Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

The effectiveness of data augmentation in code readability classification

Qing Mi[a], Yan Xiao[b], Zhi Cai[a,*], Xibin Jia[a]

[a] Faculty of Information Technology, Beijing University of Technology, China
[b] School of Computing, National University of Singapore, Singapore

ARTICLE INFO

Keywords:
Code readability classification
Data augmentation
Generative adversarial network
Deep learning
Empirical software engineering

ABSTRACT

Context: Training deep learning models for code readability classification requires large datasets of quality pre-labeled data. However, it is almost always time-consuming and expensive to acquire readability data with manual labels.*Objective:* We thus propose to introduce data augmentation approaches to artificially increase the size of training set, this is to reduce the risk of overfitting caused by the lack of readability data and further improve the classification accuracy as the ultimate goal.*Method:* We create transformed versions of code snippets by manipulating original data from aspects such as comments, indentations, and names of classes/methods/variables based on domain-specific knowledge. In addition to basic transformations, we also explore the use of Auxiliary Classifier GANs to produce synthetic data.*Results:* To evaluate the proposed approach, we conduct a set of experiments. The results show that the classification performance of deep neural networks can be significantly improved when they are trained on the augmented corpus, achieving a state-of-the-art accuracy of 87.38%.***Conclusion:*** We consider the findings of this study as primary evidence of the effectiveness of data augmentation in the field of code readability classification.

1. Introduction

Code readability refers to a human judgment of how easy a piece of source code is to understand [1]. The research of code readability classification has drawn increasing attention from the software engineering community. To classify a source code into a *Readable* or *Unreadable* class, most prior studies built machine learning models based on a set of handcrafted surface-level features (e.g., the number of identifiers) [1,2]. While in our latest research [3], we proposed to introduce deep learning techniques to capture complicated features automatically from the source code. Although the experimental results showed that our approach outperformed the state-of-the-art, we argue that the model performance is still limited by the shortage of training data. Actually, there are only a few hundred human-annotated code snippets available in the literature (see Section 2.1 for details), which may not be sufficient to sustain the training process. The problem can lead to undesirable overfitting and therefore impede the model performance, which inspires this research into finding effective ways to artificially enlarge the training set, with an underlying goal of further improving the classification accuracy.

Current practice for collecting readability data is to perform a large-scale survey, inviting as many domain experts as possible to rate code snippets by readability using a five-point Likert scale [1,4]. However, the survey process is usually associated with quite a high cost [5]. Considering that it is always expensive (and sometimes impractical) to obtain adequate code snippets with manual labels for model training, we propose to augment existing readability data to support code readability classification. Specifically, the major contributions of this paper are:

- We propose a group of domain-specific transformation techniques to generate additional code snippets. We also make use of Auxiliary Classifier GANs to produce synthetic data. To the best of our knowledge, this study is the first to adapt data augmentation for code readability classification.
- We conduct a series of experiments to validate the effectiveness of the proposed approach using robust statistical tests, i.e., the Brunner-Munzel test and the Cliff's δ effect size. The empirical results show that the model trained on the augmented corpus performs significantly better on code readability classification, reaching up to 87.38% accuracy.

2. Proposed approach

We begin by briefly reviewing existing readability data. Based on these data, we design two types of augmentation schemes. The workflow of our research is illustrated in Fig. 1.

2.1. Existing readability data

We use datasets provided by previous studies as ground-truth to perform data augmentation, namely, D_{Buse}, D_{Dorn}, and $D_{Scalabrino}$. The details

* Corresponding author.
E-mail addresses: miqing@bjut.edu.cn (Q. Mi), dcsxan@nus.edu.sg (Y. Xiao), caiz@bjut.edu.cn (Z. Cai), jiaxibin@bjut.edu.cn (X. Jia).

https://doi.org/10.1016/j.infsof.2020.106378
Received 19 December 2019; Received in revised form 12 May 2020; Accepted 19 July 2020

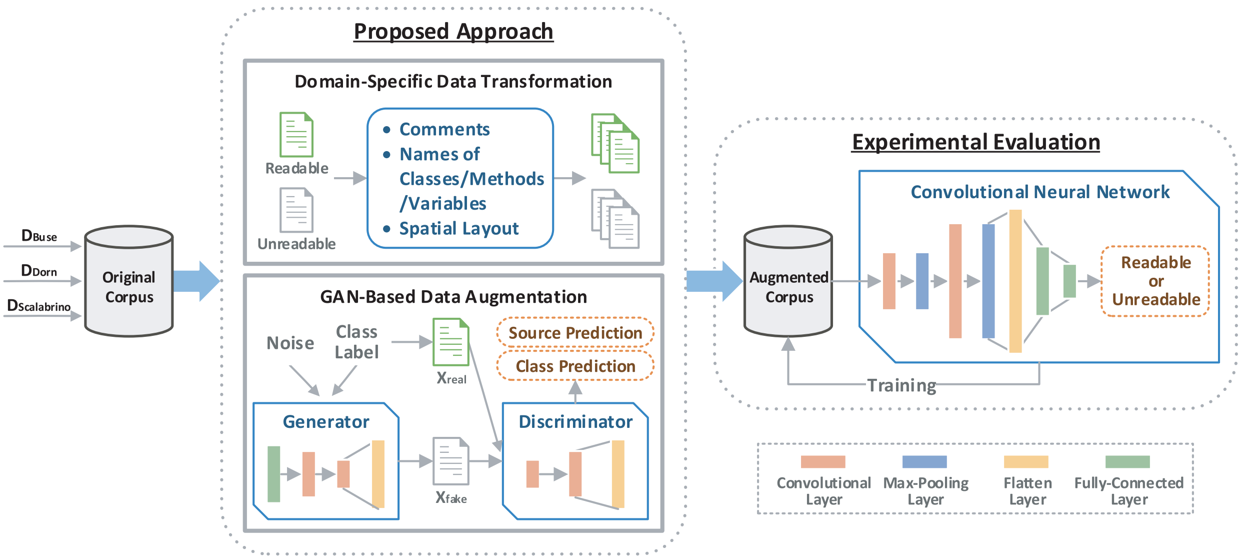

Fig. 1. Approach overview.

Table 1
Statistical summary of existing readability data.

Dataset	Source	# of code snippets	# of annotators	Avg. lines of code
D_{Buse}	Provided by Buse et al. [1]	50	120	7.80
D_{Dorn}	Provided by Dorn et al. [4]	60	5468	30.81
$D_{Scalabrino}$	Provided by Scalabrino et al. [2]	100	9	26.61

are shown in Table 1. In particular, we get a code corpus containing a total of 210 code snippets for which the readability level has been manually assessed by human annotators. The code corpus is composed of two comparison groups: the *Readable* group and the *Unreadable* group, each has equal amounts of code snippets.

Note that we mix D_{Buse}, D_{Dorn}, and $D_{Scalabrino}$ in a single corpus to sustain the training process of deep learning models. In other words, we cannot assure that the code corpus is consistent. We consider this as a possible threat to the validity of our study.

2.2. Domain-specific data transformation

Considering that data augmentation has been applied with high success for image classification, we propose to explore its effectiveness in the field of code readability classification. To augment image data, the most common strategy is to use a combination of transformations, such as rotating, shifting, and cropping. However, we cannot simply apply these transformation techniques on code snippets, mainly because source code is context-sensitive and naturally structured by the syntax of programming languages, it is definitely unreasonable to scale or distort a code snippet. Therefore, we have to propose some domain-specific transformation techniques according to our purposes.

It is generally accepted that naming conventions and comments have significantly positive impact on code readability [2,6]. Additionally, programs that are well indented or spaced are more readable than those that do not [1,4]. Based on these domain-specific conclusions, we manipulate code snippets from three aspects: comments, names of classes/methods/variables, and the spatial layout of the source code. In other words, we would like to make the readability level of code snippets increase or remain unchanged in the *Readable* group, while decrease or remain unchanged in the *Unreadable* group. The intent is to create transformed versions of code snippets that belong to the same class as the original ones. In order to do so, we propose the following label-preserving transformation techniques with respect to each group.

- The *Readable* group: (1) Add some comments; (2) Apply naming conventions (either camelCase or under_score style) to names of classes/methods/variables; (3) Indent source code consistently. This is to generate synthetic code snippets with same or higher readability than original ones.
- The *Unreadable* group: (1) Remove some comments; (2) Replace names of classes/methods/variables with meaningless strings of alphanumeric characters; (3) Randomly remove some whitespaces (e.g., horizontal tabs and line terminators). This is to generate synthetic code snippets with same or lower readability than original ones.

Note that both readability and understandability are subjective concepts [7]. While readability focuses more on the visual appearance of source code [1], which serves as a basic prerequisite for understandability. Although we have made our best to manipulate code snippets only in terms of readability, we cannot exclude the possibility that some transformations affect understandability as well. This could be considered as a possible threat to the validity of our study.

2.3. GAN-based data augmentation

In addition to basic transformations, we also experiment with Generative Adversarial Networks (GANs) to augment existing readability data. Because GANs have made great success in the area of image synthesis, we propose to treat code snippets as images and then utilize GANs to produce new samples through adversarial training. In particular, we adopt Auxiliary Classifier GANs (AC-GANs) that extend the basic architecture of GAN models with an auxiliary classifier to stabilize the training process [8]. As shown in Fig. 1, AC-GANs consist of a *Generator* and a *Discriminator* competing with each other. The *Generator* is composed of a fully-connected layer, two convolutional layers, and a flatten layer successively, which uses the noise z and the class label c as the input to generate fake code snippets $X_{fake} = G(z, c)$. The *Discriminator* is composed of two convolutional layers followed by a flatten layer, which makes predictions on sources and class labels of given code snippets. The loss function is then given by:

$$L_S = E\left[\log P\left(S = real \mid X_{real}\right)\right] + E\left[\log P\left(S = fake \mid X_{fake}\right)\right] \tag{1}$$

$$L_C = E\left[\log P\left(C = c \mid X_{real}\right)\right] + E\left[\log P\left(C = c \mid X_{fake}\right)\right] \tag{2}$$

where L_S refers to the log-likelihood of the correct source and L_C refers to the log-likelihood of the correct class. The min-max game is thus approximated by training *Generator* to maximize $L_C - L_S$, while training *Discriminator* to maximize $L_C + L_S$.

3. Experimental setup

To explore whether the proposed approach can reduce the risk of overfitting and help improve classification accuracy, we plan to conduct a series of experimental evaluations. In particular, we aim to answer the following RQs:

- RQ1: To what extent does data augmentation help improve code readability classification?
- RQ2: How do different levels of data augmentation influence model performance?

RQ1 is to verify whether data augmentation can enhance model performance when used for code readability classification, which is the main purpose of this research. RQ2 is to investigate how model performance varies with different training set sizes.

3.1. Classification model

To evaluate our approach, we implement a Convolutional Neural Network as the classifier and then feed the augmented corpus (containing both real and synthetic data) into the classifier for training. As shown in Fig. 1, the network has a relatively simple architecture, which is composed of two sets of convolutional and max-pooling layers, followed by a flatten layer and two fully-connected layers. The network takes code snippets of size 50×305 as input. While training, we use Adam as the optimizer and binary cross-entropy as the loss function. For fair comparison, we do not perform any dataset-specific tuning.

3.2. Performance measure

Analogous to previous code readability studies [1,2,4,5], we report our results in terms of classification accuracy, which refers to the percentage of instances that are classified correctly. Specifically, we first divide the code corpus (see Section 2.1 for details) into two parts: 80% for training (as well as for data augmentation) and 20% for testing. Then we train the classifier using the (original or augmented) training set. After that, we test the trained classifier with code snippets in the test set and record its accuracy. We repeat the whole process five times and report the overall average accuracy as the final result.

3.3. Analysis procedure

To address RQ1, we train the classifier on the original and the augmented corpus respectively. Then we employ the Brunner-Munzel test (significance level $\alpha = 0.05$) to examine whether there is a significant difference between the classification accuracy obtained with and without synthetic data. To measure the magnitude of the difference, we compute Cliff's δ effect size as recommended by Kitchenham et al. [9]. The value of δ is interpreted as follows: *Negligible* for $\delta < 0.112$, *Small* for $\delta < 0.276$, *Medium* for $\delta < 0.428$, and *Large* for otherwise.

To address RQ2, we vary the augmentation level by creating different sizes of training set using the proposed approach and evaluate their effects on model performance.

In addition, we adopt SamplePairing [10] as the baseline for comparison in all RQs. SamplePairing is a simple data augmentation technique that synthesizes new samples by taking an average of two code snippets randomly selected from the training set.

4. Results and discussion

In this section, we present experimental results with respect to each RQ and discuss the findings. For simplicity, we denote the size of the original corpus as *N*.

RQ1: To what extent does data augmentation help improve code readability classification?

This RQ aims to determine whether model performance is affected positively or negatively by the addition of synthetic data. We observe that the model trained on real data (of size *N*) can correctly classify 80.71% of code snippets in test set. When we include synthetic data generated by basic transformations (of size *N*) and Auxiliary Classifier GANs (of size *N*) respectively, the accuracy increases to 87.38% and 83.81%. The *p*-value for the Brunner-Munzel test is 0.00 (< 0.05) and 0.02 (< 0.05), while the *d*-value for Cliff's δ effect size is 0.82 (*Large*) and 0.42 (*Medium*), both implying statistically significant improvements in code readability classification as compared with using only real data. Besides, it should be noted that the classification accuracy with SamplePairing (of size *N*) is 82.38%, which is not significant (*p*-value = 0.26).

The empirical results show that we can obtain 3% to 7% increase in classification accuracy after employing data augmentation approaches. While we have expected performance gains through the use of synthetic data, we are surprised at the magnitude of the gains. We consider that

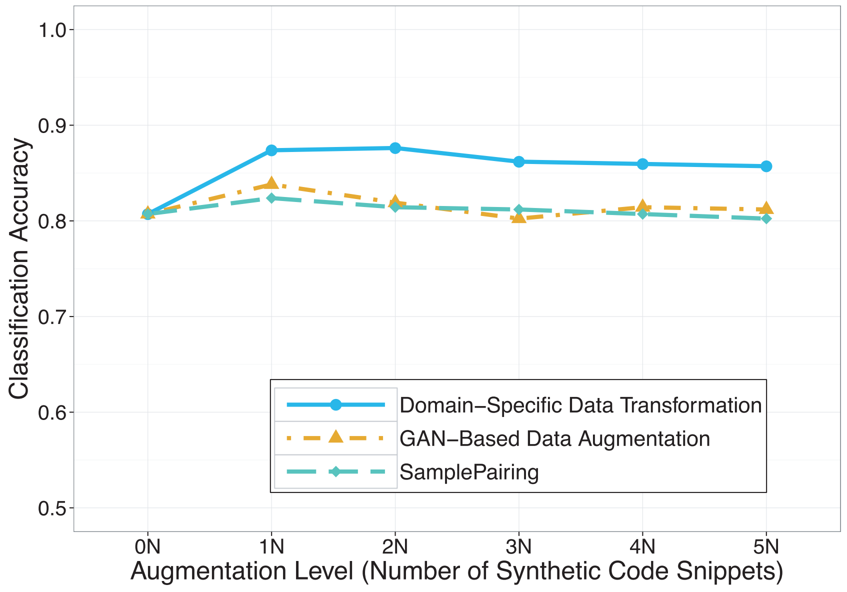

Fig. 2. Classification performance for different training set sizes.

there are two main reasons why data augmentation is beneficial: (1) It helps to increase the size of training set, while the performance of deep learning models often improves when they are trained with more data. (2) It helps to improve the generality and versatility of deep learning models by creating wider variations in training set.

RQ2: How do different levels of data augmentation influence model performance?

This RQ aims to investigate the optimal augmentation level. To achieve this goal, we build augmented corpuses of different sizes (from *1N* to *5N*) and compare their effects on model performance. As shown in Fig. 2, the classification accuracy is plotted regarding each training set size. It can be observed that: (1) The optimal augmentation level is *2N* for domain-specific data transformation and *1N* for GAN-based data augmentation. (2) The model performance varies within a small range (less than 10 percentage points). (3) A performance degradation appears when we increase the proportion of synthetic data. The caveat is that adding too much synthetic data may be unhelpful for code readability classification.

Note that all our experiments are conducted using a simple Convolutional Neural Network (ConvNet) as the classifier. This is mainly because the focus of this study is to evaluate the feasibility and validity of data augmentation approaches. We are not aiming for the best model. Therefore, we consider a simple ConvNet to be sufficient for our empirical evaluations. In fact, our experiments show that a simple ConvNet with little hyper-parameter tuning is able to yield comparable results.

5. Conclusions and future work

In this pioneering research, we investigated different strategies to augment existing readability data to support code readability classification. The experimental results showed that deep neural networks trained with the augmented corpus performed significantly better than those trained with only real data. The improvement in accuracy ranges from 3% to 7%, confirming the notable effectiveness of data augmentation approaches in code readability classification.

Our work is a first step toward artificially increasing the amount of readability data to improve code readability classification. In the future, we will extend the proposed approach and evaluation methods in a systematic manner. Besides, we plan to continue our investigation on more ground-truth data and other classifiers. With larger datasets and more complex deep learning models, we are likely to achieve better classification performance.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported by the Beijing Municipal Natural Science Foundation (No. 4202004).

References

[1] R.P.L. Buse, W.R. Weimer, Learning a metric for code readability, IEEE Trans. Softw. Eng. 36 (4) (2010) 546–558, doi:10.1109/TSE.2009.70.

[2] S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, R. Oliveto, Improving code readability models with textual features, in: 2016 IEEE 24th International Conference on Program Comprehension (ICPC), 2016-July, IEEE, 2016, pp. 1–10, doi:10.1109/ICPC.2016.7503707.

[3] Q. Mi, J. Keung, Y. Xiao, S. Mensah, Y. Gao, Improving code readability classification using convolutional neural networks, Inf. Softw. Technol. 104 (November 2017) (2018) 60–71, doi:10.1016/j.infsof.2018.07.006.

[4] J. Dorn, A General Software Readability Model, University of Virginia, Charlottesville, Virginia, 2012, pp. 1–62.

[5] D. Posnett, A. Hindle, P. Devanbu, A simpler model of software readability, in: Proceeding of the 8th Working Conference on Mining Software Repositories - MSR '11, 11, ACM Press, New York, New York, USA, 2011, p. 73, doi:10.1145/1985441.1985454.

[6] D. Binkley, M. Davis, D. Lawrie, C. Morrell, To camelcase or under_score, in: 2009 IEEE 17th International Conference on Program Comprehension, IEEE, 2009, pp. 158–167, doi:10.1109/ICPC.2009.5090039.

[7] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, R. Oliveto, Automatically assessing code understandability: how far are we? in: ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp. 417–427, doi:10.1109/ASE.2017.8115654.

[8] A. Odena, C. Olah, J. Shlens, Conditional image synthesis with auxiliary classifier GANs, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, JMLR.org, 2017, pp. 2642–2651.

[9] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, A. Pohthong, Robust statistical methods for empirical software engineering, Empir. Softw. Eng. 22 (2) (2017) 579–630, doi:10.1007/s10664-016-9437-5.

[10] H. Inoue, Data Augmentation by Pairing Samples for Images Classification (2018).