# N-Gram Models

February 10, 2015

## 1.- Create a Naïve Trigram Modeling Program (25 pts.)

For this first part you will have to create two python programs. One to read a text file and compute trigram, bigram and unigram probabilities with the existing words. Your program should be called `n-trigrams.py` and should be run like this:

```
python n-trigrams.py <input-file> <model-file>.
```

The second program should be called `test-model.py` and should be run like this:

```
python test-model.py <model-file> <sentences-file>
```

For each of the `<sentences-file>` it should output it's probability to the screen. Therefore, if there are 10 sentences in the file, it will output 10 probabilities. The sentences are separated by a period. There can be more than one sentence on each file.

The format of the `<model-file>` file must be a tab separated file with two columns: The first column is the ngram. if a unigram, then just the word. If a bigram or trigram, the words separated by space. The second column is the count of that n-gram. Add STOP as a special word. This will allow you to know how many sentences are there in the corpus.

For example, for a file containing simply:

"I am Sam. Sam I am"

the corresponding `<model>` file should look like this:

```
I       2
am      2
Sam     2
STOP     2
I am    2
am Sam  1
Sam I   1
Sam STOP        1
* Sam    1
* I      1
am STOP          1
* * I   1
* I am   1
I am Sam         1
am Sam STOP      1
* * Sam          1
* Sam I          1
Sam I am         1
I am STOP        1
```

### *Here are some steps to help you modularize your code*

- create a function that replaces punctuation and other noise (accents, tildes, newlines,etc.) in a string. End of sentence punctuation should be replaced with `<STOP>`; accents and tildes can be replaced with their corresponding English equivalent (this is optional). Newlines, commas and apostrophes should be replaced with a space.
- create a function that reads a file. For each line read it should replace punctuation and then find unigrams, bigrams and trigrams and add them to a dictionary (you can use `collections.Counter` here. The key to the dictionary can be a string or a tuple For example: `(word1,word2,word3)` for trigrams.
- create a function to save the resulting dictionary to a file and a function to read that file. The file should conform to the format in the instructions.
- Lastly, create a function to compute the probability of any bigram present in the corpus.

## 2.- Enhance your trigram model (20 pts.)

You will have to create a second program called `s-test-model.py` that works just like the `test-model.py` program from the previous question, but implements a simple interpolation to find out the probability of non-existing bigrams.

## 3.- Evaluate Perplexity (25 pts.)

Using interpolation, create a python program `preplexity.py` that should be run as follows:

```
python perplexity.py <model-file> <sentences-file>
```

and outputs the perplexity of the `<sentences-file>`

## 4.- Comparing perplexities. The naïve case (10 pts.)

Is Othello a good model for Quijote? If you train on Othello and test on Don Quijote, what is the value of perplexity there? What is the perplexity if you train on Othello and test on the Interviews from the 1970s? Lastly, If you train on Othello, what is the perplexity of The Two Gentlemen of Verona? Why do you think you get the perplexities you get?

## 5.- Comparing perplexities with Smoothing (10 pts.)

What is the value of the perplexity when trained on Othello and tested on the same files as above, but without smoothing. Why do you think this happens? What could improve this?

## 6.- Submitting your work (10 pts.)

Please submit all of your python files, plus a typed PDF file answering the last two questions as one ZIP file. Your programs and PDF files should reside in the root directory of the ZIP file (basically, do not zip your homework folder, but zip the files within that folder). Failure to comply with this specification (or any part of it) will discount 10 points from your total grade.