CIS 430/530 Fall 2012 HW 1

Instructor: Ani Nenkova
TA: Kai Hong, Jessy Li, Achal Shah

Released: September 10, 2012
Due: 11:59PM September 26, 2012

Overview

This assignment focuses on helping you get started with using NLTK and Python by performing simple text analysis. You will also be asked to use a topic-word tool to extract topic words from a collection of corpus. You may find the first two chapters of the NLTK book very useful. All the files you need to finish the homework are in: `'/home1/c/cis530/hw1/'`

You will be asked to submit the code for the functions you have implemented. There are in total 100 points for the five homework problems in this homework. Teams of two should submit one homework, with two people's name and E-mail at the beginning of the code. For example, if Achal works with Jessy, then the beginning of code should be like:

```
# Achal Shah: achals@seas.upenn.edu
# Jessy Li: ljunyi@cis.upenn.edu
```

NOTE: If you do not have an account on Eniac, please contact the TA immediately.

Submitting your work

The code for your assignment should be placed in a single file called hw1 code yourpennkey.py where yourpennkey is your Penn Key. Since my (Kai) pennkey is "hongkai1", I would submit the file named: hw1_code_hongkai1.py. You should also submit two topic signature files: Starbucks.ts and H.J.Heinz.ts.

To electronically submit homework, if you are not already working on Eniac, you need to place the file containing your solution on your SEAS account storage. One way to do this would be to use an SFTP client such as FileZilla or WinSCP.

Then connect via ssh to seas.upenn.edu and use the turnin command to submit your files for grading:

```
% turnin -c cis530 -p hw1 hw1_code_yourpennkey.py Starbucks.ts H.J.Heinz.ts
```

This should print out a confirmation message. You can run turnin multiple times before the deadline. Each time you run turnin, it overwrites your previous submission for that assignment. You can check that the homework was submitted successfully:

```
% turnin -c cis530 -v
```

This will show you the list of file(s) you have submitted.

Code Guidelines

You can use in-built NLTK modules whenever possible unless specified otherwise. However, only import the modules that you are actually using. For example:

```
from nltk import * # Bad!
from nltk import FreqDist, ConditionalFreqDist # Good!
```

You may write any extra functions that you think are necessary, but all the functions defined in this document should be present. Please add descriptive comments to all functions that you write. Do not do any preprocessing/filtering of data unless explicitly mentioned. For example, the words of a set of documents, refers to all tokens, even non-alphabetic ones.

Your code will be automatically graded by calling your functions from one script. We will be giving you solutions to the problems by running on a relatively small corpus so that to help you checking your answers (`'/home1/c/cis530/hw1/hw1_ans.txt'`). However, your score will be graded on a much larger corpus.

Data

Rather than loading the Corpus directly from NLTK, we will be using data provided by Wharton Research Data Services, which includes CIQ Key Development news subset and daily returns data for 50 randomly selected companies out of Russell 1000. For the first homework, we will be using data from two corporations: H.J.Heinz and Starbucks.

You will be given two kinds of dataset: **Small Dataset** and **Full Dataset**. The small set will be useful for you to check your answer, and the homework will be graded on the full dataset. Topic signatue (*.ts*) files are necessary to accomplish most of the problems. Please generate the .ts files by following instructions from Problem 2.2. Below is where you can find the files:

```
>>> corpus_root_small = '/home1/c/cis530/hw1/data/small'
>>> corpus_root_full = '/home1/c/cis530/hw1/data/full'
```

Here is an example of how to load files from a particular directory to memory. You can refer to the NLTK book (`http://nltk.googlecode.com/svn/trunk/doc/book`) for more details.

```
# Import the corpus reader
>>> from nltk.corpus import PlaintextCorpusReader

# Define the folder where the files are located
>>> corpus_root = '/home1/c/cis530/hw1/data/small'

# Read all files in that directory
>>> files_all = PlaintextCorpusReader(corpus_root, '.*')

# List the first five files within this directory
>>> files_all.fileids()[:5]
['H.J.Heinz/119185.txt', 'H.J.Heinz/62769.txt', 'H.J.Heinz/63422.txt', 'H.J.Heinz/89164.txt',
'H.J.Heinz/92696.txt']

# Total number of files belonging to this directory
>>> len(files_all.fileids())
10
```

1 Loading Corpus (10 Points)

As the directories and files are already located in Eniac, we need to write a few functions to help loading sentences and tokens from directories.

1.1 Getting Files and Directories (4 Points)

a) (2 points) Write a function `get_sub_directories(directory)` to list all sub-directories from this directory. **(Relative path is enough)**

```
>>>get_sub_directories(corpus_root)
  ['H.J.Heinz', 'Starbucks']
```

b) (2 points) Write a function `get_all_files(directory)` to get all files both in this directory and its sub-directories. **(Relative path is enough)**

```
>>> get_all_files(corpus_root)
  ['H.J.Heinz/119185.txt', 'H.J.Heinz/62769.txt', 'H.J.Heinz/63422.txt', 'H.J.Hein
  z/89164.txt', 'H.J.Heinz/92696.txt', 'Starbucks/50778.txt', 'Starbucks/55370.txt
  ', 'Starbucks/55849.txt', 'Starbucks/57664.txt', 'Starbucks/60383.txt']
```

1.2 Reading Sentences and Tokens (6 Points)

a) (1.5 Points) Write a function `load_file_sentences(filepath)` that takes a full `path` and returns a list of all sentences in that file. You will need to load the file as one continuous string and then use nltk's functionality to split the file by sentence. Refer to the nltk book and API to learn how to do this. **Convert all sentences to lowercase.**

```
>>> load_file_sentences(corpus_root + '/Starbucks/50778.txt')
  ["borders group inc., starbucks corp. begun to offer wireless 'hotspots'.", "bor
  ders group inc., starbucks corp. and others have all begun to offer wireless 'ho
  tspots' in their outlets, giving patrons the ability to check e-mail or surf the
   net away from home or office.", ...]
```

b) (1.5 Points) Write another function `load_collection_sentences(directory)` to load all sentences in files within this directory. You might need the function `get_all_files(directory)` above. The return should be a list of sentences.

c) (1.5 points) Write a function `load_file_tokens(filepath)` which takes a full `path` and returns a list of all tokens (not vocabulary!) in this file. In order to manipulate this, you might use the function `nltk.word_tokenize(str)` from nltk.

```
>>> load_file_tokens(corpus_root + '/Starbucks/57664.txt')
  ['starbucks', 'reports', 'january', '2002', 'revenues', ';', 'reports', 'stronge
  st', 'monthly', 'comparable', 'store', 'sales', 'growth', 'in', 'last', '12', 'm
  onths', 'and', 'expects', 'full', 'year', 'revenues', ...]
```

d) (1.5 points) Write another function `load_collection_tokens(directory)` to load tokens in all files within this directory. The return should also be a list of tokens.

2 Most Frequent Words and Topic Words(20 points)

2.1 High Frequency Words (9 Points)

a) (5 Points) In one document, the words appeared a lot can be meaningful for this document. Write a function `get_top_words(path, n)` to load the most frequent n words associated with the file or collection (Your code should be able to identify whether the `path` is directory or file).

NOTE: You may find the NLTK function `FreqDist` useful. **Don't** exclude any punctuation or stopwords at this point.

```
>>> get_top_words(corpus_root, 100)
  ['the', ',', 'in', 'for', 'heinz', 'to', '.', 'of', 'a', 'and', 'starbucks', '20
  02', 'its', '$', 'fiscal', 'company', 'revenues', 'million', 'co.', ...]
```

b) (4 Points) You may have noticed that some articles, prepositions, punctuation and some common words have appeared within the Top Words. This is actually one of the potential problem counting would bring about. One way to deal with it is to use a stopword-list to eliminate some words. Use the stopword-list at `'/home1/c/cis530/hw1/stoplist.txt'`.

Write a function `get_top_words_with_stoplist(path, n)` to get the most frequent n words within this file or collection after deleting stopwords.

```
>>> get_top_words_with_stoplist(corpus_root, 100)
  ['heinz', 'starbucks', '2002', 'fiscal', 'company', 'revenues', 'million', 'co.'
  , 'period', 'sales', 'coffee', 'food', 'squirt', 'wireless', 'net', ...]
```

2.2 The Topic Word Tool (5 points)

A topic word is a word with significantly greater probability in a given text compared to that in a large background corpus. For each word in the text, the ratio between two hypotheses (a) and (b) is computed.

a) The word is not a topic word and so its probability would be the same in both the given text and the background collection.

b) The word is a topic word and hence its probability in the given text is greater.

This ratio λ is estimated and $-2 * log(\lambda)$ has a χ^2 distribution. Topic signatures are a very useful feature for content selection from source documents in summarization. More information on how topic words are calculated can be found here: `http://research.microsoft.com/en-us/people/cyl/col2000-final.pdf`

You will not have to implement this algorithm. We have provided a tool that will calculate topic words on a collection of documents in a directory. First, however, you must create a configuration file for the tool to use. A sample configuration file can be found at `/home1/c/cis530/hw1/TopicWords-v1/config.example` Make a copy of the file for yourself. You can use the `cp` command to copy it into your Eniac home directory (Just copy the configuration file would be enough):

```
cp /home1/c/cis530/hw1/TopicWords-v1/config.example ~
```

NOTE: You can use the `pwd` command to print your current working directory in the bash prompt.

Use a text editor (`emacs` or `vim`) to edit the config file on ENIAC. In particular, you will need to change the `inputDir` and `outputFile` fields when you run the tool on Heinz and Starbucks, and on Small/Full dataset. Pointing the tool to a directory will instruct it to process all of the documents in that directory and

merge their topic words into a single list. Play with the tool and see what result you might get for different collection of files.

To run the tool, navigate back to the directory containing the topic word tool:

```
cd /home1/c/cis530/hw1/TopicWords-v1/
```

Then use this command to execute the tool (using the config file you modified in your home directory):

```
java -Xmx1000m TopicSignatures ~/config.example
```

This will create a `.ts` file somewhere in your home directory, depending on how you configured the `outputFile` field of the configuration file. The created file will contain lists of words according to their χ^2 scores, where higher scores mean a word is more topical.

Solutions to this problem will be graded by checking the `.ts` file for both Starbucks and H.J.Heinz working on the large corpus (even though the examples use the `.ts` result working on small corpus). Adjusting the cutoff to 0.1 so that we can check your answer. For this problem, you don't need to write code. You just need to understand usage of this toolkit and modify the config.example file. Remember submitting the `Starbucks.ts` and `H.J.Heinz.ts` file together with the .py file.

NOTE: This homework uses a specially modified version of this tool. Use only the version on ENIAC.

2.3 Top Topic Words (6 Points)

(a) (3 Points) Write a function `load_topic_words(topic_file)` that, given a path to a `.ts` file, returns a **dictionary** of type {`string : float`} mapping the topic words loaded from the file to their χ^2 scores. **ts_root** is where you put your .ts files.

```
>>> load_topic_words(ts_root + '/Starbucks.ts')
  {'shop': 5.1608999999999998, 'limited': 1.7412000000000001, 'stores': 10.8757, '
  office': 0.63619999999999999, 'consolidated': 23.073499999999999, 'results': 1.6
  979, 'installed': 2.5672999999999999, 'ended': 13.9718, ...}
```

NOTE: Be sure to cast the χ^2 scores to floats when loading the `.ts` file.

(b) (3 Points) Write a function `get_top_n_topic_words(topic_words_dict, n)` that, given a topic word dict in the format above, returns the top `n` **unique** topic words from that dict, according to their χ^2 scores. Sort the returned words in descending order of χ^2.

```
>>> tw = load_topic_words(ts_root + '/Starbucks.ts')
>>> get_top_n_topic_words(tw, 10)
  ['starbucks', 'fiscal', 'revenues', 'coffee', 'consolidated', ... ]
```

3 Measuring Similarity (30 points)

3.1 Vector Space (15 points)

We will be comparing the similarity between documents for this problem. Each document could be seen as a sentence list, such as below:

```
["heinz unveils new blue ketchup.",
 "hj heinz co., which has sold the condiment in red, green, purple, pink, orange and teal,
 is adding blue to its palette.",  ...]
```

Create a pair of functions that will do the following:

(a) (8 Points) Write a function `create_feature_space(list)` to create a Python `Dictionary` mapping each unique word type in all of the sentences to a consecutive integer starting from zero (order doesn't matter). This creates a mapping between each word and the position to represent this word in vector.

```
>>> sentences = ["this is a test", "this is another test"]
>>> create_feature_space(sentences)
  {'this': 0, 'a': 1, 'is': 2, 'test': 3, 'another': 4}
```

(b) (7 Points) Write a function `vectorize(feature_space, str)` which takes a **string**(sentence) `str` as well as feature space as input and returns a vector $\vec{v}$ where each element v_i of that vector is set to 1 if it contains the i^{th} word in feature space, 0 otherwise. You should also deal with the case when word in sentence **didn't** appear in **feature_space**. You need to first chop the sentence into tokens using NLTK functions.

```
>>> feature_space = create_feature_space(sentences)
>>> vectorize(feature_space, "another test")
  [0, 0, 0, 1, 1]
```

3.2 Similarity Metrics (15 points)

Implement the following three vector similarity metrics (described in more detail on page 299 of Manning and Schütze, `http://cognet.mit.edu/library/books/view?isbn=0262133601`).

$$\text{Dice coefficient: } \frac{2|X \cap Y|}{|X| + |Y|} = \frac{2 \times \sum_{i=1}^{N} min(v_i, w_i)}{\sum_{i=1}^{N} (v_i + w_i)}$$

$$\text{Jaccard coefficient: } \frac{|X \cap Y|}{|X \cup Y|} = \frac{\sum_{i=1}^{N} min(v_i, w_i)}{\sum_{i=1}^{N} max(v_i, w_i)}$$

$$\text{Cosine: } \frac{|X \cap Y|}{\sqrt{|X| \times |Y|}} = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

In particular, write three functions:

- `jaccard_similarity(X, Y)`
- `dice_similarity(X, Y)`
- `cosine_similarity(X, Y)`

where `X` and `Y` are vectors and the similarities as above show. Below is an example:

```
>>> dice_similarity([0,1,0], [0,1,1])
  0.6666666666666666
>>> jaccard_similarity([0,1,0], [0,1,1])
  0.5
>>> cosine_similarity([0,1,0], [0,1,1])
  0.7071067811865475
```

4 Similarity Between Documents (15 Points)

From Problem 2, we know how to get **Top k word**, **Top k words excluding stopwords** and **Top k topic signatures** for a collection of documents. We are asking which of the three can represent a document the best.

a) (5 Points) To compare these three representations, we calculate the similarity between documents. The intuition is that documents from one corporation should be similar to documents of the same corporation. Even though it's not always the case, with a relatively large dataset (800+ documents), most of the time the most similar documents are from the same corporation.

To do this, we first extract **Top 50** words from Starbucks and H.J.Heinz respectively, with these three approaches. Denote the Top 50 words extracted from Starbucks with these three approaches as $S_1.S_2, S_3$, words extracted from H.J.Heinz as $H_1.H_2, H_3$, then we can get the combined word lists W_1, W_2, W_3 as:

$$W_1 = List(S_1 \cup H_1)$$
$$W_2 = List(S_2 \cup H_2)$$
$$W_3 = List(S_3 \cup H_3)$$

These are the only words needed for us to represent a document, the other words are just ignored. Seen a document as a list of words, $D' = \{d'_1, d_2, \ldots, d'_k\}$ and given a word list W, we can generate the vector $\vec{V}$ according to D' and W, where $\vec{V}$ and W have same dimensions.

$$v_i = \begin{cases} 1, & w_i \in D'; \\ 0, & w_i \notin D'. \end{cases}$$

Write a function `get_doc_vector(D', W)` which takes a **Set** of words D' for one document, a word-list W and output a vector-list $\vec{V}$ to represent the document.

b) (10 Points) Now we know how to represent documents given a word-list. In the corpus, there are k_1 Starbucks announments, k_2 H.J.Heinz announcements, in total $k_1 + k_2$ files. We assign the documents belonged to Starbucks as $S = \{s_1, s_2, \ldots, s_{k_1}\}$, documents belonged to Heinz as $H = \{h_1, h_2, \ldots, h_{k_2}\}$. The document set can then be represented as: $D = S \bigcup H = \{d_1, d_2 \ldots d_{k_1+k_2}\}$. For each document d_i, Denote the similarity between d_i and another document d' as $Sim(d_i, d')$ (We use **cosine similarity** here). After calculating all pairs: $Sim(d_i, d'), d' \in D$, we could have two scores, representing the average Cosine Similarity this document is to Starbucks and Heinz separately. For simplicity, similarity of a document to itself $Sim(d_i, d_i) = 1$ is also included in calculation.

$$Score\_S(d_i) = \frac{1}{k_1} \sum_{j=1}^{k_1} Sim(d_i, s_j), \; Score\_H(d_i) = \frac{1}{k_2} \sum_{j=1}^{k_2} Sim(d_i, h_j)$$

Suppose the document d_i is an announcement from Starbucks, then the more $Score\_S(d_i) - Score\_H(d_i)$ is, the better this metric is. Thus we can evaluate how confident it is by calculating $Score\_S(d_i) - Score\_H(d_i)$. If the file is from H.J.Heinz, we would calculate $Score\_H(d_i) - Score\_S(d_i)$ instead. Thus we have:

$$Score(d_i) = \begin{cases} Score\_S(d_i) - Score\_H(d_i), & d_i \in S \\ Score\_H(d_i) - Score\_S(d_i), & d_i \in H \end{cases}$$

And the accuracy of this metric is:

$$Accuracy(Metric) = \frac{1}{k_1 + k_2} \sum_{i=1}^{k_1+k_2} Score(d_i)$$

Writing a function `get_doc_sim(directory)` to get the Accuracy when using Top-50 words from each collection to represent a document. Your output should be a list of floats, in the order of **Top k word**, **Top k words excluding stopwords** and **Top k topic signatures**. Think about why result is like this.

```
>>> get_doc_sim(corpus_root)
  [2.7749376946000184, 3.700248499458183, 3.760018412932773]
```

Your score on problem 4.(b) will be graded according to:
(i) Derive the correct relative order of these three. (5 Points)
(ii) Generating the correct value. (5 Points)

5 Word Similarity (25 Points)

5.1 Word Contexts (15 points)

a) (10 points) Write a function `get_word_contexts(word, directory)` which returns the context of a given word in directory. Here context means the word left to (Left-gram) and right to (Right-gram) the target word. (We don't distinguish these two for the problem). For example, for the target word **'love'**, suppose we have two sentences: 'i love you', 'you love him' in corpus, then the context around 'love' should be a list: ['i', 'you', 'him'] (order doesn't matter). Output of your function should be a list (not a generator) of strings representing each **unique** unigram surrounding this word **(remove duplicates)**.

```
>>> get_word_contexts('coffee', corpus_root + '/Starbucks')
  ['shop', 'a', 'giant', 'brewer', 'pintsize', 'huge', 'taste', 'starbucks', 'co.'
  , 'open', 'seattle-based']
```

b) (5 points) Write a function `get_common_contexts(word1, word2, directory)` to return the intersection of contexts for two words within a given directory. Think about why result is like this.

```
>>> get_common_contexts('co.', 'company', corpus_root + '/H.J.Heinz')
  ['heinz', 'announced']
```

5.2 Comparing Word Similarity (10 Points)

Write a function `compare_word_sim()` to compute the similarity between **TOP-10 Topic Words** in **Starbucks** corpus. Your output should be a 10×10 matrix. Here we compute similarity by comparing the cosine similarity between word contexts. You may need to do the following things to accomplish this task:

a) Load the Top-10 Topic Words for Starbucks (For grading, you should use the .ts file for Starbucks collection from the large dataset).

b) Get word contexts for the Top-10 words in the approach described in 5.1(a) .

c) Create a feature space according to word contexts. (You can write auxiliary functions with similar functionality to `create_feature_space(list)` and `vectorize(feature_space, sentence)` to generate feature space and then vectorize the contexts for each word)

d) Compute cosine similarity between words and generate the 10×10 matrix.

```
>>> compare_word_sim(corpus_root)
  [[1.0, 0.19245008972987526, 0.0, 0.1421338109037403, 0.13608276348795434, 0.13608276348795434,
  0.10540925533894598, 0.07856742013183861, 0.11785113019775793, 0.0], [0.19245008972987526, 1.0,
  0.16666666666666666, 0.0, 0.0, 0.23570226039551587, ... ]]
```