. Soft Comput. J.*, 2014, doi: 10.1016/j.asoc.2014.11.023.
[19]. X. K. Wei, Y. H. Li, and Y. Feng, "Comparative study of extreme learning machine and support vector machine," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3971 LNCS, pp. 1089–1095, 2006, doi: 10.1007/11759966_160.
[20]. J. Chorowski, J. Wang, and J. M. Zurada, "Review and performance comparison of SVM- and ELM-based classifiers," *Neurocomputing*, vol. 128, pp. 507–516, 2014, doi: 10.1016/j.neucom.2013.08.009.
[21]. M. Uzair and A. Mian, "Extreme Learning Machine Features," *Ieee Trans. Cybern.*, vol. 47, no. 3, pp. 651–660, 2017.
[22]. P. R. Bal and S. Kumar, "Cross project software defect prediction using extreme learning machine: An ensemble based study," *ICSOFT 2018 - Proc. 13th Int. Conf. Softw. Technol.*, no. Icsoft, pp. 320–327, 2019, doi: 10.5220/0006886503200327.
[23]. J. Cao and Z. Lin, "Extreme Learning Machines on High Dimensional and Large Data Applications: A Survey," *Math. Probl. Eng.*, vol. 2015, pp. 16–18, 2015, doi: 10.1155/2015/103796.
[24]. G. Bin Huang, D. H. Wang, and Y. Lan, "Extreme learning machines: A survey," *Int. J. Mach. Learn. Cybern.*, vol. 2, no. 2, pp. 107–122, 2011, doi: 10.1007/s13042-011-0019-y.
[25]. G. H. Ã, Q. Zhu, and C. Siew, "Extreme learning machine : Theory and applications," vol. 70, pp. 489–501, 2006, doi: 10.1016/j.neucom.2005.12.126.

Naeem Hussain, et. al. "Performance comparison of Machine learning classifiers in Software Defects Prediction." *IOSR Journal of Computer Engineering (IOSR-JCE)*, 22(5), 2020, pp. 48-58.