Linguistique de corpus en anglaise
Semestre 1 – 2021/2022

Project Semester 1
Estimating a Language Model to Generate Wine Reviews

1. *Estimating the Language Model*

The objective of this exercise is to build a program capable of automatically writing wine reviews, like the one found on the Wine Spectator website, using a language model.

A language model of order n allows us to determine $p(w_i|w_{i-1}, w_{i-2}, \dots, w_{i-n+1})$, the probability of finding a word w_i after observing the previous $n-1$ words, as well as allowing us to determine the probability of a sentence.

This information is at the heart of speech recognition or machine translation systems and it can also be used to generate sentences similar to those produced by a human.

The probabilities (parameters) of the language model will be estimated from the reviews that can be found on the Wine Spectator website. These reviews can be found in the project folder on github (https://github.com/armandstrickernlp/CL_class_inalco).

The file text_reviews.txt is provided for you to test your program on a smaller, more manageable sample of data. It is recommended that you try to implement the program by hand using simple examples before using the full dataset of wine reviews.

a. Let w_i be the word at the ith position of a sentence. How is the probability of observing the word w_i knowing the two previous words w_{i-1} and w_{i-2} estimated ?

b. Write the function *make_trigrams* which returns the list of successive triplets from a string of words.
For example, when the string “I love chocolate ice-cream” is passed to this function, it should return the list :
[("I", "love", "chocolate"), ("love", "chocolate", "ice-cream"), ("chocolate", "ice-cream", ".")].

c. Write a function *make_conditional_probas* that estimates all of the probabilities p(c|a,b) from a file passed as argument. One way of doing this is by first constructing a count table which looks like the following :

```
{('BEGIN', 'NOW'): defaultdict(<class 'int'>, {'I': 2}),
 ('I', 'do'): defaultdict(<class 'int'>, {'not': 1}),
 ('I', 'like'): defaultdict(<class 'int'>, {'chocolate': 1}),
 ('NOW', 'I'): defaultdict(<class 'int'>, {'like': 1, 'do': 1}),
 ('chocolate', 'ice-cream'): defaultdict(<class 'int'>, {'.': 1}),
 ('chocolate', 'pudding'): defaultdict(<class 'int'>, {'.': 1}),
 ('do', 'not'): defaultdict(<class 'int'>, {'like': 1}),
 ('ice-cream', '.'): defaultdict(<class 'int'>, {'END': 1}),
 ('like', 'chocolate'): defaultdict(<class 'int'>,
                                    {'ice-cream': 1,
                                     'pudding': 1}),
```

The keys represent (a,b), the bigrams contained within each trigram. Their values are dictionaries which have as keys c (the following word) and as values the amount of times c followed a,b (which comes down to the trigram count).

Once the different counts have been established, it is possible to sum over all of the counts to obtain a total and to then divide each individual count by this value.

$$P(c|a,b) = \frac{count(a,b,c)}{total = \sum_{i=1}^{n}(a,b,c_i) = count(a,b)}$$

The output probability table should look something like this:

```
{('BEGIN', 'NOW'): {'I': 1.0},
 ('I', 'do'): {'not': 1.0},
 ('I', 'like'): {'chocolate': 1.0},
 ('NOW', 'I'): {'do': 0.5, 'like': 0.5},
 ('chocolate', 'ice-cream'): {'.': 1.0},
 ('chocolate', 'pudding'): {'.': 1.0},
 ('do', 'not'): {'like': 1.0},
 ('ice-cream', '.'): {'END': 1.0},
 ('like', 'chocolate'): {'ice-cream': 0.5, 'pudding': 0.5},
 ('not', 'like'): {'chocolate': 1.0},
 ('pudding', '.'): {'END': 1.0}})
```

d. Two words (BEGIN NOW) have been added to the beginning of each review. Why ? Why are **2** words added and not 1 or 5 for example ?

2. Generation

Once the language model has been estimated, new sentences can be generated as follows: the i[th] word of the sentence is chosen randomly according to the probability $p(w_i|h)$ where h is a history composed of the two words w_{i-2}, w_{i-1} chosen during steps $i-1$ *and* $i-2$. The history is then updated $h \longleftarrow w_{i-1}, w_i$ and the procedure is repeated until the word END is generated.

If a distribution X generating events a_i with probability p_i is represented by a dictionary whose keys are the a_i and values the probabilities p_i, it is possible to generate an element a_i with probability p_i using the following function:

```
import numpy as np

def sample_from_discrete_distrib(distrib):
    words, probas = zip(*distrib.items())
    probas = np.asarray(probas).astype('float64')/np.sum(probas)
    return np.random.choice(words, p=probas)
```

As an example, using the dictionary {'ice-cream': 0.5, 'pudding': 0.5}, this function will generate one of the words following with respect to their given probabilities.

1. How should you initialize the history ?

2. Implement this algorithm by creating a function called *generate* which has as parameter the probability table. What do you think of the sentences obtained ?

3. How could we estimate the quality of the sentences produced ?