

CIS 430/530 Fall 2011 HW 1

Instructor: Ani Nenkova
TA: Alexander Shoulson

Released: September 16, 2011
Due: 11:59PM September 30, 2011

Overview

This assignment focuses on helping you get started with using NLTK and Python by performing simple text analysis. You will also begin to develop intuitions about different analytical techniques. The first two chapters of the NLTK book should suffice for this assignment. For information about how to get started with the NLTK tool on eniac, please see the instructions posted at:

http://www.cis.upenn.edu/~cis530/hw_2011/generalinfo.pdf

Deliverables

You will be asked to submit the code for the functions you have implemented as well as a brief writeup of your observations and results. Write ample comments in your code.

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. The writeup should be in plain text format and named as `hw1_writeup_yourpennkey.txt`. Since my (Alex) pennkey is “shoulson”, I would submit the following files: `hw1_code_shoulson.py`, `hw1_writeup_shoulson.txt`

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 hw1_writeup_yourpennkey.txt
```

This should print out a confirmation message. If you are prompted for a section name, type ‘ALL’. 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 and matplotlib 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 helper 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.

Do not add any code other than variables in the body of your file. All global variables must have `__yourpennkey_` as a prefix. This is useful for caching loaded data to make your code faster, for example:

```
__shoulson_loaded_words = None

def load_data():
    if __shoulson_loaded_words is None:
        __shoulson_loaded_words = brown.words(categories=None)
    return __shoulson_loaded_words
```

This is the **only** appropriate use of code in the body of your file. Place any test or execution code in your `if __name__ == '__main__':` block.

Data

For these problems we will be using the Brown Corpus. This corpus is explained in Chapter 2, Section 1 of the NLTK book (<http://nltk.googlecode.com/svn/trunk/doc/book/ch02.html>). The 1.15M words of this corpus are organized into categories based on genre (news, reviews, etc.). You can access this corpus as follows:

```
# Import the Corpus
>>> from nltk.corpus import brown

# Get all of the words (1.15M) from the corpus
>>> corpus_text = brown.words()
# NB: You can also use brown.words(None)

# Get the text from the reviews category
>>> reviews_text = brown.words(categories='reviews')

# Print out the first few words of the whole corpus
>>> corpus_text
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

# Print out the first few words of the reviews section
>>> reviews_text
['It', 'is', 'not', 'news', 'that', 'Nathan', ...]

# Count the words in the whole corpus
```

```
>>> len(corpus_text)
1161192
```

1 Counting Words (55 points)

1.1 Probability Distribution of Words (5 points)

- a) Write a function `get_prob_word_in_category(word, category)` that returns the probability of the given word appearing in the given category (or the entire corpus, if no category is given).

```
>>> get_prob_word_in_category('president', 'government')
0.0005270799769278198
```

For *this* question, you should not use the `FreqDist` built-in class from NLTK.

1.2 Types and Tokens (15 points)

- a) (5 points) Write a function `get_vocabulary_size(category)` that returns the size of the vocabulary for a single category from the corpus. If no category is given, the function should return the vocabulary size for the entire corpus.

```
>>> get_vocabulary_size()
56057
>>> get_vocabulary_size('news')
14394
```

- b) (5 points) Write a function `get_type_token_ratio(category)` that returns the type to token ratio for the given topic. Tokens are the number of words (which includes alphanumeric entries, punctuations). Types are the unique words (which includes alphanumeric entries, punctuations).

```
>>> get_type_token_ratio()
0.048275392872152066
>>> get_type_token_ratio('news')
0.14314696580941583
```

- c) (5 points) Write a function `get_entropy(category)` that returns the entropy of the given category's word distribution (or the whole corpus if no category is given). You have already computed the probability distribution in the first part of the question. For the definition of entropy refer to the course notes.

1.3 Word Frequency (35 points)

- a) (5 points) Write functions `get_top_n_words(n, category)` and `get_bottom_n_words(n, category)` that return the most frequent and least frequent n words from a category (or the entire corpus).

```
>>> get_top_n_words(5)
['the', ',', '.', 'of', 'and']
>>> get_top_n_words(5, 'news')
['the', ',', '.', 'of', 'and']
>>> get_bottom_n_words(5)
['pigen', 'mosaics', 'Lust', 'northerly', 'jawbone']
>>> get_bottom_n_words(5, 'news')
['alienated', 'Dorenzo', 'Marsicano', 'Frog-marched', 'Ceil']
```

Hint: Use `FreqDist` (NLTK book chapter 1)

- b) (5 points) [WRITEUP] Find the top and bottom 30 words for the full corpus. What do you notice? Why do you think this is?
- c) (10 points) We will use the `matplotlib` library to visualize some of our results (it's already installed on Eniac). Using the `matplotlib.pyplot.hist()` function, we want to display a histogram of word frequency. Write a function `plot_word_counts()` that produces a word frequency histogram **for the news category of the corpus**. For this plot, the X axis will be number of occurrences of a word (say, 60 times), and the Y axis will be the number of words that appear that many times. The `bins` parameter will be the number of bins to use for the histogram (say, 1000).

Here's some sample code for plotting a histogram of random points generated by a Gaussian distribution with $\mu = 0$ and $\sigma = 1$:

```
import random
import matplotlib.pyplot as plt

def plot_random():
    # Generate a list of 100,000 random numbers from a
    # Gaussian distribution with mean 0 and std. dev. 1
    x = [random.gauss(0, 1) for i in range(100000)]

    # Prepare the histogram
    plt.hist(x, bins=3000)

    # Annotate the graph
    plt.xlabel('Random Value')
    plt.ylabel('Value Frequency')
    plt.title('Random Values by Frequency')

    # Set the axis ([Min X, Max X, Min Y, Max Y])
    plt.axis([-4, 4, 0, 200])

    plt.show()
```

For your function, try using the following values:

- Bins: 3,000
 - X Axis Range: [0, 500]
 - Y Axis Range: [0, 500]
- d) (5 points) [WRITEUP] Describe the distribution (you do not need to include a picture). What do you notice about the shape of the graph? Why do you think this is?
- e) (5 points) [WRITEUP] Look at the categories “news”, “fiction”, “editorial”, and “religion”. Which has the most lexical diversity? Which has the least? Why?
- f) (5 points) [WRITEUP] For simplicity, we did not perform any normalization on the text (removing punctuation, converting everything to lowercase, etc.). Do you think standardizing the text in this way would change our results? What sort of information would be gained or lost? When would you want to normalize the data?

2 Context and Similarity (60 points)

2.1 Word Contexts (20 points)

- a) (10 points) Write a function `get_word_contexts(word)` that returns each context for the given word **in the news category of the corpus**. For the sake of this problem, the context of word is the pair of tokens immediately before and immediately after the word (two in total). For simplicity, if the word appears at the beginning or end of the corpus, do not return that context. The output of your function should be a list (not a generator) of pairs of strings representing each **unique** context (remove duplicates), as follows:

```
>>> get_word_contexts('U.N.')
[('the', '.'), ('his', 'Ambassador'), ('the', 'or'), ('the', 'forces'), ('obstructed',
'efforts'), ('the', 'troops'), ('the', 'to'), ('the', 'into'), ('the', 'army'), ('the',
'three'), ('the', 'eject'), ('at', 'headquarters')]
```

- b) (5 points) Write a function `get_common_contexts(word1, word2)` that returns the **unique** contexts shared by word1 and word2 **in the news category of the corpus**, as follows:

```
>>> get_common_contexts('he', 'she')
[(',', 'in'), ('', 'said'), ('which', 'has'), ('', 'went'), ('what', 'could'),
('said', 'did'), ('and', 'and'), ('', 'has'), ('that', 'was'), ('when', 'was')]
```

- c) (5 points) [WRITEUP] Find the average number of shared contexts in the news category for each pair of the following city names: “Washington”, “Philadelphia”, “Boston”, “London”.

2.2 Measuring Similarity (40 points)

We will be comparing the similarity between a number of sentences, given as a list of strings:

```
["the quick brown fox jumped over the lazy dog",
 "the fox leaped over the tired dog",
 "the fox",
 "the lazy dog the lazy dog the lazy dog"]
```

- a) (10 points) Create a pair of functions that will do the following:

- `create_feature_space(sentence_list)` will create a Python dict 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 element in each vector that will represent it.

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

- `vectorize(feature_space, sentence)` takes a sentence 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 the feature space, zero otherwise.

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

- b) (10 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> and on page 666 of J&M):

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

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

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

In particular, write three functions `jaccard_similarity(X, Y)`, `dice_similarity(X, Y)`, and `cosine_similarity(X, Y)` that take two vectors `X` and `Y` and return their similarity metrics.

- c) (5 points) [WRITEUP] For each of the three metrics, rank each pair of sentences (six in total, do not compare a sentence to itself) in descending order of similarity.
- d) (5 points) [WRITEUP] What can you say from these rankings? How are these three metrics similar and different?
- e) (10 points) [WRITEUP] Write code for a fourth metric of your own design. Rank the six pairs by this metric and discuss how it compares to the others.

3 Modeling Word Distributions (45 points)

Based on text in the Brown corpus, we will build language models and use these to compute for a set of given sentences, the probability that they were generated by this language model.

3.1 A Sliding Window (10 points)

- a) Create a function `make_ngram_tuples(samples, n)` that returns a sequence of all the `n`-grams seen in the input, in order. The function returns a sequence of tuples where each tuple is of the form (context, event). The context is `None` (the Python built-in value `None`, not the string “None”) in the case of `n = 1` (unigrams), and for larger values of `n` it is a tuple of the preceding `n - 1` words for each sample.

```
>>> samples = ['her', 'name', 'is', 'rio', 'and', 'she', 'dances', 'on', 'the', 'sand']
>>> make_ngram_tuples(samples, 1)
[(None, 'her'), (None, 'name'), (None, 'is'), (None, 'rio'), (None, 'and'), (None, 'she'),
 (None, 'dances'), (None, 'on'), (None, 'the'), (None, 'sand')]
>>> make_ngram_tuples(samples, 2)
[(('her',), 'name'), (('name',), 'is'), (('is',), 'rio'), (('rio',), 'and'),
 (('and',), 'she'), (('she',), 'dances'), (('dances',), 'on'), (('on',), 'the'),
 (('the',), 'sand')]
>>> make_ngram_tuples(samples + samples, 2)
[(('her',), 'name'), (('name',), 'is'), (('is',), 'rio'), (('rio',), 'and'),
 (('and',), 'she'), (('she',), 'dances'), (('dances',), 'on'), (('on',), 'the'),
 (('the',), 'sand'), (('sand',), 'her'), (('her',), 'name'), (('name',), 'is'),
 (('is',), 'rio'), (('rio',), 'and'), (('and',), 'she'), (('she',), 'dances'),
 (('dances',), 'on'), (('on',), 'the'), (('the',), 'sand')]
```

Note that in Python a tuple displayed as `(x,)` is a tuple consisting of one element, `x`. The comma is there simply to establish that this is a tuple, and not just parentheses around a value. The final usage example makes it clear that this function simply delivers all n-grams seen in order and does not remove any duplicates; this is a crucial behavior of this function.

3.2 Building Language Models (20 points)

a) (15 points) Define a class `NGramModel` with the following instance methods.

- `def __init__(training data, n)`: builds a n-order language model using the list of tokens supplied in training data. Hint: Use the `make_ngram_tuples` function from the previous question to make the n-order language model
- `prob(context, event)`: returns the probability of the event given the context. Depending on the value of `smooth` with which the language model was initialized, the probability estimate should be smoothed with the appropriate method or an unsmoothed estimate will be returned. At this point, write the `prob` function such that it works for the unsmoothed version. You will implement smoothing in the next question. Hint: Always try to check your code on a small list of words. You can always find some words from the brown corpus or you can make one list for yourself. For finding the probability you can use the conditional frequency distribution (NLTK book Chapter 2).

```
>>> words = nltk.corpus.brown.words()[0:100]
>>> words
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> model = NGramModel(words,2)
>>> model.prob( ('The',), 'Fulton' )
0.33333333333333331
```

b) (5 points) [WRITEUP] How would you handle the probability for a word or context you didn't see in the data when creating the `NGramModel`?

3.3 Generating Text (15 points)

a) (10 points) Using each of the language models in turn, generate some text. Add a function `generate(n, context)` to the `NGramModel` class that generates a sentence of length upto `n` words starting with the given context. For the unigram language model, only `None` will be provided as the context. Context will be supplied as a tuple, of length 1 for bigrams and 2 for trigram model. Use the Shannon/Miller/Selfridge method (refer to the lecture notes) to randomly select the new word. It is possible to hit a dead-end while generating. Understanding why this happens is left as an exercise, but if it happens during generation, you can simply stop and return the sentence generated so far.

```
words = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
model1 = NGramModel(words,2)
>>> model1.generate( 5, ('The',))
['The', 'Fulton', 'County', 'Grand', 'Jury']
```

Please note that the sample examples are with a different wordlist. We expect your code to work with any wordlist. We will check on the words in the corpus.

b) (5 points) [WRITEUP] Train an `NGramModel` on the news category of the Brown corpus. Generate three sentences each with unigrams, bigrams, and trigrams. Discuss and explain any issues that arise.