Natural Language Processing with Python

Lab Session-I

Dr. Jasmeet Singh
Assistant Professor,
CSED, TIET

Introduction

- Technologies based on NLP are becoming increasingly widespread.
- For example,phones and handheld computers support predictive text and handwriting recognition;
- web search engines give access to information locked up in unstructured text;
- Machine translation allows us to retrieve texts written in English and read them in Spanish.
- Python programming language provides an open source library called the Natural Language Toolkit (NLTK) for computational linguistics.

NLTK

- NLTK was originally created in 2001 as part of a computational linguistics course in the Department of Computer and Information Science at the University of Pennsylvania.
- Since then it has been developed and expanded with the help of dozens of contributors.
- NLTK includes extensive software, data, and documentation, all freely downloadable from http://www.nltk.org/.
- Distributions are provided for Windows, Macintosh, and Unix platforms.

NLTK Modules

- The most important NLTK modules are:

Language processing task	NLTK modules	Functionality
Accessing corpora	nltk.corpus	Standardized interfaces to corpora and lexicons
String processing	nltk.tokenize, nltk.stem	Tokenizers, sentence tokenizers, stemmers
Collocation discovery	nltk.collocations	t-test, chi-squared, point-wise mutual information
Part-of-speech tagging	nltk.tag	n-gram, backoff, Brill, HMM, TnT
Classification	nltk.classify, nltk.cluster	Decision tree, maximum entropy, naive Bayes, EM, k-means
Chunking	nltk.chunk	Regular expression, n-gram, named entity
Parsing	nltk.parse	Chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	nltk.sem, nltk.inference	Lambda calculus, first-order logic, model checking
Evaluation metrics	nltk.metrics	Precision, recall, agreement coefficients

Installing NLTK

- NLTK is downloadable for free from http://www.nltk.org/
- It can be directly installed in terminal using following command

  sudo pip install nltk
- pip is used to download and install packages directly from PyPI (Python Package Index).
- PyPI is hosted by Python Software Foundation. It is a specialized package manager that only deals with python packages.

Installing NLTK Contd….

- Once we have installed NLTK, start up the Python interpreter by simply typing *python.*
- The NLTK package is then included using the following command.
- import nltk
- Install the data required by typing the following command:
- nltk.download( )
- Install the collection book (to obtain all data required for the examples and exercise. It consists of about 30 compressed files requiring about 100Mb disk space)

Installing NLTK Contd….

- Once the data is downloaded to our machine, we can load it using the Python interpreter
- from nltk.book import *
- (from NLTK's book module, load all items)
- Any time we want to find out about these texts, we just have to enter their names at the Python prompt:
- text1

Searching Text

- There are many ways to examine the context of a text apart from simply reading it.
- A concordance view shows us every occurrence of a given word, together with some context. A concordance permits us to see words in context
- We can look up the word monstrous in Moby Dick by entering text1 as follows:
- text1.concordance("monstrous")
- Similarly in other texts we can check for following words:
- text2.concordance("affection")
- text3.concordance("lived")

Searching Text Contd…..

- A similar view permits us to see what words appear in a similar range of contexts.
- We can find out by appending the term similar to the name of the text in question, then inserting the relevant word in parentheses:
- text1.similar("monstrous")

Searching Text Contd…..

- The *common_contexts* view allows us to examine just the contexts that are shared by two or more words, such as monstrous and very.
- We have to enclose these words by square brackets as well as parentheses, and separate them with a comma:
- text2.common_contexts(["monstrous", "very"])

Counting Vocabulary

The length of a text from start to finish, in terms of the words and punctuation symbols can be computed using *len* command as follows:
len(text3)

To find unique words in the text we use *set* command as follows
set(text1)

The list of unique terms can be sorted using *sorted* command as follows:
sorted(set(text1))

The number of unique words can be counted by using *len and set* command as follows:
len(set(text1))

Counting Vocabulary Contd….

- The frequency of any word in the text can be computed using *count* command as follows*:*
- text1.count("the")

- The percentage of text taken up by a word can be computed as follows:
- 100 * text1.count("the") / len(text1)

- Lexical richness of the text tells that how many times each word occurs on an average in the text. It is computed as follows:
- len(text2)/len(set(text2))

Counting Vocabulary Contd….

- We can also define function for computing lexical diversity and percentage as follows:

```
def lexical_diversity(text):
…                 return len(text) / len(set(text))
...
def percentage(count, total):
…                 return 100 * count / total
…
```

Texts as Lists of Words

A text in Python is a list of words, represented using a combination of brackets and quotes. For example sentence1=['my','name','is','Jasmeet','.']

Concatenation of lists:

Python's addition operator act as concatenation of lists.

Adding two lists creates a new list with everything from the first list, followed by everything from the second list. For example,

sent4 + sent1

Texts as Lists of Words (Contd...)

Appending text:
Appending text to a list add a single item to a list. It is done through *append* command.
sent1.append("hi")

Indexing Lists:
We can identify the elements of a Python list by their order of occurrence in the list. The number that represents this position is the item's index. For example,
text1[4]

Texts as Lists of Words (Contd...)

We can also find the index (first occurrence) of a word as follows:
text1.index("awaken")

Python permits us to access sublists as well, extracting manageable pieces of language from large texts, a technique known as **slicing**. For example,
text1[20:300]
By convention, m:n means elements m...n-1. we can omit the first number if the slice begins at the start of the list , and we can omit the second number if the slice goes to the end

Strings

The methods like indexing, slicing, concatenation works also on individual words or strings.
For example, name="abcdef" or name='abcdef'
Name[0:4], name[0], name+name, name*5

We can join the words of a list to make a single string, or split a string into a list, as follows:

```
>>> ' '.join(['abcdef', 'Python'])
'abcdef Python'
>>> 'abcdef Python'.split()
['abcdef', 'Python']
```

Frequency Distribution

FreqDist function gives the frequency distribution of the words of a given text.

```
fdist=FreqDist(text1)
vocab=fdist.keys() // gives the unique words of the text
vocab[:50] // top 50 words of the text in order of their occurrence.
fdist.items() //list the frequency of each vocabulary of text1
fdist.plot(50,cumulative=True) //plots the cumulative frequency graph of top 50 words
Fdist.most_common() // gives most frequently occurring words
fdsit.hapaxes() // gives words which occur once only.
fdist['monstrous'] //Count of the number of times a given sample occurred
fdist.freq('monstrous')// Frequency of a given sample
fdist.N()// Total number of samples
fdist.max() //Sample with the greatest count
fdist.tabulate() //Tabulate the frequency distribution
fdist1 < fdist2 //Test if samples in fdist1 occur less frequently than in fdist2
```

Fine Grained Selection of Words

Long Words

We would like to find the words from the vocabulary of the text that satisfy a property P, (say for long words length more than 15 characters)

Now we can express the words of interest using mathematical set notation as shown in (1a).The corresponding Python expression is given in (1b). This means “the set of all w such that w is an element of V (the vocabulary) and w has property P.”

(1) a. $\{w \mid w \in V \ \& \ P(w)\}$

(1) b. [w for w in V if p(w)]

Fine Grained Selection of Words Contd….

```
V = set(text1)
long_words = [w for w in V if len(w) > 15] // give long words of text1
sorted([w for w in set(text5) if len(w) > 7 and fdist5[w] > 7])
// give words in text 5 with length and frequency greater than 7.
```

Counting Distribution of Word lengths

```
[len(w) for w in text1]
fdist = FreqDist([len(w) for w in text1])
fdist.keys()
```

Conditions

[w for w in text if condition] ,
where condition is a Python “test” that yields either true or false.
Numerical Condition Operators
< Less than
<= Less than or equal to
== Equal to (note this is two “ = ”signs, not one)
!= Not equal to
> Greater than
>= Greater than or equal to
Word Comparison Operators
s.startswith(t) Test if s starts with t
s.endswith(t) Test if s ends with t
t in s Test if t is contained inside s
s.islower() Test if all cased characters in s are lowercase

Conditions Contd….

s.isupper() Test if all cased characters in s are uppercase
s.isalpha() Test if all characters in s are alphabetic
s.isalnum() Test if all characters in s are alphanumeric
s.isdigit() Test if all characters in s are digits
s.istitle() Test if s is titlecased (all words in s have initial capitals)
We can also create more complex conditions. If c is a condition, then not c is also a condition. If we have two conditions c 1 and c 2 , then we can combine them to form a new condition using conjunction and disjunction: c 1 and c 2 , c 1 or c 2 .

Operating on Every element
Expressions of the form [f(w) for ...] or [w.f() for ...] , where f is a function operates on each word.
For example, [len(w) for w in text1]
[w.upper() for w in text1]

Looping With Conditions

Example 1

```
for word in sent1:
…                 if word.endswith('l'):
…                         print word
…
```

Example 2

```
for token in sent1:
…       if token.islower():
…               print token, 'is a lowercase word'
…       elseif token.istitle():
…               print token, 'is a titlecase word'
…       else:
…               print token, 'is punctuation'
...
```

NGrams

```
from nltk import ngrams
sentence = 'this is a foo bar sentences and i want to
ngramize it'
n = 6
sixgrams = ngrams(sentence.split(), n)
for grams in sixgrams:
    print (grams)
```

Collocations

```
from nltk.collocations import *
bigram_measures =
nltk.collocations.BigramAssocMeasures()
trigram_measures=nltk.collocations.TrigramAssocMe
asures()
finder = BigramCollocationFinder.from_words(text1)
finder1=TrigramCollocationFinder.from_words(text1)
finder.nbest(bigram_measures.pmi,100)
finder1.nbest(trigram_measures.pmi,100)
```