**The Study of Data Processing Methods and Machine Learning Classification Algorithms for Software Bugs Prediction**

Xinyi He, Zhenjie Jia

Abstract:

Keywords: bug prediction; software engineering; machine learning

# 1.Introduction

Software has an increasing influence on the world. It is very essential to build durable software with low cost. Statistically, the effort in finding and fixing bugs in a software will consume near 80% of the budget of a software development [1]. Therefore, the defect prediction is very important to improve the software quality and reduce the effort to find the bugs as the size of the software is getting larger.

The software defect or bug prediction can be seen as a classification problem of whether a class in the software has bugs or not. The process of software defect prediction requires the following steps:

1. Feature extraction: To represent the status of software of different languages, we need software’s feature matrix to be used as the train set.
2. Model training: Using different ML methods like random forest, logistic regression, neural networks, etc to train the model.
3. Results comparison: Testing on different software or same software’s latest version.

This field of research is now facing the following challenges:

1. The lack of the dataset. Only a few open source software have public and clear bug tracker that can be used to generate the data set.
2. The dataset is imbalanced. The number of data with “bugs” label in data set is about 15%, while the rest of the data in a dataset all have the “zero bugs” label.
3. Different projects’ feature metrics are different.
4. The data in the dataset vary a lot. Many features are zero, some features’ numbers are big while some are extremely small

This project focus on

1) discussing some ML techniques.

2) the challenge of software defect prediction.

3) the analysis of the raw dataset and the comparison of the dataset after different data preprocessing methods.

4) trying different Machine Learning methods on the software defect model and the comparison of the results.

The project goal is to find an improved process to predict the bugs in the software.

# 2. Literature review

(Describe other peoples’ methods and results on the same dataset above)

(overview)

That process on predicting the bugs on the software using algorithms starts at the beginning of the computer science history. Back to 1971, Akiyama et al. (1971) first attempted to use software size based metrics and regression model predict the defects of some software.

But the progress is slow. Kamei & Shihab (2016) summarized that challenges on this topic back to early 2000s are: 1)Lack of data since there weren’t much open source software and the bugs are always given at subsystem or file levels, not the function level; 2) lack of variety of independent and dependent variables. The most independent variables used are size-based metrics, especially code complexity metrics, while the most dependent variables are post-release defects.

In recent years, with the trend of the machine learning, the software defects prediction is combined closely with many machine learning algorithms. Tantithamthavorn et al. (2016) used an off-the-shelf automated parameter optimization technique on 11 machine learning algorithms including Naive Bayes, KNN, SVM, neural network on data sets from different sources, such as NASA’s (xxx) data set and Eclipse (xxx).

(results on different datasets than us)

Zoltan Toth, et al selected 15 Java projects from GitHub to construct a public bug database from. They matched the already known and fixed bugs with the corresponding source code elements (classes and files) and calculated a wide set of product metrics on these elements. After creating the desired bug database, they investigated whether the built database is usable for bug prediction. they used 13 machine learning algorithms to address this research question and finally they achieved F-measure values between 0.7 and 0.8.[5]

Kai Pan, et al used the program slicing and the Understand for C++ metrics computed for 887 revisions of the Apache HTTP project and 76 revisions of the Latex2rtf project to classify source code files or functions as either buggy or bug-free. They then compared their classification prediction accuracy. Program slicing metrics have slightly better performance than the Understand for C++ metrics in classifying buggy/bug-free source code. Program slicing metrics have an overall 82.6% (Apache) and 92% (Latex2rtf) accuracy at the file level, better than the Understand for C++ metrics with an overall 80.4% (Apache) and 88% (Latex2rtf) accuracy.[6]

Xin Xia, et al evaluated the effectiveness of various supervised learning algorithms to predict if a bug report would be reopened. They chose 7 state-of-the-art classical supervised learning algorithms in machine learning literature, including kNN, SVM, SimpleLogistic, Bayesian Network, Decision Table, CART and LWL, and 3 ensemble learning algorithms, including AdaBoost, Bagging and Random Forest, and evaluated their performance in predicting reopened bug reports. The experiment results showed that among the 10 algorithms, Bagging and Decision Table (IDTM) achieved the best performance. They achieved accuracy scores of 92.91% and 92.80%, respectively, and reopened bug reports F-Measure scores of 0.735 and 0.732, respectively. These results improved the reopened bug reports F-Measure of the state-of-the-art approaches proposed by Shihab et al. by up to 23.53%.[7]

(results from the same datasets)

(result from <http://bug.inf.usi.ch/index.php>)

Zhang Feng, et al used spectral Random Forest classifier which calculate the laplacian matrix and perform the eigendecomposition on the matrix to normalize the matrix. They tested it in D'Ambros’ dataset and achieved 81%, 78% of accuracy within projects and 87%, 71% of accuracy when crossing the projects for JDT and PDE. [8]

CesarCouto, et al built a model that will use some alarm threshold functions to select the variations in the metrics that may have contributed to the occurrence of defects. They remove the classes with zero defects and got 61% precision, 53% recall for JDT, 27% precision 54% of recall for PDE. [9]

Haidar Osman, et al revealed that tuning model hyperparameters has a statistically signiﬁcant positive effect on the prediction accuracy of the models. The prediction accuracy was improved by up to 20% in KNN and by up to 10% in SVM.[10]

Yasutaka Kamei, et al showed that package-level predictions were not more effective than ﬁle-level predictions and the effectiveness of package-level predictions can improve if we performed our predictions at the ﬁle-level then lift it to the package-level instead of collecting all metrics at the package-level. However the new model still did not outperform ﬁle-level predictions when considering the quality assurance efforts.[11]

Shivkumar Shivaji, et al proposed a feature selection technique applicable to classiﬁcation-based bug prediction.[12]

(result from <https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/>)

Thomas Zimmermann, et al conducted a work on the code base of the Eclipse programming environment. They extended their data with common complexity metrics and the counts of syntactic elements. Then they built logistic regression models for the Eclipse bug data set to predict whether files/packages have post-release defects. For file level with version 3.0, the accuracy scored, recall score and precision score they get are about 0.869, 0.224 and 0.675 while For package level with version 3.0, the accuracy scored, recall score and precision score they get are about 0.855, 0.789 and 0.892.[13]

|  |  |
| --- | --- |
| Dataset | Link |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Paper | Techniques used | Dataset | Results |
| Toth et al. (2017) | 15 machine algorithms |  | They achieved an f-measure of 0.7. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 3. Machine Learning algorithms

the logistic regression,the random forest, the neural network. These are top three classifier after trying all the classifiers.

**3.1 Logistic regression**

Logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc... Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one.

Image

**3.2 Random forest**

Random forests or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks that operates by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set).

The first algorithm for random decision forests was created by [Tin Kam Ho](https://en.wikipedia.org/wiki/Tin_Kam_Ho) using the [random subspace method](https://en.wikipedia.org/wiki/Random_subspace_method), which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by [Leo Breiman](https://en.wikipedia.org/wiki/Leo_Breiman) and [Adele Cutler](https://en.wikipedia.org/wiki/Adele_Cutler), who registered "Random Forests" as a [trademark](https://en.wikipedia.org/wiki/Trademark). The extension combines Breiman's "[bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating)" idea and random selection of features, introduced first by Ho and later independently by Amit and [Geman](https://en.wikipedia.org/wiki/Donald_Geman) in order to construct a collection of decision trees with controlled variance.

3.3 Multilayer perceptron **(MLP)**

Multilayer perceptron (MLP) is a class of [feedforward](https://en.wikipedia.org/wiki/Feedforward_neural_network) [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network). An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear [activation function](https://en.wikipedia.org/wiki/Activation_function). MLP utilizes a [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) technique called [backpropagation](https://en.wikipedia.org/wiki/Backpropagation) for training. Its multiple layers and non-linear activation distinguish MLP from a linear [perceptron](https://en.wikipedia.org/wiki/Perceptron). It can distinguish data that is not [linearly separable](https://en.wikipedia.org/wiki/Linear_separability).

Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

# 4. Experiments

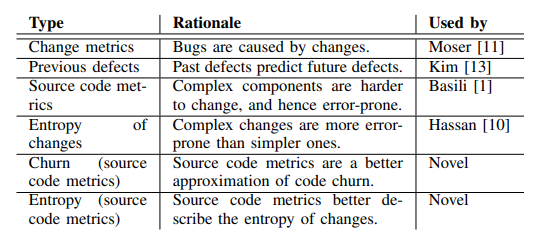
## 4.1 the dataset used in the software bug prediction:

The dataset used in this research includes:

1. Dataset from NASA: KC1, KC2, PC1, CM1, JM1: <http://promise.site.uottawa.ca/SERepository/datasets-page.html>
2. Dataset from *Marco D'Ambros, Michele Lanza, Romain Robbes* : <http://bug.inf.usi.ch/index.php>
3. Dataset from Zimmermann : <https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/>

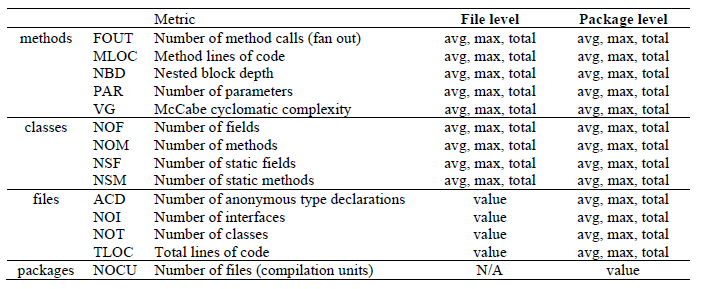
We used the first two datasets to test the model on different software projects and the last dataset to test the model on different versions of the same software.

table 1 - *Marco D'Ambros’s data set*



we combined all the combined all the columns from the above datasets to form a new larger dataset with 40 columns as input attributes.

table 2 - Zimmermann’s data set



## 4.2 Preprocessing

Over sample: Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don’t have a ton of data to work with.

Under sample: Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

Syn sample: A technique similar to upsampling is to create synthetic samples. Here we will use imblearn’s SMOTE or Synthetic Minority Oversampling Technique. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.

PCA:

Standardization:

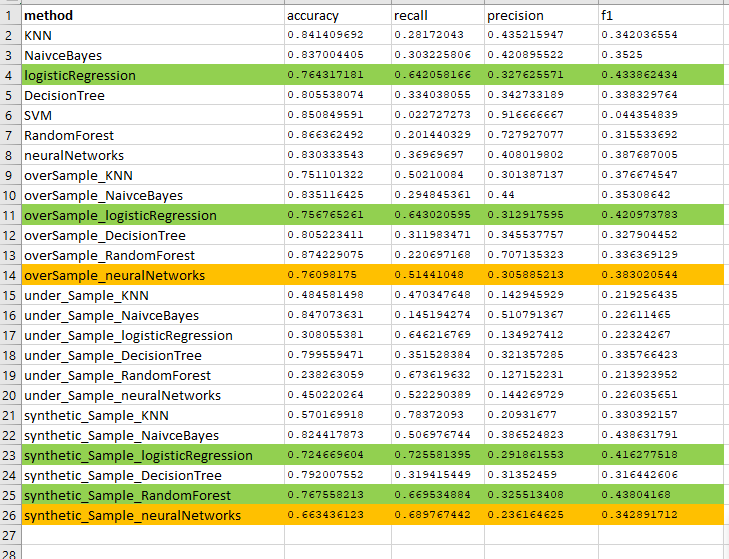
And describe how the data is distributed after these methods.

## 4.3 trying different ML methods within the projects and describe the result

table 3 - parameters for different algorithms

|  |  |
| --- | --- |
| algorithm name | parameters |
|  |  |

table4 - results with the methods with the same project (<http://bug.inf.usi.ch/index.php>)

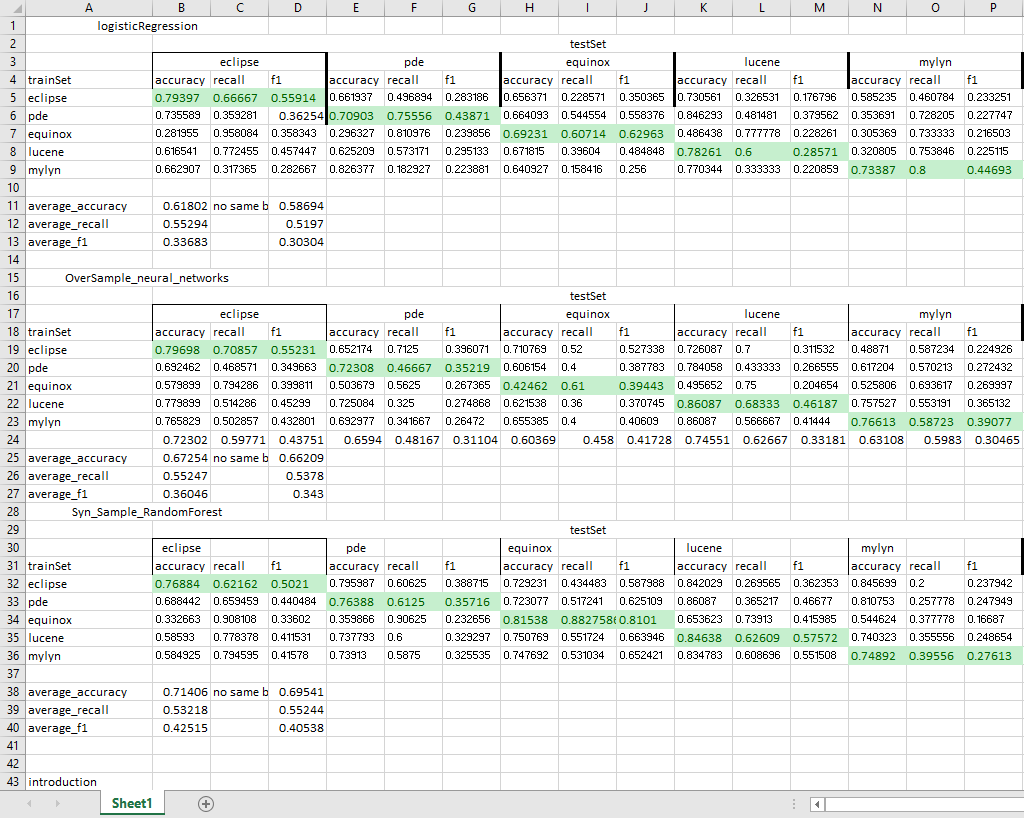


## 4.4 trying different ML methods cross the projects and describe the result

table 5 - parameters for different algorithms

|  |  |
| --- | --- |
| algorithm name | parameters |
|  |  |

table6 - results with the methods cross the projects (<http://bug.inf.usi.ch/index.php>)



## 4.5 trying different ML methods cross same projects’ different versions and describe the result

## 5 Conclusions

## 6 References

[1] Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project, 7007(011)*, 429-489.

[2] Akiyama, F. . (1971). An example of software system debugging. In IFIP Congress (1, 353-359.

[3] Kamei, Y. , & Shihab, E. . (2016). Defect Prediction: Accomplishments and Future Challenges. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE.

[4] Tantithamthavorn, C. , Mcintosh, S. , Hassan, A. E. , & Matsumoto, K. . (2016). [acm press the 38th international conference - austin, texas (2016.05.14-2016.05.22)] proceedings of the 38th international conference on software engineering - icse \"16 - automated parameter optimization of classification techniques for defect prediction models. 321-332.

[5] Zoltán Tóth, Péter Gyimesi, & Ferenc, R. . (2016). A Public Bug Database of GitHub Projects and Its Application in Bug Prediction. International Conference on Computational Science & Its Applications.

[6] Pan, K. , Kim, S. , & Jr, E. J. W. . (2006). Bug Classification Using Program Slicing Metrics. IEEE International Workshop on Source Code Analysis & Manipulation. IEEE.

[7] Xia, X. , Lo, D. , Wang, X. , Yang, X. , Li, S. , & Sun, J. . (2013). A Comparative Study of Supervised Learning Algorithms for Re-opened Bug Prediction. European Conference on Software Maintenance & Reengineering. IEEE Computer Society.

[8] Zhang, F. , Zheng, Q. , Zou, Y. , & Hassan, A. E. . (2016). [ACM Press the 38th International Conference - Austin, Texas (2016.05.14-2016.05.22)] Proceedings of the 38th International Conference on Software Engineering - ICSE \"16 - Cross-project defect prediction using a connectivity-based unsupervised classifier. (pp.309-320). IEEE.

[9] Couto, C. , Pires, P. , Valente, M. T. , Bigonha, R. S. , & Anquetil, N. . (2014). Predicting software defects with causality tests. Journal of Systems and Software, 93, 24-41.

[10] Osman, H. , Ghafari, M. , & Nierstrasz, O. . (2017). Hyperparameter Optimization to Improve Bug Prediction Accuracy. MaLTeSQuE. IEEE.

[11] Kamei, Y. , Matsumoto, S. , Monden, A. , Matsumoto, K. I. , Adams, B. , & Hassan, A. E. . (2010). [IEEE 2010 IEEE 26th International Conference on Software Maintenance (ICSM) - Timi oara, Romania (2010.09.12-2010.09.18)] 2010 IEEE International Conference on Software Maintenance - Revisiting common bug prediction findings using effort-aware models. Proc of IEEE International Conference on Softw Maintenance (pp.1-10).

[12] Shivaji, S. , Whitehead, E. J. , Akella, R. , & Kim, S. . (2009). Reducing Features to Improve Bug Prediction. IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society.

[13] Zimmermann, T. , Premraj, R. , & Zeller, A. . (2011). Predicting defects for eclipse [revised for dataset version 2.0a]. Proc International Workshop on Predictor Models in Software Engineering, 9.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |