Lucino Chiafullo

Averaged Perceptron Report

My implementation of a Part-of-Speech tagger, using the Averaged Perceptron and Viterbi algorithms, achieves an accuracy of .944 on the development set and .95 on the test set, consistent with the expected results. This will be discussed in full later in the report. First, I will describe the structure of the code.

The provided files were used, and some functions were changed (for example, the functions to read data into sentence objects take an additional, optional vocabulary argument). To run the project, the given python call[1] can be used for `train_test_tagger.py`, which trains the tagger on the training data and tags the development and test data. At the end of each epoch in the perceptron algorithm, a message will print to the console indicating that a training epoch has finished. Tagged development and test data are stored in the files `dev.tagged` and `test.tagged`. To evaluate the results, call `scorer.py` with the command line arguments `[gold].tagged` and `{dev/test}.tagged`. The scorer file has only been modified to print three decimal places of the accuracy. The bulk of my work was done in the provided files `data_structures.py` and `perceptron_pos_tagger.py`; `data_structures.py` stores the textual content of a sentence in a property of the object, and stores the features for each token in the sentence in a list of lists – inner lists hold the features for each token, and can be indexed as such. There are measures to ensure that if a word does not appear in the vocabulary from the training data, it will be replaced with the Unknown feature vector (all zeroes). Meanwhile, `perceptron_pos_tagger.py` has three main functions: `viterbi`, `tag`, and `train`. The parameters are stored in a dictionary called `self.weights`, where each feature is a key with a 45-length vector as a value (each index corresponds to a part of speech, the categories we are trying to predict).

I experimented first with the desired number of perceptron training epochs, then with the features to include (the ablation study). The set of epochs tested is {1, 2, 3, 5, 10}, all on the full feature-set for consistency. Results are reproduced below:

Epoch	**Accuracy**
1	0.925
2	0.937
3	0.941
5	0.944
10	**0.947**

Performance gain after 5 epochs was fairly small; 5 additional epochs, taking twice as long to train, resulted in .003 points of performance gain. We do see performance increase a good amount when

[1]
```
python train_test_tagger.py train/ptb_02-21.tagged dev/ptb_22.tagged dev/ptb_22.snt test/ptb_23.snt
```

training with additional epochs early in the process; .012 points from 1 to 2 epochs, .004 from 2 to 3, and .003 from 3 to 5. For this reason, I performed feature experiments with 5 training epochs, where the model seemed to converge.

Before ablating features, I compared performance of the perceptron versus the averaged perceptron model. By commenting out line 102 in the perceptron file, the model is no longer an averaged perceptron model, as the weights are not replaced by the average weights. As we see below, the averaged perceptron model shows slightly better performance, and was used for the feature experiments as a result.

Model	**Accuracy**
Perceptron	.935
Averaged Perceptron	**.944**

The ablation study involved removing one feature at a time in order to measure each feature's contribution to the greater good of the project. I also performed an experiment using only the current word and previous tag features, which I considered to be the "base" features. Results are reproduced below:

Feature set	**Accuracy**
Full feature set	**.944**
No "current word" feature	.890
No "past words" features	.933
No "future words" features	.925
No "prefix/suffix" features	.929
Only current word/last tag	.903

We see losses in performance with the removal of each feature. When removing the "current word" feature, performance drops by .054; over 5 percentage points of accuracy. This is certainly significant, proving that it is one of the most important features. When removing the two past words, performance decreases by .011 points. This context seems less important than the future words, the removal of which decreases the accuracy by .019 points. Finally, I found that removing the prefix and suffix features (which were constructed by naïvely slicing the first and final 2 characters from each word, paying no attention to the tag) lost .015 points in the accuracy. The performance of the model when only using the current word and previous tag features scored an accuracy of .903, a loss of .041 points from the fully featured model. It seems from the experiments that other than the previous tag feature, which is integral to the Viterbi algorithm, the most important individual feature is the current word itself.

I was interested to see how quickly this algorithm could be trained; an extremely accurate part-of-speech tagger can be trained and used in five minutes using a very elegant algorithm. I look forward to working towards improving my model in the future.