Natural Language Processing
Assignment 1 (Programming)

Rhea Goel (rg2936)

Problem 4.

1. All words with total frequency < 5 were replaced with **'_RARE_'** in the training file.
2. Using the counts produced by count freqs.py, **emission parameters** were computed. (stored in emissionParamsRare.txt)
3. A **baseline tagger** was implemented, that **produce the tag with the highest emission probability value** and tested on the development data.
4. Results of the tagger were evaluated using the script provided.

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

	precision	recall	F-1 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

Observations:

- The simple baseline tagger that just predicts the tag with the highest emission probability performs rather poorly. This is because it does not capture the local context of the word (n-grams) and simply outputs the tag, which is most likely to produce the word.
- Since this tagger is based on emission parameters alone, all rare words are assigned the same tag.

(EXTRA TRIAL)

5. In the tagger above, only **unseen** words in the development dataset were treated as _RARE_ words. Next, I tried to replace all **infrequent/rare** words in the development data with _RARE_ as well. However, the results deteriorated (as shown below). Such behavior can be attributed to the fact that the test data may have a lot of infrequent words, and some of which may not have been rare in the training data.

Found 17499 NEs. Expected 5931 NEs; Correct: 2547.

	precision	recall	F-1 score
Total:	**0.145551**	**0.429439**	**0.217414**
PER:	0.335385	0.118607	0.175241
ORG:	0.399183	0.218984	0.282819
LOC:	0.105798	0.874591	0.188761
MISC:	0.452830	0.469055	0.460800

Problem 5.

1. Using the counts produced by count freqs.py, **transition parameters q** were computed. (stored in qParamValues.txt)
2. The **Viterbi Algorithm** was implemented, which used the **log values of emission parameters, and q parameters** computed above.
3. Execution time: About 1 minute
4. Results of the tagger were evaluated using the script provided.

Found 4648 NEs. Expected 5931 NEs; Correct: 3656.

	precision	**recall**	**F1-Score**
Total:	**0.786575**	**0.616422**	**0.691181**
PER:	0.743641	0.604461	0.666867
ORG:	0.659729	0.473842	0.551544
LOC:	0.895994	0.695202	0.782929
MISC:	0.825974	0.690554	0.752218

Observations:

- The precision, recall and **F-measure have improved considerably** from the baseline tagger which produced tags based only on the emission parameters, to the HMM tagger implemented using the Viterbi algorithm, which takes into account the **local context of a word** as well.
- Given the nature of the problem, there were several bigram and trigram counts equal to zero, thereby giving 0 value to the corresponding q and emission parameters. Here, the log values of the zero q and e parameters have been set to -100000 (because log(0) is –inf).
- Cases where trigram and bigram counts are 0, degrade the performance of the tagger. To further enhance the performance, one can add linear interpolation.
- Another observation while programming: the algorithm takes a lot of time, if the implementation is done poorly. I optimized my code, using more and more dictionaries to decrease the run time.

Problem 6.

Improved the way the HMM tagger (implemented above) deals with low-frequency words by grouping them based on informative patterns rather than just into a single class of rare words.

Intuition for patterns: a class could contain all capitalized words (possibly proper names), all words that contain only capital letters and dots (possibly abbreviations for first names or companies), or all words that consists only of numerals

Hence, **Patterns used:**

- *NUM*: words with all numerals
- *CAPS_DOTS*: words containing only capital letters and dots
- *CAPITALIZED*: words that start with a capital letter of the alphabet

New counts file was generated, and the steps repeated. Viterbi Algorithm was run, and results of the new (fancy) tagger were evaluated using the script provided.

Found 5832 NEs. Expected 5931 NEs; Correct: 4282.

	precision	**recall**	**F1-Score**
Total:	**0.734225**	**0.721969**	**0.728046**
PER:	0.785438	0.774755	0.780060
ORG:	0.531215	0.699552	0.603871
LOC:	0.865462	0.705016	0.777043
MISC:	0.824377	0.682953	0.747031

Observations:

- The precision, recall and F-measure values have **further improved**. This could be attributed to the fact that we're now grouping them based on informative patterns rather than just into a single class of rare words.
- This way, the tagger is **handling the unseen words more smartly**/intelligently than the previous case.