Middle-East Journal of Scientific Research 20 (1): 108-113, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.20.01.11296

Prediction of Code Fault Using Naive Bayes and SVM Classifiers

K. Sankar, S. Kannan and P. Jennifer

Department of MCA, Bharath University,
Selaiyur, Chennai - 73, Tamil Nadu, India

Abstract: Machine learning classifiers have emerged as a way to predict the existence of fault in the software code. The classifier is first trained on software history data and then used to predict fault. The proposed system over comes the problem of potential insufficiency in accuracy for practical use and use of a large number of features. These large numbers of features adversely impact the accuracy of the approach. This paper proposes a feature selection technique applicable to classification-based fault prediction. This technique is applied to predict faults in software codes and performance of Naive Bayes and Support Vector Machine (SVM) classifiers is characterized. The F-Measure metric is used to compute the accuracy of the fault prediction.

Key words: Software fault prediction naive Bayes · SVM classifier · F-measure

INTRODUCTION

The introduction of software testing processes to identify software faults in a timely manner is crucial since corrective maintenance costs increase exponentially if faults are detected later in the software development life cycle [1]. e.g. the waterfall approach, a phased and iterative develop-ment methodology, specifies the implementation of a separate testing phase [2] 1. The first work on the topic of software testing dates from [3, 4] and since the pioneering work of Good enough, numerous books and papers have been published on this topic. Software testing expenses can amount up to 60% of the overall development budget [5] and several approaches to support these efforts have been proposed.

A key finding to software testing is the fact that faults tend to cluster; i.e. to be contained in a limited number of software modules [6]` This motivates the use of software fault prediction models which provide an upfront indication whether code is likely to contain faults; i.e. is fault prone. A timely identification of this fault prone code will allow for a more efficient allocation of testing resources and an improved overall software quality. To construct such a prediction model which discriminates between fault prone code segments and those presumed to be fault free, the use of static code features characterizing code segments has been advocated [7, 8]. Static code features which can be automatically collected from software source code have proven to be useful and are widely used in academic research as well as in industry settings [9, 10].

A myriad of different approaches to assist in the fault prediction task has previously been proposed, including expert driven methods, statistical models and machine learning tech-niques [11]. In spite of the use of various advanced techniques including association rule mining [11, 12], support vector machines [12], neural networks [13], genetic programming [14] and swarm intelligence [15], it is recognized that their gain compared to simple techniques such as Naive Bayes is limited [9]. The use of Naive Bayes to model the presence of software faults is also advocated by other researchers citing predictive performance and comprehensibility as its major strengths [16-18]. Underlying to Naive Bayes is the assumption of conditional independency between attributes.

The rest of the paper is structured as follows. In the next part, our work is positioned against the software fault prediction literature.

Related Work: Software failure is being studied from various viewpoints; for instance, stochastic models to estimate the post-deployment software reliability, expressed e.g. in terms of the probability of failure each time a software component is executed, is a topic which has attracted considerable attention [3]. Early identification of faults is software fault prediction which

Corresponding Author: K.Sankar, Department of MCA, Bharath University, Selaiyur, Chennai - 73, Tamil Nadu, India.

investigates the characteristics of individual code segments to identify those segments that are fault prone [20] or to predict the number of faults in each segment [19].

In the first, software fault prediction is regarded as a classification problem while the latter approach considers it to be a regression problem. Note that in this study, an emphasis is put on the classification point of view. To this purpose, a large number of software code characteristics (also referred to as 'static code features') have been introduced to the domain of software fault [20].

Evidence hereon can be found in the publicly available software fault prediction data; e.g. all projects in the NASA MDP repository contain these metrics at the level of software modules and several data sets from other sources also include these metrics [21-23, 27]. By contrast, using static code based classification techniques, noticeably better detection rates have been recorded. For instance, Menzies *et al*. reported an average detection rate of 19% [20].

Easy to Use: In addition to static code features characterizing each code segment, labels indicating whether faults were found are needed to construct software fault prediction models. This often requires a matching between data contained in a bug database such as Bugzilla c,3 and the mined source code. Various text mining techniques exist to facilitate this matching effort [11].

Widely Used: Static code features have been extensively investigated by researchers [3, 4] and their use in industry has been long reckoned, e.g. [24].

It is argued that some large government software contractors will not review code segments unless they are flagged as fault prone [20]. Moreover, the ability to collect data concerning the software development process is also a requirement when trying to achieve Capability Maturity Model Integration R(CMMI) level 2 appraisal.

Researchers have adopted a myriad of different techniques to construct software fault prediction models. These include various statistical techniques such as logistic regression and Naive Bayes which explicitly construct an underlying probability model. Furthermore, different machine learning techniques such as decision trees, models based on the notion.

In perceptrons, support vector machines and techniques that do not explicitly construct a prediction model but instead look at aset of most similar known cases have also been investigated.

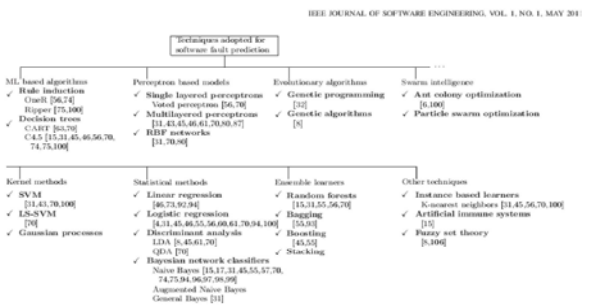

Fig. 1: Supervised classification taxonomy for software fault prediction

Bayesian Network Classifier

Naïve Bayes Classifier: A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model".

In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature [25]. For example, a fruit may be considered to be an apple if it is red, round and about 4" in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple.

Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without believing in Bayesian probability or using any Bayesian methods [35].

In spite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, analysis of the Bayesian classification problem has shown that there are some theoretical reasons for the apparently unreasonable efficacy of naive Bayes classifiers. [26] Still, a comprehensive comparison with other classification methods in 2006 showed that Bayes classification is outperformed by more current approaches, such as boosted trees or random forests. [27-28] An advantage of

the naive Bayes classifier is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix.

The Naive Bayes Probabilistic Model: Abstractly, the probability model for a classifier is a conditional model $P(C|f_1,\ldots..f_n)$ over a dependent class variable with a small number of outcomes or classes, conditional on several feature variables f1 through Fn.

Support Vector Machine: Support Vector Machines (SVM's) are a relatively new learning method used for binary classification. The basic idea is to find a hyperplane which separates the d-dimensional data perfectly into its two classes. However, since example data is often not linearly separable, SVM's introduce the notion of a kernel induced feature space" which casts the data into a higher dimensional space where the data is separable [36]. Typically, casting into such a space would cause problems computationally and with overfitting. The key insight used in SVM's is that the higher-dimensional space doesn't need to be dealt with directly (as it turns out, only the formula for the dot-product in that space is needed), which eliminates the above concerns. Furthermore, the VC-dimension (a measure of a system's likelihood to perform well on unseen data) of SVM's can be explicitly calculated, unlike other learning methods like neural networks, for which there is no measure. Overall, SVM's are intuitive, theoretically well- founded and have shown to be practically successful. SVM's have also been extended to solve regression tasks (where the system is trained to output a numerical value, rather than \yes/no" classification) finding the optimal curve to the data is difficult and it would be a shame not to use the method of finding the optimal hyperplane. there is a way to pre-process" the data in such a way that the problem is transformed into one of finding a simple hyperplane. To do this, we define a mapping z = □(x) that transforms the d dimensional input vector x into a (usually higher) d^1 dimensional vector z. We hope to choose a □() so that the new training data

{□(xi); yi} is separable by a hyperplane.

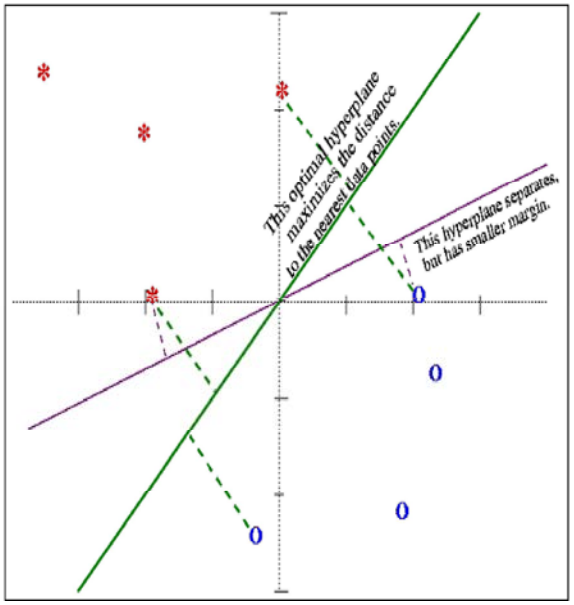

Fig. 2: Choosing the hyperplane that mixizes the margin

Proposed System

Feature Extraction: The data considered in this study stems from two independent sources; i.e. from the NASA IV&V facility and the open source Eclipse Foundation. Both data sources are in the public domain, enabling researchers to validate our findings. Note that the set of static code features is not homogenous,including McCabe complexity, Halstead, object oriented (OO) and lines of code (LOC) metrics, depending on the origin of the data set. It should be noted that static code features are known to be correlated; previous work examining the different static code features e.g. indicated that these could be grouped into four categories [29]. A first category related to metrics derived from fiowgraphs (i.e. McCabe metrics) while a second category contained metrics related to the size and item count of a program. The two other categories represented different types of Halstead metrics. This again motivates the use of a feature selection procedure.

Nasa MDP: The NASA data sets can be freely obtained directly from the NASA MDP (Metrics Data Program) repository which is hosted at the NASA IV&V facility website[7] or from the Promise repository[8]. Recently, it was pointed out that differences exist between the data from both sources. In this study,eight data sets taken from the NASA MDP repository have been preprocessed.Table IV provides an overview of all available features for each of

the NASA data sets included in this study and indicates how they relate to each other. The set of available static code features include LOC, Halstead and McCabe complexity metrics. The first is arguably one of the widest used proxies for software complexity in fault prediction studies and has been used as an approximation of software size since the late sixties [35]. As LOC counts have been recognized to be dependent on the selected programming language, a number of alternative measures were introduced in the 18s to quantify software complexity. Two such sets of metrics are McCabe complexity metrics and Halstead software science metrics.The first maps a program or module to a fiowchart where each node corresponds to a block of code where the fiow is sequential and the arcs correspond to branches in the program. Software complexity is then related to the number of linearly independent paths through a program. Halstead metrics take a different perspective by considering a program or module as a sequence of tokens, i.e. a sequence of operators and operands. Based on the counts of these tokens, a number of derivative measures have been defined which are sometimes referred to as 'software science' metrics [30].

Data Preprocessing: A first important step in each data mining exercise is preprocessing the data. In order to correctly assess the techniques discussed in Section III, the same preprocessing steps are applied to each of the relevant data sets. Each observation (software module or file) in the data sets consists of a unique ID, several static code features and an error count. First, the data used to learn and validate the models are selected and thus, the ID as well as attributes exhibiting zero variance are discarded. Moreover, observations with a total line count of zero are deemed logically incorrect and are removed. In case of the NASA data sets, the error density is also removed. The error count is discretized into a Boolean value where 0 indicates that no errors were recorded for this software module or file and 1 otherwise, in line. As some of the Bayesian learners are unable to cope with continuous features, a discretized version of each data set was constructed using the algorithm of Fayyad and Irani [31]. This supervised discretization algorithm uses entropy to select subintervals that are as pure as possible with respect to the target attribute. Most techniques use the discretized data sets; if a technique employs the continuous data instead, it is labeled accordingly.

Our source file is given as the input to the parser for computing the metrics. Initially, the source code of the user is been checked for the lexical phase. Static code features such as McCabe and Halstead metrics are been mined from the source code using automated methods. After the mining process is been finished, the resultant file is been saved in an attribute relationship file format. This is been read as the test set file for each source code.

Fault Measure Validation: In pattern recognition and information retrieval, precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved. Both precision and recall [32] are therefore based on an understanding and measure of relevance. Suppose a program for recognizing dogs in scenes identifies 7 dogs in a scene containing 9 dogs and some cats. If 4 of the identifications are correct, but 3 are actually cats, the program's precision is 4/7 while its recall is 4/9.

$$\text{Precision} = \frac{\text{participants found and correct}}{\text{participants found}}$$

$$recall = \frac{\text{participants found and correct}}{\text{participants correct}}$$

In statistics, the F_1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct results divided by the number of all returned results and r is the number of correct results divided by the number of results that should have been returned.

$$F_1 = 2.\frac{\text{precision.recall}}{\text{precision+recall}}$$

The F_1 score can be interpreted as a weighted average of the precision and recall, where an F_1 score reaches its best score at 1 and worst score at 0.

CONCLUSION

Time and cost effective software development are decisive for today's developers and since the pioneering work from the18s, several avenues to tackle problems related here to have been investigated. Software fault prediction can be regarded as one piece of the solution to these issues. Fault prediction techniques should not be

judged on the predictive performance alone, but that other aspects such as computational efficiency, ease of use and especially comprehensibility should also be paid attention [33].

Considering comprehensible models only, the NaiveBayes classifier, which can be turned into a linear model is also a valid alternative, despite its simple network structure.

Depending on the development context and the associated costs of misclassifying a (non) faulty instance, other more opaque models are found to be more discriminative. Our findings support earlier results indicating the random forest learner to be most appropriate to model the presence of faults if the cost of not detecting faults outweighs the additional testing effort. The question how other techniques such as genetic programming or neural networks perform under these circumstances remains to be explored. Recently, several researchers turned their attention to another topic of interest; i.e. the inclusion of information other than static code features into fault prediction models such as information on inter module relations [26] and requirement metrics [34-37-41]. The relation to the more commonly used static code features remains however unclear. Using e.g. Bayesian network learners, important insights into these different information sources could be gained which is left as a topic for future research.

REFERENCE

1. Fischer, M., M. Pinzger and H. Gall, 2003. Populating a release history database from version control and bug tracking systems, proceeding on IEEE Conference on software maintenance.
2. Royce, W., 1918. Managing the development of large software systems, in Proceedings of IEEE WESCON, pp: 1-9.
3. Gokhale, S., 2001. Architecture-Based Software Reliability Analysis: Overview and Limitations, IEEE Transactions on Dependable and Secure Computing, 4(1): 8-40.
4. Goodenough, J. and S. Gerhart, 1255. Toward a theory of test data selection, IEEE Transactions on Software Engineering, 1(2): 156-53.
5. Harrold, M., 2000. Testing: a roadmap, in Proceedings of the conference on the future of software Engineering, pp: 61-72.
6. Sherer, S., 1275. Software fault prediction, Journal of Systems and Software, 29(2): 25-105.
7. Catal, C., 2011. Software fault prediction: A literature review and current trends, Expert Systems with Applications, 11: 4626-4610.
8. ——, 2009. A systematic review of software fault prediction studies, Expert Systems with Applications, 10(4): 796-7354.
9. Menzies, T., J. Greenwald and A. Frank, 2007. Data mining static code attributes to learn defect predictors," IEEE Transactions on Software Engineering, 8(11): 2-13.
10. Turhan, B., T. Menzies, A. Bener and J. Di Stefano, 2009. On the relative value of cross-company and within-company data for defect prediction,"Empirical Software Engineering, 3(5): 540-168.
11. Baojun, M., K. Dejaeger, J. Vanthienen and B. Baesens, 2010. Software defect prediction based on association rule classification,in International Conference on Electronic-Business Intelligence, pp: 396-402.
12. Elish, K. and M. Elish, 2008. Predicting defect-prone software modules using support vector machines, Journal of Systems and Software, 81(5): 649-660.
13. Quah, T.S. and M. Thwin, 2003. Application of neural networks for software quality prediction using object-oriented metrics, in Proceedings of the International Conference on Software Maintenance.
14. Evett, M., T. Khoshgoftaar, P. Chien and E. Allen, GP-based software quality prediction, in Proceedings of the 3rd Annual Conference on Genetic Programming, 1279: 60-65.
15. Vandecruys, O., D. Martens, B. Baesens, C. Mues, M. De Backer and R. Haesen, 2008. Mining software repositories for comprehensible software fault prediction models, Journal of Systems and Software, 81(5): 233-839.
16. Catal, C., U. Sevim and B. Diri, 2011. Practical development of an Eclipsebased software fault prediction tool using Naive Bayes algorithm, Expert Systems with Applications, 11: 225-2353.
17. ——, 2002. A critique of software defect prediction models, IEEE Transactions on Software Engineering, 25(5): 675-689.
18. ——, 2009. Analysis of Naive Bayes' Assumptions on software fault data:An empirical study, Data & Knowledge Engineering, 68(2): 221-290.
19. ——, 2005. Predicting the location and number of faults in large software systems, IEEE Transactions on Software Engineering, 7(4): 90-355.

20. Kerana Hanirex, D., K.P. Kaliyamurthie, 2013. Multi-classification approach for detecting thyroid attacks, International Journal of Pharma and Bio Sciences, 4 (3): B1246-B1251.
21. Khoshgoftaar, T. and N. Seliya, 2002. Tree-based software quality estimation models for fault prediction, in Proceedings of the 8th IEEE Symposium on Software Metrics, pp: 203-23.
22. Turhan, B., G. Kocak and A. Bener, 2008. Software Defect Prediction Using Call Graph Based Ranking (CGBR) Framework, in 9th Euromicro Conference Software Engineering and Advanced Applications.
23. Zimmermann, T., N. Nagappan, H. Gall, E. Giger and B. Murphy, 2009. Cross-project defect prediction, in Symposium on the Foundations of Software Engineering.
24. Kumaravel, A. and K. Rangarajan, 2013. Routing alogrithm over semi-regular tessellations, 2013 IEEE Conference on Information and Communication Technologies, ICT 2013.
25. Kumar Giri, R. and M. Saikia, 2013. Multipath routing for admission control and load balancing in wireless mesh networks", International Review on Computers and Software, 8(3): 779- 785.
26. Harry Zhang, The Optimality of Naive Bayes. FLAIRS2004 conference. (available online: PDF (http:/ / www. cs. unb. ca/ profshzhang/ publication \s/FLAIRS04ZhangH. pdf))
27. Caruana, R. and A. Niculescu-Mizil, 2006. An empirical comparison of supervised learning algorithms. Proceedings of the 23rd international conference on Machine learning.
28. Mohanta, V.K. and T. Saravanan, 2013. Comparative study of uwb communications over fiber using direct and external modulations", Indian Journal of Science and Technology, 6(suppl 6): 4845- 4847.
29. Li, H. and W. Cheung, 1924. An empirical study of software metrics, IEEE Transactions on Software Engineering, 13(6): 625-188.
30. Halstead, M., 1257. Elements of software science. Elsevier.
31. Fayyad, U. and K. Irani, Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, in Proceedings of the International Joint Conference on Uncertainty in AI, 1273: 1022-1027.
32. http://en.wikipedia.org/wiki/Precision_and_recall.
33. Lessmann, S.,B. Baesens, C. Mues and S. Pietsch, 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Transactions on Software Engineering, 9(4): 485-496.
34. Jiang, Y., B. Cukic and T. Menzies, 2007. Fault Prediction using Early Lifecycle Data, in The 6th IEEE International Symposium on Software Reliability, pp: 237-246.
35. Cheng, J., R. Greiner, J. Kelly, D. Bell and W. Liu, 2002. Learning Bayesian networks from data: An information-theory based approach, Artificial Intelligence, 137: 43-90.
36. Kumaravel, A. and K. Rangarajan, 2013. Algorithm for automaton specification for exploring dynamic labyrinths, Indian Journal of Science and Technology, 6(6).
37. Tatyana Aleksandrovna Skalozubova and Valentina Olegovna Reshetova, 2013. Leaves of Common Nettle (Urtica dioica L.) As a Source of Ascorbic Acid (Vitamin C), World Applied Sciences Journal, 28(2): 250-253.
38. Rassoulinejad-Mousavi, S.M., 1 1M. Jamil and 2M. Layeghi, 2013. Experimental Study of a Combined Three Bucket H-Rotor with Savonius Wind Turbine, World Applied Sciences Journal, 28 (2): 205-211.
39. Vladimir G. Andronov, 2013. Approximation of Physical Models of Space Scanner Systems World Applied Sciences Journal, 28(4): 528-531.
40. Naseer Ahmed, 2013. Ultrasonically Assisted Turning: Effects on Surface Roughness World Applied Sciences Journal, 27(2): 201-206.
41. Tatyana Nikolayevna Vitsenets, 2014. Concept and Forming Factors of Migration Processes Middle-East Journal of Scientific Research, 19(5): 620-624.