Celal Bayar University Journal of Science
Volume 14, Issue 3, 2018, p 297-302

Software Fault Prediction in Object Oriented Software Systems Using Ensemble Classifiers

Emin Borandağ*, Fatih Yücalar, Kamil Akarsu

Department of Software Engineering, Faculty of Technology, Manisa Celal Bayar University, Manisa, Turkey
*emin.borandag@cbu.edu.tr

Received: 17 May 2018
Accepted: 21 September 2018
DOI: 10.18466/cbayarfbe.424521

Abstract

The main aim of software projects is developing software programs to meet functional and non-functional requirements within the project budget and at a particular time. The greatest challenge in reaching this goal is the software errors that were found in the software projects. The most basic technique that is used to solve software errors is testing the software programs according to the methods in the literature. These methods are the software tests that are basically conducted by software developers, although they have different methods of verification and validation according to their size, experience, techniques or tools they use. When software is tested, it is very significant that software errors are found in the early phases. Software error estimation is a proven method of effectiveness and validity that increases the quality of software and reduces the cost of software development. In this study, by using machine learning algorithms and software metrics; software error estimation has been carried out with a developed software.

Keywords: Data mining, software fault prediction, rotation forest algorithm, ensemble learning.

1. Introduction

To obtain successful results, software projects should be completed at the planned time and within the limits of project budget, and satisfy customers' functional and non-functional requirements. However, the biggest challenge faced in achieving this aim is software faults detected during the development of software projects. The most basic technique used to correct software faults is testing the software according to the methods in literature. While these are diverse confirmation and validation methods varying based on the size, experience, techniques or tools of the software development firms, what is done basically is software testing. It is highly important that software faults are detected at earlier stages. There are huge differences between the cost of correcting a fault detected in needs analysis and a fault found at later stages. By keeping an account of the faults encountered in the software, estimations are made as to whether there is any fault in the next module in an attempt to reduce fault correction costs. Software fault predication can be made using software metrics [1]. In addition, the use of software metrics makes it easier to carry out software maintenance and allows adding new properties to the software in a shorter time [1]. Besides, the use of software metrics makes the software more reliable, robust, high performing and more easily testable [2]. In this context, software fault prediction can be defined as estimating where defects can be in newly developed modules or current modules by keeping an account of the defects in the previous software and using software metrics [3].

Software fault prediction is a method with proven efficiency and validity that enhances software quality while reducing software development cost [4]. This method is used to determine the relationships between software metrics and faults. Many researchers have attempted to develop various prediction methods to reveal the relationship between software metrics and software faults [5].

Part 2 in the study provides basic information about software metrics and explains the importance of software testing. Part 3 provides an insight into similar studies in literature while Part 4 clarifies evaluation criteria used for the accuracy of classification. The data set used in the study, the software and test results are explained in Part 5 and Part 6. The final part of the study presents results and future work.

2. Background

2.1 Software Metrics

Software faults affect software quality negatively. Although it is difficult to say anything directly about the quality of software projects, it is possible to have information about the quality of the source codes of the projects through assessments on the source codes based on the metrics [6].

Software metrics help us to understand technical processes used in the development of a software product. Software assessment is a necessary process for understanding, controlling and managing software processes. It helps to analyze software faults, decrease software complexity, and monitor and evaluate processes. Metrics are standards that define measurable attributes of objects. Software metrics are values that can be assessed or calculated based on assessments. There are also definitions for different metrics such as the number of faults detected during the development of software or the number of faults detected in the code review. The metrics recorded during software development process help to monitor the progress of the software and increase its quality. The use of these metrics also helps to see whether testing process is going as planned. Here, the most fundamental metric is the number of running and failed test cases. While looking at the number of faults, significance level of faults is also taken into account. Even if there is no problem with software metrics being calculated, more software faults can always be found since not all circumstances and situations are tested in the software. However, software tests or metrics do not indicate that there is no fault in the software.

2.2 Software Testing

There are two types of software testing: white-box testing and black-box testing. White-box and black-box tests are performed together to reduce software faults by validating software interface and verifying internal functioning of the software. Software can be tested using automatic code review tools; however, diverse methods need to be applied to detect the location of software faults with greater possibility. For this purpose, machine learning classification algorithms are used. Classification is determining in which class the new data belongs when the class of the data collected in the past is already known [7]. Thus, the objects whose class is known (learning data set) are used to establish a model. This model is tested with the objects that are not part of the learning set (testing data set) to measure its performance.

There are various types of classification algorithms in literature such as decision trees, bayes classifiers, rule-based classifiers, artificial neural networks, nearest neighbors' classifiers (KNN), support vector machine and community learning methods.

3. Related Work

In the past 15 years, people from diverse areas like software developers, software engineers and researchers in the sector have been conducting studies on software fault prediction approaches [1, 3, 4 and 7].

In his study in 1999, Kaszycki used process metrics and product metrics in software fault prediction. He used 4 product metrics obtained from the source code in the study. As evaluation metric, true negative and true positive rates were used. Thanks to this study, a risk assessment tool was developed [8].

In another study conducted by Xu, Khoshgoftaar and Allen (2000), feature selection and fuzzy nonlinear regression (FNR) methods were applied for the first time in software fault prediction. They used the data from 24 method-level metrics obtained from the software of a very large legacy telecommunications system written in Protel (a high-level language) in order to estimate faulty modules in the software [9].

Reformat conducted a study in 2003 using fuzzy rule-based models in fault prediction. He used 11 method-level metrics obtained through commercial medical imaging software for defect prediction [10].

Menzies and Di Stefano (2004) conducted a study using the data set created by NASA for software fault prediction. They used Linear Standard Regression (LSR) method in the study. The study indicated that Cyclomatic complexity was directly related to software faults. They detected software faults when Cyclomatic complexity value was 10 or above [11].

In 2007, Cagatay Catal and Banu Diri developed a model based on class level metrics. They used Artificial Based Immune Recognition (AIRS) algorithm for fault prediction in the system they developed [12].

In their study in 2010, Marco D'Ambros, Michele Lanza and Ramain Robbes created a data set consisting of the defects in Eclipse JDT Core, Eclipse PDE UI, Equinox framework, Mylyn and Apache Lucene software systems. In the study, they predicted software faults using this data set and software metrics (CK and OO metrics) [13].

In another study conducted by Ayse Tosun Mısırlı, Ayşe Başar Bener and Burak Turhan (2011), classifier ensembles were used for fault prediction in embedded software systems. In the study conducted through ensemble classifiers, the researchers observed a 15% FP fall rate [14].

In their study in 2012, Supreet Kaur, and Dinesh Kumar applied Density Based Clustering approach in object-oriented software systems. In the study conducted through Density Based Spatial Clustering Applications with Noise (DBSCAN) algorithm, 8 different attributes and NASA's Metrics Data Program (MDP) data repository were used as data set [15].

4. Assessment Criteria

In this study, Confusion Matrix is used for the accuracy of the classification model. In Confusion Matrix, rows display class label while columns show prediction results. A confusion matrix example is given in Table 1.

Table 1. Confusion Matrix.

Estimated Class		
	Prediction No (-)	**Prediction Yes (+)**
No (Actual)	Tn	Fp
Yes (Actual)	Fn	Tp

Accuracy rate (ACC) is a widely used measure to identify distinction capacity of the classifier. It is defined as the percentage of test samples that are correctly classified by the algorithm. ACC is one of the primary metrics used in the assessment of classifier performances [16]. ACC rate is calculated using the formula given in Equation 4.1.

$$\mathrm{ACC} = \frac{(\mathrm{TP} + \mathrm{TN})}{\mathrm{Positive} + \mathrm{Negative}} \quad \textbf{(4.1)}$$

Another widely used metric is the area under ROC curve (Area Under Curve - AUC). The higher the AUC rate, the better is the accuracy rate of the classifier. AUC values vary between 0.5 - 1.0. It is calculated using the formula in Equation 4.2 [16].

$$\mathrm{AUC} = \frac{1}{2}\left(\frac{(\mathrm{TP})}{\mathrm{TP} + \mathrm{FN}} + \frac{(\mathrm{TN})}{\mathrm{TN} + \mathrm{FP}}\right) \quad \textbf{(4.2)}$$

In machine learning algorithms and particularly in the studies on fault prediction, there are other indicators as well such as KE, Probability of Detection, True Negative Rate, G_mean_1 and F-measure [16].

5. Experimental Study

5.1 Metrics Used in the Study

The study uses the metrics obtained from a software project. The metrics are as follows: Lines, Statements, Comment Rate, Docs Rate, Classes Number, Methods/Class, Class/Method, Maximum Complexity, Statements/Method Maximum Depth, Average Depth, Average Complexity and Fault rates. To calculate Cyclomatic Complexity (CC) metric, the following formula was used [17, 18].

$$\mathrm{CC} = \mathrm{E} - \mathrm{N} + 2\mathrm{P} \quad \textbf{(5.1)}$$

E is the number of edges of the graph, N is the number of nodes of the graph and P is the number of connected components [18]. Comment value was determined in percentage. A percentage value was taken in the statement.

$$Comment\ Rate = \frac{Comment\ Statements}{Total\ Statements} \quad \textbf{(5.2)}$$

For calculating average depth, the block depths of each line in the code were added, and the result was divided by total statement value [19].

$$Avg\ Depth = \sum_{i=0}^{n} \frac{(\mathrm{BlockDepth(i)} * \mathrm{Statements(i)})}{Total\ Statements} \quad \textbf{(5.3)}$$

5.2 Data Set

The data set obtained from the software consists of 472.443 lines and 241.377 statements in total. There are 1893 classes in the software. Other metrics in the software are as follows: 2.8% Comment rate, 9.1% Doc rate, 14.2 Methods/Class ratio, 5 Calls/Method ratio, 8.82 Stmts/Method ratio, 6.76 Max complexity ratio, 3.94 Max Depth ratio, 1.91 Avg Depth ratio, and 2.75 Avg Complexity ratio, on average. The parameter value of the statement with the highest block depth was taken as Max Depth value in the data set. The highest subroutine value in the code was taken as the Biggest Subroutine value.

Table 2. The description of datasets

File Name	Lines	Stmts	Comments (%)	Docs (%)	Methods/ Class	Calls/ Method	Stmts/ Method	Max C.	Max Depth	Avg. Depth	Avg. C.	Error
Sample1	304	119	3	17.8	10	2	10	3	3	2.18	3	No
Sample2	155	64	0	26.5	5	2	10	3	3	2.03	3	No
Sample3	630	240	1.3	24.1	21	2	10	3	3	2.27	3	No
Sample4	397	152	3.3	17.9	13	2	10	3	3	2.22	3	No
Sample5	393	152	3.3	17.3	13	2	10	3	3	2.22	3	No
Sample6	186	75	3.2	15.6	6	2	10	3	3	2.08	3	No
Sample7	719	368	5	8.1	11	20.55	31.36	6	5	2.94	4.09	Yes
Sample8	243	148	2.1	16.9	5	17.4	26.2	4	5	2.33	3.2	No
Sample9	1284	705	3.3	12.1	22	19.68	30.5	6	5	2.78	3.86	No

C.: Complexity; Avg.: Average; Stmts: Statements

Table 2 presents the examples taken from the data set. The USP-1299 dataset has been made publicly available for researchers to conduct their own studies [20].

5.3 Evaluation Criteria

Using Weka data mining software, 10 base classification algorithms and 10 ensemble classifiers algorithms that are used for software fault prediction in literature were tested through leave-one-out method. These algorithms can be seen in Table 3 and Table 4.

In this method called as 10-fold cross validation, classification accuracy is computed 10 times, each time leaving out one of the subsamples from the computations and using that subsample as a test sample for cross validation. In this structure, each subsample is used 9 times in the learning sample and just once as the test sample [16].

Table 3. Base Classifiers Algorithms

No	Algorithm
1	Bayes Net
2	Naïve Bayes (NB)
3	Logistic Regression (LR)
4	RBF Network (RBF)
5	Sequential Minimal Optimization (SMO)
6	Pegasos (PG)
7	Voted Perceptron (VP)
8	Instance Based Learner (IB1)
9	KStar (KS)
10	Jrip (JR)

Table 4. Ensemble Classifiers Algorithms

No	Algorithm
1	AdaboostM1 (AB)
2	Logic Boost (LB)
3	MultiBoost AB (MAB)
4	Bagging (BG)
5	Decorate
6	Dagging (DG)
7	Rotation Forest (ROF)
8	Stacking (ST)
9	Multi Class Classifier (MCC)
10	Voting (VT)

5.4 Developed Software

Thanks to the software developed using Weka library and Java programming language, data with .csv and .arff extensions can be selected. The selected data can be run according to machine learning algorithm. In addition, ACC and F-Measure values can be calculated. The screen shot of the software is given in Figure 1. The developed software has been made publicly available for researchers to conduct their own studies [20].

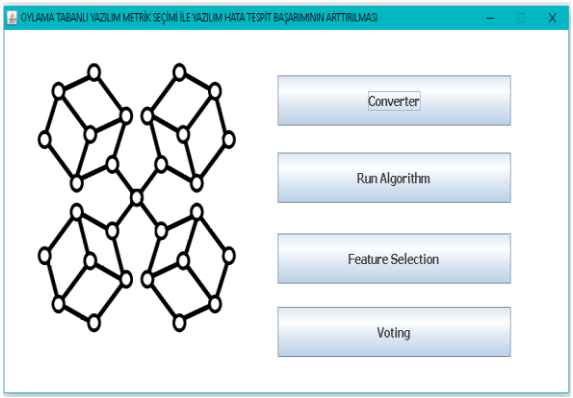

Figure 1. Developed Software

6. Experimental Results

In the study, the data set including software faults and created with the metrics obtained from the software being developed was tested using 10 base classifier algorithms and 10 ensemble classifiers algorithms. The test results are given in Table 5 and Table 6.

Table 5. The Base Classifiers

No	Class	Algorithm	ACC	AUC
1	Bayes	Bayes Net	86.61	89.2
2	Bayes	Naive Bayes	92.07	87.9
3	Function	Logistic Regression	93.37	87.4
4	Function	RBF Network	92.80	85.2
5	Function	SMO	91.91	95.8
6	Function	Pegasos	91.68	95.6
7	Function	Voted Perceptron	91.68	95.7
8	Lazy	Instance Based Learner	90.68	95.0
9	Lazy	KStar	91.14	95.2
10	Rules	JRip	92.76	96.1
		Average	**91.47**	**92.31**

Table 6. The Ensemble Classifiers

No	Class	Algorithm	ACC	AUC
1	Meta	AdaboostM1 (AB)	93.61	96.5
2	Meta	Logic Boost (LB)	93.22	96.4
3	Meta	MultiBoost AB (MAB)	94.07	96.8
4	Meta	Bagging (BG)	91.68	95.7
5	Meta	Decorate	93.14	96.3
6	Meta	Dagging (DG)	91.91	95.8
7	Meta	Rotation Forest (ROF)	93.30	96.4
8	Meta	Stacking (ST)	91.68	95.7
9	Meta	Multi Class Classifier	93.37	96.5
10	Meta	Vote	91.68	95.7
		Average	**92.77**	**96.18**

6.1 Assessment

Evaluation of Table 5 and Table 6 together shows that ensemble classifier algorithms increase ACC metric value by 1.30% and AUC metric value by 3.87% compared to the base classifiers algorithms. The highest accuracy rate in ACC rate in the base classifiers is obtained from Logistic Regression algorithm by 93.37% while the highest accuracy rate in ACC rate in the ensemble classifiers is obtained from MultiBoost AB algorithm by 94.07%. The highest accuracy rate in ACU rate in the base classifiers is obtained from JRip algorithm by 96.1%, and the highest accuracy rate in ACU rate in the ensemble classifiers is obtained from MultiBoost AB algorithm again by 96.8%. The graphs displaying the results of the study are given in Figure 2 and Figure 3. MultiBoost AB algorithm again by 96.8%. The graphs displaying the results of the study are given in Figure 2 and Figure 3.

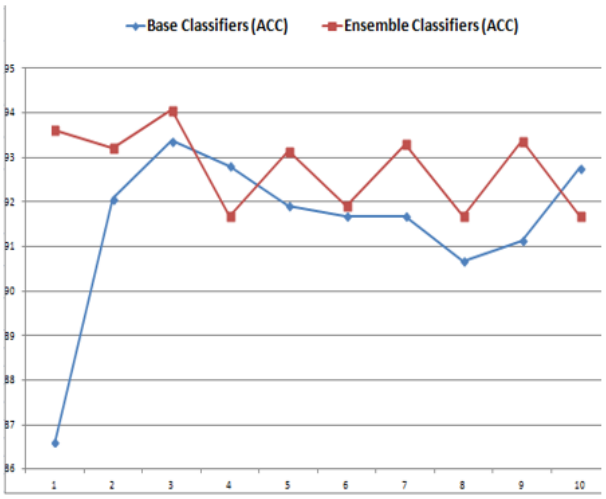

Figure 2. ACC Rates (Base Classifiers & Ensemble Classifiers)

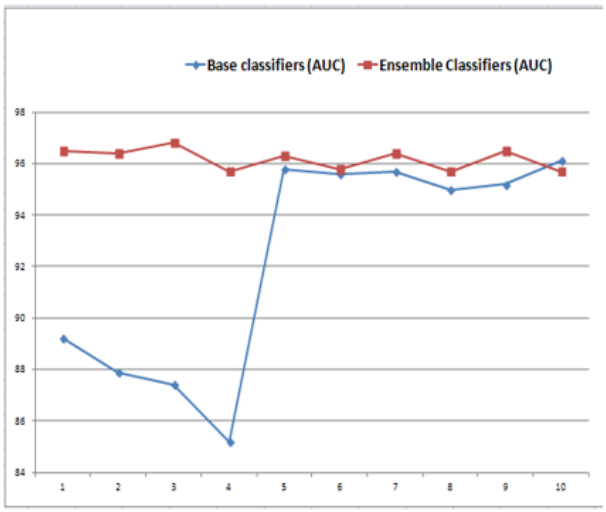

Figure 3. AUC Ratios (Base Classifiers & Ensemble Classifiers)

7. Conclusion

This study created a data set consisting of about half a million lines and 1893 classes to be used in software defect prediction. The data set includes metrics such as Cyclomatic Complexity, Comment Rate, Avg Depth, Biggest Depth and Biggest Subroutine. On the data set, 10 base classifier algorithms and 10 ensemble classifier algorithms were tested through leave-one-out method. For testing, we used a software using Weka library and developed in the Java programming language. The test results show that ensemble classifier algorithms increase ACC rate by 1.30% and AUC ratio by 3.87% compared to the base classifier algorithms. To repeat the studies, the data set is open to public access. Also, the newly developed software is freely available for researchers.

References

1. Song, O, Sheppard, M, Cartwright, M, Mair, C, Software Defect Association Mining and Defect Correction Effort Prediction, *IEEE Transactions on Software Engineering*, 2006, 32(2), 69-82.

2. McGregor, J, Sykes, D, A Practical Guide to Testing Object-Oriented Software, Addison-Wesley Professional, 2001.

3. Catal, C, Sevim, U, Diri, B, Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm, *Expert Systems with Application*, 2011, 38(3), 2347-2353.

4. Catal, C, Sevim, U, Diri, B, Software Fault Prediction of Unlabeled Program Modules, Proceedings of the World Congress on Engineering, London, UK, 2009, vol.I.

5. Patton, R, Software Testing, 2nd edition, Sams Publishing, 2006.

6. Craig, R, D, Jaskiel, S, P, Systematic Software Testing, Artech House, 2002.

7. Catal, C, Diri, B, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, *Information Sciences*, Elsevier, 2009, 179(8), 1040–1058.

8. Kaszycki, G, Using process metrics to enhance software fault prediction models, In Proceedings of Tenth International Symposium on Software Reliability Engineering, Boca Raton, Florida, Academic Press, 1999.

9. Xu, Z, Khoshgoftaar, T, M, Allen, E, B, Prediction of software faults using fuzzy nonlinear regression modeling, In Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000), Albuquerque, New Mexico, USA, 2000, 281-290.

10. Reformat, M, A fuzzy-based meta-model for reasoning about number of software defects, International Fuzzy Systems Association World Congress, IFSA 2003: Fuzzy Sets and Systems, Istanbul, Turkey, 2003, pp 644-651.

11. Menzies T, Di Stefano, J, How good is your blind spot sampling policy, Eighth IEEE International Symposium on High Assurance Systems Engineering, Tampa, Florida, USA, 2004, pp 129-138.

12. Cagatay, C, Diri, B, Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System, International Conference on Product Focused Software Process Improvement, Springer-Verlag, Berlin Heidelberg, 2007, pp 300–314.

13. D'Ambros, M, Lanza, M, and Robbes, R, An Extensive Comparison of Bug Prediction Approaches, 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa, 2010.

14. Mısırlı, A, T, Bener, A, B, Turhan, B, An industrial case study of classifier ensembles for locating software defects, *Software Quality Journal*, Springer Science, 2011, 19(3), 515-536.

15. Kaur, S, Kumar, D, Software Fault Prediction in Object Oriented Software Systems Using Density Based Clustering Approach, *International Journal of Research in Engineering and Technology (IJRET)*, 2012, 1(2).

16. Ozcift, A, Gulten, A, Classifier Ensemble Construction with Rotation Forest to Improve Medical Diagnosis Performance of Machine Learning Algorithms, *Computer Methods and Programs in Biomedicine*, 2011, 104(3), 443-451.

17. McCabe, T, J, A Complexity Measure, *IEEE Transactions on Software Engineering*, 1976, 2(4), 308-320.

18. Shepperd, M, A Critique of Cyclomatic Complexity as a Software Metric, *Software Engineering Journal*, 1988, 3(2), 30-36.

19. Sharma, R, Singh, P, Sharma, S, Deviation Causing Factors in a Code based on Environment of Analysis, *International Journal of Applied Information Systems (IJAIS)*, 2012, 12(9), 23-30.

20. Online: Dataset USP-1299 and Developed Software, https://github.com/KamilAkarsu/Voting, (accessed date 24.09.2018).