Python NLTK

Introduction to NLTK

- Initial Release 2001
- Open source software with Apache License version 2.0
- To check if NLTK is installed, do

```
import nltk
nltk.__path__
```

Downloading Corpus

- Diverse set of Corpa (nltk.corpus)
- In python command prompt: nltk.download()
- Linux default download location: '/<home_folder>/nltk_data/corpora/'
- Few basic data access methods
  - words()
  - sents()
  - paras()
  - raw()
- Different corpus reader classes which inherit base CorpusReader
  - PlaintextCorpusReader
  - WordlistCorpusReader
- Dataset eg: treebank, inaugural
- To create our own corpus

```
import nltk
from nltk.corpus import PlaintextCorpusReader
root = '/Users/rohitbhoopalam/rohit/uta_courses/
sem2/6339/6339Tutorial/wiki_sample_data'
new_corpus = PlaintextCorpusReader(root, '.*')
new_corpus.words()
```

Tokenizing Text

Custom tokenizing with regular expressions

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
tokenizer.tokenize('Eighty-seven miles to go, yet.  Onward!')
```

Finding frequent Bigram/Trigram

```
import nltk
from nltk.collocations import *
import re

bigram_measures = nltk.collocations.BigramAssocMeasures()
trigram_measures = nltk.collocations.TrigramAssocMeasures()

data = nltk.corpus.genesis.words('english-web.txt')
print len(data), "words data"

finder = BigramCollocationFinder.from_words(data)
finder.apply_word_filter(lambda x: False if re.match('\w', x) else
True)
print finder.nbest(bigram_measures.raw_freq, 20)


finder1 = TrigramCollocationFinder.from_words(data)
finder1.apply_word_filter(lambda x: False if re.match('\w', x) else
True)
print finder1.nbest(trigram_measures.raw_freq, 20)
```

Finding frequency distribution of word length in a corpus

```
import nltk
from nltk.corpus import inaugural
import matplotlib.pyplot as plt
```

```
data = inaugural.raw()

tokens = nltk.word_tokenize(data)

count_dict = {}
count_list = []
plot_list = []

for token in tokens:
    plot_list.append(len(token))
    try:
        count_dict[len(token)] += 1
    except KeyError:
        count_dict[len(token)] = 1


for i in range(max(count_dict.keys())):
    try:
        count_list.append(count_dict[i])
    except KeyError:
        count_list.append(0)

plt.hist(plot_list, normed=True)
print count_list
print count_dict.keys()
plt.show()
```

Sentence tokenizing

```
import nltk
from nltk.tokenize import sent_tokenize

data = """"""

sent = sent_tokenize(data)

print sent
print len(sent)
```

POS Tagging

```
import nltk

data = """ """

tokens = nltk.word_tokenize(data)
tagged = nltk.pos_tag(tokens)

print tagged
```

Possible list of tags in the default POS tagger can be found at http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Named Entity Recognizer

```
import nltk

data = “"" """

tokens = nltk.word_tokenize(data)

tagged = nltk.pos_tag(tokens)

entities = nltk.chunk.ne_chunk(tagged)

print entities
```

Sentiment analysis using Naive Bayes Classifier

```
"""
Reference: http://streamhacker.com/2010/05/10/text-classification-
sentiment-analysis-naive-bayes-classifier/
"""
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import movie_reviews

def word_feats(words):
    return dict([(word, True) for word in words])

negids = movie_reviews.fileids('neg')
posids = movie_reviews.fileids('pos')

negfeats = [(word_feats(movie_reviews.words(fileids=[f])), 'neg') for
f in negids]
posfeats = [(word_feats(movie_reviews.words(fileids=[f])), 'pos') for
f in posids]

negcutoff = len(negfeats)*3/4
poscutoff = len(posfeats)*3/4

trainfeats = negfeats[:negcutoff] + posfeats[:poscutoff]
testfeats = negfeats[negcutoff:] + posfeats[poscutoff:]
print 'train on %d instances, test on %d instances' %
(len(trainfeats), len(testfeats))

classifier = NaiveBayesClassifier.train(trainfeats)
print 'accuracy:', nltk.classify.util.accuracy(classifier, testfeats)
classifier.show_most_informative_features()
```