# Homework #5: CMPT-413

Distributed on Mon, Feb 21; Due on Mon, Mar 7

Anoop Sarkar – anoop@cs.sfu.ca

(1) **Finding Proper Names**: Consider a part of speech tagged sentence represented as a list of strings where each element in the list is a word and associated part of speech tag:

```
@sentence1 = ("Show/VB", "me/PRP", "the/DT", "flights/NNS", "from/IN",
"San/NNP", "Diego/NNP", "to/TO", "Toronto/NNP");

@sentence2 = ("From/IN", "Washington/NNP", "D/NNP", "C/NNP", "to/TO",
"Philadelphia/NNP", "on/IN", "Wednesdays/NNPS");
```

In these sentences, there are some meaningful strings that span multiple words. For example, *"San Diego"*, *"Toronto"* and *"Washington D C"* are names of cities. If we wanted to find names, places, or days of the week in part of speech tagged text then a useful heuristic to apply is to search for the sequences of words that are tagged as *proper nouns*.

Assume that you are using the Penn Treebank tagset: NNP is the tag for singular proper noun, e.g. Toronto/NNP; and NNPS is the tag for plural proper noun, e.g. Wednesdays/NNPS.

Write a function in Perl called chunkProperNouns that takes a part of speech tagged sentence represented as a list of word/tag strings and merges sequences of proper nouns into a single string and returns a list of all such merged strings.

chunkProperNouns(@sentence1) returns ("San/NNP Diego/NNP", "Toronto/NNP")
chunkProperNouns(@sentence2) returns ("Washington/NNP D/NNP C/NNP", "Philadelphia/NNP", "Wednesdays/NNPS")

Submit a Perl program called proper_nouns.pl that prints out the output of the function chunkProperNouns on the two sentences given above. Make sure that the proper nouns groups are explicitly shown in your output. For example, here is a sample output for the two sentences above (note the commas that separate each merged proper noun sequence):

```
San/NNP Diego/NNP , Toronto/NNP
Washington/NNP D/NNP C/NNP , Philadelphia/NNP , Wednesdays/NNPS
```

(2) **Smoothing *n*-grams**: This question will use two data files: atis3.pos (the ATIS part of speech tagged corpus) and wsj3_00.pos (Section 0 from the Penn Treebank Wall Street Journal part of speech tagged corpus). The file wsj3_00.pos will be our source of *training data* and the file atis3.pos will be our source of *test data*. Use the supplied Perl program clean-tagged.pl to create the files wsj3_00.pos.tags and wsj3_00.pos.emit from wsj3_00.pos. Also, create the files atis3.pos.tags and atis3.pos.emit from atis3.pos.

Then use the supplied Perl programs skip.pl, paste.pl, clean-ngram.pl, ngramCounts.pl and ngramLogProb.pl to produce a bigram model of the part of speech tag sequences from the file wsj3_00.pos.tags (this is our training data). Before computing the bigram counts using ngramCounts.pl, use clean-ngram.pl to remove some unwanted bigrams (see the code for more info).

The output of ngramLogProb.pl will be a bigram model of tag sequences stored as log probabilities: $\log_2 P(t_i \mid t_{i-1})$.

Let $T = s_0, \ldots, s_m$ represent the test data with sentences $s_0$ through $s_m$.

$$P(T) = \prod_{i=0}^{m} P(s_i) = 2^{\sum_{i=0}^{m} \log_2 P(s_i)}$$

$$\log_2 P(T) = \sum_{i=0}^{m} \log_2 P(s_i)$$

where $\log_2 P(s_i)$ is the log probability assigned by the bigram model to the sentence $s_i$ (note that in this homework, each $s_i$ is a sequence of part of speech tags). These log probabilities are provided by `ngramLogProb.pl`. Let $W_T$ be the length of the text $T$ measured in part of speech tags. The *cross entropy* for $T$ is:

$$H(T) = -\frac{1}{W_T}\log_2 P(T)$$

The cross entropy corresponds to the average number of bits needed to encode each of the $W_T$ words in the *test data*. The *perplexity* of the test data $T$ is defined as:

$$PP(T) = 2^{H(T)}$$

a. Write a Perl program to compute the cross entropy and perplexity for a given input file. The input to the program is a bigram model over part of speech tag sequences and an input file with sentences that are part of speech tag sequences.

   Using this program, print out the cross entropy and perplexity for the training data `wsj3_00.pos.tags` and the test data `atis3.pos.tags`.

   On the test data, when a bigram in unseen, the probability for that bigram is zero. However, since we are using log probabilities, we cannot use a probability of zero (as $\log_2 (0)$ is not defined). Instead, use the value $\log_2 P(t_i \mid t_{i-1}) = -99999$, when a bigram $(t_{i-1}, t_i)$ is unseen.

   Remember that cross entropy and perplexity are both positive real numbers, and the lower the values, the better the model over the test data.

b. Implement the following Jelinek-Mercer style *interpolation* smoothing model:

$$P_{interp}(t_i \mid t_{i-1}) = \lambda P(t_i \mid t_{i-1}) + (1-\lambda)P(t_i)$$

   Set $\lambda = 0.8$ and using $P_{interp}$ recompute the cross-entropy and perplexity for the training data `wsj3_00.pos.tags` and the test data `atis3.pos.tags`. Your program should print out the cross-entropy and perplexity values. After you run the program for $\lambda = 0.8$, find a value for $\lambda$ that results in a better model of the test data and provide the output of your program that implements interpolation smoothing using this better $\lambda$ value.

   Use the simplifying assumption that if the unigram $t_i$ is unseen then $\log_2 P(t_i) = -99999$.

c. Implement add-one smoothing to provide counts for every possible bigram $(t_{i-1}, t_i)$. Using `ngramLogProb.pl` and your Perl program from Question 2a, recompute the cross-entropy and perplexity for the training data `wsj3_00.pos.tags` and the test data `atis3.pos.tags`. Your program should print out the cross-entropy and perplexity values. Do not smooth the unigram model $P(t_i)$.

d. After you have done *both* Question 2b and 2c, reduce test set perplexity even further by using add-one smoothing to augment the interpolation model.

Submit all the Perl programs needed to provide the outputs for each question above. Write and submit a Perl program `smooth-runall.pl` which is used to **run** all the programs as follows: `smooth-runall.pl` should start by training the bigram model, then calling the various programs written for Question 2a through Question 2d and printing out the output required for each question. A sample output is shown in the file `smooth-runall-output.txt` (this file does not contain answers, only the format of the output).