A Transformer-based approach to highly granular source code authorship attribution

Chongzheng Shi

China University of Mining and Technology, Jiangsu, China

shichongzheng@cumt.edu.cn

Abstract. Traditional source code authorship identification methods often rely on features such as textual similarity, programming style or metadata, however, these methods often struggle to extract the precise source code authoring style when dealing with large-scale code bases or complex programming patterns, resulting in poor performance. Therefore, this paper proposes a Transformer-based high fine-grained source code author attribution method.Aiming at the problem of roughness of existing literature on word segmentation, this paper proposes a high fine-grained source code segmentation method to extract higher fine-grained features. Aiming at the problem of feature dimension redundancy, this paper adopts the Transformer network to locate sensitive features that can characterise the author's style.To verify the effectiveness of the model, it was tested on GCJ-C++ and GCJ-Java datasets. The experimental results show that the proposed method model achieves higher recognition accuracy in the source code authorship attribution problem compared to the traditional methods.

Keywords: Source code authorattribution, Transformer network, Feature engineering.

1. Introduction

Source code authorship attribution techniques play an important role in several domains by ana-lysing the stylistic features of a given code to effectively identify its authorship. However, the problem of source code author recognition faces many challenges. Firstly, pro-gramming language recognition is more complex compared to natural language text recognition, although there are extensive studies on natural language text recognition in the literature [1,2,3], there is still a dearth of research on source code authorship attribution, mainly because the written code expressions established by the compiler's syntactic rules have an inherent constrainability, which increases the difficulty of the recognition work.[4] Secondly, in real programming environments, the programming styles of programmers are usually influenced by a variety of factors, such as the programming paradigm of the company or team, and the individual's work experience. These factors may lead to a gradual convergence of the programming styles of programmers who originally have very different styles after joining the same company, thus increasing the difficulty of identification. In addition, even the same programmer may have different code styles when programming in different languages, and such cross-language style differences also pose a chal-lenge for source code authorship attribution.

In order to solve the above challenges, in recent years researchers have started to try to apply deep learning methods to the field of source code authorship attribution. Currently, the accuracy of deep learning-based source code authorship attribution has reached more than 90%.Although deep learning has made some progress in source code authorship identification, there are still some problems to be solved. Firstly, for large-scale text processing, how to choose the appropriate segmentation method is a key issue. Secondly, which kind of neural network structure to choose is also an issue that needs to be explored.

In order to explore the impact of the partitioning method and different neural network choices on the accuracy of source code authorship attribution, this paper proposes an improved technique for source code authorship attribution using deep learning based on Abuhamad et al. [4]

2. Related work

Early research relied heavily on textual features of codes, such as lexical features and N-gram techniques.With the continuous development of research, researchers started to look for more in-depth analysis methods.Caliskan-Islam et al. [5] made a breakthrough in this area by introducing Abstract Syntax Tree and Syntactic Features, and finally achieve 93% and 98% of the results on the datasets with 1,600 and 250 developers. WangNingfei et al. [6] got rid of the disadvantage of most research methods focusing only on static features of source code, integrated static and dynamic style measurement analysis.They achieved an accuracy higher than 95.45% on the python test dataset. And then with the gradual development of machine learning and deep learning, Abuhamad et al. [4] proposed a deep learning based source code authorship attribution system (DL-CAIS). They successfully processed data from 8,903 programmers and maintained an accuracy of up to 92.3%. Egor Bogomolov et al. [7], by using AST paths and adapting the code2vec neural network, proposed PbRF (Path-Based Random Forest) and PBNN (Path-Based Neural Network) models, and the two models achieve recognition accuracies of 97.6% (PbNN) and 98.5% (PbRF) in terms of authorship attribution in Java.In addition to source code level studies, other scholars have focused on author attribution of binary files. Aylin et al. [8] have achieved pioneering results in this area by proposing the first author traceability method for executable binaries, and demonstrating for the first time that automated decompilation of executable binaries provides an additional useful feature for de-anonymisation of code authorship.

3. Related technologies

3.1. TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical method widely used in the field of information retrieval and text mining.Many researchers have used them in the field of code authorship attribution [2,9,10].TF-IDF combines two metrics, TF and IDF, by calculating the product of the two to determine the weight of a word in a document set.

TF (Term Frequency) indicates how often a term appears in the text, this number is usually normalised to prevent it from biasing long documents.

Formula: $TF_{i,j} = \frac{n_{i,j}}{\Sigma_k n_{k,j}}$,where $n_{i,j}$ denotes the number of occurrences of the word i in the d_j document.$\Sigma_k n_{k,j}$ is the sum of the occurrences of all words in the d_j document.

IDF (Inverse Document Frequency) is a measure of the general importance of a word. The IDF of a particular word can be obtained by dividing the total number of documents by the number of documents containing the word, and then taking the logarithm of the quotient. Generally speaking, if the fewer the documents containing the term, the larger the IDF, then the term has a good ability to distinguish between categories.

Formula.$IDF_i = log\frac{|D|}{1+|\{j:n_i \varepsilon d_j\}|}$,where$|D|$ denotes the total number of documents in the corpus, and$|\{j: n_i \varepsilon d_j\}|$ denotes the number of documents containing the word n_i. $+1$ is to prevent the word not being in the corpus, resulting in a denominator of 0.

For the n_i word of the d_j document, the $TF - IDF(n_i) = TF_{i,j} * IDF_i$

3.2. Transformer Neural Network

Transformer neural network is a neural network architecture based on self-attention mechanism. It was proposed by Ashish Vaswani et al. [11], which was initially used for sequence-to-sequence learning in natural language processing tasks.

The core of the Transformer model lies in its self-attention mechanism, which is able to globally model each element in a sequence and create links between the elements. This mechanism captures the importance of each element in the sequence by computing the query matrix, key matrix and value matrix, and using the softmax function to obtain a probability distribution.

The Transformer model has better parallel performance and shorter training time than the traditional Recurrent Neural Network (RNN) model.

4. The Framework of this paper

The overall approach to source code author attribution proposed in this paper can be divided into three phases: preprocessing, representation through learning, and output, and the overall structure is shown in Figure 1, which is explained in detail below.

Preprocessing stage, the first stage is the data preprocessing stage, which mainly consists of two steps, word-splitting and generating TF-IDF feature representations. In the segmentation stage, this paper adopts a high fine-grained segmentation method based on naming styles to segment the code files.

Learning features stage, Transformer neural network is mainly used in this stage. In this stage, the Transformer encoder is introduced and the TF-IDF feature representation obtained in the preprocessing stage is learnt using the Transformer network.

In the output stage, three fully connected layers are built in this stage to further learn the results obtained by the Transformer network and output the final recognition accuracy.

4.1. Highly Fine-Grained Segmentation Methods

In the preprocessing stage, firstly, the code file is segmented, and then the resulting segmentation results are used to generate feature representations using TF-IDF, and then the mutual information between the generated TF-IDF feature representations and the author tags is calculated, and finally, among n features, the TF-IDF feature representation with the top k high mutual information is selected as the input of the Transformer Network. When performing segmentation, this paper improves the problem of coarse segmentation and proposes a highly fine-grained segmentation method based on the naming method. Currently, the naming of variables and functions mainly follow four canonical naming laws such as camel naming, Pascal naming, underscore naming, Hungarian naming, etc. There are also some naming methods that do not follow any naming law, only by the stack of chaotic characters or divided by other special characters and numbers as the boundaries.This paper writes matching rules for the first five naming rules, and uses regular expressions to filter the word separation, and if the naming is piled up with messy characters, it will be directly discarded and not processed.

4.2. Using the Transformer Network

The classification method is based on transformer, which captures the dependencies in the sequence by calculating the correlation between each position and all other positions in the input sequence, and has excellent performance in processing sequence data, which is often used in NLP (Natural Language Processing). The Transformer model mainly consists of two parts: encoder and decoder, but since this experiment only needs to classify and not need to output the sequence, so only the encoder is used to simplify the model and reduce the consumption of resources.

In the encoder, each layer consists of two main sublayers: a self-attention layer and a feedforward neural network layer. The self-attention layer captures the dependencies in the sequence by calculating the attentional weights between each position in the input sequence and all other positions to generate a weighted representation. The feed-forward neural network layer then further pro-cesses the output of the self-attention layer to extract higher-level features that can effectively locate sensitive features characterising the author's style.

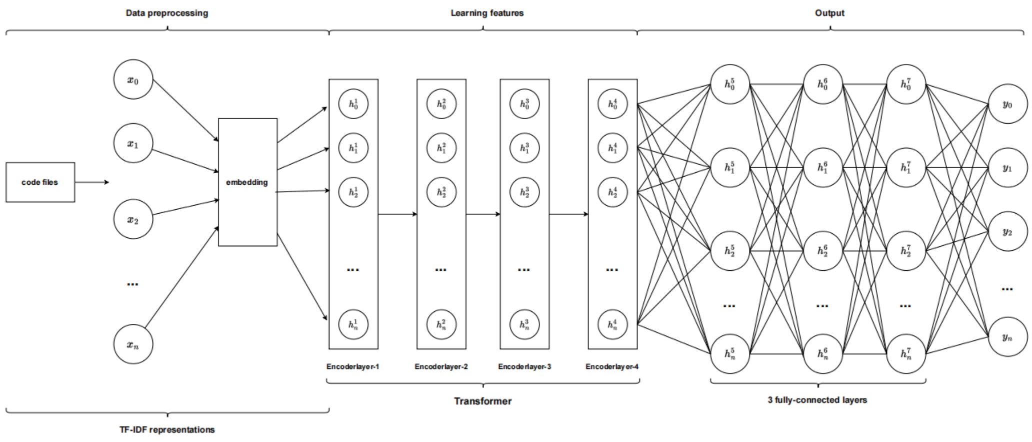

Figure 1. The framework of this work's method.

5. Experiments

5.1. Data sets

Google Code Jam (GCJ) is an international programming competition organised by Google since 2008. This paper uses some of the C++ and Java codes from the competition as datasets for experiments.

The datasets are listed below:

(1) Dataset 1: For C++ experiments, the training set contains 205 authors and the test set contains 20 authors;

(2) Dataset 2: For the Java experiments, the training set contains 74 authors and the test set contains 20 authors.

5.2. Effectiveness of the Transformer Network

It is found that the accuracy of Transformer on the GCJ-C++ and GCJ-Java datasets is always no less than that of LSTM and RNN regardless of the number of feature dimensions in the range of feature dimensions selected in this paper, and the highest accuracy of Transformer can reach 95% (C++) and 98.61% (Java), which is higher than 90% (C++) and 97.22% (Java) for LSTM and 90% (C++) and 97.22% (Java) for RNN, as shown in Figure 2 and Figure 3. This paper also compares the recognition speeds of Transformer and LSTM in different feature dimensions, as shown in Figure 4 and Figure 5.The results show that the recognition time of Transformer is much smaller than that of LSTM, and the recognition time of LSTM increases with the increase of feature dimensions, while the recognition time of Transformer does not change much. It is assumed that Transformer has no looping structure and does not need to process the data sequentially, so it has little effect when the feature dimensions increase; whereas, LSTM has sequence dependence, and the updating of the hidden state relies on the hidden state of the previous time step, so it is not able to carry out the parallel computation, and LSTM has a obvious loop structure, with the increase of feature dimension, the computational complexity will also increase, so the recognition time will gradually increase.

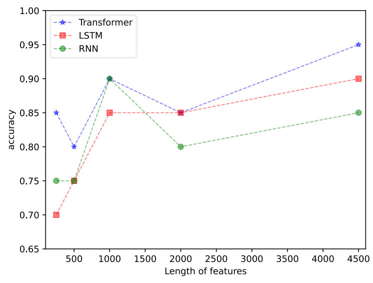

Figure 2. The accuracy of C++ for different neural networks.

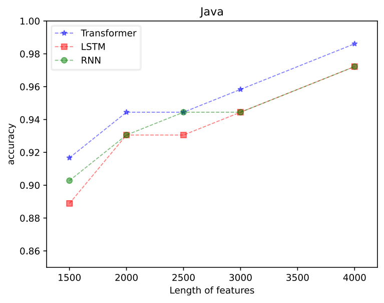

Figure 3. The accuracy of Java for different neural networks.

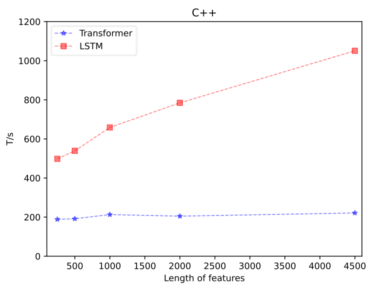

Figure 4. Runtime of Transformer and LSTM on C++.

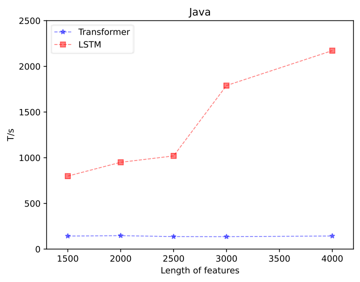

Figure 5. Runtime of Transformer and LSTM on Java.

5.3. Effectiveness of High-Fine-Grained Segmentation Methods

This paper tests the effectiveness of high fine-grained segmentation methods under the Transformer network. One group uses the segmentation method proposed in this paper, which is referred to as *new* in the following figure, and the other group uses the segmentation method based on the word habit, which is referred to as *old* in the following figure, and tests are conducted on the GCJ-C++ and GCJ-Java datasets, re-spectively. The accuracy of the C++ and Java tests under different TD-IDF feature dimensions are shown in Figure 6 and Figure 7. From the figures, it can be seen that when the selected TF-IDF feature dimension is small, the high fine-grained segmentation method proposed in this paper outperforms the method based on word habit segmentation in both language environments. In particular, on the Java dataset, the advantage of the high fine-grained segmentation method is more obvious when the TF-IDF feature dimension is small, and the advantage of the high fine-grained segmentation method decreases particularly when the IF-IDF feature dimension is large. The advantage of the high-fine-grained segmentation method decreases especially when the TF-IDF feature dimension is large, and the change is not especially obvious on the C++ dataset. When the TF-IDF feature dimension is particularly large (4500), the accuracy of the two overlap. It is spec-ulated that the reason for this may be that high fine-grained disambiguation can better capture the structural and semantic information of the code, resulting in a better representation of the IF-IDF features, which leads to improved accuracy.

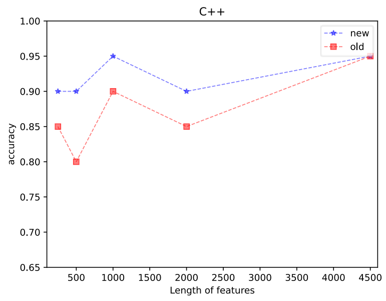

Figure 6. The accuracy of C++ using different word segmentation methods.

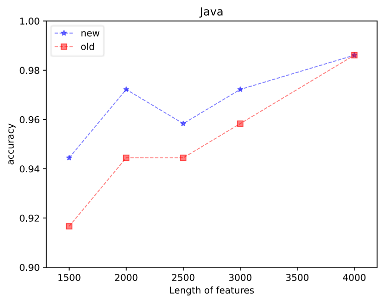

Figure 7. The accuracy of Java using different word segmentation methods.

6. Summary

In this paper, I use a finer-grained word segmentation method and Transformer network for source code authorship recognition, and tested on GCJ-C++ and GCJ-Java datasets, and the accuracy rate reaches 95% (C++) and 98.61% (Java) , and at the same time, due to the parallelism of Transformer network, compared with LSTM and other sequence models, Transformer network can greatly improve the speed of source code author identification, but compared with other related literature [4,7,8], the dataset selection is small, and the effect of running on large-scale code datasets is not yet known. In the future, tests on larger datasets are needed to further validate the feasibility of the conclusions of this paper.

References

[1] Patrick Juola. 2006. Authorship attribution. Found. Trends Inf. Retr. 1, 3 (December 2006), 233–334.

[2] Vlado Kešelj, Fuchun Peng, Nick Cercone, and Calvin Thomas. 2003. N-gram-based Author proiles for authorship attribution. In Proceedings of the conference paciic association for computational linguistics, PACLING, Vol. 3. 255-264.

[3] Moshe Koppel, Jonathan Schler, and Shlomo Argamon. 2009. Computational methods in authorship attribution. Journal of the Association for Information Science and Technology 60, 1 (2009), 9-26.

[4] Mohammed Abuhamad, Tamer AbuHmed, Aziz Mohaisen, and DaeHun Nyang. 2018. Large-Scale and Language-Oblivious Code Authorship Identification. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). Association for Computing Machinery, New York,NY,USA,101–114.

[5] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing programmers via code stylometry. In Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15). USENIX Association, USA, 255–270.

[6] Ningfei Wang, Shouling Ji, and Ting Wang. 2018. Integration of Static and Dynamic Code Stylometry Analysis for Programmer De-anonymization. In Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security (AISec '18). Association for Computing Machinery, New York, NY, USA, 74–84.

[7] Egor Bogomolov, Vladimir Kovalenko, Yurii Rebryk, Alberto Bacchelli, and Timofey Bryksin. 2021. Authorship attribution of source code: a language-agnostic approach and applicability in software engineering. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021). Association for Computing Machinery, New York, NY, USA, 932–944.

[8] Caliskan, A., Yamaguchi, F., Dauber, E., Harang, R., Rieck, K., Greenstadt, R., & Narayanan, A. (2018). When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries. In 25th Annual Network and Distributed System Security Symposium, NDSS 2018 (25th Annual Network and Distributed System Security Symposium, NDSS 2018). The Internet Society.

[9] Steven Burrows and S. M. M. Tahaghoghi. 2007. Source code authorship at-tribution Using n-grams. In Proceedings of the Twelfth Australasian Document Computing Symposium (ADCS'07). Spink A, Turpin A, Wu M (eds), 32-39.

[10] Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, and Sokratis Katsikas. 2006. Effective identification of source code authors using byte-level information. In Proceedings of the 28th international conference on Software engineering (ICSE '06). Association for Computing Machinery, New York, NY, USA, 893–896.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.