Language Modelling and Tokenization

Name	Rhythm
Roll No.	2021101081

Task

The assignment required us to build tokenizer for the corpuses provided and build language models using **Linear Interpolation** and **Good Turing Smoothing** techniques on top of that. The task also involved generation of most probable sequences of words given a context sequence.

Corpus

- Pride and Prejudice
- Ulysses James Roy

Tokenization

The corpus were taken as input and broken into sentences and further into tokens using regex. Number of sentences in *Pride and Prejudice* were ~7000 and in *Ulysses James Roy* about ~22000.

Further, numbers, urls, mentions and mail IDs were converted into placeholders to better the tokenization and remove once-occuring unique words that convey no information. Some modifications were made on the text to better of the sentence tokenization such as modifying 'Mr. and Mrs.' to 'Mr and Mrs'.

The code for tokenization lies in **tokenizer.py** file and can be run from command line or imported as module

Ngram Model

From the tokens, the Ngram model is build. It involves no smoothing and returns a dictionary object containing the ngrams as the keys and their frequency in the corpus as the value

The ngrams can be build using the **generate_ngrams** function in **language_model.py** code file. Throughout the code for all parts, the ngrams are generated by importing and calling the **generate_ngrams** function.

Language Model

There are 2 language models build:

- 3-gram Linear Interpolation Language model
- 3-gram Good Turing Smoothing Language model

Both the language models are trained on both the corpuses which gives total of 4 LMs. The language models code lies in **language_model.py** and can be invoked by specifying the type and corpus path in command line arguments as directed in the README.md.

The model average perplexity on train-test split of the corpus are as follows

Model	Train	Test
LM-1	12000	483
LM-2	10	323
LM-3	4353	417
LM-4	20	633

Note:

- LM-1, LM-3 are language model with Good Turing Smoothing
- LM-2, LM-4 are language model with Linear Interpolation

*The perplexity scores for each sentence and splits are given in the text files in **Results** folder*

The results show that the lm with good turing smoothing gives very high probability to unseen events and thus gives much lower perplexity on the test set. Lm with interpolation gives much higher probability to seen events and have very low perplexity on the train set.

Also in Interpolation the case of OOV words have been handled by using an <UNK> placeholder with count = count of words occuring once.

In good turing All the unseen events for a given context are assigned count of r*(0) together. This estimate works because the no. of unseen trigrams given a context were about the same and very close to the total vocabulary size. This replacement thus makes the assumption that the events of words given a context remain disjoint. Thus

$$P(Triagrams) = \sum_{context} \sum_{w_i} P(w_i|context)$$

$$\sum_{w_i} P(w_i|context) = \frac{\sum_{seenwords} count^*(w_i) + count*(0)_{UnseenWords}}{\sum_{seenwords} count^*(w_i) + count*(0)_{UnseenWords}} = 1$$

thus $P(Triagrams) = 1$ Hence the estimator is justified with the given assumption.

Generation

The generator code takes in the lm type and the corpus path, builds the language model and takes in an input sequence. The code then predicts the next words out of the corpus Vocabulary using the language models the given input sequence. Instructions for running the **generator.py** code lies in README.md. The code only involves generation using ngram model or lm with Linear interpolation. In cases of OOD where denominator becomes 0, the probability for that factor becoems 0.

In NGRAM generation, on experimenting with different n’s, triagram model was more reliable.

Eg.

- With n = 3, corpus = Pride_and_Prejudice_-_Jane_Austen.txt, k = 5
  best next-words were for ‘How are’

```
Input Sentence: How are
Output
('you',) 0.6666666666666666
('they',) 0.3333333333333333
('newsletter',) 0.0
('resented',) 0.0
('size',) 0.0
```

- With n = 3, corpus = Pride_and_Prejudice_-_Jane_Austen.txt, k = 5
  best next-words were for ‘Is Rhythm good’ (OOD sequence since rhythm isnt in Vocab)

```
('newsletter',) 0.0
('Devilled',) 0.0
('digests',) 0.0
('Mity',) 0.0
('Lubricate',) 0.0
```

- With lm = Interpolation, corpus = Pride_and_Prejudice_-_Jane_Austen.txt, k = 5
  best next-words were for ‘How are’

```
Input Sentence: How are
Output
('you',) 0.3586691860001095
('they',) 0.1762090678431166
('not',) 0.030444904651255555
(',',) 0.020465660018921777
('to',) 0.01575292281855608
```

- With lm = Interpolation, corpus = Ulysses_James_Joyce.txt, k = 5
  best next-words were for ‘How are’

```
Input Sentence: How are
Output
```

```
('you',) 0.20596447079805225
('things',) 0.14628048106915492
('the',) 0.12059701676785949
('all',) 0.10116641893591881
('.',) 0.0276273965712427
```

- With lm = Interpolation, corpus = Pride_and_Prejudice_-_Jane_Austen.txt, k = 5 best next-words were for ‘How are’ (OOD category)

```
Input Sentence: Is Rhythm good
(',',) 0.02515017338680057
('humour',) 0.016992885423569992
('.',) 0.016523947688912133
('opinion',) 0.011845540004650781
('deal',) 0.009182611778942078
```

- With lm = Interpolation, corpus = Ulysses_James_Joyce.txt, k = 5 best next-words were for ‘How are’ (OOD category)

```
Input Sentence: Is Rhythm good
('.',) 0.022879866407745784
(',',) 0.016959451126752633
('the',) 0.012600813720020018
('as',) 0.009999706846132439
('and',) 0.009876118443640605
```

Thus we can see that OOD categories are handled in Interpolation but not well in Ngrams model. Also since Ulysses is a bigger corpus, it gives better predictions on average.