AI for S/W Eng Assignment #1 Report
Recommending code tokens via N-gram models

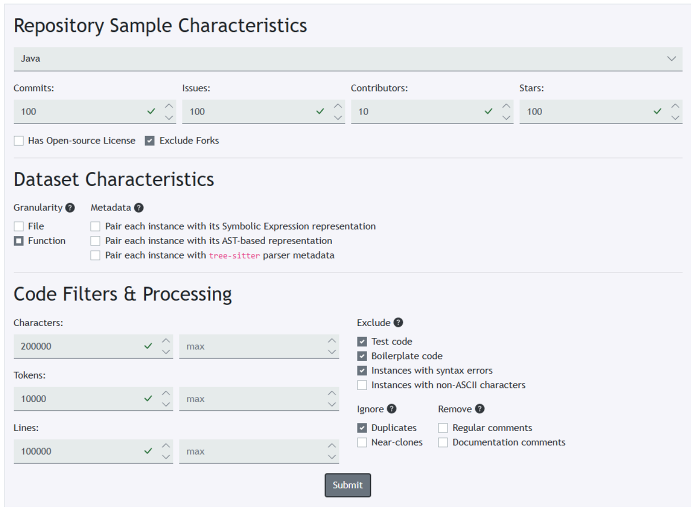
Nishat Sultana, Danny Otten, Joseph Call

Dataset Creation Process

Our initial plan was to create a new corpus of at least 25,000 Java methods, but due to DataHub creating overly large datasets or failing to generate them entirely, we had no access to data early on in the model development process. Therefore, we decided to use the example dataset generated in class on DataHub to test our N-Gram model code. This was enough to get us started in terms of:

- Data format
- Data preprocessing

We then created our own dataset on DataHub for **Java** methods using the following parameters:



Repository Sample Characteristics

Java

Commits:	Issues:	Contributors:	Stars:
100	100	10	100

☐ Has Open-source License ☑ Exclude Forks

Dataset Characteristics

Granularity
- ☐ File
- ☑ Function

Metadata
- ☐ Pair each instance with its Symbolic Expression representation
- ☐ Pair each instance with its AST-based representation
- ☐ Pair each instance with `tree-sitter` parser metadata

Code Filters & Processing

Characters:	200000	max
Tokens:	10000	max
Lines:	100000	max

Exclude
- ☑ Test code
- ☑ Boilerplate code
- ☑ Instances with syntax errors
- ☐ Instances with non-ASCII characters

Ignore
- ☑ Duplicates
- ☐ Near-clones

Remove
- ☐ Regular comments
- ☐ Documentation comments

This dataset was about 6 gigabytes when compressed and 40 gigabytes when uncompressed, which is too large to use for a full training run. Because of this, we created a subset of this file which we used to actually train our models. Our final dataset was 134 megabytes and included 30,000 Java methods.

Model Training Methodology

For simplicity, we decided to write our model code in Python. Our control file input parameters such as:

- Number of models to train and compare
- Number of grams to train the model on
- Model input data path (pre-tokenized Java code)

- Number of classes to use in the training set
- Number of classes to use in the testing set

The model training process takes on average <1 minute, and we found optimal performance when training with 3 or fewer grams. When a model is finished training, its performance will be evaluated by calculating the perplexity, and the model with lowest perplexity out of all those created will be returned as the best.

From there, the user is able to enter code snippets for code generation. The user can then interact with the model.

Code

We used a GitHub repository to manage our code during this project, which can be found here: https://github.com/dsotten/plm