METL – TP3

Alexandre Kabbach
alexandre.kabbach@unige.ch

16.03.2017

Deadline: 06.04.2017

Implementing IBM model 1 (50 points)

Introduction

Aligning words of two parallel sentences is a core task in statistical machine translation. A variety of techniques have been proposed, but the most common approach by far is to train a probabilistic alignment model and use its predictions to align words. You will implement the simplest model that works well, IBM Model 1.

Word alignment is applied to pairs of sentences. Each pair consists of a sentence $\mathbf{f}$ in a foreign source language and its translation $\mathbf{e}$ to the target language (typically English). IBM Model 1 is a generative probabilistic word alignment model that generates each word e_i of an English sentence independently from other words of the sentence and conditioned on some word f_{a_i} in the corresponding foreign sentence. The likelihood of a foreign sentence therefore factors across words:

$$P(\mathbf{e}, a \mid \mathbf{f}) \sim \prod_i P(e_i \mid f_{a_i}).$$

The alignment a establishes for each sentence pair the correspondence between the source and the target words. Knowing the correct alignment, we can find the translation probabilities $P(e \mid f)$ for all pairs of e and f. The expectation-maximization algorithm iteratively estimates both translation probabilities and maximum probable alignment for IBM Model 1 given a non-annotated set of parallel source and target sentences.

The iterative EM update for this model is straightforward. For every pair of a target word type e and a source word type f, you count the expected (fractional) number of times that tokens of f are aligned with tokens of e. At the end, one normalizes the counts for each e, which will give a new estimate of the translation probabilities $P(e \mid f)$, which leads to new alignment posterior probabilities.

Data

We will make English-Spanish parallel data available on Chamilo in order to train and test your model. The parallel data is given in pairs of files with the same name and appropriate language suffixes *.en* or *.es*. Such files are sentence aligned, such that the i-th sentence of the *.en*-file is a

IBM Model 1 and EM: Pseudocode

Input: set of sentence pairs $(\mathbf{e}, \mathbf{f})$
Output: translation prob. $t(e|f)$

initialize $t(e|f)$ 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$(e|f)$ += $\frac{t(e|f)}{\text{s-total}(e)}$
        total(f) += $\frac{t(e|f)}{\text{s-total}(e)}$
      end for
    end for
  end for
  // estimate probabilities
  for all foreign words f **do**
    for all English words e **do**
      $t(e|f) = \frac{\text{count}(e|f)}{\text{total}(f)}$
    end for
  end for
end while

translation of the i-th sentence of the *.es*-file. The data we provide is already preprocessed (i.e., lowercased and tokenized).

The test data in the folder *test/* contains 100 pairs of parallel sentences. The gold alignments for these sentences are available in a file with an extension *.align*. For each sentence pair of the test corpus it provides a list of the word alignment pairs, as (i, j), the pairs of indices of word positions in each sentence (notice that the word indexing starts at zero, i.e, 0 is not a symbol for NULL word).

The training data in the folder *training/* contains 50'000 pairs of parallel sentences. To speed up the development of your program, you can use a portion of the data, for example 1000 or 10'000 pairs, but you should be aware that using less data will affect the final performance of your system.

Details and requirements

More precisely, we ask you to implement a Python program which should include the following functions:

train() This function should be an implementation of the (soft) EM algorithm of IBM Model 1, described in class. A comprehensive pseudocode is presented in Philipp Koehn's SMT book (Figure 4.3, page 91, reproduced above), optimised for efficiency, unlike what we saw in class. The output of the training process is the set of translation probabilities $P(e \mid f)$.

align() The function outputs the most probable alignment of two sentences given the translation probabilities. As the translation decision for each word is independent of anything else, the calculation is straightforward:

$$a^* = \max_a P(\mathbf{e}, a \mid \mathbf{f}) = \prod_i \max_{a_i} P(e_i \mid f_{a_i}).$$

score() This function should evaluate the alignments produced by your model against the gold alignments given in a file. It must output the precision, recall and F1 measures using the counts of the correct and wrong guesses summing over all sentence pairs.

write_translations() This function should write a file containing all translations with their probabilities $P(e \mid f)$:

$e_1 \quad f_1 \quad P(e_1 \mid f_1)$
$e_2 \quad f_2 \quad P(e_2 \mid f_2)$
...

Sort it so that the most probable translation pairs appear at the top of the list.

write_alignments() This function should write the alignment links to a file (one sentence per line), according to the format given in the *test.align* file.

Additionally, it must be possible to run the Python script that you create with the following options (e.g. by using the *argparse* module):

-n specifies the number of sentences for training

-i specifies the number of EM iterations

-t specifies the name of the training data file

-s specifies the name of the test data file

Note 1: To test the correctness of your training algorithm you can use the example from the SMT book which follows the IBM Model 1 discussion (pp. 91–92, Figure 4.4). The table in Figure 4.4 provides the translation probabilities of 3 iterations of the EM algorithm. The probabilities that your EM implementation produces should be the same at each iteration.

Note 2: While developing and testing your program you can use a small training dataset (e.g. 1000 sentences) and only a few (e.g. 5) iterations of EM.

Thus, in the final version of your program, the main procedure should do the following:

1. Estimate translation probabilities on the training data using train().
2. Write these translation probabilities to a file using write_translations().
3. Produce word alignments for the test data using align().
4. Write these alignements to a file using write_alignments().
5. Evaluate the correctness of these alignments using score().

Submit your script as well as the files containing the translation table[1], and the alignments.

Report

Please also submit a short report (2-3 pages) that includes a brief description of the work you did and of the settings that produced the results you present. Report the performance of your model on the test data and list the first 20 entries of the translation table as well as the running

[1] In case the entire translation table is too large to be submitted, submit a truncated file containing the top 5000 entries.

time of your program for all experiments. Indicate how your script handles out-of-vocabulary words in the test data.

Additionally, answer the following questions in the report:

1. Run EM with one iteration. Report the performance of this model. Is it good? Does it improve when using **more data**?
2. Run EM with **more iterations**. Does the performance improve with respect to the previous model? Can you explain why?
3. Run your script with the **opposite direction** of translation. With this script you can train a model that generates the foreign sentence given the English one, but also vice versa. Does the performance remain the same? Take as an example a pair of parallel sentences and explain why two models (en-es and es-en) produce different alignments.