CoSi 139a: Machine Translation
Assignment #2 Due 3/26/12

The task in this assignment is to implement a word-based English-to-Spanish statistical translation model following the pattern of IBM Model 1, as well as the process for training this model via Estimation-Maximization. Your solution may be in any programming language you choose. Please submit it by email to both dwkirsch@brandeis.edu and nkrishna@brandeis.edu.

Components:

To succeed, your solution will need to be able to do the following:

For both training and evaluating:
- Accept instructions from command-line arguments (specified below)
- Read in data (format for evaluation data specified below)
- Tokenize
- Represent alignment permutations between sentence pairs
- Access a lexicon
- Compute Model 1 alignment probabilities

For training:
- Initialize, store, access, and update a lexicon
- Create, store, and access a language model
- Represent alignment permutations between sentence pairs
- Perform EM algorithm
- Detect convergence

For evaluating:
- Choose the best among sets of translation candidates using Model 1

The core aspects of this assignment are the alignment representation, the Model 1 probability computation, and the EM algorithm. These should be implemented from scratch in your solution.

For other aspects, you may incorporate any tool at your disposal. For example, feel free to use:
- any existing tokenizer (hint: there's one in the tools on the Europarl download page).
- any existing command-line argument parser (for example GetOpt not @ARGV in Perl, optparse not sys.argv in Python, Trollop not ARGV in Ruby). This is a good habit.
- any database engine (though I strongly recommend using SQLite) and any database interface library (e.g. Python's sqlite module, Ruby's DBI gem, etc).

Any tools (scripts, modules, etc) required by your solution should either be included in your solution directly or have their installation described by easy-to-follow instructions in an included readme file.

Command line format:

Your solution should accept the following command line options:

--mode, -m	Options: train, evaluate
--source, -s	Location of target languag data file (for either training or evaluating)
--target, -t	Location of foreign (Spanish) data file (for either training or evaluating)
-i	Maximum iterations for EM (stop by this # even without convergence)
--help, -h	Print this usage information (a good habit)

Feel free to include others that seem sensible to you or are useful to you during development.

Data and file formats:

For training and development, please download the Spanish-English section of the Europarl corpus here (170MB, available here: http://www.statmt.org/europarl/). The file you download will be a gzipped tar archive -- you can extract the files by running "tar -xzvf es-en.tgz". The content consists simply of two plain-text files, one each for English and Spanish, with roughly 1.7 million sentences separated by line breaks. You will want to split out sensibly-sized subsets of the corpus for your development testing and ultimately for training.

As you're testing your solution, keep in mind the following rule of thumb (hardware environment issues notwithstanding): a well-designed solution should be able to run each EM iteration on 100 sentence pairs in under a second and each iteration on 1000 sentence pairs in a handful of seconds. The Koehn textbook has refactorizations of the model which are important for efficiency, described in pages 89-91. Please use them!

In training mode, your program should expect files in the format downloaded from Europarl. In evaluation mode, your program will need to accept multiple English candidate translations for each Spanish sentence. Therefore, please use the following format to align evaluation files:
1<tab>First candidate for Spanish sentence 1.
1<tab>Second candiate for Spanish sentence 1.
2<tab>First candidate for Spanish sentence 2.
The Spanish data should also follow this format, but will have only one sentence for each id.

Grading:

70% Correctness
15% Efficiency/performance
15% Style and documentation

This breakdown is basically just an encoding of the following qualitative rules:

-A solution that installs cleanly, computes everything correctly, and has clean, organized, readable code will earn a perfect score.
-A solution that pretty much does what it needs to, but has some minor errors can still receive an excellent grade if the code is efficient and organized.
-A solution that fails to execute a major component can still receive a passing score if other components work well, and effort has been made to keep things efficient and organized.
-A solution that correctly implements everything in the components list but is horribly slow, has a underspecified installation process, and has poorly organized code will pass, but barely.