Machine Learning, Spring 2013, HW 3: Structured Prediction

due Thursday May 16, 11:59pm on blackboard

In this assignment you will

1. train a part-of-speech tagger using averaged perceptron;
2. experiment with feature engineering in discriminative models;
3. (optional) experiment with learning with greedy search.

The dataset `hw3-data.tgz` contains three files

1. `train.txt.unk` – training data (488 sentences)
2. `dev.txt.unk` – development data (49 sentences)
3. `test.txt.unk` – test data (48 sentences)

Each line in these files is a sentence made of a sequence of word/tag pairs:

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

Also note that words occuring once or less in the training data has been replaced by `<unk>`. You just need to treat `<unk>` as a normal word.

1 Structured Perceptron

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

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

   Try training **unaveraged** perceptron for 20 iterations. **Plot the learning curve:** the x axis is the training iteration (number of passes over the whole training data), and the y axis is the accuracy on dev set.

   Which iteration gives the best accuracy on dev, and what's the testing accuracy using the weights from that iteration?

   Hint: at the end of each iteration, your code should output the weights (to the file specified in the command-line) 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 to the standard error, which would output something like this:

   ```
   ...
   at iteration 5, updates_on_train= 50, dev= 90.34%, |W|= 37437, time= 12.3m
   ...
   ```

   so that you can use `gnuplot` to plot the learning curve very easily.

   To summarize, the suggested command-line would be (use `gflags` – Google it):

```
python trainer.py --train train.wordtags --dev dev.wordtags -i 20 \
                  --dict train.dict --out weights 2>logs

cat test.wordtags | ./tagger.py -w weights --dict train.dict
```

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

2. Now implement the averaged perceptron and do the same experiment (simply add a command-line switch `--avg`). What is the best iteration on dev, and what is the new accuracy on test?

   Submit a single plot that includes the learning curves of both unaveraged and averaged perceptron. What did you find from the comparison? Look at the logging files, how much slower is averaged perceptron?

   Important: computing the averaged weights at the end of each iteration should **not** alter the (unaveraged) weights which is used as the initial weights of the next iteration.

3. Explain why we replaced all one-count and zero-count words by `<unk>`, and what would happen if we don't do that.

2 Feature Engineering

From now on set `--avg` **to be true by default.**

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 a (gzipped) weights file from the best model you got.

3 Structured Perceptron with Greedy Search (extra credit)

Try any of the following update methods (of course with averaging), and compare them with standard averaged perceptron on the plot (see slides for details and references). Which one was the best?

Also, include another plot with x-axis being the wall-clock time instead of the number of iterations. What can you observe on this new plot?

1. early update (Collins and Roark, 2004).
2. max-violation update (Huang et al, 2012).
3. LaSO update (Daume and Marcu, 2006).

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)?