



Recognizing Faults in Software Related Difficult Data

Michał Choraś^{1,2(✉)}, Marek Pawlicki^{1,2}, and Rafał Kozik^{1,2}

¹ ITTI Sp. z o.o., Poznań, Poland

mchoras@itti.com.pl

² UTP University of Science and Technology, Bydgoszcz, Poland

marek.pawlicki@utp.edu.pl

Abstract. In this paper we have investigated the use of numerous machine learning algorithms, with emphasis on multilayer artificial neural networks in the domain of software source code fault prediction. The main contribution lies in enhancing the data pre-processing step as the partial solution for handling software related difficult data. Before we put the data into an Artificial Neural Network, we 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. We have also evaluated a wide range of hyperparameter setups for the network, and compared the results to the state of the art, cost-sensitive approaches - Random Forest, AdaBoost, RepTrees and GBT.

Keywords: Pattern recognition · Faults detection · ANN · Data clustering

1 Introduction and Context

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. However, the analysis of software related data causes many problems and possible pitfalls due to intrinsic data difficulties. The aspects of data difficulties and motivation for this work are presented in details in Sect. 2.

To this point 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 [1].

Software quality is a fundamental competitive factor for the success of contemporary software houses. The Horizon 2020 Q-Rapdis project aims to augment the strategic decision-making procedures of software development by supplying strategic indicators of quality requirements. Fault-proneness is one of such metrics, as it can significantly affect the overall cost of the software. The Q-Rapdis strategic indicators stem directly from the metrics and factors calculated from the software development-related data through the use of various data mining and machine learning procedures.

The paper is structured as follows: in Sect. 2 we discuss the problems and difficulties in analyzing the realistic software related data. In Sect. 5.1 the used benchmark dataset is described in details. Section 3 is devoted to Artificial Neural Network and the algorithms used in this work, while Sect. 4 addresses the problem of data imbalance. Results and the comparison with other standard machine learning approaches are presented in Sect. 5, while conclusions are given thereafter.

2 Problems and Difficulties in Real Software Related Data

In this paper we focus on pre-processing and recognizing (detecting faults/bugs) the software related data. But why is software data considered difficult anyways? There are many reasons and answers to such question, e.g. the following aspects contribute:

- Software related data from real SW companies/developers is considered sensitive commercial data. Commercial companies and SW houses are not eager to share SW related data, even if it is not directly the code. Most companies use software management and monitoring tools such as SonarQube, JIRA, Git-Lab, Jenkins and many others. Still the data retrieve from those tools provide information about processes, metrics, quality, testing aspects and much more, and all of those can reveal information about companies and teams as such. Moreover, such data might contain personal information (such as the names of programmers), therefore the privacy and GDPR aspects should also be taken into account [2].
- From machine learning perspective, software related data is often a one-shot learning. If you train any system/classifier on a data from one project or from one company, it is still not representing other projects and companies, so the training and adjusting algorithms have to be repeated all the time. Indeed, especially now in the era of RSD (Rapid Software Development) and agile/lean methodologies, it is difficult to observe long-term patterns in the way of working. Projects, developers and approaches change often, causing the sudden changes in the data as well, meaning that the trained models might not be relevant anymore. Therefore, lifelong learning approach to machine learning is beneficial and required.
- Software related data contains noise, and almost always the manual work on the data adaptation is needed. The good example is that, for instance each

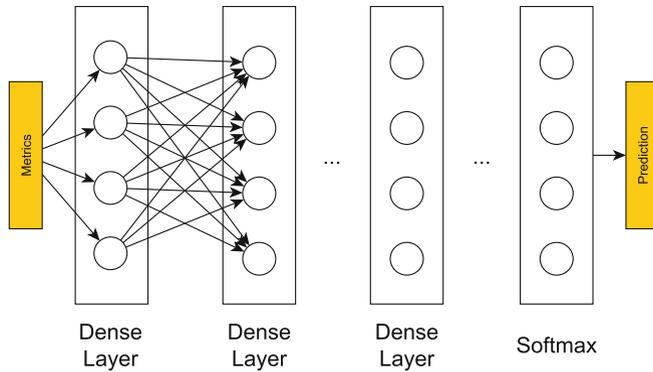


Fig. 1. A general architecture of the neural network adapted for the bug prediction (the depth of the network varies depending on the experiment).

team (even within the same company) might use different naming conventions for GitLab labels. The processes, e.g. of tickets cycles can also vary. Software data also suffer from the problem of data imbalance. The classes containing faults/bugs are under-represented, which causes the typical difficult data (d2) problem for machine learning techniques [3].

Therefore, the abovementioned problems motivate our research towards improving machine learning techniques for improving the quality of rapid software development.

3 Proposed Method

Artificial Neural Networks (ANN) constitute a functional instrument for creating machine learning models. They are a widely used tool for data mining, as they equip the user with classification, regression, clustering and time series analysis abilities. The assertion of an ANN is that it tries to imitate the learning capabilities of a biological neural network, abstractly simplified [4].

The impressive modeling capability of an ANN in fields relying on pattern recognition lies in direct proportion to its striking adaptability to data. Its extensive approximation capability is notably beneficial in handling real-world data, when there is plenty of data, but the patterns buried in the data are yet to be uncovered. Not only can the network figure out the interconnections among the variables, but it can generalize to a sufficient extent so as to provide satisfactory achievements on novel data [5]. An Artificial Neural Network is essentially like fitting a line, plane, or hyper-plane though a dataset, defining the relationships that might exist among the features [6].

A multilayer neural network is constructed with the use of multiple computational (hidden) layers. The data flows from the input layer to the following layer with adequate arithmetic along the way, and then is supplied to the following

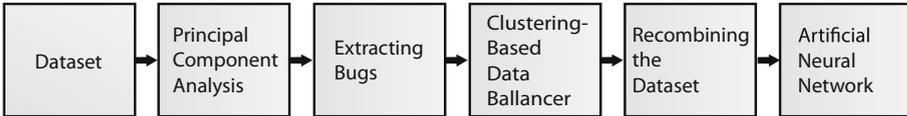


Fig. 2. The procedure pipeline.

layer and so on until it arrives at the output layer. A model illustrating the general architecture of a multilayer neural network can be seen in Fig. 1. This mechanism is dubbed the feed-forward neural network [7]. The number of neurons and the number of layers depends on the complexity of the required model and on the availability of data [5]. Using hidden layers with the number of nodes below the number of inputs creates a loss in representation, which frequently betters the network’s performance. This might come as a result of eliminating the noise in data.

Designing a network with too many neurons can result in overfitting. Overfitting, or overtraining, means that the model fitted itself to extremally specific patterns of the training dataset, thus it will perform poorly on new, novel data, as it is not general enough [7].

The proposed method uses Principal Component Analysis for dimensionality reduction, singles out the bug instances in the dataset, clusters the ‘clean’ examples to the number of clusters that balances the number of bugs and re-merges the dataset to achieve a balanced dataset, which is then fed to the classifier. The pipeline is shown in Fig. 2.

4 Data Imbalance

A set is referred to as imbalanced when the classes are not represented in an equal manner [3]. What might initially seem like a negligible issue can cause machine learning algorithms to fail. An instance supplied in [8] explains a situation where a mammography dataset includes no more than 2% of abnormalities. In that case a classification of all the samples to the majority class would output an accuracy of 98%, strikingly missing the point of creating a machine learning algorithm to identify the minority class. Dataset imbalance exists in most of real-world research problems, so two resolutions to the challenge have been developed. Resampling - like subsampling the majority class and oversampling the minority class. Additionally, one could attach a specific cost function to the training samples [8]. Inspired by the emergence of the granular computing (GrC) paradigm, as it proved valuable in multiple scenarios [9] and after the successful application of the paradigm in [10] we are now investigating the feasibility of GrC for dataset balancing. An interesting approach using k-NN algorithm and rough sets can be observed in [11]. Our proposed method stems from the idea that one can include the characteristics of the dataset by clustering the majority class so the number of clusters matches the number of data samples in the minority class. While this method has it’s drawbacks - namely the problem of

the overlapping clear and bug granules, it still improves the accuracy by 10% as compared with a simple subsampling approach.

Table 1. Results of all datasets combined, using 10-fold cross validation

	Precision	Recall	F1-score	Support
False	0.61	0.57	0.59	80
True	0.64	0.67	0.66	91
Accuracy				0.6612

Table 2. Results of 4 datasets combined, new project classification

	Precision	Recall	F1-score	Support
False	0.91	0.42	0.58	1288
True	0.18	0.76	0.28	209
Accuracy				0.469

5 Results and Comparison with Classic ML Methods

5.1 Bug Prediction Dataset

The dataset [12] which is utilised consists of an aggregation of class-level software development metrics. As mentioned in the accompanying paper, the main aim of the dataset is in providing a benchmark as an experimental field to test-run novel approaches. The set supplies characteristics derived from source code metrics in conjunction with historical and process information. A number of bugs and their impact is also supplied. Since the data is provided at a class level, the defect prediction can also be performed at the class level. The data could, however, be combined into package or subsystem level by summing class metrics.

The dataset contains metrics of 5 projects, these are: Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene and Mylyn. Every project comes with a range of derived metrics, of which change log data in the form of comma separated files was used, for the use in this paper. The features were suggested by [13].

5.2 Results

In order to prove that the analysed data is difficult and to demonstrate that the proposed method allowed us to achieve superior results, we have compared various classical machine learning methods. In order to do that we have used a ROC

Table 3. Training on the ‘Eclipse’ project, classification on ‘pde’

	Precision	Recall	F1-score	Support
False	0.89	0.90	0.90	1288
True	0.34	0.30	0.32	209
Accuracy				0.8196

curve (Fig. 3) to report the effectiveness in terms of the number of false positives (number of false alarms) and true positives (number of correctly predicted bugs). In these experiments we have considered the following classifiers:

- Random Forest
- AdaBoost
- Ensemble of RepTrees
- Gradient Boosted Trees

All of these methods have been wrapped with a metaclassifier that made the base classifier cost-sensitive. More precisely, the metaclassifier weights the training instances according to the total cost assigned to each class.

In this experiment, the Random Forest classifier is composed of 300 Random Trees that are combined together using the bagging technique. Each bag contains roughly 20% of data. During the training we control the depth of the trees. We set a hard limit to 10.

For the AdaBoost method we have used the classical approach. The ensemble is composed of one-level decision trees (decision stumps). We have noticed that increasing the ensemble size above 100 does not improve the quality.

The ensemble of RepTrees is build similarly to a Random Forest. However, instead of a Random Tree as a base classifier we have adapted a well-known Rep-Tree decision tree (Reduced Error Pruning Tree). This machine learning technique uses a pruned decision tree. First, the method generates multiple regression trees in each iteration. Afterwards, it chooses the best one. It uses regression tree adapting variance and information gain (by measuring the entropy). The algorithm prunes the tree using a back fitting method.

The GBT stands for Gradient Boosted Trees classifier. The method uses an additive learning approach. In each iteration a single tree is trained and is added to the ensemble in order to fix errors (optimise the objective function) introduced in the previous iteration. The objective function measures the loss and the complexity of the trees comprising the ensemble.

As it is shown in Fig. 3 and Table 6, we have achieved the best results for the Random Forest classifier. Although, the recall for this method is higher from that presented for our method in Table 1, the precision and f1-score remain far inferior.

When researching the ANN method multiple scenarios were evaluated all throughout the duration of the experiments. The best accuracy results along with their respective hyperparameter setups are found in Table 4, followed by

the specific parameters of the comparison algorithms in Table 5. As mentioned earlier, the dataset [12] provided the metrics of 5 different coding projects. This situation differs from the one evaluated in [14] and in [15], where in order to fulfil the requirement stated by the software house’s executives data from platforms like GITlab and SonarQube were proposed. One of the approaches evaluated how would the algorithm perform if it was trained on one project and tested on another. The detailed results can be seen in Table 3. A different scenario evaluated how the ANN trained on 4 of the projects would perform on a new project, as seen in Table 2. Finally a 10-fold cross validation of all the datasets combined resulted in the performance depicted in Table 1.

Table 4. Best hyperparameter setups found for the ANN’s accuracy, a summary of multiple gridsearches.

4 hidden layers					
Epochs	Optimizer	Neurons	Batch_size	Activation	Accuracy
6000	adam	10	100	hard_sigmoid	0.647557
4500	rmsprop	7	500	hard_sigmoid	0.665147
4500	rmsprop	7	500	hard_sigmoid	0.665147
3 hidden layers					
Epochs	Optimizer	Neurons	Batch_size	Activation	Accuracy
4500	adam	8	500	hard_sigmoid	0.663844
4000	rmsprop	5	500	hard_sigmoid	0.663844
4000	adam	5	500	hard_sigmoid	0.663844
4000	rmsprop	10	500	hard_sigmoid	0.669055
4000	adam	10	500	hard_sigmoid	0.665798
5000	adam	10	500	hard_sigmoid	0.664495
5000	adam	5	1000	hard_sigmoid	0.663192
5000	rmsprop	10	1000	hard_sigmoid	0.665798
2 hidden layers					
Epochs	Optimizer	Neurons	Batch_size	Activation	Accuracy
10	rmsprop	5	5	sigmoid	0.644300
1 hidden layer					
Epochs	Optimizer	Neurons	Batch_size	Activation	Accuracy
4500	adam	8	500	hard_sigmoid	0.663844
4000	rmsprop	10	500	hard_sigmoid	0.669055
3000	rmsprop	10	500	hard_sigmoid	0.663844
3000	rmsprop	10	500	sigmoid	0.663844
2000	rmsprop	5	200	sigmoid	0.670358

Table 5. Comparison ML algorithms setups

Classifier name	Ensemble size	Meta-algorithm	Base classifier	Max depth
Random Forest	300	Bagging	Random Tree	10
AdaBoost	100	Boosting	Decision stump	
RepTrees	300	Bagging	Decision Tree	10
GBT	300	Boosting	Decision Tree	

Table 6. Comparison of methods on all datasets combined, using 10-fold cross validation

	Precision	Recall	F1-score
RandomForest	0.392	0.709	0.505
AdaBoost	0.294	0.765	0.425
Bag of RepTrees	0.308	0.814	0.447
GBT	0.300	0.761	0.431

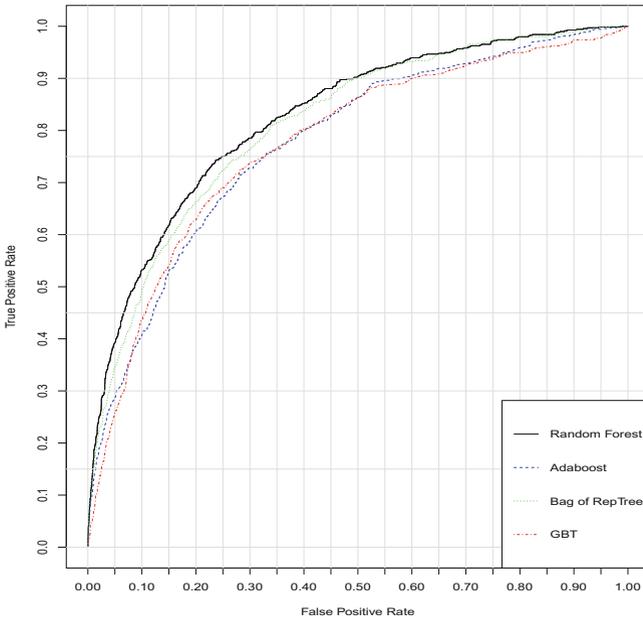


Fig. 3. ROC curve obtained for various classifiers: Random Forest, Adaboost, Bag of RepTrees, Gradient Boosted Trees (GBT).

6 Conclusions

In this paper we tackle the problem of the analysis of difficult software-related data. In general, such data can be analyzed in order to improve the software quality, detect faults and bugs or improve programming patterns. However, quite often the results are tampered by the nature of the data. Hereby, we propose to use machine learning techniques in order to detect bugs while addressing the problem of data imbalance. The presented results (Table 7) are comparable to other approaches, and we currently work to use them in practice on real commercial data from industrial software products.

Table 7. An example of ANN optimisation results - 3 hidden layers

Epochs	Optimizer	Neurons	Batch_size	Activation	ACC
2000	rmsprop	5	200	relu	0.582410
2000	adam	5	200	relu	0.595440
2000	SGD	5	200	relu	0.163518
2000	rmsprop	10	200	relu	0.578502
2000	adam	10	200	relu	0.593485
2000	SGD	10	200	relu	0.327036
2000	rmsprop	15	200	relu	0.553094

Acknowledgments. This work is funded under Q-Rapids project, which has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 732253.

References

1. Lo, J.: The implementation of artificial neural networks applying to software reliability modeling. In: 2009 Chinese Control and Decision Conference, pp. 4349–4354, June 2009
2. Choraś, M., Kozik, R., Renk, R., Hołubowicz, W.: A practical framework and guidelines to enhance cyber security and privacy. In: International Joint Conference - CISIS 2015 and ICEUTE 2015, 8th International Conference on Computational Intelligence in Security for Information Systems/6th International Conference on European Transnational Education, 15–17 June 2015, Burgos, Spain, pp. 485–495 (2015)
3. Kozik, R., Choraś, M.: Solution to data imbalance problem in application layer anomaly detection systems. In: Martínez-Álvarez, F., Troncoso, A., Quintián, H., Corchado, E. (eds.) HAIS 2016. LNCS (LNAI), vol. 9648, pp. 441–450. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32034-2_37
4. Maimon, O., Rokach, L.: Data Mining and Knowledge Discovery Handbook, 2nd edn. Springer, Boston (2010). <https://doi.org/10.1007/978-0-387-09823-4>

5. da Silva, I.N., Spatti, D.H., Flauzino, R.A., Liboni, L.H.B., dos Reis Alves, S.F.: Artificial Neural Networks. A Practical Course. Springer, Cham (2017). <https://doi.org/10.1007/978-3-319-43162-8>
6. Bassis, S., Esposito, A., Morabito, F.C., Pasero, E.: Advances in Neural Networks. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-33747-0>
7. Aggarwal, C.C.: Neural Networks and Deep Learning. A Textbook. Springer, Cham (2018). <https://doi.org/10.1007/978-3-319-94463-0>
8. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE synthetic minority over-sampling technique. *J. Artif. Int. Res.* **16**(1), 321–357 (2002)
9. Pawlicki, M., Choraś, M., Kozik, R.: Recent granular computing implementations and its feasibility in cybersecurity domain. In Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, 27–30 August 2018, Hamburg, Germany, pp. 61:1–61:6 (2018)
10. Kozik, R., Pawlicki, M., Choraś, M., Pedrycz, W.: Practical employment of granular computing to complex application layer cyberattack detection. *Complexity* **2019**, 1–9 (2019)
11. Borowska, K., Stepaniuk, J.: Granular computing and parameters tuning in imbalanced data preprocessing. In: Saeed, K., Homenda, W. (eds.) CISIM 2018. LNCS, vol. 11127, pp. 233–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99954-8_20
12. D'Ambros, M., Lanza, M., Robbes, R.: An extensive comparison of bug prediction approaches. In: Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories), pp. 31–41. IEEE CS Press (2010)
13. Moser, R., Pedrycz, W., Succi, G.: 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 2008, pp. 181–190. ACM, New York (2008)
14. Choraś, M., Kozik, R., Puchalski, D., Renk, R.: Increasing product owners' cognition and decision-making capabilities by data analysis approach. *Cogn. Technol. Work* **21**, 191–200 (2019)
15. Kozik, R., Choraś, M., Puchalski, D., Renk, R.: Q-rapids framework for advanced data analysis to improve rapid software development. *J. Ambient Intell. Humaniz. Comput.* **10**(5), 1927–1936 (2019)