COMS 4705 Natural Language Processing (Fall 2018)
Problem Set #1 (Programming)

Shreyas Mundhra - ssm2211@columbia.edu

September 23, 2018

In addition to the required files, the following files are present:

1. `utils.py`: Contains utility functions that are commonly needed across the problems.
2. `algorithms.py`: Contains all the algorithms (only Viterbi algorithm in this case).

Question 4

How to run

The following steps need to be followed to generate the required output for this question:

1. Run `4_1.py`. This creates the file `ner_train_rare.dat`, the data file where infrequent words (occurring less than 5 times) are replaced with the `_RARE_` symbol.
2. Run `4_2.py`. This creates the file `4_2.txt`, the tagged `ner_dev.dat` data with the extra log likelihood column. This is a simple named entity tagger that only uses the emission parameters to assign tags. Hence, this tagger assumes that the tag for a word in a sentence only depends on that word and nothing else.

Performance

The performance of the model is shown in the figure below:

Found 12082 NEs. Expected 5931 NEs; Correct: 2931.

	precision	recall	F1-Score
Total:	0.242592	0.494183	0.325432
PER:	0.435451	0.231230	0.302061
ORG:	0.475936	0.399103	0.434146
LOC:	0.159484	0.768811	0.264169
MISC:	0.491689	0.610206	0.544574

Figure 1: Performance of the model

Question 5

How to run

The following steps need to be followed to generate the required output for this question:

1. Run `5_1.py`. This reads the file `trigrams.txt` and the counts file `ner_rare.counts` and creates the file `5_1.txt`, which contains the trigrams and their respective log transition probabilities.

2. Run `5_2.py`. This creates the file `5_2.txt`, the tagged `ner_dev.dat` data in the same format as `4_2.txt` but tagged using the Viterbi tagger. The counts file `ner_rare.counts` is used to compute the probabilities. Rare or unseen words in a sentence are replaced by the `_RARE_` symbol before running the Viterbi algorithm.

Performance

The performance of the model is shown in the figure below:

```
Found 4698 NEs. Expected 5931 NEs; Correct: 3643.

         precision      recall         F1-Score
Total:   0.775436       0.614230       0.685483
PER:     0.759749       0.593580       0.666463
ORG:     0.611855       0.478326       0.536913
LOC:     0.876458       0.696292       0.776056
MISC:    0.836627       0.689468       0.755952
```

Figure 2: Performance of the model

Observations

1. The recall for the named entity LOC decreased i.e. a fewer number of locations were actually predicted to be locations.

2. Except the recall for LOC, all other metrics were better than the baseline tagger and the total precision, total recall and total F1-Score were also higher, indicating that this is a much better model as a whole.

Question 6

How to run

1. Run 6.py. This serves the following purposes:
   (a) It creates the file ner_train_cats.dat, where low-frequency words in ner_train.dat are grouped and replaced by the symbols for their corresponding categories.
   (b) It creates the counts file ner_cats.counts using ner_train_cats.dat.
   (c) It creates the file 6.txt, the tagged ner_dev.dat data in the same format as 4_2.txt and 5_2.txt but tagged using the improved Viterbi tagger. The counts file ner_cats.counts is used to compute the probabilities. Rare or unseen words in a sentence are replaced by the symbol for their category before running the Viterbi algorithm.

Categories

Rare or unseen words were divided into the following categories:

1. _NUMBER_: Words that are actually numbers
2. _SUBNUMBER_: Words that contain numbers
3. _ABV_: Words that are abbreviations. These are assumed to be words that contain at least one upper case letter, at least one "." and no other kind of characters.
4. _LOWER_AND_DOTS_: Words that contain at least one lower case letter, at least one "." and no other kind of characters.
5. _ALL_CAPS_: Words that contain only upper case letters.
6. _ALL_LOWER_: Words that contain only lower case letters.
7. _WORD_COMB_: Words that are combinations of two or more words. These are assumed to be words that contain at least one letter, at least one "-" and no other kind of characters.
8. _RARE_: Words that fall into none of the above categories.

How categories were chosen

The following iterative process was followed in order to decide the categories:

1. Store all words from ner_train.dat with _RARE_ category in a text file.
2. Manually inspect the text file and identify commonly occurring patterns in the words.
3. Add one or more categories that take into account the commonly occurring patterns.

4. Go to step 1.

The goal was to reduce the number of words in `_RARE_` category to a reasonable number. In the end, out of 18710 words in `ner_train.dat`, there were 6201 words in `_RARE_` category and the remaining 12509 words were in other categories.

Performance

The performance of the model is shown in the figure below:

```
Found 5821 NEs. Expected 5931 NEs; Correct: 4337.

         precision      recall         F1-Score
Total:   0.745061       0.731243       0.738087
PER:     0.809978       0.786181       0.797902
ORG:     0.541262       0.666667       0.597455
LOC:     0.841977       0.752454       0.794702
MISC:    0.826667       0.673181       0.742071
```

Figure 3: Performance of the model

Observations

1. The precision for the named entity ORG decreased i.e. a fewer number of predicted organizations were actually organizations.
2. The precision for the named entity LOC decreased i.e. a fewer number of predicted locations were actually organizations.
3. The precision for the named entity MISC decreased i.e. a fewer number of predicted miscellaneous names were actually miscellaneous names.
4. The recall for the named entity MISC decreased i.e. a fewer number of miscellaneous names were actually predicted to be miscellaneous names.
5. Since both precision and recall for MISC decreased, the final F1-Score for MISC also decreased.
6. All the remaining metrics for each named entity increased.
7. While the total precision decreased, the total recall increased. However, since the total F1-Score increased, we can conclude that this model is better as a whole compared to the naive Viterbi algorithm implemented in the previous question.