Outline of Data Collection and raining Methodology

Sean McClary

Tools implemented:

- GHS database search
- Jsonl file formatting
- GitHub API for python
- Javalang python library

Data Collection Overview:

File: corpus.py

1. Used filters to find repos with a high volume of java content on GHS.
2. Mined the repositories using GitHub API for python.
    a. Functions:
        i. mining()
        ii. get_files(repo).
    b. Opened repo, then used nested loops, searched through the directory for java files.
    c. Rejected files containing the word “test” to keep quality.
3. Refined the data into usable corpus and test sets.
    a. Functions:
        i. tokenize(input_file, output_file)
        ii. remove_junk(input_file, output_file
        iii. just_tokens(input_file,output_file, test_file)
    b. Tokenized all the files, which also removes comments
    c. Removed package and import phrases from files
    d. Split the files, and normalized the corpus
4. Revert the test set into java files
    a. File: testfiles.py
    b. This was so the model would require .java files as input for testing so alternative files can be used in testing.

Model Training:

File: model.py

1. I created functions to read in the corpus and test files, so that if substitutions were desired, little would need changing. Normalization of test files occurred here.
    a. GetCorpus(input_file, corpus_size)
    b. GetTest(test_file)
2. N gram creation consisted of pure python logic.
    a. For each series of tokens, every occurrence of a set number of sequential tokens is stored in a dictionary.
        i. GetNgrams(token_lists, n)
    b. A separate function can then calculate the probability distribution for the n grams. It is separate so each component can be done independently, and for easier testing in development of the app.
        i. GetProbs(ngrams)
    c. The model would be created as an empty list. For each instance of a context window in a test, the model corresponding to the context window was placed in the list at the index of the context window size.
        i. This allows for a single model to be created and used in repeated testing upon the same set.
            1. Model(corp_size,cw)
3. The prediction happens independently and one token at a time.
    a. PredictNext(sequence, probs, cw)
4. An overarching test function in model.py calls each of the preceding functions in turn.
    a. Test()
    b. This is where user configuration occurs.
    c. Multiple tests of different context window sizes occur at once, and the results are printed.
    d. The corpus size can be changed, but extensive testing has shown a plateau at the foreseen 500 classes mark.