Statistical NLP - Assignment 4

Due October 25 - at 5pm by email to petrov@cs.nyu.edu

In this assignment, you will explore the problem of word alignment, one of the critical steps in machine translation, and shared by all current statistical machine translation systems.

Setup: The code for this assignment is already included in the harness that you have been using. The data is available on the web page as usual, and 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`. Maximum values of `k` usable by your code will probably be between 10000 and 100000, depending on how much memory you have, and how efficiently you encode things. (There are over a million sentence pairs there if you want them – to use anywhere near that much, you will probably have to change some of the harness code.)

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. 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.

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 lecture slides in more detail). 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 or test sets with your model 1. You can improve performance by training on additional sentences as mentioned above. However, even if you do, 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. A common choice is to have these probabilities proportional to $\exp(-\alpha|a_i - i \cdot |\mathbf{e}|/|\mathbf{f}||)$, but other options exist, such as bucketing distances and learning more general distributions.

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.

Everyone should now have at least three systems, a non-iterative surface-statistics method, an implementation of model 1, and an implementation of model 2. If you don't have enough, feel free to experiment more, but it is not required for the assignment. Using some extra sentences and model 2 or better, you should be able to get your best AER down below 40% very easily and below 25% fairly easily, but getting it much below 15% will require extra work. Note: 5% is possible!

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 as hw4/output.txt.