Statistical NLP - Assignment 4

due Wednesday December 4 - at 11:59pm by email to aparikh@cs.nyu.edu and urvish@nyu.edu

Deliverables:

- Code zip emailed to instructors. You should use Python / numpy to implement the assignment. You can use the provided code as a starting point.
- Write up with your results and answers to other questions in the homework.
- **Submission Format:** Please title your submission email **Assignment 4 submission**. You should submit a zip file (firstname_lastname_hw4.zip) containing the report in pdf format (firstname_lastname_hw4.pdf) and the zipped code directory. Write your netID and your collaborators' netID (if any) in the report.

In this assignment, you will explore hidden Markov models and the viterbi algorithm. This assignment is from COMS W4705 at Columbia (taught by Michael Collins). Please go to the following site:

`http://www.cs.columbia.edu/%7Emcollins/cs4705-spring2019/homeworks.html`

and download the following files to help you with this assignment:

`count_freqs.py`
`eval_ne_tagger.py`
`ner_train.dat`
`ner_dev.dat`
`ner_dev.key`

In this programming problem, you are going to build a trigram HMM tagger for named entities. The joint probability of a sentence $x_1, x_2, ..., x_n$ and a tag sequence $y_1, y_2, ..., y_n$ is defined as:

$$p(x_1, ..., x_n, y_1, ..., y_n) = q(y_1, *, *)q(y_2|y_1, *)q(\texttt{STOP}|y_{n-1}, y_{n-2}) \times \prod_{i=3}^{n} q(y_i|y_{i-1}, y_{i-2}) \prod_{i=1}^{n} e(x_i|y_i)$$

$*$ is a padding symbol that indicates the beginning of a sentence, `STOP` is a special HMM state indicating the end of a sentence.

We provide a training data set `ner.train.dat` and a development set `ner.dev.key`. Both files are in the following format (one word per line, word and token separated by space, a single blank line separates sentences.)

```
U.N. I-ORG
official O
Ekeus I-PER
heads O
for O
Baghdad I-LOC
.  O
Senior O
United I-ORG
Nations I-ORG
arms O
official O

...
```

There are four types of named entities. person names (PER), organizations (ORG), locations (LOC) and miscellaneous names (MISC). Named entity tags have the format I-TYPE. Only if two phrases of the same type immediately follow each other, the first word of the second phrase will have tag B-TYPE to show that it starts a new entity. Word marked with O are not part of a named entity.

The script `count_freqs.py` reads in a training file and produces trigram, bigram, and emission counts. Run the script on the training data and pipe the output into some file:

```
python count_freqs.py ner.train.dat > ner.counts
```

Each line in the output contains the count for one event. There are two types of counts:

- Lines where the second token is `WORDTAG` contains emission counts $Count(y, x)$, for example

  ```
  13 WORDTAG I-ORG University
  ```

  indicates that `University` was tagged 13 times as `I-ORG` in the training data.

- Lines where the second token is n-GRAM (where n is 1,2, or 3) contain unigram counts $Count(y)$, bigram counts $Count(y_{n-1}, y_n)$ or trigram counts $Count(y_{n-2}, y_{n-1}, y_n)$. For example,

  ```
  3792 2-GRAM O I-ORG
  ```

  indicates that there were 3792 instances of an I-ORG tag following an O tag and

  ```
  1586 3-GRAM O I-ORG I-ORG
  ```

  indicates that in 1586 cases the bigram O I-ORG was followed by another I-ORG tag.

Question 1 (25 points): Use the counts produced by `count_freqs.py`, write a function that computes emission parameters

$$e(x|y) = \frac{Count(y, x)}{Count(y)}$$

We need to predict emission probabilities for words in the test data that do not occur in the training data. One simple approach is to map infrequent words in the training data to a common class and to treat unseen words as members of this class. Replace infrequent words ($Count(x) < 5$) in the original training data file with a common symbol `_RARE_`. Then re-run `count_freqs.py` to produce new counts. Submit your code.

Question 2 (25 points): As a baseline, implement a simple named entity tagger that always produces the tag $y^* = \text{argmax}_y e(x|y)$ for each word x. Make sure your tagger uses the `_RARE_` word probabilities for rare and unseen words. Your tagger should read in the counts file and the file `ner.dev.dat` (which is the `ner.dev.key` without the tags) and produce output in the same format as the training file, but with an additional column in the end that contains the log probability for each prediction. For instance

```
Nations I-ORG -9.2103403719761818
```

Submit your program and report on the performance of your model. You can evaluate your model using the evaluation script:

```
python eval_ne_tagger.py ner.dev.key prediction.file
```

In addition, write up any observations you made.

Question 3 (25 points): Using the counts produced by `count_freqs.py`, write a function that computes parameters

$$q(y_i|y_{i-1}, y_{i-2}) = \frac{Count(y_{i-2}, y_{i-1}, y_i)}{Count(y_{i-2}, y_{i-1})}$$

for a given trigram y_{i-2}, y_{i-1}, y_i. Make sure your function works for the boundary cases $q(y_1|*, *)$, $q(y_2|*, y_1)$ and $q(\texttt{STOP}|y_{n-1}, y_n)$.

Write a program that reads in lines of state trigrams y_{i-2}, y_{i-1}, y_i (separated by space) and prints the log probability for each trigram. Submit the program.

Question 4 (25 points): Using the maximum likelihood estimates for transitions and emissions, implement the Viterbi algorithm to compute

$$\text{argmax}_{y_1,...,y_n} p(x_1, ..., x_n, y_1, ..., y_n)$$

Your tagger should have the same basic functionality as the baseline tagger. Instead of emission probabilities the third column should contain the log-probability of the tagged sequence up to this word. Submit your program and report on the performance of your model. In addition, write up any observations you made.