J Inf Process Syst, Vol.18, No.2, pp.268~281, April 2022
https://doi.org/10.3745/JIPS.04.0241

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS

An Approach to Applying Multiple Linear Regression Models by Interlacing Data in Classifying Similar Software

Hyun-il Lim*

Abstract
The development of information technology is bringing many changes to everyday life, and machine learning can be used as a technique to solve a wide range of real-world problems. Analysis and utilization of data are essential processes in applying machine learning to real-world problems. As a method of processing data in machine learning, we propose an approach based on applying multiple linear regression models by interlacing data to the task of classifying similar software. Linear regression is widely used in estimation problems to model the relationship between input and output data. In our approach, multiple linear regression models are generated by training on interlaced feature data. A combination of these multiple models is then used as the prediction model for classifying similar software. Experiments are performed to evaluate the proposed approach as compared to conventional linear regression, and the experimental results show that the proposed method classifies similar software more accurately than the conventional model. We anticipate the proposed approach to be applied to various kinds of classification problems to improve the accuracy of conventional linear regression.

Keywords
Machine Learning, Linear Regression, Similar Software Classification, Software Analysis

1. Introduction

In recent years, software has played an ever more prominent role in information systems. To this end, various analytic approaches are needed in order to develop reliable software. Analysis of software is helpful for understanding its characteristics, and applied for a variety of purposes. Similar software classification refers to a method of classifying similar software through analysis of the characteristic of software. The relationship of similar software is technically applicable from the copy relation described in [1,2]. The copy relation denotes the relationship between similar software that can be derived from the same works via modifications or semantic preserving transformations (e.g., obfuscation or optimization). Approaches to classifying similarity of software have been studied as a basic software analysis technology, one comprising a variety of applications, including common module search [3], copyright violation and theft detection [4,5], and malicious code detection [6].

To compare features of software and classify similar software accurately, we must be able to under-

Manuscript received September 15, 2020; first revision February 1, 2021; second revision March 17, 2021; accepted March 26, 2021.
* **Corresponding Author:** Hyun-il Lim (hilim@kyungnam.ac.kr)
Dept. of Computer Engineering, Kyungnam University, Changwon, Korea (hilim@kyungnam.ac.kr)

stand the binary data structures from which software is formed. A given piece of software consists of a set or sequence of binary data whose complex structure is difficult to understand effectively through direct automatic program analysis. Therefore, complex analytic algorithms are required for analyzing software features.

With the rapid development of computing power in information systems in recent years, many real-world problems are now being solved using machine learning approaches [7-13]. Unlike more traditional algorithm-based approaches, machine learning methods generate models for solving specific problems from scratch through learning from training data. In recent research, as just one of a whole range of application areas in which machine learning methods have been growing more effective, machine learning has increasingly been applied to aspects of software analysis.

Linear regression [7,14] forms the basis of a method of machine learning that can estimate results by modeling the linear relationship between input data and output results. Generally, multi-dimensional input data are used to find a model for representing the presumed linear relationship of the output to the input data. To generate an accurate problem-solving model, it is important to process the training data effectively. In this paper, we propose a method of interlacing the training data to generate multiple linear regression models. The interlaced data features are used to train and generate multiple prediction models, which are then integrated into a single model to improve classification accuracy over the conventional linear regression approach. The approach proposed here is evaluated with a set of benchmark data containing Java applications.

This paper is organized as follows. Section 2 introduces machine learning approaches in software analysis and describes the concept of interlacing data in the linear regression context. Section 3 describes the design of multiple linear regression models using interlaced data to improve on conventional linear regression accuracy in classifying similar software. Section 4 shows the experimental results of evaluating the proposed approach as compared to conventional linear regression. Section 5 discusses the existing methods on software similarity analysis. Finally, Section 6 presents the key conclusions of this paper.

2. Linear Regression in Machine Learning

In this section, we summarize related work within the literature on machine learning and software classification. We also present the concept of interlacing data to construct multiple linear regression models and thus improve the accuracy of machine learning in software classification.

2.1 Machine Learning and Software Classification

With the recent proliferation of applications of machine learning technology, more and more studies have applied machine learning approaches to modeling and solving real-world problems. Various approaches to using machine learning have been studied, including regression [15,16], support vector machines [12,16-19], and neural networks [11,13,20]. In the specific application of using machine learning to analyze software features, approaches that have been studied include neural networks as methods for detecting source code plagiarism [21,22], naïve Bayes classification, and the *k*-nearest neighbor algorithm [15,23]. To analyze the similarity of binary code, the deep learning approach [24] has been used to generate images of binary code for classification. Neural network models [20] and support vector machine [17] have been used with training data to train analytic modeling of common features of binary code. In this paper, a method of interlacing data to generate multiple linear regression models is proposed to improve the accuracy of software classification over straightforward linear regression.

2.2 Linear Regression for Similar Software Classification

Linear regression is one method among statistical approaches to analyzing data and has been effective as an approach for machine learning. Linear regression methods analyze training data and generate a model for describing the linear relationship between input and output data. In other words, linear regression is a method of finding the best-fit linear relationship between independent input data and dependent output data. Generally, this method models the relationship between multi-dimensional input data and single-dimension output values. So, for independent input data $[x_1,\ldots, x_k]$ and a dependent output y, linear regression finds a linear model of the relationship between the input data vector $x = [x_1,\ldots, x_k]$ and the output y by analyzing statistical relationships among the data.

A code vector containing distribution information of each opcode in binary code of software can express the instruction level characteristics of software [14]. In designing linear regression for similar software classification, the difference of code vectors between software is used as independent input data. The corresponding output is labeled 1 or 0 according to the similarity of the two software. For example, for the code vectors of two software $[a_1,\ldots, a_k]$ and $[b_1,\ldots, b_k]$, the difference of the two vectors, $[x_1,\ldots, x_k]$, can be obtained from $[a_1 - b_1,\ldots, a_k - b_k]$, and it describes the degree of difference in characteristics of the binary opcode. The difference data $[x_1,\ldots, x_k]$ is used as independent input data for linear regression, and the data is the basis of classifying the similarity of software. The model thus generated is used to estimate output y from features of difference data $[x_1,\ldots, x_k]$, and the linear model to be analyzed using linear regression is described as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon, \tag{1}$$

where each β_i is the regression coefficient for representing the relationship between the i-th input data x_i and output y, and ε is a constant term that completes the model. Eq. (1) shows the combination of these coefficients to construct a linear model between the input data x and the output y. To determine the best-fit model, all the regression coefficients $\beta_0,\ldots, \beta_k$ are adjusted through the process of learning from training data. After constructing an estimation model for output data from the input, we can apply the model to estimate output results for other input data.

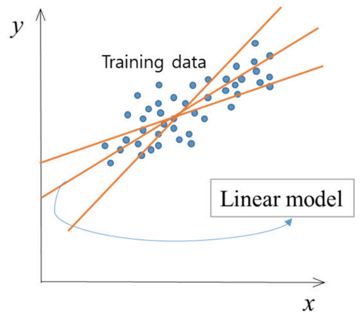

Fig. 1. An example of linear regression for input data x and output y.

Fig. 1 shows an example of applying linear regression to one-dimensional input data x and output y. To construct an effective model for estimating output from input data, the model should minimize errors over all the training data to improve accuracy in estimating results for other input data. To evaluate the error values for estimating results, an error function such as least squares or maximum likelihood estimation is applied to adjust the regression coefficients. The mean squared error of the training data is generally used as a linear regression error function to adjust the coefficients of the linear regression model. The mean squared error is a method of using the squares of the errors to express the degree of errors in analyzing the training data. As errors arising from the training data increase, the squared errors increase more steeply, so that larger deviations from the model are more heavily penalized. The error function keeps the degree of error as low as possible by adjusting the coefficients in such a way as to minimize the value of the error function. The linear regression model is thus constructed by minimizing the sum of errors over all the training data according to the chosen error function. Thus, in Fig. 1, the particular line that minimizes errors over all training data is used to describe the relationship between input and output data in the linear regression model.

3. Design of the Interlaced-Data Linear Regression Model

3.1 Concept of Interlacing Data

For linear regression in machine learning, the input data consist of various features of the data. Generally, the data are expressed as multi-dimensional values, with each dimension representing information about a single descriptive feature of the data. In a linear regression model, all these feature values affect the estimation of results in the trained model. However, some values among all the feature data may be noise values that hinder estimating the correct results. By eliminating such values effectively in training the linear regression model, its estimation accuracy can be improved.

However, much time and effort may be needed to analyze feature data thoroughly enough to eliminate noise values accurately in the training step. Because each feature value is an independent information strand of the input data, it is difficult to be precise about differentiating and removing noise values that tend to reduce prediction accuracy. To mitigate this difficulty, we propose a method of interlacing input data by alternately extracting feature values from the original input data. The conventional linear regression model uses all of the input data for training a single linear regression model. Our proposed method, by contrast, uses interlaced data to train multiple linear regression models independently.

Fig. 2 illustrates an example of interlaced data alternately extracted from original input data. The original input data consists of many feature values. One-third of each set of original feature values are alternately extracted to construct each of three interlaced datasets. Interlacing feature values from original data in this way helps to reduce the impact of noise values on training individual linear regression models, for the following reason.

Let $x = [x_1, \ldots, x_k]$ be an original input data vector for machine learning. If the dimension of the vector k is a multiple of three, the three interlaced data $x_{(1)}$, $x_{(2)}$, and $x_{(3)}$ are obtained as follows:

$$\begin{cases} x_{(1)} = [x_1, x_4, x_7, \cdots, x_{k-2}] \\ x_{(2)} = [x_2, x_5, x_8, \cdots, x_{k-1}] \\ x_{(3)} = [x_3, x_6, x_9, \cdots, x_k] \end{cases} \tag{4}$$

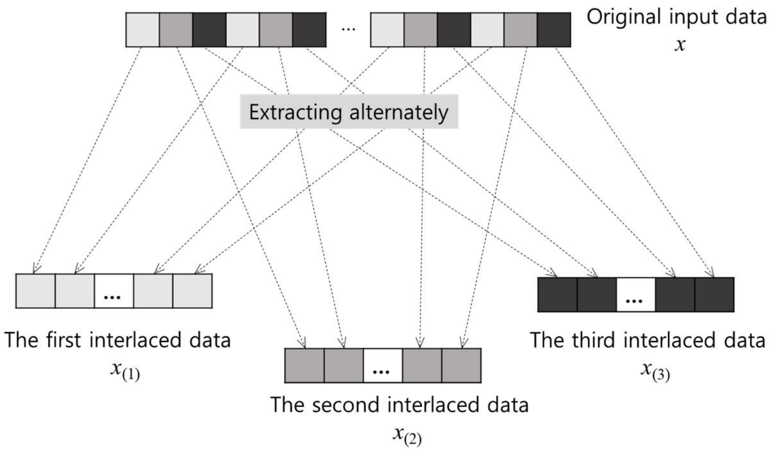
Original input data
x
Extracting alternately
The first interlaced data
$x_{(1)}$
The second interlaced data
$x_{(2)}$
The third interlaced data
$x_{(3)}$

Fig. 2. An example of interlaced data generated by alternately extracting features from original data.

Eq. (2) shows the three interlaced data vectors achieved by alternately extracting feature values from the original data x. If the dimension k of the original data is not a multiple of three, dummy constant values can be appended to the original data to match up the dimension of the interlaced data with each other.

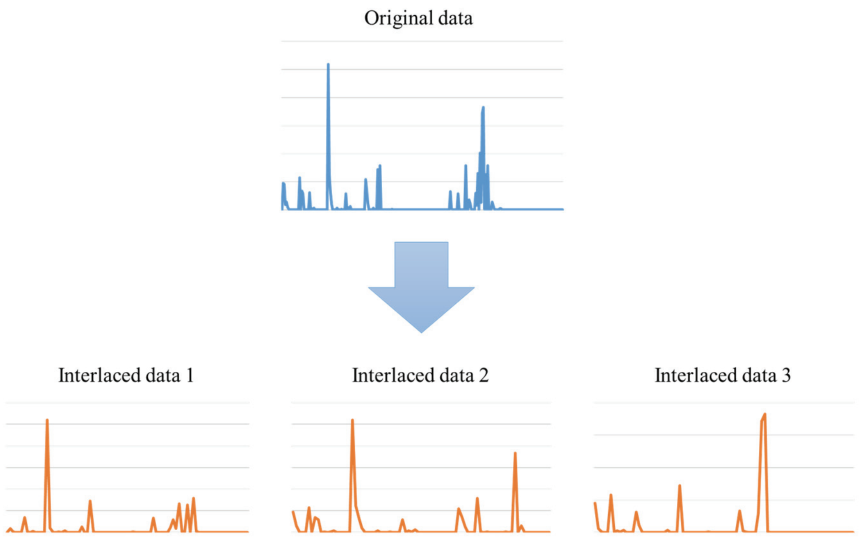

Fig. 3. An example of interlaced data generated from original Java bytecode data.

Fig. 3 shows an example of interlaced data generated from original data representing bytecode information for Java software. The graph shows the distribution information of bytecodes, and the spike in the graph is a unique characteristic of each software. The original data, which contains the distribution information of bytecodes in Java software, is divided into three interlaced datasets by alternately extracting feature data from the original datasets. The feature value contained in each interlaced dataset only affects the training of models to which the value belongs. Therefore, the scope of the influence of a noise value that interferes with the training of a machine learning model is limited only to the model of the interlaced dataset in which that noise value occurs. For example, let the feature value x be a noise value that impedes training the model to estimate the correct results. In conventional linear regression, because the original data can generate only one model, the *x* affects the overall results of the trained model. If the original data are distributed to three interlaced datasets, on the other hand, the noise value *x* is included in only one interlaced dataset, while the other interlaced data are not affected by *x* in training their individual machine learning models.

Since interlaced data are extracted alternately from original data, the scope of influence of each feature value in estimating results by linear regression is limited. This limitation of the scope of influence of noise is expected to reduce errors in prediction models trained with noisy data. The three interlaced datasets are used to construct three different linear regression models, and the overall estimation result can be obtained as a combination of these individual models, thereby achieving a more reliable result.

3.2 Integration of Linear Regression Models

After feature values from the original data are interlaced, only these interlaced values among all values from the original data are applied in training and generating a linear regression model. This implies that if only a single model trained with the interlaced data were used as a model for predicting results, feature values contained in the other interlaced data would not be reflected in the model's prediction results. To mitigate these shortcomings, individual models trained with each of the three interlaced datasets are combined to obtain a more reliable result. The three interlaced datasets generate three linear regression models, and although the original data contain noise values that hinder accurate estimation, the noise can be effectively suppressed by combining models separately trained on the individual interlaced datasets.

Fig. 4 shows the proposed design of multiple machine learning model with three interlaced datasets extracted from the original input data. The feature values of the original input data are distributed into the interlaced data by alternate extraction. The interlaced data are used as training data to generate a machine learning model for each interlaced dataset. Each trained model can be used as a stand-alone model, but the overall result can be obtained as a combination of the estimation results predicted from these individual interlaced-data models. Such a combination of models can be expected to produce more reliable results than a single model trained with the conventional linear regression approach.

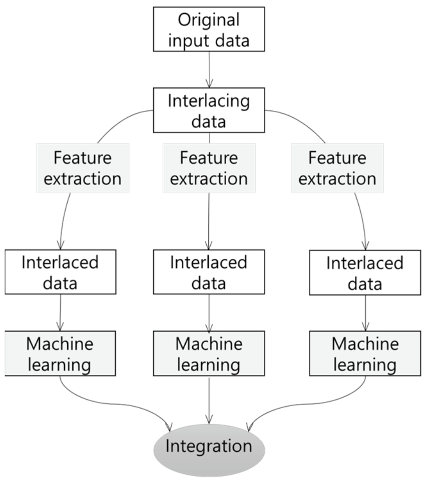

Fig. 4. Design of multiple machine learning models with interlaced data from original input data.

The three linear regression models generated on the interlaced data are trained using a conventional linear regression approach. The software similarity classification model used here is trained to predict 0 or 1 according to the similarity of software on feature data. After training models with the interlaced data, each linear regression model can be independently used to classify similar software. However, a single integrated model can be designed by combining the three classification models to achieve more reliable classification results.

Integration of machine learning models is achieved by combining the trained models to predict the final results of a problem. Let $ml(x)$ be the machine learning model of classifying similar software for training data x. Then, the machine learning model $ml_{(i)}$ for the i-th interlaced data $x_{(i)}$ can be formulated as follows:

$$ml_{(i)} = \begin{cases} 1, & \text{if } ml\left(x_{(i)}\right) = True \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Eq. (3) shows the individual machine learning model for the i-th interlaced data, and the results are classified as 1 or 0 according to the prediction results of the models of the interlaced data. Then, the three machine learning models are integrated as follows:

$$ml_{\text{integrated}} = \begin{cases} True, & \text{if } \sum_{i=1}^{3} ml_{(i)} \geq \frac{3}{2} \\ False, & \text{otherwise} \end{cases} \tag{4}$$

Eq. (4) shows the integrated model of the three models for the interlaced data. The integrated model determines the final prediction results based on the results of three different models of the interlaced data. Even if one model fails, the integrated model can predict the correct results if the other two models predict the correct results. Therefore, the integration of models is expected to improve the reliability of the prediction results.

Integration of the individual linear regression models trained with the interlaced data yields a combined model for classifying similar software. In other words, the similar software classification model is designed to put together the three interlaced-data models. Fig. 5 shows the structure of the similar software classification model with interlaced data designed in this paper. The Java bytecode analyzer analyzes Java software and generates the code vector of bytecode from input Java software to extract the features of software. The code vectors are compared with each other to generate the code difference data which can be used as training data of linear regression. By using the data interlacing analyzer, the original data are distributed into three interlaced datasets. Linear regression analysis is then applied to the three interlaced datasets independently to generate a unique model for classifying similar software for each interlaced dataset. After training, the three analytic models are integrated into one software similarity classification model to improve the reliability of the results. When integrating the three linear regression models, the result of the classification model is determined by the majority vote of the three models depending on the similarity of the feature values in the software.

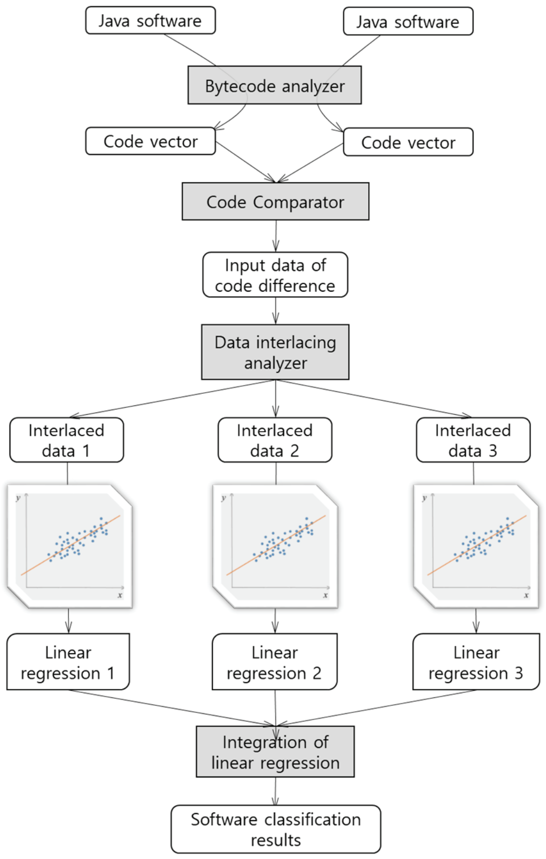

Fig. 5. Structure of the interlaced-data linear regression model for classifying similar software.

4. Experimental Evaluation

4.1 Experimental Environment

In this paper, we have presented a machine learning approach to classifying similar software that involves applying multiple linear regression models generated by interlacing data. This section describes an experiment to evaluate the accuracy of the proposed approach as compared to a conventional linear regression approach. Table 1 presents the experimental environment. To perform this experiment, Java software has been used as a benchmark software set for training and evaluating the proposed method. The proposed approach for classifying similar Java software has been implemented with Python [25] and the scikit-learn library [26] and was performed in a Microsoft Windows 7 environment with 16 GB main memory. The conventional linear regression model presented in [14] was applied to compare the prediction results of similar software classification.

Table 1. Experimental environment

CPU	RAM	Operating system	Language	Benchmark software set
Core i7-4790	16 GB	MS Windows 7	Python, Scikit-learn	Java software, Jakarta ORO, ANTLR

Table 2 shows the specification of the benchmark software for this experiment. We have independently used benchmark software sets for training and evaluation. The Java application, ANTLR [27], has been used as a training dataset for building linear regression models for interlaced features from the original training data. We generated the test dataset to evaluate the proposed approach from Jakarta ORO [28]. To reflect similar versions of software for classification, we have used the Smokescreen obfuscator to generate similar but modified versions of original benchmark software. Smokescreen changes the name used in software and modifies the control flow structures and instruction patterns of the software to modify the original program into a similar but obfuscated version that is difficult to understand. In other words, the Smokescreen obfuscator can transform an original program into similar versions with modified structures. The numbers of datasets we used for training and testing are 13,689 and 2,500, respectively. From the benchmark software, the code vectors were generated and compared with each other to generate the input data of the proposed multiple linear regression model.

Table 2. Benchmark dataset for training and test used in experimental evaluation of the proposed linear regression model

	Training data	**Test data**
Software data	ANTLR 3.5.2	Jakarta ORO 2.0.8
Number of class files	117	50
Maximum number of bytecodes	1,646	923
Average number of bytecodes	172	144
Number of datasets	13,689	2,500

4.2 Experiments Results

Table 3 shows experimental results for the combined interlaced data model as compared to a conventional linear regression model that is presented in [14]. Three strands of interlaced data were used, and the total number of test datasets was 2,500. The proposed model was a combination of three models trained with the interlaced feature data. The results of software similarity classification were evaluated as correct if the results achieved with a model could classify similar and dissimilar software correctly. In the experimental results for the conventional linear regression model, 2,252 classifications were correct out of 2,500 test data items, for overall classification accuracy of 90.08%. For the proposed model, 2,356 classifications were correct out of 2,500, so the classification accuracy was 94.24%, which was significantly more accurate than the conventional linear regression model. Based on these experimental results, we conclude that the proposed approach is more effective than conventional linear regression for classifying similar software.

Table 3. Experimental results for the proposed approach as compared to a conventional linear regression model

	Conventional linear regression model [14]	**Proposed model using interlaced data**
Number of interlaced datasets	1	3
Total number of classifications	2,500	2,500
Number of correct classifications	2,252	2,356
Overall accuracy (%)	90.08	94.24

From the experimental results, the classification accuracy of the proposed approach was about four percentage points higher than that of the conventional method. Moreover, the numbers of false positives and false negatives decreased in the proposed approach as well. This enhanced performance was to be expected: the conventional method produces only one classification model that is then trained with the original training data, so that noise values within the training data may lead the model to make incorrect classifications for similar software. The advantage of the method proposed in this paper is that it distributes the effects on classification results of feature values, including noise values, by separating the original data into several interlaced datasets. Since the interlaced datasets are generated by alternately extracting features from the original training data, the scope of the effect on classification results of interlaced data is limited to the particular linear regression model to which the data belong. Moreover, combining multiple linear regression models trained with interlaced data can produce more reliable results in classifying similar software.

In the experimental results, the proposed linear regression model with interlaced data improved the accuracy of the results in classifying similar software. This confirmed that separating the effects of the feature values of training data helps improving the classification accuracy of similar software in the linear regression approach. More generally, the proposed method of interlacing data can be effective at improving the accuracy of linear regression whenever training data contain noise feature values. The application of multiple models with interlaced data is anticipated to design improved linear regression models that reduce estimation errors in classification problems.

5. Related Work and Discussion

5.1 Related Work

In this section, the existing methods for software similarity analysis are discussed. Table 4 shows the related work on software similarity analysis. According to the approaches for analyzing the similarity of software, the methods are classified into five groups. The source code analysis is optimized for source code comparison through neural networks [21,22], *k*-NN [23], or token strings [29]. These methods are limited to apply in environments where the source code of software is available. Software birthmark means the inherent characteristics of software that can be used to distinguish different software. This approach analyzes various characteristics of software, such as runtime API call sequences [1], whole program path [4], static information of Java software [5], *k*-gram of opcode [30], or dynamic opcode *n*-gram [31]. The analyzed birthmarks express the inherent characteristics of software and the birthmarks are compared to distinguish different software. This approach requires designing specific code analysis algorithms to extract the birthmark data based on information of interest. Similarly, the value-based approach focuses on the specific values of static or dynamic environments, such as features of API calls [6], the semantics of basic blocks [32], critical runtime value [33], or program logic [34]. Machine learning-based methods apply several approaches of machine learning to analyze the similarity of software, such as linear regression [14], support vector machine [17], and neural networks [20,24].

This approach classifies similar software through learning from previously known training data. So, the quality of training data and the design of the machine learning model are important. The proposed approach is a variation of machine learning with linear regression. The proposed method is an approach to improve existing methods by applying multiple models by interlacing input data. The interlacing of data tries to localize the effects of noise values that may be contained in training data, and the integration of multiple models can improve the reliability of analytical results. The software similarity analysis can be useful in various fields in computer science, such as detection of illegal code reuse, identification of similar algorithm or software, and malware detection.

Table 4. Related work on software similarity analysis

	Source code analysis	**Software birthmark**	**Value-based**	**Machine learning**	**Proposed approach**
Analysis approach	Static	Static [5,29] or Dynamic [1,4,30]	Static [31] or Dynamic [6,32,33]	Static	Static
Target software	Source code	Binary code	Binary code	Binary code	Binary code
Comparison approach	Neural net [21,22] *k*-NN [23] Token [29]	API call [1] Whole program path [4] Static info. [5] *k*-gram [30] Dynamic opcode [31]	API call [6] Basic block [32] Critical runtime value [33] Program logic [34]	Linear regression [14] SVM [17] Neural net [20,24]	Linear regression of interlaced data
Requirement	Applicable for source code only	Design of code analysis algorithm	Design of code analysis algorithm that is specific to the structure of binary code	Design of machine learning and training data	Design of multiple linear regression models on interlaced data
Applicability	Detection of source code plagiarism	Comparison of binary code through analyzed code info.	Comparison of binary code through static or runtime data or values	Classification of similar software through machine learning	Classification of similar software through integration of linear regression
Advantage	Optimized for source code comparison	Efficient comparison of binary code	Optimized for specific values or data	Comparison through machine learning	Improved comparison accuracy with multiple models

5.2 Discussion and Future Work

The integration of three independent models can improve reliability and accuracy because they can be calibrated through different models, even if the prediction results are wrong in one model. On the other hand, there are several issues to consider to apply the proposed method. Although the accuracy and reliability of the classification result can be improved, the design of the proposed model is more complicated than the conventional model. In the design of multiple linear regression models, the procedure of several processes is required, including interlacing data, learning and generating independent models, integrating individual models, and finally predicting the results. Because the complex procedure can be overhead in the performance of learning and predicting results, the proposed model may be disadvantageous for applying in performance-critical environments such as real-time processing.

Overfitting is one of the main concerns in applying machine learning approaches to predict the results of real-world problems. The proper control of the occurrence of overfitting is important in designing machine learning models. Because the proposed model integrates independent multiple models for classifying similar software, overfitting may appear in different aspects depending on the features of the interlaced data of the models. Therefore, much more attention is needed to find and control the presence of overfitting compared to the existing ones. In future work, we plan to study how to find overfitting in different individual models and how to reduce the occurrence of overfitting in multiple linear regression models.

6. Conclusion

The importance of software in information systems has been continuously increasing in recent years. In view of this increasingly important role, ongoing effort is needed to better understand the characteristics of software. Such efforts will be helpful for improving productivity and safety in software development. In recent software analysis studies, machine learning approaches have become widely used. To accurately train models and improve accuracy in applying machine learning, it is important to reflect the characteristics of data in the training process.

Linear regression is widely used in estimation problems that can be solved by modeling a linear relationship between input and output data. The conventional linear regression model can be used to classify similar software by training with data representing the software features. In this paper, we proposed an approach to this kind of machine learning that involves applying multiple linear regression models generated by interlacing data, which is expected to improve classification accuracy for similar software. We presented a method of interlacing data to generate multiple linear regression models, including the design of the combined linear regression model derived from the interlaced data. We then conducted experiments to evaluate the proposed approach as compared to conventional linear regression models for classifying similar software, and the experimental results show that the proposed method can indeed classify similar software more accurately than conventional linear regression. The proposed approach is expected to be an effective method in linear regression contexts for improving the accuracy of results. The application of multiple models with interlaced data is anticipated to reduce estimation errors in classification problems appropriate for linear regression.

References

[1] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. Matsumoto, "Dynamic software birthmarks to detect the theft of windows applications," in *Proceedings of International Symposium on Future Software Technology (ISFST)*, Xian, China, 2004.

[2] S. Cesare, "Software similarity and classification," Ph.D. dissertation, Deakin University, Geelong, Australia, 2013.

[3] H. Park, H. I. Lim, S. Choi, and T. Han, "Detecting common modules in Java packages based on static object trace birthmark," *The Computer Journal*, vol. 54, no. 1, pp. 108-124, 2011.

[4] G. Myles and C. Collberg, "Detecting software theft via whole program path birthmarks," in *Information Security*. Heidelberg, Germany: Springer, 2004, pp. 404-415

[5] H. Tamada, M. Nakamura, A. Monden, and K. I. Matsumoto, "Java birthmarks: detecting the software theft," *IEICE Transactions on Information and Systems*, vol. 88, no. 9, pp. 2148-2158, 2005.

[6] M. Alazab, R. Layton, S. Venkataraman, and P. Watters, “Malware detection based on structural and behavioural features of API calls,” in *Proceedings of the 1st International Cyber Resilience Conference*, Perth, Australia, 2010, pp. 1-10.

[7] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.

[8] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, UK: Cambridge University Press, 2014.

[9] P. Domingos, “A few useful things to know about machine learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78-87, 2012.

[10] D. T. Ramotsoela, G. P. Hancke, and A. M. Abu-Mahfouz, “Attack detection in water distribution systems using machine learning,” *Human-centric Computing and Information Sciences*, vol. 9, article no. 13, 2019. https://doi.org/10.1186/s13673-019-0175-8

[11] D. H. Kwon, J. B. Kim, J. S. Heo, C. M. Kim, and Y. H. Han, “Time series classification of cryptocurrency price trend based on a recurrent LSTM neural network,” *Journal of Information Processing Systems*, vol. 15, no. 3, pp. 694-706, 2019.

[12] M. J. J. Ghrabat, G. Ma, I. Y. Maolood, S. S. Alresheedi, and Z. A. Abduljabbar, “An effective image retrieval based on optimized genetic algorithm utilized a novel SVM-based convolutional neural network classifier,” *Human-centric Computing and Information Sciences*, vol. 9, article no. 31, 2019. https://doi.org/10.1186/s13673-019-0191-8

[13] C. Cicceri, F. De Vita, D. Bruneo, G. Merlino, and A. Puliafito, “A deep learning approach for pressure ulcer prevention using wearable computing,” *Human-centric Computing and Information Sciences*, vol. 10, article no. 5, 2020. https://doi.org/10.1186/s13673-020-0211-8

[14] H. I. Lim, “A linear regression approach to modeling software characteristics for classifying similar software,” in *Proceedings of 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Milwaukee, WI, 2019, pp. 942-943.

[15] W. Liu, P. Wang, Y. Meng, C. Zhao, and Z. Zhang, “Cloud spot instance price prediction using kNN regression,” *Human-centric Computing and Information Sciences*, vol. 10, article no. 34, 2020. https://doi.org/10.1186/s13673-020-00239-5

[16] W. Li, X. Li, M. Yao, J. Jiang, and Q. Jin, “Personalized fitting recommendation based on support vector regression,” *Human-centric Computing and Information Sciences*, vol. 5, article no. 21, 2015. https://doi.org/10.1186/s13673-015-0041-2

[17] H. I. Lim, “Design of similar software classification model through support vector machine,” *Journal of Digital Contents Society*, vol. 21, no. 3, pp. 569-577, 2020.

[18] M. J. Ding, S. Z. Zhang, H. D. Zhong, Y. H. Wu, and L. B. Zhang, “A prediction model of the sum of container based on combined BP neural network and SVM,” *Journal of Information Processing Systems*, vol. 15, no. 2, pp. 305-319, 2019.

[19] M. Zouina and B. Outtaj, “A novel lightweight URL phishing detection system using SVM and similarity index,” *Human-centric Computing and Information Sciences*, vol. 7, article no. 17, 2017. https://doi.org/10.1186/s13673-017-0098-1

[20] N. Shalev and N. Partush, “Binary similarity detection using machine learning,” in *Proceedings of the 13th Workshop on Programming Languages and Analysis for Security*, Toronto, Canada, 2018, pp. 42-47.

[21] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, “Deep learning code fragments for code clone detection,” in *Proceedings of 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Singapore, 2016, pp. 87-98.

[22] D. Heres, “Source code plagiarism detection using machine learning,” Master's thesis, Utrecht University, Utrecht, Netherlands, 2017.

[23] U. Bandara and G. Wijayarathna, “A machine learning based tool for source code plagiarism detection,” *International Journal of Machine Learning and Computing*, vol. 1, no. 4, pp. 337-343, 2011.

[24] N. Marastoni, R. Giacobazzi, and M. Dalla Preda, “A deep learning approach to program similarity,” in *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, Montpellier, France, 2018, pp. 26-35.

[25] Python programming language [Online]. Available: https://www.python.org/.

[26] Scikit-learn: machine learning in Python [Online]. Available: http://scikit-learn.org/stable/index.html.

[27] ANTLR (ANother Tool for Language Recognition) [Online]. Available: https://www.antlr.org/.

[28] The Apache Jakarta Project [Online]. Available: https://jakarta.apache.org/oro/.

[29] L. Prechelt, G. Malpohl, and M. Philippsen, “Finding plagiarisms among a set of programs with JPlag,” *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016-1038, 2002.

[30] G. Myles and C. Collberg, “K-gram based software birthmarks,” in *Proceedings of the 2005 ACM Symposium on Applied Computing*, Santa Fe, NM, 2005, pp. 314-318.

[31] B. Lu, F. Liu, X. Ge, B. Liu, and X. Luo, “A software birthmark based on dynamic opcode n-gram,” in *Proceedings of the International Conference on Semantic Computing (ICSC)*, Irvine, CA, 2007, pp. 37-44.

[32] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, “Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection,” *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1157-1177, 2017.

[33] Y. C. Jhi, X. Jia, X. Wang, S. Zhu, P. Liu, and D. Wu, “Program characterization using runtime values and its application to software plagiarism detection,” *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 925-943, 2015.

[34] F. Zhang, D. Wu, P. Liu, and S. Zhu, “Program logic based software plagiarism detection,” in *Proceedings of 2014 IEEE 25th International Symposium on Software Reliability Engineering*, Naples, Italy, 2014, pp. 66-77.

Hyun-il Lim **https://orcid.org/0000-0003-2802-834X**

He received his B.S., M.S., and Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1995, 1997, and 2009, respectively. He is currently a professor in the School of Computer Science and Engineering, Kyungnam University. His current research interests include software security, software analysis, machine learning, and program analysis.