NLP Lab Session Week 2
January 28, 2010

Processing Text, Frequency Distributions, and Bigram Distributions

Installing NLTK Toolkit

Reinstall nltk-2.0b7.win32.msi and Copy and Paste nltk_data from H:\ nltk_data to C:\ nltk_data

Processing Text

In addition to the examples that we used last week that were especially for the NLTK book, the NLTK has a number of other corpora, described in Chapter 2.

Start your Python IDLE window.

Type in

```
>>> import nltk
```

You can then view some books obtained from the Gutenberg on-line book project:
```
>>> nltk.corpus.gutenberg.fileids()
```

For purposes of this lab, we will work with the first book, Jane Austen's "Emma".
```
>>> file1 = nltk.corpus.gutenberg.fileids( ) [0]
>>> file1
```

We can get the original text, using the raw function:
```
>>> len(nltk.corpus.raw(file1))
>>> emmatext = nltk.corpus.gutenberg.raw(file1)
```

Since this is quite long, we can view part of it, e.g. the first 120 characters
```
>>> emmatext[:120]
```

NLTK has several tokenizers available to break the raw text into tokens; we will use one that separates by white space and also by special characters (punctuation):
```
>>> emmatokens = nltk.wordpunct_tokenize(emmatext)
>>> len(emmatokens)
>>> emmatokens[:50]
```

We probably want to use the lowercase versions of the words:
```
>>> emmawords = [w.lower( ) for w in emmatokens]
>>> emmawords[:50]
>>> len(emmawords)
```

We can further view the words by getting the unique words and sorting them:
```
>>> emmavocab = sorted(set(emmawords))
>>> emmavocab[:50]
```

We can certainly see that we will want to get rid of these special characters – Regular Expressions to the Rescue! (as in xkcd _), but we'll work on that a little later.

Frequency Distributions

NLTK has a set of functions that use a data structure called a Frequency Distribution, FreqDist. This structure is an extension of the Python dictionary structures. These are described in the NLTK book, at the end of Chapter 1. One way to set up access to FreqDist is to:

```
>>> from nltk.book import *
```

One way to make a Frequency Distribution, is to create one with a list of words. It will do all the counting for you and create a distribution in which the set of keys are all the words, and the set of values are the frequency (count) of each word. The keys( ) function produces the list of words in order of decreasing frequency.

```
>>> fdist = FreqDist(emmawords)
>>> fdist.keys( )[:50]
```

We can look at the frequencies of individual words:

```
>>> fdist['emma']
>>> fdist['the']
```

For the development in the rest of the lab, it will be convenient to use a shorter version of this text. We can create a list with the 100 words following the title and author.

```
>>> shortwords = emmawords[11:111]
>>> shortwords
```

Let's create a frequency distribution of these words:

```
>>> shortdist = FreqDist(shortwords)
>>> shortdist.keys( )
>>> shortdist['the']
>>> shortdist['emma']
```

Exercise 1 (will not be submitted). Create a frequency distribution for the words in text1 from the NLTK book (Moby Dick).
Key steps:
1. Make a short version of text1, e.g. the first 100 words, text1[0:99] – assign this to a variable.
2. Create a frequency distribution. Print out the keys and pick several words and look at the frequencies.

Bigram distribution

Next we are going to create a frequency distribution to count all the bigrams of a set of words. Since these examples will be longer, you mayl want to start keeping the examples in a separate file that you can copy and paste into the IDLE window. These examples will be written as functions, described in chapter 1 of the book, under the section on Counting Vocabulary. Later, we will collect all the function definitions into one file.

One way to create the bigram frequencies is to start with an empty Frequency Distribution, iterate over the words pairs (bigrams) and increment the count of each word pair in the distribution.

```
>>> def bigramDist(words):
        biDist = FreqDist()
        for i in range(1, len(words)):
                biword = words[i-1] + ' ' + words[i]
                biDist.inc(biword)
        return biDist
```

```
>>> shortbidist = bigramDist(shortwords)
>>> shortbidist
>>> shortbidist.keys()
>>> shortbidist.max( )
```

Here's another way to view the distribution with keys and values:

```
>>> for pair in shortbidist.keys():
        print pair, shortbidist[pair]
```

Or just the ones with frequency greater than 1:

```
>>> for pair in shortbidist.keys():
        if shortbidist[pair] > 1:
                print pair, shortbidist[pair]
```

Regular Expressions

Now we'll tackle the problem with getting rid of the words which are really just punctuation or special symbols. For this, we use a regular expression pattern that will match word that contains any character which is not lowercase alphabetical. We first must import the regular expression package from Python (re).

```
>>> import re
```

Define a pattern p that will match any words that contains non-alphabetical characters.

```
>>> p = re.compile('.*[^a-z].*')
```

The function p.match tests if a word matches, e.g. the symbol "-" matches because it is non-aphabetical.

```
>>> m = p.match('-')
>>> if m: 'yes, it matches'
```

We'll add this to our bigram distribution to select only alphabetic words.

```
>>> def bigramDistA(words):
        biDistA = FreqDist()
        p = re.compile('.*[^a-z].*')
        for i in range(1, len(words)):
                m0 = p.match(words[i-1])
                m1 = p.match(words[i])
                if m0 or m1: continue
                biwordA = words[i-1] + ' ' + words[i]
                biDistA.inc(biwordA)
        return biDistA
>>> shortbidistA = bigramDistA(shortwords)
>>> shortbidistA.keys( )
```

Now let's test this function on all the Emma text:

```
>>> bigDist = bigramDistA(emmawords)
>>> bigDist.keys()[:99]
>>> bigDist['to be']
```

This shows another problem, which is that our most common words are all the simple words. One way to work around this is to make a list of common words to ignore: the stop word list. Here is a simple start to a stop word list.

```
>>> stopwords = ['to', 'be', 'of', 'the', 'in', 'it', 'was', 'i', 'am', 'she', 'had', 'been', 'is', 'have', 'could',
'not', 'her', 'he', 'do', 'and', 'would', 'such', 'a', 'his', 'must']
```

When we want to add stop words to the list, we can use append for one word, or extend for a list of words.

```
>>> stopwords.append('all')
>>> stopwords
```

In our next version of the bigram distributions, we will not count any bigram that contains a word on the stop word list

```
>>> def bigramDistAS(words, stoplist):
        biDistAS = FreqDist()
        p = re.compile('.*[^a-z].*')
        for i in range(1, len(words)):
                m0 = p.match(words[i-1])
                m1 = p.match(words[i])
                m2 = words[i-1] in stoplist
                m3 = words[i] in stoplist
                if m0 or m1 or m2 or m3: continue
                biwordAS = words[i-1] + ' ' + words[i]
                biDistAS.inc(biwordAS)
        return biDistAS
>>> shortbiDistAS = bigramDistAS(shortwords, stopwords)
>>> shortbiDistAS.keys()
```

As a final version of our bigram distribution functions, we can add the idea of having a frequency threshold; that is, we will not count any bigram that contains a word that is infrequent in the corpus, where infrequent means that its frequency is less than the threshold.

```
def bigramDistAST(words, stoplist, threshold):
        biDistAST = FreqDist()
        uniDist = FreqDist(words)
        p = re.compile('.*[^a-z].*')
        for i in range(1, len(words)):
                m0 = p.match(words[i-1])
                m1 = p.match(words[i])
                m2 = words[i-1] in stoplist
                m3 = words[i] in stoplist
                m4 = uniDist[words[i-1]] < threshold
                m5 = uniDist[words[i]] < threshold
                if m0 or m1 or m2 or m3 or m4 or m5: continue
                biwordAST = words[i-1] + ' ' + words[i]
                biDistAST.inc(biwordAST)
        return biDistAST

>>> shortbiDistAST = bigramDistAST(shortwords, stopwords,2)
>>> shortbiDistAST.keys()
['with very']
```

Now we can test this on our entire text of Emma.

```
>>> emmaDistAST = bigramDistAST(emmawords, stopwords, 5)
>>> emmaDistAST.keys()[:50]
```

Or we can view the key, frequency pairs for the top frequency keys

```
>>> for k in emmaDistAST.keys( )[:50]:
        print k, emmaDistAST[k]
```

This bigram frequency function is what we need to use in the Association Ratio, sometimes also called Mutual Information, that we will use in Homework 1.

Exercise2

For this exercise, you may work in groups of 2-3, or you can choose to do your own work.

Choose a file that you want to work on, either one of the files from the book corpus, or one from the Gutenberg corpus.

- First, define a variable that contains the list of words in your example.
- Run the different bigram frequency distribution functions on your corpus and look at the top 20 keys from each of them: BigramDist, BigramDistA, BigramDistAS, BigramDistAST.

Do you see any improvements that should be made to these distributions?

To complete the exercise, choose one of your top 20 frequency lists to report to show to post. There should be only one post per group. Write an introductory sentence or paragraph telling what text you chose and what bigram distribution function you chose. Put this and the frequency list in a discussion posting in the iLMS system under IST 664. Be sure to include the names of the people in your group, preferably also in the title of the post.