SMT and Parallel Corpora: Exercise 2

Implementing IBM Model 1

March 8, 2016

Submission: March 22, 16:00

This is a **mandatory** exercise and the result will be part of your final mark. The solution must be uploaded to the OLAT folder "Exercise 2" prior to **March 22, 16:00 - March 22, 15:59 at the very latest**. Late submissions will not be accepted. The exercise can be done in group of **two people**.

Submit the following files in a zipped archive:

- Your implementation of the IBM Model 1 EM algorithm (*.py or the appropriate file extension, depending on what language you wrote your code in)
- Your report, summarising your insights (*.pdf)

Make sure the archive is named [firstname]_[lastname].zip (for example *mathias_mueller.zip*).

If you have any question regarding this exercise, do not hesitate to either post in the OLAT forum (if you think everybody would benefit from an answer) or contact us by email.

1 Modelling Word Alignment

Parallel corpora are usually available in "Moses format", that is, one sentence per line. So, we already know which pairs of sentences are translations of each other. If our system were to only translate sentences that we have already seen in the training data, that would already be the end of the alignment story. But, because of the recursive nature of human languages, it is incredibly likely that the input to our SMT system will include sentences we have never seen before. In order to translate those unknown sentences, we have to break them down into small pieces – for which the probability that we have seen them already is much higher.

But, translating pieces smaller than a sentence introduces another challenge: We need to find out which pieces are translations of each other. This process is called *Word Alignment*. In this exercise, we are going to implement the simplest of all lexical translation models, IBM Model 1 – that relies on alignments between words.

1.1 Learn about IBM Model 1

As you know by now, alignments between words are not known beforehand and are something that must be learned from the data. It is difficult to learn them because we do not yet have a lexical translation model like IBM Model 1, which in turn is built on top of word alignments – a very typical chicken and egg problem. That is why we are learning the model iteratively, checking our intermediate model against the actual data and refining it in each iteration. Fortunately, we do not have to make such an algorithm up out of thin air: we are going to implement an existing one, the Expectation Maximization (EM) Algorithm for IBM Model 1.

Before you implement this algorithm, make sure you are familiar with IBM Model 1. Have another look at the lecture slides, read Chapter 4 ("Word-Based Models") in the book by Philipp Koehn[1] or watch an entertaining video on Youtube: `https://www.youtube.com/watch?v=n58-akQIlQ4`.

1.2 Implement IBM Model 1

Implement IBM Model 1, following the pseudo code described on page 91 of Koehn (2010):

```
intitialize t(e|f) uniformly
while not converged do
   // initialize
   count(e|f} = 0 for all e,f
   total(f) = 0 for all f
   for all sentence pairs (e,f) do
      // compute normalization
      for all words e in e do
         s-total(e) = 0
         for all words f in f do
            s-total(e) += t(e|f)
         end for
      end for
      // collect counts
      for all words e in e do
         for all words f in f do
            count (e|f) += t(e|f) / s-total(e)
            total(f) += t(e|f) / s-total(e)
         end for
      end for
   end for
   // estimate probabilities
   for all foreign words f do
      for all English words e do
         t(e|f) = count(e|f) / total(f)
      end for
   end for
end while
```

Please note:

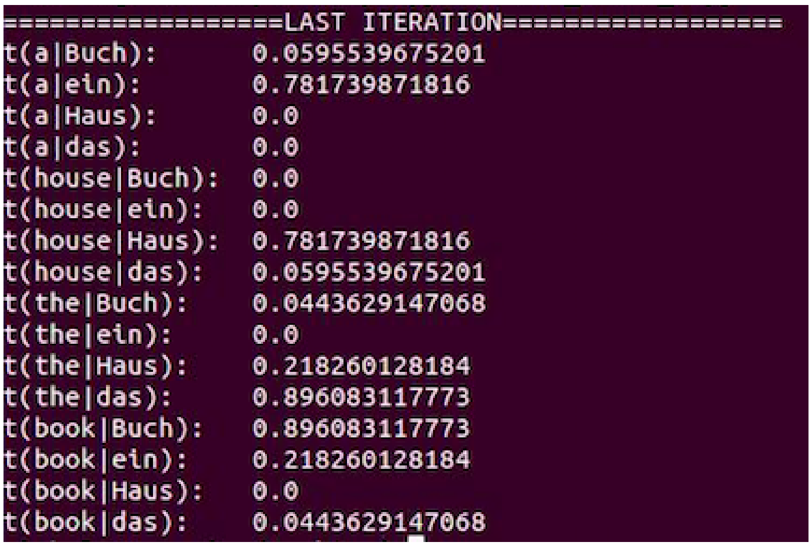
- Python is a language that is well-suited for this task and we suggest you use Python to implement IBM Model 1. Still, you are free to use any other programming language, but make sure a) you clearly state the language of your code and b) the output is correct.
- Do not start from scratch, use `ibm-model-1.py` as a blueprint. You will find this script in the zip folder of this exercise. All relevant data structures, examples for input sentence pairs and the initialisation of all translation probabilities (the first line of pseudo code above) are already included.
- There is no need to reach a state of convergence (usually measured by the *perplexity* of the model), simply run your program for **5 iterations**.

If implemented properly, using the following sentences as input:

[1]Phillip Koehn (2010): Statistical Machine Translation. Cambridge University Press.

```
f_sentences = ["das Haus", "das Buch", "ein Buch"]
e_sentences = ["the house", "the book", "a book"]
```

and after running 5 iterations, your code should yield something similar to the following result. You **have to** format you output like the one in the image: you final code should only print the last iteration. Eventually, the algorithm would converge towards values of 0 or 1 in this case.



```
==================LAST ITERATION==================
t(a|Buch):      0.0595539675201
t(a|ein):       0.781739871816
t(a|Haus):      0.0
t(a|das):       0.0
t(house|Buch):  0.0
t(house|ein):   0.0
t(house|Haus):  0.781739871816
t(house|das):   0.0595539675201
t(the|Buch):    0.0443629147068
t(the|ein):     0.0
t(the|Haus):    0.218260128184
t(the|das):     0.896083117773
t(book|Buch):   0.896083117773
t(book|ein):    0.218260128184
t(book|Haus):   0.0
t(book|das):    0.0443629147068
```

For instance, after 50 iterations, the probabilities look like

```
------------ iteration  50 ------------
Translation probability: a Buch 2.04459611037e-14
Translation probability: house Buch 0.0
Translation probability: the Buch 3.19375466346e-15
Translation probability: book Buch 1.0
Translation probability: a ein 0.989095122384
Translation probability: house ein 0.0
Translation probability: the ein 0.0
Translation probability: book ein 0.0109048776165
Translation probability: a Haus 0.0
Translation probability: house Haus 0.989095122384
Translation probability: the Haus 0.0109048776165
Translation probability: book Haus 0.0
Translation probability: a das 0.0
Translation probability: house das 2.04459611037e-14
Translation probability: the das 1.0
Translation probability: book das 3.19375466346e-15
```

Save your script as `ibm-model-1.py` **(or another file extension, depending on the programming language). Write a report describing your insights. For example,**

- **Investigate whether word order or lemmatisation influence IBM Model 1.**
- **Monitor two extreme word pairs (e, f) in different iterations: A pair (e, f) where you expect the translation probability to decrease and another pair (e, f) where you expect the probability will increase. Explain their behaviour.**