1 Methodology of the N-gram Model

The methodology for training our n-gram model can be explained in three stages. Our code can be found in the publicly available GitHub repository [1].

1. Generating paths from the parsed PD files saved in our dataset
2. Preparing our training dataset and test dataset
3. Training and evaluating the performances of the n-gram models

We explain the stages in the following sections.

1.1 Generating paths from the parsed PD files saved in our dataset

We utilized the parsed contents of the PD files from our dataset [2] to generate paths from the PD source code. We extracted a list of connections between the PD file objects using the `Contents` table from our dataset for each parsed content. We constructed a directed graph using Python's `NetworkX` [3] library for each parsed content. For instance, if there was a connection from `object_0` to `object_1`, we added both `object_0` and `object_1` as nodes in the graph, if not already added, and created a directed edge between them. In this context, `object_0` represents the `source` of the connection, while `object_1` is the `destination`.

This approach allowed us to define `roots` as nodes in our directed graph that serve as sources for some connections but are not destinations themselves. Similarly, we defined `leaves` as nodes that act as destinations for some connections but never serve as sources. We generated paths with lengths less than or equal to 5 from each source to each leaf using the `all_simple_paths` [4] function from NetworkX. We only considered a subset of all paths to reduce computation time and memory usage.

1.2 Preparing our training dataset and test dataset

After employing the methodology outlined in 1.1 to generate paths from the parsed source code, 8,765,849 paths were generated from the parsed PD files, each with a path length of less than or equal to 5. Initially, we partitioned the PD projects into an 80-20 split for training and testing purposes, utilizing the `train_test_split` [5] function from the `scikit-learn` [6] library in Python.

We gathered the hashes corresponding to the revisions of the PD file contents for both the train and test projects. Following this, we refined the test hashes by eliminating any matches found in the training set to assess our models' performances on previously unseen data. Employing the paths associated with each hash value, we generated distinct training and test datasets. This process guarantees that our training and test data originate from distinct projects, minimizing overlaps between them.

After that, for training our n-gram model, we selected a subset of the training data to reduce the computation time and resources. We randomly shuffled the training dataset using the `shuf` command in Linux and extracted 50,000 paths for training our n-gram model. Similarly, we selected 5,000 path data from our shuffled test dataset for evaluating the models' performances.

We applied padding to each path in the training data by adding start and end tokens. Specifically, we employed `<s>` as the start token and `</s>` as the end token. Each path was padded with 10 start tokens and 10 end tokens. Similarly, in our test data, each path was padded with 10 start tokens. This padding was applied to both the training and test data to ensure that the model possesses adequate context at the beginning and end of each sequence.

1.3 Training and evaluating the performances of the n-gram models

We developed 7 n-gram models with orders 2, 3, 4, 6, 8, 10, and 15, utilizing modified Kneser-Ney smoothing, and subsequently trained them on our training data. For this purpose, we employed the modified MIT Language Modeling (MITLM) toolkit, used by Campbell *et al.* [7, 8]. To evaluate our models, we utilized our padded test data of length `n`, where we considered the first `n-1` tokens as the context for the models and the last token as the true next token. We applied the `predict` function of the modified MITLM, as implemented by the authors, to predict the next token given a context. By comparing the true next token with the predicted next token, we computed accuracy, precision, recall, and f1 score for each model.

Additionally, we opted for `weighted` and `np.nan` as the values for the `average` and `zero_division` parameters, respectively, within the precision, recall, and f1 score calculation functions from the scikit-learn library. The selection of `np.nan` handles instances of zero division during the computation of these metrics by disregarding them within the averaging process. Similarly, the choice of `weighted` for the average parameter ensures that metrics are computed for each label and subsequently averaged, with the average weighted by support [9]. The performance results of these n-gram models are depicted in Figure 1.

As shown in Figure 1, there is no significant change in the models' performances after the 4-gram model. In other words, utilizing higher-order n-gram models does not alter the outcomes for the test data. The highest precision score, reaching 37.01%, is observed in the 4-gram and other higher-order models. Meanwhile, the highest accuracy, recall, and f1-score at 24.57%, 24.57%, and 29.71%, respectively, are achieved by the 3-gram model. Notably, identical accuracy and recall values are observed across all models.

2 Limitations

The following factors might question the validity of our methodology.

2.1 Subset of all paths

In the first and second phases of our methodology, during the generation of paths from the parsed source code, we specifically focused on a subset of these paths. The `cutoff` parameter within the `all_simple_paths` function of NetworkX disregarded

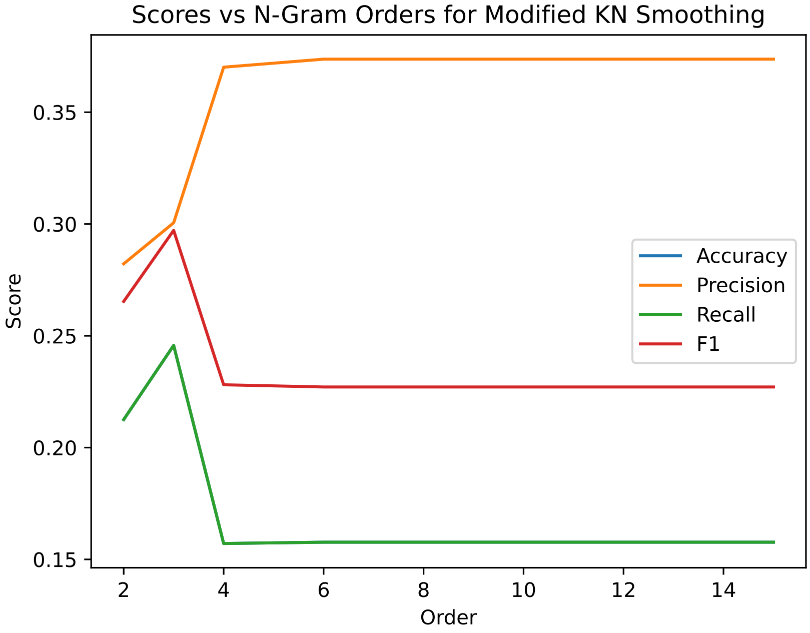

Figure 1: Performance comparison between different n-gram models

paths exceeding a certain length determined by the cutoff value. For instance, when presented with two paths: `[0,1,2,3]` and `[0,3]`, and a cutoff parameter set to 2, the `all_simple_paths` function will discard the first path and only retain the second. We also considered a subset of the training and test data for training and evaluating our n-gram models. Consequently, these actions restrict the breadth of context in our dataset for subsequent stages.

2.2 Unsuccessful next token prediction for some sequences

The `predict` function within the modified MITLM code encountered difficulty in predicting the next token for certain paths. Specifically, for higher-order n-grams (≥ 4), the modified MITLM could predict the next token for 3,399 out of 5,000 test paths but encountered failures for some. For instance, the model could not predict the next token for a padded sample path with the following context: `<s> <s> <s> <s> <s> <s> <s> <s> <s> <s> floatatom floatatom pack send msg`.

References

[1] Anisha Islam. Code for the n-gram. https://github.com/anishaislam8/Using-PyMITLM-for-next-token-prediction. Accessed: 2024-02-19.

[2] Anisha Islam. Opening the Valve on Pure-Data Dataset. https://archive.org/details/Opening_the_Valve_on_Pure_Data, 2023. Accessed: 2024-02-19.

[3] NetworkX Developers. NetworkX. https://networkx.org/. Accessed: 2024-02-19.

[4] NetworkX Developers. all_simple_paths. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html. Accessed: 2024-02-19.

[5] scikit-learn developers (BSD License). sklearn.model_selection.train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. Accessed: 2024-02-19.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Joshua Charles Campbell, Abram Hindle, and José Nelson Amaral. Syntax Errors Just Aren't Natural: Improving Error Reporting with Language Models. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, page 10, 2014.

[8] Bo-June Hsu and James Glass. Iterative language model estimation: efficient data structure & algorithms. In *Ninth Annual Conference of the International Speech Communication Association*. Citeseer, 2008.

[9] scikit-learn developers (BSD License). sklearn.metrics.precision_recall_fscore_support. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html. Accessed: 2024-02-19.