CSE 517, Winter 2013: Assignment 4

Due: Friday, March 8th at 5pm

In this assignment, you will implement three phrase-based decoders and test them with the provided harness. The data and support code are available on the course Dropbox in Catalyst. Please submit two files: a PDF (with your name) containing your writeup and a single archive (zip, tar.gz, etc.) including all the code and your built binaries. Don't submit the data.

The testing harness we will be using is `MtDecoderTester` (in the `edu.berkeley.nlp.assignments.assign2` package). To run it, first unzip the data archive to a local directory (`PATH`). Build the submission jar using `ant`. Then, try running

```
java -cp mt-lib.jar:mt-submit.jar -server -mx2000m
    edu.berkeley.nlp.assignments.decoding.MtDecoderTester
    -path PATH -decoderType MONO_GREEDY
```

You will see the tester do some translation of 100 French sentences, which will take a few seconds, printing out translations along the way (note that printing of translations can be turned off with `-noprint`). The decoder is a very simple monotonic decoder which greedily translates foreign phrases one at a time. Its accuracy should be poor (BLEU score around 20), as should its model score (around 5216; see below for more about the model).

1 Building Better Decoders

Your task is to implement three decoders and test them. Each of your decoders will attempt to find the translation with maximum score under a linear model. This linear model consists of three features: a trigram language model; a linear distortion model; and a translation model which assigns scores to phrase pairs. Each of these features is weighted by some pre-tuned weights, so you should not need to worry about these weights (though you may experiment with them if you wish). The translation model table and language model are provided, so you can focus on decoding.

You will need to implement three decoders of increasing complexity. We provide results of one specific implementation to guide you, without specifying exactly how it works. You are **not** required to get these exact numbers. However, each decoder should show improvement in both model score and BLEU score over the previous one. The only exception is the first one, which is expected to have lower performance when compared to the provided baseline.

To improve performance, you will adopt more complex models, but might also need to implement better beam and lattice management strategies. You are welcome to implement any approaches you want, including for example using different beams per subsets of foreign words or the lattice approach in the Collins notes, but should be careful to document what you did in the writeup.

You first two decoders will be monotonic, while the third will allow distortion. In a monotonic decoder, no distortion is allowed. Specifically, you must select phrases to translate from the foreign sentence in a left to right fashion, without skipping any foreign words at each step. When you add distortion, such skipping will be allowed.

The first decoder should implement a monotonic search with no language model. You should return an instance of such a decoder from

`MonotonicNoLmDecoderFactory`

Using our implementation, with beams of size 2000, we decode the test set in 16 seconds and achieve a model score of 5276 and BLEU score of 17.2. Note that this decoder performs poorly even compared to the greedy decoder since no language model is used. Remember, given your choice of strategy for managing the beam(s) or lattice, you should not expect to match these numbers exactly. But, it would be surprising if your numbers are significantly lower, and it is definitely possible that you could do better!

The second decoder, which should be constructed in

`MonotonicWithLmDecoderFactory`

It should implement monotonic search with an integrated trigram language model. Using our implementation, with beams of size 2000, we decode the test set in 2 minutes and achieve a model score of 3781 and BLEU score of 26.5.

The third decoder, which should be constructed in

`DistortingWithLmDecoderFactory`

It should implement a search strategy that permits limited distortion, as discussed in class. The distortion score can be retrieved from DistortionModel class, as can the distortion limit. Using our implementation, using beams of size 2000, we decode the test set in about 20 minutes and achieve a model score of 3529 and BLEU score of 27.5.

2 Coding Tips

In general, expect the decoders to be slower as they become more complex. Pay attention to your implementation, or you will end up with a very slow development process. One data structure that is key to this kind of work is `PriorityQueue`. Unfortunately, The implementation provided by Java is relatively slow. However, we provide faster implementations. Look for the interface `edu.berkeley.nlp.util.PriorityQueue` and the classes implementing it. We strongly recommend using it.

3 Evaluation

The task of an MT decoder is to take a foreign sentence and find the highest scoring English translation according to a particular model. If the model is a good one, then translation accuracy (BLEU) will correlate with model score, though this will not always be the case. As such, we will primarily evaluate your decoder on what model score it achieves on the training data, though we will measure BLEU as well. Also, with all decoders, there is a speed vs. accuracy tradeoff: you can always achieve a higher model score with a slow decoder, or decode very quickly if you do not care about making large numbers of search errors and achieving a low model score.

For your writeup, please describe your pruning strategy for managing the beams of your decoders, with a brief description of why you selected it. Additionally, if you tried approaches that were not able to get consistent improvements across the three conditions, please include a paragraph explaining what you tried and why it did not work.

You should report both BLUE and model scores. Additionally, please report decoding time, which is printed by the test harness. Since we all use different machines, please report CPU speed and allocate 2GB of memory to your JVM.

To generate results you will need to run the following commands:

```
java -cp mt-lib.jar:mt-submit.jar -server -mx2000m
    edu.berkeley.nlp.assignments.decoding.MtDecoderTester
    -path PATH -decoderType MONO_NOLM
java -cp mt-lib.jar:mt-submit.jar server -mx2000m
```

```
    edu.berkeley.nlp.assignments.decoding.MtDecoderTester
    -path PATH -decoderType MONO_LM
java -cp mt-lib.jar:mt-submit.jar -server -mx2000m
    edu.berkeley.nlp.assignments.decoding.MtDecoderTester
    -path PATH -decoderType DIST_LM
```

Before you submit your code, please verify that these commands execute properly.

4 Writeup (max. 4 pages)

For the write-up, we want you to describe what you've built and analyze your results. For each decoder, please provide a description that will allow us to understand what you implemented and why. Include tables or graphs of BLEU, runtime, model score, etc., of your systems, as well as enough error analysis to convince us that you looked at the specific behavior of your systems and thought about what it's doing wrong and how you'd fix it. Your error analysis should include interesting sample sentences with detailed failure analysis.

The assignment was adapted from Dan Klein's CS 288 Course at UC Berkeley.