Machine Learning, **Individual** HW 3: Structured Prediction

due Wednesday Nov 22, 11:59pm on Canvas

In this assignment you will

1. understand (but not implement) (a) the dynamic programming algorithm used in the Viterbi decoder for part-of-speech tagging and (b) maximum likelihood estimation for the HMM model;
2. train a part-of-speech tagger using averaged perceptron;
3. experiment with feature engineering in discriminative models;

The dataset `hw3-data.tgz` contains:

1. `train.txt.lower.unk` – training data (488 sentences)
2. `dev.txt.lower.unk` – development data (49 sentences)
3. `test.txt.lower.unk.unlabeled` – semi-blind test data (48 sentences)
4. `tagger.py` – my Viterbi decoder and HMM MLE estimator

Each line in the data files (except test) is a sentence made of a sequence of word/tag pairs:

```
word_1/tag_1 word_2/tag_2 ...
```

Also note that for your convenience, I have done the following preprocessing steps: (a) all words are lowercased. (b) words occuring once or less in the training data has been replaced by `<unk>`. You just need to treat `<unk>` as a normal word.

0 Viterbi Decoder and Maximum-Likelihood Estimation for HMMs

To simply your work, I have implemented for you both the Viterbi decoder (dynamic programming) and the maximum likelihood estimation for HMMs. You need to understand every single line of my `tagger.py`.

Run `python tagger.py train.txt.lower.unk dev.txt.lower.unk`, and you should get:

```
train_err 2.73%
dev_err 5.26%
```

Note: in this HW, for each word w, only consider the tags that associate with it in the training set.

1 Structured Perceptron

You should implement the structured perceptron using an HMM Viterbi decoder (see slides).

1. First just use two feature templates: $\langle t, t' \rangle$ and $\langle t, w \rangle$.

   Training **unaveraged** perceptron for 5 epochs. Which epoch gives the best error rates on dev?

   Hint: at the end of each epoch, your code should output the weights only if it achieves a new highest accuracy on dev (so that at the end of training this file has the best weights).

   You should also output logging info which would output something like this:

```
epoch 1, updates 102, features 291, train_err 3.90%, dev_err 9.36%
```

Note: `trainer.py` should `import tagger` and reuse the Viterbi decoder and the dictionary.

2. Now implement the averaged perceptron. What is the new best error rate on dev?
3. How much slower is averaged perceptron? Compare the time between unaveraged and averaged version.
4. Plot a single plot that includes four curves: {unaveraged, averaged} x {train_err, dev_err}. What did you find from this plot?
5. Explain why we replaced all one-count and zero-count words by `<unk>`.

2 Feature Engineering (based on averaged perceptron)

Now you can play with your (averaged) perceptron to include as many feature templates as you want.

Report what templates you tried, which helped and which did not; your best set of templates, and the best accuracies on dev and test you get.

Hint: to simplify your experiments, you might want to define the feature templates in a file and let your code automatically figure out those templates on the fly, i.e., you can define $t_i (i = -2, -1, 0)$ to be a tag in the local window with t_0 being the tag for the current word, and similarly $w_i (i = -2, -1, 0, 1, 2)$ to be a word with w_0 being the current word. So for example you can define the following two templates:

```
t0_t-1
t0_w0
```

which corresponds to the model in Section 1, and you can add

```
t0_t-1_t-2
t0_w-1_w+1
...
```

and so on, so that your `trainer.py` does not need to be changed (first make sure your Viterbi is extended to handle trigrams).

Make sure each template is smoothed (i.e., all back-off versions are included: e.g., if you include `t0_t-1_w0`, then also include its three backoff versions: `t0`, `t0_t-1`, and `t0_w0`) and don't use over-complicated templates since they will be too sparse. Also note that using more t_i's slows down dynamic programming, while using more w_i's doesn't (since they are input and static; recall that discriminative models do not model input and do not impose independence assumptions on it). However, the sparsity of the latter increases the dimensionality of feature vector very quickly, which also slows down the whole thing. In general, you should avoid word trigrams (like `w0_w1_w2`). Also, every template should include `t0` (why?).

Include your the best accuracy you achieved on the dev set, and label the test file using that model. Include both `dev.lower.unk.best` and `test.lower.unk.best` (the resulting word/tag sequences).

Debriefing

Please answer these questions in **debrief.txt** and submit it along with your work. **Note:** You get 5 points off for not responding to this part.

1. How many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Are the lectures too fast, too slow, or just in the right pace?
4. Any other comments (to the course or to the instructors)?