Natuurlijke Taalmodellen & Interfaces
Tejaswini Deoskar
Part A (Step 1)

Lucas van Berkel, 10747958
Ruben Gerritse, 10760326
Gideon Mooijen, 10686290

Report

In a text or speech, we can differentiate individual sequences as a *'n-gram'*, where *'n'* corresponds with the amount of words that the sequence exists of. N-grams are used in a variety of fields, but in the field of linguistics one of the most beneficial aspects of n-gram analysis is the fact that we can distillate probabilities regarding predicting the next item.

The importance of *n-gram analysis* can best be described by mentioning the wide range of applications it is used for. Among others:

1. Spellchecker: predicting which word the user tried to spell (when a misspelled word occurs)
2. Speech recognition: when difficulties with computational comprehension occurs, predictions about the most probable outcomes can help identifying the - *otherwise* - unrecognizable word.
3. Machine translation: when a concept can not be translated precisely, frequent n-grams can help a computer to translated it in a way that it matches the source in a natural way.
4. *Lots of other applications.*

A common theme in all these applications is the *predicting feature* of this concept. In most applications, n-gram analysis presents a few candidates, words that are most likely to be the goal word. The second step is to compare the candidates with the interpreted word. This can be done by evaluating the distance between the interpreted word and the generated candidates. There are multiple ways to do so. For a spellchecker, one can count the distance between the candidate and the interpreted word, expressed in errors. For speech recognition, one could use *dynamic time warping* to align to sounds and calculate the difference. Ultimately, a trade-off between the most probable candidate and the candidate that was closest represented by the input word concludes which candidate is the best one.

To obtain the most frequent sentences, we wrote 96 rules of code. The code is enclosed, but we will briefly comment on the few steps of the program (without technical details).

1. Convert the corpus to one large sentence (wordlist)
2. Construct a dictionary, with all unique sequences as keys, and their occurrences as values
3. Print the top occuring sentences
4. Print the sum of all frequencies of all sequences for the given n.

For a more elaborate explanation: our code is well annotated.

Answers

1. The 10 most frequent sequences.

	n = 1	**n = 2**	**n = 3**
1	the	of the	I do not
2	to	to be	I am sure
3	and	in the	in the world
4	of	I am	She could not
5	a	of her	would have been
6	her	to the	I dare say
7	I	it was	as soon as
8	was	had been	a great deal
9	in	she had	it would be
10	it	to her	could not be

2. The sum of all frequencies of all sequences for that n.

	n = 1	n = 2	n = 3
1	617091	617091	617091

3. To use the program, run the following command in your terminal.

```
python BerkelGerritseMooijen_1.py [PATH TO CORPUS] [VALUE N] [VALUE M]
```