CS505 HW4 Writeup
Yuhao He

PART I Constrained Method

Instead of just having word itself as a feature, I add more attributes on features list like the part of speech, suffix, prefix, word shape et al based on typical feature for a feature-based NER system in our textbook. You can find more detail in partI.py line 46 ~ line 63.

identity of w_i, identity of neighboring words
embeddings for w_i, embeddings for neighboring words
part of speech of w_i, part of speech of neighboring words
base-phrase syntactic chunk label of w_i and neighboring words
presence of w_i in a **gazetteer**
w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
w_i is all upper case
word shape of w_i, word shape of neighboring words
short word shape of w_i, short word shape of neighboring words
presence of hyphen

For the graph below I run all the features with the initial model provided in original zip package.

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results.txt
processed 52923 tokens with 4351 phrases; found: 5019 phrases; correct: 2849.
accuracy:  95.05%; precision:  56.76%; recall:  65.48%; FB1:  60.81
              LOC: precision:  52.45%; recall:  76.12%; FB1:  62.11  1428
             MISC: precision:  28.68%; recall:  35.51%; FB1:  31.73  551
              ORG: precision:  57.19%; recall:  62.71%; FB1:  59.82  1864
              PER: precision:  74.49%; recall:  71.69%; FB1:  73.06  1176
```

After playing with several models to find the best fit, Logistic Regression have the best performance. I have tried –MLPClassifier, LinearSVC, Logistic Regression and PassiveAggressiveClassifier. The graph below is the score for all features with Logistic Regression.

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results.txt
processed 52923 tokens with 4351 phrases; found: 4889 phrases; correct: 2953.
accuracy:  95.43%; precision:  60.40%; recall:  67.87%; FB1:  63.92
              LOC: precision:  56.01%; recall:  77.13%; FB1:  64.90  1355
             MISC: precision:  30.23%; recall:  35.73%; FB1:  32.75  526
              ORG: precision:  60.47%; recall:  65.06%; FB1:  62.68  1829
              PER: precision:  78.80%; recall:  76.02%; FB1:  77.38  1179
```

Increase the window size to [-2, 2] around the token

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results.txt
processed 52923 tokens with 4351 phrases; found: 4695 phrases; correct: 3082.
accuracy:  95.90%; precision:  65.64%; recall:  70.83%; FB1:  68.14
              LOC: precision:  60.66%; recall:  78.35%; FB1:  68.38  1271
             MISC: precision:  36.48%; recall:  40.00%; FB1:  38.16  488
              ORG: precision:  66.91%; recall:  68.76%; FB1:  67.83  1747
              PER: precision:  81.08%; recall:  78.89%; FB1:  79.97  1189
```

Delete suffix and prefix 1, 2 & 4.

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results.txt
processed 52923 tokens with 4351 phrases; found: 4668 phrases; correct: 3082.
accuracy:  95.90%; precision:  66.02%; recall:  70.83%; FB1:  68.34
              LOC: precision:  61.44%; recall:  79.67%; FB1:  69.38  1276
             MISC: precision:  37.42%; recall:  40.45%; FB1:  38.88  481
              ORG: precision:  66.92%; recall:  68.41%; FB1:  67.66  1738
              PER: precision:  81.42%; recall:  78.15%; FB1:  79.75  1173
```

Combine train_sents and dev_sents dataset and make a prediction on test_sents data, which return the highest FB1 score I got so far.

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results.txt
processed 51533 tokens with 3558 phrases; found: 3896 phrases; correct: 2668.
accuracy:  96.98%; precision:  68.48%; recall:  74.99%; FB1:  71.59
              LOC: precision:  75.74%; recall:  73.15%; FB1:  74.43  1047
             MISC: precision:  36.39%; recall:  41.00%; FB1:  38.56  382
              ORG: precision:  66.08%; recall:  77.36%; FB1:  71.27  1639
              PER: precision:  78.86%; recall:  88.84%; FB1:  83.56  828
```

PART III

I use SGDClassifier in this part, as I think that the loss function would gave us a better result than the logistic regression. As we can see its documentation below that it has a bunch of loss algorithm provided to us.

loss : str, 'hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron', or a regression loss: 'squared_loss', 'huber', 'epsilon_insensitive', or 'squared_epsilon_insensitive'
The loss function to be used. Defaults to 'hinge', which gives a linear SVM. The 'log' loss gives logistic regression, a probabilistic classifier. 'modified_huber' is another smooth loss that brings tolerance to outliers as well as probability estimates. 'squared_hinge' is like hinge but is quadratically penalized. 'perceptron' is the linear loss used by the perceptron algorithm. The other losses are designed for regression but can be useful in classification as well; see SGDRegressor for a description.

The graph below is the best score I got, with loss = 'hinge' and feature word_shape removed

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results_partIII.txt
processed 51533 tokens with 3558 phrases; found: 3933 phrases; correct: 2677.
accuracy:  97.00%; precision:  68.07%; recall:  75.24%; FB1:  71.47
             LOC: precision:  74.16%; recall:  73.62%; FB1:  73.89  1076
            MISC: precision:  35.94%; recall:  40.71%; FB1:  38.17  384
             ORG: precision:  66.36%; recall:  77.93%; FB1:  71.68  1644
             PER: precision:  78.41%; recall:  88.44%; FB1:  83.12  829
```

I also tried other algorithm like 'squared_hinge', but it returns a pretty low score.

```
Yuhaos-MacBook-Pro:code yuhao$ python3 conlleval.py results_partIII.txt
processed 51533 tokens with 3558 phrases; found: 4079 phrases; correct: 2235.
accuracy:  95.64%; precision:  54.79%; recall:  62.82%; FB1:  58.53
             LOC: precision:  64.93%; recall:  62.18%; FB1:  63.52  1038
            MISC: precision:  15.94%; recall:  28.02%; FB1:  20.32  596
             ORG: precision:  58.48%; recall:  65.29%; FB1:  61.69  1563
             PER: precision:  62.59%; recall:  75.10%; FB1:  68.27  882
```

After several trials with the features set stay the same as part I, I decided to uncomment some feature and see what is the performance. Surprisingly, in part I, keep word shape in feature set usually returns a better result, but in SGDClassifier, remove the word shape will gave us a better score.