Summary of my Ngram model:

Few important notes:

1. I have saved the training sets inside a folder and called it “sources” but no need to add anything in the command line
2. I have a flag at the top of the file “TRIGRAM” that when its True, I will train and test with trigrams, otherwise I will do bigrams only
3. I used “UTF-8” encoding for my program.
4. I used nltk.words dictionary to handle unknown tokens
5. I added start/end tokens while making bigrams/trigrams
6. I have chosen to use “good-turing” for smoothing and “interpolate” for trigrams
7. The program is not super fast and will take some time to train and also predicting without test flag.
8. Coefficients I took for interpolation are also defined at the top of the file.
9. My ngram function is written specifically for bigrams and trigrams since I add start and end strings while making these ngrams

How to run the program and what to expect:

With “-test” flag:

The program will start building the language models based on text files in the “authorlist.txt”.
I am saving the files from authorlist inside a “sources” folder to look more organized. But you don’t need to add “sources/…” to give program the directory, I have taken care of that inside the code.
I’m assuming every line is a sentence, so I won’t do sentence tokenization, instead I just remove punctuations or any special characters and save everything in lowercase. Then I test every line for all language models that I have built and save the probabilities inside of an array. The highest number will be the author.
The program doesn’t keep track of the whole test file and instead prints the result for each line (as suggested by HW writeup)

Without “-test” flag:

The program will create 2 different files with ”_train” and “_dev” added to the end of the file name.
I will count the paragraphs of the original file, and will pick 20% of the paragraphs randomly.
I will then create 2 different files with ”_train” and “_dev” added to the file name inside sources folder and will write that random 20% of the file in dev set, and rest in train set.
I will then train my language model based on the new train set and will test the new dev set.
Sentence tokenization will be done on dev set and will then calculate the probabilities for each sentence.
*Note that this part will take a while and at the end the prediction percentage for each dev set will be printed out.

Language Model:

I have made a class called Ngram_Model and will make an instance of it while training.
It consists of sentence and word tokenization, vocabulary, unigram/bigram/trigram frequency, probability calculation and finally NCount and “good-turing” for smoothing.
I am using nltk library sentence/word tokenization methods and will replace punctuations and special character after tokenization using regex library.
By using ntlk words corpus, I will make a vocabulary for each language model, will automatically add “<s>” and “</s>” as start and end tokens and “junk” as unknown token to the vocabulary.
At the same time I will make a dictionary of unigram frequency, and for each token that doesn’t exist in the nltk.words() dictionary, I will increment “ink” token.
Respectively bigram and trigram list and frequency dictionary will be made.

Tokenization:

I am using nltk sentence and word tokenization methods and will replace all punctuations and special characters after tokenization. Everything in my language model will be in lowercase.
The encoding I am using is “UTF-8”.

Smoothing:

While running program with bigrams only, I am using a smoothing threshold which is defined at the top of the file and its equal to 6 (based on our discussions in lectures).
While counting the bigrams, if the frequency is less or equal to 6, I will use “good-turing” smoothing method.
Ncount method inside my class is for calculating Ns which would be useful for smoothing.

Results:

The program will take a while to do the prediction for this part (apologies…)

With “-test” flag:

Using bigrams only, here are my results:
Ran “austen_test_sentc.txt” provided by Dr.Exley (removed some lines to make the file smaller), and my correct prediction for Austen was 1577 / 3067 = 51%
Using trigram with mix interpolation and “good-turing” smoothing, here are my results:
Ran “austen_test_sentc.txt” provided by Dr.Exley (removed some lines to make the file smaller), and correct prediction for Austen was 1410 / 3067 = 45%

Without “-test” flag:

Using bigrams only, here are my results: (This takes a lot of time!)
Austen: 439 / 879 49.9%
Dickens: 267 / 991 = 26.9%
Tolstoy: 615 / 1759 = 34.9%
Wilde: 319 / 1108 = 28.8%

Using trigram with mix interpolation and “good-turing” smoothing, here are my results:
Austen: 396 / 860 = 46%
Dickens: 220 / 1054 = 20.8%
Tolstoy: 436 / 1628 = 26.79%
Wilde: 353 / 1017 = 234.7%

Tricks:
1. Since I chose UTF-8 encoding, I saw some special characters like ´ , so while tokenization I replaced all special characters with regex and even for clitics whether during training or testing, I removed them all.
2. There are several places that I added error handlers specially when ngram wasn’t seen before and dictionary was returning None and subsequently I was getting KeyErros.
3. I added ZeroDivisionError handler for probabilities just in case.
4. I am mixing “interpolation” with “good-turing” when TRIGRAM is True, meaning not only I account for lambdas for all 3 probabilities, but also if the frequency is below my smoothing threshold for trigrams and bigrams, I will use “good-turing” to calculate my probability.