CS591 HW4
Ziran Min
U59274427

1. Constrained Method

To achieve a recognizer with higher F1 score, I need to create more features from words and sentences. In the original baseline code, it only has three features: the current word, the previous word, and the next word. These are not enough and only give us F1-score = 49.75% under Perceptron model.

After reading the Chapter 17.1 of the text book written by Daniel Jurafsky and James H. Martin, I was inspired to create the following features for each word (token) w_i in a sentence:

(1) Identity of w_i
(2) Part of speech of w_i
(3) The prefix of w_i with length ≤ 4
(4) The suffix of w_i with length ≤ 4
(5) Whether all letters of w_i are capitalized
(6) Whether the first letter of w_i is capitalized
(7) Whether w_i is a string of digits
(8) Whether w_i contains " - "
(9) Whether w_i contains " ' "
(10) The Spanish stem of w_i
(11) The word shape of w_i
(12) The short word shape of w_i
(13) Whether w_i is the beginning token of the sentence
(14) Whether w_i is the ending token of the sentence

For each w_i, I also capture above features (except (13) and (14)) for its neighbor tokens within the length of from -2 to 2 (the original is from -1 to 1).

After adding those features into the baseline code, the Perceptron model returns the following evaluation results:

```
processed 52923 tokens with 4351 phrases; found: 5091 phrases; correct: 2872.
accuracy:  95.10%; precision:  56.41%; recall:  66.01%; FB1:  60.83
             LOC: precision:  55.82%; recall:  70.63%; FB1:  62.36  1245
            MISC: precision:  24.12%; recall:  40.00%; FB1:  30.09  738
             ORG: precision:  56.08%; recall:  60.53%; FB1:  58.22  1835
             PER: precision:  76.20%; recall:  79.38%; FB1:  77.76  1273
```

The F1-score has increase to 60.83% on validation set which has been over 60%.

In order to reach higher score, I want to try other classification model.

In Chapter 8.5 of the text book, when introducing Maximum Entropy Markov Model (MEMM) for Part-of-Speech Tagging, the author argues that MEMM is similar to Logistic Regression and it's a kind of discriminative sequence version of Logistic Regression, so I decide to use Logistic Regression Model first (LR is easier to implement than MEMM). By including all features which I describe above, I got F1- score of 66.84% on validation set.

```
processed 52923 tokens with 4351 phrases; found: 4718 phrases; correct: 3031.
accuracy:  95.84%; precision:  64.24%; recall:  69.66%; FB1:  66.84
              LOC: precision:  59.51%; recall:  78.25%; FB1:  67.60  1294
             MISC: precision:  35.91%; recall:  40.67%; FB1:  38.15  504
              ORG: precision:  65.62%; recall:  67.35%; FB1:  66.47  1745
              PER: precision:  79.57%; recall:  76.51%; FB1:  78.01  1175
```

Considering there might be high correlation between my features that could affect the performance of Logistic Regression, I decide to exclude features of prefixes of length of (1, 2, 4), suffixes of length (1, 2, 4), and short word shape.

Now Logistic Regression gives me:

```
processed 52923 tokens with 4351 phrases; found: 4703 phrases; correct: 3084.
accuracy:  95.91%; precision:  65.58%; recall:  70.88%; FB1:  68.12
              LOC: precision:  60.78%; recall:  79.37%; FB1:  68.84  1285
             MISC: precision:  37.35%; recall:  41.80%; FB1:  39.45  498
              ORG: precision:  66.32%; recall:  68.35%; FB1:  67.32  1752
              PER: precision:  81.76%; recall:  78.15%; FB1:  79.92  1168
```

The F1-score rises to 68.12%. I will combine training set and validation set to train my Logistic model to compute final constrained output on testing set, and try to achieve higher F1 in unconstrained method.

2. Unconstrained Method

In part one, Logistics Regression has a big weakness for Named Entity Recognition: It can't take previous predicted labels as new inputs for predictions of later words. A discriminative sequence model might be a better choice. In chapter 17.1.2, we can know that Maximum Entropy Markov Model (MEMM) or Conditional Random Field (CRF) are two sequential versions of Logistics Regression. Here I decide to implement the later one, CRF.

In a CRF, each feature function is a function that takes in as input:

- a sentence *s*
- the position *i* of a word in the sentence
- the label l_i of the current word
- the label l_{i-1} of the previous word

We can score each label by:

$$score(l|s) = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1})$$

Lambdas are weights that will be trained, and the probability equation that we need to maximize is:

$$p(l|s) = \frac{exp[score(l|s)]}{\sum_{l'} exp[score(l'|s)]} = \frac{exp[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s,i,l_i,l_{i-1})]}{\sum_{l'} exp[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s,i,l'_i,l'_{i-1})]}$$

(Note: the above equation is very similar to the probability of Logistics Regression.)

I implement CRF with the help of pycrfsuite, and later find best model by parameter tuning and comparing different training methods.

First, by setting both regularized terms C1 (for L1) and C2 (for L2) to 0s, maximum iteration numbers to 100, all possible transitions to True to let CRF generate state features that associate all of possible combination between attriubtes and labels, and training algorithm to 'lbfgs'(Gradient Descent using the L-BFGS method), we get the following output with F1 equals 72.01% on validation set:

```
processed 52923 tokens with 4351 phrases; found: 4250 phrases; correct: 3097.
accuracy:  95.88%; precision:  72.87%; recall:  71.18%; FB1:  72.01
             LOC: precision:  63.92%; recall:  78.15%; FB1:  70.32  1203
            MISC: precision:  52.66%; recall:  46.74%; FB1:  49.52  395
             ORG: precision:  75.22%; recall:  70.18%; FB1:  72.61  1586
             PER: precision:  86.96%; recall:  75.86%; FB1:  81.03  1066
```

To avoid overfitting, I set C1 = 0.1 and C2 = 0.1 to use Elastics Net Regularization. Without changing other parameters, I got F1 = 74.20% and output:

```
processed 52923 tokens with 4351 phrases; found: 4229 phrases; correct: 3183.
accuracy:  96.07%; precision:  75.27%; recall:  73.16%; FB1:  74.20
             LOC: precision:  64.80%; recall:  78.96%; FB1:  71.19  1199
            MISC: precision:  57.99%; recall:  48.09%; FB1:  52.58  369
             ORG: precision:  78.44%; recall:  72.35%; FB1:  75.28  1568
             PER: precision:  88.01%; recall:  78.72%; FB1:  83.11  1093
```

I tried to write nested loops for giving different values to C1 and C2 to find the combination can give me the highest F1-score, but I always meet running time error. Therefore, I will keep C1 = 0.1 and C2 = 0.1 for later models.

Now if I change the max_iteration to 1000, I got:

```
processed 52923 tokens with 4351 phrases; found: 4192 phrases; correct: 3194.
accuracy:  96.19%; precision:  76.19%; recall:  73.41%; FB1:  74.77
             LOC: precision:  65.63%; recall:  79.17%; FB1:  71.76  1187
            MISC: precision:  61.03%; recall:  45.39%; FB1:  52.06  331
             ORG: precision:  78.55%; recall:  73.24%; FB1:  75.80  1585
             PER: precision:  88.89%; recall:  79.21%; FB1:  83.77  1089
```

The F1-score only increased a little bit to 74.77%, but actually the running time is much longer than that of max_iteration = 100, so I will keep max_iteration = 100.

Now I want to compare how different training methods affects the F-1 score. Setting C1 = 0.1 and C2 = 0.1, max_iteration = 100, and feature.possible_transitions = True:

1. if training algorithm is “lbfgs” (Gradient Descent using L-BFGS method), F1-score = 74.20%

processed 52923 tokens with 4351 phrases; found: 4229 phrases; correct: 3183.
accuracy: 96.07%; precision: 75.27%; recall: 73.16%; FB1: 74.20
LOC: precision: 64.80%; recall: 78.96%; FB1: 71.19 1199
MISC: precision: 57.99%; recall: 48.09%; FB1: 52.58 369
ORG: precision: 78.44%; recall: 72.35%; FB1: 75.28 1568
PER: precision: 88.01%; recall: 78.72%; FB1: 83.11 1093

2. if training algorithm is “l2sgd” (Stochastics Gradient Descent with L2 regularization term), F1-score = 74.34%

processed 52923 tokens with 4351 phrases; found: 4204 phrases; correct: 3180.
accuracy: 96.18%; precision: 75.64%; recall: 73.09%; FB1: 74.34
LOC: precision: 64.19%; recall: 79.98%; FB1: 71.22 1226
MISC: precision: 61.09%; recall: 45.17%; FB1: 51.94 329
ORG: precision: 78.38%; recall: 72.71%; FB1: 75.43 1577
PER: precision: 89.18%; recall: 78.23%; FB1: 83.35 1072

3. if training algorithm is “ap” (Averaged Perceptron), F1-score = 73.15%

processed 52923 tokens with 4351 phrases; found: 4253 phrases; correct: 3147.
accuracy: 96.06%; precision: 73.99%; recall: 72.33%; FB1: 73.15
LOC: precision: 65.71%; recall: 78.86%; FB1: 71.69 1181
MISC: precision: 56.31%; recall: 50.11%; FB1: 53.03 396
ORG: precision: 76.28%; recall: 70.94%; FB1: 73.51 1581
PER: precision: 86.03%; recall: 77.09%; FB1: 81.31 1095

4. if training algorithm is “pa” (Passive Aggressive), F1-score = 72.89%

processed 52923 tokens with 4351 phrases; found: 4281 phrases; correct: 3146.
accuracy: 95.99%; precision: 73.49%; recall: 72.31%; FB1: 72.89
LOC: precision: 64.83%; recall: 79.07%; FB1: 71.25 1200
MISC: precision: 56.31%; recall: 50.11%; FB1: 53.03 396
ORG: precision: 75.30%; recall: 70.47%; FB1: 72.80 1591
PER: precision: 86.56%; recall: 77.50%; FB1: 81.78 1094

5. if training algorithm is “arow” (Adaptive Regularization of Weights Vector), F1-score = 61.84%

```
processed 52923 tokens with 4351 phrases; found: 4841 phrases; correct: 2842.
accuracy:  94.78%; precision:  58.71%; recall:  65.32%; FB1:  61.84
             LOC: precision:  57.10%; recall:  73.58%; FB1:  64.30  1268
            MISC: precision:  28.19%; recall:  42.25%; FB1:  33.81  667
             ORG: precision:  61.13%; recall:  63.47%; FB1:  62.28  1765
             PER: precision:  74.58%; recall:  69.64%; FB1:  72.03  1141
```

Therefore, the second training methods (Stochastics Gradient Descent) gives me the highest F1-score.

Finally, by setting C1 = 0.1 and C2 = 0.1, max_iteration = 100, and feature.possible_transitions = True, algorithm = “l2sgd”, and training CRF on combined training and validation data, I could get my final unconstrained output on testing set.

For future, I will try to find better parameters (like C1 and C2) for CRF, and try to implement MEMM algorithm and compare it with CRF.