CSE 250B Project 2

For this project you should work in a team of exactly two students. The joint report for your team must be submitted in hard copy at the start of the lecture on Tuesday February 11, 2014.

The experimental task is to learn a conditional random field (CRF) model that can add punctuation (commas, exclamation marks, etc.) to English sentences in which the punctuation is absent. The intended application is for typing on a smartphone, where the keyboard may be too small to show letters and punctuation marks simultaneously. We want to allow the user to type only the words of a sentence, with punctuation added automatically by software. The dataset for training will consist of over 10,000 sentences with punctuation taken from email messages, and will be available at `http://www.cs.ucsd.edu/users/elkan/punctuation/`.

You must understand the basic algorithms for conditional random fields (CRFs) thoroughly, and implement these algorithms yourself. The implementation should handle any small set of output tags. For this application, the tags are punctuation symbols. For each sentence, there is always exactly one output tag for each word in the sentence. The tags should include a symbol meaning that there is no punctuation after a word. Make a reasonable decision about what tags to use for the notional positions immediately before and after the first and last words of a sentence.

Design and implement a process for generating feature functions automatically from high-level specifications such as "word with 'ing' as its last three letters." In your report, explain this process. The implementation of the process does not have to be general-purpose, that is the code does not need to be usable for other applications. However, the process itself should be systematic, meaning that a relatively small program should enumerate all feature functions in several large classes. Make sure that each feature function has value zero for most word-tag pairs, and that this common case is handled efficiently both during data generation and during CRF training.

Implement and do experiments with two different training methods. The first method should be Collins' perceptron algorithm, and the second should be one of the following:

1. stochastic gradient following.

2. a general-purpose numerical optimization package such as an implementation of L-BFGS, or

3. contrastive divergence.

For each training method, implement all needed CRF-specific algorithms yourself. Note that perceptron training requires only the Viterbi algorithm, while contrastive divergence needs you to understand and implement Gibbs sampling. L-BFGS and stochastic gradient following require computing the gradient.

It is crucial to be confident that basic algorithms are implemented correctly. Your report must convince the reader that this is true. For methods that use gradients, use a function to verify that computed derivatives are correct. After you are sure that your algorithm implementations are correct, it is also vital to make them fast. Use a small random subset of sentences to maximize the speed of your code. One of the big advantages of Matlab is that it provides an excellent profiler that can show you easily which parts of your code are slow.

After you have a correct, fast, and robust implementation of each learning algorithm, tune its settings using a validation set (not cross-validation). Use word-level accuracy as the performance metric. This is simply the fraction of words for which the algorithm predicts correctly the following punctuation symbol.