Neural Representation Learning Based Binary Code Authorship Attribution

Zhongmin Wang, Zhen Feng, and Zhenzhou Tian(✉)

Xi'an University of Posts and Telecommunications, Xi'an, China
{zmwang,tianzhenzhou}@xupt.edu.cn

Abstract. Authorship attribution on binary code is of great value in applications such as malware analysis, software forensics, and code theft detection. Inspired by the recent great successes of neural network and representation learning in various program analysis tasks, this study proposes NMPI to achieve fine-grained program authorship attribution by analyzing the binary codes of individual functions from the perspective of sequence and structural. To evaluate the NMPI, the study constructs a large dataset consisting of 268796 functions collected from Google CodeJam. The extensive experimental evaluation shows that NMPI can achieve 91% accuracy for the function-level binary code authorship attribution task.

Keywords: Authorship attribution · Binary code · Neural network · Representation learning

1 Introduction

The authorship attribution has extracted a set of features from the code to represent the programmer's code style to identify the anonymous programmer. In recent years, relatively few works have been conducted on authorship attribution, which mainly falls into two categories: authorship attribution on source code [1,2,6] and authorship attribution on binary code [4]. The former is often difficult to obtain source code in actual situations. The latter, the accuracy of these methods, much depends on the quality of manually-crafted feature extraction strategies. In this paper, we attempt to adopt some of the most popular representation learning algorithms to achieve fast and accurate fine-grained programmer identification on function level.

Supported by National Natural Science Foundation of China (61702414), the Natural Science Basic Research Program of Shaanxi (2018JQ6078), the Science and Technology of Xi'an (2019218114GXRC017CG018-GXYD17. 16) and the International Science and Technology Cooperation Program of the Science and Technology Department of Shaanxi Province, China (Grant No. 2019KW-008) and Science and Technology Project in Shaanxi Province of China (Program No. 2019ZDLGY07-08.).

S. Goel et al. (Eds.): ICDF2C 2020, LNICST 351, pp. 244–249, 2021.
https://doi.org/10.1007/978-3-030-68734-2_15

Our contributions are summarized as follows:

- We propose to reveal fine-grained programmer code style details for individual functions by designing a lightweight function abstraction strategy and adapting typical sequence-oriented and structure-oriented model NMPI (**N**eural **M**odeling based **P**rogrammer **I**dentification).
- We use clustering to treat programmer with similar coding styles as a class, which alleviates the impact on results when there are too many programmer.
- We evaluated NMPI performance of revealing the programmers on a large dataset consisted of 268796 unique functions that we constructed via Google Code Jam. The experimental results show that the recognition rate of NMPI for 50 programmers is over 91%.

The rest of our work is structured as follows. Section 2 describes the model. Section 3 reports the experimental results. Section 4 concludes the paper.

2 The Approach

Our approach overall architecture is as follows (See Fig. 1). After disassembling the binary code to extract the function information, it is sent to our model NMPI to identify the programmers.

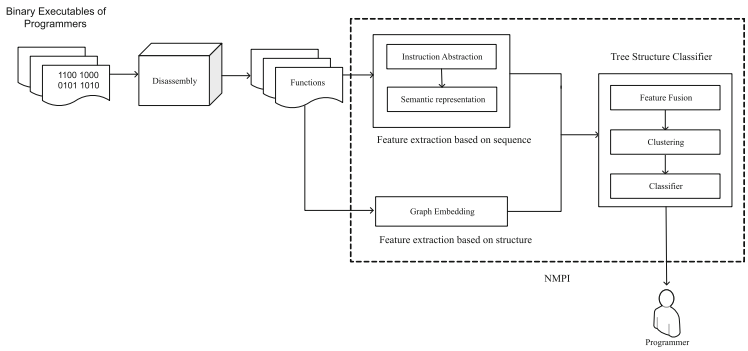

Fig. 1. The figure shows the overall framework for identifying programmer.

2.1 Feature Extraction Based on Structure

Deepwalk [7] is the most common graph embedding algorithm. We will use Deepwalk to extract the structure information of the binary code (See Fig. 2). Convert the control flow graph of function to a directed graph $G = (B, E)$. Get multiple paths by random walking. One of the paths is expressed as $path = \{B_1, B_3, B_7, B_9, ..., B_L\}$. Treat this obtained path use word2vec to learn, and get the structural representation vector VB_i^{T} of the basic block. The basic block node vector under each function is passed through the average pooling layer to obtain the structure-based feature V_f^T of the function f.

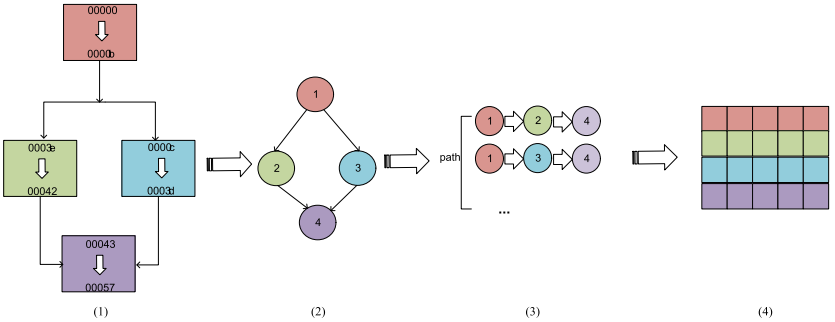

Fig. 2. The figure shows the extract structure features of binary code using Deepwalk.

2.2 Feature Extraction Based on Sequence

Instruction Abstraction. Firstly, we need to represent a function as $f = \{B_1, B_2, B_3, ..., B_n\}$, where n denotes the number of basic blocks within the function. We formulated the following rules for instruction abstraction. Firstly, the mnemonics remain unchanged. Secondly, all registers in the operands remain unchanged. Thirdly, all base memory addresses in the operands are substituted with the symbol MEM. Finally, all immediate values in the operands are substituted the symbol IMM.

As an example, with the above abstraction rules, the instruction "$add\ eax, 6$" will become "$add\ eax, IMM$", the instruction $mov\ ebx, [0\times3435422]$ will become "$mov\ ebx, MEM$".

Semantic Representation. Given a set of abstracted assembly instruction sequences, it is promising to utilize skip-gram [5] to learn the embedding for each instruction. LSTM [3] is known to learn the sequential dependency. In this work, we employ Long Short-term Memory (LSTM) to address the vanishing and exploding gradient issue (See Fig. 3). Each instruction in the input sequence embedded in vector space. Afterwards, the model reads the input instruction sequence through LSTM units to obtain the basic block vector $VB_i^S = O(H(i-1)\times ins_i)$. Finally, each basic block node vector under the function sends a pooled average layer to obtain the function's sequence-based features.

2.3 Tree Structure Classifier

Tree structure classifier framework is divided into three parts (See Fig. 4).

Feature Fusion. We obtain the sequence-based and structure-based features of the function through the above two feature extraction methods. In order to consider the programmer's coding style from multiple perspectives, we fuse the two features. For example, a function extracts sequence-based features as V_f^S ,

structure-based features as V_f^T. We get the feature expression of the function $V_f = V_f^S \oplus V_f^T$ through feature fusion. The dimension of V_f depends on the dimension of V_f^S and V_f^T. Finally, the sequence V_f^S and structural features V_f^T of the function V_f are included.

Clustering. Firstly, we use $L_a = \{V_{f1}, V_{f2}, V_{f3}, ..., V_{fn}\}$ to represent all the feature vectors of a label a in the training set. These vectors form a matrix $R^{n \times k}$, where n is the number of functions under label a in the training set. Through average pooling layer get the matrix $V_a^{1 \times k}$ into a representative vector label a.

Secondly, we use K-means to cluster the representative vectors. The number of clusters is set to 2 categories, and the reason will be discussed in Section IV. Construct two classifiers based on A and B categories. When we input the code snippet, we first determine which category it belongs to, and then input it into the classifier under the corresponding category for classification. This avoids adverse effects on the results when there are too many labels.

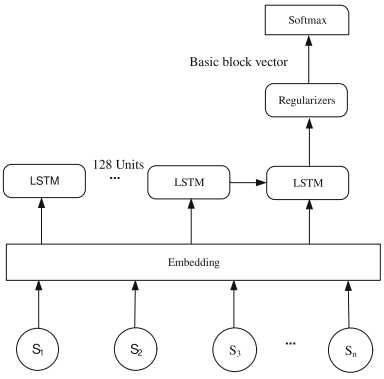

Fig. 3. LSTM model structure

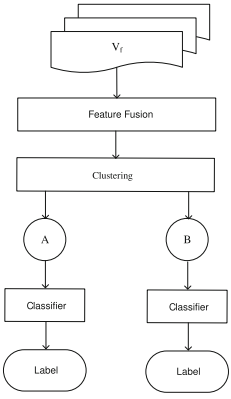

Fig. 4. Overview of tree structure classifier framework implementation

3 Experimental Evaluation

3.1 Dataset

To evaluate the performance of NMPI, we follow the google code jam used in previous authorship attribution work as the data set, collect C++ code written by 191 programmers from Google Code Jam, it consists of 13112 programs and 268796 functions.

3.2 Evaluation

The Results of NMPI's Use of Different Classifiers to Authorship Attribution. We use DNN, SVM, and Random forest for comparison experiments, and the results are shown in Table 1. We observe that DNN has the highest accuracy. The reason for the DNN network can adjust its structure according to the task and perform automatic learning.

Table 1. Experimental result.

Classifier	Top-1	Top-5	Precision	Recall
DNN	91.8%	96%	91.3%	91.6%
SVM	87.9%	94.3%	87.1%	87.3%
Random forest	84.5%	92.3%	84.4%	84.6%

Comparision with Other Programmer Identification Techniques. Rosenblum et al. [8] achieved 70% accuracy among 50 programmers and 51% accuracy among 191 programmers. We compare the results of NMPI programmer identification with Rosenblum as in Table 2. The result showing that we have improved accuracy.

Table 2. Comparison with existing methods.

The Approach	Number of Programmer	Accuracy	Classifier
Rosenblum [8]	50	70%	SVM
NMPI	50	86%	SVM
NMPI	50	91%	DNN
Rosenblum [8]	191	51%	SVM
NMPI	191	87%	DNN

Determination of the Number of Clusters. We set the number of clusters to 2, 3, 4, and 5. The results are shown in Fig. 5. It can be seen that when the number of classifiers is two, the classification effect is the best.

Classification Results for Different Features and Determination of Sample Size. We respectively used independent features for classification and combined features for classification. The result is shown in Fig. 6. The results show that our fusion feature classification results are much higher than the individual feature classification results. Then, we explore the relationship between sample size and accuracy, as in Fig. 6. We can find that when the sample size is 50, the accuracy rate is close to 90%.

Tree Structure Classifier Vs Simple Classifier. We set up two classifiers, one is a tree structure classifier, and the other is a simple classifier. We compare the accuracy of the two classifiers to verify our opinion, as in Fig. 7. The results show that when we use the tree structure classifier, as the number of programmers increases, the accuracy rate decreases slowly, improving the accuracy rate of 5% under the same number of programmers.

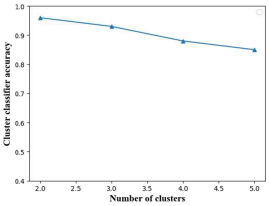

Fig. 5. Number of clusters

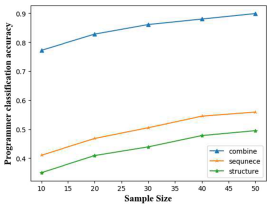

Fig. 6. Sample size and feature type

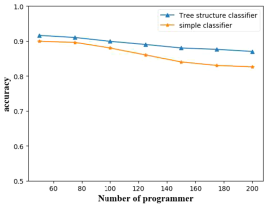

Fig. 7. Comparative analysis on accuracy

4 Conclusion

In this paper, we designed a model NMPI, extract features based on sequence and structure, and use SVM, DNN, Random Forest to classify. We compare the accuracy of classification of single features and classification of combined features. Observed the relationship between the number of samples per label and accuracy. Our experiment prove that DNN as classifier can better capture the characteristics of the author.

References

1. Burrows, S., Uitdenbogerd, A.L., Turpin, A.: Application of information retrieval techniques for source code authorship attribution. In: Zhou, X., Yokota, H., Deng, K., Liu, Q. (eds.) DASFAA 2009. LNCS, vol. 5463, pp. 699–713. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00887-0_61
2. Burrows, S., Uitdenbogerd, A.L., Turpin, A.: Comparing techniques for authorship attribution of source code. Softw. Pract. Experience **44**(1), 1–32 (2014)
3. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
4. Meng, X., Miller, B.P.: Binary code multi-author identification in multi-toolchain scenarios (2018)
5. Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space (2013)
6. Pellin, B.N.: Using classification techniques to determine source code authorship. Department of Computer Science, University of Wisconsin, White Paper (2000)
7. Perozzi, B., Alrfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710 (2014)
8. Rosenblum, N., Zhu, X., Miller, B.P.: Who wrote this code? identifying the authors of program binaries. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 172–189. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_10