DEPARTMENT *of* COMPUTING

College of Business & Technology

EAST TENNESSEE STATE UNIVERSITY

CSCI 5260 – Artificial Intelligence

Lab 12 – Natural Language Processing

Overview

The most commonly used Python library for Natural Language Processing is nltk (Natural Language Tool Kit). This lab will serve as a tutorial for using many of the common features of nltk. This tutorial will compare two works by Mary Shelley: *Frankenstein* (or the *Modern Prometheus*) [1831] and *The Last Man* [1826].

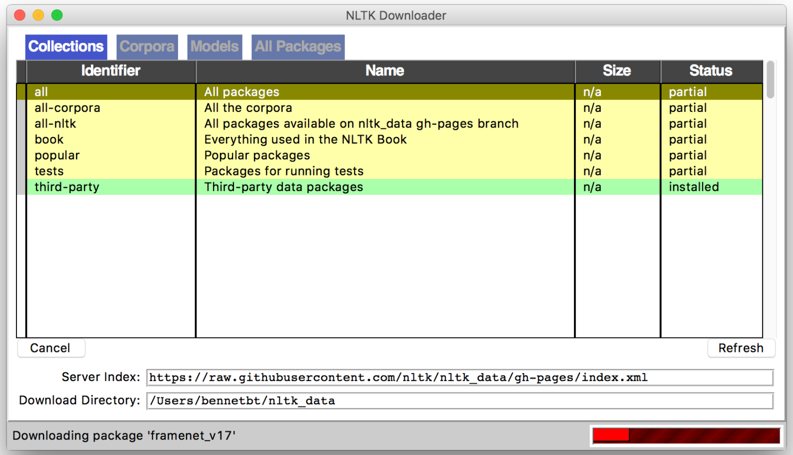
NLTK Downloader

1. At a command prompt, type `conda install nltk` to ensure NLTK is installed on your system. If it is already installed, run `conda upgrade nltk`.
2. Open a new Python file and name it **lab12.py**. In the file, type the following code, save it, and run it:

```
import nltk

nltk.download()
```

3. The following dialog box will appear, and will allow you to download NLTK packages:



Identifier	Name	Size	Status
all	All packages	n/a	partial
all-corpora	All the corpora	n/a	partial
all-nltk	All packages available on nltk_data gh-pages branch	n/a	partial
book	Everything used in the NLTK Book	n/a	partial
popular	Popular packages	n/a	partial
tests	Packages for running tests	n/a	partial
third-party	Third-party data packages	n/a	installed

Server Index: `https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml`

Download Directory: `/Users/bennetbt/nltk_data`

Downloading package 'framenet_v17'

When you need to download NLTK packages, you may want to use this technique. However, you may also install packages directly via `nltk.download('package-name')`.

Part 1: Reading and Tokenization

Step 1 – Reading a Corpus from the Web

1. In lab12.py, add the following code that pulls a web page and prints it to the screen.

```
import urllib.request


print("Reading Text and Tokenizing...")

response = urllib.request.urlopen(https://www.gutenberg.org/files/42324/4232
4-h/42324-h.htm')

html = response.read()

print (html)
```

Notice that the output has many HTML tags and special characters that need to be removed.

Step 2: Cleaning HTML

2. To clean the HTML from the text, use the BeautifulSoup library as follows:

```
from bs4 import BeautifulSoup

import urllib.request


print("Reading Text and Tokenizing...")

response = urllib.request.urlopen('https://www.gutenberg.org/files/42324/423
24-h/42324-h.htm')

html = response.read()

soup = BeautifulSoup(html,"html5lib")

text = soup.get_text(strip=True)

print (text)
```

This displays a document that has been cleansed of HTML tags and is ready to be tokenized.

Step 3: Tokenization

3. To convert the text into tokens, simply split it using the Python split command, as follows:

```
from bs4 import BeautifulSoup
import urllib.request

print("Reading Text and Tokenizing...")

response = urllib.request.urlopen('https://www.gutenberg.org/files/42324/423
24-h/42324-h.htm')
html = response.read()
soup = BeautifulSoup(html,"html5lib")
text = soup.get_text(strip=True)
tokens = [t for t in text.split()]
print (tokens)
```

4. Now, use the NLTK built-in tokenizers and look at how they tokenize the document differently:

```
from nltk.tokenize import sent_tokenize, word_tokenize
...
s_tokens = sent_tokenize(text)
w_tokens = word_tokenize(text)
```

Part 2: Removing Punctuation

Punctuation marks can mess things up when doing NLP. So, you should likely get rid of them. To do this, you should use the RegexpTokenizer in NLTK to ignore all non-alphanumeric characters. In this instance, we're also going to make everything lowercase.

```
print("Removing Punctuation...")
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r'\w+')
tokens = tokenizer.tokenize(text)
tokens = [t.lower() for t in tokens]
```

Part 3: Counting Words and Eliminating Stop Words

Step 1: Obtaining Word Counts

1. Complete the following code to count instances of each number. Take note of the number of items in the dictionary:

```
print("Frequency Analysis...")
freq = nltk.FreqDist(tokens)     # lowercase, non-punctuated tokens
for key,val in freq.items():
    print (str(key) + ':' + str(val))
print("Length of Unique Items:", len(freq.items()))
freq.plot(20, cumulative=False)
```

This code will plot a graph of the 20 most frequently used words. In the graph, you likely will see the, a, and, to, was, etc. as the most common words. Obviously this is not going to be the best words for analysis. These are called “stop words” and are often removed for further analysis.

Step 2: Stop Word Removal

2. To get rid of the stop words, use the English stop words list from the NLTK corpus library. To do this, your code should look as follows:

```
from bs4 import BeautifulSoup
import urllib.request
import nltk
from nltk.corpus import stopwords

print("Reading Text and Tokenizing...")
response = urllib.request.urlopen('https://www.gutenberg.org/files/42324/423
24-h/42324-h.htm')
html = response.read()
soup = BeautifulSoup(html,"html5lib")
text = soup.get_text(strip=True)
tokens = [t for t in text.split()]

import string

print("Removing Punctuation...")
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r'\w+')
tokens = tokenizer.tokenize(text)
tokens = [t.lower() for t in tokens]

print("Removing Stop Words...")
clean_tokens = tokens[:]
sr = stopwords.words('english')
for token in tokens:
    if token in stopwords.words('english'):
        clean_tokens.remove(token)

print("Frequency Analysis...")
```

```
freq = nltk.FreqDist(clean_tokens)
for key,val in freq.items():
    print (str(key) + ':' + str(val))
print("Length of Unique Items:", len(freq.items()))
freq.plot(20, cumulative=False)
```

Note that stop word removal can take a bit of time to complete. It is walking through all tokens.

Part 4: Stemming and Lemmatization

Step 1: Stemming

Look through the frequency list. You'll notice that you have instances of the same word, but with different endings. Stemming removes the endings of the words so they can be more easily counted. To perform stemming, we will use NLTK's PorterStemmer. You may also use the SnowballStemmer. Add the following code after your stopword removal, and before your frequency analysis. Be sure to change your frequency distribution to stemmed_tokens instead of clean_tokens.

```
. . .
print("Stemming...")
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(token) for token in clean_tokens]

print("Frequency Analysis...")
freq = nltk.FreqDist(stemmed_tokens)
. . .
```

Step 2: Lemmatization

Lemmatization is similar to stemming. Stemming may chop the end of a word off, resulting in a nonsense word. The result of lemmatization is always a real word. To do this in Python, add the following just after your stemming code:

```
print("Lemmatizing...")
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(token) for token in clean_tokens]

print("Frequency Analysis...")
freq = nltk.FreqDist(lemmatized_tokens)
```

Part 5: Parts of Speech Analysis

Sometimes, you would like to analyze the parts of speech within the document. Let's build a parts-of-speech tagger that will assign the POS to each token. Place the following snippit at the end of your code. You should probably comment out the `freq.plot(20, cumulative=False)` line. While this information may not be useful in a format displayed here, it can be very useful in sentence analysis.

```
print("POS Analysis...")
import operator
import matplotlib.pyplot as plt
pos = nltk.pos_tag(lemmatized_tokens)
pos_counts = {}
for key,val in pos:
    print(str(key) + ':' + str(val))
    if val not in pos_counts.keys():
        pos_counts[val] = 1
    else:
        pos_counts[val] += 1

print(pos_counts)
plt.bar(range(len(pos_counts)), list(pos_counts.values()), align='center')
plt.xticks(range(len(pos_counts)), list(pos_counts.keys()))
plt.show()
```

PART 6: N-GRAMS

An n-gram is a grouping of n terms for a single token. Often, single words will not produce an appropriate way to analyze a document. Therefore, people use n-grams to gain further insights. In the following code, you will tokenize the code using tri-grams (3-grams). N-grams can be used as tokens instead of single words. Note that you may want to comment out `plt.show()`.

```
print("Tri-Grams...")
from nltk import ngrams
trigrams = ngrams(text.split(), 3)
for gram in trigrams: print(gram)
```

PART 7: DOCUMENT-TERM MATRIX

NLP often requires the analysis of multiple documents. There are two ways to analyze multiple documents: using a Document-Term Matrix or a Term-Document Matrix. DTMs have the documents on the left, the terms across the top, and counts in the center. TDMs are the transposition of DTMs. Unfortunately, NLTK doesn't include a method for doing DTMs and TDMs. To do this, you have to use one of Python's Machine Learning libraries, scikit-learn, and pandas, a data analysis library. At the end of your code file, add the following (you may want to comment out some of the unnecessary steps above it to improve runtime):

```
print("Document-Term Matrix...")
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

response = urllib.request.urlopen('https://www.gutenberg.org/cache/epub/1824
7/pg18247.html')
html = response.read()
soup = BeautifulSoup(html,"html5lib")
text2 = soup.get_text(strip=True)

docs = [text, text2]
vec = CountVectorizer()
X = vec.fit_transform(docs)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
print("Instances of 'fear' in both documents:")
print(df["fear"])    # Show the count for this word in both documents

print("Instances of 'hope' in both documents:")
print(df["hope"])    # Show the count for this word in both documents
print(df)            # Show the full data frame
```

SUBMISSION

Submit your completed **lab12.py** file.

Submit to the Lab 12 dropbox at or before Monday, April 26, 2021 by 11:59 PM.

Grading

A letter grade will be assigned for each response. The letter grades are based on both correctness and the adequacy of answers. Points are assigned as follows:

	A	B	C	D	F	Zero
	Excellent	Above Average	Average	Below Average	Poor	No Attempt
	10	8	6	4	2	0
Part 1-1: Reading Corpus						
Part 1-2: Cleaning HTML						
Part 1-3: Tokenization						
Part 2: Punctuation						
Part 3-1: Frequency Analysis						
Part 3-2: Stop Words						
Part 4-1: Stemming						
Part 4-2: Lemmatization						
Part 5: Parts of Speech						
Part 6: N-Grams						
Part 7: Document-Term Matrix						