DeepSCC: Source Code Classification Based on Fine-Tuned RoBERTa

Guang Yang   Yanlin Zhou   Chi Yu   Xiang Chen*

School of Information Science and Technology, Nantong University, Nantong, China
1930320014@stmail.ntu.edu.cn 1159615215@qq.com yc_struggle@163.com xchencs@ntu.edu.cn

Abstract

In software engineering-related tasks (such as programming language tag prediction based on code snippets from Stack Overflow), the programming language classification for code snippets is a common task. In this study, we propose a novel method DeepSCC, which uses a fine-tuned RoBERTa model to classify the programming language type of the source code. In our empirical study, we choose a corpus collected from Stack Overflow, which contains 224,445 pairs of code snippets and corresponding language types. After comparing nine state-of-the-art baselines from the fields of source code classification and neural text classification in terms of four performance measures (i.e., Accuracy, Precision, Recall, and F1), we show the competitiveness of our proposed method DeepSCC.

1   Introduction

Recently, multiple programming languages (such as Java, Python, C++) are often used together in the large-scale software development process, since different development tasks often use different programming languages. When developers ask questions on Stack Overflow [1] [2], the answers to the questions are closely related to the type of programming language. Therefore, Stack Overflow needs to use the correct programming language tag of posts to match the related answers for users, and the source code classification task can effectively solve this problem.

In the previous studies, this task is often modeled as a text classification problem. Then machine learning methods can be used to classify the source code's language type. For example, Khasnabish et al. [3] used a Naive Bayesian classifier. Alrashedy et al. [4] used a random forest classifier and XGBoost. Motivated by the research progress of neural text classification and code semantic learning [5], we propose a novel method DeepSCC by fine-tuning the pre-trained model RoBERTa [6] to perform the source code classification task.

To verify the effectiveness of our proposed method DeepSCC, we choose a corpus collected from Stack Overflow, which contains 224,445 pairs of code snippets and corresponding language types. We first perform data pre-

processing on this corpus, such as word segmentation, discarding noisy code snippets. Then, we use the corpus to fine-tuning the RoBERTa model [6]. We compared DeepSCC with nine state-of-the-art baselines. For these chosen baselines, two baselines are selected from the source code classification field [4, 7], and the remaining baselines are selected from the neural text classification field [8–12]. In terms of four performance measures (Accuracy, Precision, Recall, and F1), we find DeepSCC can outperform these baselines.

The main contributions of our study can be summarized as follows:

(1) We propose a novel method DeepSCC by fine-tuning the pre-trained model RoBERTa [6], which can classify the language type of the source code. We share our trained classification model for other researchers to follow and replicate our study[1].

(2) We choose corpus gathered from Stack Overflow as our experimental subject. Then we choose two baselines proposed by Alrashedy et al. [4] (i.e., in the source code classification field) and seven baselines based on TextCNN and Transformer (i.e., in the neural text classification field). Final experimental results show that DeepSCC can improve the performance of source code classification.

2   Related Work

In previous studies on source code classification, Kennedy et al. [13] proposed a software language model to recognize the entire source code file from Github. Their classifier is based on five natural language statistical models. They gathered corpus from GitHub and considered 19 programming languages. Khasnabish et al. [3] collected more than 20,000 source code files. These source codes are downloaded from multiple repositories in GitHub. The model uses the Bayesian classifier and aims to predict ten programming languages. Klein et al. [14] collected 41,000 source code files from GitHub as the training set and randomly selected 25 source code files as the test set. However, their methods, which are based on supervised learning and feature selection methods, can only achieve 48% accuracy at most. Alrashedy et al. [7] proposed the method SCC to classify source code snippets via Naive Bayes classifier, with an accuracy of about 75%. This method can also distinguish programming language families (such as

---

[1]`https://huggingface.co/NTUYG/DeepSCC-RoBERTa`

C, C# and C++) with an accuracy of 80%, and can identify programming language versions (such as C#3.0, C#4.0, and C#5.0) with an accuracy of 61%. Recently, Alrashedy et al. [4] classified the language types for code snippets in Stack Overflow. They used the random forest classifier and XGBoost to build classifiers. Different from the previous studies, we are the first to introduce a pre-trained model to this task and then proposed a novel method DeepSCC. The final results show the competitiveness of our proposed method when compared to state-of-the-art baselines.

3 Method

3.1 Overview of DeepSCC

In this section, we show the framework of DeepSCC in Figure 1. In particular, we first preprocess the corpus, including data cleaning, filtering, and word segmentation. Then we fine-tune the pre-trained model RoBERTa to predict the type of programming language.

3.2 Data Preprocessing Phase

In this phase, the data cleaning and filtering are consistent with previous work [4]. However, we find that the previous code classification methods treat the code word as the basic unit. Its disadvantage is that it cannot effectively solve the out-of-vocabulary (OOV) problem. That means there exist some words, which are not in the training set but in the testing set. To solve the OOV problem, we use the Byte-Pair Encoding (BPE) proposed by Sennrich et al. [15]. It is a mixture between character-level and word-level representations. Using BPE can avoid a large number of "[UNK]" symbols in the test set, as "[UNK]" symbols may decrease the performance of the pre-trained model. For example, the original code snippet is "def split_lines(s): return s.split('\n')", and the result after using BPE segmentation is "def", "Ġsplit", "_", "lines", "(", "s", ")", ":", "Ġreturn", "Ġs", ".", "split", "(", "''", "Ċ", "''", ")". Here Ġ means that it is the first subword of a subword division.

3.3 Fine-tuning Model Phase

In this phase, we continue to pre-train the RoBERTa model on the code corpus with the MLM (mask language model) method, and then use the pre-trained model to fine-tune the code classification task. RoBERTa [6] is similar to Bert (Bidirectional Encoder Representations from Transformers) [12], and DeepSCC uses Transformer as the method's main framework because Transformer can more thoroughly capture the bidirectional relationship in the text. In particular, we treat the code as text and use the method MLM to constantly fine-tuning the roberta-base model[2] on the corpus to obtain our pre-trained language model. During the fine-tuning process, we do not tune the parameters

---

of the model's bias and LayerNorm.weight weights, and use the AdamW method to fine-tune the other parameters.

Consider that different layers of the neural network can capture different levels of syntactic and semantic information. In our study, we choose the last layer of Encoder as the feature representation of the whole code snippet, feed it into the linear layer, and obtain the model prediction label by Softmax, which can be used to calculate the cross-entropy loss with the real label. Then we use AdamW as the optimizer to perform gradient descent and back propagation to update the model parameters. Finally, we can obtain our fine-tuned model.

4 Experiment

4.1 Experimental Subject

We choose the corpus shared by Alreshedy et al. [7] as our experimental subject. Alreshedy et al. gathered code snippets from 21 programming languages (i.e., Bash, C, C#, C++, CSS, Haskell, HTML, Java, JavaScript, Lua, Objective-C, Perl, PHP, Python, R, Ruby, Scala, SQL, Swift, Visual Basic, and Markdown). After manual analysis on their gathered corpus, we find: (1) The number of the code snippets related to Markdown is only 1,210, which is significantly lower than that of other languages. (2) In the code snippets related to HTML, we find most of these code snippets also include CSS and JavaScript code segments. Therefore, we remove the code snippets related to these two languages. Finally, we use 179,556 code snippets for model training and 44,889 code snippets for model testing via stratified sampling.

4.2 Performance Measures

To compare the performance between our proposed method and the baselines, we choose the following four performance measures: Accuracy, Precision, Recall, and F1. Before introducing these measures, we first illustrate the following concepts:

- True Positive (TP): The positive sample is successfully predicted as positive.

- True Negative (TN): The negative sample is successfully predicted as negative.

- False Positive (FP): The negative sample is wrongly predicted as positive.

- False Negative (FN): The positive sample is wrongly predicted as negative.

Then the four performance measures can be computed as follows:

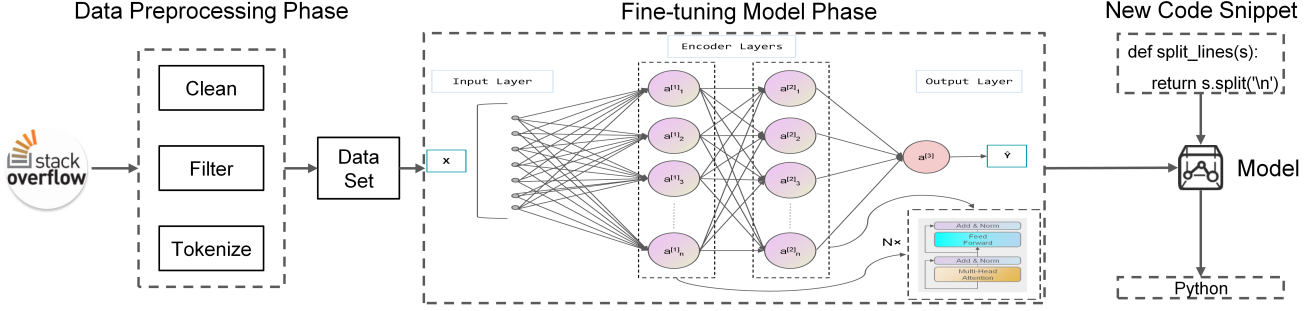$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Figure 1: The framework of our proposed method DeepSCC

$$\text{Precision } = \frac{TP}{TP + FP} \qquad (2)$$

$$\text{Recall } = \frac{TP}{TP + FN} \qquad (3)$$

$$\text{F1} = \frac{2 \times \text{ Precision } \times \text{ Recall}}{\text{Precision } + \text{ Recall}} \qquad (4)$$

Accuracy is the most intuitive performance measure, and it is the ratio of the correctly predicted observations to the total observations. Precision is the ratio of the correct predicted positive observations to the total predicted positive observations. Recall indicates how many positive examples in the sample are predicted correctly. F1 is the average of Precision and Recall.

4.3 Baselines

In the RQ, we first compare our proposed method Deep-SCC with two state-of-the-art methods from source code classification (i.e., Random Forest and XGBoost methods used in SCC++ [4]). We also choose TextCNN [8] and Transformer [11] with/without pre-trained word vectors (i.e., FastText [9] or Word2Vec [10]) from the neural text classification field as baselines. Besides, we also select BERT [12] as a baseline for the pre-trained model.

4.4 Implementation Details

In our study, we use Pytorch 1.6.0 to implement our proposed method. For baselines in the source code classification field, we run their shared code on our preprocessed corpus. For baselines in the neural text classification field, we re-implemented these baselines according to the corresponding description by Pytorch. For BERT and RoBERTa, we pre-train the model in the transformer library.

It needs to be noticed that pre-trained models (i.e., BERT and RoBERTa) use the method BPE for code segmentation by default. For other baselines, we choose the word_tokenize method provided by the NLTK library for code segmentation.

We run all the experiments on a computer with an Inter(R) Core(TM) i7-9750H 4210 CPU and a GeForce GTX3090 GPU with 24 GB memory. The running OS platform is Windows 10.

Table 1: The comparison results between DeepSCC and baselines

Method	Accuracy(%)	Precision(%)	Recall(%)	F1(%)
Random Forest	78.728	79.362	78.825	78.874
XGBoost	78.803	79.925	78.891	79.217
TextCNN	82.662	83.561	82.706	82.964
TextCNN+FastText	84.201	84.719	84.285	84.369
TextCNN+Word2Vec	84.600	85.071	84.677	84.764
Transformer	79.035	79.801	79.107	79.272
Transformer+FastText	75.624	76.026	75.526	75.986
Transformer+Word2Vec	74.325	75.050	73.243	73.765
BERT	86.946	87.292	87.004	87.116
DeepSCC	**87.202**	**87.424**	**87.276**	**87.135**

4.5 Result Analysis

Table 1 shows the comparison results between DeepSCC and baselines. Table 2 shows the detailed results for each language type in terms of three performance measures (Precision, Recall, and F1) and support (i.e., the number of code snippets related to the given programming language in the test set).

According to the analysis of the experimental results, we can find: (1) From Table 1, we can find that our method can outperform baselines and achieves the best performance in source code classification. Specifically, it can achieve a maximum performance improvement of 17%, 16%, 19%, and 18% in terms of Accuracy, Precision, Recall, and F1 respectively. The results show that the two-way transformer encoder can learn the deep semantics of the code snippets more effectively, which is helpful to obtain a better classification performance. (2) Not all the baselines in the neural text classification field outperform the baselines in the code classification field. That means some traditional machine learning methods can outperform deep learning-based methods in this task. (3) For baselines in the field of neural text classification, Transformer is not as effective as TextCNN in code classification. This may be because Transformer learns too little code semantics. After adding pre-trained word vectors (such as Word2Vec and FastText), the performance of TextCNN can be slightly improved, but the performance of Transformer is decreased. This shows that pre-trained word vectors can better capture the feature

Table 2: The detailed performance for each programming language

	Precision	Recall	F1	Support
Bash	0.89	0.84	0.87	2427
C	0.79	0.84	0.81	2396
C#	0.82	0.83	0.83	2407
C++	0.82	0.82	0.82	2442
CSS	0.85	0.89	0.87	2362
Haskell	0.91	0.94	0.93	2320
Java	0.85	0.87	0.86	2417
JavaScript	0.83	0.82	0.82	2459
Lua	0.92	0.90	0.91	1647
Objective-C	0.90	0.94	0.92	2410
Perl	0.87	0.85	0.86	2378
PHP	0.81	0.86	0.83	2455
Python	0.85	0.87	0.86	2445
R	0.92	0.93	0.92	2362
Ruby	0.90	0.85	0.88	2390
Scala	0.94	0.92	0.93	2341
SQL	0.86	0.84	0.85	2410
Swift	0.96	0.92	0.94	2474
VB.Net	0.92	0.85	0.88	2347

representation of the code when the structure is CNN in this task. (4) From Table 2, we can find that DeepSCC can achieve high performance in most of the programming languages. Then we analyze the cause of the poor performance when the programming languages are C/C++ and CSS/JavaScript. Specifically, 8% of the code snippets with the actual category of C are predicted to be C++, and 10% of the code snippets with the actual category of C++ are predicted to be C. Since C++ is almost a superset of C, this indicates that some C++ code snippets and C code snippets are indistinguishable, which poses a challenge for the source code classification problem. 6% of the code fragments with the actual category of CSS are predicted to be JavaScript, and 7% of the code snippets with the actual category of JavaScript are predicted to be CSS. Because CSS as a style language often appears in the scripting language JavaScript, it is used to dynamically update page elements. This leads to the simultaneous appearance of JavaScript and CSS in the code snippets, which also poses another challenge for the source code classification problem.

5 Conclusion

In this paper, we propose a novel method DeepSCC for source code classification, which is based on fine-tuned RoBERTa [6]. To show the effectiveness of DeepSCC, we used four widely used performance measures to evaluate the performance of DeepSCC. The results show the competitiveness of DeepSCC when compared to nine state-of-the-art baselines from the fields of source code classification and neural text classification.

References

[1] X. Chen, C. Chen, D. Zhang, and Z. Xing, "Sethesaurus: Wordnet in software engineering," *IEEE Transactions on Software Engineering*, 2019.

[2] K. Cao, C. Chen, S. Baltes, C. Treude, and X. Chen, "Automated query reformulation for efficient search based on query logs from stack overflow," in *Proceedings of the International Conference on Software Engineering*, 2021.

[3] J. N. Khasnabish, M. Sodhi, J. Deshmukh, and G. Srinivasaraghavan, "Detecting programming language from source code using bayesian learning techniques," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, 2014, pp. 513–522.

[4] K. Alrashedy, D. Dharmaretnam, D. M. German, V. Srinivasan, and T. A. Gulliver, "Scc++: predicting the programming language of questions and snippets of stack overflow," *Journal of Systems and Software*, vol. 162, p. 110505, 2020.

[5] D. Chen, X. Chen, H. Li, J. Xie, and Y. Mu, "Deepcpdp: Deep learning based cross-project defect prediction," *IEEE Access*, vol. 7, pp. 184 832–184 848, 2019.

[6] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[7] K. Alreshedy, D. Dharmaretnam, D. M. German, V. Srinivasan, and T. A. Gulliver, "Scc: automatic classification of code snippets," *arXiv preprint arXiv:1809.07945*, 2018.

[8] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pretraining of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] J. K. Van Dam and V. Zaytsev, "Software language identification with natural language classifiers," in *Proceedings of International Conference on Software Analysis, Evolution, and Reengineering*, 2016, pp. 624–628.

[14] D. Klein, K. Murray, and S. Weber, "Algorithmic programming language identification," *arXiv preprint arXiv:1106.4064*, 2011.

[15] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.