Programming by example samples extractor

Samuel Soukup

18. 09. 2023

Programming by example (PbE) assistants are an emerging tool for software development that automatically suggest new text edits based on user-provided input-output examples. For example if user does the following refactoring: `def getX()` → `def getValueX()`, we can predict for a line `def getY` a refactoring of `def getValueY` - both adding `Value` after the `get` token.

Despite this technique providing vast benefits for the users, these tools require a powerful synthesis engine to produce meaningful results. State of the art techniques still use program synthesis for approaching this problem [2]. But with neural network - based language models becoming state of the art solution for many linguistics problems [1], machine learning models could show higher performance.

For these we've create PbExtractor. A customizable tool, which leverages existing open source repositories to collect predictable edits and collects them into a dataset.

1 Algorithm overview

PbExtractor collects changes from a provided git repository and filters them. Then it reports the probable predictable results. It runs through all the diffs (change in a file from a previous commit to a new one) in a repository and for each diff we look at all 1 line edits (or first line from the new text and last line from the old text, if diff is longer than one line) and check these, whether they fulfill the following requirements:

1. Regularized Levenshtein distance between the old and new text is at most 0.5 (configurable)
2. Regularized Levenshtein distance between this edit and the first saved edit in the problem is at most 0.5 (configurable)
3. The edit is not a simple trimmed copy of the old text (e.g. '"a "→ "a„)
4. The whole problem has at least one edit, which is synthesizable (for definition see below) from the first one (first edit excluded)

2 Filtering

In the following sections, we describe:

- **Sample** as a one line from a diff (either original line or edited line)
- **Example** as a pair of input and output line (original to edited)
- **Problem** a sequence of *examples*, where each subsequent *example* meets conditions 1. and 2. from the previous section.

The most comprehensive filtering criterion is a simple program synthesis algorithm (PS). For an example to be accepted as PbE, it needs to be solvable by PS, meaning PS is able to predict at least one subsequent *example*, based on the first one collected in this *problem.*

As this is a very computationally expensive check, we first need to filter out *examples*, which have low probability of success.

3 Synthesizability

For PS we use the following approach. We split given strings to tokens by using a regex tokenizer, splitting on all whitespaces, special characters and camelCase.

For the PS programs, we have the following operations:

- `Insert(text, index)` - insert text at index
- `Delete(text, index)` - delete text at index
- `Replace(before, after, index)` - replace before with after at index

And for checking whether a given operation is applicable to a given example, we have the following conditions:

- `OnIndex(index)` - the operation is applicable on the given index
- `PreviousToken(token)` - the operation is applicable on the token before the given token
- `NextToken(token)` - the operation is applicable on the token after the given token
- `ThisToken(token)` - the operation is applicable on the given token

Then we use a simple DFS in the space of all programs to find a program, which generates the output from the input. We describe an ‘x‘ synthesizable by ‘y‘ when there exists a program ‘p‘ such that ‘p(x) = y‘.

As this framework is only used for filtering and not actual synthesizing, we do not need to take care of text transformations(`lowercase` → `UPPERCASE`, `snake_case -¿ camelCase`, ...), whitespace changes etc., as these are out of the scope of this project.

4 Dataset availability

As an example dataset, we provide a small dataset gathered from the first 200k commits from the intellij-community repository. This has 8932 different *problems*, with over 210k different pairs of examples annotated by synthesizability.

5 Python usage

We also provide a python notebook and a python script showing, how to use the dataset in python. We create simple features definitions for the generated problems and use these to

- create visualization using t-SNE and
- train a simple classifier for predicting whether a given problem is synthesizable from the first example.

For more detailed usage info see the README file in the attached repository.

6 Data and code availability

All the code and data is available in this GitHub repository.

Reference

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[2] Jiarong Wu, Lili Wei, Yanyan Jiang, Shing-Chi Cheung, Luyao Ren, and Chang Xu. Programming by example made easy. *ACM Transactions on Software Engineering and Methodology*, jul 2023.