How to perform some common NLP tasks using NLTK

Michael Gabilondo

CAP 5636, Fall 2011

Outline

What is NLTK?

- Natural Language Toolkit (NLTK) is a large collection of Python modules to facilitate natural language processing
- It also includes a large amount of optional data, such as annotated text corpora and WordNet
- NLTK: http://www.nltk.org/
- Free NLTK Book: http://www.nltk.org/book

Outline

Detect sentence boundaries

- sent_tokenize is NLTK's current recommended method to tokenize sentences

```
>>> from nltk import tokenize
>>> text = "Abbreviations like Mr. and Mrs. contain periods but don
't end sentences. The tokenizer should not split those."
>>> sentences = tokenize.sent_tokenize(text)
>>>
>>> print sentences
["Abbreviations like Mr. and Mrs. contain periods but don't end sen
tences.", 'The tokenizer should not split those.']
>>>
>>> for sent in sentences:
...     print sent
...
Abbreviations like Mr. and Mrs. contain periods but don't end sente
nces.
The tokenizer should not split those.
```

Tokenizing Words

- word_tokenize is NLTK's current recommended method to tokenize words
- It uses TreebankWordTokenizer, "A regular-expression based word tokenizer that tokenizes sentences using the conventions used by the Penn Treebank."

```
>>> from nltk import word_tokenize
>>> sent = "John's big idea isn't all that bad."
>>> tokens = word_tokenize(sent)
>>> print tokens
['John', "'s", 'big', 'idea', 'is', "n't", 'all', 'that', 'bad', '.']
```

- word_tokenize should be fed one sentence at a time

Outline

Tagging Example

```
>>> import nltk
>>> tokens = nltk.word_tokenize("They refuse to permit us t
o obtain the refuse permit")
>>> print tokens
['They', 'refuse', 'to', 'permit', 'us', 'to', 'obtain', 't
he', 'refuse', 'permit']
>>>
>>> tagged_tokens = nltk.pos_tag(tokens)
>>> print tagged_tokens
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit
', 'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('
the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

- pos_tag is NLTK's currently recommended POS tagger, trained on Penn Treebank

Outline

1. NLTK
2. Example NLP Pipeline
   - Segmentation of Sentences and Words
   - POS Tagging
   - Parsing
3. WordNet
   - WordNet
4. Classifiers
   - Spam Detection Example

Stanford Parser

- NLTK does not have a good parser; I recommend using Stanford Parser
- Stanford parser can perform sentence splitting, word tokenization and POS tagging; so you can give it plain text English paragraphs as input
- You can also give it POS tagged text; the parser will try to use your tags if they make sense
  - You might want to do this if the parser makes tagging mistakes in your text domain

Stanford Parser Example

- They refuse us to obtain the refuse permit
- They/PRP refuse/VBP us/PRP to/TO obtain/VB the/DT refuse/NN permit/NN

```
(ROOT
  (S
    (NP (PRP They))
    (VP (VBP refuse)
      (S
        (NP (PRP us))
        (VP (TO to)
          (VP (VB obtain)
            (NP (DT the) (NN refuse) (NN permit)))))))))
```

Stanford Parser: Dependencies Example

- They refuse us to obtain the refuse permit

```
nsubj(refuse-2, They-1)
nsubj(obtain-5, us-3)
aux(obtain-5, to-4)
xcomp(refuse-2, obtain-5)
det(permit-8, the-6)
nn(permit-8, refuse-7)
dobj(obtain-5, permit-8)
```

Reading the output of Stanford Parser using NLTK

- NLTK has a good Tree class that inherits from list (so Trees are also regular Python lists)
- You can read in Parse trees produced by Stanford parser with this class

```
>>> import nltk
>>> tparse = nltk.tree.Tree.parse
>>> tree = tparse("(NP (DT the) (JJ fat) (NN man))")
>>> for subtree in tree:
...     print subtree, '---', subtree.node
...
(DT the) --- DT
(JJ fat) --- JJ
(NN man) --- NN
>>> print tree.node
NP
>>> print len(tree)
3   _
```

Outline

1. NLTK
2. Example NLP Pipeline
   - Segmentation of Sentences and Words
   - POS Tagging
   - Parsing
3. WordNet
   - WordNet
4. Classifiers
   - Spam Detection Example

WordNet Noun Ontology

- WN's noun ontology is organized into **synsets**, which represent concepts/senses of words
- e.g., the **word** bat appears in 5 different **synsets**, each of which corresponds to a different sense of the word bat
  - **{bat, chiropteran}** – (mammal)
  - **{bat, at-bat}** – (a turn trying to get a hit)
  - **{squash racket, squash racquet, bat}**
  - **{cricket bat, bat}**
  - **{bat}** – (a club used for hitting a ball in various games)
- Synsets are mainly related to each other by hypernymy and hyponymy (is-a relations)
  - {bat, chiropteran} is a **hyponym** of {placental, placental mammal, eutherian, eutherian mammal}
  - {placental, placental mammal, eutherian, eutherian mammal} is a **hypernym** of {bat, chiropteran}

Accessing the synsets of a word

```
>>> from nltk.corpus import wordnet as wn
>>> for syn in wn.synsets('bat', 'n'):
...     print syn.name, '---', [lemma.name for lemma in syn.lemmas]
...
bat.n.01 --- ['bat', 'chiropteran']
bat.n.02 --- ['bat', 'at-bat']
squash_racket.n.01 --- ['squash_racket', 'squash_racquet', 'bat']
cricket_bat.n.01 --- ['cricket_bat', 'bat']
bat.n.05 --- ['bat']
```

- wn.synsets('bat', 'n') returns the list of synsets to which the noun 'bat' belongs to
- Given a synset syn, syn.lemmas returns the list of lemma objects (words) in that synset
  - lemma.name gets you the actual word string
  - [lemma.name for lemma in syn.lemmas] constructs a list containing the words in the synset

Hypernyms and Hyponyms of a Synset

```
>>> from nltk.corpus import wordnet as wn
>>> bat1 = wn.synsets('bat', 'n')[0]
>>> print bat1
Synset('bat.n.01')
>>> print bat1.hypernyms()
[Synset('placental.n.01')]
>>> print bat1.hyponyms()
[Synset('fruit_bat.n.01'), Synset('carnivorous_bat.n.01')]
>>> for path in bat1.hypernym_paths():
...     print path
...
[Synset('entity.n.01'), Synset('physical_entity.n.01'), Synset('object.n.0
1'), Synset('whole.n.02'), Synset('living_thing.n.01'), Synset('organism.n
.01'), Synset('animal.n.01'), Synset('chordate.n.01'), Synset('vertebrate.
n.01'), Synset('mammal.n.01'), Synset('placental.n.01'), Synset('bat.n.01'
)]
```

Semantic Similarity

- NLTK provides 6 different similarity measures for synsets; they return a score denoting how similar two synsets (word senses) are
- "This Fun, Safe and Easy walk through passages free of **bats** and **mosquitoes** takes about 1 hour"

```
>>> from nltk.corpus import wordnet as wn
>>> mosquito1 = wn.synset('mosquito.n.01')
>>> for bat_sense in wn.synsets('bat', 'n'):
...     print bat_sense.name, bat_sense.path_similarity(mosquito1)
...
bat.n.01 0.0909090909091
bat.n.02 0.0526315789474
squash_racket.n.01 0.0666666666667
cricket_bat.n.01 0.0666666666667
bat.n.05 0.0666666666667
```

Stemming words

```
>>> from nltk.corpus import wordnet as wn
>>> wn.morphy('dogs', 'n')
'dog'
>>> wn.morphy('ran')
'run'
>>> wn.morphy('ate') # Ate is also a noun in WN
'ate'
>>> wn.morphy('ate', 'v')
'eat'
>>> wn.morphy('deserts', 'n') # as in "he got his just deserts"
'deserts'
>>> wn._morphy('deserts', 'n')
['deserts', 'desert']
```

Outline

Classifiers

- nltk.classify includes decision tree, maximum entropy and naive bayes classifiers, as well as classifiers that make use of the external WEKA package (data mining software in Java)
- I will provide an example based off of a spam-detector example by Shankar Ambady (http://shankarambady.com/nltk.pdf)
- The training and testing sets come from a dataset of 200,000+ Enron emails which contain both "spam" and "ham" emails
    - http://labs-repos.iit.demokritos.gr/skel/i-config/downloads/enron-spam/preprocessed/

Preliminaries

```python
from nltk import word_tokenize, NaiveBayesClassifier
from nltk import classify
from nltk import WordNetLemmatizer
from nltk.corpus import stopwords
import random
import os, glob,re

wordlemmatizer = WordNetLemmatizer()
commonwords = stopwords.words('english')
```

- WordNetLemmatizer uses the "morphy" function, but unlike morphy, it returns the original word if morphy was unable to stem the word (e.g., if it's not in WN)
- stopwords corpus includes high-frequency grammatical words that will not be used as features
  - 'off', 'very', 'own', 'of', 'while', 'what', 'about', 'that', 'because', 'than', 'an', 'with', 's', 'has', 'my', 'doing', 'myself', 'no', 'i', 'been', 'all', 'too', 'where', 'whom', ...

Read in the “spam” and “ham” emails

```
def main():

    hamtexts = []
    spamtexts = []

    for filename in glob.glob( 'ham/*.txt' ):
        fin = open(filename)
        hamtexts.append(fin.read())
        fin.close()

    for filename in glob.glob( 'spam/*.txt' ):
        fin = open(filename)
        spamtexts.append(fin.read())
        fin.close()
```

...

```
if __name__ == "__main__":
    main()
```

- hamtexts and spamtexts are lists of strings (emails)

Label each email with its category and shuffle, extract features

```
46     mixedemails = [(email,'spam') for email in spamtexts]
47     mixedemails += [(email,'ham') for email in hamtexts]
48     random.shuffle(mixedemails)
49
50     featuresets = [(feature_extractor(email), label) \
51                    for (email,label) in mixedemails]
```

- Line 46: create mixedemails, a list of tuples [(email, 'spam'), ...]
- Line 47: Extend mixedemails with another list of tuples, [(email, 'ham'), ...]
- Line 48: Randomize the tuples in the list
- Line 50: create featuresets, a list of tuples [(features, label), ...], where the features are extracted from each email by feature_extractor()

Feature Extractor

```
19 def feature_extractor(sent):
20     features = {}
21     wordtokens = [wordlemmatizer.lemmatize(word.lower()) \
22                   for word in word_tokenize(sent)]
23
24     for word in wordtokens:
25         if word not in commonwords:
26             features[word] = True
27
28     return features
```

- features is a dict, a mapping type
  - features[**word**] maps to True if **word** is in **sent**
  - These types of features are called bag-of-words
- Line 21: wordtokens is a list stemmed tokens from sent
- Lines 24-26: Construct features for words in wordtokens which are not in the list of commonwords

Training and Testing

```
53    size = int(len(featuresets) * 0.7)
54    train_set, test_set = featuresets[size:], featuresets[:size]
55    print 'train_set size = %d,  test_set size = %d' \
56          % (len(train_set), len(test_set))
57
58    classifier = NaiveBayesClassifier.train(train_set)
59
60    print classify.accuracy(classifier,test_set)
61    classifier.show_most_informative_features(20)
62
63    while True:
64        featset = email_features(raw_input("Enter text to classify: "))
65        print classifier.classify(featset)
```

- Lines 53-54: Make a 30%/70% split between the training and testing sets
- Line 58: Train the Naive Bayes classifier
- Lines 60-61: Classify emails in the test set, output the accuracy; show top 20 most informative features
- Lines 63-65: Classify new emails by reading from standard input

Spam-detector output

```
train_set size = 1552,  test_set size = 3620
0.944198895028
Most Informative Features
                     hou = True              ham : spam   =     77.9 : 1.0
                  farmer = True              ham : spam   =     59.4 : 1.0
                     nom = True              ham : spam   =     54.3 : 1.0
                      cc = True              ham : spam   =     50.9 : 1.0
                thousand = True             spam : ham    =     49.2 : 1.0
                    2001 = True              ham : spam   =     48.0 : 1.0
                     ect = True              ham : spam   =     47.8 : 1.0
                      pm = True              ham : spam   =     40.9 : 1.0
              nomination = True              ham : spam   =     35.5 : 1.0
                    spam = True             spam : ham    =     34.5 : 1.0
                 generic = True             spam : ham    =     31.6 : 1.0
                      ex = True             spam : ham    =     31.6 : 1.0
                       | = True             spam : ham    =     31.6 : 1.0
                    logo = True             spam : ham    =     30.1 : 1.0
                   super = True             spam : ham    =     28.7 : 1.0
                 profile = True             spam : ham    =     28.7 : 1.0
                     men = True             spam : ham    =     28.7 : 1.0
                   woman = True             spam : ham    =     27.2 : 1.0
               investing = True             spam : ham    =     27.2 : 1.0
                      xl = True              ham : spam   =     26.8 : 1.0
Enter text to classify: investing
spam
Enter text to classify: investing farmer
ham
Enter text to classify: microsoft
spam
```