Natural Language Processing and Machine Learning

Using Python

Shankar Ambady

Microsoft New England Research and Development Center, December 14, 2010
Example Files

Hosted on Github

https://github.com/shanbady/NLTK-Boston-Python-Meetup
i. What is “Natural Language Processing”?
   i. Where is this stuff used?
   ii. The Machine learning paradox
   ii. A look at a few key terms
iii. Quick start – creating NLP apps in Python
What is Natural Language Processing?

• Computer aided text analysis of human language.

• The goal is to enable machines to understand human language and extract meaning from text.

• It is a field of study which falls under the category of machine learning and more specifically computational linguistics.

• The “Natural Language Toolkit” is a python module that provides a variety of functionality that will aide us in processing text.
Natural language processing is heavily used throughout all web technologies:

- Search engines
- Consumer behavior analysis
- Site recommendations
- Banking fraud detection
- Spam filtering
- Automated customer support systems
- Knowledge bases and expert systems
Paradoxes in Machine Learning

Sentiment

• Sarcasm
• Slang

Intent

Ambiguity

• Emphasis
• Time and date
  • Since when did “google” become a verb?
Little sister: What’s your name?

Me: Uhh....Shankar..?

Sister: Can you spell it?

Sister: WRONG! It’s spelled “I-T”
Ambiguity

“I shot the man with ice cream.”

- A man with ice cream was shot
- A man had ice cream shot at him
Language translation is a complicated matter!

Go to: http://babel.mrfeinberg.com/

The problem with communication is the illusion that it has occurred
The problem with communication is the illusion that it has occurred.

Das Problem mit Kommunikation ist die Illusion, dass es aufgetreten ist.

The problem with communication is the illusion that it arose.

Das Problem mit Kommunikation ist die Illusion, dass es entstand.

The problem with communication is the illusion that it developed.

Das Problem mit Kommunikation ist die Illusion, die sie entwickelte.

The problem with communication is the illusion, which developed it.
The problem with communication is the illusion that it has occurred

The problem with communication is the illusion, which developed it

EPIC FAIL
The “Human Test”

• Turing test
  – A test proposed to demonstrate that truly intelligent machines capable of understanding and comprehending human language should be indistinguishable from humans performing the same task.
Key Terms
Classification:

- Automatically organizing text by subject and tagging it with a proper category.
- Two types:
  - Supervised
  - Unsupervised

Tagging:

- Attaching part of speech, tense, related terms, and other properties to tokens of text.
Tokenizing:

- Process of breaking text into defined segments (usually using regexes or simple delimiters).

Stemming:

- Process of breaking words to their stem removing plural forms, tense etc...
  
  **Jump**: jump-*ing*, jump-*ed*, jump-*s*
Collocations

• Short sequences of words that commonly appear together.
• Commonly used to provide search suggestions as users type.

N-Grams

• Tokens consisting of one or more words:
  • Unigrams
  • Bigrams
  • Trigrams
Setting up NLTK

- Source downloads available for mac and linux as well as installable packages for windows.
- Currently only available for Python 2.5 – 2.6
- [http://www.nltk.org/download](http://www.nltk.org/download)
- `easy_install nltk`
- Prerequisites
  - NumPy
  - SciPy
First steps

• NLTK comes with packages of corpora that are required for many modules.

• Open a python interpreter:

```python
import nltk
nltk.download()
```

If you do not want to use the downloader with a gui (requires TKInter module)

Run: python -m nltk.downloader <name of package or “all”>
You may individually select packages or download them in bulk.
Let’s dive into some code!
from nltk import pos_tag, word_tokenize

sentence1 = 'this is a demo that will show you how to detects parts of speech with little effort using NLTK!

tokenized_sent = word_tokenize(sentence1)
print pos_tag(tokenized_sent)
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential &quot;there&quot;</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition or subordination conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective- comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective- superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun- singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun- plural</td>
</tr>
<tr>
<td>NP</td>
<td>Proper noun- singular</td>
</tr>
<tr>
<td>NPS</td>
<td>Proper noun- plural</td>
</tr>
</tbody>
</table>
nltk.clean_html(rawhtml)

from nltk.corpus import brown
from nltk import Text

brown_words = brown.words(categories='humor')

brownText = Text(brown_words)

collocations()

count("car")

cordance("oil")

dispersion_plot(['car', 'document', 'funny', 'oil'])
similar('humor')
import nltk
from nltk.corpus import wordnet as wn

synsets = wn.synsets('phone')

print [str(syns.definition) for syns in synsets]

1) 'electronic equipment that converts sound into electrical signals that can be transmitted over distances and then converts received signals back into sounds'
2) '(phonetics) an individual sound unit of speech without concern as to whether or not it is a phoneme of some language'
3) 'electro-acoustic transducer for converting electric signals into sounds; it is held over or inserted into the ear'
4) 'get or try to get into communication (with someone) by telephone'
Meronyms and Holonyms
Meronyms and Holonyms are better described in relation to computer science terms as:

- Meronym terms: “has a” relationship
- Holonym terms: “part of” relationship
- Hyponym terms: “Is a” relationship

- Meronyms and holonyms are opposites
- Hyponyms and hypernyms are opposites
Burger is a holonym of:
Cheese, beef, tomato, and bread are meronyms of burger.
from nltk.corpus import wordnet as wn

synsets = wn.synsets('phone')

print [str(syns.definition) for syns in synsets]

"syns.definition" can be modified to output hypernyms, meronyms, holonyms etc:
<table>
<thead>
<tr>
<th><code>&lt;synset&gt;.hyponyms</code></th>
<th>Hyponyms of synset</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;synset&gt;.hyponyms</code></td>
<td>Hyponyms of synset</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.root_hypernym</code></td>
<td>A hypernym of synset that is highest in the hierarchy</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.common_hypernym</code></td>
<td>Common hypernyms of two synsets</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.lowest_common_hypernym</code></td>
<td>A common hypernym of two synsets that appears at the lowest level in the hierarchy</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.member_holonym</code></td>
<td>Groups consisting of the specified members</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.member_meronym</code></td>
<td>Members of the specified group</td>
</tr>
</tbody>
</table>

Source: http://www.sjsu.edu/faculty/hahn.koo/teaching/ling115/lecture_notes/ling115_wordnet.pdf
<table>
<thead>
<tr>
<th>Synset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;synset&gt;.substance_holonyms</code></td>
<td>Things made of the specified substance</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.substance_meronyms</code></td>
<td>Substance of the specified thing</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.part_holonyms</code></td>
<td>Things consisting of the specified parts</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.part_meronyms</code></td>
<td>Parts of the specified whole</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.attributes</code></td>
<td>List of synsets that describes the attributes of synset</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.entailments</code></td>
<td>What is entailed by the specified synset</td>
</tr>
<tr>
<td><code>&lt;synset&gt;.similar_tos</code></td>
<td>List of similar adjectival senses</td>
</tr>
</tbody>
</table>

Source:
http://www.sjsu.edu/faculty/hahn.koo/teaching/ling115/lecture_notes/ling115_wordnet.pdf
from nltk.corpus import wordnet as wn

synsets = wn.synsets('car')

print [str(syns.part_meronyms()) for syns in synsets]
from nltk.corpus import wordnet as wn
synsets = wn.synsets('wing')
print [str(syns.part_holonyms()) for syns in synsets]

[Synset('airplane.n.01')]
[Synset('division.n.09')]
[Synset('bird.n.02')]
[Synset('car.n.01')]
[Synset('building.n.01')]


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

synsets = wn.synsets('trees')
print [str(syns.part_meronyms()) for syns in synsets]
```

- synset('burl.n.02')
- synset('crown.n.07')
- synset('stump.n.01')
- synset('trunk.n.01')
- synset('limb.n.02')
```python
from nltk.corpus import wordnet as wn

for hypernym in wn.synsets('robot')[0].hypernym_paths()[0]:
    print(hypernym.lemma_names)
```

```python
['entity']
['physical_entity']
['object', 'physical_object']
    ['whole', 'unit']
    ['artifact', 'artefact']
    ['instrumentality', 'instrumentation']
    ['device']
    ['mechanism']
    ['automaton', 'robot', 'golem']
```
Fun things to Try
Feeling lonely?

Eliza is there to talk to you all day! What human could ever do that for you??

```python
from nltk.chat import eliza
eliza.eliza_chat()
```

......starts the chatbot

**Therapist**

---------

Talk to the program by typing in plain English, using normal upper- and lower-case letters and punctuation. Enter "quit" when done.

=======================================================================

Hello. How are you feeling today?
from nltk.book import *
babelize_shell()

Babel> the internet is a series of tubes
Babel> german
Babel> run
0> the internet is a series of tubes
1> das Internet ist eine Reihe SchlSuche
2> the Internet is a number of hoses
3> das Internet ist einige SchlSuche
4> the Internet is some hoses
Babel>
A new study in the journal Animal Behavior shows that dogs rely a great deal on face recognition to tell their own person from other people. Researchers describe how dogs in the study had difficulty recognizing their owners when two human subjects had their faces covered.

Let’s have NLTK analyze and generate some gibberish!

```python
import nltk
words = 'text'
tokens = nltk.word_tokenize(words)
text = nltk.Text(tokens)
print text.generate()
```
A new study in the journal Animal Behavior shows that dogs rely a great deal on face recognition to tell their own person from other people. Researchers describe how dogs in the study had difficulty recognizing their owners when two human subjects had their faces covered. Researchers describe how dogs in the study had difficulty recognizing their owners when two human subjects had their faces covered.
#!/usr/bin/env python
from nltk.corpus import wordnet as wn

similar = []

Aword = 'language'
Bword = 'barrier'
groupA = \[wn.synset(str(synset.name)) \textbf{for} \text{synset} \textbf{in} \text{synsetsA}\]
groupB = \[wn.synset(str(synset.name)) \textbf{for} \text{synset} \textbf{in} \text{synsetsB}\]
path_similarity()

“Path Distance Similarity: Return a score denoting how similar two word senses are, based on the shortest path that connects the senses in the is-a (hyponym/hypernym) taxonomy.”

wup_similarity()

“Wu-Palmer Similarity: Return a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node).”

```python
for sseta in groupA:
    for ssetb in groupB:
        path_similarity = sseta.path_similarity(ssetb)
        wup_similarity = sseta.wup_similarity(ssetb)

        if path_similarity is not None:
            similars.append({
                'path': path_similarity,
                'wup': wup_similarity,
                'wordA': sseta,
                'wordB': ssetb,
                'wordA_definition': sseta.definition,
                'wordB_definition': ssetb.definition
            })
```

"how similar" (continued)
`similars = sorted(similars, key=lambda item: item['path'], reverse=True)`

```python
for item in similars:
    print(item['wordA'], "- ", item['wordA_definition'])
    print(item['wordB'], "-", item['wordB_definition'])
    print('Path similarity - ', item['path'], '
```

"how similar" (continued)
the cognitive processes involved in producing and understanding linguistic communication

Similarity: 0.111~

any condition that makes it difficult to make progress or to achieve an objective
It trickles down from there

Synset('linguistic_process.n.02')
the cognitive processes involved in producing and understanding linguistic communication

  Synset('barrier.n.02')
  any condition that makes it difficult to make progress or to achieve an objective
  Path similarity - 0.111111111111

Synset('language.n.05')
the mental faculty or power of vocal communication

  Synset('barrier.n.02')
  any condition that makes it difficult to make progress or to achieve an objective
  Path similarity - 0.111111111111

Synset('language.n.01')
a systematic means of communicating by the use of sounds or conventional symbols

  Synset('barrier.n.02')
  any condition that makes it difficult to make progress or to achieve an objective
  Path similarity - 0.1

Synset('language.n.01')
a systematic means of communicating by the use of sounds or conventional symbols

  Synset('barrier.n.03')
  anything serving to maintain separation by obstructing vision or access
  Path similarity - 0.1
Poetic Programming

• We will create a program to extract “Haikus” from any given English text.

• A haiku is a poem in which each stanza consists of three lines.

• The first line has 5 syllables, the second has 7 and the last line has 5.
Inspired by a GitHub project “Haiku Finder”:  

https://github.com/jdf/haikufinder

We will be re-implementing this program and adding a few other little features.

You will need

- The nltk_contrib package from Google Code:  
  http://code.google.com/p/nltk/downloads/list
- The following corpora:  
  • Wordnet  
  • Cmudict
- A few paragraphs of text that we will use to create haikus from
They want to deliver vast amounts of information over the Internet. And again, the Internet is not something that you just dump something on.

It's not a big truck. It's a series of tubes.

And if you don't understand, those tubes can be filled and if they are filled, when you put your message in, it gets in line and it's going to be delayed by anyone that puts into that tube enormous amounts of material, enormous amounts of material...
I want to share with you an interesting program for two reasons, one, it's interesting, and two, my wife thought of it or has actually been involved with it; she didn't think of it. But she thought of it for this speech.

This is my maiden voyage. My first speech since I was the president of the United States and I couldn't think of a better place to give it than Calgary, Canada.
poem = ''
wordmap = []

words = word_tokenize(textchunk)

for iter, word in enumerate(words):
    # if it is a word, add a space to it
    if word.isalpha():
        word += " \\

syls = syllables_en.count(word)
wordmap.append(((word, syls)))

_tokenize the words

NLTK function to count syllables
Define a function to provide a fallback word in case we end up with lines that do not have the syllable count we need.

```python
def findSyllableWord(word, syllableSize):
    synsets = wn.synsets(word)
    for syns in synsets:
        name = syns.name
        lemmas = syns.lemma_names
        for wordstring in lemmas:
            if(syllables_en.count(wordstring) == syllableSize and wordstring != word):
                return {'word':word, 'syllable':syllableSize}
    return {'word':word, 'syllable':syllables_en.count(word)}
```

Given a word, this function tries to find similar words from WordNet that match the required syllable size.
We loop through the each word keeping tally of syllables and breaking each line when it reaches the appropriate threshold.

```python
lineNo = 1
charNo = 0
tally = 0
for syllabicword in wordmap:
    s = syllabicword[1]
    wordtoAdd = syllabicword[0]
    if lineNo == 1:
        if tally < 5:
            if tally + int(s) > 5 and wordtoAdd.isalpha():
                num = 5 - tally
                similarterm = findSyllableWord(wordtoAdd, num)
                wordtoAdd = similarterm['word']
                s = similarterm['syllable']
                tally += int(s)
                poem += wordtoAdd
            else:
                poem += " ---"+str(tally)+"\n"
        if wordtoAdd.isalpha():
            poem += wordtoAdd
            tally = s
    else:
        poem += " ---"+str(tally)+"\n"
    lineNo = 2
    if lineNo == 2:
        .... Abridged
```
print poem

Its not perfect but its still pretty funny!

I want to share with you an interesting program for two reasons, one it's interesting and two, my wife thought of it or has actually been involved with it; she didn't think of it. But she thought of it for this speech. This is my maiden voyage. My first speech since I was the president of.... Abridged
Let’s build something even cooler
Let's write a spam filter!

A program that analyzes legitimate emails “Ham” as well as “spam” and learns the features that are associated with each.

Once trained, we should be able to run this program on incoming mail and have it reliably label each one with the appropriate category.
What you will need

1. NLTK (of course) as well as the “stopwords” corpus

2. A good dataset of emails; Both spam and ham

2. Patience and a cup of coffee
   (these programs tend to take a while to complete)
Finding Great Data: The Enron Emails

- A dataset of 200,000+ emails made publicly available in 2003 after the Enron scandal.
- Contains both spam and actual corporate ham mail.
- For this reason it is one of the most popular datasets used for testing and developing anti-spam software.

The dataset we will use is located at the following url:
http://labs-repos.iit.demokritos.gr/skel/i-config/downloads/enron-spam/preprocessed/

It contains a list of archived files that contain plaintext emails in two folders, Spam and Ham.
1. Extract one of the archives from the site into your working directory.

2. Create a python script, lets call it “spambot.py”.

3. Your working directory should contain the “spambot” script and the folders “spam” and “ham”.

```python
from nltk import word_tokenize,\       
WordNetLemmatizer,NaiveBayesClassifier\  
,classify,MaxentClassifier

from nltk.corpus import stopwords
import random
import os, glob, re
```
wordlemmatizer = WordNetLemmatizer()
commonwords = stopwords.words('english')
hamtexts  = []
spamtexts  = []

for infile in glob.glob( os.path.join( 'ham/', '*.txt' ) ):
    text_file = open(infile, "r")
    hamtexts.append(text_file.read())
    text_file.close()

for infile in glob.glob( os.path.join( 'spam/', '*.txt' ) ):
    text_file = open(infile, "r")
    spamtexts.append(text_file.read())
    text_file.close()
label each item with the appropriate label and store them as a list of tuples

mixedemails =  [[(email,'spam') for email in spamtexts]  
mixedemails +=  [(email,'ham') for email in hamtexts]]

From this list of random but labeled emails, we will defined a “feature extractor” which outputs a feature set that our program can use to statistically compare spam and ham.

random.shuffle(mixedemails)  
lets give them a nice shuffle
def email_features(sent):
    features = {}
    wordtokens = [wordlemmatizer.lemmatize(word.lower()) for word in word_tokenize(sent)]
    for word in wordtokens:
        if word not in commonwords:
            features[word] = True
    return features

featuresets = [(email_features(n), g) for (n, g) in mixedemails]
- The features you select must be binary features such as the existence of words or part of speech tags (True or False).

- To use features that are non-binary such as number values, you must convert it to a binary feature. This process is called “binning”.

- If the feature is the number 12 the feature is: (“11<x<13”, True)
Let's grab a sampling of our featureset. Changing this number will affect the accuracy of the classifier. It will be a different number for every classifier, find the most effective threshold for your dataset through experimentation.

\[
\text{size} = \text{int}(\text{len(featuresets)} \times 0.7)
\]

Using this threshold grab the first \( n \) elements of our featureset and the last \( n \) elements to populate our “training” and “test” featuresets.

\[
\text{train\_set, test\_set} = \text{featuresets}[\text{size:}], \text{featuresets}[:\text{size}]
\]

Here we start training the classifier using NLTK’s built in Naïve Bayes classifier.

\[
\text{classifier} = \text{NaiveBayesClassifier.train(train\_set)}
\]
print classifier.labels()  # This will output the labels that our classifier will use to tag new data

['ham', 'spam']

The purpose of creating a “training set” and a “test set” is to test the accuracy of our classifier on a separate sample from the same data source.

print classify.accuracy(classifier, test_set)

0.983589566419
classifier.show_most_informative_features(20)

<table>
<thead>
<tr>
<th>Ham</th>
<th>Spam</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ect</td>
<td>ham</td>
<td>46:1</td>
</tr>
<tr>
<td>cc</td>
<td>ham</td>
<td>40.2:1</td>
</tr>
<tr>
<td>kitchen</td>
<td>ham</td>
<td>39.1:1</td>
</tr>
<tr>
<td>wednesday</td>
<td>ham</td>
<td>29.8:1</td>
</tr>
<tr>
<td>shirley</td>
<td>ham</td>
<td>29.1:1</td>
</tr>
<tr>
<td>meeting</td>
<td>ham</td>
<td>26.8:1</td>
</tr>
<tr>
<td>houston</td>
<td>ham</td>
<td>24.3:1</td>
</tr>
<tr>
<td>thursday</td>
<td>ham</td>
<td>23.9:1</td>
</tr>
<tr>
<td>2001</td>
<td>ham</td>
<td>19.1:1</td>
</tr>
<tr>
<td>mary</td>
<td>ham</td>
<td>19.1:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Spam</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>spam</td>
<td>43.5:1</td>
</tr>
<tr>
<td>removed</td>
<td>spam</td>
<td>42.9:1</td>
</tr>
<tr>
<td>thousand</td>
<td>spam</td>
<td>38.2:1</td>
</tr>
<tr>
<td>prescription</td>
<td>spam</td>
<td>34.2:1</td>
</tr>
<tr>
<td>doctor</td>
<td>spam</td>
<td>28.9:1</td>
</tr>
<tr>
<td>super</td>
<td>spam</td>
<td>26.9:1</td>
</tr>
<tr>
<td>quality</td>
<td>spam</td>
<td>26.9:1</td>
</tr>
<tr>
<td>drug</td>
<td>spam</td>
<td>26.9:1</td>
</tr>
<tr>
<td>remove</td>
<td>spam</td>
<td>26.1:1</td>
</tr>
<tr>
<td>cheap</td>
<td>spam</td>
<td>24.9:1</td>
</tr>
</tbody>
</table>
While True:
    featset = email_features(raw_input("Enter text to classify: "))
    print classifier.classify(featset)

We can now directly input new email and have it classified as either Spam or Ham
- The quality of your input data will affect the accuracy of your classifier.

- The threshold value that determines the sample size of the feature set will need to be refined until it reaches its maximum accuracy. This will need to be adjusted if training data is added, changed or removed.
- The accuracy of this dataset can be misleading; In fact our spambot has an accuracy of 98% - but this only applies to Enron emails. This is known as “over-fitting”.

- Try classifying your own emails using this trained classifier and you will notice a sharp decline in accuracy.
Chunking
Complete sentences are composed of two or more “phrases”.

– Noun phrase:
  • *Jack and Jill* went up the hill

– Prepositional phrase:
  • Contains a noun, preposition and in most cases an adjective
  • *The NLTK book is on the table* but perhaps it is best kept in a bookshelf

– Gerund Phrase:
  • Phrases that contain “−ing” verbs
  • Jack fell down and broke his crown and *Jill came tumbling after*
Take the following sentence .....  

Jack and Jill *went* up the hill

Noun phrase

Noun Phrase
Chunkers will get us this far:

\[
[ \text{Jack and Jill} ] \text{ went up } [ \text{the hill} ]
\]

Chunk tokens are non-recursive – meaning, there is no overlap when chunking.

The recursive form for the same sentence is:

\[
( \text{Jack and Jill went up (the hill)} )
\]
Verb phrase chunking

Jack and Jill *went up the hill* to *fetch a pail of water*

Verb Phrase

Verb Phrase
from nltk.chunk import *
from nltk.chunk.util import *
from nltk.chunk.regexp import *
from nltk import word_tokenize, pos_tag

text = "Jack and Jill went up the hill to fetch a pail of water"
tokens = pos_tag(word_tokenize(text))

chunk = ChunkRule("<.*>+", "Chunk all the text")
chink = ChinkRule("<VBD|IN|\./>" , "Verbs/Props")
split = SplitRule("<DT><NN>", "<DT><NN>", "determiner+noun")

chunker = RegexpChunkParser([chunk, chink, split], chunk_node='NP')
chunked = chunker.parse(tokens)
chunked.draw()
Chunkers and Parsers ignore the words and instead use part of speech tags to create chunks.
from nltk import ne_chunk, pos_tag
from nltk.tokenize.punkt import PunktSentenceTokenizer
from nltk.tokenize.treebank import TreebankWordTokenizer

TreeBankTokenizer = TreebankWordTokenizer()
PunktTokenizer = PunktSentenceTokenizer()

text = ""
text on next slide"

sentences = PunktTokenizer.tokenize(text)
tokens = [TreeBankTokenizer.tokenize(sentence) for sentence in sentences]
tagged = [pos_tag(token) for token in tokens]
chunked = [ne_chunk(taggedToken) for taggedToken in tagged]
The Boston Celtics are a National Basketball Association (NBA) team based in Boston, MA. They play in the Atlantic Division of the Eastern Conference. Founded in 1946, the team is currently owned by Boston Basketball Partners LLC. The Celtics play their home games at the TD Garden, which they share with the Boston Blazers (NLL), and the Boston Bruins of the NHL. The Celtics have dominated the league during the late 50's and through the mid 80's, with the help of many Hall of Famers which include Bill Russell, Bob Cousy, John Havlicek, Larry Bird and legendary Celtics coach Red Auerbach, combined for a 795 - 397 record that helped the Celtics win 16 Championships.
The Boston Celtics are a National Basketball Association team based in Boston, MA. They play in the Atlantic Division of the Eastern Conference. Founded in 1946, the team is currently owned by Boston Basketball Partners which include Bill Russell, Bob Cousy, John Havlicek, Larry Bird and legendary Celtics coach Red Auerbach.
chunked[0].draw()
Thank you for coming!

Special thanks

Ned Batchelder and Microsoft

My latest creation.
Check out weatherzombie.com on your iphone or android!
Further Resources:

• This presentation with all the slides can be downloaded from my website
  – http://www.shankarambady.com

• “Natural Language Processing with Python” by Steven Bird, Ewan Klein, and Edward Loper


• Great NLTK blog:
  – http://streamhacker.com/