The task was to implement three language models, a Unigram, Bigram and Bigram with smoothing to guess the missing term in a string of text from two given possible answers. All three have been implemented; however, the bigram with smoothing does not use the add 1 smoothing as suggested. This method of smoothing performs badly and gives to much mass to unseen bigrams and requires working through the full bigram model adding mass, which is computationally expensive. Instead a smoothing approach has been added that reduces the weight of very common bigrams and increases the weight of rare bigrams in a way that gives all potential answers a chance. Even if the bigram containing the possible answer has been not been encountered before in the corpus. In this assignment possible answers only need information on the previous and following bigram to determine whether they are plausible.

Code Implementation Explanation

Execution

The file is executed in Terminal using, *python3 lab2.py news-corpus-500k.txt questions.txt* where news-corpus-500k.txt is the corpus from which the models will be built from and questions.txt contains the questions to assess the models. It is assumed that the data folder is in the same directory as the executable python file.

Pre-processing the Corpus and Questions

To decrease the number of unique symbols in the corpus and remove symbols not necessary in this level of language modelling, the corpus is cleaned of all punctuation and converted to lowercase. New line markers, /*n* are replaced with <*s*> which indicates the start and end of sentences. The pre-processing is implemented in the *CommandLine* class by the *preprocessCorpus* method. Replacing new line markers with <*s*> allows for the corpus to be split into individual symbols with the structure of the corpus being conserved by the sentence end and start markers.

The questions also need to be pre-processed to make testing the models easier. The questions contain both the query and the potential answers all on one line, the line is split into the query and answers using the *processQuestions* method. The query symbol target of ____ is replaced with *'xxqqxx'*, to avoid accidental deletion of the query target when punctuation is removed. The separated query is then split into individual terms and the sentence start and end character <*s*> added to the beginning and end of the query.

Building the Unigram Model

The Unigram model is created by the *Unigram* class and *buildUnigram* method. Iterating through each symbol in the training corpus, if a symbol is previously unseen; this symbol is added as a new key in the unigram dictionary with its value set to an initial count of 1. If the symbol is already present as a key in the unigram dictionary, its value count is increased by 1. During each iteration the overall symbol count is increased by 1.

Once the whole corpus has been iterated through, the value of each symbol key is divided by the total symbol count to return the probability of an individual symbol. All these probabilities sum to 1.

$$P(Sentence) = \prod_{n=1}^{N} P(Symbol\ in\ Sentence\ ) = \prod_{n=1}^{N} \frac{number\ of\ times\ symbol\ occured\ in\ corpus}{total\ number\ of\ symbols\ in\ corpus}$$

The Unigram final structure:
- *{SymbolA: probability, SymbolB: probability, SymbolC: probability}*

Building the Bigram Model

The Bigram model is created by the *Bigram* class and *buildBigram* method. As with the unigram model the training corpus is iterated through, bar the final symbol as the final bigram covers the last two symbols in the corpus. The term of the iteration and the immediately adjacent term make up the bigram pair. The model is checked to see whether the first part of the pair exists as a key, if it does then the values are checked to see whether the second part of the pair exists. If it does, then the count for this bigram pair is updated by adding 1. If the first part of the pair is not a valid key in the model dictionary, then a new key is created with the second part of the pair added to the value as a nested dictionary. Once all symbols have been iterated through the *symbolCounts* dictionary is consulted. All the values for the nested dictionaries belonging to a bigram key are divided by the total count for that bigram symbol in the corpus to return the probability of each bigram pair.

The Bigram final structure is:
- *{PairFirstHalf: {PairSecondHalfA: probability}, {PairSecondHalfB: probability}, {PairSecondHalfC: probability}}*

The Smoothed Bigram Model

The Smoothed Bigram Model is not built as an individual model. This assignment gives the possible answers, so only bigrams containing those answers need to be smoothed. Furthermore, only bigrams relative to the query and the answers need to be smoothed. If the possible answers were not given, then the whole bigram model would require smoothing as all bigram pairs would be valid.

Smoothing is required if no bigram exists between the possible answer and the previous adjacent symbol and if no bigram exists between the possible answer and the next adjacent symbol in the query text. In these scenarios the probability of the unencountered bigram would be 0. Therefore, the possible answer would be ignored in a regular bigram model. The smoothing applied works to give any unencountered bigrams a probability, this probability is based on the occurrence of the possible answer in the corpus. The more common the possible answer the less weight the bigram it is included in has. This stops common terms overpowering bigrams, meaning that when possible answers are compared, a possible answer that has neither heavily over or underpowered bigrams is chosen.

Results and Evaluation

Computer specs: MacMini 2018, macOS Version 10.14.3, 6-Core 3.2GHz Intel Core i7-8, 32GB 2667MHz DDR4 RAM.

Query (Target to be replaced denoted by ____)	Unigram Answers	Bigram Answers	Smoothed Bigram Answers	Correct Answers
I don't know ____ to go out or not.	whether	whether	whether	whether
We went ____ the door to get inside.	through	through	through	through
They all had a ____ of the cake.	peace	piece	piece	piece
She had to go to ____ to prove she was innocent.	court	court	court	court
We were only ____ to visit at certain times.	allowed	allowed	allowed	allowed
She went back to ____ she had locked the door.	check	check	check	check
Can you ____ me?	here	hear	hear	hear
Do you usually eat ____ for breakfast?	serial	cereal	cereal	cereal
She normally ____ with her mouth closed.	choose	choose	chews	chews
I'm going to ____ it on the internet.	sell	sell	sell	sell
Percentage Correctly Answered	**60%**	**90%**	**100%**	

The results of the models do fit with what is expected. The unigram model performs the worst, this is because the unigram model stores no information about the structure of text. The structure of text, as described by markov, depends only the text structure seen previously. The lack of knowledge the unigram has about previous text structure is clearly shown in the 8th question. If the previous symbol of "eat" had been known then the answer of "serial" would have been correctly discounted, the structure of "eat serial" makes no sense. The unigram model works off which answer is most probable and therefore is highly skewed to the corpus used to train the model. This corpus contains more occurrences of the word "peace" than "piece" hence the incorrect answer for the 3rd question. A different corpus would have most likely returned different results for all questions. If a unigram model was required for a domain, such as computers, then the corpus used to build the model should be related to computers to reduce skew.

The bigram model returned a 30% increase in accuracy over the unigram model. The bigram stores information on text structure so mistakes made by the unigram such as in the 3rd question, are avoided. Using bigrams it is clear to see that "eat serial" is not a valid text segment and its probability will be lower than "eat cereal" which is a valid text segment, unigrams fail to distinguish this. Bigrams, as with unigrams, are skewed to the corpus. If a valid bigram is present in the query but unencountered in the corpus then it is given a probability of 0. This discounts potentially plausible answers because the corpus is finite. This is seen in the 9th question. The word "chews" is found in the corpus only 5 times whereas "choose" is found 58 times, "choose" therefore is part of far more bigrams than "chews". This increases the chance that the bigram "normally chews" and "chews with" is not found in our model and so this answer is given a probability of 0 and the incorrect answer of "choose" is returned.

The smoothed bigram model performs the best, correctly answering all questions. Both text structure and unencountered bigrams are accounted for resulting in model that gives all possible answers a fair chance. It is possible that if further questions were created then this model would begin to answer incorrectly, this would most likely happen in the case where both possible answers have very low occurrences in the corpus. To stop this and further improve these language models, more of the text structure could be retained by using higher value N-grams, this may however lead to very sparse models with very low probabilities for each gram. Moreover, the higher the N-gram value the larger the model, this would slow down computation considerably. Given the good results and fast computation of the bigram this would be suitable for many language modelling tasks.