Fault-Prone Software Classes Recognition via Artificial Neural Network with Granular Dataset Balancing

Marek Pawlicki[1], Agata Giełczyk[1] Rafał Kozik[12], and Michał Choras[12]

[1] UTP University of Science and Technology, Bydgoszcz, Poland
marek.pawlicki@utp.edu.pl,
[2] ITTI Sp. z o.o., Poland

Abstract. In this paper we have investigated the fault proneness of the software source code using artificial intelligence methods. The main contribution lies on improving the data pre-processing step. Before we put the data into an Artificial Neural Network, are implementing PCA (Principal Component Analysis) and k-means clustering. The data-clustering step improves the quality of the whole dataset. Using the presented approach we were able to obtain 10% increase of accuracy of the fault detection. In order to ensure the most reliable results, we implement 10-fold cross-validation methodology during experiments.

Keywords: pattern recognition, faults detection, ANN, data clustering

1 Introduction

The development of a reliable software system, especially at a low cost, can be a significant challenge. The product also has to be market-ready in a reasonable time. Failure detection and defect proneness prediction become crucial tools for reliable software creation, helping with decision making and resource allocation. Various metrics, such as code complexity, or number of revisions can help spot classes with high probability of bugs. Bug prediction, therefore, is a classification problem. Numerous classification methods have been employed to deal with this challenge, along with Artificial Neural Networks(ANN). While some researchers are reluctant to employ ANNs for their lack of transparency, however their prowess in modeling nonlinear functional relationships seem to make them well suited for the problem of defect prediction [13].

2 State of the Art

Software development is at times an amazingly complex and iterative process. However research indicates that there are ways to augment the process with knowledge inferred with known data mining procedures. For example, Yanguang Shen and Jie Liu conduct a research into the application of Data Mining in Software Testing and Defects Analysis. The authors bring up the following data mining methods for defect testing:

– Association Analysis - an outline of the relations between data sets.
– Cluster Analysis - aggregates data points to form different clusters of features; can be used to clarify the patterns of error types, error stages etc.
– Classification of defects - based on an input set and a set of causes of defects, used to ensure the accuracy of quantitative analysis of software defect.
– Sequence Analysis - used to spot defective software by analyzing the trends of a time-ordered transaction dataset, allowing for detection of potential defects in early stages of development.

Some papers warn of the over-reliance on existing tools geared towards software assessment. In Assessing the Precision of FindBugs by mining Java Projects developed at a University by Antonio Vetro, Marco Torchiano and Maurizio Morisio [1] the authors undertake an assessment of a bug finding tool called FindBugs. Historically, bug finding tools experience a myriad of problems, including a high number of false positives, detecting bugs only partially, prioritizing the bugs in a manner not suited for the project at hand and raising the question of general economic feasibility. The authors hone down on the precision question of those tools. After a series of tests on data gathered from university Java coding assignments, the authors conclude that a proper way to use a bug finding tool is to prioritize the issues report so that the bugs that are most likely to lead to a defect are on top of the list. Almost 30% of the reported issues are bad predictors and should most likely be best left untampered with [1]. In a fairly old paper (2010), Applying Data Mining Techniques in Software Development authors survey the feasibility of using Data Mining principles to augment the software creation process. The identified process consists of Understanding and Analysis of users needs, Interpretation of data and noise removal, preprocessing of data and choosing a proper algorithm, creating a mining model, evaluating and interpreting the model. The paper briefly delves into Frequent Itemsets Mining, Timing Mining, and Classification. Frequent Itemsets Mining is used to uncover defect detection rules, and then association rules create a process model with Apriori and FP2 growth algorithms. The execution timing is briefly touched upon in Timing Mining part as the authors conclude that the technology was then in its infancy. Classification methods are used to predict unknown class label object class [6]. The most recent evaluations of the subject delve into the applications of the Data Mining principles to a far greater extent. For example, in [9] the authors propose a mining approach centred on figuring out the software vulnerability characteristics based on open source vulnerability datasets - Common Vulnerability and Exposure (CVE), Common Weakness Enumeration (CWE) and National Vulnerability Database (NVD). Vulnerability in this article is understood as any weakness in software or hardware logic that results in a negative effect on the confidentiality, integrity or availability of the system when exploited. The illustrated procedure performs the extraction, identification, and mining of the crucial characteristics of software vulnerabilities. The authors relate the probability of a vulnerability occurrence with the chance of a distinct vulnerability category or source code containing the root of the problem explained in the report. This allows to create a knowledge database and to form a procedure to mine the features of software vulnerabilities. After the extraction and identification of features, the approach proceeds to execute the mining by specifying the mining rules. Association rules are used as a determinant of all rules that are present in the database, which fulfil a certain confidence condition. Classification rules are used to designate a minute set of rules in the database which allows forming an accurate classifier. The classifiers performance is evaluated by taking inventory of two metrics - precision and recall, where precision is the rate of true positives to all positives, and recall is the rate of true positives to true positives augmented with the number of false negatives. After data preprocessing a dictionary of vulnerabilities is created, and textual indicators are listed. The method uses Frequency-Inverse Document Frequency (tf-idf) to assign weights to the textual indicators. Those are categorized as essential and non-essential vulnerabilities. The study demonstrates the effectiveness of the approach, noting vast improvement of the Data Mining approach over the manual method, with reported recall around 70% and precision around 60% [9]. In [17] authors target detecting fault-proneness in software classes. Fault proneness is a quality attribute in software development that lends itself to the prediction of potential software faults via machine learning methods. The authors aim to establish a link between object-oriented indicators and fault proneness at the class level. An adaptive neuro-fuzzy inference system (ANFIS) is used, as it amalgamates the benefits of the Fuzzy Inference System and the Artificial Neural Network approaches, along with a Levison-Marquardt updating. Six metrics serve as input, Weighted Methods/Class, Number Of Children, Depth of Inheritance, Response For a Class, Coupling Between Objects and Lack of Cohesion in Methods. The system returns a prediction rate of the class fault-proneness. The authors claim the effectiveness of predicting fault-prone classes at a level of 80% correct classifications and the ability to discover 90% of faulty classes in the best case scenario [17]. One more method fusing fuzzy approaches with data mining is considered in [2]. Multivariate analysis of variance MANOVA is a method of figuring out the effect multiple independent variables have on one dependent variable which uses covariance. The Gini Algorithm is used to create classification and decision trees, and fuzzy logic handles modelling uncertainty by employing partial memberships. The proposed approach uses a mix of those methods on the NASA dataset to detect possible software defects. MANOVA identifies the attributes that have the most impact on defects, and the remaining ones are disregarded. The method develops a fuzzy model for this concentrated dataset. The range values of membership functions are calculated with the Gini Algorithm. The resulting model is successfully tested and achieves an accuracy of 86% in identifying a non-defective software [2].

3 Bug Prediciton Dataset

The dataset [8] used constitutes an accumulation of software development metrics. As the authors explain themselves, the ambition of the dataset is to establish a benchmark to test various bug prediction procedures and to judge weather a new routine is an enhancement over the preceding ones. The set provides features derived from source code metrics along with historical and process data. A number of bugs and severity of them is also provided. The dataset allows for defect prediction at the class level, therefore the data could be aggregated into package or subsystem level by aggregating class metrics.

The dataset accomodates information about 5 projects, these are: Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene and Mylyn. Every project comes with a number of associated metrics, for the use of this work the change log data in the form of comma separated files was used, with features suggested by [15].

4 Principal Component Analysis

The process of finding a feature vector which contains the essence of the data, but with a reduced set of features is called dimensionality reduction. In other words, it is the challenge of building an n-dimensional projection that constitutes the representation of the data in a k-dimensional space. Besides the obvious computational benefits, dimensionality reduction techniques help prevent high dimensionality of input, which induces the phenomenon called 'curse of dimensionality'. The phenomenon causes various machine learning classifiers to underperform in when the number of dimensions inflates. [14] This comes with an exponential increase of samples needed for the classifiers to be accurate.

Principal Component Analysis (PCA) is an established method of dimensionality reduction. Essentially PCA finds the projection of the data in which the data's variance is maximised. In a classic example given in [14], if the data is distributed over a line, performing PCA would quickly inform that the variance over all the other directions is 0, therefore the features responsible for those distributions can be discarded. In this case despite the data-gathering process providing a strong signal in one direction, the data usually contains noise in numerous features. If the strength of the signal overbears the noise to a sufficient extent, extracting the projection which explains maximum variance has a significant chance of containing the essence of the data. The process can be repeated in order to find the next dimension with the largest variance.

5 Data Imbalance

A set is considered imbalanced when the classes are not uniformly represented. This seemingly trivial matter can cause machine learning algorithms to fall short of the expected performance. An example brought to attention in [4] considers a situation where a mammography dataset consists of no more than 2% of abnormalities. In that situation a classification of all the samples to the majority class would yield an accuracy of 98%, completely missing the point of constructing a machine learning algorithm detecting the minority class. With the prevalence of dataset imbalance in most of real-world research problems, two mainstream approaches to dealing with the challenge have emerged. These are different ways of data resampling, either by subsampling the majority class or by oversampling the minority class, and attaching a specific cost function to the training samples [4].

Additionally, recognising the aforementioned issue with accuracy as a performance metric a range of other methods of determining the efficiency of machine learning algorithms were adopted.

6 Artificial Neural Network

Artificial Neural Networks (ANN) are a versatile tool for modeling. Applied in a wide array of uses, they are a standard utility for data mining, providing classification, regression, clustering and time series analysis abilities. The premise of an ANN is that it mimics the learning capabilities of a biological neural network, with emphasis on the properties of neural networks found in human brains, however abstractly simplified [14].

The astonishing modeling ability of ANN as applied to pattern recognition stems from its high adaptability to data. This universal approximation ability is immensely valuable when dealing with real-world data, when the data is abundant, but the patterns concealed in the data are yet to be discovered.

The weights of the ANN are revised by the algorithm with the survey of consecutive data instances, enabling gaining knowledge from experience. Not only can the network attain the relations between the variables, but it can generalize to a sufficient extent so as to allow adequate performance on unforeseen data [7]. An Artificial Neural Network is essentially like fitting a line, plane, or hyperplane though a dataset, defining the relationships that perhaps exist among the features [16].

An ANN with only one computational layer is frequently called a perceptron. This simplest form of ANN contains an input and an output (computational) layer. The input layer provides the data to the output layer, where computations are performed. Said input layer consists of d nodes that represent d features $X = [x_1...x_d]$ and edges of weight $W = [w_1...w_d]$. The output neuron computes $W \cdot X = \sum_{i=1}^{d}(w_i x_i)$. The binary prediction of either -1 or 1 is a mapping based on the sign of the real value of the result of that computation. Adding bias b helps the model perform in environments with high class distribution imbalance. Thus, the prediction of $\hat{y}$ is the result of the equation 1.

$$\hat{y} = sign\{W \cdot X + b\} = sign\{\sum_{i=1}^{d} w_i x_i + b\} \tag{1}$$

In this example, the *sign* plays the role of the actvation funciton $\Phi(v)$. Different activation functions will be used in ANNs with multiple hidden layers, usually either the Rectified Linear Unit (ReLU) or Hard Tanh is utilised for the ease in training multilayered networks. The error of the prediciton can be expressed as the difference between the real-life test value and the predicted value, namely $E(X) = y - \hat{y}$. If the error is different than 0 the weights should be revised.

It becomes the aim of the perceptron to minimise the least-squares between y and $\hat{y}$, for all instances belonging to dataset D. This objective is called the loss function (Equation 2).

$$\sum_{(X,y)\in D} (y - sign\{W \cdot X\}) \tag{2}$$

The loss function is defined over the whole dataset X, the weights W are updated with the learning rate α, and the algorithm iterates over the entire dataset until it converges. This procedure is refered to as stochastic gradien-descent, also expressed by equation 3 [3].

$$W \Leftarrow W + \alpha E(X)X \tag{3}$$

A multilayer neural network features multiple computational layers, called the hidden layers. The name refers to the black box nature of those layers, as the computations are obscured from the users point of view. The data is fed from the input layer to the successive layer with adequate computations along the way, and then fed to another layer and so on until it reaches the output layer. This mechanism is referred to as the feed-forward neural network [3]. The particular count of neuron nodes in the foremost hidden layer usually deviates from the number of inputs. The number of nodes and the number of layers depends on the complexity of the required model and on the availability of data [7]. While there are some instances where using a fully-connected layer of neurons is the norm, utilising hidden layers with the number of nodes below the number of inputs allows for a loss in representation, which actually betters the network's performance. This might come as a result of eliminating the noise in data [3].

Constructing a network with too many neurons tends to lead to overfitting. Overfitting, or overtraining, means that the model adjusted itself to very specific patterns of the training dataset, and therefore, not being general enough, will perform poorly on new, unforseen data [3].

7 Cross-Validation

The models created by various machine learning algorithms, including the ones utilised in this paper, come with a number of risks influencing their performance, including overfitting, or selection bias. The models experiencing those problems can perform outstandingly on the test set, but severely underperform when deployed on unforseen data. To mitigate this risk the models undergo a cross-validation procedure. K-fold cross-validation, or rotation estimation, is more effective in evaluating the effectiveness of machine learning models than simple dataset split methods, like random sub-sampling. The final result is an average of all the measures across all the folds, therefore it is a more fitting assessment of the used methods effectiveness. The k parameter in the very name of the procedure stands for the number of folds the method will use. A fold can be defined as a randomly sampled partition of the data, both equal in size and mutually exclusive with all the other partitions. The partitions are rotated as the test set for validating the model, with the remaining subsets constitute the training set. After repeating the procedure k-times the the evaluation scores are averaged. The most common cross-validation procedure involves k=10 folds, hence the name, 10-fold cross-validation. It is the method utilised in works [11, 10].

8 Experiments

The experiment was conducted using the 10-fold cross-validation methodology.

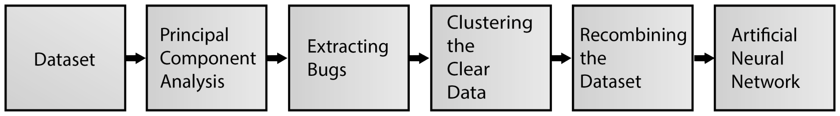
Dataset
Principal Component Analysis
Extracting Bugs
Clustering the Clear Data
Recombining the Dataset
Artificial Neural Network

Fig. 1. The process pipeline

The whole process performed in the research was presented in 1. The [8] dataset consists of 5 projects and an array of metrics dispersed throughout separate files. The files use comma-separated-values (CSV) format. That is a standard. For this particular approach, the bug-metrics, change-metrics, complexity-code-change and single-version-ck-oo datasets were used. Firstly, the datasets are pooled together. Since the metrics are gathered at the class-level, the datasets are merged with regard to the classname column. This results in a single, comprehensive dataframe containing vectors of 42 features (columns), the dependent variable and the classname, for each of the 5 projects in the dataset. Then those dataframes are concatenated to form an aggregare of all five projects. Thus, the dataframe containing the final dataset contains 44 columns and 5371 rows, out of which 853 are classes with with after-release reported bugs. At this stage of research the algorithm will perform binary classification, deciding either if the class will be faulty, or not. For that reason the dependent variable is changed to a Boolean, with $Y > 0$ resulting in a $Y = True$. Secondly, after the dataframe has been formed, the data is scaled and a Principal Component Analysis is performed. The variance test reveals that in this dataset 30 features account for 99.2% of the variability. numberOfVersionsUntil - 0.00172. Thus, the feature vector can be safely reduced to 5, instead of 42 features. After performing PCA the bugged classes and the clear classes are manually divided, and then the clear classes are clustered using a k-means algorithm with the number of centroids matching the number of bugged classes. This balances the dataset, preventing selection bias.

Data prepared in this way is concatenated again, and the rows are randomly mixed. The dataset is then fed to an Artificial Neural Network, with 30 nodes on the input layer, two hidden layers - one fully connected, one with 15 nodes, and a single output node.

9 Results

The [8] datasets for all the projects were utilised. The dataset provides 42 features, 30 of witch were chosen through performing PCA without impacting the prediction accuracy. In the future data collected from tools such as GitLab or SonarQubeThe will be utilised, as presented in [12]. The data was severely unbalanced with collective 853 examples of classes with bugs in a total of 5371 records, constituting 15,89% of the dataset. The Artificial Neural Network (ANN) classified strongly in favour of the majority class, therefore two data balancing strategies were evaluated. A subsampling approach by matching the number of bug records with the same exact number of randomly picked samples of the clear dataset yielded and average accuracy of 0.553 in a 10-fold cross-validation test. The clustering approach achieved an accuracy of 0.643, proving an almost 10% point increase in accuracy. Furthermore, we have noticed that the classifier relied heavily on the bug metrics features. Eliminating all the columns which referred to bugs found before release resulted in the ANN of improved precision and recall with similar accuracy of 0.656. In Table 1 and Table 2 some more obtained results were presented, where:

- Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to all the observations in the actual class.
- f1-score is expressed with Eq. 4.

$$f1 - score = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{4}$$

Table 1. Model including the bug metrics

	precision	recall	f1-score	support
False	0.62	0.71	0.66	87
True	0.65	0.55	0.59	84
micro avg	0.63	0.63	0.63	171
macro avg	0.63	0.63	0.63	171
weighted avg	0.63	0.63	0.63	171

10 Conclusions

This paper evaluated the use of data-balancing methods in conjunction with artificial neural networks for the recognition of fault-prone software classes based on the bug prediction [8] dataset. There is room for future research in this area, in all the touched-upon aspects. Firstly, in order to fulfil the requirement expressed

Table 2. Model not including the bug metrics

	precision	recall	f1-score	support
False	0.72	0.63	0.67	84
True	0.68	0.76	0.72	87
micro avg	0.70	0.70	0.70	171
macro avg	0.70	0.69	0.69	171
weighted avg	0.70	0.70	0.69	171

by software house's senior staff to improve the understanding of the developed projects and improve their ability to plan and assign tasks and sprints in rapid software development processes data from platforms like GITlab and SonarQube should be utilised, as seen in [5]. There is room for improvement both in the clustering procedure, and the ANN architecture. However, the current results are promising, and we are already working on improving the presented algorithm.

11 Acknowledgements

This work has received funding from the European Unions Horizon 2020 research and innovation programme under Grant agreement no. 732253. We would like to thank all the members of the Q-Rapids H2020 project consortium

References

1. M. Torchiano A. Vetro' and M. Morisio. Assessing the precision of findbugs by mining java projects developed at a university. In *Second International Conference on Intelligent Computation Technology and Automation, Changsha, Hunan*, pages 110–113. 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010.
2. M. F. Adak. Software defect detection by using data mining based fuzzy logic. In *Sixth International Conference on Digital Information, Networking, and Wireless Communications (DINWC), Beirut*, pages 65–69, 2018.
3. Charu C. Aggarwal. *Neural Networks and Deep Learning A Textbook.* 2018.
4. Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.
5. Michał Choraś, Rafał Kozik, Damian Puchalski, and Rafał Renk. Increasing product owners' cognition and decision-making capabilities by data analysis approach. *Cognition, Technology & Work*, 2019.
6. Z. Chun-mei and L. Zhi-ling. Applying data mining techniques in software development. In *2nd IEEE International Conference on Information Management and Engineering, Chengdu*, pages 535–538, 2010.
7. Ivan Nunes da Silva Danilo Hernane Spatti Rogerio Andrade Flauzino Luisa Helena Bartocci Liboni Silas Franco dos Reis Alves. *Artificial Neural Networks A Practical Course.* 2017.

8. Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*, pages 31 – 41. IEEE CS Press, 2010.
9. X. Li et al. A mining approach to obtain the software vulnerability characteristics. In *Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 296–301, 2017.
10. Witten D. Hastie T. & Tibshirani R. James, G. An introduction to statistical learning. In *Cluster Comput (2018).*, 2013.
11. R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, pages 1137–1145, 1995.
12. Rafał Kozik, Michał Choraś, Damian Puchalski, and Rafał Renk. Q-rapids framework for advanced data analysis to improve rapid software development. *Journal of Ambient Intelligence and Humanized Computing*, 2019.
13. J. Lo. The implementation of artificial neural networks applying to software reliability modeling. In *2009 Chinese Control and Decision Conference*, pages 4349–4354, June 2009.
14. Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook, 2nd ed.* 01 2010.
15. Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 181–190, New York, NY, USA, 2008. ACM.
16. Simone Bassis Anna Esposito Francesco Carlo Morabito Eros Pasero. *Advances in Neural Networks.* 2016.
17. Rajkumar N. & Duraisamy S. Viji, C. Prediction of software fault-prone classes using an unsupervised hybrid som algorithm. In *Cluster Comput (2018).*, 2018.