**Feedback for WS2**

We received 18 submissions for the second worksheet. Here is some feedback upon the review of your submissions:

**[E1] :**

As most of you already figured out, we need to calculate the probability of wnwn given the prefix sequence wn−1,wn−2wn−1,wn−2 which is represented by the conditional probability P(wn|wn−1,wn−2)P(wn|wn−1,wn−2).

P(wn|wn−1,wn−2)=C(wn−2,wn−1,wn)/C(wn−2,wn−1)P(wn|wn−1,wn−2)=C(wn−2,wn−1,wn)/C(wn−2,wn−1) (CC denotes the count of trigram and prefix bigram sequences)

**[E2] :**

The motivation behind this question is to let you think about the necessary steps to compute unigram and bigram probabilities for a given corpus. The library *nltk* already provides the necessary functionality for this, but you can implement those functions by yourself to get a better understanding of the concepts.

You need to find the conditional probabilities for each possible bigram and probabilities for each word in the corpus for unigram model.

**[E3]:**

You need to define two separate corpora to work on for this exercise. You can use the built-in corpora in nltk (brown, gutenberg etc.), load the corpora from external files or create the corpora inside your code. The most common words (i.e. most common unigrams) in English corpora are usually stop-words such as *the, a, and* etc. So, if you only look at the most common words in a corpus, you may not learn a lot about that corpus' content but can learn about the language you are working on. But after the most common words (starting from the 11th common word may be?), you would see interesting words such as names, terminology etc. relevant to the domain the text belongs to. It is actually worth to think about what the common bigrams can say us about the language that the corpus data is generated in and the domain that the data belongs to.

**[E4]:**

An easy way to implement what is needed here is to add a new function to your code which generates random sentences from the existing n-grams. There is already an informative discussion on Piazza regarding this question (question@16 in hw2), so please refer it in case you have some doubts. As a hint, it is a good start to think about how you can mark the sentence boundaries (start and the end of the sentences) in your corpus so that you can start with reasonable tokens to your sentence and stop adding new tokens if you already reached to a (possible) end of a sentence.

**[E5]:**

Since your code is unsmoothed, a sentence with a word which was not existing in the training set (remember why we use smoothing) will result in a probability of 0 (i.e., will be judged as impossible by the model), even if that sentence seems to fit to the domain well.

**[E6]:**

Replacing the least frequent words with the placeholder UNK and adding 1 to all counts (Laplace smoothing) are two common ways of smoothing. You can apply these methods separately and see their impact on the probabilities.

You will observe that unknown words no longer lead to zeros in both cases. But in Laplace case, not only the infrequent words but probabilities of other words are also affected. It would be interesting to check how are the probabilities changed after Laplace smoothing.

You can observe that the probabilities P("the" | "end") and P("end" | "the") are affected differently by smoothing. The possible change would depend on the probabilities of bigrams ("the end" and "end the") and the unigram probabilities of "the" and "end" in your corpus, and how they are affected by the smoothing.