Fine-tuning codeT5 for Ruby code summarisation task

Salerno Fabio

Abstract

Source code summarisation, aims to generate concise and coherent human-readable summaries for code comprehension. CodeT5, a Transformer-based model fine-tuned for code summarisation, is evaluated on Ruby source code using three variations: CodeT5-base-multi-sum (multilingual), CodeT5-small (fine-tuned on Ruby), and CodeT5-base-multi-sum (fine-tuned on Ruby). This project aims to investigate the effectiveness of these models in summarising Ruby code and explores the potential of leveraging information from other languages to enhance summarisation.

Keywords: Source Code Summarisation, CodeT5, pre-trained Transformer model

Contents

1. Introduction

Source code summarisation is a natural language processing (NLP) task that involves generating a concise and coherent summary in human-readable language for a given piece of source code. The goal is to provide a high-level understanding of the code's functionality and purpose, making it easier for developers to comprehend and maintain the codebase. CodeT5 [6] is a Transformer-based model for code understanding and generation based on the T5 architecture; this model through fine-tuning can be applied for the source code summarisation task. CodeT5-base-multi-sum is CodeT5-base model fine-tuned on CodeSearchNet data in a multilingual training setting (Ruby / JavaScript / Go / Python / Java / PHP) for code summarisation. Ruby is a dynamic, object-oriented programming language that is designed for simplicity and productivity. It was created by Yukihiro Matsumoto in the mid-1990s with the goal of combining the best features of various programming languages. The aim of the project is to evaluate different models on the Ruby source code summarisation task. More specifically the pre-trained CodeT5-base-multi-sum (multilingual) will be compared with:

- the **Code T5-small** fine-tuned on Ruby, and
- the **pre-trained CodeT5-base-multi-sum** fine-tuned on Ruby.

Some building blocks remain the same between programming languages, so useful information from other languages could also be leveraged for the Ruby summarisation task. Overall the project aims to investigate this assumption.

2. The dataset

The Ruby source code used in this project comes from CodeXGLUE code-to-text dataset [4]. This dataset is a "filtered" version of the CodeSearchNet [2] where are removed:

- examples that codes cannot be parsed into an abstract syntax tree;
- examples that has a number of tokens of documents smaller than 3 or grater than 256;
- examples that documents contain special tokens (e.g. "img" or "https");
- examples that documents are not English.

The Ruby section of the dataset includes 26588 observations, divided in **train**(24927), **validation** (1400) and **test** (1261) set. The dataset includes several variables, but the one of interests for the project are:

- **code**: is the actual code function/method, which we aim to summarise.
- **codestring**: it's the top-level comment or the documentation string, which will be used as the ground truth during the tuning phase, and also during the evaluation phase to assess the quality of the output summarisation.

3. Fine-Tuning

The final goal of the model is generating a documentation string given some code. In this project two models were fine-tuned: Code T5-small and the pre-trained CodeT5-base-multi-sum. The first one is a base model that necessarily needs to be tuned for the summarisation task while the second one is already trained for this task but is more general because it was trained on many programming languages (Ruby, Python, Java, etc.). The pre-processing step of the dataset is common for both models, while each model will be fine-tuned appropriately taking into account its previously stated characteristics. A pre-processing function was designed to prepare the data for training. This function takes as input batches of code and docstrings and performs the tokenization and other additional operations such as: adding a prefix "Summarise Ruby" to each code variable, and also replacing the index of the padding by -100 such that they are not taken into account by the CrossEntropyLoss. The function then was applied to all the data in batches, by calling .map() on the HuggingFace Dataset object. After processing the data, it underwent fine-tuning using the PyTorch Lightning framework. The fine-tuning process was monitored using the wandb.ai platform, providing a real-time dashboard to oversee the entire procedure. The training of Code T5-small model was setted to a maximum of 15 epochs, with an early stop callback on the validation loss with a patience of 3 steps. The training stopped after 8 epochs.

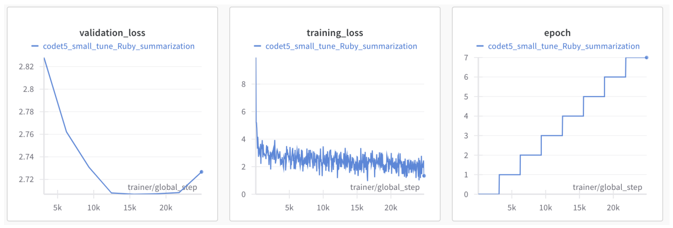

Figure 1 T5-small (tuned on Ruby) fine-tuning statistics.

CodeT5-base-multi-sum, is already tuned and architecturally is a bigger model than the T5-small (computationally more expensive to train) due these reasons it was decided to perform the tuning on 3 epochs.

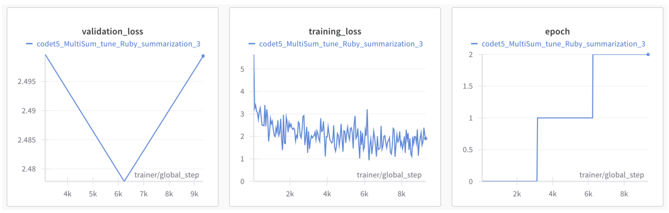

Figure 2 T5-multi-sum (tuned on Ruby) fine-tuning statistics.

Once the fine-tuning is completed, the obtained models can be evaluated.

4. Evaluation

Inference was performed with each model on the same test set. For computational reasons the methods were processed in batches of twenty observations each. A comprehensive function that covers all the inference process in batches was designed taking as inputs a set of methods and outputting the corresponding summarizations. To evaluate the results the following metrics were used:

- ROUGE [3]
- BLEU [5]
- METEOR [1]
- BERT_score [7]

The results are shown in the tables below. The metrics were calculated by evaluating the entire test set. In addition, for a more complete understanding of the results, the standard deviation is presented, which gives an idea of the variability within the test set.

Table 1 ROUGE score for each code T5 model

	ROUGE		
	Rouge1	Rouge2	RougeLsum
base-multi-sum (General)	0.23(±0.11)	0.06(±0.07)	0.21(±0.11)
small(Ruby tuned)	0.28(±0.15)	0.09(±0.01)	0.27(±0.15)
base-multi-sum(Ruby tuned)	**0.29**(±0.15)	**0.1**(±0.11)	**0.28**(±0.15)

Table 2 BLEU score for each code T5 model

	BLEU		
	Overall	Unigram	Bigram
base-multi-sum (General)	0.051(±0.031)	0.22(±0.148)	0.073(±0.056)
small(Ruby tuned)	0.080(±0.077)	0.282(±0.22)	0.086(±0.099)
base-multi-sum(Ruby tuned)	**0.083**(±0.081)	**0.3**(±0.218)	**0.138**(±0.142)

Table 3 METEOR score for each code T5 model

	METEOR
	Overall
base-multi-sum (General)	0.177(±0.11)
small(Ruby tuned)	0.227(±0.154)
base-multi-sum(Ruby tuned)	**0.234**(±0.157)

Table 4 Bert score for each code T5 model

	BERT_score		
	F1	Recall	Precision
base-multi-sum (General)	77%	77%	77%
small(Ruby tuned)	78%	**80**%	77%
base-multi-sum(Ruby tuned)	**79**%	**80**%	**78**%

The metrics computed suggest some observations. The CodeT5-small tuned on Ruby gained almost the same performances of the CodeT5-base-multi-sum pretrained (slightly better). While comparing CodeT5-base-multi-sum pre-trained with its Ruby fine-tuned version all the metrics improve, there's a 3% improvement on the Bleu, a 6% improvement on the Meteor and a 7% improvement on the RougeLsum. Taking into account the BERTScore metric, which distinguishes itself from the previous metrics by harnessing contextual embeddings from pre-trained BERT models, introduces a heightened ability to incorporate semantic and contextual similarity. Remarkably, all three models under examination achieve scores surpassing the 75% threshold. Notably, the pre-trained CodeT5-base-multi-sum model, fine-tuned on Ruby, emerges as the overall top performer with a BERTScore of 79%. Despite the apparent distinction, it's noteworthy that the disparity in BERTScore values across the various models is marginal, amounting to merely a one-percentage-point difference. Overall both three models according to BERTscore achieve a good degree of semantic and contextual similarity with the ground truth. Finally taking also into account the size of the model, the tuning, and inference computational costs; it's interesting how the The CodeT5-small tuned on Ruby is able to summarise Ruby like larger and more general models.

References

[1] Satanjeev Banerjee and Alon Lavie. "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ed. by Jade Goldstein et al. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 65–72. URL: https://aclanthology.org/W05-0909.

[2] Hamel Husain et al. "Codesearchnet challenge: Evaluating the state of semantic code search". In: *arXiv preprint arXiv:1909.09436* (2019).

[3] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013.

[4] Microsoft. *CodeXGLUE: A Benchmark Dataset and Open Challenge for Code Intelligence*. https://github.com/microsoft/CodeXGLUE/tree/main/Code-Text/code-to-text. Accessed: Month Day, Year. Year.

[5] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

[6] Yue Wang et al. *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation*. 2021. arXiv: 2109.00859 [cs.CL].

[7] Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL].