NLP Lab Session Week 4
February 11, 2010

Reading files, Stemming and Lemmatization

Installing NLTK Toolkit

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

Getting Started

In this lab session, we will use two saved files of python commands and definitions first, and then also work together through a series of small examples using the IDLE window and that will be described in this lab document. All of these examples can be found in python files on the iLMS system, under Resources.

Bigram.py, processtextfile.py, Labweek4examples.py, desert.txt, Smart.English.stop

Put all of these files in one directory and open an IDLE window.

Reading Text from a File

To make it easier to use the bigram distribution functions that we have defined, they are all placed into one python file, which python would call a Module. The name of the file is Bigram.py and if the file is in the same directory as another Python program, you can say "import Bigram" to access the function definitions. Then you use the name of the module to use the function names, for example, Bigram.assocratio2.

In your IDLE window, use the File menu to open the Bigram.py file and also processtextfile.py.

Read through the processtextfile.py and observe how it reads text from desert.txt and how it creates a stop word list from the file Smart.English.stop. Note that is calls the assocratio2 function, which computes a bigram distribution with the association ratio measure computed in a window of 2.

Run the processtextfile by selecting Run Module from the Run menu. The results will appear in your main IDLE window. Try the association ratio with and without stopwords, by using a comment character # to switch between the two statements. Also try thresholds of 3, and perhaps 5.

Stemming and Lemmatization

For this part, we will use raw text from the Gutenberg Corpus as we did before. You can find all these examples in Labweek4examples.py. Here are the steps to get set up to use the text from the first document, Emma.

```
import nltk
nltk.corpus.gutenberg.fileids( )

file0 = nltk.corpus.gutenberg.fileids( ) [0]
emmatext = nltk.corpus.gutenberg.raw(file0)
emmatokens = nltk.wordpunct_tokenize(emmatext)
emmawords = [w.lower( ) for w in emmatokens]
```

Note that emmatokens has words with regular capitalization and emmawords has lower-case words with no capitalization.

NLTK has two stemmers, Porter and Lancaster, described in section 3.6 of the NLTK book. To use these stemmers, you first create them.

```
porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()
```

First, we'll compare how the stemmers work using both the regular-cased text and the lower-cased text.

```
emmaregstem = [porter.stem(t) for t in emmatokens]
emmaregstem[1:100]

emmalowerstem = [porter.stem(t) for t in emmawords]
emmalowerstem[1:100]
```

Try the same examples with the Lancaster stemmer.

The NLTK suggests that we try building our own simple stemmer by making a list of suffixes to take off.

```
def stem(word):
    for suffix in ['ing', 'ly', 'ed', 'ious', 'ies', 'ive', 'es', 's', 'ment']:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word

#try the above stemmer with 'friends'
stemmedword=stem('friends')
stemmedword
```

The NLTK has a lemmatizer that uses the WordNet on-line thesaurus as a dictionary to look up roots and find the word.

```
wnl = nltk.WordNetLemmatizer()
emmalemma=[wnl.lemmatize(t) for t in emmawords]
emmalemma[1:100]
```

Regular Expression Tokenizer

So far, we have depended on the NLTK wordpunct tokenizer for our tokenization. Not only does the NLTK have other tokenizers, but we can custom-build our own tokenizer if we wish. Here is a regular expression that will match any sequence of word characters, or anything that starts with a non-space character optionally followed by some word characters. We can use the regular expression function findall to apply this pattern to text:

```
import re
p = re.compile('\w+|\S\w*')
emmatext[:50]
re.findall(p,emmatext[:50])
```

But NLTK has built a tokenizing function that helps you do this by just giving it the compiled pattern. Regular expressions can also be written down in the “verbose” version that allows the alternatives to be on different lines with comments.

```
pattern = r''' (?x)         # set flag to allow verbose regexps
        ([A-Z]\.)+          # abbreviations, e.g. U.S.A
        | \w+(-\w+)*        # words with internal hyphens
        | \$?\d+(\.\d+)?%?    # currency and percentages, $12.40, 50%
        | \.\.\.        # ellipsis
        | [][.,;"'?():-_']    # separate tokens
        '''
```

We can test this on Emma and also on some text with additional interesting tokens.

```
nltk.regexp_tokenize(emmatext[:100], pattern)
nltk.regexp_tokenize('That U.S.A. poster-print costs $12.40', pattern)
```

Exercise

Run the regexp tokenizer on the sentence “Mr. Black and Mrs. Brown attended the lecture by Dr. Gray.” Design and add a line to the pattern of this tokenizer so that titles like “Mr.” are tokenized as having the dot inside the token. Can you think of other examples of titles or other things that this tokenizer doesn’t catch?

Post your title tokenizer line to the Discussion in the iLMS for Week 4 (they will most likely all be very similar) and any examples that you think of that need additional regular expressions to be tokenized.