# Linguistics 165, Natural Language Processing Lab 4 Spring 2013 Kelli Wiseth

# Identifying the most informative words in text documents

This lab assignment uses two different approaches to identifying the most informative words in a given document, specifically:

* term-frequency and inverse document frequency (tf-idf)
* log-likelihood ratio (llr)

concepts of document processing is selecting keywords from the document. This exercise and sample code steps-through a Python script that processes a text-file version of a roughly 2,000 word article from the *New Yorker* magazine and identifies the ‘important’ words using two different approaches:

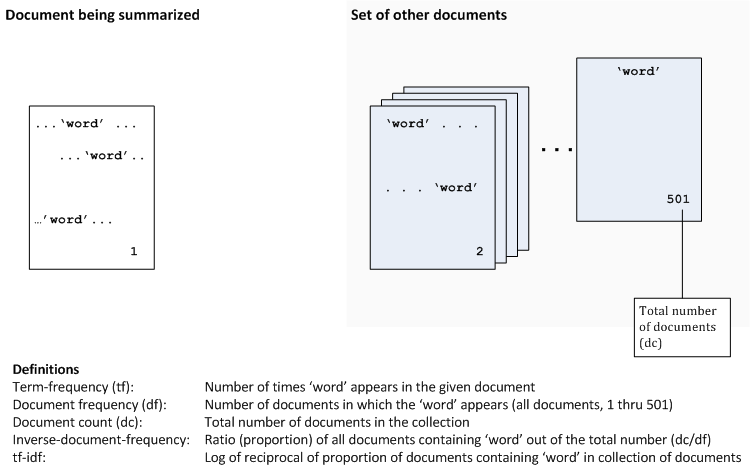
tf-idf

llr

The ‘important words’ are identified

These two different approaches vary in how they quantify ‘informative,’ but generally, the

Log-likelihood Ratio



# Lab components (inputs)

|  |  |
| --- | --- |
| /home/students/ling165/lab4/newyorker.txt | Text-file-version of a New Yorker article. The file comprises 2,021 words. |
| /data/brown | Directory containing 500 Brown Corpus documents whose words are marked-up with POS tags, as in “word/POSlabel”. |
| /home/students/ling165/lab4/clean.py | Script (provided by Professor Hahn Koo) to clean-up character encoding in text files. |
| /home/students/ling165/lab4/llr.py | Script (provided by Professor Hahn Koo) containing functions for log-likelihood calculations. |
| /home/students/kwiseth/lab4/ling165\_lab4.py | My Python implementation of the tf-idf algorithm and processing resources using llr. My script assumes that newyorker.txt has been processed in advance using the clean.py script and its output saved to ‘newyawker.txt’ file, and that ‘newyawker.txt’ is available in the local path at runtime.  Script displays status messages during processing and then displays a two-column report of top-twenty words (per each approach) in descending order, starting from most informative. |

This script assumes that newyorker.txt has been processed in advance using the clean.py script and its output saved to ‘newyawker.txt’ file, and that ‘newyawker.txt’ is available in the local path at runtime.

This project would comprise only a part of a full-scale text retrieval or summarization. It demonstrates two alternative approaches to determining the most “important” words in a document.

Required output from our program is two different lists of top-twenty words, one created using tf-idf approach, the other using the log-likelihood approach.

I’d like my script to perform both tasks in the same script so that I can print out a report at the end of the run. The report would look as follows:

Lab 4 Results

Top-twenty words (newyorker.txt)

tf-idf llr

word1 word1

word2 word2

word3 word3

. . . . . .

word20 word20

[kwiseth@gray lab4]$ python ling165\_lab4.py

To do this, I’ll have to keep track of the rankings per approach.

# Pseudo-code, processing flow for tf-idf

### tf

The first task is to dictionary of words and their counts from the newyorker.txt file. this will provide the tf (term frequency) information.

1. Open the newyorker.txt file
2. Go through the document line by line and build up dictionary of words and counts.
3. When this dictionary is created, we’ll have a complete inventory of words and their counts.
4. Use this dictionary to process all 500 of the documents in /data/brown in the next phase (idf).

sum(tf.values()) Total word count in our tf{dict}. We don’t need this for tf-idf, but will for LLR. This is item (14) total number of words in the given document” from lab instructions.

### idf

for each word in tf dictionary:

Make the list of documents in /data/brown

Open each document

Break on “/” or replace “/” with “<spaceband>” so that the words stand alone??? (if we break on “/,” will we end up with just the words and their POS separated?)

Regular expression: (/.\*) ?? No—easier to just break the words on the slash.

Get the first word in the term-frequency dictionary:

If the tf[word] in doc:

add one to the doc\_ctr for this word [THIS NEEDS TO BE CREATED SOMEWHERE—FOR EACH WORD.   
  
Maybe the tf[dict] has two values as a tuple: { word : (int, int) } ??? So by the end of the first phase, we’ll have a dictionary containing counts of all the words in the newyorker.txt, but the second digit should be 0 (or maybe 1—should we count the newyorker.txt itself at this point, or later, for purposes of calculating idf??

if key in brown\_doc\_words:

doc\_ctr += 1

return doc\_ctr

If we build the dictionary to keep track of both numbers (word count in newyorker, document count for that word in data/brown, at this point we need to use the data to perform the calculation—“the log of the reciprocal of the proportion of documents containing the word in a collection of documents:”

tf = tf[word][0] gives us the word count

idf = 501/[word][1]

tf-idf = tf \* math.log(idf) result; order these from largest to smallest

I have the **tf** dictionary built, but it still has some issues. Specifically, we have some punctuation as words (hyphen, single quote, quotation mark), which would be simple enough to clean up, but we also are breaking-up some words that are hyphenated to begin with (“forty-two,” for example)—these should be retained as words.

Also, word sequences such as “output per hour”—these would be nice to maintain as single units. Similarly, names such as “San Francisco”—we are counting these as instances of “san” and “francisco,” but these should probably be kept together.

brown documents: I have a dictionary containing a document id number (beginning with 0) and just the text of the document as one big string.

In general, this is an issue with the concept of “term frequency” in any summarization or retrieval task (full discussion in *Introduction to Information Retrieval* (Manning, Raghavan, and Schütze: 2008; 21-27).

### Normalizing the New Yorker article text

“Token normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens.” (Manning, Raghavan, Schütze: 26)

### Brown Corpus

Initial runs of my script are identifying the years (2005, 2004, for example) with high tf-idf scores, but then I realize this is likely due to the fact that Brown Corpus is from the 1960s (with an update in the 1970s it seems), so I should probably disregard dates and numbers (“1.5” is registering a high tf-idf). I think I’ll look solely for words as terms, and disregard numeric values.

for key in tf:

if key in brown\_doc:

tf[‘key’][1] += 1

### tf dictionary

For my dictionary, I’d like to keep hyphenated words together as unit (“forty-eight”), so this implies that the words should be collected without first eliminating punctuation—which means that Hahn’s script adds spaces around such words and therefore treats them as single words “forty”, “eight”.

I should try this entire script both with and without the various cleanup routines and see how the results vary.

Some other words should likely also be kept together: for example, “output per hour” is a single concept, and conceivably could be hyphenated. This implies a lookup table or list of some kind, though, since we can’t make decisions about keeping words together without some additional materials. Look at Jurafsky text closer and see what his chapter says.

Some alternatives to consider for the dictionary: Minimize the amount of preliminary cleanup and just delete punctuation and other detritus from the dictionary after it’s built.

Similarly, delete the **stopwords** from the dictionary after it’s built.

TODO: Do I want to try to add the counts from the brown documents to the {tf} dictionary for my idf processing? The dictionary would ultimately look like:

{ word : (int1, int2)} where **int1** is the count of the word in the neworker.txt and **int2** is the number of brown docs in which the word appears.

To the int2 we add a 1 to account for the newyorker.txt count.

### brown data

For the brown data, I’m just breaking on the “/” and then gathering up the first word and disregarding the “POS-tag,” restring together as first a list, then build new string. The end result is on big string containing the entire text of the document. This is associated with an index number in a dictionary. I’m also disregarding punctuation, such as periods, commas, and quotes.

TODO: Check how well my routines handle possessives. Are there any hyphenated words in my resulting brown data??? If not, it means I shouldn’t care if my dictionary {tf} has hyphenated words or not. I should be consistent in how I gather words for {tf} and how I gather words in my brown documents.

### find the words from the dictionary in the documents

Need to go through the dictionary, get each word, and then go through all 500 documents in my brown {dictionary} and keep track of the number of documents in which the word in question appears.

## Log-likelihood Ratio

For LLR, we also build a dictionary. We can use the same general loop as tf, but our structure is different.

fd = {} ‘word’

Mapping the instructions to variables in my code:

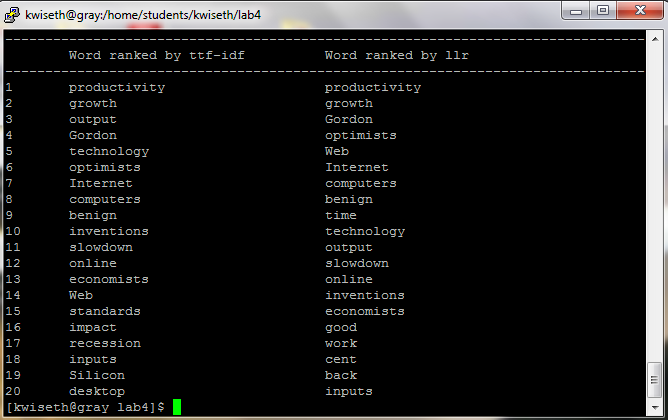
(3) corpus size = 500 documents + 1 (New Yorker article) = 501

|  |  |  |
| --- | --- | --- |
|  |  | Variable name, value |
| (3) we have a corpus size… | 501 | len(brown\_doc\_map) + 1 |
| (4) summarize one of them… | newyorker.txt |  |
| (5) is word x a keyword | tf[key] (key = *word*) | {tf[key] : int } |
| (6) count of word in document | int value |
| (7) number of other documents containing that word |  | brown\_doc\_ctr |
|  |  |  |

tf dictionary { word : int } where *int* is count of word in newyorker.txt

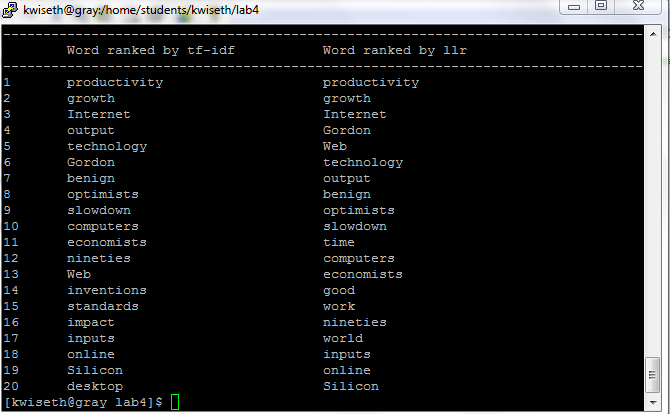
brown\_doc\_map { int : str } where *int* is index number of document from list (0-499) and *str* is text of brown doc as one big string

## Results



The result (Figure 4.1) uses a previous versionloop that does not remove dates. Since the brown corpus is from the 1960s (and I believe updated in the 1970s), it makes sense that dates in the future would have few tokens in the brown corpus (is my thinking correct here?), and thus these dates would emerge as “informative.” However, if we decide that dates should not be counted as “terms” and we remove them, the results are different (see Figure 4.2).

## 28 – August [Results after figuring out my brown-doc loop was WRONG and fixing it ]



Things I concerned myself with unnecessarily:

stopwords, hyphens, dates and numbers—I had initially thought that I’d need to clean these things up, but discovered that it was unnecessary—which, I suppose, is the whole point of these algorithms (tf-idf and llr)! they identify the most informative words based strictly on counts of things…

Initially,

I also tried a version of the script that performed minimal clean-up of the text, but the punctuation and many stopwords surfaced in the top-20 counts, so this final version includes a lot of massaging of the text.

Reasons:

Eliminating dates and years:

Since we’re using the Brown corpus (which was last updated in the 1970s), years such as “2004” emerge in the top-20 list, since this appears in the document but perhaps not at all in the brown document counts.

Since the documents in the /data/brown directory include POS tags, we have to get just the words without the tags. Also, we should leave the capitalization alone, both in the New Yorker article and in the /data/brown files.

These results using the “get\_words()” function.

