A Project Report

On

Software Defect Prediction

BY

R Adarsh

2019A7PS0230H

Under the supervision of

Dr. N L Bhanu Murthy

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F266: STUDY PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(December, 2021)

ACKNOWLEDGMENTS

I would like to thank Dr. N L Bhanumurthy and Dr. Lov Kumar for their constant support and guidance throughout the duration of the project. This would not have been possible without their insights into the project area and their direction for the research.

I would also like to extend my gratitude to my collaborators Rohan Maheshwari and Pratyush Banerjee for their help and contributions.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Software Defect Prediction**" submitted by Mr. R ADARSH (ID No. 2019A7PS0230H) in partial fulfillment of the requirements of the course CS F266, Study Project Course, embodies the work done by him under my supervision and guidance.

Date:

(Dr. N L Bhanu Murthy)

BITS- Pilani, Hyderabad Campus

ABSTRACT

Software defect prediction techniques aim to find defect prone modules in the early stage of the Software Development Lifecycle(SDLC). This helps in quality checking the software and reduces cost incurred during the testing phase of software development. Network embeddings are used to transform nodes, edges and their features into low-dimensional vector spaces while preserving the structural information of graphs. In recent years, network embeddings have been studied to solve problems in the areas of social networks and biomolecules. They have proven to be useful in tasks such as link prediction and multi-label classification. In this paper, we study the effectiveness of network embeddings for predicting defects in software systems. We use network embeddings in combination with traditional object-oriented metrics in an attempt to create better models for defect prediction. We use 6 different kinds of network embedding techniques, namely ProNE, node2vec, LINE, HOPE, GraRep and DeepWalk. We evaluate our models on 10 open source projects. The experimental results demonstrate that network embeddings significantly improve the performance of defect prediction models which use only traditional software engineering metrics.

CONTENTS

1. INTRODUCTION

Most of the organizations and companies want to predict the defects in the code or identify the fault-prone modules in software systems before deployment. Software quality is one of the most critical elements in user functionality. Machine learning techniques/models can help in predicting faults and hazard analysis and thereby improving the code quality. This study aims to try and identify some software metrics/features and approaches that are useful in classifying a code module as defective or not. Based on the literature review, we have identified a consolidated set of metrics that could be used in software defect detection tasks. The comprehensive set of metrics like complexity metrics , object-oriented metrics etc. takes into account the information from different aspects of source code which could act as useful features for our model.

Network embeddings on the other hand have not seen much use in this domain. Notably Qu et al 2018 have done some research using a network embedding approach to encode classes and try to use these embeddings to improve upon the existing software defect prediction models. Here we have aggregated all the network embeddings that can be useful in software defect prediction and have analysed the performance of network embeddings vs traditional metrics and also observed the results of the combined model on several tera promise software engineering repositories.

As a part of our project work, we have also studied the effectiveness of network embeddings for predicting defects in software systems. We use network embeddings in combination with traditional object-oriented metrics in an attempt to create better models for defect prediction.
The experimental results demonstrate that network embeddings significantly improve the performance of defect prediction models which use only traditional software engineering metrics.

2. SOFTWARE METRICS

A considerable amount of work has been done in the field of software metrics and defect prediction. However, it is a difficult task to select the right set of metrics that would be effective in software defect prediction. Marian Jureczko(2011) tries to study the importance of various metrics in defect prediction and perform a correlation analysis to understand the relation between number of defects and software metrics. We classified our study in 2 categories namely static code components and change metrics. We further also investigated the use of network metrics and embeddings for our task. The following metrics were analyzed:

2.1. Product Metrics

● **CK Metrics :**

1. Weighted Methods per Class (WMC) : Number of methods in a class (assuming unit weight for each method)

2. Depth of Inheritance tree (DIT): Number of inheritance levels in the hierarchy .

3. Number of Children (NOC) : Number of direct descendants of a class

4. Coupling between object classes (CBO): Number of classes coupled to given class (method calls, field access, inheritance etc.)

5. Response for a class (RFC): Sum of number of methods called within a class and the number of methods present in class.

6. Lack of Cohesion of methods(LCOM): (No of pairs of member functions without shared instance variables) - (No of pairs of member functions with shared instance variables)

● **Martin's metrics** :

1. Afferent Coupling (Ca) : Number of classes depending on measured class.

2. Efferent Coupling (Ce) : Number of classes the measured class depends on.

● **McCabe's Cyclomatic complexity**:

1. Data Complexity : Number of independent paths through data logic

2. Cyclomatic Complexity: Number of linearly independent paths. Useful to estimate the complexity of a module.

● **Halstead metrics** : These metrics were published in 1977 and their ongoing use is a testament to their strength.

- Program Length
- Program Volume
- Program Level and Program Difficulty
- Intelligent Content

We also included the very simple metrics like Lines of Code (LOC), Number of global variables, functions, parameters. Fan In and Fan Out which indicate number of calling and callee classes/methods.

2.2. Process/Change Metrics

Change Metrics entail how a program has changed over time. Below we discuss some of the more advanced and popular change metrics that have been published over the years.

● Revisions count

● Distinct committers count

● Number of modified lines(NML)

● Number of defects in previous version(NDPV)

● **Change Burst Metrics**: Considers minimum distance between changes and minimum number of changes in burst. Change bursts are determined by two parameters:

- Gap size - minimum distance between two changes
- Burst size - minimum number of changes in a burst

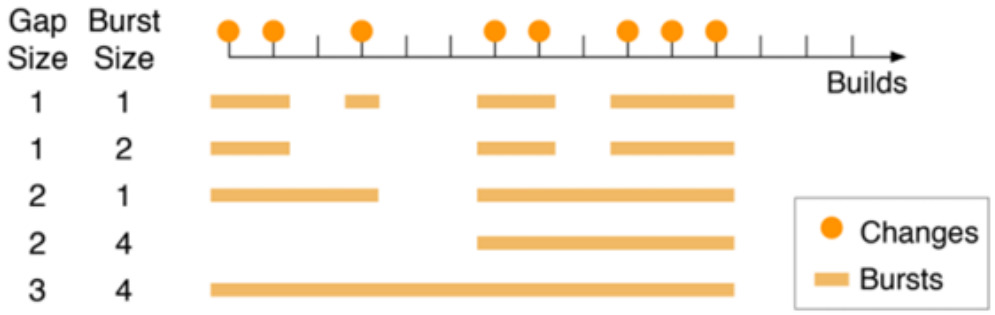


Figure 1. How gap size and burst size determine change burst detection from a sequence of changes.

The most important metrics that have been discussed in this area are:

a. Change Metrics

1. Number of changes
2. Number of consecutive changes
3. Number of change bursts
4. Early and late metrics

b. Temporal Metrics

1. Time of first burst
2. Time of last burst
3. Time of max burst

c. People Metrics

1. Total people
2. Total people in burst
3. Max people in burst

● **Code churn metrics** : Measures the changes taking place in a software module over a period of time (ex. files churned/filecount , weeks of churn/files churned etc.). It can be easily extracted from a software's change/version history.

Relative Measures have been proven to perform better than Absolute Measures hence we would only be focusing on them. It is important to note that these measures are calculated with respect to Binaries i.e., versions of a software. The measures in consideration are:

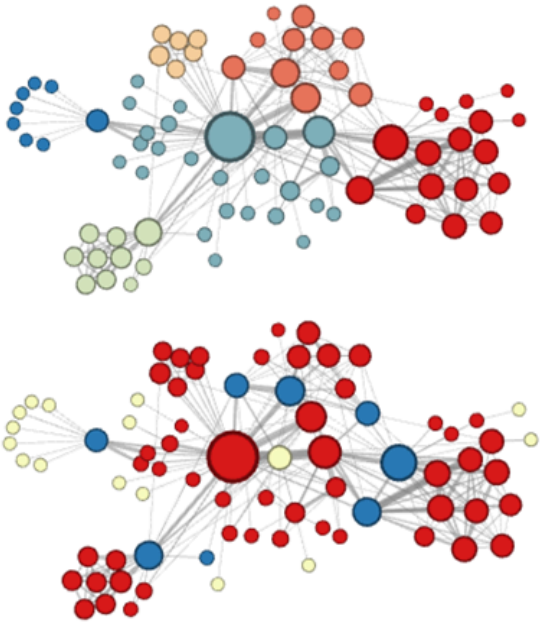
1. Churned LOC divided by Total LOC
2. Deleted LOC divided by Total LOC
3. Files churned divided by File count
4. Churn count divided by Files churned
5. Weeks of churn divided by File count
6. Lines worked on divided by Weeks of churn
7. Churned LOC divided by Deleted LOC
8. Lines Worked On divided by Churn Count.

2.3. Network Metrics

Zimmerman and Nagappan proposed some network metrics for software defect prediction with the motivation behind them being to see how dependencies correlate and predict defects. They proved that the metrics they proposed are indeed helpful in identifying defects.They carried out their study at the node level and the global level and proposed various metrics for the same.

Simple metrics like Size, Ties, Pairs etc. were coupled with more advanced metrics like ReachEfficiency and Brokerage to constitute the metrics at the node level subnetwork. They also defined metrics based on Structural Holes concept which gives us an indication of which nodes are at an advanced position over other nodes (more paths have to traverse through them).

Some of the metrics were constituted around Centrality which identifies the binaries that are specially exposed to dependencies, like by being the target of many dependents. Degree, Closeness and Betweenness are some of the ways we can numerically view this similarity.



3. Network Embedding Techniques

Recent studies in the field of representational learning for NLP have shown new methods of feature learning. One such powerful model is the Skip-gram model, which focuses on learning the continuous feature representations of words (considered discrete objects) by optimizing the likelihood function using SGD with negative sampling. This model has inspired recent researchers to establish the analogy between networks and documents.

3.1. ProNE

ProNE is a relatively faster model to construct large scale network embeddings. The idea behind prone is to first initialize embeddings in an efficient manner and then enhance the power of these embeddings It uses sparse matrix factorization to efficiently achieve initial node representations. In this technique we first express every node as a linear combination of its neighbours and then we minimize the distance between the vector representing a given node in the embedded space, and the weighted sum of its neighbours embeddings. Then it utilizes higher orders of the Cheeger's inequality to modulate the network's spectral space and propagate the learned embeddings in the modulated network. This will help understand the community structure.

3.2. DeepWalk

Deepwalk is a model that aims to learn latent representation of nodes in a social network. For each node, random walks of fixed length are used to convert the non-linear graph structure into linear vector representation which can be used by statistical models. The distance between latent dimensions should represent a metric for evaluating social similarity between corresponding nodes in the network.

Since this method is not allowed to have a global view of the network, we can see why it is not performing as well as the other models because global properties like structural equivalence are not being recognised by this model.

3.3. Node2vec

Node2vec algorithm learns the feature representation for nodes in a graph network and generates the corresponding vector embeddings. The main intuition behind the node2vec algorithm is that random walks in a graph network can be considered as sentences in a document. Sampling sequences of nodes from the network generates an ordered sequence of nodes analogous to sentences in a document. However, the previous studies fail to provide any kind of flexibility in node sampling from the network. The node2vec algorithm tackles this shortcoming by providing tunable parameters for the search space that has been explored instead of relying on some specific sampling strategy. Node2vec uses a flexible neighborhood strategy that involves a trade-off between how deep we go in the random walk. This is achieved by using a second-order biased random walk method that explores the neighborhood in a DFS as well as BFS fashion. These two search strategies present two different scenarios of exploration namely - homophily (belonging to the same community) and structural equivalence(having the same neighbourhood). The neighborhoods traversed by BFS generates embeddings closely corresponding to structural equivalence property while the DFS can explore greater parts of the network which helps in identifying communities based on homophily .

Grover & Leskovec(2016) showed that the node2vec is scalable and more robust to perturbations like missing edges, noisy edges etc. Their experiments also confirmed that the overhead due to learning the best search parameters is minimal for node2vec and the semi-supervised algorithm is capable of learning the parameters efficiently even with very less labeled data.

3.4. LINE

Large-Scale Information Network Embedding("LINE") is a novel embedding approach that addresses the problem of mapping the high-dimensional information graph networks to a low-dimension vector embedding in such a way that the semantics present in the original graph are preserved. The LINE model utilizes carefully designed objective functions that preserve information about local as well as global structures. Local network structures are described using the first-order proximity (local pairwise proximities) between vertices. The second-order proximity (similarity between neighborhood network components of vertices) is complementary to the first order proximities and preserves information regarding network structure.

An efficient edge-sampling technique is used for optimization instead of the classic stochastic gradient descent optimization. For very large networks, it is not practical to directly use SGD because of weighted edges which cause high variance. We may run into the exploding gradient problem and degrade the performance of the model. To overcome this shortcoming, the edge sampling method is used in which the edges are sampled in accordance with probabilities

proportional to the edge-weights and then treating these sampled edges as binary edges for updating the model.

The LINE model is a fairly robust model in terms of scalability for very large information networks and generalizes well for graphs with arbitrary edge types : directed/undirected, weighted/unweighted etc. It is one of the early models developed to generate the network embeddings.

3.5. HOPE

HOPE is a matrix factorization based spectral method. Unlike many graph embedding methods, it preserves symmetric transitivity, which is a vital property of directed graphs. Asymmetric transitivity describes the correlation between directed edges i.e. there is a large likelihood of a directed edge from u to v if there is a directed path from u to v. Asymmetric transitivity may aid in capturing graph structures.

HOPE approximates high-order proximity by formulating the original problem of factorizing the Katz matrix as a generalized SVD problem. This is to avoid the computationally expensive calculation of the proximity matrix. Therefore, it leads to a scalable embedding algorithm for large graphs.

3.6. GraRep

GraRep is a matrix factorization based method for learning network embeddings. It aims to incorporate global structural information of the graph into the process of learning a graph representation. By preserving different k-step relational information in distinct subspaces, it addresses a drawback of the skip-gram model, which has been used in works like DeepWalk, that projects all k-step relational information into a common subspace.

The GraRep algorithm first gets the k-step probability transition matrices for different values of k. Then it gets each k-step representation using a spectral dimensionality reduction technique, namely Singular Value Decomposition (SVD). Finally, it concatenates all k-step representations to form a global representation. Thus, it exploits k-step relational information that uncovers global structural information of the graph. This comes with the limitation of expensive computation due to the high time complexity for computing the power of a matrix and SVD.

4. Methodology

4.1. Data preprocessing

Class Dependency Network was obtained from the source code using SciTools Understand. This was used to create an edge list and a node mapping. The edge list was used as an input for the network embedding techniques. The node mapping was used to combine the defect data with the network embedding data.

4.2. Models

For our experiments, we developed three kinds of models:

1. Using only traditional object-oriented metrics such as WMC (Weighted Methods per Class), CBO (Coupling between object classes) and LCOM (Lack of Cohesion on Methods).
2. Using only network embeddings obtained from 6 methods: ProNE, DeepWalk, node2vec, LINE, HOPE and GraRep.
3. Combining traditional metrics and network embeddings by concatenating the features.

4.3. Classification

We used the following classifiers in our experiments:

1. **Multinomial Naive Bayes**
   Multinomial Naive Bayes is a variant of the Naive Bayes classifier. It is based on the Bayes theorem. It assumes that features are generated by a multinomial distribution. Since the multinomial distribution reflects the likelihood of counts in a range of categories, Multinomial Naive Bayes is best suited for features that represent frequencies.

2. **Logistic Regression**
   Logistic Regression is a popular algorithm for classification tasks. It uses the binary logistic function and uses gradient descent to find the optimal value.

3. **Gaussian Naive Bayes**
   Gaussian Naive Bayes is another variant of the Naive Bayes. It is based on the assumption that the data from each label is generated by a normal distribution.

4. **Decision Tree**
   Decision Tree represents the classification task as a tree-like structure, with nodes formed depending on the feature values from the training data. Gini values are used to determine the splitting of nodes such that each feature is taken into account.

5. RESULTS

Table 1. Accuracy and AUC for 3 models with 6 embeddings

	Accuracy				AUC			
	MNB	**LOG**	**GNB**	**DT**	**MNB**	**LOG**	**GNB**	**DT**
	ProNE							
OOD	**0.802168**	**0.814363**	**0.803523**	**0.768293**	**0.790322**	**0.806722**	**0.808102**	**0.671696**
EMB	**0.776423**	**0.783198**	**0.617886**	**0.737127**	**0.644846**	**0.710429**	**0.663451**	**0.619366**
BOTH	**0.806233**	**0.838753**	**0.802168**	**0.802168**	**0.807097**	**0.843244**	**0.819657**	**0.725877**
	node2vec							
OOD	**0.802168**	**0.814363**	**0.803523**	**0.764228**	**0.790322**	**0.806722**	**0.808102**	**0.651817**
EMB	**0.776423**	**0.775068**	**0.696477**	**0.703252**	**0.589217**	**0.674763**	**0.701301**	**0.582342**
BOTH	**0.807588**	**0.834688**	**0.807588**	**0.787263**	**0.803469**	**0.846793**	**0.82374**	**0.703332**
	LINE							
OOD	**0.802168**	**0.814363**	**0.803523**	**0.754743**	**0.790322**	**0.806722**	**0.808102**	**0.641393**
EMB	**0.776423**	**0.775068**	**0.762873**	**0.657182**	**0.533397**	**0.576879**	**0.537707**	**0.515992**
BOTH	**0.804878**	**0.813008**	**0.806233**	**0.739837**	**0.800613**	**0.813189**	**0.805992**	**0.642583**
	HOPE							
OOD	**0.802168**	**0.814363**	**0.803523**	**0.768293**	**0.790322**	**0.806722**	**0.808102**	**0.660908**
EMB	**0.776423**	**0.779133**	**0.742547**	**0.696477**	**0.53007**	**0.699413**	**0.671209**	**0.574272**

BOTH	**0.804878**	**0.830623**	**0.787263**	**0.787263**	**0.800487**	**0.812164**	**0.754048**	**0.681755**
	GraRep							
OOD	**0.802168**	**0.814363**	**0.803523**	**0.773713**	**0.790322**	**0.806722**	**0.808102**	**0.670871**
EMB	**0.776423**	**0.784553**	**0.686992**	**0.726287**	**0.654598**	**0.723555**	**0.711323**	**0.60367**
BOTH	**0.806233**	**0.833333**	**0.810298**	**0.769648**	**0.807224**	**0.829996**	**0.821831**	**0.679042**
	DeepWalk							
OOD	**0.802168**	**0.814363**	**0.803523**	**0.773713**	**0.790322**	**0.806722**	**0.808102**	**0.664398**
EMB	**0.776423**	**0.775068**	**0.609756**	**0.689702**	**0.571056**	**0.688006**	**0.659813**	**0.556354**
BOTH	**0.804878**	**0.840108**	**0.798103**	**0.771003**	**0.808504**	**0.850161**	**0.805727**	**0.658337**

CONCLUSION

In this paper, we discussed the different traditional metrics which could be useful defect-indicators in software systems. We looked at the metrics in broad categories called product metrics , process metrics and network metrics . Further the effectiveness of network embeddings for software defect prediction was discussed. It was found that using network embeddings alone for the defect prediction task yields worse results compared to using just traditional object-oriented metrics. However, when network embedding techniques are used in combination with the traditional metrics, we obtain better results. Thus, network embedding techniques improve the performance of defect prediction models which only use traditional object-oriented metrics.

REFERENCES:

1. K. Punitha and S. Chitra, "Software defect prediction using software metrics - A survey," *2013 International Conference on Information Communication and Embedded Systems (ICICES)*, 2013, pp. 555-558, doi: 10.1109/ICICES.2013.6508369.
2. Jureczko, Marian. (2011). Significance of Different Software Metrics in Defect Prediction. Software Engineering: An International Journal. 1. 86-95.
3. Cao, S., Lu, W., & Xu, Q. (2015). GraRep: Learning Graph Representations with Global Structural Information. Proceedings of the 24th ACM International on Conference on Information and Knowledge Management.
4. Grover, A., & Leskovec, J. (2016). node2vec: Scalable Feature Learning for Networks. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
5. Ou, M., Cui, P., Pei, J., Zhang, Z., & Zhu, W. (2016). Asymmetric Transitivity Preserving Graph Embedding. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
6. Qu, Y., Liu, T., Chi, J., Jin, Y., Cui, D., He, A., & Zheng, Q. (2018). node2defect: Using Network Embedding to Improve Software Defect Prediction. 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 844-849.
7. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). LINE: Large-scale Information Network Embedding. Proceedings of the 24th International Conference on World Wide Web.
8. Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: online learning of social representations. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.
9. Tim Menzies, Rahul Krishna, and David Pryor. 2015. The Promise Repository of Empirical Software Engineering Data. http://openscience.us/repo. North Carolina State University, Department of Computer Science.
10. Zhang, J., Dong, Y., Wang, Y., Tang, J., & Ding, M. (2019). ProNE: Fast and Scalable Network Representation Learning. IJCAI.
11. S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object-Oriented Design,” IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476-493,1994
12. Gyimothy et al, Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,2005

13. Thomas J McCabe, “A Complexity Measure”, IEEE Transactions on Software Engineering, vol. se-2, no. 4, December 1976

14. Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, and Brendan Murphy. 2010. Change bursts as defect predictors. (ISSRE). IEEE, IEEE Computer Society, Washington, DC, USA, 309–318.

15. Nagappan and Ball, Use of Relative Code Churn Measures to Predict System Defect Density, ICSE’05