CSEP 517: Homework 3 Structured Learning with Feature-Rich Models

Question 1 - Variations on the Cocke–Younger–Kasami (CKY) Algorithm

The current CKY algorithm assumes that the grammar is in CNF which means that the grammar rules follow the structure X -> Y Z. This means that the rules have at most 2 children. This results in the following algorithm/pseudocode:

```
▪ Input: a sentence s = $x_1$ .. $x_n$ and a PCFG = <N, Σ ,S, R, q>
▪ Initialization: For i = 1 … n and all X in N
      $\pi(i,i,X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$
▪ For l = 1 … (n-1)                         [iterate all phrase lengths]
    ▪ For i = 1 … (n-l) and j = i+l         [iterate all phrases of length l]
        ▪ For all X in N                    [iterate all non-terminals]
              $\pi(i,j,X) = \max_{X \rightarrow YZ \in R, s \in \{i \ldots (j-1)\}} (q(X \rightarrow YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z))$
        ▪ also, store back pointers
              $bp(i,j,X) = \arg\max_{X \rightarrow YZ \in R, s \in \{i \ldots (j-1)\}} (q(X \rightarrow YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z))$
```

Now to edit this so that we are in 3-CNF means dealing with the extra rule of X -> V Y Z. This rule will be implemented in a similar way but instead of 1 split point we now have 2 split points. If the sentence had 4 parts (x1,x2,x3,x4), instead of having the possible splits be (x1) (x2 x3 x4), (x1 x2) (x3 x4), and (x1 x2 x3) (x4). We also now have (x1) (x2) (x3 x4), (x1) (x2 x3) (x4), (x1 x2) (x3) (x4). We will keep everything we have seen the same, except for the max over 1 split point for the rule X -> Y Z. We will have another loop to max over the rule X -> V Y Z for 2 split points.

$$\max_{\substack{X \rightarrow V\,Y\,Z \in R \\ s \in \{i \ldots (j-1)\} \\ s2 \in (s+1 \ldots (j-1))}} (q(X \rightarrow V\,Y\,Z) \times \pi(\mathrm{i, s, V}) \times \pi(\mathrm{s+1, s2, Y}) \times \pi(\mathrm{s2+1, j, Z}))$$

This gives us the second set of options from the above list. We would then take the max of the 2-cnf π equation and the equation i provided. The back pointer will be the argmax of the same equation. We would store the appropriate back pointer depending on which max we took. Everything else in the algorithm would stay the same. Technically i think we can have just 1 max if I start s2 at s and not at s+1. It made things clearer to split it for me so that we can see how we get all the different combinations of splits and from which equation they come.

Question 2 - Named-Entity Recognition (NER) with Structured Perceptrons

<u>Features and Feature Templates:</u>

- **Features used in my code**

1. Bigram features class
    a. This was total of #tags*#tags features encompassing every possible bigram from the possible seen NER tags.
    b. Included a set of tag_stopWord features
    c. Included a set of startWord_tag features
2. Word shape features class
    a. Features that check if first letter of the word is uppercase for the tag.
    b. Features that check if the whole word is uppercase for the tag.
3. Current tag based feature
    a. Just the current tag feature
4. Word feature class
    a. For each possible tag/word pair there was a feature.
    b. The extremely expanded our feature space

- **Additional Possible Features**

1. Pos tag based features
    a. Could use the pos tag for each word/ner tag
    b. Could also do pos tag bigrams
2. Chunk tag features
    a. Use the chunk tag for each word/ner tag
    b. Could do chunk tag bigram features
3. More word shape features
    a. Prefix and suffix based features
    b. Numerals or special characters in the word (such as -)
4. Positional features
    a. Feature for first letter is capital when it isn't the first word of the sentence

Factoring and dependencies:

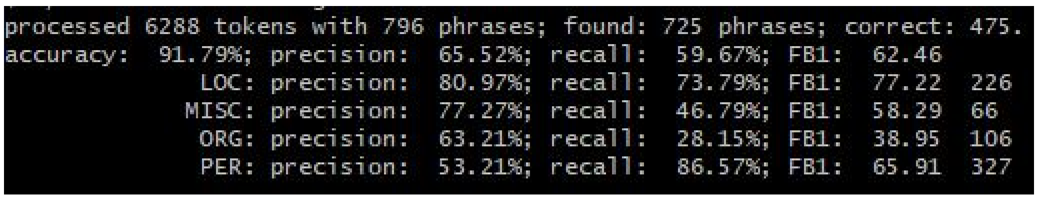
To properly and efficiently determine the tags for the whole sentence we have to divide our task into subparts. We will therefore factor the feature vector into smaller feature vectors that determine the features for a given word. We will at the end sum up these smaller feature vectors to get our main feature vector back. This split results in the equation $w \cdot \Phi(x_1 \ldots x_n, s_1 \ldots s_n) = \sum_{j=1}^{n} w \cdot \phi(x, j, s_{j-1}, s_j)$ where w is the weight vector, x1….xn are the words in the sentence, and s1….sn are the NER tags for those words. The question then becomes how to determine the tags to give to the feature vector function as input since each word now has a dependency on the previous words tag. The previous tag is also reliant on its previous tag and so this reminds us of our handy dandy Viterbi algorithm! In the viterbi algorithm we start with the start word/tag and then determine the best score for each possible next tag. The max score can be defined by the equation $\pi(i, s_i) = \max_{s_1 \ldots s_{i-1}} w \cdot \Phi(x_1 \ldots x_i, s_1 \ldots s_i)$. Since its a max we can factor out the last value and split this equation into two maxes, one for the current word/tag and one that spans across all previous word/tags. This results in the max score $\pi(i, s_i) = \max_{s_{i-1}} w \cdot \phi(x, i, s_{i-1}, s_i) + \max_{s_1 \ldots s_{i-2}} w \cdot \Phi(x_1 \ldots x_{i-1}, s_1 \ldots s_{i-1})$. The first part comes from the fact that the large feature vector is the sum of all the smaller word vectors, so factoring out one word equates to factoring out one words feature vector. The second half of the equation is now looking like the recursion we are used to in viterbi, $\pi(i-1, s_{i-1})$. This results in the final equation being $\max_{s_{i-1}} w \cdot \phi(x, i, s_{i-1}, s_i) + \pi(i-1, s_{i-1})$. This allows for easy decoding without exponential time. In practice this becomes, calculate the score of the current tag(feature vector given the tag * weight vector) and add it to the best possible previous score.

Results and analysis:

To determine the number of iterations I ran multiple tests until there was little to gain from more iterations given the amount of time it took. To do this I trained with the training set and evaluated on the dev set. The numbers are as follows:

T	1	10	20	30
F1	52.18	68.01	70.28	70.94
Accuracy	91.01	94.37	94.67	94.88
Time Taken	00:20.44	02:17.35	03:25.40	04:47.47

As you can see going from 20 to 30 iterations gave very little yield. The data can be found in the files eng.dev.small.result.t1 through eng.dev.small.result.t30. Using 20 iterations the result on the test set is:



```
processed 6288 tokens with 796 phrases; found: 725 phrases; correct: 475.
accuracy:  91.79%; precision:  65.52%; recall:  59.67%; FB1:  62.46
              LOC: precision:  80.97%; recall:  73.79%; FB1:  77.22  226
             MISC: precision:  77.27%; recall:  46.79%; FB1:  58.29  66
              ORG: precision:  63.21%; recall:  28.15%; FB1:  38.95  106
              PER: precision:  53.21%; recall:  86.57%; FB1:  65.91  327
```

This data can be seen in the file eng.test.small.result.

Analysis on dev set:

1) I-ORG and I-LOC confusion
    a) There are many instances where an organization’s name is the name of the location. Example query “Rotor Volgograd must play their next home game behind closed doors after fans hurled bottles and stones at Dynamo Moscow players during a 1-0 home defeat on Saturday that ended Rotor 's brief spell as league leaders.” In this example we have confusion on Moscow since that is a location but in this sentence it is referencing a soccer team.
    b) This confusion was seen 47 times in the dev set
2) I-ORG and I-PER confusion
    a) Since most organization names are unique entities they have a lot of similarities to names (unique). This leads to confusion where a new unique name doesn't have many features to define if name or organization. Finding the features that would differentiate between them is also hard. Example sentence: “Essex and Kent both face tense finishes on Monday as they attempt to keep pace with title hopefuls Derbyshire and Surrey ,convincing three-day victors on Saturday , in the English county championship run-in.” We confuse the word Derbyshire. This is probably

due to the rarity of the word and how its surrounded with O tags. Names are usually 1 word and therefore this makes sense.

b) This confusion was found 57 times in the dev set

3) I-LOC and O confusion

a) There seems to be a lot of cases of clear mistakes in tagging where the word should clearly be LOC but isnt tagged at all. In this example query "SOCCER - IRELAND BEAT LIECHTENSTEIN 5-0 IN WORLD CUP QUALIFIER ." we confuse both ireland and Liechtenstein as O instead of tagging as a location. One thought is that its due to the words being in all caps, confusing with other words since locations are usually not all caps. Another thing could be due to "beating" not being a tell of countries since countries usually don't beat each other.

b) This confusion was found 42 times in the dev set

Ablation Study:

In this ablation study we will look at 4 different sets of the possible features and how they compare with the full set of all the features.

1. Uppercase word
2. First letter is upper case
3. Current tag
4. start/stop bigrams

Running with 20 iterations as determined by previous experiments

Feature set	Dev Accuracy	Dev F1	Test Accuracy	Test F1
No uppercase word	94.67	69.30	91.71	62.30
No first letter uppercase	89.19	45.12	86.82	40.54
No current tag	94.99	72.02	91.76	63.18
No word based	87.80	37.38	85.40	35.77
Full model	94.67	70.28	91.79	62.46

Analysis:

1) Upper case word
    a) There didn't seem to be much impact on whether we tracked this feature or not. There was a slight gain but probably not one that is significant enough to make an impact on sentence understanding
2) Upper case first letter
    a) This had a much larger impact on accuracy and F1. The F1 score drop 30+%. This tells us that we either rely on this feature too much for disambiguating tags from O or that it is actually a super important feature in recognizing word differences.
    b) The accuracy didn't drop as much because of how sparse the data is. Most words are tagged as O and as such, even if we guess O for everything we would have about 85% accuracy. Even then 5% drop is a significant enough drop.
3) Current tag
    a) This feature was just "what is the current tag". I was very surprised that removing this feature resulted in an increase! Almost 2% on dev F1 and almost 1% on test F1. This means that this feature actually confused the tags together and did not help in defining the boundaries between them.
    b) The accuracy also increased .3% Which means this helped us tag more things that we mistook as O
4) Word based
    a) This was there to see how well this model works when not relying on the basics. When working in this assignment I set it up to mirror hmms, with bigram transition features and emission features. Removing these was to see how well we did without emission features. First result was the run time was much faster (since we have significantly less features), one minute vs 3 min normal runtime.
    b) We can see from the F1 drop that this was a crucial part in determining our tags. A 30% and a 35% drop on the dev and test sets respectively. Without this feature we get the lowest scores we see out of any other features meaning this has the most impact. This makes sense as some words are more commonly seen and tagged as specific NER tags, and other words like "a" and "the" are usually only see if the tag is O.
    c) The accuracy doesn't seem to have dropped as much but thinking about it, the drop is significant. Given that assigning O to everything yields 85% range, getting an accuracy without this feature of 87% means we are barely tagging anything not O. We extremely lose the ability to disambiguate between the words needing tags and those who don't.

BIO vs IO:

There is a place for each of the tagging strategies depending on the data you have. In this assignment we used BIO. One pro of BIO is that we can differentiate between 2 words the are of the same tag, but are different entities. We can have B-PER I-PER B-PER I-PER in a row, while if this was in IO format we would see I-PER I-PER I-PER I-PER and would assume that this is one long person name and not 2 separate entities. This will in turn increase accuracy and F1 scores since during training we will be able to learn these entity/sentence structures such that if they repeat together or separate in the test set we would still tag them.

A negative of using BIO is that it take a large toll on performance of the model. If we have 4 possible tags PER, LOC, MISC, ORG, each one can be B or I + the O tag, means we have 9 possible tags. When creating features we use our template on each of these possible tags, for example word#tag feature where we are basically measuring emissions. With 9 possibilities and 1000 words we have 9000 features. If we were to use IO instead, we would have 5000 features. This affects the time taken to compute the feature vectors, adding, and updating weights. Another issue is that this increases performance in decoding. The runtime for viterbi is in Tn^2 where n is the number of states. Although this translates to a constant increase, and so O() would not be affected, in practice, almost double the features means double the runtime which will make these sort of taggers less usable in production environments

The cases in which IO performs better than BIO might be where the chances of the same tag appearing concurrently is really low. In those cases, the performance would be comparable. Since there are fewer states to work with the learning algorithm does not have to split the weights between and I and a B feature, it can just indicate what is a strong signal for the tag without the worry of B or I.