NTMI 2015: Project Exercises
Part A (steps 1-3)

This project has four steps. The steps are gradual and allow you to build the project in blocks that you can re-use in subsequent steps.

1 STEP 1: Extracting ngram statistics

Write a program which takes as input a natural number n and a large text file (a corpus). The program should construct a table of all ngrams – word sequences of length n – in the corpus together with the number of times each sequence appears in the corpus (the ngram corpus frequency of the sequence). The word sequences should be exactly as they appear in the corpus, so 'The' $\neq$ 'the'.

Extend your program to take an additional argument m, and output the m most frequent sequences (together with their frequencies and in decreasing frequency order).

Test your program with the AUSTEN TRAIN corpus (`http://www-nlp.stanford.edu/fsnlp/statest/austen.txt`) as the input corpus. In this exercise we consider the whole corpus as one long sequence of words (in subsequent exercises we will split the corpus into sentences).

Submit the following:

1. For $n = 1$, $n = 2$ and $n = 3$, submit the list of the 10 most frequent sequences.
2. For $n = 1$, $n = 2$ and $n = 3$, submit the sum of all frequencies of all sequences for that n.
3. Submit your program with clear instructions as to how it should be compiled and run.

Command line format:[1]

```
./a1-step1 -corpus [path] -n [value] -m [value]
```

2 STEP 2: Markov language models using ngram statistics

For this exercise, assume that the beginnings and ends of paragraphs correspond to the beginnings and ends of sentences (that is, each paragraph is one sentence). Therefore, add a $START/END$ symbol at the beginning and end of each paragraph.

Write a program that performs the following tasks on an input corpus (test this on the AUSTEN TRAIN corpus `http://www-nlp.stanford.edu/fsnlp/statest/austen.txt`).

1. Given a natural number n and a corpus, the program constructs two tables: (1) $(n-1)$-grams and (2) n-grams from the corpus, together with their frequencies.

   Report the 10 most frequent bigrams in the corpus together with their frequencies.

   ```
   NOTE: You may use the program from STEP 1 for this, but notice that
   by adding the START/STOP symbols, the frequencies may have changed.
   ```

   Command line:

   ```
   ./a1-step2 -corpus [path] -n [value]
   ```

2. The program should accept an additional file as input. This file should contains sequences of n words $w_1 \dots w_n$ (one sequence per line, words separated by white space and ordered from left to right). Your program may ignore lines containing sequences of length other than n.

   For each sequence $w_1 \dots w_n$ in the file, the program should report $P(w_n | w_1, \dots, w_{n-1})$ (the probability that the sequence $w_1, \dots, w_{n-1}$ is followed by w_n). The program should calculate this probability

[1] `a1-step1` can be a bash/csh script file invoking a Java virtual machine or a python interpreter. If you are not familiar with scripts, it is sufficient if your java class accepts the specified parameters. Then you command line may look like: "`java -cp . A1Step1 -corpus [path] -n [value] -m [value]`." This applies to all assignments.

based on the $(n-1)$-gram and n-gram tables constructed from the corpus.

Command line:

```
./a1-step2 -corpus [path] -n [value] -conditional-prob-file
[path]
```

3. The program should also accept a file with a list of sentences as input. Every line (=sentence) in this file is a list of words (separated by white space) of arbitrary length. For each sentence $w_1 \dots w_m$ in the file, the program should report the probability of the sentence based on the n-gram model estimated from the corpus. For this, the program should use the following formula:

$$P(w_1, \dots, w_m) = \prod_{i=1}^{i=m+1} P(w_i | w_{i-n+1}, \dots, w_{i-1})$$

   where $w_j = START$ for $j \leq 0$ and $w_{m+1} = STOP$.

   Remarks:

   (a) If $n > 2$, this formula adds more than one $START$ symbol at the beginning of the sentence. This is the same as using shorter sequences for the first words of the sentence (for example, if $n = 3$, we can use $START$, w_1 instead of $START$, $START$, w_1 for the first word in the sentence).

   (b) Because sentences are long, probabilities may be very small. Make sure small positive probabilities are not rounded to zero by the built-in floating point operations.

   Command line:

```
./a1-step2 -corpus [path] -n [value] -sequence-prob-file [path]
```

4. Given are the following two sets of words:

$$A = \{know, I, opinion, do, be, your, not, may, what\}$$

$$B = \{I, do, not, know\}$$

   (a) Write a procedure which reads in a set of words and generates all their permutations.

(b) For each such permutation, calculate the probability of that permutation as a sentence (that is, add start and stop symbols). For this calculation, use a first order Markov model ($n = 2$ of parts 1 - 3 above). For each set of words, report the two permutations with the highest probability.

(c) Report the output of this procedure on set A and set B given above.

Command line:
```
./a1-step2 -corpus [path] -scored-permutations
```

3 STEP3: Smoothing ngram statistics

In this exercise, we will again use the Austen Train corpus of the previous exercises. For training we will use the same corpus as before (`http://www-nlp.stanford.edu/fsnlp/statest/austen.txt`), but we will test the model on additional data (`http://www-nlp.stanford.edu/fsnlp/statest/ja-pers-clean.txt`).

1. Construct a bigram language model based on the training corpus. For this, use the program of the previous exercise.
2. Implement Add-1 smoothing and apply it to the bigram model of the first part.
3. Implement Good-Turing smoothing and apply it to the bigram model of the first part.
   To avoid trouble, use smoothing only for bigrams which appeared $k \leq 5$ times in the training corpus. Use the standard Good-Turing formula (4.27)[2] for $r = 0$ *and distribute mass uniformly over unseen (zero occuring* $r = 0$*) bigrams*, and the following formula (4.31 in the book) for $1 \leq r \leq k$:
   $$r^* = \frac{(r+1)\frac{n_{r+1}}{n_r} - r\frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$$
   Use no smoothing for $r > 5$ (keep $r^* = r$ for $k > 5$).

Remarks:

[2] I am referring to formulas in Jurafsky and Martin Edition 2, 2009.

1. Assume that $|V|$ (the number of words in the vocabulary) is equal to the number of word types in the full AUSTIN corpus.
2. To calculate the smoothed probability $\hat{P}(w_i|w_{i-1})$, you need to use the formula $\hat{P}(w_i|w_{i-1}) = \frac{Count^*(w_{i-1}w_i)}{\sum_{w \in V \cup \{STOP\}} Count^*(w_{i-1},w)}$ (where $Count^*$ is the frequency after smoothing).[3] Make sure you calculate the sum $\sum_{w \in V \cup \{STOP\}} Count^*(w_{i-1}, w)$ in an efficient way.
3. Test the model without smoothing and with both forms of smoothing (with $k = 5$ in the second smoothing model) on the test corpus, as follows:
   (a) Report the percentage of sentences which are assigned probability zero by each of the models.
   (b) Report the first 5 sentence in the test corpus which are assigned a probability of zero by each of the models.
   (c) Explain the differences between the models.
4. Please submit the following:
   - The program with clear instructions as to how to compile and run it.
   - The output for part 3.

The requested information should printed when run with following parameters:

```
./a1-step3 -train-corpus [path] -test-corpus [path] -n [value] -smoothing
[no|add1|gt]
```

4 STEP 4: POS tagging

This step will be available soon...

[3]Why is it no longer possible to use the formula $P(w_i|w_{i-1}) = \frac{Count(w_{i-1}w_i)}{Count(w_{i-1})}$?