EPJ Web of Conferences **214**, 05007 (2019) https://doi.org/10.1051/epjconf/201921405007
CHEP 2018

Data Mining Techniques for Software Quality Prediction in Open Source Software:

An Initial Assessment

Marco Canaparo[1,*] and *Elisabetta* Ronchieri[1,**]

[1]INFN-CNAF Viale Berti Pichat 6/2 40126 Bologna

Abstract. Software quality monitoring and analysis are among the most productive topics in software engineering research. Their results may be effectively employed by engineers during software development life cycle. Open source software constitutes a valid test case for the assessment of software characteristics. The data mining approach has been proposed in literature to extract software characteristics from software engineering data.
This paper aims at comparing diverse data mining techniques (e.g., derived from machine learning) for developing effective software quality prediction models. To achieve this goal, we tackled various issues, such as the collection of software metrics from open source repositories, the assessment of prediction models to detect software issues and the adoption of statistical methods to evaluate data mining techniques. The results of this study aspire to identify the data mining techniques that perform better amongst all the ones used in this paper for software quality prediction models.

1 Introduction

The software used in scientific environment (e.g. the HEP software) is a rich mixture of in-house software and software taken from the large open source community [1]. Computer scientists are therefore striving to produce and employ high quality software that, at the same time, has been increasing in size and complexity. In order to produce high quality software and save effort, scientists need to know which software modules are defective [2].

Data mining is the process of discovering interesting patterns and knowledge from large amounts of data [3]. Figure 1 shows the transformation from static software engineering data to active data, performed by data mining. In regards to software engineering data, there are two important types of data sources: the former is the revision control systems (such as CVS, Subversion and Git) that manage the ongoing status of development, the latter is the defects tracking software (such as BugZilla and JIRA) [4]. The aforementioned data constitute the input of one or more data mining techniques (such as Random Forest, Bagging and Support Vector Machine). The output of these techniques help software engineers to mine patterns and detect violation of patterns, which are likely to be defects. Through data mining, data are converted into knowledge that can help in conducting the most common software engineering tasks: programming, defect detection, testing and maintenance [5].

*e-mail: marco.canaparo@cnaf.infn.it
**e-mail: elisabetta.ronchieri@cnaf.infn.it

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

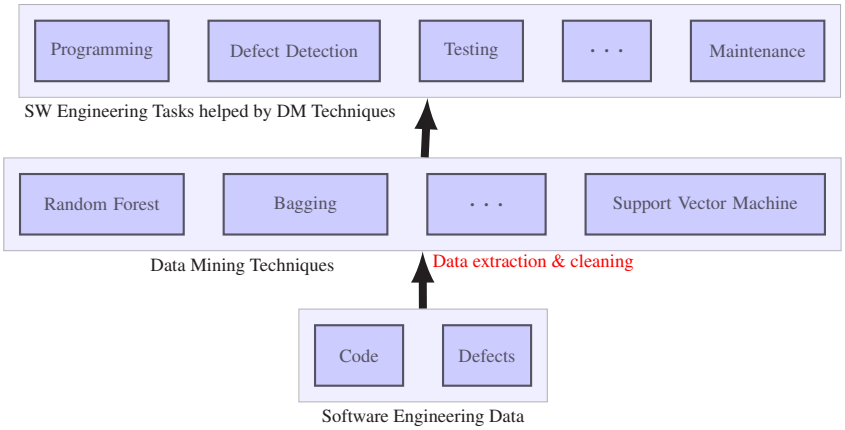

Figure 1. Data Mining and Software Engineering Data

In literature, there are many different studies that deal with software quality prediction and data mining techniques. However, to the best of our knowledge, there is no a comprehensive study that explains the practical aspects of software analytics models [6]. This study aims at providing an initial comparative performance analysis of different data mining techniques for software quality prediction through a well-documented methodology. Due to the amount of data, this paper provides a subset of results, whose discussion is going to be published in a forthcoming paper.

The remainder of this paper is structured as follows. Section 2 summarizes our research methodology; section 3 describes the study setup; section 4 provides some of the collected results with a brief discussion; finally section 5 draws our conclusions.

2 Research Methodology

Our approach is composed of two steps. In the first step, we have conducted a research in the field of data mining for software engineering issues and, more in detail, about software quality prediction to identify defect-prone software modules. In the second step, we have attempted to reproduce and expand previous studies on data mining techniques comparison by selecting a subset of software metrics, online datasets, data mining techniques, free data mining tools and packages, and performance criteria.

We collected the most used data mining techniques and metrics by leveraging existing literature.

Support Vector Machine (e.g. SMO): is a supervised techniques that searches for the optimal hyperplane to separate training data. The hyperplane found is intuitive: it is the one which is maximally distant from the two classes of labelled points located in each side [7, 8].

Decision Tree (e.g. J48): is a flow-chart like tree structure. It is composed of: nodes which represent a test on a attribute value; branches which show the outcome of the tests; leaves, that indicate the resulting classes [3].

Naive Bayes: relies on the Bayesian rule of conditional probability. It assumes that all the attributes are independent and analyses each of them individually [9].

Ensemble Classifier (e.g. Random Forest): consists of training multiple classifiers and then combining their predictions [10]. This technique leads to a generalized improvement of the

ability of each classifier [11]. According to the way the component classifiers are trained, parallel or sequential, we can distinguish two different categories of ensemble. Bagging [12] and Random Forest [13] are both parallel classifiers. Bagging creates multiple version of the classifier by replicating the learning set in parallel from the original on and the final decision is made by majority voting strategy. Random Forest adopts a combination of tree predictors, each depending on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Adaboost [14] is an example of a sequential classifier since each classifier of this technique is applied sequentially on the training samples misclassified by the previous one.

Deep Learning: is applied to feature hierarchy where features of higher levels are formed by the composition of lower level ones. Deep learning techniques leverage learning intermediate representations that can be shared across tasks and, as a consequence, they can exploit unsupervised data and data from similar tasks to improve performance on problems characterised by scarcity of labelled data [15–17].

As concerns metrics we collected all the metrics used in literature over time, some of them are:

McCabe (e.g. Cyclomatic Complexity, Essential Complexity): is used to evaluate the complexity of a software program. It is derived from a flow graph and is mathematically computed using graph theory. Basically, it is determined by counting the number of decision statements in a program [18, 19].

Halstead (e.g. Base Measures, Derived Measures): is used to measure some characteristics of a program module - such as the "Length", the "Potential Volume", "Difficulty", the "Programming Time" - by employing some basic metrics like number of unique operators, number of unique operands, total occurrences of operators, total occurrences of operands [20, 21].

Size (e.g. Lines of Code, Comment Lines of Code): the Lines of Code (LOC) is used to measure a software module and the accumulated LOC of all the modules for measuring a program [22].

Chidamber and Kemerer (e.g. Number of Children, Depth of Inheritance): is used for object-oriented programs and is the most popular for performing software analysis and prediction. It has been adopted by many software tool vendors and computer scientists [23, 24]. Some metrics of the suite are: Weighted Method Per Class, which measures the number of methods which is in each class; Depth of Inheritance Tree, which measures the distance of the longest path from a class to the root in the inheritance tree; Number Of Children, which measures the number of classes that are direct descendants of each class.

3 Study Setup

Unlike previous literature, we also consider Deep Learning techniques [15, 25], which have gained importance in recent years. In the past, their employment have mainly been on Computer Vision, Natural Language Processing and Speech Recognition [26].

In the past, authors have focused their attention mainly on the NASA Defect Dataset [27–32] that can be found in online repositories [33, 34]. On the other hand, we have decided to widen our scope by including some datasets related to open source projects such as Eclipse [35], Android and Elastic Search [36]. Table 1 shows a summary of the most important characteristics of these datasets in terms of number of projects, metrics, modules and percentage of defective modules per projects, reporting their range whenever possible. We have collected the performance criteria used by previous literature. All the definitions below (see Eqs. 1, 2, 3, 4) are based on the confusion matrix shown in Table 2. Accuracy (see Eq. 1) is the percentage of modules correctly classified as either faulty or non-faulty. Precision (see Eq. 2) is the percentage of modules classified as faulty that are actually faulty. Recall (see Eq. 3) or

Table 1. Summary of the datasets employed

Repository	#Projects	#Metrics	#Modules	%Defective Modules
NASA Defect Datasets	11	[30,41]	[101, 5589]	[0.41, 48.80]%
Eclipse Datasets	5	17 each	[324, 1863]	[9.26, 39.81]%
Android Datasets	6	102 each	[74, 124]	[0, 27.02]%
Elastic Search Datasets	12	102 each	[1860, 7263]	[0.16, 11.47]%

Table 2. Confusion Matrix

		Predicted by model +	**Predicted by model** -
Actual value	+	True Positive (TP)	False Negative (FN)
	-	False Positive (FP)	True Negative (TN)

Completeness is the percentage of faulty modules that are predicted as faulty. Mean Absolute Error determines how close the values of predicted and actual fault rate differ. F-measure (see Eq. 4) is a combined measure of recall and precision, the higher value of this indicator the better is the quality of the learning method for software prediction.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1) \qquad Precision = \frac{TP}{FP + TP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3) \qquad F - measure = \frac{2 \times Recall \times Precision}{Precision + Recall} \quad (4)$$

In this paper, we have provided results for the only accuracy performance indicator. A more extensive analysis is going to be included in a forthcoming paper.

Figure 2 describes our solution workflow. We have selected some online datasets (i.e., NASA, Eclipse, Android and Elastic Search), including all the metrics contained in those repositories. We have performed some cleaning operations when needed, replacing missing values with the mean of the other values related to the same metric [27]. The obtained data have been used as input of many data mining techniques (both supervised and unsupervised) by employing three different free open source tools: Weka [37], scikit learn [38] and R [39]. We have collected the output of all the executions of the algorithms and we have compared their values according to the performance indicators.

4 Initial Assessment

The histograms show the average accuracy computed for the considered datasets of the data mining techniques taken into account. Each value has been obtained by computing the accuracy of each dataset. The compared data mining techniques are: Naive Bayes, Multi Layer Perceptor, Support Vector Machine, AdaBoost, Bagging, Random Forest, J48, K-Nearest Neighbor, RBF Neural Network, DeepLearning4j.

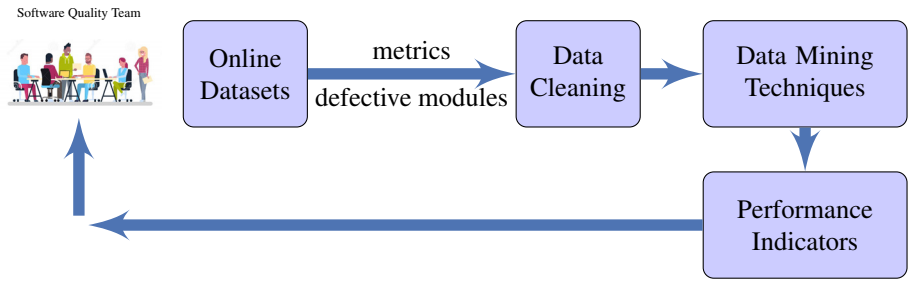

Figure 2. Solution Workflow

Figures 3 shows the values of accuracy obtained by exploiting the NASA dataset (on the left) and the Eclipse dataset (on the right). Bagging and Random Forest are the techniques that have performed the best for the first dataset, while the SVM technique has performed better in the other case. Figure 4 shows the values of accuracy obtained by exploiting the

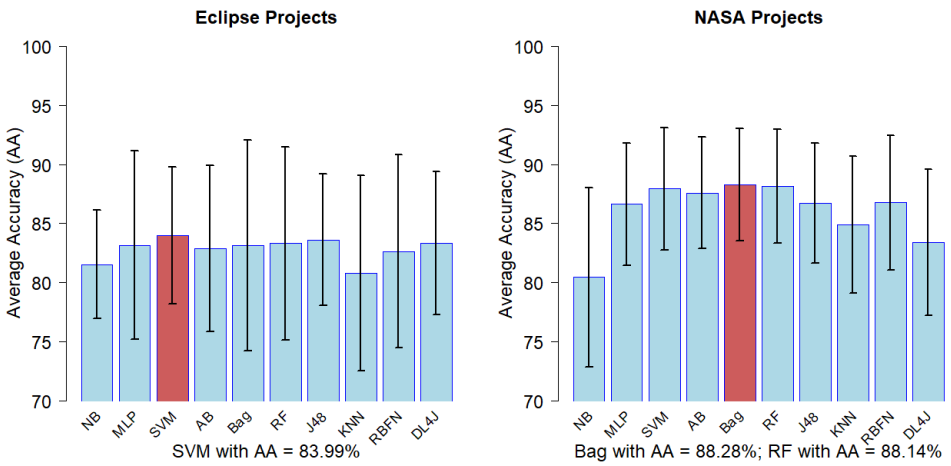

Figure 3. Average Accuracy for the NASA and Eclipse datasets

Android dataset (on the left) and Elastic Search dataset (on the right). The techniques that have performed the best are Random Forest and Bagging. In conclusion, we have noticed that in 3 cases out of 4 the techniques that have got the best score are the ones belonging to the ensemble learning category. This is consistent with literature, i.e. ensemble learning algorithms, by integrating more classifiers to build a classification model, improve defect prediction. Interestingly, deep learning techniques have obtained good results, but never the best score.

5 Conclusion

In this study, we have shown an initial comparison of data mining techniques in the context of software defect prediction. To achieve this goal, we have leveraged existing literature to

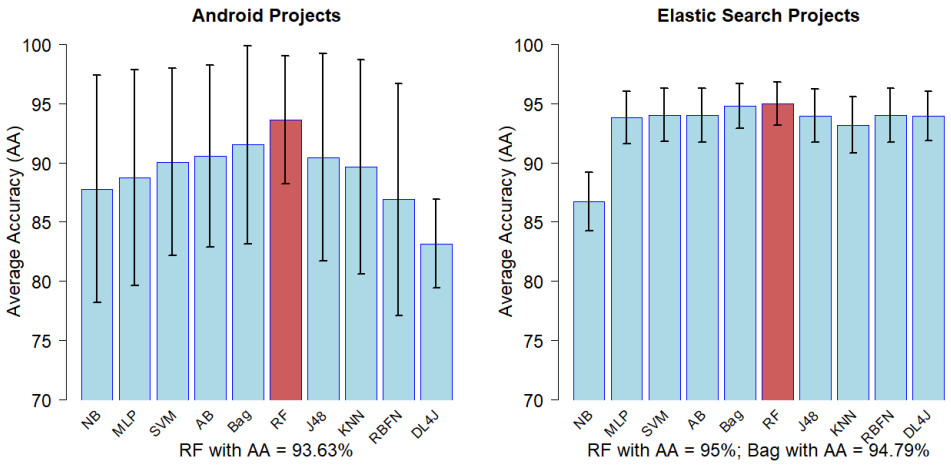

Figure 4. Average Accuracy for the Android and ElasticSearch datasets

collect online dataset, used techniques and performance criteria. Unlike previous studies, we have paid more attention to dataset related to open source projects and to Deep Learning techniques. By analysing the results, we can conclude that Bagging and Random Forest have achieved the best average accuracy over all the datasets. We have also shown that data mining can constitute a valid helping hand in determining and predicting software quality, and can be used together with statistical analysis.

Currently, we are experimenting using the same techniques on software used in HEP.

This research was supported by INFN CNAF.

Appendix

A Glossary

Software Quality, according to IEEE, is the degree to which a system meets specified requirements or customer or user's needs or expectations. According to ISO, quality is the degree to which a set of inherent characteristics fulfils requirements.
Data Mining is the process of discovering interesting patterns and knowledge from large amount of data contained in datasets.
Defect is an imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be repaired or replaced.

B Datasets

Table 3 shows details of the datasets: AR1 and AR6 [34]; CM1, KC3, KC4, MC2, MW1, PC1, PC2, PC3 and PC4 [33]; Eclipse [35] and Android [36].

References

[1] T. Wenaus, J. Phys.: Conf. Ser. **898**, 1 (2017)

Table 3. Datasets

Projects	#Metrics	#Modules	%Defective Modules	Projects	#Metrics	#Modules	%Defective Modules
AR1	30	121	7.44%	EclipseJDTCore	17	997	20.66%
AR6	30	101	14.85%	EclipsePDE	17	1497	13.96%
CM1	41	505	9.50%	Equinox	17	324	39.81%
KC3	41	458	9.39%	Lucene	17	691	9.26%
KC4	41	125	48.80%	Mylyn	17	1863	13.20%
MC2	41	161	32.30%	Android_2013_1	102	74	27.02%
MW1	41	403	7.68%	Android_2013_2	102	99	17.17%
PC1	41	1107	6.86%	Android_2013_3	102	110	1.81%
PC2	41	5589	0.41%	Android_2014_1	102	116	6.03%
PC3	41	1563	10.24%	Android_2014_2	102	119	1.68%
PC4	41	1458	12.21%	Android_2015_1	102	124	0%

[2] A. Kaur, I. Kaur, Journal of King Saud University - Computer and Information Sciences **30**, 2 (2016)

[3] J. Han, M. Kamber, *Data Mining Concepts and Techniques* (Morgan Kauffman, 2006)

[4] Q. Taylor, C. Giraud-Carrier, International Journal of Data Analysis Techniques and Strategies **2**, 243 (2010)

[5] N.S. Ali, V. Pawar, Int. J. Adv. Res. Comput. Sci. **4**, 172 (2013)

[6] H.K. Dam, T. Tran, A. Ghose, *Explainable Software Analytics*, in *40th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)* (ACM, New York, NY, USA, 2018), pp. 53–56

[7] V. Vapnik, *The Nature of Statistical Learning Theory*, 2nd edn. (2000)

[8] C. Campbell, Y. Ying, *Learning with Support Vector Machine* (Morgan & Claypool Publishers, 2011)

[9] A. McCallum, K. Nigam, *A comparison of Event Models for Naives Bayes Text Classification*, in *Learning for Text Categorization: Papers from the 1998 AAAI Workshop* (1998), pp. 41–48, `http://www.kamalnigam.com/papers/multinomial-aaaiws98.pdf`

[10] T.G. Dietterich, *"Ensemble Learning" The Handbook of Brain Theory and Neural Networks* (2002), Vol. 2, pp. 405–408, 2nd edn.

[11] Q. He, B. Shen, Y. Chen, *Software Defect Prediction Using Semi-Supervised Learning with Change Burst Information*, in *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)* (2016), pp. 113–122

[12] L. Breiman, Tech. rep., Department of Statistics, University of California, Berkley, California 94720 (1994), `https://www.stat.berkeley.edu/~breiman/bagging.pdf`

[13] L. Breiman, Mach. Learn. **45**, 5 (2001)

[14] Y. Freund, R.E. Schapire, J. Comput. Syst. Sci. **55**, 119 (1997)

[15] Y. Bengio, Found. Trends Mach. Learn. **2**, 1 (2009)

[16] R. Raina, A. Battle, H. Lee, B. Packer, A.Y. Ng, *Self-taught Learning: Transfer Learning from Unlabeled Data*, in *24th international conference on Machine Learning* (2007), pp. 759–766

[17] R. Collobert, J. Weston, *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*, in *25th International Conference on Machine Learning* (2008)

[18] T. McCabe, IEEE Trans. Softw. Eng. **Se-2**, 308 (1976)

[19] T.J. McCabe, C. Butler, Artif. Intell. Language Process. **32**, 1415 (1989)

[20] V.Y. Shen, S.D. Conte, Tech. rep., Department of Computer Science - Purdue University (1981)
[21] M.H. Halstead, *Elements of Software Science (Operating and programming systems series)* (Elsevier Science Inc., New York, NY, USA, 1977)
[22] W. Li, S. Henry, *Maintenance Metrics for the Object Oriented Paradigm*, in *First International Software Metrics Symposium* (IEEE, 1993)
[23] S.R. Chidamber, C.F. Kemerer, IEEE Trans. Softw. Eng. **20**, 476 (1994)
[24] C. Catal, B. Diri, Expert Syst. Appl. **36**, 7346 (2009)
[25] Y. LeCun, Y. Bengio, G. Hinton, Nature **521**, 436–444 (2015)
[26] J. Deshmukh, K.M. Annervaz, S. Podder, *Towards Accurate Duplicate Bug Retrieval Using Deep Learning Techniques*, in *IEEE International Conference on Software Maintenance and Evolution (ICSME)* (IEEE, 2017), pp. 115–124
[27] M. Shepperd, Q. Song, Z. Sun, C. Mair, IEEE Trans. Softw. Eng. **39**, 1208 (2013)
[28] Z. Zhang, X. Jing, T. Wang, Autom. Softw. Eng. **24**, 47 (2017)
[29] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, IEEE Trans. Softw. Eng. **37**, 356 (2011)
[30] D. Gray, D. Bowes, N. Davey, *The miuse of the NASA metrics data program data sets for automated software defect prediction*, in *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE)* (2011)
[31] Y. Zhou, H. Leung, IEEE Trans. Softw. Eng. **32**, 771 (2006)
[32] S. Aleem, L.F. Capretz, F. Ahmed, Int. J. Softw. Eng. Knowl. Eng. **6**, 11 (2015)
[33] *Nasa Defect Dataset*, `https://github.com/klainfo/NASADefectDataset`
[34] *Machine Learning & Data Mining Algorithms*, `http://tunedit.org/repo/PROMISE/DefectPrediction`
[35] *Bug prediction dataset, Evaluate your bug prediction approach on our benchmark*, `http://bug.inf.usi.ch`
[36] Z. Tóth, P. Gyimesi, R. Ferenc, *Bug Database of GitHub Projects - A Public Bug Database of GitHub Projects and its Application in Bug Prediction*, in *Computational Science and Its Applications – ICCSA 2016. Lecture Notes in Computer Science*, edited by O. Gervasi et al. (Springer, Cham, 2016), Vol. 9789, `http://www.inf.u-szeged.hu/~ferenc/papers/GitHubBugDataSet/`
[37] *Weka 3: Data Mining Software in Java*, `https://www.cs.waikato.ac.nz/ml/weka/`
[38] *scikit-learn, Machine Learning in Python*, `http://scikit-learn.org/stable/`
[39] *The R Project for Statistical Computing*, `https://www.r-project.org/`