# NLTK and Language Models

**Lab Session**[*]

The aims of this lab session are to 1) explore the different uses of language in different documents, authored by different people 2) introduce the construction of language models using Python's Natural Language Tool Kit (NLTK) 3) explore the Laplace (add-1), Lidstone (add-$\alpha$) and Good-Turing smoothing methods for language models and 4) explore the use of language models in authorship identification.

## 1 Natural Language Tool Kit

This lab session will make use of the Natural Language Tool Kit (NLTK) for Python. NLTK is a platform for writing programs to process human language data, that provides both corpora and modules. For more information on NLTK, please visit: `http://www.nltk.org/`.

Before getting started, you should install NLTK. If you haven't already done so, follow the instructions on `http://www.nltk.org/install.html` to download the version required for your platform.

Once you've installed NLTK, start up the Python (or iPyhton) interpreter, and install the NLTK corpora by typing the following two commands at the Python prompt

```
>>> import nltk
>>> nltk.download()
```

A new window should open, showing the NLTK Downloader. Click on the File menu and select Change Download Directory to change the location. Next, select `all-corpora` to download.

Before continuing with this lab sheet, please download a copy of the lab template (lab1.py) for this lab from the moodle course website. This template contains code that you should use as a starting point when attempting the exercises for this lab. For each exercise, edit the corresponding function in the template file (e.g. ex1 for Exercise 1), then uncomment the lines which prepare for and invoke that function.

## 2 Accessing Corpora

NLTK provides many corpora and covers many genres of text. Some of the corpora are listed below:

- Gutenberg: out of copyright books
- Brown: a general corpus of texts including novels, short stories and news articles
- Inaugural: U.S. Presidential inaugural speeches

To see a complete list of available corpora, type:

```
>>> import os
>>> print os.listdir( nltk.data.find("corpora") )
```

Each corpus contains a number of texts. We'll work with the inaugural corpus, and explore what the corpus contains. Make sure you have imported the `nltk` module

first and then load the inaugural corpus by typing the following:

```
>>> from nltk.corpus import inaugural
```

To list all of the documents in the inaugural corpus, type:

[*]Based on Goldwater and Thompson, 2016.

```
>>> inaugural.fileids()
```

From this point on we'll work with President Barack Obama's inaugural speech from 2009 (2009-Obama.txt). The contents of each document (in a corpus) may be accessed via a number of corpus readers. The plaintext corpus reader provides methods to view the raw text (raw), a list of words (words) or a list of sentences:

```
>>> inaugural.raw('2009-Obama.txt')
>>> inaugural.words('2009-Obama.txt')
>>> inaugural.sents('2009-Obama.txt')
```

### Exercice 1

Find the total number of words (tokens) in Obama's 2009 speech.
Find the total number of distinct words (word types) in the same speech.
*Now uncomment the test code and check your results.*

### Exercice 2

Find the average word type length of Obama's 2009 speech.
*Now uncomment the test code and check your results.*

## 3 Frequency Distribution

A frequency distribution records the number of times each outcome of an experiment has occurred. For example, a frequency distribution could be used to record the number of times each word appears in a document:

Obtain the words from Barack Obama's 2009 speech:

```
>>> obama_words = inaugural.words('2009-Obama.txt')
```

Construct a frequency distribution over the lowercased words in the document

```
>>> fd_obama_words = nltk.FreqDist(w.lower() for w in obama_words)
```

Find the top 50 most frequently used words in the speech

```
>>> fd_obama_words.most_common(50)
```

Plot the top 50 words

```
>>> fd_obama_words.plot(50)
```

Find out how many times the words peace and america were used in the speech:

```
>>> fd_obama_words['peace']
>>> fd_obama_words['america']
```

## Exercise 3

Compare the top 50 most frequent words in Barack Obama's 2009 speech with George Washington's 1789 speech.
What can knowing word frequencies tell us about different speeches at different times in history?
*Now uncomment the test code and check your results.*

## 4 Language Models

A statistical language model assigns a probability to a sequence of words, using a probability distribution. Language models have many applications in Natural Language Processing. For example, in speech recognition they may be used to predict the next word that a speaker will utter. In machine

translation a language model may be used to score multiple candidate translations of an input sentence in order to
nd the most fluent/natural translation from the set of candidates.

## 4.1 Building a Language Model

NLTK provides a module for building language models. A language model is created in three steps from training data:

1. build a vocabulary object, e.g. `vocab = build_vocabulary(cutoff, texts)`, where **cutoff** is the minimum frequency for a word to be considered a part of the vocabulary, and **texts** is the list of the words in the training data;
2. build an ngram counter, e.g. `counter = count_ngrams(order, vocab, training_texts)`, where **order** is the order of the language model (2 for bigram), **vocab** is the vocabulary object and **training_texts** is a list of the training sentences. Each training sentences are automatically padded on the left with `<s>` and on the right with `<s/>`.
3. build the language model from the ngram counter object, e.g. `lm = lm = MLENgramModel(counter)`. There are currently three available estimators: **MLE**, **Laplace** (add-1) and **Lidstone** (add-$\alpha$).

### Exercise 4

Use **MLENgramModel** to build a language model based on the text of Sense and Sensibility by Jane Austen. The language model should be a bigram model. The model uses the MLE estimator. *Now uncomment the test code and check your results.*

## 4.2 Computing Probabilities

Using the language model, we can work out the probability of a word given its context. In the case of the bigram language model built in Exercise 4, we have only one word of context. To obtain probabilities from a language model, use **NgramModel.prob**:

```
lm.prob(word,[context])
```

Where **word** and **context** are both unigram strings when working with a bigram language model. For higher order language models, context will be a list of unigram strings of length order-1.

### Exercise 5

Using the bigram language model built in Exercise 4, compute the following probabilities
(a) `reason` followed by `for`
(b) `the` followed by `end`
(c) `end` followed by `the`
*Now uncomment the test code and check your results.*
The result for (c) above is perhaps not what you expected. Why do you think it happened?

# 5 Smoothing

Zero probabilities for unseen n-grams cause problems. Suppose for example you take a bigram language model and use it to score an automatically generated sentence of 10 tokens (say the output of a machine translation system). If one of the bigrams in that sentence is unseen, the probability of the sentence will be zero.

Smoothing is a method of assigning probabilities to unseen n-grams. As language models are typically trained using large amounts of data, any n-gram not seen in the training data is probably unlikely to be seen in other (test) data. A good smoothing method is therefore one that assigns a fairly small probability to unseen n-grams.

## 5.1 MLE vs. Laplace vs. Lidstone

We'll implement these three methods ourselves including two different smoothing methods: **Laplace** (add-one) and **Lidstone** (add-$\alpha$). But first, we will try the implementation provided in NLTK.

### Exercise 6

Beside MLE, try the other estimators which *do* smoothing, and see what happens to all three of the bigram probabilities.

## 5.2 Maximum-Likelihood estimation

Before implementing any smoothing, you should make sure you understand how to implement maximum likelihood estimation. We've used NLTK to do this for us by training a bigram language model with an MLE estimator. We could then use the language model to find the MLE probability of any word given its context. Here, you'll do the same thing but without using NLTK, to make sure you understand how. We will also compare the smoothed probabilities you compute later to these MLE probabilities.

### Exercise 7

Code has been provided that extracts all the words from Jane Austen's "Sense and Sensibility", and then computes a list of bigram tuples by pairing up each word in the corpus with the following word. Using these unigrams and bigrams, fill in the remaining code to compute the MLE probability of a word given a single word of context. Then uncomment the test code to compute the probabilities:

(a) a. $P_{MLE}$("end" | "the")
(b) b. $P_{MLE}$("the" | "end")

## 5.3 Laplace (add-1)

Laplace smoothing adds a value of 1 to the sample count for each "bin" (possible observation, in this case each possible bigram), and then takes the maximum likelihood estimate of the resulting frequency distribution.

### Exercice 8

Assume that the size of the vocabulary is just the number of different words observed in the training data (that is, we will not deal with unseen words). Add code to the template to compute Laplace smoothed probabilities. Now uncomment the test code and look at the estimates for:

(a) a. $P_{+1}$("end" | "the")
(b) b. $P_{+1}$("the" | "end")

How do these probabilities differ from the MLE estimates?

## 5.4 Lidstone (add-$\alpha$)

In practice, Laplace smoothing assigns too much mass to unseen n-grams. The Lidstone method works in a similar way, but instead of adding 1, it adds a value between 0 and 1 to the sample count for each bin.

### Exercise 9

Fill in the code to compute Lidstone smoothed probabilities, then uncomment the test code and look at the probability estimates that are computed for the same bigrams as before using various values of alpha. What do you notice about using $\alpha = 0$ and $\alpha = 1$? (Compare to the probabilities computed by the previous methods.) What about when alpha = 0.01? Are the estimated probabilities more similar to MLE or Laplace smoothing in this case?

## 6 Author Identification

### 6.1 Cross-entropy

In language modeling, a model is trained on a set of data (i.e. the training data). The cross-entropy of this model may then be measured on a test set (i.e. another set of data that is different from the training data) to assess how accurate the model is in predicting the test data. Another way to look at this is: if we used the trained model to generate new sentences by sampling words from its probability distribution, how similar would those new sentences be to the sentences in the test data? This interpretation allows us to use cross-entropy for authorship detection, as described below.

### Exercise 10

We can use cross-entropy in authorship detection. For example, suppose we have a language model trained on Jane Austen's "Sense and Sensibility" (training data) plus the texts for two other novels (test data), one by Jane Austen and one by another author, but we don't know which is which. We can work out the cross-entropy of the model on each of the texts and from the scores, determine which of the two test texts was more likely written by Jane Austen.

Use:

- A trigram language model with a Lidstone probability distribution, trained on Jane Austen's "Sense and Sensibility" (austen-sense.txt)
- text a: austen-emma.txt (Jane Austen's "Emma")
- text b: chesterton-ball.txt (G.K. Chesterton's "The Ball and Cross")
- The langauge model's entropy function: **lm.entropy(...)**

Compute both the total cross-entropy and the per-word cross entropy of each text. (Separate function templates are provided.) *Now uncomment the test code and check your results.* Which text has higher total cross entropy? Which has higher per-word cross entropy? What can these comparisons tell us about the authorship of the two texts?

## 7 Katz Back-Off Estimation

In this section we will implement the Katz back-off estimator.

### Exercise 11

Implement a python class called **KatzLangageModel** that implement Katz back-off smoothing.