Note: I didn't use command line in my code. So just run it directly, because I've set all the parameters in the code. If you want to change the parameters, please read the following instructions.

Also, Ngram.py is used to generate some intermediate variables and implement some functions called by other python files. But you can still run the code to take a look at those variables.

Files:
q3_lm: The original data, including test_set, train_set and dev_set.
Add1-Generate_Sentence.txt: 20 sentences I generated using add-1 smoothing.
Simple_Interpolation_GenerateSentece.txt: 20 sentences I generated using simple interpolation smoothing.

There are 5 python files, specifically:

1. Ngram.py: Read the file, and process the text data to N-gram Model. This program is mostly for computing perplexity and generate sentence.

a. Parameters: The parameters in this file include training file path(fpath) and N for N-gram.

b. Function process: Firstly given fpath, function Read_train_data and Read_test_data will return the vocabulary, the size of vocabulary and a list with the length of the number of data, in which each sub-list contains the processed word of each sentence.
   eg: [['UNK','okay'], ['so'], ['guess'],...].
   The main difference between Read_train_data() and Read_test_data() is that for Read_train_data I use UNK to replace some infrequent words, while in Read_test_data I don't do that. Also, in Ngram.py I didn't use Read_test_data(), and I'll import this function when computing perplexity.

   Function Ngram() is to convert the processed data to n-gram tokens, and we need to set N.

c. Run the code: You can just run the code since I've set all the parameters. It will print the results after applying Ngram on the data.

   Specifically, in my program, I've set the fpath to be training data path (line 28 – line 30), so you don't need to change them. Line 236 - line 256 is the part where I called the functions and output the trigram, bigram and unigram results. You can just use the parameters I've set to run the code. If you want to try to run it by yourself, first you should get the results of Read_train_data, like what I did between line 236 and line240. Then you can use Ngram() function to get the n-gram model, like what I did between line 236 and line 256.

Also, I pint some process in each step, which will help you to know which procedure it is doing during running time.

2. Add1_Perplexity.py: This code computes the perplexity using add-1 smoothing.

a. Parameters: In this code I import some variables from Ngram.py, including Read_test_data function, Vocabulary, vocabulary size and Trigram results. The only parameter you need to set in this program is the file path of test data, in line 51 and 52.

b. Function process: Firstly, I read the test data and process the test data. Then I compute Bigram tokens and their counts using training set results, and then use add-1 smoothing to compute the perplexity of each test sample with function test-Prob.

c. Run the code: Similarly, I've set all the parameter that you can just run the code. The only parameter is the test data path in line 51 and line 52. But please make sure that Ngram.py can run properly first, because here you need to use some variables of Ngram.py.

3. Add1-Generate_Sentence.py: Generate sentence using add-1 smoothing. I've store 20 sentences I generated in the Add1-Generate_Sentence.txt, but you can also run the code and see if you will get some more interesting sentences.

a. Parameters: In this code I import some variables from Ngram.py, including Vocabulary size, Bigram result, Trigram result and Unigram result. No other parameters are needed.

b. Function process: I use function get_prob() to return the probability of a trigram token, function random_pick() to randomly pick a variable according to their probability, and function Generate_S() to generate and print sentences.

c. Run the code: Well, you don't need to set any parameters, just run the code and it will give you 20 sentences generating with add-1 smoothing. This may take a little long time but just be patient. Also, make sure Ngram.py run properly since I use the variables from Ngram.py.

4. Simple_interpolation_perplexity.py: This code computes the perplexity using simple interpolation. Lambdas I use are 0.3, 0.4, 0.3.

a. Parameters: In this code I import some variables from Ngram.py, including Read_test_data function, Vocabulary, Trigram result, Bigram result and Unigram result. The only

parameter you need to set in this program is the file path of test data, in line 51 and 52.

b. Function process: Firstly, I read the test data and process the test data. Then I compute the perplexity of each test sample with function test_Prob(), using Tigram, Bigram and Unigram results.

c. Run the code: I've set all the parameter that you can just run the code. The only parameter is the test data path as I said before in line 51 and line 52. But please make sure that Ngram.py can run properly first, because here you need to use some variables of Ngram.py.

5. Simple_Interpolation_GenerateSentece.py: Generate sentence using simple interpolation smoothing. I've store 20 sentences I generated in the Simple_Interpolation_GenerateSentence.txt, but you can also run the code and see if you will get some more interesting sentences.

a. Parameters: In this code I import some variables from Ngram.py, including Bigram result, Trigram result and Unigram result. No other parameters are needed.

b. Function process: I use function get_prob() to return the probability of a trigram token, function random_pick() to randomly pick a variable according to their probability, and function Generate_S() to generate and print sentences.

c. Run the code: Well, you don't need to set any parameters, just run the code and it will give you 20 sentences generating with simple interpolation smoothing. This is much faster than add1 smoothing. Also, make sure Ngram.py run properly since I use the variables from Ngram.py.