11-711: Algorithms for NLP
Assignment 4: Word Alignment

Due 11:59pm ET, Dec 4, 2017

Collaboration Policy

You are allowed to discuss the assignment with other students and collaborate on developing algorithms at a high level. However, your writeup and all of the code you submit must be entirely your own.

Setup

As usual you will need:

1. `assign_align.tgz`
2. `data_align.tar.gz`

You will find it useful to consult Vogel, Ney and Tillmann 1996 "HMM-based Word Alignment in Statistical Translation."

Preliminaries

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

Data The data for this assignment 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:

```
edu.berkeley.nlp.assignments.align.AlignmentTester
```

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

```
java -cp assign_align.jar -server -mx500m
  edu.berkeley.nlp.assignments.align.AlignmentTester
  -path path/to/align_data -alignerType BASELINE -data miniTest -printAlignments
  -justAlign -maxTrain 0
```

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` and `-data test`, which will load the real validation set and test set respectively, and run the baseline on these sentences. 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, as will be necessary to get reasonable performance, you can set the flag `-maxTrain` to a value k, which will load in k additional sentences of training data. 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'll have a machine with large amounts of RAM.)

You'll 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.

Of course, the actual word to word alignments aren't the only thing we're interested in. The main use of these alignments are as input to the rule extraction procedure. So in addition to the word alignments themselves, we're also going to look at the phrase tables that are extracted from the alignments. To see this, try running the following command:

```
java -cp assign_align.jar -server -mx500m
     edu.berkeley.nlp.assignments.align.AlignmentTester
     -path path/to/align_data -alignerType BASELINE -data miniTest
     -phraseTableOut phraseTable.txt
     -randomLm -maxTrain 100
```

You should see a brief description of a phrase table being trained, and then several (very bad) translations. The test harness uses the same data as the aligner (the alignment test set, plus however many training sentences you specified with `-maxTrain`) to extract a phrase table, and then uses it to translate the decoding test set. The phrase table that was extracted will be written to `phraseTable.txt`, so you can examine it to see what kinds of errors are resulting from your word alignments. The translations you just saw are bad for two reasons: first, the phrase table is just very small, being trained on a trivial amount of data. Second, even if run on more data, the baseline word aligner is quite bad and will result in a lot of bogus translation rules. Your goal for this assignment will be to write more credible word aligners, being careful to ensure that they are still efficient enough to scale up to more reasonable amounts of training data, as both the quality of the alignments and the number of training sentences are important in improving translation quality (note: you should be sure remove the `-randomLm` option to run real experiments, so that an actual language model is employed).

Description

In this assignment, you will implement three word-alignment models:

1. A heuristic aligner
2. IBM Model 1
3. The HMM model of Vogel et al. (1996)

Heuristic aligner As a first step, and to get used to the data and support classes, you should edit `HeuristicAlignerFactory` and 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, described in several of the readings. Many possibilities exist; play a little and see if you can find reasonable alignments in a heuristic way.

Note that this method should easily scale up to use all of the training data; simply omit the `-maxTrain` flag (or set it to a high value) to run on all training examples.

Model 1 The first probabilistic model to implement is IBM Model 1, built by `Model1AlignerFactory`. 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 as follows. 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).

HMM If you inspect the output of your Model 1 aligner, you should notice that errors in alignments frequently arise from the model "jumping around" dramatically, whereas we know that for these two closely related languages, alignments should be roughly monotonic. To fix these errors, you need to introduce a distortion model that can learn that alignments tend to clump together, with adjacent English words usually aligning to adjacent French words.

For the HMM model (built by `HmmAlignerFactory`, the distortion component is augmented to include the previous alignment choice a_{i-1}:

$$P(a_i = j|i, a_{i-1}, |\mathbf{e}|, |\mathbf{f}|)$$

One common parameterization is to set $d = j - a_{i-1}$, and learn a distribution over values of d, but other options exist, such as bucketing distances and learning more general distributions. Compared to Model 1, you will now need to compute edge marginals $p(a_i, a_{i-1})$ in your E-step so that you will be able to optimize these parameters in the M-step.

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 HMM model 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.

Using some extra sentences and the HMM 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 some work (5% is possible!). For reference, our implementation of intersected[1] Model 1 achieves a test-set AER of 33% when trained with 10000 sentences, while our intersected HMM implementation achieves 16%. As for BLEU scores, they will be low for this assignment, unless you scale up to large amounts of data. Our implementation of intersected Model 1 achieves a BLEU of 10.7 when trained on 10000 sentences. Our implementation of the intersected HMM Model achieves 18.2 on 10000 sentences and 23.3 on 100000 sentences.

Submission and Grading

Write-ups You should turn in a 2-3 page write-up as usual. You should clearly describe what implementation choices you made, compare the performance of the various aligners, and provide error analysis.

[1] By "intersected", we mean that a model was trained independently in each direction, and the resulting alignments were intersected in post-processing.

Submission You will submit `assign_align-submit.jar` and a PDF of your writeup to an online system. We will sanity-check with the following command:

```
java -cp assign_align.jar:assign_align-submit.jar -server -mx50m
     edu.berkeley.nlp.assignments.align.AlignmentTester
     -path path/to/align_data -alignerType HMM -sanityCheck -maxTrain 0
```

We will grade with the following command:

```
java -cp assign_align.jar:assign_align-submit.jar -server -Xmx8g
     edu.berkeley.nlp.assignments.align.AlignmentTester
     -path path/to/align_data -alignerType HMM -maxTrain 10000
     -data test
```

Grading You should aim to get a comparable AER and BLEU score to those of the reference with your HMM aligner on 10000 training sentences. Your ability to scale up and use more data will also be evaluated; you should give results for large-scale experiments in your writeup. We are not too concerned with the performance of your heuristic aligner – we just want to see that you thought about the problem and designed something reasonable. As always, the best submissions will be those that perform thoughtful error analysis and/or describe interesting extensions to the basic aligners in this assignment.

Although there no hard requirements on training time, due to limited compute resources for autograding, we will expect your aligners to take no more than one hour to train with `-maxTrain 10000`. For larger-scale experiments, you should report training time in your writeup.