Automatically Learning Semantic Features for Defect Prediction

Akshay Nalwaya
North Carolina State University
analway@ncsu.edu

***Abstract*— Existing traditional software defect prediction models often fail to capture the semantic differences of programs, and such a capability is needed for building accurate prediction models. To bridge the gap between programs semantics and defect prediction features, this paper proposes to leverage a powerful representation-learning algorithm, deep learning, to learn semantic representation of programs automatically from source code. Specifically, we leverage Deep Belief Network (DBN) to automatically learn semantic features from token vectors extracted from programs Abstract Syntax Trees (ASTs). Our evaluation on ten open source projects shows that our automatically learned semantic features significantly improve both within-project defect prediction (WPDP) and cross-project defect prediction (CPDP) compared to traditional features. Our semantic features improve WPDP on average by 14.7% in precision, 11.5% in recall, and 14.2% in F1. For CPDP, our semantic features based approach outperforms the state-of-the-art technique TCA+ with traditional features by 8.9% in F1.**

I. INTRODUCTION

Software defect prediction techniques have been proposed to detect defects and reduce software development costs. Defect prediction techniques build models from software data, and use the models to predict whether new instances of code regions, e.g., files, changes, and methods, contain defects. Efforts of previous studies towards building accurate prediction models fall into two main directions: one is manually designing new features or new combinations of features to represent defects more effectively; the other is using new and improved machine learning algorithms. To bridge the gap between programs semantic information and features used for defect prediction, this paper proposes to leverage deep-learning to learn semantic representation of programs automatically and use the representation to improve defect prediction. Specifically, we use Deep Belief Network (DBN) to automatically learn features from token vectors extracted from programs ASTs, and then utilize these features to train a defect prediction model.

RQ1: Do semantic features outperform traditional features for within-project defect prediction?

RQ2: Do semantic features outperform traditional features for cross-project defect prediction?

RQ3: What is the time and space cost for the proposed DBN-based feature generation process?

II. BACKGROUND

A. Defect Prediction

In traditional approaches, the first step for defect prediction is to label data as buggy or clean based on post-release defects for each file. A file is buggy if the file contains bugs. Otherwise, the file is clean. The second step is to collect corresponding traditional features of these files. Instances with features and labels are used to train machine learning classifiers. Finally, trained models are used to predict new instances as buggy or clean.

```
1 int i = 9;
2 if (i == 9) {
3   foo();
4   for (i = 0; i < 10;
        i++) {
5     bar();
6   }
7 }
```

File1.java

```
1 int i = 9;
2 foo();
3 for (i = 0; i < 10; i
      ++) {
4   if (i == 9) {
5     bar();
6   }
7 }
```

File2.java

Fig. 1. A Motivating Example

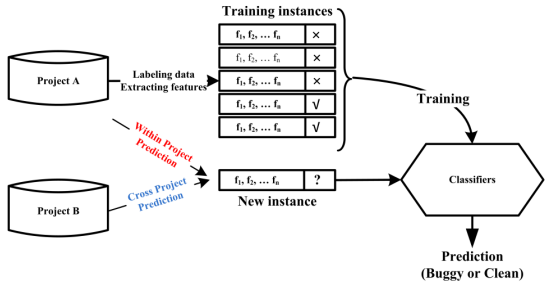

Fig. 2. Defect Prediction Process

B. Deep Belief Network

A Deep Belief Network is a generative graphical model that uses a multi-level neural network to learn a representation from training data that could reconstruct the semantic and content of input data with a high probability. DBN contains one input layer and several hidden layers, and the top layer is the output layer that used as features to represent input data. Each layer consists of several stochastic nodes. The size of learned semantic features is the number of nodes in the top layer. The idea of DBN is to enable the network to reconstruct the input data using generated features by adjusting weights between nodes in different layers.

III. APPROACH

The approach consists of four major steps: 1) parsing source code into tokens, 2) mapping tokens to integer identifiers, which are the expected inputs of DBN, 3) leveraging DBN to automatically generate semantic features, and 4) building defect prediction models and predicting defects.

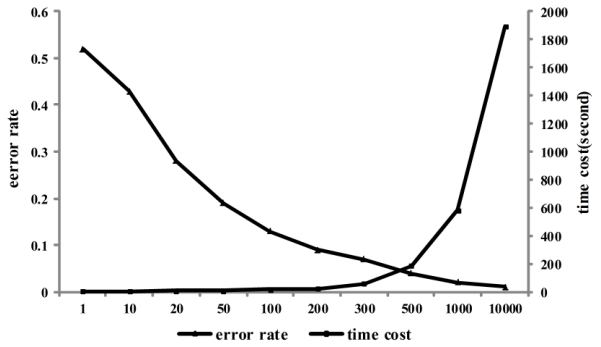

Fig. 3. Average error rate and time cost for different numbers of iterations

Project	Version (Tr->T)	Naive Bayes		Logistic Regression	
		Semantic	PROMISE	Semantic	PROMISE
ant	1.5->1.6	**63.0**	56.0	**91.6**	50.6
	1.6->1.7	**96.1**	52.2	**92.5**	54.3
camel	1.2->1.4	**45.9**	30.7	**59.8**	36.3
	1.4->1.6	**48.1**	26.5	34.2	**34.6**
jEdit	3.2->4.0	**58.3**	48.6	**55.2**	54.5
	4.0->4.1	**60.9**	54.8	**62.3**	56.4
log4j	1.0->1.1	**72.5**	68.9	**68.2**	53.5
lucene	2.0->2.2	**63.2**	50.0	**63.0**	59.8
	2.2->2.4	**73.8**	37.8	62.9	**69.4**
xalan	2.4->2.5	**45.2**	39.8	**56.5**	54.0
xerces	1.2->1.3	**38.0**	33.3	**47.5**	26.6
ivy	1.4->2.0	34.4	**38.9**	**34.8**	24.0
synapse	1.0->1.1	47.9	**50.8**	**42.3**	31.6
	1.1->1.2	**57.9**	56.5	**54.1**	53.3
poi	1.5->2.5	**77.0**	32.3	**66.4**	50.3
	2.5->3.0	**77.7**	46.2	**78.3**	74.5
Average		**60.0**	45.2	**59.7**	49.0

Fig. 4. Comparison of F1 scores between semantic features and PROMISE features.

A. *Parsing Source Code*

Java Abstract Syntax Tree (AST) is used to extract syntactic information from source code. Three types of AST nodes are extracted as tokens: 1) nodes of method invocations and class instance creations, 2) declaration nodes, and 3) control-flow nodes.
In summary, for each file, we obtain a vector of tokens of the three categories.

B. *Handling Noise and Mapping Tokens*

To prune noisy data, kNN based approach is used. It identifies the k-nearest neighbors for each instance and examines the labels of its neighbors. If a certain number of neighbors have opposite labels, the examined instance will be flagged as noise. Post this, the tokens are mapped to some integer values to be given to the predictive model.

C. *Training DBN and Generating Features*

For simplfying the model, number of nodes were kept same in each layer. For each node, DBN learns probabilities of traversing from this node to the nodes of its top level. DBN requires the values of input data ranging from 0 to 1, values in the data vectors of the training and test sets are normalized by using min-max normalization. Once the model is tuned completely, semantic features for the training and test data from the output layer of the DBN are obtained.

D. *Building Models and Performing Defect Prediction*

The generated semantic features from the DBN are used and then test data is used to evaluate the performance of the built defect prediction models.

IV. RESULTS

A. ***RQ1: Do semantic features outperform traditional features for within-project defect prediction?***

The proposed DBN-based approach is effective in automatically learning semantic features, which improves the performance of within-project defect prediction. The semantic features automatically learned from DBN improve within-project defect prediction and the improvement is not tied to a particular classification algorithm.

B. ***RQ2: Do semantic features outperform traditional features for cross-project defect prediction?***

Our proposed DBN-CP improves the performance of cross-project defect prediction. The semantic features learned by DBN are effective and able to capture the common characteristics of defects across projects.

C. ***RQ3: What is the time and space cost for the proposed DBN-based feature generation process?***

Among all the projects, the time cost of automatically generating semantic features varies from 8.0 seconds (ivy) to 32.0 seconds (camel). As for the memory space cost, it takes less than 6.5MB for all the examined projects. Using our proposed DBN-based approach to automatically learn semantic features is applicable in practice.

V. CONCLUSION

Our evaluation on ten open source projects shows that the automatically learned semantic features could significantly improve both within-project and cross-project defect prediction compared to traditional features. Our semantic features improve the within-project defect prediction on average by 14.7% in precision, 11.5% in recall, and 14.2% in F1 comparing with traditional features. For cross-project defect prediction, our semantic features based approach improves the state-of-the-art technique TCA+ built on traditional features by 8.9% in F1.

VI. REFERENCES

[1] Automatically Learning Semantic Features for Defect Prediction by Song Wang, Taiyue Liu and Lin Tan.