FNLP: Lab Session 1

Corpora and Language Models

1 Aim

The aims of this lab session are to 1) explore the different uses of language in different documents, authored by different people and 2) introduce the construction of language models using Python's Natural Language Tool Kit (NLTK). Successful completion of this lab is important as the first assignment for FNLP builds on some of the concepts and methods that are introduced here. By the end of this lab session, you should be able to:

- Access the corpora provided in NLTK
- Compute a frequency distribution
- Train a language model
- Use a language model to compute bigram probabilities

2 Running NLTK and Python Help

2.1 Running NLTK

NLTK is a Python module, and therefore must be run from within Python. To get started on DICE, type the following in a terminal window:

```
$: python
>>> import nltk
```

2.2 Python Help

Python contains an inbuilt help module that runs in an interactive mode. To run the interactive help, type:

```
>>> help()
```

To exit, press CTRL-d.

If you know the name of the module that you want to get help on, type:

```
>>> import <module_name>
>>> help(<module_name>)
```

To exit, type `q` (for `quit`).

If you know the name of the module **and** the method that you want to get help on, type:

```
>>> import <module_name>
>>> help(<module_name>.<method_name>)
```

To exit, type `q` (for `quit`).

3 Introduction

The FNLP lab sessions 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 continuing with this lab sheet, please **download a copy of the lab template** (lab1.py) for this lab from the FNLP 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.

If you're unfamiliar with developing python code, you may want to look at the second lab for ANLP, which assumes much less background experience and has a detailed step-by-step guide to using python for the first time:

http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab2.html

4 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:

```
>>> 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')
```

Exercise 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*

Exercise 2
Find the average word type length of Obama's 2009 speech *Now uncomment the test code and check your results*

5 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

6 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 find the most fluent/natural translation from the set of candidates.

6.1 Building a Language Model

NLTK provides the **NgramModel** module for building language models. The initialisation method looks like this:

```
def __init__(self, n, train, pad_left=False, pad_right=False,
             estimator=None, *estimator_args, **estimator_kwargs):
```

Where:

- **n** = order of the language model. 1=unigram; 2=bigram; 3=trigram, etc.
- **train** = the training data (supplied as a list)
- **pad_left** and **pad_right** = sentence initial and sentence final padding
- **estimator** = method used to construct the probability distribution. May or may not include smoothing. Arguments to the estimator are optional.

Exercise 4
Use **NgramModel** to build a language model based on the text of *Sense and Sensibility* by Jane Austen. The language model should be a bigram model, and you can let it use the default **nltk.MLEProbDist** estimator.
Hint, fill in the gaps in with the information already provided in the code / comments.
Now uncomment the test code and check your results

6.2 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?

Exercise 6
Update your definition of the **ex5** function to include a (boolean) **verbose** argument, which is passed through to **NgramModel.prob**. Use this to see if it gives any insight on the `(end, the)` bigram.

7 Going further

7.1 Smoothing

Try using an estimator which *does* do smoothing, and see what happens to all three of the bigram probabilities. Try `help(NgramModel)` for help with the operation of this class and how to supply estimators.

```
>>> from nltk import probability
>>> help(probability)
```

to see what estimators are available.

7.2 Tokenisation Padding

So far you've treated the data as a flat list of 'words', which doesn't fully address the place of words within sentences. Using `gutenberg.sents(...)` explore the impact of the `pad_left` and `pad_right` argument to `NgramModel`. Compare the following:

```
>>> lm.prob('The',['<s>'])
>>> lm.prob('the',['<s>'])
>>> lm.prob('End',['<s/>'])
>>> lm.prob('end',['<s/>'])
>>> lm.prob('.',['<s/>'])
```

7.3 Costs vs. probabilities

Redo the previous two sub-sections using *costs* instead of probabilities.

```
>>> help(lm.logprob)
```