Assignment 4: CoNLL2002 Spanish Named Entity Recognition

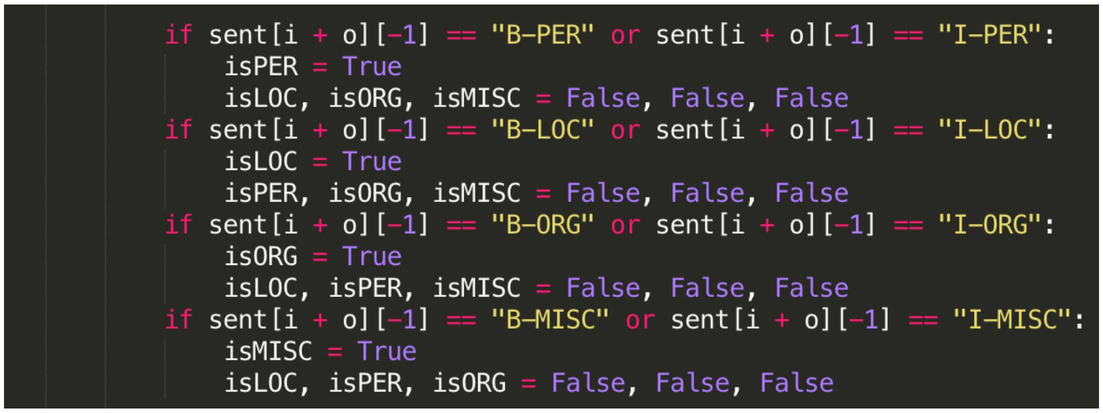
Section 1: Constrained

The **results** are shown in Figure 1 below:

```
processed 51533 tokens with 3558 phrases; found: 3676 phrases; correct: 3345.
accuracy:  99.48%; precision:  91.00%; recall:  94.01%; FB1:  92.48
              LOC: precision:  96.27%; recall:  97.69%; FB1:  96.98  1100
             MISC: precision:  62.38%; recall:  74.34%; FB1:  67.83  404
              ORG: precision:  91.64%; recall:  93.93%; FB1:  92.77  1435
              PER: precision:  97.56%; recall:  97.82%; FB1:  97.69  737
```

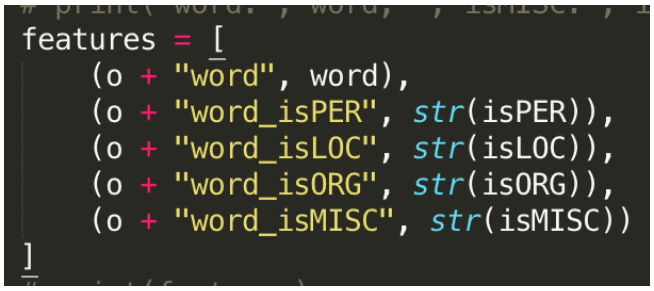
Figure 1: Evaluation result of constrained_results.txt

The **features** I used to achieve Figure 1 is to add 4 extra dimension/features. To explain in brief, I implemented a variable for each word to know if they are within the category of PER, LOC, ORG and MISC. I thought this was very important because this further generalizes the category or bins to match the observation(actual word) to the state (NER). The code for the additional features are shown in Figure 2 and 3.



```
if sent[i + o][-1] == "B-PER" or sent[i + o][-1] == "I-PER":
    isPER = True
    isLOC, isORG, isMISC = False, False, False
if sent[i + o][-1] == "B-LOC" or sent[i + o][-1] == "I-LOC":
    isLOC = True
    isPER, isORG, isMISC = False, False, False
if sent[i + o][-1] == "B-ORG" or sent[i + o][-1] == "I-ORG":
    isORG = True
    isLOC, isPER, isMISC = False, False, False
if sent[i + o][-1] == "B-MISC" or sent[i + o][-1] == "I-MISC":
    isMISC = True
    isLOC, isPER, isORG = False, False, False
```

Figure 2: Implementing additional features



```
features = [
    (o + "word", word),
    (o + "word_isPER", str(isPER)),
    (o + "word_isLOC", str(isLOC)),
    (o + "word_isORG", str(isORG)),
    (o + "word_isMISC", str(isMISC))
]
```

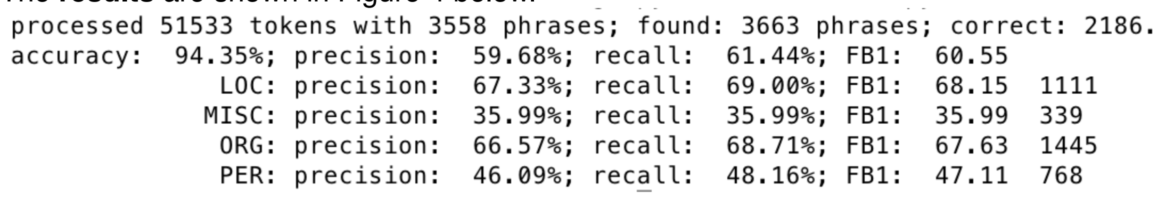
Figure 3: Adding additional features

Section 2: Hmm

The main work in this section is preparing the start, transition and emission probability matrix.

- **Hidden state**: the NER, such as B-LOC.etc
- **Observation**: is the actual word, such as “La”.
- **Start probability matrix**: represents the probability for the first word of each sequence/phrase.
- **Transition probability matrix**: represents the probability for a given hidden state to transfer to another hidden state.
- **Emission probability matrix**: represents the probability for an observation to occur, given a hidden state.

The **results** are shown in Figure 4 below.



```
processed 51533 tokens with 3558 phrases; found: 3663 phrases; correct: 2186.
accuracy:  94.35%; precision:  59.68%; recall:  61.44%; FB1:  60.55
              LOC: precision:  67.33%; recall:  69.00%; FB1:  68.15  1111
             MISC: precision:  35.99%; recall:  35.99%; FB1:  35.99  339
              ORG: precision:  66.57%; recall:  68.71%; FB1:  67.63  1445
              PER: precision:  46.09%; recall:  48.16%; FB1:  47.11  768
```

Figure 4: Hmm results

There are 2 main implementations that were crucial to achieve > 60 F1 score.

The first is **handling unknown observations**. These are words that exist in the test set, but not in the train set. I handled it by randomly choosing a word from the observation(word) list generated from the train set. The idea is that if the guess is random, it would not skew towards any category/answer.

The second is **replacing 0(zeros) in the probability matrix with a very small number**. The idea behind this is to prevent backward propagation to end up in zero.

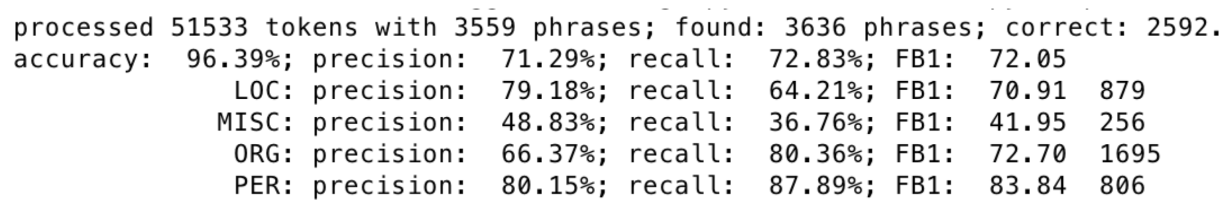
Section 3: Sky is the Limit

For this section, I used **LSTM-CRF** (https://github.com/glample/tagger). I spent a lot of time debugging and re-writing parts of the module in order for it to work (and also for it to execute easily).

Currently, training takes a long time; even running 5 epochs. However, the good thing with the implementation is that it allows us to stop early and does evaluation along the way. This can be done using “ctrl+c”. Another problem is that the model trained takes too much space, and therefore, you will need to train the model and then run prediction script. Instructions on how to run this is in the README.

The **results** with LSTM-CRF is shown in Figure 5 below. This result is definitely better than hmm’s result in every way. It makes sense, because LSTM is a better model for nlp tasks, such as this NER task.

In the future, more research into the parameters to fine tune this might be yield better results.



```
processed 51533 tokens with 3559 phrases; found: 3636 phrases; correct: 2592.
accuracy:  96.39%; precision:  71.29%; recall:  72.83%; FB1:  72.05
             LOC: precision:  79.18%; recall:  64.21%; FB1:  70.91  879
            MISC: precision:  48.83%; recall:  36.76%; FB1:  41.95  256
             ORG: precision:  66.37%; recall:  80.36%; FB1:  72.70  1695
             PER: precision:  80.15%; recall:  87.89%; FB1:  83.84  806
```

Figure 5: LSTM-CRF (Epoch=5) results