6.863J/9.611J: Natural Language Processing Prof. Robert C. Berwick

Assignment 4: HMM named-entity tagging

My posse consists of: Sam Ainsley *Salman Ahmad (saahmad@mit.edu)*

In case it is useful, you can run all of the taggers by simply running `bash eval.sh` in my code directory.

1. **Problem 1**

   The results from the unigram are:

```
Found 14043 NEs. Expected 5931 NEs; Correct: 3117.

 precision  recall  F1-Score
Total:  0.221961 0.525544 0.312106
PER:  0.435451 0.231230 0.302061
ORG:  0.475936 0.399103 0.434146
LOC:  0.147750 0.870229 0.252612
MISC:  0.491689 0.610206 0.544574
```

2. **Problem 2**

   The results from the bigram are:

```
Found 4472 NEs. Expected 5931 NEs; Correct: 3128.

 precision  recall  F1-Score
Total:  0.699463 0.527398 0.601365
PER:  0.617253 0.400979 0.486148
ORG:  0.531476 0.384903 0.446467
LOC:  0.841415 0.700109 0.764286
MISC:  0.756066 0.642780 0.694836
```

   The results seem to be an improvement over the unigram model.

3. **Problem 3**

   The results from the trigram are:

```
Found 3926 NEs. Expected 5931 NEs; Correct: 3270.

 precision  recall  F1-Score
Total:  0.832909 0.551340 0.663488
PER:  0.861290 0.435800 0.578757
ORG:  0.712644 0.417040 0.526167
LOC:  0.860634 0.710469 0.778375
MISC:  0.869814 0.660152 0.750617
```

   As can be seen, the results improved from both the unigram and bigram models.

   The one thing that stood out in development is that you needed to rely on smoothing to get decent results. Computing the trigram probabilities will often result in a bigram in the denominator that has never been seen before. For this problem, I just implemented Laplace smoothing to my trigrams and that worked well enough.

4. **Problem 4**

The results for my final tagger are:

```
Found 4237 NEs. Expected 5931 NEs; Correct: 3670.

 precision  recall  F1-Score
Total:  0.866179 0.618783 0.721873
PER:  0.912338 0.611534 0.732248
ORG:  0.787841 0.474589 0.592351
LOC:  0.867598 0.711014 0.781540
MISC:  0.872126 0.659066 0.750773
```

As can be seen, all of the F1 scores increased over the old trigram tagger. To achieve this performance I added the following classes:

- _ABBREVIATION_: any string that consisted of a mix of uppercase letters and '.'
- _NUMBER_: any string that consistent only of numbers, '-' and '.'
- _CAPITALIZED_: any string that had the first letter uppercase but the rest lowercase
- _UPPERCASE_: any string that had all of their letters uppercased
- _LOWERCASE_: any string that has all of their letters lowercased
- _RARE_: any string that did not meet any of the above criteria

While these classes seem fairly boring I also explored many other more interesting classes. For example, I had a special class for _PERSONAL_ABBREVIATIONS_ (strings that consisted just of an upper case first letter and then a period), _HYPHENS_ (strings that had a hyphen in them), and even classes based on length like _SHORT_ and _LONG_.

The interesting thing is that with these sophisticated features my performance dropped substantially. In some cases my total F1 score would fall as low as 0.622202. My guess is that I was over fitting and there was not enough data to justify these very specific tags.

Some other things that I explored was playing around with the threshold for infrequent words. While 5 seemed reasonable, I played around with other numbers just to build intuition. However, I found that sticking with 5 seemed to generate the best results.