A Machine Learning Approach to Predict Software Faults

Satyarth Jha
Student
Department of Information Technology,
Maharaja Surajmal Institute of Technology,
GGSIPU, New Delhi

Samarth Jain
Student
Department of Information Technology,
Maharaja Surajmal Institute of Technology,
GGSIPU, New Delhi

Shubham Aggarwal
Student
Department of Information Technology,
Maharaja Surajmal Institute of Technology,
GGSIPU, New Delhi

Suman Mann
Associate Professor
Department of Information Technology,
Maharaja Surajmal Institute of Technology,
GGSIPU, New Delhi

ABSTRACT

Software fault prediction is a research area that ensures software is bug-free before being put into production during the development and testing phases[5]. Predicting software breakdown is a critical factor to consider in software development as it ensures software reliability and quality. We classified National Aeronautics and Space Administration (NASA) Promise datasets, namely CM1, JM1, and PC1. It is imperative to anticipate faults in software such as components, classes, and modules at an early point in the development cycle. Costs and time will be reduced as a result. Finding as measured by the f1 score three defect datasets yielded a classification performance of 99-100% with the best execution using the Support Vector matrix.

Keywords: Fault Prediction, Software Engineering, Testing, Software Metrics, Fault Analysis, Machine Learning, Defect Prediction, Random Forest, decision tree, support vector matrix, promise dataset, Support vector matrix

1. INTRODUCTION

Software fault prediction (SFP) is a research area which contributes to the development & testing phases which ensures software of a good grade. Fault prediction is the topic of numerous research studies. Many methods are recommended for the prediction of software faults, which include machine learning, and Statistical methods[6][8]. This method helps developers to lower maintenance costs by analyzing our codebase in development.

This feature helps the software to work more efficiently and lessens the faults, time and cost. In this paper, a software defect predictive development model using McCabe and Halstead metrics is implemented using machine learning techniques. Predicting software breakdowns is an important aspect to consider in software development as it ensures software reliability and quality. This is important for publishing software versions that are dependent on predefined metrics because of historical defects in the software. A high-quality software product is composed of fewer flaws and failures.

Predicting software failures is critical to the software quality assurance process as it looks at the threats and vulnerabilities of software products for different aspects and makes the software less vulnerable to failure. If we have the faults before the launch of the software then it would save the company a lot of money and its respect in the market.

The faults are not evenly distributed within the software parts. Many classes tend to have a relatively higher number of faults compared to others and are grouped into a limited number of classes. Source code quality is estimated through internal metrics whereas external metrics are used to calculate the functionality or behaviours of the software.

Techniques of segregation are used to labialize the code as either fault-less or with fault by using metrics set with erroneous data. The software quality is made better by finding faulty instances in the software by using fault-predicting models. The model performance influences the model technique.

2. Literature Review

To predict and prevent bugs in production researchers have implemented and worked on various machine-learning approaches. It is known that software maintenance is the most expensive phase in the software development lifecycle[4]. A software defect predictive model enables organizations to help to reduce the maintenance effort, time and cost overall on a software project [3] [4]. The various researched algorithms are a result of various findings and correlations between some software metrics and fault proneness[11].This paper uses 4 of many ML classifiers as suggested by a recent systematic literature study[4]. As two studies stated that machine learning-based models for software fault prediction[11][12]. Naive Bayes is a generic and good algorithm but is not efficient to be used with larger datasets[7].

3. Propose Method

3.1 Dataset

This research paper has used 3 publicly available data from PROMISE Software Engineering Database by NASA[1]. There are 22 different attributes from software defect datasets, including 21 independent metrics such as McCabe's cyclomatic & essential complexity, Halstead time estimator, program length & effort. One metric is outcome-based. i.e., which instance is faulty and which is not.

The three datasets are CM1 (498 instances), PC1 (1109 instances) and JM1 (10885 instances).

3.1.1 McCabe Metrics:

McCabe suggested and showed that the codes with more branching and easy flow structure are more error-prone [1]. The following McCabe metrics are thus

1. McCabe Cyclomatic Complexity [v(G)]: Number of linear independent path decisions that the compiler has to make through a program module. Empirical evidence shows that dependency cycles are known to be detrimental to software quality attributes such as understandability, testability, reusability, build-ability and maintainability[2].

2. $V(G) = E - N + 2$ where E - graph edge, N graph node

3. McCabe Essential Complexity [ev(G)]: Number of linear independent path decisions that can be reduced by decomposing a program module.

4. ev(G) = v(G) – m, where m is the number of sub-flowgraphs of “G” that are D-structured primes.

5. McCabe Design Complexity [iv(G)]: It tries to relate to the difficulty of understanding a design due to difficulty in understanding the program module calling patterns to different program modules.

6. McCabe Line of Code: It counts the number of lines computed by the compiler/interpreter.

3.1.2 Halstead Metrics

Halstead suggested and showed that the harder the code is to read, it is more likely to error-prone. The following Halstead metrics are thus:

1. uniq_Op: It is a measure of how many unique operators are present in the program.
2. uniq_Opnd: It is a measure of how many unique operands are present in the program.
3. total_Op: To: It is a measure of total operators present in the program.
4. total_Opnd: It is a measure of total operands present in the program.

5. Halstead size [n]: Operators and operands that are present in total.
   n = totalOp + total_Opnd
6. Halstead vocabulary [N]: Unique operators and operands that are present in total.
   N = uniq_Op + uniq_Opnd
7. Halstead Volume [v]: It describes the total size of the implementation of the codebase.
   v = N*log2(n)
8. Halstead Difficulty [d]:
   d = (uniq_Op/2) * (total_Opnd/unique_Opnd)
9. Halstead Effort [e]:
   e = v * d
10. Halstead Errors [b]:
    b = v / 3000
11. Halstead Time Estimator [t]: Shows time (in minutes) required to transform the current algorithm to the desired program language for compilation.
    t = e / k
    k = 18 (stroud number, arbitrary value)
12. loCode: It is Halstead's line count.
13. loComment: It is Halstead's count of the line of comments.
14. loBlank: It is Halstead's count of blank lines.
15. loCodeAndComment: It is sum of loCode and loComment.
16. branchCount: It is the total branches/ decision path created due to the flow of code.
17. defects: Boolean’s value represents if there has/has not been one or more reported defects.

3.2 Covariance Matrix

A statistical method for determining the relationship between two variables. Calculated (here) to get the estimation of our dataset. The positive covariance value indicates a positive relationship between variables, and the negative value indicates a negative relationship.

3.3 Confusion Matrix

The confusion matrix is a technique for evaluating machine learning classification performance. The matrix used here is

	Module has defect	**Modul have no defect**
Classifier predicts no defects	a	b
Classifier predicts some defects	c	d

Hence, we can calculate the following parameters follows

Hence, we can calculate the following parameters follows

1. Accuracy (acc) = (a + d) / (a + b + c + d)
2. Probability of some defects (pd or recall) = d / (b + d)
3. Probability of false alarm (pf) = c / (a + c)
4. Precisions (prec) = d / (c + d)
5. F1 score = (2 * prec * recall) / (prec + recall)

3.3.1 Time of completion

Each algorithm's time is computed on the same machine under the same conditions.

3.3.2 Inserted Attribute

Added a new value with the column label complexityEvaluation for each instance. In the case of (instance.n x 300) && (instance.v x 1000) && (instance.d x 50) && (instance.e x 500000) && (instance.t x 5000), we add 'Success' if otherwise 'Redesign'.

3.4 Naive Bayes Algorithm

This is a supervised machine learning algorithm. It is based on Bayes Theorem. It is predominantly used in text-based classification that includes a poly-dimensional training dataset. The naive Bayes Algorithm is a probabilistic classifier. It predicts objects based on their probability. Naïve Bayes consists of two words which can be described as

Naive- It means that the occurrence of one will not be dependent on the other.

Bayes- It relies on Bayer's Theorem, which is what gives it its name.

Bayes Theorem-

Bayes Theorem states that the conditional probability of an event is based on the occurrence of another event and is equal to the probable occurrence of the second event given the first event multiplied by the probability of the first event.

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$ Where,

P(A|B) - Posterior probability: Hypothesis A's probability of being true for the observed event B.

P(B|A) - Likelihood probability: Given that a hypothesis has a high probability of being true, this statistic shows how likely it is to be true.

3.5 Decision Tree

It's a method of supervised learning that may be applied to classification and regression. It's a tree-structured classifier, where internal nodes stand in for a dataset's features, branches for the decision-making process, and each leaf node for the classification result.

The features of the given dataset are used for the execution of the test or to make decisions. It's a graphical definition for carrying all feasible answers to a choice or problem grounded on predetermined conditions. It's known as a decision tree because, like a tree, it begins with the root node and also spreads on other branches to form a structure suggesting a tree.

To construct a tree, the CART algorithm which stands for Classification and Regression Tree algorithm is used. A decision tree only poses a question and divides the tree into subtrees according to the response(Yes/ No).

3.6 Random Forest Algorithm

The Random Forest algorithm belongs to the supervised learning category of machine learning algorithms. It is used for Regression and Classification. Random Forest has a number of decision trees on various subsets of the given dataset. Random Forest takes the average to improve the probable accuracy of the dataset. Random Forest takes decisions from each tree instead of relying on one decision tree. Then based on the majority votes of predictions, Random Forest Algorithm predicts the final output. The higher number of trees in the forest leads to greater precision.

The random Forest Algorithm is used in this project because it takes less time compared to other Algorithms. Random Forest Algorithm predicts the output of the dataset with high accuracy. When a large proportion of data is missing, Random Forest Algorithm maintains its accuracy.

Random Forest Algorithm works in mainly two parts. In the first part, it combines various decision trees and in the second part, it makes predictions for each tree it combines in the first part. It then gives the average by using the above two mentioned steps. Further, this process can be divided into five parts-

1. It selects Random points from the training dataset.
2. It builds the decision trees associated with the selected data points (Subsets).
3. Choose the number of decision trees that are needed.
4. Repeat Steps 1 & 2.
5. When any new data point comes, it finds the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

3.7 Support Vector Machine Algorithm

Support Vector Machine, or SVM, is used to break Classification and Regression problems. The SVM algorithm's ideal is to establish the best line or decision boundary that can divide n-dimensional space into classes, allowing us to snappily classify fresh data points in the future. Optimal decision boundaries are called hyperplanes.

Extreme vectors and points are selected by SVM which aid in the creation of the hyperplane[10]. The algorithm is known as a Support Vector Machine because these extreme exemplifications are known as support vectors.

Face identification, image classification, and text categorization. may all be done using the SVM system. SVM comes in two kinds:-

Linear SVM: Linear SVM is used for data which can be segregated into two classes by using a single straight line. This type of data is called linearly divisible data, and the classifier employed is known as a Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means that a dataset is considered non-linear if it cannot be classified using a straight line. Non-linear SVM classifiers are used to categorise similar datasets.

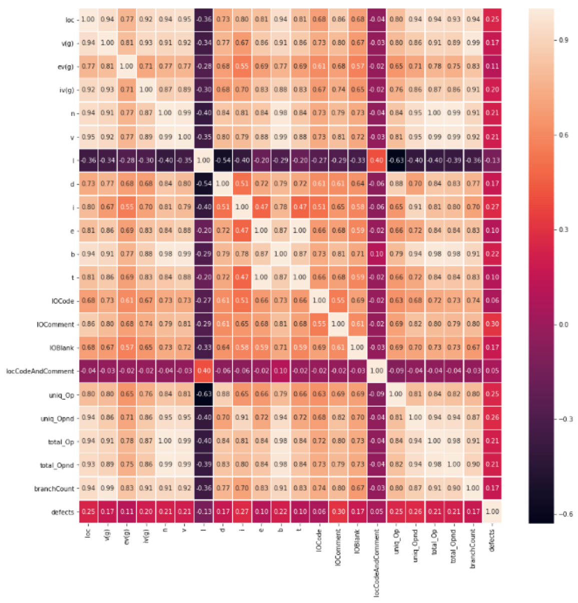

Figure 1: Histogram to visualize the covariance factor in the CM1 dataset

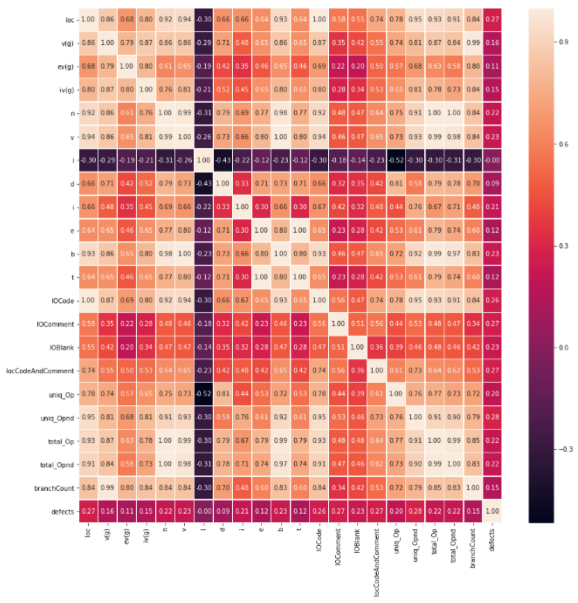

Figure 2: Histogram to visualize the covariance factor in the PC1 dataset.

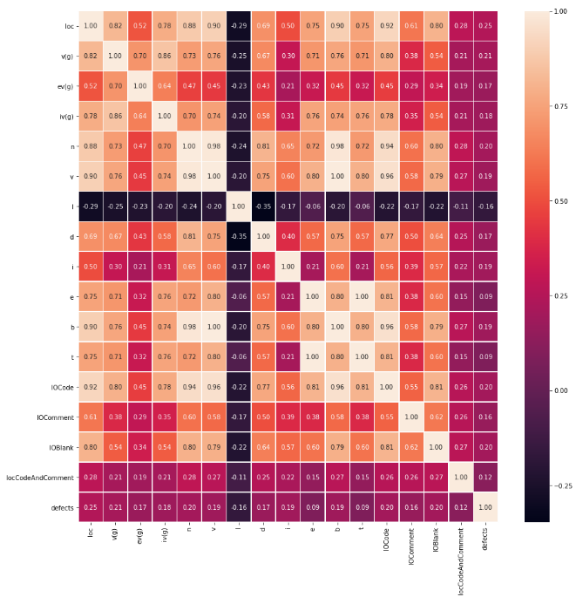

Figure 3: Histogram to visualize the covariance factor in the JM1 dataset

4. Implementation and Result

For the purpose of this research paper, we use the Sklearn library to train our model and perform various computations using it. As a result, the following variables have been used to determine the outcome

```
validation_size = 0.20
seed = 7
```

For naive Bayes, we have used K-fold cross-validation with parameter values as n_splits = 10, and random_state = seed.

```
model = GaussianNB()
scoring = 'accuracy'
```

```
kfold = model_selection.KFold(n_splits = 10,
random_state = seed)
```

We construct a decision tree model as

```
model = tree.DecisionTreeClassifier()
```

For random forest, we have used a n_estimator equal to 100

```
model=RandomForestClassifier(n_estimators
=100)
```

For SVM we have used kernel = ‘linear’, C = 0.01’

```
model = svm.SVC(kernel='linear', C=0
```

Algorithm	Precision	Recall	F1-score	Accuracy	Time Taken
CM1 Dataset					
Naive Bayes	1.00	1.00	1.000000	1.000000	669ms
Decision Tree	1.00	1.00	1.000000	1.000000	530ms
Random Forest	1.00	1.00	1.000000	1.000000	546ms
SVM	1.00	1.00	1.000000	1.000000	400ms
PC1 Dataset					
Naive Bayes	0.96	0.96	0.96	0.954959	1246ms
Decision Tree	0.97	0.97	0.97	0.959459	1156ms
Random Forest	0.99	0.99	0.99	0.990990	800ms
SVM	1.00	1.00	1.00	0.995495	648ms
JM1 Dataset					

Naive Bayes	0.98	0.98	0.98	0.981626	6488ms
Decision Tree	1.00	1.00	1.00	0.989540	5684ms
Random Forest	1.00	1.00	1.00	0.989621	3154ms
SVM	1.00	1.00	1.00	0.999540	1265ms

The above comparison of the results shows that as the dataset size increases there is a noticeable difference between all applied algorithms. As a result, SVM is found to be the most accurate alternative with an accuracy of 99.95% and the fastest average runtime of all applied algorithms.

5. Conclusion & Future Scope

The use of machine learning techniques is increasing and a variety of performance metrics are being used to measure their effectiveness. Hence as part of the analysis in this paper, a literature review is conducted in order to determine the effectiveness of measurements on SFPs. It was carried out by compiling, classifying, and analyzing published studies that looked at fault prediction. A total of 21 measures have been found in public datasets in a variety of combinations of 21 different measures. The use of machine learning techniques is increasing and a variety of performance metrics are being used to measure their effectiveness.

With a focus on performance and accuracy, this study will let researchers efficiently and effectively compare past studies from measurements, techniques, datasets, performance evaluation metrics, and experimental results viewpoints. We hope that some researchers may employ these components discussed in this paper in their subsequent work and try to evaluate different metrics from the ones we used. Additionally, there will be a lot of work to be done using regression employing machine learning approaches for error prediction with these metrics.

6. References

[1] PROMISE Software Engineering Repository (Sayyad and Menzies, 2005, Promise Software Engineering Repository Public, 2013).

[2] T.D. Oyetoyan *et al.,* A study of cyclic dependencies on defect profile of software components, Journal of Systems and Software, 2013

[3] I Gondra, Applying machine learning to software fault-proneness prediction, Journal of Systems and Software, 2008

[4] P. Oman and J. Hagemeister, “Construction and testing of polynomials predicting software maintainability,” J.

Syst. Softw., vol. 24, no. 3, pp. 251–266, Mar. 1994.

[5] T. Anderson, P. A. Barrett, D. N. Halliwell, and M. R. Moulding, “Software Fault Tolerance: An Evaluation,” IEEE Trans. Softw. Eng., vol. SE-11, no. 12, pp. 1502–1510, Dec. 1985.

[6] L. Son et al., “Empirical Study of Software Defect Prediction: A Systematic Mapping,” Symmetry (Basel)., vol. 11, no. 2, p. 212, Feb. 2019.

[7] Burak Turhan, Ayse Bener, Analysis of Naive Bayes’ assumptions on software fault data: An empirical study, Data & Knowledge Engineering, Volume 68, Issue 2, 2009, Pages 278-290

[8] Azeem, M.I.; Palomba, F.; Shi, L.; Wang, Q. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Inf. Softw. Technol. 2019, 108, 115–138.

[9] Ghannem, A.; Boussaidi, G.E.; Kessentini, M. Model refactoring using interactive genetic algorithm. In International Symposium on Search-Based Software Engineering; Springer: Berlin/Heidelberg, Germany, 2013; pp. 96–110.

[10] Cervantes, J.; Garcia-Lamont, F.; Rodríguez-Mazahua, L.; Lopez, A. A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing 2020, 408, 189–215.

[11] C. Catal and B. Diri, “A systematic review of software fault prediction studies,” Expert Syst. Appl., vol. 36, no. 4, pp. 7346–7354, May 2009.

[12] V. U. B. CHALLAGULLA, F. B. BASTANI, I.-L. YEN, and R. A. PAUL, “EMPIRICAL ASSESSMENT OF MACHINE LEARNING BASED SOFTWARE DEFECT PREDICTION TECHNIQUES,” Int. J. Artif. Intell. Tools, vol. 17, no. 02, pp. 389–400, Apr. 2008.

[13] P. N. Tan, M. Steinbach, and V. Kumar, Introduction to data mining. Pearson Addison Wesley, 2005.

[14] Sushant Kumar Pandey, Ravi Bhushan Mishra, Anil Kumar Tripathi, Machine learning based methods for software fault prediction: A survey, Expert Systems with Applications, Volume 172, 2021