Statistical Machine Translation

Homework 2

To be sent (pdf, Zip) to `waszczuk@phil.hhu.de` by 20.11.2018.

Exercise 1 - Practice

The probability of a word sequence $\boldsymbol{w}$ of length n can be decomposed using the chain rule:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \ldots \cdot P(w_n \mid w_1, \ldots, w_{n-1})$$

where w_i corresponds to the ith word in $\boldsymbol{w}$.
A **Bigram Language Model** is a Markov chain of 1st order. So we assume that:

$$P(w_i \mid w_1, \ldots, w_{i-1}) = P(w_i \mid w_{i-1})$$

When training a language model, these probabilities are determined (estimated) according to data. A familiar and intuitive way to do this is *Maximum Likelihood Estimation*:

$$\hat{P}_{MLE}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{\sum_{w \in V} C(w_{i-1}w)} = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

(The P has a hat on to show that it is our estimation). C is a function which returns the frequency of a sequence of words in the training data (*count*). V is the vocabulary.

Two points make the estimations a bit more complicated:
(1) **Unknown words:** We cannot make any statement about the words that appear in the test data, but did not occur in the training data. One solution is to replace words which rarely occur in the training corpus (once for example) by a specific word (eg `<UNK>`) and not to add them to V. Words $w \notin V$ will also be replaced by `<UNK>` in the test data.
(2) **unseen bigrams**: When a bigram $w_{i-1}w_i$ does not occur in the training data, we have $C(w_{i-1}w_i) = 0$ and therefore $\hat{P}(w_i \mid w_{i-1}) = 0$. Consequently, the whole sequence of words in which the bigram occurs during testing has probability 0. To avoid this, there are various *Smoothing methods* that adjust the bigram frequencies that are observed in

the training corpus. The simplest (not very good) is *Add-One-Smoothing*: Each bigram frequency is increased by one.

$$\hat{P}_{Add1}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + |V|}$$

Your job is to implement such a bigram language model that can cope with both unknown words and unseen bigrams. The probabilities should be determined based on the given training data `data/train.de.tok.50.h1000`. At the end, the output should be the language model probability for a test set.

As for the previous session, the framework is already in place and is designed to implement certain methods. Download the code for the exercise and save it locally. The code is made available as an Eclipse project. Start Eclipse and import the archive file as an existing project (File → Import → Existing Projects into Workspace → Select archive file). At the end of the session you should export your project again as a zip file and keep a copy of it (email, USB).

`de.hhu.phil.smt.lm.Uebung2` is the `Main`-class. It takes no argument. The code should be compilable and executable. The output shows the number of lines read. Start by getting familiar with the code and the data. You mainly need to add code in `BigramModel.java`. This time the most important data structures are already provided.

General note: You do not have to stick strictly to the structures suggested in the code. In any case, make sure that anyone can understand your code, for example, by commenting it. If the original call of the program no longer works, please add a readme file that specifies how to compile and execute the code. You can of course use classes from the Java API (`http://docs.oracle.com/javase/8/docs/api/`), for example, sets, maps (associative arrays), etc.

Exercise 2 - Theory

We consider the Markov model specified as follows (we use here the introduced short forms):

1. $P(a|\ltimes) = 0,4,$
2. $P(b|\ltimes) = 0,6,$
3. $P(\rtimes|\ltimes) = 0;$
4. $P(a|a) = 0,8,$
5. $P(b|a) = 0,1,$
6. $P(\rtimes|a) = 0,1;$
7. $P(a|b) = 0,2,$

8. $P(b|b) = 0,7$,
9. $P(\rtimes|b) = 0,1$.

In general, we have:

1. $P(\ltimes|x) = 0 : x \in \{a, b, \rtimes\}$,
2. $P(\ltimes) = 1$ ($\ltimes$ is the beginning),
3. $P(x|\rtimes) = 0 : x \in \{a, b\}$ ($\rtimes$ is the end).

Calculate the probability of the following events in the associated probabilistic language:

1. The third letter in a word is an a.

2. A word has length ≥ 3.

Tip: Of course you can calculate this with the sum rule; P(Event 1) is the sum of all probabilities of all words with a as the third letter (similar for 2). But there are easier ways!