Experiment - 3

Aim:

1. Calculate bigrams from a given corpus and calculate probability of a sentence.
2. To apply add-one smoothing on sparse bigram table

Theory:

A combination of words forms a sentence. However, such a formation is meaningful only when the words are arranged in some order.
Eg: Sit I car in the
Such a sentence is not grammatically acceptable. However some perfectly grammatical sentences can be nonsensical too!
Eg: Colorless green ideas sleep furiously

One easy way to handle such unacceptable sentences is by assigning probabilities to the strings of words i.e, how likely the sentence is.
Probability of a sentence can be calculated by the probability of the sequence of words occurring in it. We can use Markov assumption, that the probability of a word in a sentence depends on the probability of the word occurring just before it. Such a model is called the first order Markov model or the bigram model.

$$P(W_n \mid W_{n-1}) = P(W_{n-1}, W_n)/P(W_{n-1})$$

Here, Wn refers to the word token corresponding to the nth word in a sequence.

Probability of a sentence

If we consider each word occurring in its correct location as an independent event,the probability of the sentences is : P(w(1), w(2)..., w(n-1), w(n))

Using chain rule: = P(w(1)) * P(w(2) | w(1)) * P(w(3) | w(1)w(2)) ... P(w(n) | w(1)w(2) ... w(n-1))

Bigrams

We can avoid this very long calculation by approximating that the probability of a given word depends only on the probability of its previous words. This assumption is called Markov assumption and such a model is called Markov model- bigrams. Bigrams can be generalized to the n-gram which looks at (n-1) words in the past. A bigram is a first-order Markov model.

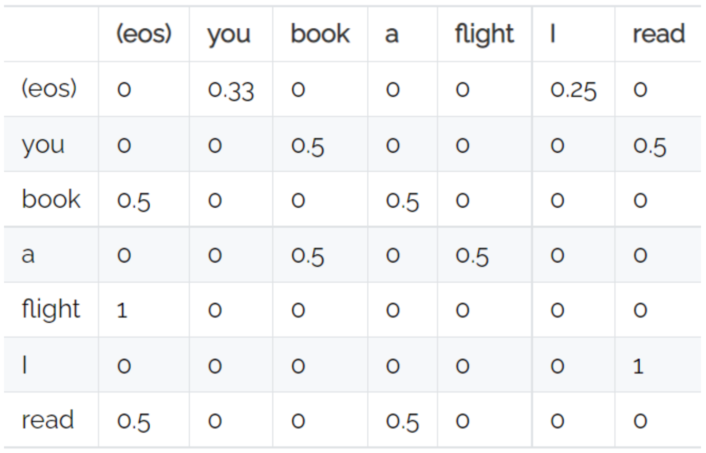
Therefore , P(w(1), w(2)..., w(n-1), w(n)) = P(w(2)|w(1)) P(w(3)|w(2)) ... P(w(n)|w(n-1))

We use the (eos) tag to mark the beginning and end of a sentence.

A bigram table for a given corpus can be generated and used as a lookup table for calculating probability of sentences.

Eg: Corpus - (eos) You book a flight (eos) I read a book (eos) You read (eos)

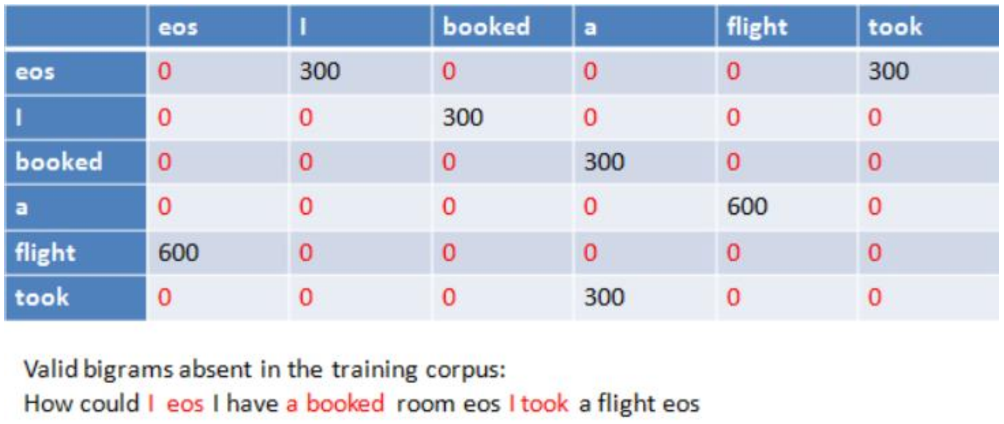
<u>Bigram Table</u>:



	(eos)	you	book	a	flight	I	read
(eos)	0	0.33	0	0	0	0.25	0
you	0	0	0.5	0	0	0	0.5
book	0.5	0	0	0.5	0	0	0
a	0	0	0.5	0	0.5	0	0
flight	1	0	0	0	0	0	0
I	0	0	0	0	0	0	1
read	0.5	0	0	0.5	0	0	0

P((eos) you read a book (eos))
= P(you|eos) * P(read|you) * P(a|read) * P(book|a) * P(eos|book)
= 0.33 * 0.5 * 0.5 * 0.5 * 0.5
= 0.020625

N-Grams smoothing:
One major problem with standard N-gram models is that they must be trained from some corpus, and because any particular training corpus is finite, some perfectly acceptable N-grams are bound to be missing from it.
We can see that the bigram matrix for any given training corpus is sparse. There are a large number of cases with zero probability bigrams and that should really have some non-zero probability. This method tends to underestimate the probability of strings that happen not to have occurred nearby in their training corpus.
There are some techniques that can be used for assigning a non-zero probability to these 'zero probability bigrams'. This task of reevaluating some of the zero-probability and low-probability N-grams, and assigning them non-zero values, is called smoothing.



	eos	I	booked	a	flight	took
eos	0	300	0	0	0	300
I	0	0	300	0	0	0
booked	0	0	0	300	0	0
a	0	0	0	0	600	0
flight	600	0	0	0	0	0
took	0	0	0	300	0	0

Valid bigrams absent in the training corpus:
How could I eos I have a booked room eos I took a flight eos

Add-One Smoothing:

In Add-One smoothing, we add one to all the bigram counts before normalizing them into probabilities. This is called add-one smoothing.

Application on unigrams:

The unsmoothed maximum likelihood estimate of the unigram probability can be computed by dividing the count of the word by the total number of word tokens N.

P(wx) = c(wx)/sumi{c(wi)} = c(wx)/N

Let there be an adjusted count c.

ci = (c i+1 * N/(N+V))

where V is the total number of word types in the language.
Now, probabilities can be calculated by normalizing counts by N.

pi* = (c i+1)/(N+V)

Application on bigrams:

Normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

P(wn|wn-1) = C(wn-1wn)/C(wn-1)

For add-one smoothed bigram counts we need to augment the unigram count by the number of total word types in the vocabulary V:

p*(wn|wn-1) = (C(wn-1wn)+1)/(C(wn-1)+V)

Input:

1. Text Corpus of sufficient length. For example movie reviews, newspaper articles, etc.
2. Sparse Bigram table

Output:

1. Calculated bigrams and probability of sentences
2. Add-one smoothed bigram table

Curiosity Questions

1. **What is proble is solved by smoothing and how does it solve the problem?**
2. **What is perplexity?**
3. **Can N-Gram understand syntax?**
4. **What are the limitations of N-Gram model? What are the alternatives?**

References:

- https://www.geeksforgeeks.org/n-gram-language-modelling-with-nltk/
- https://medium.com/swlh/language-modelling-with-nltk-20eac7e70853
- https://web.stanford.edu/~jurafsky/slp3/3.pdf

- https://www.educative.io/edpresso/what-is-a-bigram-language-model
- https://www.youtube.com/watch?v=dZvCHz6lcGU