This project is my implementation of applying perceptron algorithm on POS tagging of Penn Treebank dataset. And I use Viterbi to decode the sequence. On average, the program takes 3 hours to run on my mac, including loading data, feature representation, training and evaluating. The classifier's accuracy is around 93%.

Feature representation

We use dictionary to store global weighs of features. The keys are features represented by tuple, and corresponding values are weights.
Keys has following representation for a given tag
1. Current word: word0 – (tag, word0, 0)
2. Previous words: word-1, word-2 – (tag, word-1, -1), (tag, word-2, -2)
3. Next words: word1 word2 – (tag, word1, 1), (tag, word2, 2)
4. Tag of previous word – (tag, prev_tag)
5. Prefix and suffix of current word (first, last 3 characters) – (tag, prefix, 'p'), (tag, suffix, 's')
6. Bias – (tag, 'BIAS')

Code Structure

data_structures.py turn each sentence to the class Sentence, use *get_em_features()*, *get_all_features()* to get emission features and all features (include transition)
perceptron_pos_tagger.py use *train()* to train perceptron algorithm, *train_avg_perceptron()* train average perceptron algorithm
train_test_tagger.py train perceptron algorithm on train set and output the predictions of dev/test set
scorer.py compute accuracy of predicted files

How to run

train perceptron algorithm python train_test_tagger.py train/ptb_02-21.tagged dev/ptb_22.tagged dev/ptb_22.snt test/ptb_23.snt
get accuracy of dev set python scorer.py dev/ptb_22.tagged dev/pred.tagged
get accuracy of test set python scorer.py test/ptb_23.tagged test/pred.tagged

Experiments

1. **perceptron vs average perceptron**

	Perceptron	Average perceptron
Dev accuracy	0.9084	0.9510
Run time	3h	15h

*trained one iteration

It takes perceptron 3 hours to run and average perceptron 15 hours to run. The reason might be we use dictionary to store the weights of features, we will traversal all elements in the dictionary during caching intermedia weights, which costs a long time. The accuracy on dev set is higher for average perceptron algorithm. However, considering of tremendous time it takes, we decide not to use average perceptron in further experiments.

For the limitation of time, we just run the training process for 1 iteration.
Features:
Current word; previous two words; next two words; tag of previous word; prefix and suffix of current word; bias

2. Different Features

Features available

1) Current word: word0
2) Previous words: word-1, word-2
3) Next words: word1 word2
4) Tag of previous word
5) Prefix and suffix of current word (first, last 3 characters)
6) Bias

	1+2+3+4+5+6	1+2+3+4+6	1+3+4+6	1+2+4+6	1+4+6
Dev accuracy	0.9367	0.9084	0.9029	0.9001	0.8871

*trained one iteration

We can tell from the results that the more features we use, the higher accuracy we can get. Predix and suffix of current word is a strong feature.

Insights

1. Average perceptron algorithm performs better than perceptron in terms of accuracy. The reason might be during online training, perceptron is more likely to be influenced by noises. Some unusual tags might change the global feature weights, which might misleading the tagging of other sequences. And average perceptron takes average of all weights during training which avoid overfitting. However, the running time is much longer for updating and caching weights during training. If we might be able to optimize by using sparse matrix instead.
2. Features
Generally, the most feature we have, the higher accuracy we can get. The prefix and suffix feature are the most important which resulting in 3% increase in accuracy. As for next two words and previous two words, they do contribute though not big.
3. The experiment is constrained by the running time, we might be able to optimize the code using numpy matrix to represent the features. We guess the accuracy might reach 95 – 96 if we can iterator two more times.