cs294-5: Statistical Natural Language Processing
Assignment 4: Word Alignment Models

Due: Nov 22nd

Setup: The code for assignment 4 is under `/home/ff/cs294-5/java/src4`. The precompiled classes are in `home/ff/cs294-5/java/classes4`. The data for this assignment is in `/home/ff/cs294-5/corpora/assignment4`.

The assignment harness is in the java file at

> `…/assignments/WordAlignmentTester.java`

Make sure you can run the main method of the `WordAlignmentTester` class. New in this assignment is a facility for command line flags, as there are a few more options to start out with. Start out running

```
java –server –mx500m edu.berkeley.nlp.assignments.WordAlignmentTester
        –path .../assignment4 –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"                Alignments (snum, e, f,
<s snum=1> A B C </s>    <s snum=1> X Y Z </s>   sure/possible)
<s snum=2> A B </s>      <s snum=2> X Y </s>     1 1 1 S
<s snum=3> B C </s>      <s snum=3> Y W Z </s>   1 2 2 S
<s snum=4> D F </s>      <s snum=4> U V W </s>   1 3 3 S
                                                 2 1 1 S
                                                 2 2 2 S
                                                 3 1 1 S
                                                 3 2 3 S
                                                 4 1 1 S
                                                 4 2 2 S
```

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

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

The hashes 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 overall precision (with respect to possible alignments), recall (with respect to sure alignments), and alignment error rate, which is something like 1.0-F1.

You should also try running the code with –data validate and –data test, which will give the validation set and test set respectively. Baseline AER on the test set should be 68.7 (lower is better). If you want to learn alignments on more than the test set, you can get an additional k 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 you encode things. (There are a million or so sentence pairs there if you want them – to use anywhere near that much, you'll have to change some of the harness code.)

You'll notice that the code is hardwired to English-French, just as the examples in class were presented. If you don't speak any French, this assignment may be a little less fun, but French has enough English cognates that you should still be able to sift usefully through the data. For example, if you see the matrix

```
[#]                         | ils
   [ ]( )      #            | connaissent
               #            | très
               #            | bien
         [#]                | le
               [#]          | problème
      #     ( )             | 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 handled correctly, as did "overproduction", but something went very wrong with the "know about" region, even without speaking any French.

Description: In this assignment, you will build several word-level alignment systems. As a first step, and to get used to the data and support classes, build a replacement for BaselineWordAligner which matches up words based on superficial statistics. One common stab is to pair each French word f with the English word w for which the ratio

$$P(f,e)/[P(f)P(e)]$$

is greatest. Other possibilities exist; play a little and see if you can detect any useful translation pairs with such methods.

Once you've gotten a handle on the data and code, the first real model to implement is IBM model 1. Recall that in models 1 and 2, the probability of an alignment a for a sentence pair (*f,e*) is

$$P(f,a \mid e) = \prod_{i=1}^{len_f} P(a_i \mid i, len_f, len_e) P(f_i \mid e_{a_i})$$

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

$$P(a_i \mid i, len_f, len_e) = P(a_i) = 1/(len_e + 1)$$

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 English word type *e* you count up how many French words are aligned with tokens of *e* (the answer will be a fraction), as well as the distribution over those French words' types. 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 numerical issues fall).

Look at the alignments produced on the 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 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: $P(a_i|i, len_f, len_e)$ is no longer independent of *i*. A common choice is to have

$$P(a_i \mid i, len_f, len_e) \propto d(bucket(a_i - i \frac{len_e}{len_f}))$$

Here, $P(a_i|i, len_f, len_e)$ is parameterized by a rough displacement from the diagonal. 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 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).

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. At this point, you should do another substantial piece of investigation, but the field is wide open. Some options:

- [Probably the easiest if you like HMMs] Implement the HMM alignment model from Vogel, Ney and Tillmann, "HMM-based word alignment in statistical translation", Proceedings of COLING 1996
    http://acl.ldc.upenn.edu/C/C96/C96-2141.pdf
- Implement the competitive linking algorithm from I. Dan Melamed, "Models of Translational Equivalence among Words", Computational Linguistics, 2000.
    http://www.cs.nyu.edu/~melamed/ftp/papers/clmote.pdf
- Investigate a phrase-based heuristic of some kind to model multi-word trends (can be a hack, if you want)
- [Probably the easiest for code-optimizers] Scale your system up to a large amount of data (defined as, say, running on at least 100K training sentences)
- [Probably the easiest for linguist types] Substantial data analysis that shows something interesting

Using some extra sentences and model 2 or better, you should be able to get your AER down below 40% very easily and below 35% fairly easily, but getting it much below 30% will likely require greater effort.