Natural Language Toolkit [NLTK]

Prakash B Pimpale
pbpimpale@gmail.com

@

FOSS(From the Open Source Shelf)
An open source softwares seminar series

Let's start

- What is it about?
  - NLTK – installing and using it for some 'basic NLP' and Text Processing tasks
  - Some NLP and Some python
- How will we proceed ?
  - NLTK introduction
  - NLP introduction
  - NLTK installation
  - Playing with text using NLTK
  - Conclusion

NLTK

- Open source python modules and linguistic data for Natural Langauge Processing application
- Dveloped by group of people, project leaders: Steven Bird, Edward Loper, Ewan Klein
- Python: simple, free and open source, portable, object oriented, interpreted

NLP

- Natural Language Processing: field of computer science and linguistics woks out interactions between computers and human (natural) languages.
- some brand ambassadors: machine translation, automatic summarization, information extraction, transliteration, question answering, opinion mining
- Basic tasks: stemming, POS tagging, chunking, parsing, etc.
- Involves simple frequency count to undertanding and generating complex text

Terms/Steps in NLP

- Tokenization: getting words and punctuations out from text

- Stemming: getting the (inflectional) root of word; plays, playing, played : play

cont..

POS(Part of Speech) tagging:

Ram NNP

killed VBD

Ravana NNP

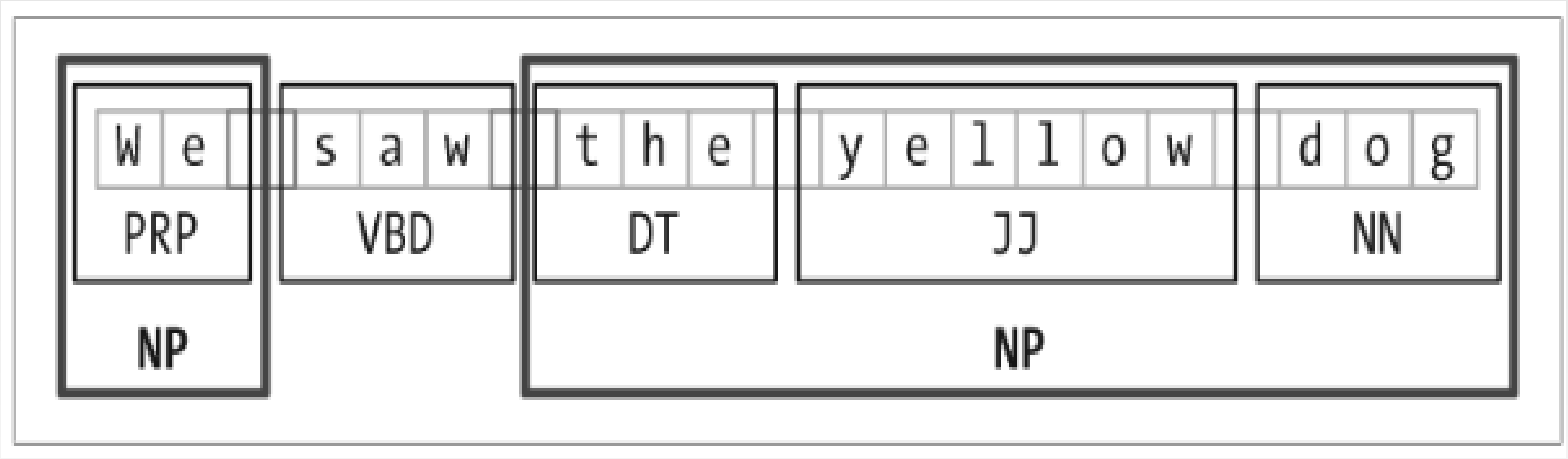
POS tags: NNP- proper noun

VBD verb, past tense

(Penn tree bank tagset)

cont..

- Chunking: groups the similar POS tags (liberal def)



cont..

Parsing: Identifying gramatical structures in a sentence: Mary saw a dog

```
*(S
   (NP Mary)
   **(VP
      (V saw)
      ***(NP
         (Det a)
         (N dog)   )***   )**   )*
```

Applications

- Machine Translation

  MaTra, Google translate
- Transliteration

  Xlit, Google Indic
- Text classification

  Spam filters

  many more.....!

Challenges

- Word sense disambiguation

  Bank(financial/river)

  Plant(industrial/natural)

- Name entity recognition

  CDAC is in Mumbai

- Clause Boundary detection

  Ram who was the king of Ayodhya killed Ravana

  and many more.....

Installing NLTK

Windows

- Install Python: Version 2.4.*, 2.5.*, or 2.6.* and not 3.0

  available http://www.nltk.org/download

  [great instructions!]
- Install PyYAML:
- Install Numpy
- Install Matplotlib
- Install NLTK

cont..

Linux/Unix

- Most likely python will be there (if not-> package manager)
- Get python-nympy and python-scipy packages (scintific library for multidimentional array and linear algebra), use package manager to install or

'sudo apt-get install python-numpy python-scipy'

cont..

NLTK Source Installation

- Download NLTK source (http://nltk.googlecode.com/files/nltk-2.0b8.zip)
- Unzip it
- Go to the new unziped folder
- Just do it!

'sudo python setup.py install'

Done :-) !

Installing Data

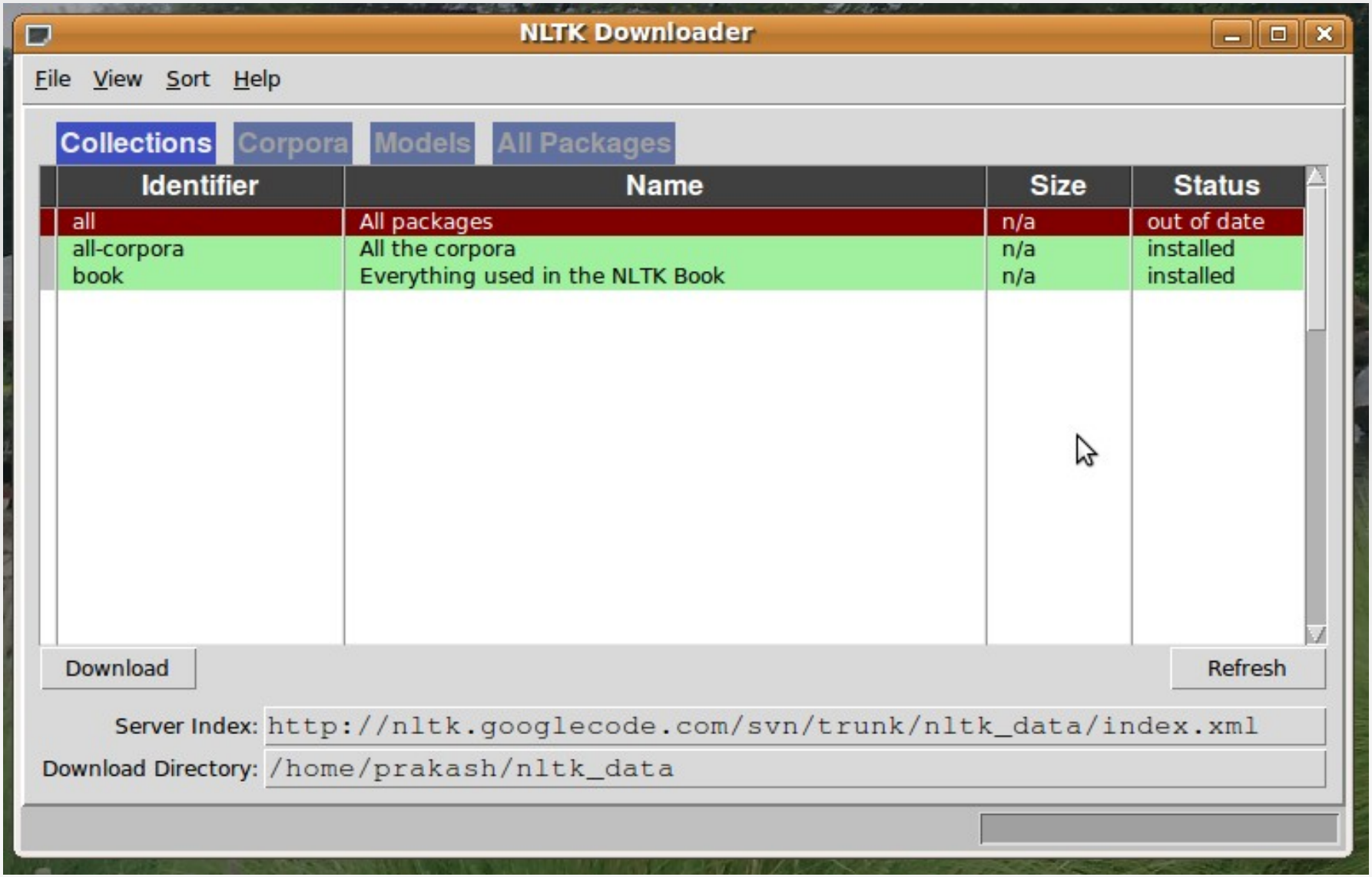
- We need data to experiment with

  It's too simple (same for all platform)...

  ```
  $python          #(IDLE) enter to python prompt
  >>> import nltk
  >>> nltk.download()
  ```

  select any one option that you like/need

cont..


NLTK Downloader
File View Sort Help
Collections Corpora Models All Packages
Identifier Name Size Status
all All packages n/a out of date
all-corpora All the corpora n/a installed
book Everything used in the NLTK Book n/a installed
Download
Refresh
Server Index: http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml
Download Directory: /home/prakash/nltk_data

cont..

Test installation

(set path if needed: as

export NLTK_DATA = '/home/...../nltk_data')

```
>>> from nltk.corpus import brown
>>> brown.words()[0:20]
['The', 'Fulton', 'County', 'Grand', 'Jury']
```

All set!

Playing with text

Feeling python

$python

>>>print 'Hello world'

>>>2+3+4

Okay !

cont..

Data from book:

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

see what is it?

```
>>>text1
```

Search the word:

```
>>> text1.concordance("monstrous")
```

Find similar words

```
>>> text1.similar("monstrous")
```

cont..

Find common context

```
>>>text2.common_contexts(["monstrous", "very"])
```

Find position of the word in entire text

```
>>>text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
```

Counting vocab

```
>>>len(text4)  #gives tokens(words and punctuations)
```

cont..

Vocabulary(unique words) used by author

```
>>>len(set(text1))
```

See sorted vocabulary(upper case first!)

```
>>>sorted(set(text1))
```

Measuring richness of text(average repetition)

```
>>>len(text1)/len(set(text1))
```

(before division import for floating point division

from __future__ import division)

cont..

Counting occurances

```
>>>text5.count("lol")
```

% of the text taken

```
>>> 100 * text5.count('a') / len(text5)
```

let's write function for it in python

```
>>>def  prcntg_oftext(word,text):
            return100*.count(wrd)/len(txt)
```

cont..

Writing our own text

```
>>>textown =['a','text','is','a','list','of','words']
```

adding texts(lists, can contain anything)

```
>>>sent = ['pqr']
>>>sent2 = ['xyz']
>>>sent+sent2
```

append to text

```
>>>sent.append('abc')
```

cont..

Indexing text

```
>>> text4[173]
```

or

```
>>>text4.index('awaken')
```

slicing

text1[3:4], text[:5], text[10:] try it! (notice ending index one less than specified)

cont..

Strings in python

```
>>>name = 'astring'
>>>name[0]
```

what's more

```
>>>name*2
>>>'SPACE '.join(['let us','join'])
```

cont..

Statistics to text

Frequency Distribution

```
>>> fdist1 = FreqDist(text1)
>>> fdist1
>>> vocabulary1 = fdist1.keys()
>>>vocabulary1[:25]
```

let's plot it...

```
>>>fdist1.plot(50, cumulative=True)
```

cont..

*what the text1 is about?

*are all the words meaningful?

Let's find long words i.e.

Words from text1 with length more than 10

in python

```
>>>longw=[word for word in text1 if len(word)>10]
```

let's see it...

```
>>>sorted(longw)
```

cont..

*most long words have small freq and are not major topic of text many times

*words with long length and certain freq may help to identify text topic

```
>>>sorted([w for w in set(text5) if len(w) > 7 and fdist5[w] > 7])
```

cont..

Bigrams and collocations:

*Bigrams

```
>>>sent=['near','river', 'bank']
>>>bigrams(sent)
```

basic to many NLP apps; transliteration, translation

*collocations: frequent bigrams of rare words

```
>>>text4.collocations()
```

basic to some WSD approaches

cont..

FreDist functions:

`fdist = FreqDist(samples)`	Create a frequency distribution containing the given samples
`fdist.inc(sample)`	Increment the count for this sample
`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.keys()`	The samples sorted in order of decreasing frequency
`for sample in fdist:`	Iterate over the samples, in order of decreasing frequency
`fdist.max()`	Sample with the greatest count
`fdist.tabulate()`	Tabulate the frequency distribution
`fdist.plot()`	Graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	Cumulative plot of the frequency distribution
`fdist1 < fdist2`	Test if samples in `fdist1` occur less frequently than in `fdist2`

cont..

Python Frequntly needed 'word' functions

Function	Meaning
`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
`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)

cont

Bigger probs:

WSD, Anaphora resolution, Machine translation

try and laugh:

>>>babelize_shell()

>>>german

>>>run

or try it here:
http://translationparty.com/#6917644

cont..

Playing with text corpora

Available corporas: Gutenberg, Web and chat text, Brown, Reuters, inaugural address, etc.

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()
>>>fileid[:4] for fileid in inaugural.fileids()]
```

cont..

```
>>> cfd = nltk.ConditionalFreqDist(
        (target, file[:4])
        for fileid in inaugural.fileids()
        for w in inaugural.words(fileid)
        for target in ['america', 'citizen']
        if w.lower().startswith(target))
>>> cfd.plot()
```

Conditional Freq distribution of words 'america' and 'citizen' with different years

cont..

Generating text (applying bigrams)

```
>>> def generate_model(cfdist, word, num=15):
    for i in range(num):
        print word,
        word = cfdist[word].max()
---
>>>text = nltk.corpus.genesis.words('english-kjv.txt')
>>>bigrams = nltk.bigrams(text)
>>>cfd = nltk.ConditionalFreqDist(bigrams)
 >>print cfd['living']
 >>generate_model(cfd, 'living')
```

cont..

WordList corpora

```
>>>def unusual_words(text):
    text_vocab = set(w.lower() for w in text if
  w.isalpha())
   english_vocab = set(w.lower() for w in
  nltk.corpus.words.words())
   unusual = text_vocab.difference(english_vocab)
   return sorted(unusual)
```

--

```
>>>unusual_words(nltk.corpus.gutenberg.words('aust
  en-sense.txt'))
```

Are the unusual or misspelled words.

cont..

Stop words:

```
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
-- Percentage of content
>>> def content_fraction(text):
        stopwords = nltk.corpus.stopwords.words('english')
        content = [w for w in text if w.lower() not in stopwords]
        return len(content) / len(text)
>>> content_fraction(nltk.corpus.reuters.words())
```

is the useful fraction of useful words!

cont..

WordNet: structured, semanticaly orinted dictionary

```
>>> from nltk.corpus import wordnet as wn
```

get synset/collection of synonyms

```
>>> wn.synsets('motorcar')
```

now get synonyms(lemmas)

```
>>> wn.synset('car.n.01').lemma_names
```

cont..

What does car.n.01 actually mean?

```
>>> wn.synset('car.n.01').definition
```

An example:

```
>>> wn.synset('car.n.01').examples
```

Also

```
>>>wn.lemma('supply.n.02.supply').antonyms()
```

Useful data in many applications like WSD, understanding text, question answering, etc.

cont..

Text from web:

```
>>> from urllib import urlopen
>>> url =
"http://www.gutenberg.org/files/2554/2554.txt"
>>> raw = urlopen(url).read()
>>> type(raw)
>>> len(raw)
```

is in characters...!

cont..

Tokenization:

```
>>> tokens = nltk.word_tokenize(raw)
>>> type(tokens)
>>> len(tokens)
>>> tokens[:15]
```

convert to Text

```
>>> text = nltk.Text(tokens)
>>> type(text)
```

Now you can do all above things with this!

cont..

What about HTML?

```
>>> url =
"http://news.bbc.co.uk/2/hi/health/2284783.stm"
>>> html = urlopen(url).read()
>>> html[:60]
```

It's HTML; Clean the tags...

```
>>> raw = nltk.clean_html(html)
>>> tokens = nltk.word_tokenize(raw)
>>> tokens[:60]
>>> text = nltk.Text(tokens)
```

cont..

Try it yourself

Processing RSS feeds using universal feed parser (http://feedparser.org/)

Opening text from local machine:

```
>>> f = open('document.txt')
>>> raw = f.read()
```

(make sure you are in current directory)

then tokenize and bring to Text and play with it

cont..

Capture user input:

```
>>> s = raw_input("Enter some text: ")
>>> print "You typed", len(nltk.word_tokenize(s)), "words."
```

Writing O/p to files

```
>>> output_file = open('output.txt', 'w')
>>> words = set(nltk.corpus.genesis.words('english-kjv.txt'))
>>> for word in sorted(words):
output_file.write(word + "\n")
```

cont..

Stemming : NLTK has inbuilt stemmers

```
>>>text =['apples','called','indians','applying']
>>> porter = nltk.PorterStemmer()
>>> lancaster = nltk.LancasterStemmer()
>>>[porter.stem(p) for p in text]
>>>[lancaster.stem(p) for p in text]
```

Lemmatization: WordNet lemmetizer, removes affixes only if the new word exist in dictionary

```
>>> wnl = nltk.WordNetLemmatizer()
>>> [wnl.lemmatize(t) for t in text]
```

cont..

Using a POS tagger:

```
>>> text = nltk.word_tokenize("And now for
something completely different")
```

Query the POS tag documentation as:

```
>>>nltk.help.upenn_tagset('NNP')
```

Parsing with NLTK

an example

cont..

Classifying with NLTK:

Problem: Gender identification from name

Feature last letter

```
>>> def gender_features(word):
        return {'last_letter': word[-1]}
```

Get data

```
>>> from nltk.corpus import names
>>> import random
```

cont..

```
>>> names = ([(name, 'male') for name in
names.words('male.txt')] +
        [(name, 'female') for name in
names.words('female.txt')])
>>> random.shuffle(names)
```

Get Feature set

```
>>> featuresets = [(gender_features(n), g) for (n,g) in
names]
>>> train_set, test_set = featuresets[500:], featuresets[:500]
```

Train

```
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Test


```
>>> classifier.classify(gender_features('Neo'))
```

Conclusion

NLTK is rich resource for starting with Natural Language Processing and it's being open source and based on python adds feathers to it's cap!

References

- http://www.nltk.org/
- http://www.python.org/
- http://wikipedia.org/

(all accessed latest on 19th March 2010)

- *Natural Language Processing with Python,* Steven Bird, Ewan Klein, and Edward Loper, O'REILLY

Thank you

#More on NLP @ http://matra2.blogspot.com/

#Slides were prepared for 'supporting' the talk and are distributed as it is under CC 3.0