Statistical NLP - Assignment 4

Due Monday November 26 - at 11:59pm by email to anhad@nyu.edu and aparikh@cs.nyu.edu

In this assignment, you will explore the problem of word alignment, one of the critical steps in phrase-based machine translation.

Deliverables:

- Result file from Part 3 in github repository.
- Result files from Part 2 emailed to instructors.
- Code zip emailed to instructors containing changed code files and the result files from Part 2. You only need to include files that you have changed with clearly commented code indicating which parts correspond to which question.
- Writeup emailed to instructors. The recommended writeup length for this assignment is $2-3$ pages. To give you some freedom to not worry about space saving, you may submit a report up to 4 pages (including any figures / charts). Reports above 4 pages will be penalized.

Submission Format: Please title your submission email **Assignment 4 submission**. You should submit a zip file (firstname_lastname_hw4.zip) containing the report in pdf format (firstname_lastname_hw4.pdf) and the zipped code directory. Write your netID and your collaborators' netID (if any) in the report.

Setup: Please download **code-fall2018-a4.zip** and **data4.zip** from the same location as the previous assignments. The data consists of sentence-aligned French-English transcripts of the Canadian parliamentary proceedings.

The assignment harness is in the Java class:

```
nlp.assignments.WordAlignmentTester
```

Make sure you can run the main method of the `WordAlignmentTester` class. There are a few more options to start out with, specified using command line flags. Start out running:

```
java -server -mx500m nlp.assignments.WordAlignmentTester
     -path DATA -model baseline -data miniTest -verbose
```

You should see a few toy sentence pairs fly by, with baseline (diagonal) alignments. The verbose flag controls whether the alignment matrices are printed. For the miniTest, the baseline isn't so bad. Look in the data directory to see the source miniTest sentences. They are:

```
"English"                    "French"
<s snum=1> A B C </s>        <s snum=1> X Y Z </s>
<s snum=2> A B </s>          <s snum=2> X Y </s>
```

```
<s snum=3> B C </s>          <s snum=3> Y W Z </s>
<s snum=4> D F </s>          <s snum=4> U V W </s>

Alignments (snum, e, f, sure/possible)
1 1 1 S      3 1 1 S
1 2 2 S      3 2 3 S
1 3 3 S      4 1 1 S
2 1 1 S      4 2 2 S
2 2 2 S
```

The intuitive alignment is X=A, Y=B, Z=C, U=D, V=F, and W=null (convince yourself of this). The baseline will get most of this set right, missing only the mid-sentence null alignment:

```
[#]    | Y
    #  | W
   [ ]| Z
______'
 B  C
```

The hashes in the output indicate the proposed alignment pairs, while the brackets indicate reference pairs (parentheses for possible alignment positions). Note that at the end of the test output you get an overall precision (with respect to possible alignments), recall (with respect to sure alignments), and alignment error rate (AER).

You should then try running the code with `-data validate`, which will load the real validation set and test set, and run the baseline on these sentences. Baseline AER on the validation set should be 71.2 (lower is better). If you want to learn alignments on more than the test set, as will be necessary to get reasonable performance, you can get an additional `k` training sentences with the flag `-sentences k`. You will generally find that using more sentences increases performance (There are over a million sentence pairs there if you want them).

Make sure you are reading the data with UTF8 encoding (there should be accents in French), otherwise even a correct implementation will get terrible performance.

You will notice that the code is hardwired to English-French, just as the examples in class were presented. Even if you don't speak any French, there should be enough English cognates that you should still be able to sift usefully through the data. For example, if you see the matrix

```
[#]                 | ils
   [ ]( )    #      | connaissent
             #      | tres
             #      | bien
         [#]        | le
               [#]  | probleme
       #    ( )     | de
            [#]     | surproduction
                 [#]| .
--------------------'
 t  k  a  t  o  p  .
 h  n  b  h  v  r
 e  o  o  e  e  o
 y  w  u     r  b
       t     p  l
             r  e
             o  m
             d
             u
             c
             t
             i
             o
             n
```

you should be able to tell that "problem" got aligned correctly, as did "overproduction," but something went very wrong with the "know about" region.

Description: In this assignment, you will build three word-level alignment systems.

Part 1 (Heuristic) - 5 points: As a first step, and to get used to the data and support classes, you should build a heuristic replacement for `BaselineWordAligner`. Your first model should not be a probabilistic translation model, but rather should match up words on the basis of some statistical measure of association, using simple statistics taken directly from the training corpora. One common heuristic is to pair each French word f with the English word e for which the ratio $c(f, e)/(c(e) \cdot c(f))$ is greatest. Another is the Dice coefficient; many possibilities exist. Play a little and see if you can find reasonable alignments in a heuristic way.

You do not need to submit any output files for this part, but please describe what you tried in your writeup.

Part 2 (IBM Model 1 / 2) - 40 points: Once you've gotten a handle on the data and code, the first probabilistic model to implement is IBM model 1. Recall that in models 1 and 2, the probability of an alignment a for a sentence pair $(\mathbf{f}, \mathbf{e})$ is

$$P(\mathbf{f}, a|\mathbf{e}) = \prod_i P(a_i = j|i, |\mathbf{e}|, |\mathbf{f}|)P(\mathbf{f}_i|\mathbf{e}_j)$$

where the null English word is at position 0 (or -1, or whatever is convenient in your code). The simplifying assumption in model 1 is that $P(a_i = j|i, |\mathbf{e}|, |\mathbf{f}|) = 1/(|\mathbf{e}| + 1)$. That is, all positions are equally likely. In practice, the null position is often given a different likelihood, say 0.2, which doesn't vary with the length of the sentence, and the remaining 0.8 is split evenly amongst the other locations.

The iterative EM update for this model is very simple and intuitive. For every pair of an English word type e and a French word type f, you count up the (fractional) number of times tokens f are aligned to tokens of e and normalize over values of e (the math is in Mike Collin's notes from Columbia, posted on the course webpage). That will give you a new estimate of the translation probabilities $P(f|e)$, which leads to new alignment posteriors, and so on. For the `miniTest`, your model 1 should learn most of the correct translations, including aligning W with null. However, it will be confused by the DF / UV block, putting each of U and V with each of D and F with equal likelihood (probably resulting in a single error, depending on how ties are resolved).

Look at the alignments produced on the real validation set with your model 1. you will still see many alignments which have errors sprayed all over the matrices, errors which would be largely fixed by concentrating guesses near the diagonals.

To address this trend, you should now implement IBM model 2, which changes only a single term from model 1: $P(a_i|i, |\mathbf{f}|, |\mathbf{e}|)$ is no longer independent of i. For this part, first choose the probabilities proportional to $\exp(-\alpha|a_i - i \cdot |\mathbf{e}|/|\mathbf{f}||)$ (There are other options such as bucketing distances and learning more general distributions which you can explore in the next part).

Again, to make this work well, one generally needs to treat the null alignment as a special case, giving a constant chance for a null alignment (independent of position), and leaving the other positions distributed as above. How you bucket those displacements is up to you; there are many choices and most will give broadly similar behavior. If you run your model 2 on the miniTest, it should get them all right (you may need to fiddle with your null probabilities). How you parameterize the alignment prior is up to you.

For this part please submit the outputs for these vanilla implementations for IBM Model 1 and IBM Model 2 on both the toy set as well as the validation set:

- `java -server -mx500m nlp.assignments.WordAlignmentTester -path DATA -model model1 -data miniTest -verbose &> model1_mini_output.txt`
- `java -server -mx500m nlp.assignments.WordAlignmentTester -path DATA -model ibm_model1 -data validate -sentences 10000 -verbose &> model1_dev_output.txt`
- `java -server -mx500m nlp.assignments.WordAlignmentTester -path DATA -model ibm_model2 -data miniTest -verbose &> model2_mini_output.txt`
- `java -server -mx500m nlp.assignments.WordAlignmentTester -path DATA -model model2 -data validate -sentences 10000 -verbose &> model2_dev_output.txt`

Your score for this part will be based on correct implementation of the models.

Part 3 (Further Experiments to Boost Performance) - 30 points: Now that you have your vanilla models implemented, the goal of this section is to boost the performance. You can experiment with many aspects such as increasing the amount of training data (via the `-sentences` flag), changing the models, parametrizing the $P(a_i|i, |\mathbf{f}|, |\mathbf{e}|)$ term and alignment prior differently etc.

Having additional computational resources will be very helpful for you. You should have access to the CIMS cluster, let the instructors know if you do not.

Whenever you run the code, an output file with the word alignments for the test set will be generated in basepath/model.out. Please submit this file to GitHub as hw4/output.txt.

To eligible for full credit on this section, you are required to achieve a score of 17.3% AER on the test set. You will be eligible for partial credit up to this score.

Part 4 (Writeup) - 25 points: Please give a $2-3$ page writeup detailing your explorations, results and findings for this assignment. Please note this is the longer than previous assignments since there is more room for open ended exploration. You will be scored on two main criteria:

- The soundness / creativity of the different techniques you tried in the above parts. For full credit you should have tried more than just varying the training data and some hyperparameters.
- Presenting the results of your experiments thoroughly (even if not all of them achieved positive results).