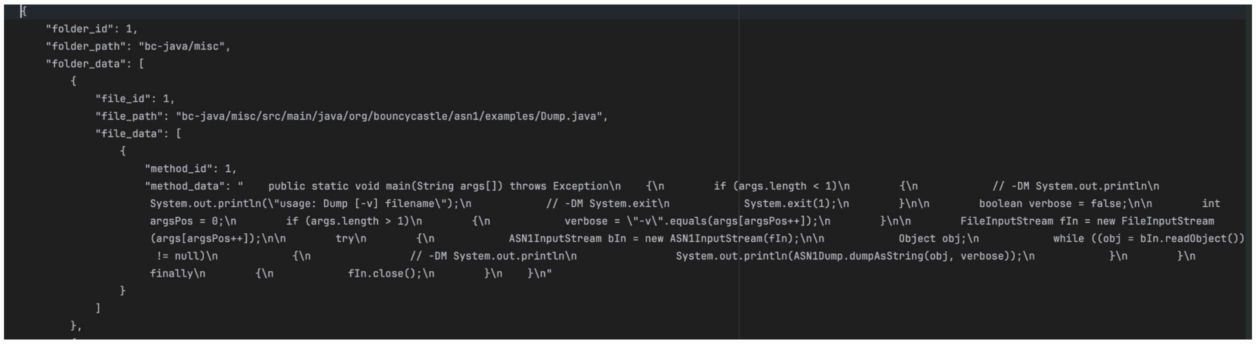
Recommending code tokens via N-gram models

Mehedi Hasan Sun, Liakot Khan Babu

Data Collection:

We used the online github search tool https://seart-ghs.si.usi.ch to find a java repository for our dataset. Using this tool we selected a repository named bc-java containing 956,735 codelines. To extract java methods from this vast repository, we used a python package called javalang. We loop through the entire dataset and collected about 48k java methods to train our model. Here is a sample of the collected dataset :



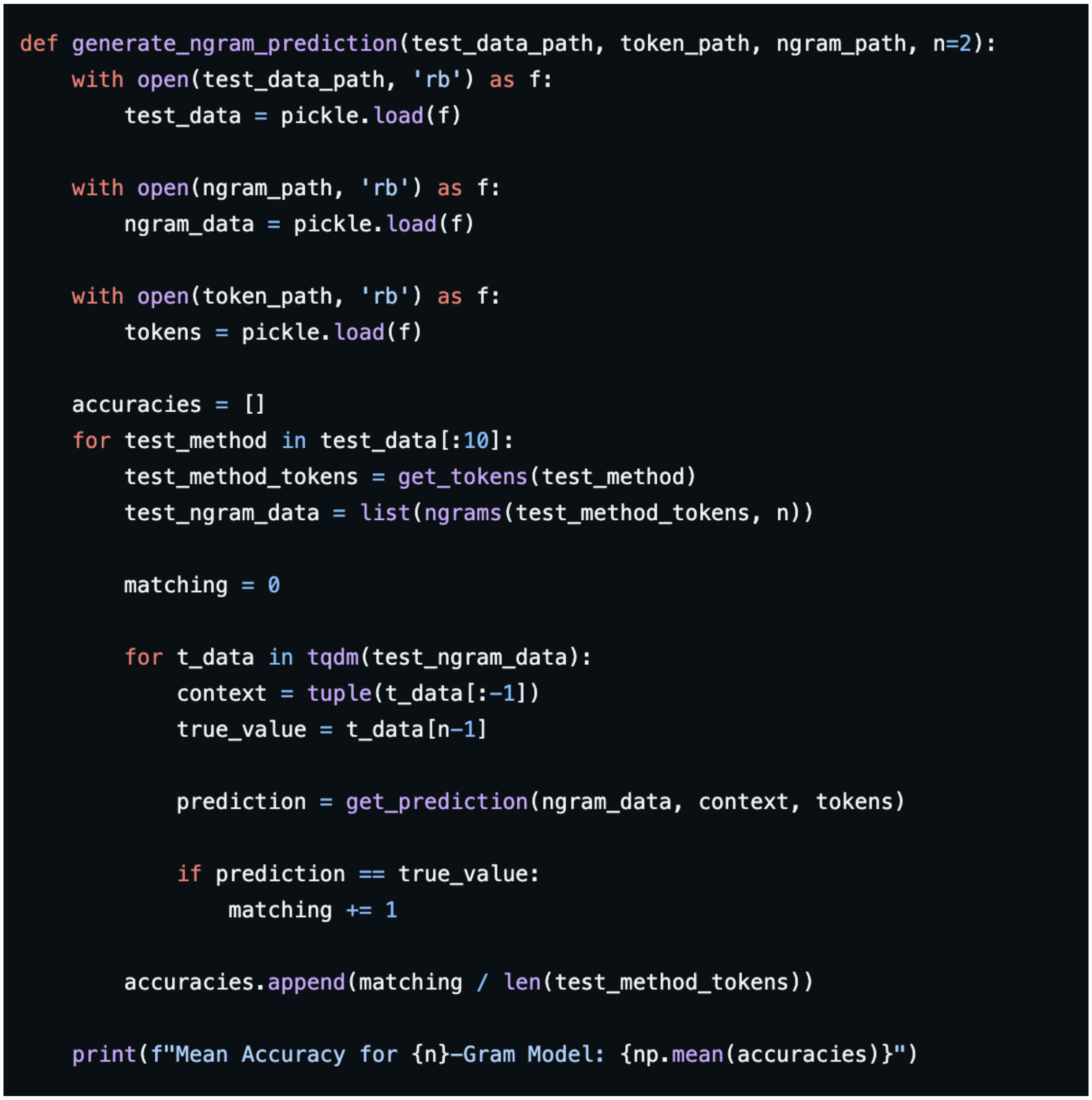
```
{
    "folder_id": 1,
    "folder_path": "bc-java/misc",
    "folder_data": [
        {
            "file_id": 1,
            "file_path": "bc-java/misc/src/main/java/org/bouncycastle/asn1/examples/Dump.java",
            "file_data": [
                {
                    "method_id": 1,
                    "method_data": "    public static void main(String args[]) throws Exception\n    {\n        if (args.length < 1)\n        {\n            // -DM System.out.println\n
                     System.out.println(\"usage: Dump [-v] filename\");\n            // -DM System.exit\n            System.exit(1);\n        }\n\n        boolean verbose = false;\n\n        int
                     argsPos = 0;\n        if (args.length > 1)\n        {\n            verbose = \"-v\".equals(args[argsPos++]);\n        }\n\n        FileInputStream fIn = new FileInputStream
                     (args[argsPos++]);\n\n        try\n        {\n            ASN1InputStream bIn = new ASN1InputStream(fIn);\n\n            Object obj;\n            while ((obj = bIn.readObject())
                      != null)\n            {\n                // -DM System.out.println\n                System.out.println(ASN1Dump.dumpAsString(obj, verbose));\n            }\n        }\n
                     finally\n        {\n            fIn.close();\n        }\n    }\n"
                }
            ]
        },
```

Data Preprocessing:

To prepare our dataset for the model, we have used few nlp techniques. First, we used a regex method to remove all the single line and multi-line comments from our dataset. Then we used the modified dataset to tokenize. Instead of using the nltk package to tokenize our dataset we have used a regex that we collected from stackoverflow. This method efficiently tokenized the entire dataset. For our model we split the dataset into a test and train set. For training we selected 90% of our data. And for testing we kept 10% of our data.

Model:

We implemented our ngram model using the python NLTK package. To find the best model for our dataset we counted the accuracy for several ngrams. For our dataset, n=4 gave us the best accuracy.



```python
def generate_ngram_prediction(test_data_path, token_path, ngram_path, n=2):
    with open(test_data_path, 'rb') as f:
        test_data = pickle.load(f)

    with open(ngram_path, 'rb') as f:
        ngram_data = pickle.load(f)

    with open(token_path, 'rb') as f:
        tokens = pickle.load(f)

    accuracies = []
    for test_method in test_data[:10]:
        test_method_tokens = get_tokens(test_method)
        test_ngram_data = list(ngrams(test_method_tokens, n))

        matching = 0

        for t_data in tqdm(test_ngram_data):
            context = tuple(t_data[:-1])
            true_value = t_data[n-1]

            prediction = get_prediction(ngram_data, context, tokens)

            if prediction == true_value:
                matching += 1

        accuracies.append(matching / len(test_method_tokens))

    print(f"Mean Accuracy for {n}-Gram Model: {np.mean(accuracies)}")
```