Homework #1: CMPT-825

Anoop Sarkar – anoop@cs.sfu.ca

For the programming questions use a `makefile` such that `make` compiles all your programs, and `make test` runs each program, and supplies the necessary input files. Submit a short readme file describing your approach as a LaTeXor ASCII file.

(1) **Global Linear Models**

This assignment will focus on the training and use of global linear models for tagging tasks in NLP. We will be following the approach laid out in the following paper:

> Michael Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. EMNLP 2002.

We will be training a perceptron model that uses global features for a tagging task. We will be tackling the task of phrase chunking in text, finding non-recursive phrasal chunks using supervised training data. The data is in the form:

```
But CC O
analysts NNS B-NP
reckon VBP B-VP
underlying VBG B-NP
support NN I-NP
for IN B-PP
sterling NN B-NP
has VBZ B-VP
been VBN I-VP
eroded VBN I-VP
by IN B-PP
the DT B-NP
chancellor NN I-NP
's POS B-NP
failure NN I-NP
```

The first column is the word, the second column is the part of speech tag which for the test data is obtained from a supervised part of speech tagger and the third column is the output we wish to produce using our model – the chunk tag that indicates where a word begins a phrase, e.g. B-NP, or is inside a phrase, e.g. I-NP or if the word is outside any chunk then it is labeled with O.

Global linear models are discriminative models and use a large set of overlapping features. For each input sentence $\mathbf{x} = x_1, \ldots, x_n$ and output chunking $\mathbf{y} = y_1, \ldots, y_n$, we create a list of global features $\Phi(\mathbf{x}, \mathbf{y})$ and pick the best chunking $\mathbf{y}$ using a weight vector $\mathbf{w}$:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}'} \mathbf{w} \cdot \Phi(\mathbf{x}, \mathbf{y}')$$

I have provided the following files for the chunking task. The files are derived from the CoNLL 2000 chunking shared task, which in turn was created from the Penn Treebank WSJ corpus by keeping only the non-recursive chunking structure of the Treebank.

- `train.txt.gz` – the training data in the format shown above

- train.feats.gz – the feature vector $\Phi(\mathbf{x}_i, \mathbf{y}_i)$ for each training data instance $\mathbf{x}_i, \mathbf{y}_i$. The representation does not include the $\mathbf{y}_i$ labels which need to be added in during the training process.
- test.txt.gz – the test data in the format shown above
- test.feats.gz – the feature vector $\Phi(\mathbf{x}_i, \mathbf{y}_i)$ for each test data instance $\mathbf{x}_i$. The representation does not include the $\mathbf{y}_i$ labels which need to be added in during the decoding process that finds the argmax chunk labels.
- conlleval – the evaluation script that provides the accuracy numbers on the test data
- perc.py – Python code that reads in the various data files and sets up the data for the training and testing functions that you will provide

Use perc.py in order to train a perceptron on the training data. You will need to implement the HMM Viterbi algorithm that uses the score $\mathbf{w} \cdot \Phi(\mathbf{x}, \mathbf{y}')$ to rank each sequence of chunk tags for a given input. As specified in the Collins paper above, the training process is a simple mistake-driven update procedure that runs through a fixed number of epochs over the training data and returns a weight vector $\mathbf{w}$ for each feature $\Phi(\mathbf{x}, \mathbf{y}')$. The weight vector is then used to find the chunk labels in the test data by simply re-using the Viterbi algorithm that was used for training but this time with a fixed weight vector obtained from the training process. Report the accuracy on the test data.