Lab #2: Document Similarity using NLTK and Scikit-Learn

DUE:

Tuesday 1/21 11:59pm

HOW TO SUBMIT:

All files should be submitted through WebSubmit (https://www.cs.duke.edu/csed/websubmit/). **Only one of your team members needs to submit on behalf of the team.** On the WebSubmit interface, make sure you select `compsci290` and the appropriate lab number. You can submit multiple times, but please have the same team member resubmit all required files each time. To earn **class participation credit**, submit a text file `team.txt` listing members of your team *who are present at the lab*. To earn **extra credit for the lab challenge**, submit the required files for challenge problems (see **WHAT TO SUBMIT** below).

A Brief Tutorial on Text Processing Using NLTK and Scikit-Learn

In homework 2, you performed tokenization, word counts, and possibly calculated tf-idf scores for words. In Python, two libraries greatly simplify this process: NLTK - Natural Language Toolkit (http://nltk.org/) and Scikit-learn (http://scikit-learn.org/stable/). NLTK provides support for a wide variety of text processing tasks: tokenization, stemming, proper name identification, part of speech identification, and so on. Scikit-learn (generally speaking) provides advanced analytic tasks: tfidf, clustering, classification, etc.

A Tour Through Shakespeare

In class, we did a basic word count of Shakespeare using the command line. Let's use NLTK for the same task.

Tokenization in NLTK

In [2]:
```
import nltk
import string

from collections import Counter

def get_tokens():
```

```
    with open('/opt/datacourse/data/parts/shakes-1.txt', 'r') as shakes:
        text = shakes.read()
        lowers = text.lower()
        #remove the punctuation using the character deletion step of translate
        no_punctuation = lowers.translate(None, string.punctuation)
        tokens = nltk.word_tokenize(no_punctuation)
        return tokens

tokens = get_tokens()
count = Counter(tokens)
print count.most_common(10)
```

```
[('the', 705), ('i', 699), ('and', 620), ('to', 532), ('you', 481), ('of', 47
6), ('a', 460), ('my', 378), ('that', 324), ('in', 300)]
```

Stop Word Removal

These are uninformative, so let's remove the stop words.

In [14]:
```
from nltk.corpus import stopwords

tokens = get_tokens()
filtered = [w for w in tokens if not w in stopwords.words('english')]
count = Counter(filtered)
print count.most_common(100)
```

```
[('lord', 207), ('parolles', 175), ('bertram', 135), ('helena', 125), ('king',
124), ('lafeu', 118), ('shall', 115), ('first', 107), ('countess', 100), ('tho
u', 94), ('sir', 92), ('well', 92), ('good', 90), ('thy', 85), ('would', 84),
('know', 83), ('second', 76), ('thee', 76), ('clown', 67), ('love', 60), ('dian
a', 59), ('say', 59), ('one', 55), ('hath', 52), ('ill', 52), ('tis', 52), ('up
on', 51), ('enter', 50), ('make', 49), ('yet', 49), ('o', 49), ('soldier', 49),
('must', 47), ('come', 47), ('let', 47), ('may', 46), ('great', 46), ('th', 4
4), ('madam', 44), ('mine', 43), ('speak', 41), ('man', 41), ('give', 39), ('th
ink', 39), ('us', 38), ('honour', 37), ('take', 37), ('see', 37), ('go', 35),
('like', 35), ('ring', 33), ('son', 32), ('widow', 32), ('gentleman', 30), ('ex
it', 30), ('wife', 30), ('ever', 29), ('never', 29), ('away', 29), ('mother', 2
9), ('time', 29), ('exeunt', 28), ('rousillon', 28), ('act', 28), ('poor', 28),
('count', 28), ('much', 27), ('knave', 27), ('leave', 26), ('pray', 26), ('noth
ing', 26), ('whose', 25), ('life', 25), ('find', 24), ('young', 24), ('scene',
24), ('duke', 23), ('hear', 23), ('heaven', 23), ('two', 23), ('might', 23),
('tell', 22), ('though', 22), ('nature', 22), ('hand', 22), ('drum', 22), ('thi
ne', 22), ('ay', 22), ('done', 22), ('marry', 22), ('captain', 21), ('serve', 2
1), ('indeed', 21), ('maid', 21), ('god', 20), ('live', 20), ('hope', 20), ('fl
orence', 20), ('art', 20), ('answer', 19)]
```

Stemming using NLTK

We can also do stemming using NLTK using a Porter Stemmer.

But will this work well on Shakespeare's writings?

In [15]:
```
from nltk.stem.porter import *

def stem_tokens(tokens, stemmer):
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

stemmer = PorterStemmer()
stemmed = stem_tokens(filtered, stemmer)
count = Counter(stemmed)
print count.most_common(100)
```

```
[('lord', 225), ('parol', 175), ('bertram', 136), ('king', 136), ('helena', 12
5), ('lafeu', 118), ('shall', 115), ('first', 107), ('countess', 100), ('know',
100), ('good', 94), ('thou', 94), ('sir', 92), ('well', 92), ('thi', 85), ('wou
ld', 84), ('love', 77), ('second', 76), ('thee', 76), ('come', 69), ('clown', 6
7), ('say', 66), ('diana', 59), ('soldier', 59), ('make', 58), ('one', 58), ('h
ath', 52), ('ill', 52), ('ti', 52), ('let', 51), ('upon', 51), ('enter', 50),
('honour', 50), ('yet', 49), ('o', 49), ('must', 47), ('man', 47), ('great', 4
7), ('may', 46), ('give', 45), ('think', 45), ('th', 44), ('madam', 44), ('min
e', 43), ('speak', 42), ('take', 42), ('count', 41), ('like', 40), ('go', 39),
('see', 38), ('us', 38), ('widow', 37), ('time', 36), ('son', 35), ('mother', 3
4), ('ring', 34), ('wife', 32), ('gentleman', 30), ('exit', 30), ('ever', 29),
('never', 29), ('away', 29), ('duke', 29), ('knave', 29), ('act', 29), ('virgi
n', 28), ('exeunt', 28), ('live', 28), ('rousillon', 28), ('poor', 28), ('marr
i', 28), ('natur', 28), ('much', 27), ('life', 27), ('noth', 27), ('find', 27),
('leav', 27), ('hear', 26), ('pray', 26), ('whose', 25), ('hand', 25), ('drum',
25), ('heaven', 25), ('father', 24), ('thank', 24), ('friend', 24), ('serv', 2
4), ('young', 24), ('fortun', 24), ('scene', 24), ('god', 23), ('two', 23), ('m
ight', 23), ('hope', 23), ('tell', 22), ('though', 22), ('thine', 22), ('ay', 2
2), ('done', 22), ('look', 22)]
```

Porter-Stemmer ends up stemming a few words here (parolles, tis, nature, marry).

What is more interesting is the counts are different - in fact, so much so that the ordering has been affected. Compare the two lists, especially the bottom of them, and you'll notice substantial differences.

Tf-Idf in Scikit-Learn

With our cleaned up text, we can now use it for searching, document similarity, or other tasks (clustering, classification) that we'll learn about later on. Unfortunately, calculating tf-idf is not available in NLTK so we'll use another data analysis library, scikit-learn. Scikit-learn has a built in [Tf-Idf (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) implementation but we can use NLTK's tokenizer and stemmer to preprocess the text.

In [28]:

```
import nltk
import string
import os

from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem.porter import PorterStemmer

path = '/opt/datacourse/data/parts'
token_dict = {}
stemmer = PorterStemmer()

def stem_tokens(tokens, stemmer):
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

def tokenize(text):
    tokens = nltk.word_tokenize(text)
    stems = stem_tokens(tokens, stemmer)
    return stems

for subdir, dirs, files in os.walk(path):
    for file in files:
        file_path = subdir + os.path.sep + file
        shakes = open(file_path, 'r')
        text = shakes.read()
        lowers = text.lower()
        no_punctuation = lowers.translate(None, string.punctuation)
        token_dict[file] = no_punctuation

#this can take some time
tfidf = TfidfVectorizer(tokenizer=tokenize, stop_words='english')
tfs = tfidf.fit_transform(token_dict.values())
```

First, we iterate through every file in the Shakespeare collection, converting the text to lowercase and removing punctuation. Next, we initialize TfidfVectorizer. In particular, we pass the TfIdfVectorizer our own function that performs custom tokenization and stemming, but we use scikit-learn's built in stop word remove rather than NLTK's. Then we call fit_transform which does a few things: first, it creates a dictionary of 'known' words based on the input text given to it. Then it calculates the tf-idf for each term found in an article.

This results in a matrix, where the rows are the individual Shakespeare files and the columns are the terms. Thus, every cell represents the tf-idf score of a term in a file.

	acclam	accomod	accomodo	accompani	accomplic	accomplish
shakes-1	0	0	0.0007116	0.00245035	0	0
shakes-2	0	0	0.0008167	0	0	0
shakes-3	0	0	0	0	0	0
shakes-4	0.0098	0	0	0	0	0
shakes-5	0	0.00002	0	0	0	0.0000514

tfidf

The table represents a sample tf-idf entry from the Shakespeare files. In general, these should be small (or 0 if the term isn't present in the document).

Inputting a New Document

So now we have a collection tf-idf numbers, what can we do with it? Here are some options:

1. Compare documents in the set to other documents in the set, using cosine similarity
2. Search - query this existing set, as described below
3. Plagiarism - compare a new document to the set to find any potential matches

To do any of these, we have to input a new document (or existing) into the model and get a tf-idf answer back. We do this by using the transform function, which will use our existing NLTK preprocessor first.

In [51]:
```
str = 'this sentence has unseen text such as computer but also king lord julie
t'
response = tfidf.transform([str])
print response
```

```
  (0, 17796)    0.309281094362
  (0, 16548)    0.15276879967
  (0, 16480)    0.394772926561
  (0, 14559)    0.226939245918
  (0, 9676)     0.156737043549
  (0, 9025)     0.164996828044
  (0, 8926)     0.544613034225
  (0, 7404)     0.169300459104
  (0, 3403)     0.544613034225
```

Text it hasn't seen gets excluded, unless you call fit_and_transform(), which adds the document to the model, whereas transform does not.

We can get the specific terms and their tf-idf score (but it's not straightforward):

In [52]:
```
feature_names = tfidf.get_feature_names()
for col in response.nonzero()[1]:
    print feature_names[col], ' - ', response[0, col]
```

```
unseen  -  0.309281094362
thi  -  0.15276879967
```

```
text  -  0.394772926561
sentenc  -  0.226939245918
lord  -  0.156737043549
king  -  0.164996828044
juliet  -  0.544613034225
ha  -  0.169300459104
comput  -  0.544613034225
```

LAB EXERCISE

In this part of the lab, we will continue with our exploration of the Reuters data set, but using the libraries we introduced earlier and cosine similarity. First, let's install NLTK and Scikit-learn.

We'll install both NLTK and Scikit-learn on our VM using pip (https://pypi.python.org/pypi/pip), which is already installed.

First: Run the sync.sh script in your vm, this should install everything required.

If for some reason that didn't work

Run the following two commands from a terminal in the VM:

```
pip install nltk
pip install scikit-learn
```

We'll also need to install models from nltk. Open up a python shell (or Enthought Canopy), and type:

In [*]:
```
import nltk
nltk.download()
```

This should bring up a window showing available models to download. Select the 'models' tab and click on the 'punkt' package, and under the 'corpora' tab we want to downlod the 'stopwords' package. You should then have everything you need for the exercises.

If that hung: You can download the data manually from https://s3.amazonaws.com/textblob/nltk_data.tar.gz. Extract this file to ~/nltk_data.

IN CLASS EXERCISE

Please answer the following questions.

[Qn 1] For every organization, list the set of news articles that mentional that organization

[Qn 2] Calculate the TFIDFs for the organizations. *Hint: What is the document associated with an organization?*

[Qn 3] Find the top 10 salient sentences that describe each organization. *Hint: You will need to tokenize the documents to get sentences. You may write your own, or use the sentence tokenizer in NLTK.*

At the end of the class, each group will be asked to give their top 10 sentences for a randomly chosen organization.

CHALLENGE QUESTION

[Qn] Find the top-5 news articles that are most similar to an organization. *Hint: Use cosine similarity between documents*

WHAT TO SUBMIT FOR THE CHALLENGE

1. A file 'orgsSim.txt' that contains one line per organization. Output the organization string, followed by a list of 5 document ids (use the NEWID attribute of the REUTERS xml element). For instance,

   opec:::(aa, bb, cc, dd, ee)

   ecafe:::(aa, bb, cc, dd, ee)

   fao:::(aa, bb, cc, dd, ee)