MOOC on Approaches to Machine Translation: Rule-based, Statistical and Hybrid

MT-MOOC

http://mooc.upc.edu

Universitat Politècnica de Catalunya

Assignment 1 (Week 2): Let's align the bible

Goal

The main goal of this assignment is to put in practice some of the concepts of word alignment. To that effect we are going to align the world's most translated book: *The Bible.*

Requirements

The exercise is intended to be done in any operating system: Windows / Mac OS X / Linux but **a python interpreter is needed** to deal with it. Please, if you are not familiar with playing with Python refer to the guide provided within the course at `https://mooc.upc.edu/courses/2/modules/items/244` (Python) and `https://mooc.upc.edu/courses/2/modules/items/243` (Jutge Software evaluation platform).

The resources needed for doing the exercise in might be downloaded to the following URL at `http://mooc.upc.edu/courses/2/assignments/33`

If you have any problem, please post a discussion at `https://mooc.upc.edu/courses/2/discussion_topics`.

We also recommend the of an advanced text editor for doing your programming such as:

Sublime Text Editor : `http://www.sublimetext.com/`
Notepad++ : `http://notepad-plus-plus.org/`

Estimated time

2 hours. Maximum 6 hours

Deadline

November 18th at 23:59:59 CET

Description

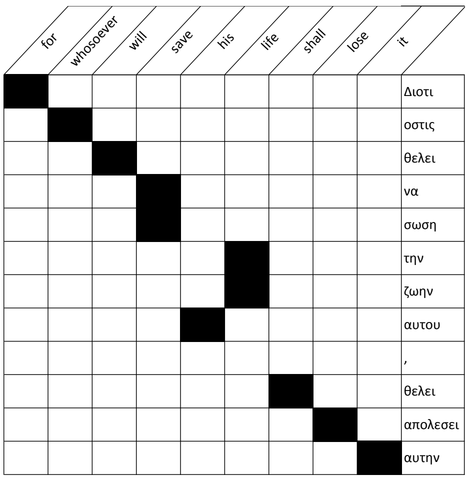
Aligning words is a key task in Statistical Machine Translation (SMT). We start with a large parallel corpus where the translation of each sentence is known. So the input consists of sentence pairs like this one:

Greek: *Διοτι οστις θελει να σωση την ζωην αυτου , θελει απολεσει αυτην .*
English: *for whosoever will save his life shall lose it .*

We are going to immerse ourselves to the world of Bible word alignment, however as we are neither Theologists nor Bible experts we are going to make use of our excellent knowledge about Word Alignment.

Thankfully, as Bible is organized in verses, therefore it is easy to obtain parallel text aligned at sentence level. Other domains are more difficult like web-portal translation or TV subtitles. On those cases more complex techniques might be required. However, the main problem is the lack of word-to-word correspondences within the sentence.

That is the goal of the assignment: Help us to finish the code and perform word alignment. For the sentence above, you might output the following correspondences:



Some words might be unaligned (e.g. Greek comma), while some words might be aligned to multiple words in the corresponding sentence (e.g. *save* aligned to *να σωση*). Sometimes words are not exact translations of each other. That is ok: even experienced bilinguals will sometimes disagree on the best alignment of a sentence. But while word alignment does not capture every nuance, it is very useful for learning translation models or bilingual dictionaries.

Material

Among the files contained within the compressed zip file, you have found:

Assignment.pdf : The current assignment description
align.py : The main code of our word alignment program.
check.py : Code that checks that output of *align.py* is alignment f-e format.
grade.py : Program that provides grades on how well our program is working.
bible.f : Greek version of the bible provided as one verse per line
bible.e : English version of the bible provided as one verse per line
bible.a : True alignment between Greek and English versions of the bible in format f-e.

The Challenge

In our task, we will complete the program align.py to implement Model 1 / Model 2 and some experimentation with the symmetrization of the heuristics. It assumes that you use Numpy and collections package. Therefore, the use of other python packages is not allowed. The only file you will have to upload to the evaluator platform will be the align.py program. The program has some parts marked as:

```
– *** YOUR CODE HERE ***
```

where you will have to make the programming.

Your task for this assignment is to improve the alignment error rate as much as possible. It should not be hard to improve it at least some: you have probably noticed that thresholding a Dice coefficient is a bad idea because alignments do not compete against one another. A good way to correct this is with a probabilistic model like IBM Model 1. It forces all of the English words in a sentence to compete as the explanation for each foreign word.

Formally, IBM Model 1 is a probabilistic model that generates each word of the foreign sentence f independently, conditioned on some word in the English sentence e. The likelihood of a particular alignment a of the foreign sentence therefore factors across words: $P(f,a|e) = \prod_i P(a_i = j|len(e)) \times P(f_i|e_{a_i})$. In Model 1, we fix $P(a_i = j|len(e))$ to be uniform (i.e. equal to $\frac{1}{len(e)}$), so the likelihood only depends upon the conditional word translation parameters $P(f|e)$.

The iterative EM update for this model is straightforward. For every pair of an English word type e and a Greek word type f, you count up the expected (fractional) number of times tokens f are aligned to tokens of e and normalize over values of e. That will give you a new estimate of the translation probabilities $P(f|e)$, which leads to new alignment posteriors, and so on. We recommend developing on a small data set (1000 sentences) and only a few iterations of EM. When you are finished, you should see some improvements in your alignments.

Developing a Model 1 aligner should be enough to beat our baseline system. However, as you can do better than only Model 1, we give you some ideas to experiment:

- Train a Greek-English model and an English-Greek model, then combine their predictions in some way. Function grow-diag-and is provided to you.[1].
- Try to finish Model 2 implementation given the pseudo-code.

Computational Limit: Even that sky is the limit because alignment is not a solved problem, due to Canvas restrictions you have to stick yourself in doing the alignment reasonably fast. **That is 150 sentences have to be aligned within a minute**. Therefore, we cannot process implementations involving dynamic programming (HMM, Models 3, 4 and 5) and you should be bound to Models 1, 2 and symmetrization.

Procedure

We have provided you the skeleton of the aligner written in Python and around one million words of the Bible. This skeleton is implemented in file called align.py. The baseline algorithm works in two phases. First, we train models. The aligner observes the training sentences and then gives a score, between 0 and 1, to each possible alignment. The baseline aligner simply computes Dice's coefficient between pairs of English and foreign words. It then aligns all pairs of words with a score over 0.5. Run the baseline heuristic model on 150 sentences using the command:

`./align.py -n 150 > dice.a` This runs the aligner and stores the output in dice.a. To view and score the alignments, run this command:

`./grade.py < dice.a` This command scores the alignment quality by comparing the output alignments against a set of human alignment annotations using a metric called the alignment error rate, which balances precision and recall of the guessed alignments (paper). Look at the terrible output of this heuristic model – it is better than chance, but not any good. Try training on 1000 sentences instead of 150, by specifying the change on the command line:

`./align.py -n 1000 | ./grade` Performance should improve, but only slightly! Another experiment that you can do is change the threshold criterion used by the aligner. How does this affect alignment error rate?

[1] **Alert:** Evaluations will be run in short sets of sentences (around 150). On those cases **grow-diag-final** does not have to be the best solution. Maybe grow-diag or grow-diag-final-and are

Steps and Hints to ease your task The skeleton has three different kind of tasks that you will have to complete:

Uncomment parts of the code : e.g. lines 257-259
Specify mathematical formulation to some variable : e.g. lines 112 or 119
Specify proper calls to specific already functions : e.g. line 269

The recommended the steps for you to follow are the following:

Step 1 Finish Model 1 implementation:

1. Comment out the call to the dice alignment parts of the code (lines 252 and 253) and uncomment the calls to model12 computation (line 257) and alignment(line 258).
2. Put the formula to obtain the normalization factor $total_s$ (partition function — line 119) and the probability given: $value = f(e_i, f_j)$ value (line 125).
3. Help the alignment function to perform argmax search for each j position by retrieving the probability from the model (line 163).

Step 2 Perform alignment from Greek to English and vice-versa (line 261) and perform heuristic symmetrization.

Step 3 Play with symmetrization heuristics (`heuristic=None` or `"FINAL"` or `"FINAL-AND"`)

Step 4 Implement Model 2 in the same way you did Model 1.

1. Adapt the formula to obtain the normalization factor $total_s$ (partition function — line 119) and the probability given: $value = f(e_i, f_j, j, i, le, lf)$ (line 125).
2. Adapt the alignment function to perform argmax search for each j position by retrieving the probability from the model (line 163).

Step 5 Congratulations! Not only you are now a Word Alignment expert but also a bible studies expert!

Note about threshold and convergence values : The final values concerning threshold and convergence have been already set within the code. Their values do not pretend to be the optimal ones. Their value has been already set according to speed limits (convergence) and a corpus size of around 200 sentences (threshold) considering that you implement both model 1 and model 2. We discourage completely to change these value when submitting your code to the Jutge Canvas platform software evaluator. However you are more than welcome to try different values while playing at your computer.

Credits This problem is adapted from one originally developed by Philipp Koehn, later modified by John DeNero and finally published by Adam Lopez, Matt Post and Chris Callison-Burch at `http://www.mt-class.org`