Home Work 3
Structured Learning with Feature-Rich Models

1 Variation on the CKY Algorithm

1.1 Definition

After added the new rules, the Chomsky Normal Form will allow the following 3 rule types:

- $X \to V\ Y\ Z$
- $X \to Y\ Z$
- $X \to w$

Where V, X, Y, Z are nonterminal symbols and w is a terminal symbol.

1.2 Dynamic Programming

The idea will be similar to the original CKY algorithm, that:

- We will store the score of the ma parse of x_i to x_i with root non-terminal X

$$\pi(i, j, X)$$

- So, we can compute the most likely parse:

$$\pi(1, n, S) = \max_{t \in T_G(s)} p(t)$$

- Via the recursion:

$$\pi(i,j,X) = \max\left(\begin{array}{l} \max\limits_{\substack{X \to YZ \in R,\\ s \in \{i \ldots (j-1)\}}} (q(X \to YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z) \\ \max\limits_{\substack{X \to VYZ \in R,\\ s,t \in \{i \ldots (j-1)\}, s<t}} (q(X \to VYZ) \times \pi(i,s,V) \times \pi(s+1,t,Y) \times \pi(t+1,j,Z) \end{array}\right)$$

- With base case:

$$\pi(i,i,X) = \begin{cases} q(X \to x_i), & if\ X \to x_i \in R \\ 0, & otherwise \end{cases}$$

1.3 Pseudo-code

The pseudo-code will also like the original version, that:

- Input will be a sentence $S = x_1 \ldots x_n$ and a $PCFG = < N, \Sigma, S, R, q >$
- Initialization: For $i = 1 \ldots n$ and all $X \in N$

$$\pi(i,i,X) = \begin{cases} q(X \to x_i), & if\ X \to x_i \in R \\ 0, & otherwise \end{cases}$$

- For $l = 1 \dots (n-1)$
  - For $i = 1 \dots (n-l)$ and $j = i + 1$
    - For all $X \in N$
      - $\pi(i,j,X) = \max\left(\begin{matrix} \max\limits_{\substack{X \to YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \to YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z) \\ \max\limits_{\substack{X \to VYZ \in R, \\ s,t \in \{i \dots (j-1)\}, s<t}} (q(X \to VYZ) \times \pi(i,s,V) \times \pi(s+1,t,Y) \times \pi(t+1,j,Z) \end{matrix} \right)$
    - Also, store the back pointers
      - $bp(i,j,X) =$ $arg\max\left(\begin{matrix} \operatorname{argmax}\limits_{\substack{X \to YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \to YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z) \\ \operatorname{argmax}\limits_{\substack{X \to VYZ \in R, \\ s,t \in \{i \dots (j-1)\}, s<t}} (q(X \to VYZ) \times \pi(i,s,V) \times \pi(s+1,t,Y) \times \pi(t+1,j,Z) \end{matrix} \right)$

2 Named-Entity Recognition (NER) with Structured Perceptron

2.1 Preprocess

In the implementation, chosen features from current input, current tag and previous tag. In order to support the previous tag, add the Start, End token and their related tags for the inputs.

2.2 Feature Selection

For features only used current data:

y_k	(w_k, y_k)	$(Lower(w_k), y_k)$	$(IsUpper(w_k), y_k)$
$(Prefix(w_k), y_k)$	$(Suffix(w_k), y_k)$	$(Shape(w_k), y_k)$	$(Contain(w_k, -), y_k)$
(POS_Tag_k, y_k)	$(Chunk\_Tag_k, y_k)$	$(POS\_Tag_k, Chunk\_Tag_k, y_k)$	

And here is the explanation for those symbols:

- w_k Current word token
- y_k Current NER tag
- $Lower(w_k)$ The lowercase of current work token
- $IsUpper(w_k)$ Whether current word token is all uppercase
- $Prefix(w_k)$ First 3 letters of current word token
- $Suffix(w_k)$ Last 3 letters of current word token
- $Shape(w_k)$ The shape of current word token
  - Will only keep information about word token's upper case, lower case, digital and special character
- $Contain(w_k, -)$ Whether current word token contain dash
- POS_Tag_k Current POS tag
- $Chunk\_Tag_k$ Current syntactic chunk tag

For features used previous tag:

(y_k, y_{k-1})	(w_k, y_k, y_{k-1})
$(Prefix(w_k), y_k, y_{k-1})$	$(Suffix(w_k), y_k, y_{k-1})$

2.3 Feature Improvement in Future

For later improvement, we can add more features based on current design, such as:

- More Bigram features
- Use previous input token (previous word token, previous POS tag, previous syntactic chunk tag
- Use trigram features

2.3.1 More Bigram Features

We can add more Bigram features, such as:

$(Shape(w_k), y_k, y_{k-1})$	$(Contain(w_k, -), y_k, y_{k-1})$
$(POS\_Tag_k, y_k, y_{k-1})$	$(Chunk\_Tag_k, y_k, y_{k-1})$
$(POS\_Tag_k, Chunk\_Tag_k, y_k, y_{k-1})$	

2.3.2 Use Previous Input Token

We can use previous input token to generate more features, such as:

$(w_k, w_{k-1}, y_k, y_{k-1})$	(w_k, w_{k-1}, y_k)
$(Prefix(w_k), y_k, y_{k-1})$	$(Suffix(w_k), y_k, y_{k-1})$
$(POS\_Tag_{k-1}, y_k, y_{k-1})$	$(Chunk\_Tag_{k-1}, y_k, y_{k-1})$
$(POS\_Tag_{k-1}, Chunk\_Tag_{k-1}, y_k, y_{k-1})$	

2.3.3 Use Trigram Features

We can use 2 back NER tag to generate the Trigram features, such as:

$(w_k, y_k, y_{k-1}, y_{k-2})$	$(Suffix(w_k), y_k, y_{k-1}, y_{k-2})$
$(Prefix(w_k), y_k, y_{k-1}, y_{k-2})$	$(Chunk\_Tag_k, y_k, y_{k-1}, y_{k-2})$
$(POS\_Tag_k, y_k, y_{k-1}, y_{k-2})$	$(POS\_Tag_k, Chunk\_Tag_k, y_k, y_{k-1}, y_{k-2})$

2.4 Viterbi Algorithm

2.4.1 Feature Dependency

From the definition of the features we used, we can say that our features are dependence on:

- Current inputs
  - Word token
  - POS tag
  - Syntactic chunk tag
- Current NER tag
- Previous NER tag

2.4.2 Decoding Logic

And the decoding logic is just a simple Bigram Viterbi algorithm:

2.4.2.1 Dynamic program for computing (for all i)

$$\pi(i, y_i) = \max_{y_i \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

2.4.2.2 Iterative computation

Base case, for $i = 0$:

$$\pi(0, y_0, y_{-1}) = \begin{cases} 1, & if\ y_{-1} == START \\ 0, & otherwise \end{cases}$$

Normal Case, for $i = 1 \dots n$:

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_i - 1)$$

2.4.2.3 Back Pointers

$$bp(i, y_i) = \underset{y_{i-1}}{\operatorname{argmax}}\, e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

2.4.2.4 Final Solution

$$y^* = \underset{y_1 \dots y_n}{\operatorname{argmax}}\, p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

3 Running Results

3.1 Reduced Data Set

On the testing on the reduced data set, we can find that:

Iteration	1	5	10	15	20	50
Dev Set Accuracy	94.50%	96.40%	96.92%	96.62%	96.79%	96.73%

Iteration	75	100	150	200	250
Dev Set Accuracy	96.95%	96.95%	96.62%	96.39%	96.73%

So, when the iteration is 75 or 100, the *Perceptron Model* will performance better on **Dev Set** (with 96.95% accuracy). The evaluation report on **Dev Set** are:

- Iteration as 75

```
processed 7315 tokens with 813 phrases; found: 804 phrases; correct: 661.
accuracy:  96.95%; precision:  82.21%; recall:  81.30%; FB1:  81.76
              LOC: precision:  86.21%; recall:  88.24%; FB1:  87.21  261
             MISC: precision:  80.00%; recall:  79.28%; FB1:  79.64  110
              ORG: precision:  78.26%; recall:  65.63%; FB1:  71.39  161
              PER: precision:  81.62%; recall:  87.06%; FB1:  84.25  272
```

- Iteration as 100

```
processed 7315 tokens with 813 phrases; found: 804 phrases; correct: 664.
accuracy:  96.95%; precision:  82.59%; recall:  81.67%; FB1:  82.13
              LOC: precision:  87.21%; recall:  88.24%; FB1:  87.72  258
             MISC: precision:  80.73%; recall:  79.28%; FB1:  80.00  109
              ORG: precision:  74.85%; recall:  66.67%; FB1:  70.52  171
              PER: precision:  83.83%; recall:  87.45%; FB1:  85.60  266
```

And when we run it against **Test Set**, the evaluation report is:

- Iteration as 75

```
processed 6255 tokens with 790 phrases; found: 783 phrases; correct: 557.
accuracy:  93.86%; precision:  71.14%; recall:  70.51%; FB1:  70.82
              LOC: precision:  79.55%; recall:  85.02%; FB1:  82.19  264
             MISC: precision:  64.08%; recall:  60.55%; FB1:  62.26  103
              ORG: precision:  64.71%; recall:  51.27%; FB1:  57.21  187
              PER: precision:  69.87%; recall:  80.81%; FB1:  74.94  229
```

- Iteration as 100

```
processed 6255 tokens with 790 phrases; found: 780 phrases; correct: 553.
accuracy:  93.84%; precision:  70.90%; recall:  70.00%; FB1:  70.45
             LOC: precision:  80.16%; recall:  83.40%; FB1:  81.75  257
            MISC: precision:  66.34%; recall:  61.47%; FB1:  63.81  101
             ORG: precision:  61.93%; recall:  51.69%; FB1:  56.35  197
             PER: precision:  70.22%; recall:  79.80%; FB1:  74.70  225
```

For simple output later, the following test will only use iteration as 75.

3.2 Full Data Set

This implementation will throw out a memory update error for the full data set.

4 Ablation Study:

4.1 Design

We can have 4 different ablations for the study of our features:

- Without $Shape(w_k)$
- Without Bigram Features
- Without POS_Tag_k and $Chunk\_Tag_k$
- Without word token status check
  - $Prefix(w_k)$
  - $Suffix(w_k)$
  - $Lower(w_k)$
  - $IsUpper(w_k)$

4.2 Result

With the same integration as previous full model, we can get the evaluation report for 4 ablations:

4.2.1 Without $Shape(w_k)$

On **Dev Set** is:

```
processed 7315 tokens with 813 phrases; found: 806 phrases; correct: 647.
accuracy:  96.50%; precision:  80.27%; recall:  79.58%; FB1:  79.93
             LOC: precision:  86.33%; recall:  86.67%; FB1:  86.50  256
            MISC: precision:  78.18%; recall:  77.48%; FB1:  77.83  110
             ORG: precision:  72.53%; recall:  68.75%; FB1:  70.59  182
             PER: precision:  80.62%; recall:  81.57%; FB1:  81.09  258
```

On **Test Set** is:

```
processed 6255 tokens with 790 phrases; found: 788 phrases; correct: 544.
accuracy:  93.33%; precision:  69.04%; recall:  68.86%; FB1:  68.95
             LOC: precision:  77.65%; recall:  83.00%; FB1:  80.23  264
            MISC: precision:  67.71%; recall:  59.63%; FB1:  63.41  96
             ORG: precision:  62.87%; recall:  53.81%; FB1:  57.99  202
             PER: precision:  65.04%; recall:  74.24%; FB1:  69.34  226
```

4.2.2 Without Bigram Features

On **Dev Set** is:

```
processed 7315 tokens with 813 phrases; found: 910 phrases; correct: 590.
accuracy:  95.63%; precision:  64.84%; recall:  72.57%; FB1:  68.49
             LOC: precision:  75.95%; recall:  78.04%; FB1:  76.98  262
            MISC: precision:  66.12%; recall:  72.07%; FB1:  68.97  121
             ORG: precision:  48.37%; recall:  54.17%; FB1:  51.11  215
             PER: precision:  66.35%; recall:  81.18%; FB1:  73.02  312
```

On **Test Set** is:

```
processed 6255 tokens with 790 phrases; found: 922 phrases; correct: 481.
accuracy:  92.61%; precision:  52.17%; recall:  60.89%; FB1:  56.19
             LOC: precision:  67.87%; recall:  76.11%; FB1:  71.76  277
            MISC: precision:  58.77%; recall:  61.47%; FB1:  60.09  114
             ORG: precision:  40.28%; recall:  36.02%; FB1:  38.03  211
             PER: precision:  44.06%; recall:  71.21%; FB1:  54.44  320
```

4.2.3 Without POS_Tag_k and $Chunk\_Tag_k$

On **Dev Set** is:

```
processed 7315 tokens with 813 phrases; found: 800 phrases; correct: 651.
accuracy:  96.66%; precision:  81.38%; recall:  80.07%; FB1:  80.72
             LOC: precision:  82.06%; recall:  84.31%; FB1:  83.17  262
            MISC: precision:  78.85%; recall:  73.87%; FB1:  76.28  104
             ORG: precision:  77.98%; recall:  68.23%; FB1:  72.78  168
             PER: precision:  83.83%; recall:  87.45%; FB1:  85.60  266
```

On **Test Set** is:

```
processed 6255 tokens with 790 phrases; found: 770 phrases; correct: 552.
accuracy:  93.80%; precision:  71.69%; recall:  69.87%; FB1:  70.77
             LOC: precision:  78.63%; recall:  83.40%; FB1:  80.94  262
            MISC: precision:  70.71%; recall:  64.22%; FB1:  67.31  99
             ORG: precision:  64.13%; recall:  50.00%; FB1:  56.19  184
             PER: precision:  70.22%; recall:  79.80%; FB1:  74.70  225
```

4.2.4 Without word token status check

On **Dev Set** is:

```
processed 7315 tokens with 813 phrases; found: 778 phrases; correct: 615.
accuracy:  96.21%; precision:  79.05%; recall:  75.65%; FB1:  77.31
             LOC: precision:  80.17%; recall:  72.94%; FB1:  76.39  232
            MISC: precision:  86.17%; recall:  72.97%; FB1:  79.02  94
             ORG: precision:  74.71%; recall:  66.15%; FB1:  70.17  170
             PER: precision:  78.37%; recall:  86.67%; FB1:  82.31  282
```

On **Test Set** is:

```
processed 6255 tokens with 790 phrases; found: 764 phrases; correct: 531.
accuracy:  93.40%; precision:  69.50%; recall:  67.22%; FB1:  68.34
             LOC: precision:  80.26%; recall:  75.71%; FB1:  77.92  233
            MISC: precision:  71.95%; recall:  54.13%; FB1:  61.78  82
             ORG: precision:  62.07%; recall:  53.39%; FB1:  57.40  203
             PER: precision:  64.63%; recall:  80.30%; FB1:  71.62  246
```

4.3 Analysis

By checking the ablations, we can find that:

4.3.1 Without $Shape(w_k)$

Without the $Shape(w_k)$ features will reduce the accuracy from 96.95% to 96.5% on **Dev Set**, and the overall precision is down from 82.21% to 80.27%. From the subset evaluation we can find that the shape of the word token will affect about 2% precision and 2% recall on the MISC, ORG, and PER tags.

4.3.2 Without Bigram Features

Without the Bigram features will reduce the accuracy from 96.95% to 95.63% on **Dev Set**, and the overall precision is down from 82.21% to 64.84%. From the subset evaluation we can find that the Bigram features will affect about 15% precision and 6%-10% recall on all NER tags.

4.3.3 Without POS_Tag_k and $Chunk\_Tag_k$

Without the POS_Tag_k and $Chunk\_Tag_k$ features will reduce the accuracy from 96.95% to 96.66% on **Dev Set**, and the overall precision is down from 82.21% to 81.38%. From the subset evaluation we can find that the POS_Tag_k and $Chunk\_Tag_k$ features will affect about 4% precision and 4% recall on LOC and MISC tags; and with these features, our model has down affect for PER tag with about 2% precision and 2% recall.

4.3.4 Without word token status check

Without the word token status check features will reduce the accuracy from 96.95% to 96.21% on **Dev Set**, and the overall precision is down from 82.21% to 79.05%. From the subset evaluation we can find that the POS_Tag_k and $Chunk\_Tag_k$ features will affect about 4% precision and 4% recall on LOC, ORG and PER tags; and for MISC tag this features set will down affect for about 6% precision and affect 6% recall. So, we can say these features just make our model more average for MISC tag.

4.4 Conclusion

The Bigram features is crucial for our model, without these features our model will performance very poorly on many cases.

And for other 3 features sets, we can say all of them can help us to build a better model, but not significant improvement like the Bigram features, some of them even will hurt the prediction on certain NER tag, but with these features will definitely performance better than without them.

5 Compare BIO and IO encoding schemes

5.1 BIO Encoding Schema

5.1.1 Pros

By using more tags type for the tagging can make:

- Boundaries would be implicit.
- Can also boost the ML algorithm performance
- The additional tags provide more useful information post-classification for interpreting problematic edge cases.

5.1.2 Cons

And it can also cause:

- Predict I-tag poorly when most of the named entities only have one word in the training set
- Should control the transition from B-tag to I-tag for different entities in HMM

5.2 IO Encoding Schema

5.2.1 Pros

- Saving the tag type when no two entities with same type next to each other
- Less complexity on decoding

5.2.2 Cons

- Can't represent two entities with same type next to each other, since there's no boundary tag

5.3 Conclusion

So, the IO encoding schema will performance better, when:

- No two entities with same type next to each other in the dataset
- No need to use the boundaries to do post-classification work