CodeStylist: A System for Performing Code Style Transfer Using Neural Networks

Chih-Kai Ting[*1], **Karl Munson**[*1], **Serenity Wade**[*1], **Anish Savla**[*1], **Kiran Kate**[2], **Kavitha Srinivas**[2]
[1] University of California Santa Cruz
[2] IBM Research
cting3@ucsc.edu, ksmunson@ucsc.edu, swade1@ucsc.edu, ansavla@ucsc.edu, kakate@us.ibm.com, kavitha.srinivas@ibm.com

Abstract

Code style refers to attributes of computer programs that affect their readability, maintainability, and performance. Enterprises consider code style as important, and enforce style requirements during code commits. Tools that assist in coding style compliance and transformations are highly valuable. However, many key aspects of programming style transfer are difficult to automate, as it can be challenging to specify the patterns required to perform the transfer algorithmically. In this paper, we describe a system called CodeStylist which uses neural methods to perform style transfer on code.

Introduction

Style is an important component of natural language. Changing the style of text involves shifting its tone while keeping the underlying information intact. Similarly, programming languages have style (Kernighan and Plauger 1982), with many enterprises enforcing style requirements to make code more readable (dos Santos and Gerosa 2018), maintainable, and performant. Some coding style attributes are related to the visual appearance of the code such as line length, indentation, blank lines, etc. However, many key style attributes are beyond code formatting. For example, the use of functional features such as list comprehensions is a well-known element of the 'Pythonic' style (Alexandru et al. 2018). We focus our discussion of coding style on such attributes.

As examples of more complex code style requirements, enterprises frequently encourage the use of comments and documentation about classes and methods. Certain language guidelines such as casing for method or variable names are also examples of style that are frequently considered desirable. Use of some language-specific features such as using generators, or object oriented techniques to encapsulate data (e.g., use of classes) are often part of style guidelines. None of these types of style changes can be performed fully automatically by state of the art symbolic methods; hence IDEs such as Eclipse or PyCharm do not offer these transformations. We describe a system in this work called CodeStylist which uses neural methods to perform style transfer on code.

To build neural style transfer models for code, we rely on extensive work in code language modeling, where large numbers of programs from open source repositories are used to pre-train models over code (Ahmad et al. (2021), Wang et al. (2021), Feng et al. (2020)), and these models have in turn, been used in a variety of downstream tasks such as code translation, or generation of code from natural language (Szafraniec et al. (2022), Chen et al. (2021)). To our knowledge, CodeStylist is the only system that exploits these language models to perform the rather complex task of code style transfer. Our approach is based like many others in this space fine tuning Wang et al. (2021) using a multi-task training regimen, where the system is simultaneously trained on transforming multiple aspects of code. We also have models specific to each style transfer task, which in general perform a bit better than the multi-task training model. CodeStylist models have been trained for Python, although the process is general, and can be adapted to any programming language.

Specifically, CodeStylist can perform the following sets of style transfers:

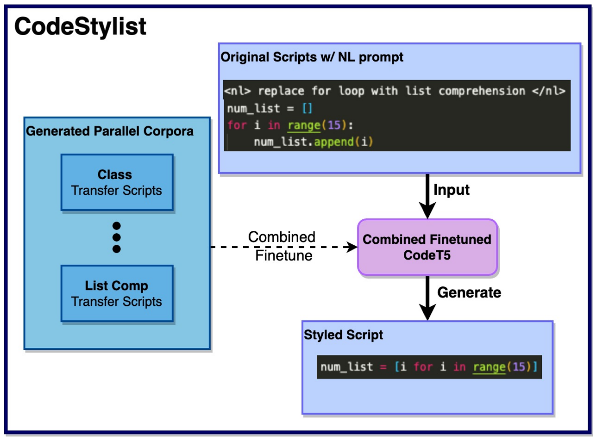
- Casing. For classes, variables and method names, convert them to snake case, as per Python PEP 8 guidelines (van Rossum Guido 2001).
- Documentation. For methods and classes in a script, add documentation.
- Comments. For methods, and overall scripts, add comments.
- List Comprehension. Translate for loops into list comprehensions, whenever it is possible to do so.
- Classes. Take a set of functions and generate a class structure from it.

Background

For more complex style related refactoring of code, most of the existing tools either require a lot of manual configuration or are semi-automatic (Mens and Tourwe 2004). Fully-automatic tools are limited in the languages they support and the transformations they handle. For example, code re-formatting is mostly automatic across programming languages. However, refactoring such as variable renaming, extracting constants and methods requires users to select the code parts being modified and provide the exact details of the transformation.

Neural models have been used for text style transfer (Riley et al. 2020; Toshevska and Gievska 2021; Tikhonov et al.

[*]These authors contributed equally.


CodeStylist
Original Scripts w/ NL prompt
Generated Parallel Corpora
Class Transfer Scripts
List Comp Transfer Scripts
Input
Combined Finetune
Combined Finetuned CodeT5
Generate
Styled Script

```
<nl> replace for loop with list comprehension </nl>
num_list = []
for i in range(15):
    num_list.append(i)
```

```
num_list = [i for i in range(15)]
```

Figure 1: CodeStylist Overview

2019) and are quite effective for the task; but none to our knowledge target code style transfer.

System Description

CodeStylist takes in a code snippet and a list of transformations to perform as inputs, as shown in Figure 1. Then a model is chosen based on the selected style transformation/s and is used to generate the transformed code. Figure 1 lists few of the the types of transforms supported by the model on the left, and illustrates the actual inference of a combined model for a single prompt.

The input code is tokenized into multiple sub-words. If multiple transformations are being performed, a tokenized prompt is prepended to the input code. The model then uses the tokenized prompt and source code to generate output code such that semantic information is preserved from the input and the only changes are those corresponding to the desired target style.

Model Design

We fine-tuned CodeT5 (Wang et al. 2021) on a parallel corpora that contains source-target code pairs for each of the transformations. We used code samples from GitHub and created a parallel corpus for each transformation by finding examples of the target feature and converting it to the source feature (e.g., converting list comprehensions to for loops). The model learnt to perform the desired transformation by generating the original code given the transformed version.

For each of the performed transformations, a separate instance of the model was trained. For multiple transformations, a combined model was trained using a multi-task learning setting.

System Output

The current user interface to display the functionality of CodeStylist is in the form of a Jupyter Lab notebook. If a script is entered into a cell with code, and a transform selected, the system outputs the transformed code.

Figure 2 shows the overall output of the system, with the multi-task model. Original code snippets were added in the figure to highlight the transformations performed by the model. Note how static methods are left unchanged, but `init` is changed to be a class method (although not named with the correct convention). Note also the change of the **for** loop into a list comprehension. In general, CodeStylist performs quite well, but in some cases it fails to generate the desired output.

```
class ReprGen(ReprGen):
def init(sourcefile, targetfile, tagsfile,
↪ returnfiles=True):
  data = parsefiles(sourcefile,
  ↪ targetfile, tagsfile,
  ↪ returnfiles=returnfiles)
  def init(self, sourcefile, targetfile,
  ↪ tagsfile, returnfiles=True):
    data = self.parsefiles(sourcefile,
    ↪ targetfile, tagsfile,
    ↪ returnfiles=returnfiles)
  @staticmethod
  def parsefiles(sourcefile, targetfile,
  ↪ tagsfile, returnfiles=True):
    with codecs.open(sourcefile,
    ↪ encoding='utf8') as source:
      sourcelines = []
    for line in source:
      sourcelines.append(line.split())
      sourcelines = [line.split() for
      ↪ line in source]
    with codecs.open(targetfile,
    ↪ encoding='utf8') as target:
      targetlines = []
    for line in target:
      targetlines.append(line.split())
      targetlines = [line.split() for
      ↪ line in target]
    with codecs.open(tagsfile,
    ↪ encoding='utf8') as tags:
      tagslines = []
    for line in tags:
      tagslines.append(line.split())
      tagslines = [line.split() for
      ↪ line in tags]
    return {'target':
    ↪ targetlines,'source':
    ↪ source_ines, 'tags': tagslines}
  def generate(dataobj=None):
  def generate(self, dataobj=None):
```

Figure 2: List comprehension + Class transfer examples: highlighted text is the original text, changed text follows highlighted block.

Conclusion

CodeStylist is a novel system for automatic code style transfer using neural fine tuned models. None of the style changes performed by CodeStylist are currently performed by any existing IDEs or tools. While there are clearly improvements that can be made, CodeStylist shows great promise in automating coding style transfers. Our system demo is available at Google Colab: https://tinyurl.com/code-style-demo.

References

Ahmad, W. U.; Chakraborty, S.; Ray, B.; and Chang, K. 2021. Unified Pre-training for Program Understanding and Generation. *CoRR*, abs/2103.06333.

Alexandru, C. V.; Merchante, J. J.; Panichella, S.; Proksch, S.; Gall, H. C.; and Robles, G. 2018. On the Usage of Pythonic Idioms. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2018, 1–11. New York, NY, USA: Association for Computing Machinery. ISBN 9781450360319.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. *CoRR*, abs/2107.03374.

dos Santos, R. M. a.; and Gerosa, M. A. 2018. Impacts of Coding Practices on Readability. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, 277–285. New York, NY, USA: Association for Computing Machinery. ISBN 9781450357142.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. arXiv:2002.08155 [cs.CL].

Kernighan, B. W.; and Plauger, P. J. 1982. *The Elements of Programming Style*. USA: McGraw-Hill, Inc., 2nd edition. ISBN 0070342075.

Mens, T.; and Tourwe, T. 2004. A Survey of software refactoring. *Software Engineering, IEEE Transactions on*, 30: 126 – 139.

Riley, P.; Constant, N.; Guo, M.; Kumar, G.; Uthus, D. C.; and Parekh, Z. 2020. TextSETTR: Label-Free Text Style Extraction and Tunable Targeted Restyling. *CoRR*, abs/2010.03802.

Szafraniec, M.; Roziere, B.; Leather, H.; Charton, F.; Labatut, P.; and Synnaeve, G. 2022. Code Translation with Compiler Representations. arXiv:2207.03578 [cs.PL].

Tikhonov, A.; Shibaev, V.; Nagaev, A.; Nugmanova, A.; and Yamshchikov, I. P. 2019. Style Transfer for Texts: Retrain, Report Errors, Compare with Rewrites. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

Toshevska, M.; and Gievska, S. 2021. A Review of Text Style Transfer using Deep Learning. *IEEE Transactions on Artificial Intelligence*.

van Rossum Guido, C. N., Warsaw Barry. 2001. Python PEP 8 Guidelines. https://peps.python.org/pep-0008/. Accessed: 2022-09-22.

Wang, Y.; Wang, W.; Joty, S.; and Hoi, S. C. H. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. arXiv:2109.00859 [cs.CL].