Contributing features-based schemes for software defect prediction

Aftab Ali[1], Mamun Abu-Tair[1], Joost Noppen[2], Sally McClean[1], Zhiwei Lin[1], Ian McChesney[1]

[1]School of computing, Ulster University, United Kingdom, BT37 0QB
{a.ali, m.abu-tair, si.mcclean, z.lin, ir.mcchesney}@ulster.ac.uk
[2]Applied Research, BT, Ipswich, United Kingdom
johannes.noppen@bt.com

Abstract. Automated defect prediction of large and complex software systems is a challenging task. However, by utilising correlated quality metrics, a defect prediction model can be devised to automatically predict the defects in a software system. The robustness and accuracy of a prediction model is highly dependent on the selection of contributing and non-contributing features. Hence, in this regard, the contribution of this paper is twofold, first it separates those features which are contributing towards the development of a defect in a software component from those which are non-contributing features. Secondly, a logistic regression and Ensemble Bagged Trees-based prediction model are applied on the contributing features for accurately predicting a defect in a software component. The proposed models are compared with the most recent scheme in the literature in terms of accuracy and area under the curve (AUC). It is evident from the results and analysis that the performance of the proposed prediction models outperforms the schemes in the literature.

1 Introduction

According to a study at Cambridge University, the cost of software defects has increased to $312 billion per annum globally [1]. The reason behind the cost increase is that developers spent most of their time on finding and fixing defects. The developer's ultimate goal is to release defect free software to the end user. Unfortunately, software defects are inevitable; for example the US Department of Defence is spending over four billion dollars for software failures per year [2].

Techniques of software testing are mostly used to reduce defects and ensure high quality systems [3, 4] [5]. However, such testing requires some tedious and exhaustive test cases to be executed, and this makes the process quite expensive, especially since the defect removal effectiveness of traditional testing activities can be very low [6]. According to NIST [7] and [8], finding defects early in the development process greatly lowers the average cost of defects. Moreover, inadequate software testing is causing $3.3 billion to U.S. software developers and users in the financial services sector. According to a study software defects cost U.S. industry $60 billion a year [9]. This supports our case that accurate and automatic software defect prediction should be an important part of the software development process.

In [10], the authors used an approach similar to our work for finding contributing features for defect prediction using logistic regression and considering object-oriented metrics. The authors performed experiments on the open source web and e-mail suite Mozilla version 1.7. The authors examined a total of eight performance metrics (i.e. LCOM (Lack of Cohesion on Methods), DIT (Depth of Inheritance Tree), RFC (Response For a Class), WMC (Weighted Methods per Class), NOC (Number Of Children), CBO (Coupling Between Object classes), LCOMN (Lack of Cohesion on Methods allowing Negative value), and LOC (Lines Of Code)), where only one metric is found to be non-contributing. In another paper, Malhotra [11] reviewed such approaches and found that logistic regression performed poorly, in terms of prediction, when compared with machine learning approaches. However, logistic regression provides good explainability by identifying which features are important in predicting bugs and quantifying this.

Our current approach therefore combines this explanatory capability of logistic regression with the diversity provided by a tree-based approach to improve prediction performance. In this paper, we propose contributing features-based logistic regression (CFLR) and Ensemble Bagged Trees-based (EBT) models to predict software defects. The proposed approaches first find those attributes (features) which are contributing or significant in terms of defect prediction by using logistic regression. Once those features are identified in the next step, CFLR and EBT prediction models are applied to the selected attributes to predict those instances which contains defects. The proposed schemes have a number of differences from [10]. The current approach examines eight datasets from different projects with different features for defect prediction. Once the most appropriate features are selected by using the CFLR scheme, then these features are passed on to the EBT algorithm for defect prediction and analysis

The rest of the paper is organised as follows: section 2 presents defect prediction related work while section 3 elaborates the proposed contributing features-based logistic regression scheme. Experimental setup, results and discussions are covered in section 4.

2 Related work

Many machine learning algorithms have been used in the literature to solve the software defect prediction problem. A systematic review of Machine Learning based schemes for software defect predictions is presented in [11]. A random forest-based defect prediction scheme is presented in [12], where the authors compare different schemes in terms of accuracy. An artificial immune system is used in [13], to predict software defects in five NASA defect prediction datasets.

Osman et al [14] studied the impact of feature selection on predicting bugs in software systems. More specifically, they considered the impact of two different feature selection methods: correlation-based feature selection (CFS) and wrapper feature selection methods. The paper considers five different bug prediction models: Linear Regression, Random Forest, K-Nearest Neighbour, Multilayer Perceptron, and Support Vector Machine. The results show that by using a proper feature selection method, the accuracy of the bug prediction models could be increased by 33%.

In [15], the authors proposed semi-supervised deep fuzzy C-mean (DFCM) clustering for software fault prediction. The proposed solution is compared with four different bug prediction models including Class Mass Normalization (CMN) methods, Low-Density Separation (LDS), Support Vector Machine (SVM) and Expectation-Maximization (EM-SEM). The paper shows that the proposed method outperforms the four selected bug prediction models in term of probability of detection using both AUC and F-Measures.

Arasteh [16] proposed a fault prediction method based on combining Neural Network and the Naïve Bayes algorithms. The author compares the proposed approach with three different bug prediction models including Support Vector Machine (SVM) , Artificial Neural Network (ANN) and Naïve Bayes.

In this paper we proposed a logistic regression-based contributing feature selection process using P-values. While utilising the advantage of logistic regression simplicity and expandability, it is also used for prediction of bugs. Moreover, to add some more diversity, the ensemble bagged trees are used alongside the logistic regression contributing feature selection process.

3 Contributing features-based Logistic Regression

Pre-processing of data to extract useful information (also referred as feature extraction) is one of the most crucial steps in classification and prediction tasks [17]. The accuracy of classification and prediction is strongly dependent on the extraction of relevant features from the raw data [18]. For the feature selection process we use logistic regression as given in the following equation.

$$\log(\frac{\pi}{1-\pi}) = a + b_1x_1 + \cdots + b_nx_n$$

Where $\frac{\pi}{1-\pi}$ is the log odds; a, b_1, … b_n are the constants which we estimate (learn) from the repository data; and x_1, … x_n are the features which we extract from the repository data e.g. loc, cyclomatic complexity etc.

Logistic regression is used for predictive analysis of data to identify the relationship between a dependent binary variable and one or more independent variables by estimating the probabilities using a logistic function [19]. The reason for choosing logistic regression is that it is quite extensible and explainable [20, 21], and it is fairly simple in terms of implementation. In logistic regression for every feature the P-value is calculated, and the features are then ranked based on the P-value. A P-value less than 0.05 indicates that a particular feature is significant. The greater the P-value the more the feature's significance decreases, while the lower the P-value the more its significance increases.

3.1 A case study of contributing feature selection

In our experiments, we used eight different datasets [22] for results and analysis. However, as a case study, only one dataset (i.e. CM1) is selected to explain and dry run the feature selection process. Below the process of contributing and non-contributing feature selection is explained with reference to the CM1 dataset.

Table 1. Description of data attributes for CM1

S.No	Attribute	Description	S.No	Attribute	Description
1.	loc	line count of code	2.	v(g)	cyclomatic complexity
3.	ev(g)	essential complexity	4.	iv(g)	design complexity
5.	n	Halstead total operators + operands	6.	v	Halstead "volume"
7.	I	numeric % Halstead "program length"	8.	d	Numeric % Halstead "difficulty"
9.	I	Halstead "intelligence"	10.	e	Halstead "effort"
11.	b	Halstead	12.	t	Halstead's time estimator
13.	lOCode	Halstead's line count	14.	lOComment	Halstead's count of lines of comments
15.	lOBlank	Halstead's count of blank lines	16.	locCodeAndComment	
17	uniq_Op	unique operators	18.	uniq_Opnd	unique operands
19.	total_Op	total operators	20.	total_Opnd	total operands
21.	branchCount	% of the flow graph	22.	defects	(false, true) module has/has not one or more reported defects

The dataset (CM1) used in the case study is a NASA spacecraft instrument written in C language. The dataset consists of a total of 498 instances, where 49 records are for the modules which contains bugs, while the rest are normal records without bugs. The data is extracted by using McCabe [23] and Halstead [24] metric extractors from the source code. A short description of the features in the dataset is given in Table 1. These features were defined to characterize code features that are associated with software quality. The McCabe and Halstead measures are calculated based on modules (functions), where a module is the smallest unit of functionality.

Similarly, for contributing feature selection, we applied a 0.08 threshold to find contributing features in the data. Contributing features are those features, which have comparatively high potential to participate in the prediction process. Table 2 shows the selected contributing features along with the corresponding P-value. In the original dataset we have 22 features as presented in Table 1, while after selecting the contributing features we have only 7 features as presented in Table 2.

Table 2. Selected Features by applying the threshold

Attribute	**P-Value**
v(g)	0.0784
iv(g)	0.0334
total_Op	0.0189
uniq_Opnd	0.0080
i	0.0033
uniq_Op	0.0016
lOComment	0.0001

4 The Prediction Models

Once the contributing features are selected (using LR), then first (step 1) a logistic regression-based prediction model is applied to predict buggy and non-buggy modules in the input dataset. A simple diagrammatic representation of the proposed contributing features-based logistic regression (CFLR) model is presented in Figure 1, where $X_1 \ldots X_n$ are the input attributes while $W_1 \ldots W_n$ are the corresponding weights for each attribute.

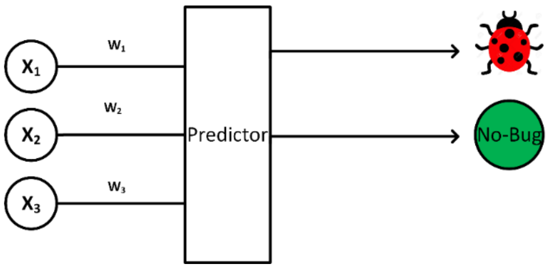

Fig. 1. Logistic Regression-based classifier

By using a tree-based approach, Step 2, we can incorporate diversity into the logistic regression parameterisation, thus allowing us to recognise and model different types of bugs with potentially different significant features and models. Ensemble Bagged Trees-based (EBT) scheme is applied on the selected contributing features (i.e. the features are first selected by using logistic regression and then the EBT scheme is applied on those features. The EBT model is a combination of several decision tree classifiers to produce better predictive performance than a single decision tree classifier. The basic principle is to create several subsets from training data chosen randomly with replacement, where, each collection of subset data is used to train the corresponding decision trees and end up with an ensemble of different models. The average of all the predictions from different trees is used resulting in a more robust model compared to a single decision tree.

In Figure 2, a generic EBT model is given, where the training data is divided in to multiple subsets (i.e. Data$_1$ … Data$_M$), and the leaners (i.e. M different trees) are trained with the different subtests of the data.

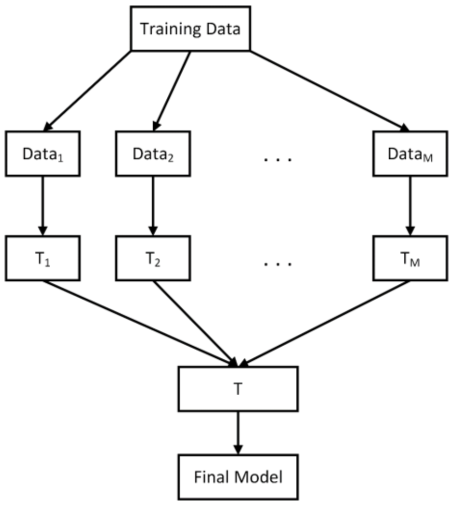

Fig. 2. A generic EBT Model

5 Experiments and Analysis

The performance of logistic regression and EBT models are evaluated using 10-fold cross validation to evaluate the predictive model and partition the original dataset into a training set to train the model and a testing set to evaluate the model. In 10-fold cross-validation, the data is divided into 10 equally sized segments, called folds, where one-fold is kept for validation, while the other 9 folds are utilized for training the model. The trained model is then applied to predict the target variable in the testing data. This process is repeated 10 times, with the performance of each model in predicting the set being hold. The performance is measured by calculating metrics such as accuracy, receiver operating characteristic (ROC), and area under curve (AUC).

The advantage of this approach is that the input dataset is used for both training and testing, and each observation is used for testing exactly once.

5.1 Results and Discussions

The ROC is a measure of predictor performance. The concern here is the area under the ROC curve which is AUC. In our experiments we achieved significantly higher values of accuracy and AUC compared to previous work, which indicates that the predictor does a good job in discriminating between the modules containing bugs and the ones without a bug which comprise our target variable.

Table 3. Accuracy and AUC comparison

	Before Feature Selection			After Feature Selection		
Dataset	**No. of Attributes**	**Accu-racy (%)**	**AUC (%)**	**No. of At-tributes**	**Accuracy (%)**	**AUC (%)**
CM1	22	86.9	57	07	90.4	80
JM1	22	81.4	71	17	82.04	71.4
AR1	30	86.7	51.2	14	87.6	67.3
KC1	22	85.6	79.5	14	86.5	80.4
KC2	22	83.12	82.2	06	85.5	83.3
PC1	22	92.43	80.9	07	93.5	81.9
PC3	38	89.6	82	12	89.6	80
PC4	38	91.56	91.6	10	91.69	92.1

Moreover, we performed the experiments on the original dataset (i.e. containing all attributes), as well as on the selected contributing features dataset (i.e. containing only reduced set of attributes). We found that generally the selected contributing feature analysis performed better in terms of accuracy and AUC as presented in Table 3. However, in many cases the accuracy and AUC are not improved significantly, but achieving slightly better accuracy and AUC with a reduced number of attributes is a good performance indication.

5.2 Comparative analysis

The proposed contributing features-based logistic regression (CFLR) and Ensemble Bagged Trees-based (EBT) schemes are compared with Huda et al

[3] and Arar et al [4] in terms of accuracy and AUC as can be seen in Table 4. In [3] the authors present SVM and ANN filters and wrappers for significant attribute selection to predict software defects. Similarly, the scheme in [4] uses ANN and Artificial Bee Colony (ABC) algorithms with a cost-sensitive function to deal with the imbalanced data for defect prediction.

Table 4 Comparative analysis

	Logistic Regression (selected features)		**Ensemble Bagged Trees**		**Huda et al scheme [3]**		**Arar et al Scheme [4]**	
Dataset	**Accuracy (%)**	**AUC (%)**	**Accuracy (%)**	**AUC (%)**	**Accuracy (%)**	**AUC (%)**	**Accuracy (%)**	**AUC (%)**
CM1	**90.4**	**80**	**90.8**	73	**82.94**	**56.8**	**68**	**77**
JM1	82.04	71.4	82.3	73	81.23	70.1	61	71
AR1	**87.6**	**69.0**	**91**	**79**	**55.9**	**52.7**	NA	NA
KC1	86.5	81	87.3	82	86.1	69.1	69	80
KC2	85.5	84	79	84	84.7	68.8	79	85
PC1	93.5	82	94	81	93.99	68.1	65	82
PC3	89.6	80	90.4	80	88.05	68.8	NA	NA
PC4	91.69	92.1	91.8	94	91.05	64.2	NA	NA

The two important parameters True Positives (TP) and False Positive (FP) should be considered for a cost-effective bug prediction process [25]. Here FP represent an instance that does not contains any bug but has been identified as a bug by the predictor, whereas TP shows the total number of accurately classified instances. Now, consider the results in Table 4, Logistic Regression has very high AUC results when compared to the schemes in [3] and [4]. This clearly demonstrates that the proposed solution has precisely predicted the bugs in all eight datasets when compared to the schemes in [3] and [4]. This can be further elaborated as, for example, where a bug affects only one in a thousand instances, a completely poor prediction model will always report “negative” (i.e. will not be able to predict that bug) but will still be 99.9% accurate. Unlike accuracy, AUC is insensitive to class imbalance; a poor prediction model would have an AUC of 0.5, which is like not having a prediction at all.

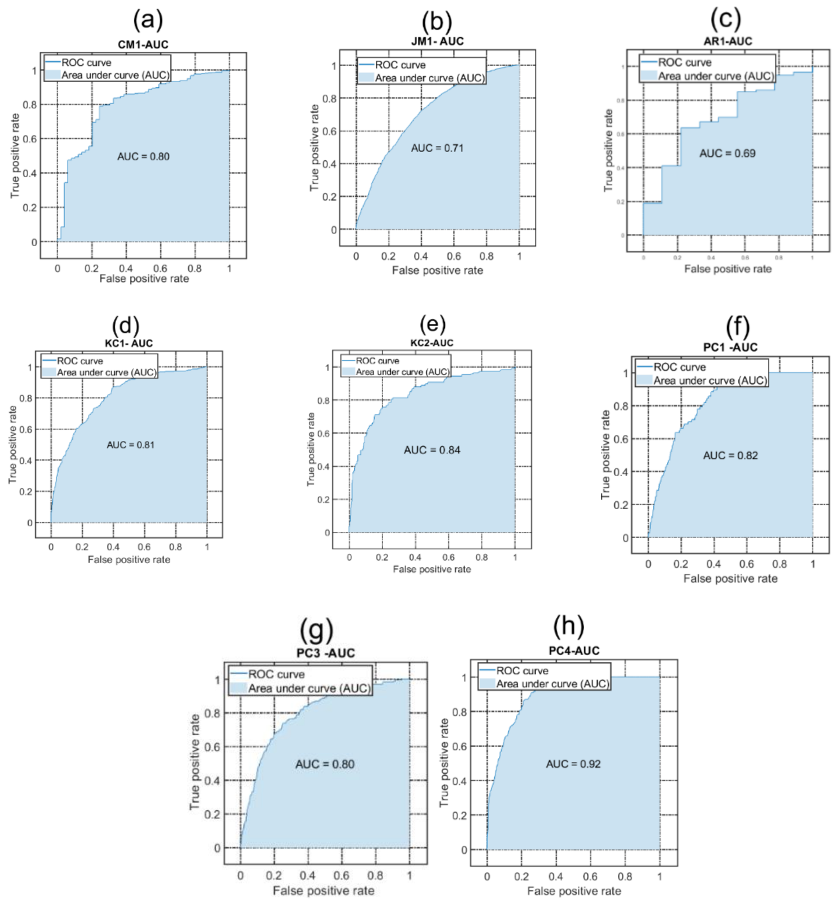

Fig. 3. ROC and AUC for (a) CM1, (b) JM1, (c) AR1, (d) KC1, (e) KC2, (f) PC1, (g) PC3, and (h) PC4 datasets using CFLR scheme

Here, the CFLR and EBT schemes perform better for most of the datasets, while they perform equally well in few cases. In the case of the CM1 dataset, it is evident that the CFLR and EBT approaches achieve 90.4% and 90.8% accuracy while maintaining 80% and 73% of AUC, while the schemes in [3] and [4] have 82.94% and 68% accuracies, and 56.8% and 77% AUC, respectively.

For the AR1 dataset, the CFLR and EBT schemes reache 87.6% and 91% accuracy, while attaining 69% and 79% AUC, while the scheme in [3] makes essentially almost no prediction at all with 55.9% accuracy and 52.7% AUC, barely exceed the 50% default .

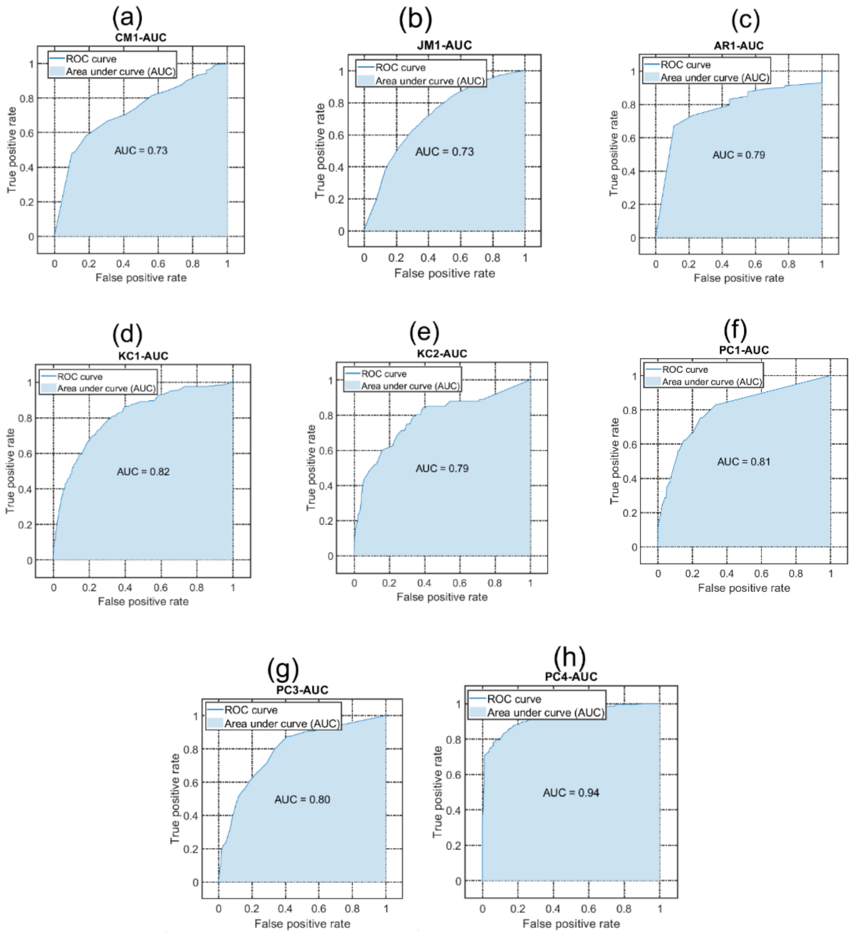

Fig. 4. ROC and AUC for (a) CM1, (b) JM1, (c) AR1, (d) KC1, (e) KC2, (f) PC1, (g) PC3, and (h) PC4 datasets using EBT scheme

Similarly, for PC1 dataset, our CFLR and EBT schemes have 93.5% and 94% accuracy with 82% and 81% AUC, the scheme in [3] has almost equal accuracy (i.e. 93.99%) but with a lower AUC (i.e. 68.1%); this decreased AUC is a sign of lower TP. However, the scheme in [4] had 82% AUC but with a lower accuracy (i.e. 65%), this lower accuracy being due to the lower True Negative (TN).

This lower TP and TN can also be observed in the case of the KC1 and KC2 datasets, where the CFLR scheme accuracies are 86.5% and 85.5% with 81% and 84% AUC, respectively. Similarly, the EBT scheme achieves 87.3% and 79% accuracy, and 84% and 82% of AUC. However, in comparison [3] has 86.1% accuracy but with 69.1% AUC for KC1, and 84.7% accuracy with 68.8% AUC for KC2. On the other hand, [4] has 69% accuracy with 80% AUC for KC1 and 79% accuracy and 85% AUC for KC2.

In case of PC3 and PC4 the CFLR scheme has 89.6% and 91.69% accuracy and 80% and 92.1% AUC, respectively. While the EBT scheme has achieved 90.4% and 91.8% accuracy, and 80% and 94% AUC. However, [3] has almost the same accuracy (i.e. 88.05% and 91.05%) but with a lower AUC (i.e. 68.8% and 64.2%), which shows that in case of PC3 and PC4 again its performance is degraded by lower TP. A comparison of the all eight datasets in terms of AUC can be seen in Figure 3 for CFLR and Figure 4 for EBT.

JM1 is the only case where the accuracy and AUC of the CFLR (i.e. 82.04% and 71.4%) and EBT (i.e. 82.3% and 73%) schemes are almost equal (i.e. 81.23% and 70.1%) to [3]. But again [4] is suffering with low TN resulting in a very low accuracy.

6 Conclusion

Due to the rework cost arising from late discovery of defects, there is a need for more accurate and efficient automated software defect prediction. There are numerous software quality metrics available in the literature which can inform defect prediction. These include traditional code metrics such as McCabe's complexity measures and Hallstead's Software Science (as already noted in the CM1 dataset), object oriented metrics of which the Chidamber and Kemerer suite [26] is the most widely cited, and process metrics such as code churn and code deltas . A challenge for the practitioner is to know which metrics are most valuable in improving the accuracy and efficiency of defect prediction and how they should be used.

In this paper, we first identify and select the most significant metrics using a logistic regression-based scheme. Once the most significant metrics are selected, we then applied the two prediction schemes (i.e. CFLR and EBT) on the selected metrics for accurate prediction of defects. The performances of the proposed schemes are compared with other schemes on eight different datasets, and it is evident from the results that the proposed schemes perform better in terms of accuracy and AUC.

For the software engineer, the role and application of such defect prediction is varied. First, since defect prediction will operate in the context of a software organization with specific process and project challenges to address, then an overall structured approach to its application is necessary. The Goal-Question-Metric approach is an established framework for identifying important dependent and independent product and process features whose values are necessary to achieve some overall software engineering management goal [26]. For example, organization goals might require post-hoc project analysis for the purpose of process improvement, or in-process defect monitoring and prediction for agile process control. There is also the distinction between defect data arising from operational use of a system and/or defect data arising during software verification and validation activities.

The availability of independent variables in a given project will depend on factors such as the integrated development environment in use (e.g. Eclipse, Microsoft Visual Studio), the availability of code analyzers for the programming languages used, the capability of tools used for collection and export of project issues (e.g. Jira, Bugzilla) and the process data available through the project’s code repository and version history.

In this context, the selection of a data mining approach will likely be iterative. The conduct of a pilot data collection exercise is a first step, to explore the fit between (a) best practice in defect prediction, (b) the available data and (c) the organization’s overall process objectives. It is also the case that defect prediction modelling would complement rather than replace the quality assurance expertise within a software team.

ACKNOWLEDGEMENT

This research is supported by the BTIIC (BT Ireland Innovation Centre) project, funded by BT and Invest Northern Ireland.

References

1. BRADY, F., *Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year*. 2013, Cambridge University.
2. Dick, S., et al., *Data mining in software metrics databases.* Fuzzy Sets and Systems, 2004. **145**(1): p. 81-110.
3. Huda, S., et al., *A Framework for Software Defect Prediction and Metric Selection.* IEEE Access, 2018. **6**: p. 2844-2858.
4. Arar, Ö.F. and K. Ayan, *Software defect prediction using cost-sensitive neural network.* Applied Soft Computing, 2015. **33**: p. 263-277.
5. Kassab, M., J.F. DeFranco, and P.A. Laplante, *Software Testing: The State of the Practice.* IEEE Software, 2017. **34**(5): p. 46-52.
6. Ebert, C. and C. Jones, *Embedded Software: Facts, Figures, and Future.* Computer, 2009. **42**(4): p. 42-52.
7. Planning, S., *The economic impacts of inadequate infrastructure for software testing.* National Institute of Standards and Technology, 2002.
8. Bedjanian, J.R.E.n.T.L.D.n.R.A.S.C.S.W.n.R.K.M.n.A.L., *The Path to Software Cost Control.* Defense Acquisition, Technology and Logistics, 2014: p. 23-27.
9. Tassey, G., *The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, 2002.* Forschungsbericht (Zitiert auf Seite 2), 1996.
10. Gyimothy, T., R. Ferenc, and I. Siket, *Empirical validation of object-oriented metrics on open source software for fault prediction.* IEEE Transactions on Software Engineering, 2005. **31**(10): p. 897-910.
11. Malhotra, R., *A systematic review of machine learning techniques for software fault prediction.* Applied Soft Computing, 2015. **27**: p. 504-518.
12. Guo, L., et al. *Robust prediction of fault-proneness by random forests*. in *15th International Symposium on Software Reliability Engineering*. 2004.
13. Catal, C. and B. Diri, *Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem.* Information Sciences, 2009. **179**(8): p. 1040-1058.
14. Osman, H., M. Ghafari, and O. Nierstrasz. *Automatic feature selection by regularization to improve bug prediction accuracy*. in *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. 2017.
15. Arshad, A., et al., *Semi-Supervised Deep Fuzzy C-Mean Clustering for Software Fault Prediction.* IEEE Access, 2018. **6**: p. 25675-25685.
16. Arasteh, B. *Software Fault-Prediction using Combination of Neural Network and Naive Bayes Algorithm* 2018.
17. Pendharkar, P.C., *A data envelopment analysis-based approach for data preprocessing.* IEEE Transactions on Knowledge and Data Engineering, 2005. **17**(10): p. 1379-1388.
18. Aparna, U.R. and S. Paul. *Feature selection and extraction in data mining*. in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*. 2016.
19. Le Cessie, S. and J.C. Van Houwelingen, *Ridge estimators in logistic regression.* Applied statistics, 1992: p. 191-201.
20. Catal, C., *Software fault prediction: A literature review and current trends.* Expert systems with applications, 2011. **38**(4): p. 4626-4636.
21. Jinu M Sunil, L.K., N L Bhanu Murthy. *Bayesian Logistic Regression for software defect prediction*. 2018.
22. Sayyad Shirabad, J.a.M., T.J., *The PROMISE Repository of Software Engineering Databases*. 2005: School of Information Technology and Engineering, University of Ottawa, Canada.
23. McCabe, T.J., *A Complexity Measure.* IEEE Transactions on Software Engineering, 1976. **SE-2**(4): p. 308-320.
24. Halstead, M.H., *Elements of Software Science* 1977: Elsevier Science Inc. 128.
25. Taylor, P., *Autonomic Business Processes*. 2015, University of York.
26. Chidamber, S.R. and C.F. Kemerer, *A metrics suite for object oriented design.* IEEE Transactions on software engineering, 1994. **20**(6): p. 476-493.