COLORADO STATE UNIVERSITY GLOBAL

# CSC435: FUNDAMENTALS OF INFORMATION RETRIEVAL AND WEB SEARCHING

**Credit Hours:** 3

**Contact Hours:** This is a 3-credit course, offered in accelerated format. This means that 16 weeks of material is covered in eight weeks. The exact number of hours per week that you can expect to spend on each course will vary based upon the weekly coursework, as well as your study style and preferences. You should plan to spend 14-20 hours per week in each course reading material, interacting on the discussion boards, writing papers, completing projects, and doing research.

**Faculty Information:** Faculty contact information and office hours can be found on the faculty profile page.

## COURSE DESCRIPTION AND OUTCOMES

### Course Description:

The course introduces the processes and principles of information retrieval and web searching. Students will explore problems in natural language processing that apply to web searching and other information retrieval systems. The course will focus on understanding efficient text indexing, document clustering and classification, and machine-learning based ranking.

### Course Overview:

In this course "information retrieval" deals with methods for searching in (very large) text collections. It introduces algorithms and procedures for text preprocessing, query languages, relevance models, and specific problems with search engine efficiencies. You will also explore information retrieval algorithmic basics as well as concrete applications. Modules 1-6 will be accompanied by a Critical Thinking Assignment based on either Java or Python. There is also a final Portfolio Project due in Module 8 with a project milestone in Weeks 4 and 7 to help you prepare for a polished final submission in Week 8. In addition, each module requires active participation in a discussion forum that explores more depth of the module topic and completion of a mastery quiz. This deepens the learned methods through practical implementation.

### Course Learning Outcomes:

1. Apply Boolean retrieval techniques to an information retrieval problem.
2. Identify data structures to use for efficient information retrieval.
3. Discuss information retrieval indexing and construction.
4. Identify indexing algorithms that can be used for index construction.
5. Implement techniques for index compression.
6. Apply statistical techniques to information retrieval problems.

## Participation and Attendance

Prompt and consistent attendance in your online courses is essential for your success at CSU-Global Campus. Failure to verify your attendance within the first seven days of this course may result in your withdrawal. If for some reason you would like to drop a course, please contact your advisor.

Online classes have deadlines, assignments, and participation requirements just like on-campus classes. Budget your time carefully and keep an open line of communication with your instructor. If you are having technical problems, problems with your assignments, or other problems that are impeding your progress, let your instructor know as soon as possible.

## Course Materials

**Required:**

Manning, C., Raghaven, P., & Schutze, H. (2009, April). *An introduction to information retrieval*. Cambridge, England: Cambridge University Press. ISBN-13: 978-0521865715

*Note: All non-textbook required readings and materials necessary to complete assignments, discussions, and/or supplemental or required exercises are provided within the course itself. Please read through each course module carefully.*

## Course Schedule

### Due Dates

The Academic Week at CSU-Global begins on Monday and ends the following Sunday.

- **Discussion Boards:** The original post must be completed by Thursday at 11:59 p.m. MT and peer responses posted by Sunday at 11:59 p.m. MT. Late posts may not be awarded points.
- **Opening Exercises:** Take the Opening Exercise before reading each week's content to see which areas you will need to focus on. You may take these exercises as many times as you need. The Opening Exercises will not affect your final grade.
- **Mastery Exercises:** Students may access and retake Mastery Exercises through to the last day of class until they achieve the scores they desire.
- **Critical Thinking:** Assignments are due Sunday at 11:59 p.m. MT.

## Weekly Reading and Assignment Details

### Module 1

**Readings**

- Python | Lemmatization with NLTK. (n.d.). Retrieved from https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

- Removing stop words with NLTK in Python. (n.d.). Retrieved from https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

- Tokenize text using NLTK in Python. (n.d.). Retrieved from https://www.geeksforgeeks.org/tokenize-text-using-nltk-python/
- Tokenizing words and sentences with NLTK. (n.d.). Retrieved from ttps://pythonspot.com/category/nltk/

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Critical Thinking (60 points)**

**Option 1: Python Information Retrieval**

Given the following list of plurals (written in Python):

```
>>> plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
...            'died', 'agreed', 'owned', 'humbled', 'sized',
...            'meeting', 'stating', 'siezing', 'itemization',
...            'sensational', 'traditional', 'reference',
'colonizer',
...            'plotted']
```

remove the morphological affixes from each word in the list, leaving only the word stem. Extend your script to prompt the user for a word which your stemming script will process. Submit your Python code in the form of a Python script (.py) file.

**Option 2: Python Information Retrieval**

Using the following Python commands:

```
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer=WordNetLemmatizer()
>>> nltk.download('wordnet')
```

lemmatize each word in the following list:

[‘dogs’, ’geese’, ’cacti’, ’children’, ’feet’, ’corpora’].

Extend your script to prompt the user for a word which your lemmatizing script will process. Submit your Python code in the form of a Python script (.py) file.

## Module 2

**Readings**

- Chapter 1 in *An Introduction to Information Retrieval*
- What does *tf-idf* mean? (n.d.). Retrieved from http://www.tfidf.com/

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Critical Thinking (60 points)**

**Option #1: Querying a Document-Term Matrix**

In text mining, it is important to create the document-term matrix (DTM) of the corpus we are interested in. A DTM is basically a matrix, with documents designated by rows and words by columns, that the elements are the counts or the weights (usually by tf-idf). Subsequent analysis is usually based creatively on DTM.

Exploring with DTM, therefore, becomes an important issue with a good text-mining tool. How do you perform exploratory data analysis on DTM using Python? You will demonstrate it using the data set of the U. S. Presidents' Inaugural Address, preprocessed, which can be downloaded here.

In Python, text mining can be done using the package **shorttext** (https://pypi.org/project/shorttext/).

To complete this CTA, you will need to load the packages "pandas" and "stemming."

1. Load all packages first:

```
import shorttext
import numpy as np
import pandas as pd
from stemming.porter import stem

import re
```

2. Define the processing pipelines:

```
pipeline = [lambda s: re.sub('[^\w\s]', '', s),
        lambda s: re.sub('[\d]', '', s),
        lambda s: s.lower(),
        lambda s: ' '.join(map(stem, shorttext.utils.tokenize(s)))
 ]
txtpreproceesor = shorttext.utils.text_preprocessor(pipeline)
```

The text preprocessor removes all digits and punctuations, makes all letters lowercase, and stems all words using Porter stemmer.

3. Load the dataset:

```
usprezdf = pd.read_csv('inaugural.csv')
```

4. The corpus needs to be preprocessed before putting into the DTM:

```
docids = list(usprezdf['yrprez'])    # defining document IDs
corpus = [txtpreproceesor(speech).split(' ') for speech in usprezdf['speech']]
```

Then create the DTM:

```
dtm = shorttext.utils.DocumentTermMatrix(corpus, docids=docids, tfidf=False)
```

To get the top-most common words in 2009's Obama's speech, enter:

```
dtm.get_doc_tokens('2009-Obama')
```

Or look up which speeches have the word "change":

```
dtm.get_token_occurences(stem('change'))
```

Or to get the document frequency of the word:

```
dtm.get_doc_frequency(stem('change'))
```

For this assignment, create a Python dictionary for the most common word found in each of the President's speeches. Prompt the user for the word and then write your dictionary to a text file. Submit your Python code in the form of a Python script (.py) file.

**Option #2: Querying a Document-Term Matrix**

Using the data set of U. S. Presidents' Inaugural Address, preprocessed, and which can be downloaded here, write a Python script to build a corpus of two presidential speeches. Prompt the user for the two presidents. Compute the TF-IDF score for each word in the corpus and save your results to a text file. Hint: Review the Recommended Readings for guidance on how to implement your Python script as well as the examples shown in option one of this Critical Thinking Assignment. Submit your Python code in the form of a Python script (.py) file.

## Module 3

**Readings**

- Chapter 2 in *An Introduction to Information Retrieval*
- Chapter 3 in *An Introduction to Information Retrieval*

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Critical Thinking (60 points)**

**Option #1: Removing Noise from Text**

For this assignment, you will remove "noise" from a document, which is essential for information retrieval. Examples of noise removal from documents consists of the following operations:

1. Removing text file headers;
2. Removing HTML, XML (for example), markup, and metadata; and,
3. Extracting valuable data from other formats, such as JSON.

For this assignment, you will need to install the following Python libraries:

- NLTK—The Natural Language ToolKit is one of the best-known and most-used NLP libraries in the Python ecosystem, useful for all sorts of tasks from tokenization, to stemming, to part of speech tagging, and beyond.
- BeautifulSoup—BeautifulSoup is a useful library for extracting data from HTML and XML documents.

- Inflect—This is a simple library for accomplishing the natural language related tasks of generating plurals, singular nouns, ordinals, and indefinite articles, and (of most interest to us) converting numbers to words.
- Contractions—Another simple library, solely for expanding contractions.

If you have nltk installed, yet require the download of its any additional data, click here.

For this assignment, use the *BeautifulSoup* Python library and regular expressions to remove open and close double brackets and anything in between from the following text. Save your output to a text file.

```
sample = """<h1>Title Goes Here</h1>
<b>Bolded Text</b>
<i>Italicized Text</i>
<img src="this should all be gone"/>
<a href="this will be gone, too">But this will still be here!</a>
I run. He ran. She is running. Will they stop running?
I talked. She was talking. They talked to them about running. Who ran to the talking runner?
[Some text we don't want to keep is in here]
¡Sebastián, Nicolás, Alejandro and Jéronimo are going to the store tomorrow morning!
something... is! wrong() with.,; this :: sentence.
I can't do this anymore. I didn't know them. Why couldn't you have dinner at the restaurant?
My favorite movie franchises, in order: Indiana Jones; Marvel Cinematic Universe; Star Wars; Back to the Future; Harry Potter.
Don't do it.... Just don't. Billy! I know what you're doing. This is a great little house you've got here.
[This is some other unwanted text]
John: "Well, well, well."
James: "There, there. There, there."
  
There are a lot of reasons not to do this. There are 101 reasons not to do it. 1000000 reasons, actually.
I have to go get 2 tutus from 2 different stores, too.
22    45   1067   445
{{Here is some stuff inside of double curly braces.}}
{Here is more stuff in single curly braces.}
[DELETE]
</body>
</html>"""
```

The following Python methods will help you get started:

```
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()
```

```
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
```

Submit your Python code in the form of a Python script (.py) file.

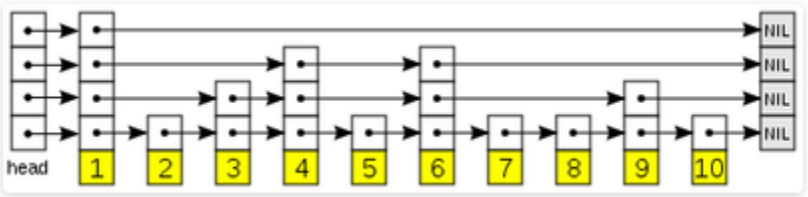
**Option #2: Implementing Skip Lists in Python**

A skip list is a probabilistic data structure that allows efficient search, insertion, and removal operations. Skip lists are much like multiple linked lists with some randomization.

In the first level, we have a regular linked list with the elements sorted.

Each element of this list has a probability p to be also present in the level above.

The second level will probably contain fewer elements and each of these elements will also have a chance p to be on the third level, and so on.




The figure shows an example of a skip list.

In this assignment you will implement a skip list using Python. To start, we define a skip list node. We will represent each level where the node appears by a list of pointers to the next nodes.

```
class SkipNode:
    def __init__(self, height = 0, elem = None):
        self.elem = elem
        self.next = [None]*height
```

Our skip list is just a sentinel skip node with height initially set to 0 and that stores a null element.

```
class SkipList:
    def __init__(self):
        self.head = SkipNode()
```

Now, let us implement the search operation of this list.

Search

---

To search for an element q in a skip list we begin in the topmost level of the header.

We go through the list in this level until we find node with the largest element that is smaller than q.

We then go to the level below and search again for node with the largest element that is smaller than q, but this time we began the search from the node we found in the level above.

When we find such node, we go down again and repeat this process until we reach the bottom level. The node x found in the bottom level will be the largest element that is smaller than q in the whole list and if q is in this list, it will be to the right of x.

Also, we want to keep the nodes found in each level right before going down to the level below since it will make the insertion and deletion operations very simple.

This idea can be translated into the following code:

```
def updateList(self, elem):
    update = [None]*len(self.head.next)
    x = self.head

    for i in reversed(range(len(self.head.next))):
        while x.next[i] != None and \
            x.next[i].elem < elem:
            x = x.next[i]
        update[i] = x

    return update
```

It returns a list of nodes in each level that contains the greatest value that is smaller than elem.

The actual find function returns the node corresponding to the query element or None if it is not present in the skip list.

```
def find(self, elem, update = None):
    if update == None:
        update = self.updateList(elem)
    if len(update) > 0:
        candidate = update[0].next[0]
        if candidate != None and candidate.elem == elem:
            return candidate
    return None
```

Insertion

---

The insertion consists in deciding the height of the new node, using randomHeight()

and for each of the levels up to this height, insert this new node after the node

specified in update.

```
def insert(self, elem):
     node = SkipNode(self.randomHeight(), elem)
```

```
     while len(self.head.next) < len(node.next):
         self.head.next.append(None)

    update = self.updateList(elem)
    if self.find(elem, update) == None:
        for i in range(len(node.next)):
            node.next[i] = update[i].next[i]
            update[i].next[i] = node
```

Deletion

---

The deletion is pretty much like the insertion, but now we delete the node found using find() from all levels in which it appears.

```
def remove(self, elem):
     update = self.updateList(elem)
    x = self.find(elem, update)
    if x != None:
        for i in range(len(x.next)):
            update[i].next[i] = x.next[i]
            if self.head.next[i] == None:
```

An alternative to the skip list is, of course, a linked list.

A simple implementation of a linked list is as follows:

```
class Node:
    def __init__(self, elem):
        self.elem = elem
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = Node(None)
        self.len = 0
    def __len__(self):
        return self.len
    def find(self, elem):
        x = self.head
        while x.next != None and x.next.elem <= elem:
            x = x.next
        return x
    def insert(self, elem):
        x = self.find(elem)
```

```
        if x.elem == elem:
            return
        node = Node(elem)
        node.next = x.next
        x.next = node
        self.len += 1
    def printList(self):
        x = self.head.next
        while x != None:
            print x.elem,
            x = x.next
        print ''
```

For this assignment, you are to implement a skiplist program, as described, and you are to compare the performances of a skip list with a linked list in terms of inserting data.

To compare the performances, you will use an increasing sequence of numbers, a decreasing sequence of numbers, and a random sequence of numbers. Use the following Python methods to generate these sequences:

```
from random import randint, seed
def randomSequence(n):
    seed(0)
    i = 0
    while i < n:
        yield randint(1, n)
        i += 1

def increasingSequence(n):
    i = 0
    while i < n:
        yield i
        i += 1

def decreasingSequence(n):
    i = n-1
    while i >= 0:
        yield i
        i -= 1
```

Create a table of the times to process each sequence type using a skip list and a linked list. Write your table to a text file. Submit your Python code in the form of a Python script (.py) file.

## Module 4

**Readings**

- Chapter 4 in *An Introduction to Information Retrieval*
- Lucene tutorial. (n.d.). Retrieved from https://www.tutorialspoint.com/lucene/

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Critical Thinking (60 points)**

**OPTION #1: Implementing the BSBI Algorithm**

In this assignment you will build an inverted index for a corpus and implement Boolean conjunctive queries. You will implement a blocked sort-based indexing algorithm in Python. The main corpus folder is named "corpus" and each subdirectory is a block. Most of the work is completed for you. Your task is to use the included Python script *index.py* and the packed zip files in the corpus folder to build an inverted index. Once your index is built, you should be able to query the corpus using *query.py.*

Indexing

The index.py script should take two command-line arguments:

- Input data dir: a string-valued argument. This describes the directory location of the input corpus data.
- Output index dir: a string-valued argument. This is the location of the output directory containing the generated index.

You should assume the output directory does not exist yet. The index.py script prints only the total number of files in the corpus to stdout.

Boolean Conjunctive Retrieval

The script query.py will take one command-line argument, which is the directory location of the index you built. The Python script query.py reads from stdin a sequence of word tokens separated by space, which forms the Boolean conjunctive query. There could be any non-zero number of query terms. The program prints to stdout the list of documents containing the query terms, one document file name on each line, sorted in lexicographical order. The document file name displays only the subdirectory name and the file name. For example:

0/crypto.stanford.edu_

0/crypto.stanford.edu_DRM2002_

1/crypto.stanford.edu_cs142_

1/crypto.stanford.edu_cs155_hw_and_proj_

If the conjunctive query has no results (this could also be caused by any of the terms in the query not being found in the corpus), the program outputs no results found to stdout.

The distribution for this assignment includes three Python scripts:

common.py

index.py

query.py

and the corpus zip file.

Your task is to modify the query.py script so that, instead of reading from standard input, it will read the query tokens from a file specified by the user. Each line in the file consists of separate query request. The results of the query processing should be written to an output file.

**OPTION #2: Implementing the BSBI Algorithm**

In this assignment, you will analyze the BSBI implementation within the Python scripts *index.py* and *common.py.* See Option #1 of this CTA for details. The BSBI index construction algorithm is presented in Figure 4.2 of *An Introduction to Information Retrieval.*

```
BSBINDEXCONSTRUCTION()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4     block ← PARSENEXTBLOCK()
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK(block, f_n)
7  MERGEBLOCKS(f_1, ..., f_n; f_merged)
```

Identify how the codes in index.py and common.py implement each line of the algorithm. Rewrite the codes for index.py and common.py so that the code for BSBI-INVERT resides in a separate Python script, the codes for WRITEBLOCKTODISK reside in a separate Python script, and the code for MERGBLOCKs resides in a third Python script. You should produce four Python scripts: a driver for the index construction, an implementation for block parsing (line 4), an implementation for writing blocks to disk, and an implementation for merging block.

**Portfolio Milestone (25 points)**

**Options 1 & 2:**

Make appropriate corrections to the code you submitted in Modules 1-3 (Critical Thinking Assignments). Corrections should reflect feedback from your instructor and improvements in execution, organization, and style. Resubmit your programs from Modules 1-3 with all outlined corrections.

## Module 5

**Readings**

- Chapter 5 in *An Introduction to Information Retrieval*
- Bird, S., Klein, E., & Loper, E. (n.d.) Natural language processing with Python. Retrieved from https://www.nltk.org/book/

- Rahman, A. (2016, October. 3). How to use Python to find the Zipf distribution of a text file. Retrieved from https://code.tutsplus.com/tutorials/how-to-use-python-to-find-the-zipf-distribution-of-a-text-file--cms-26502

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Critical Thinking (80 points)**

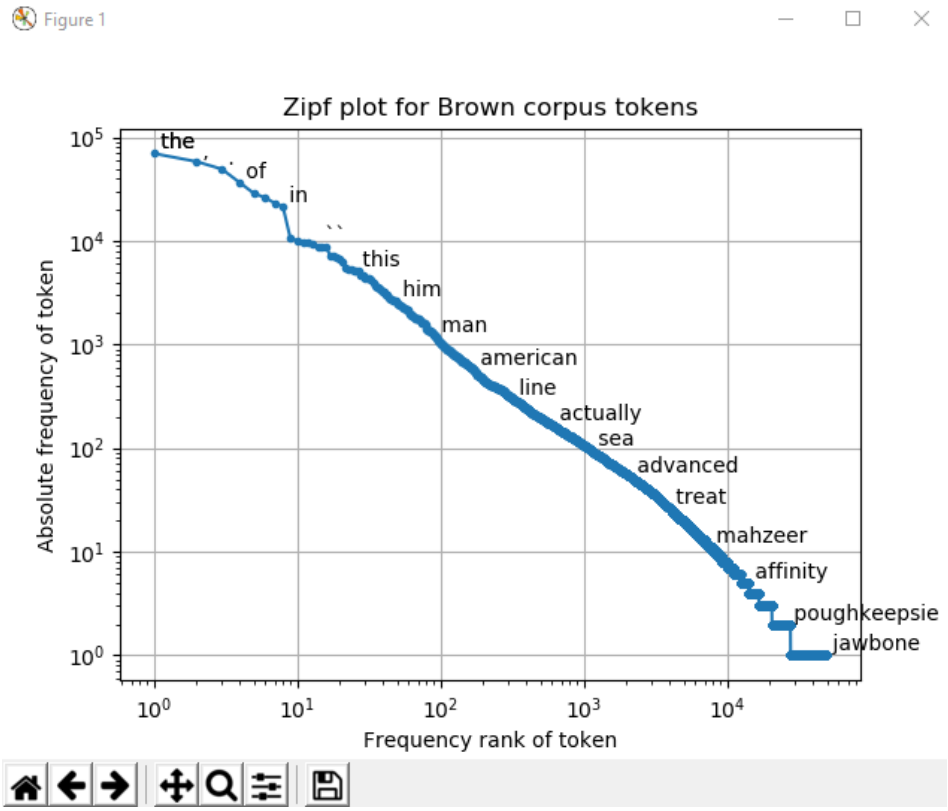
**Option #1: Constructing a Zipf Plot**

The following Python script constructs a Zipf plot of the Brown corpus from the nltk package. Modify this Python script to construct a Zipf plot of the Reuters corpus from the nltk package.

```
from __future__ import division
import matplotlib.pyplot as plt
import nltk
import nltk.tokenize
import collections
import numpy as np

from nltk.tokenize import word_tokenize

from itertools import *
from pylab import *
from nltk.corpus import brown
from string import lower
from collections import Counter

nltk.download('brown')
tokens_with_count = Counter(imap(lower, brown.words()))
counts = array(tokens_with_count.values())
tokens = tokens_with_count.keys()

# A Zipf plot
ranks = arange(1, len(counts)+1)
indices = argsort(-counts)
frequencies = counts[indices]
loglog(ranks, frequencies, marker=".")
title("Zipf plot for Brown corpus tokens")
xlabel("Frequency rank of token")
ylabel("Absolute frequency of token")
grid(True)
```

```
for n in list(logspace(-0.5, log10(len(counts)), 20).astype(int)):
    dummy = text(ranks[n], frequencies[n], " " + tokens[indices[n]],
                 verticalalignment="bottom",
                 horizontalalignment="left")

show()
```

Hint: Read the details found here: https://www.nltk.org/book/ch02.html.




Submit your Python script and a screenshot of your plot.

**Option #2: Verifying Heaps' Law**

In this exercise, you will use two Python scripts and a document corpus to verify Heaps' Law.

This first Python script, *stats.py*, computes the cumulative unique words and total words from the files in the corpus. The second Python script, *index.py*, writes the corpus statistics to an Excel spreadsheet. A zip file containing the corpus is included with this assignment.

```
#stats.py

import glob
import re
import os


dir_name = os.path.dirname(__file__)
```

```
regex = re.compile('\w+')
cache = {}

def text_in_dir_stats(root_dir='texts', maxfiles=9223372036854775807):

        dictionary = {}
        files = 0
        fname = []
        for root, dirnames, filenames in os.walk(root_dir):
                for filename in filenames:
                        fname.append(os.path.join(root,filename))
         for filename in fname:
                files += 1
                if files > maxfiles:
                        break
                if filename in cache:
                        words = cache[filename]
                else:
                        words = regex.findall(open(filename, 'r').read())
                for word in words:
                        if word == "s": # remove 's endings
                                pass
                        dictionary[word] = dictionary.get(word, 0) + 1
        return dictionary

#index.py

from stats import text_in_dir_stats
import xlsxwriter


# Root directory with all texts
root_dir='texts'

# Check Heaps' law
print('Working on Heaps\' law...')
files = 500 # eat 500 files
workbook = xlsxwriter.Workbook('HeapsLaw.xlsx')
worksheet = workbook.add_worksheet()
```

```
worksheet.write(0, 0, 'Texts analyzed')
worksheet.write(0, 1, 'Unique Word Count')
worksheet.write(0, 2, 'Total words in texts')
for max_files in range (1, files):
        dictionary = text_in_dir_stats(root_dir, max_files)
        worksheet.write(max_files, 0, max_files)
        worksheet.write(max_files, 1, len(dictionary))
        worksheet.write(max_files, 2, sum(dictionary.values()))
        print('Progress: ' + str(round(100 * max_files / files)) + '%')
workbook.close()
print('\nDone!')
```

Using the text data from the corpus, plot the number of unique words vs. the number of words in the texts for each count of texts analyzed. Compare this plot with a plot of Heap's Law using the formula: $v(k) = A * k^B$ for A = 20 and B = 0.538. The values of k are the same values of as the number of words in the texts for each count of texts analyzed in your spreadsheet. Submit plots in the form of a Word document and submit your Python code as .py files.

## Module 6

**Readings**

- Chapter 6 in An Introduction to Information Retrieval
- Algorithm to detect similar documents in Python script. (2008, September 19). Retrieved from https://stackoverflow.com/questions/101569/algorithm-to-detect-similar-documents-in-python-script
- Applications and differences for Jaccard similarity and cosine similarity. (2015, February 12). Retrieved from https://datascience.stackexchange.com/questions/5121/applications-and-differences-for-jaccard-similarity-and-cosine-similarity
- Polamuri, S. (2015, April 11). Five most popular similarity measures implementation in Python. Retrieved from http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Critical Thinking (50 points)**

**Option #1: Computing Similarity Measures**

Consider the following the following matrix:

| | k1 | k2 | k3 | E(di,q) | M(di,q) | di.q |
|---|---|---|---|---|---|---|
| d1 | 1 | 0 | 1 | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **d2** | 1 | 0 | 0 | | | |
| **d3** | 0 | 1 | 1 | | | |
| **d4** | 1 | 0 | 0 | | | |
| **d5** | 1 | 1 | 1 | | | |
| **d6** | 1 | 1 | 0 | | | |
| **d7** | 0 | 1 | 0 | | | |
| | | | | | | |
| **q** | 1 | 2 | 3 | | | |

The parameters ($k_1$, $k_2$, and $k_3$) are components of vector listed in the first column. For instance, d1 = (1,0,0). E($d_i$, q) is the Euclidean distance between vectors $d_i$ and q. M($d_i$, q) is the Manhattan distance between vectors d and q: $\sum_{i=1}^{3} |d_i - q_i|$ and $d_i \circ q$ is the inner product of the two vector.

Write a Python script to compute the similarity measures; that is, fill in the table using a Python script. Write your tabular output to either an Excel spreadsheet or a text file. Submit your Python code and output file.

**Option 2: Computing Cosine Similarity**

Write a Python program that prompts the user for two text strings and which computes the cosine similarity between the two strings. Submit your Python source script. Hint: Review the Recommended Readings.

## Module 7

**Readings**

- Chapter 7 in An Introduction to Information Retrieval
- Thakur, N., Singh, P., Dhawan, S., & Agarwal, S. (2015). Implementation of an efficient fuzzy logic based information retrieval system. EAI Endorsed Transactions on Scalable Information Systems 2(5), 1-7. Retrieved from https://arxiv.org/ftp/arxiv/papers/1503/1503.03957.pdf

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Portfolio Milestone (25 points)**

**Options 1 & 2:**

Make appropriate corrections to the code you submitted in Modules 4-6 (Critical Thinking Assignments). Corrections should reflect feedback from your instructor and improvements in execution, organization, and style. Resubmit your programs from Modules 4-6 with all outlined corrections.

## Module 8

**Readings**

- Chapter 8 in An Introduction to Information Retrieval
- Jalali, V., & Matash Borujerdi, M. (2011). Information retrieval with concept-based pseudo-relevance feedback in MEDLINE. *Knowledge and Information Systems, 29*(1), 237-248
- Losada, D.E., Parapar, J., & Barreiro, A. (2017). Multi-armed bandits for adjudicating documents in pooling-based evaluation of information retrieval systems. *Information Processing and Management, 53*(5), 1005-1025.
- Meng, L., Huang, R., & Gu, J. (2013). A review of semantic similarity measures in WordNet. *International Journal of Hybrid Information Technology, 6*(1): 1-12. Retrieved from http://www.cartagena99.com/recursos/alumnos/ejercicios/Article%201.pdf
- Sunny, S., & Angadi, M. (2018). Evaluating the effectiveness of thesauri in digital information retrieval systems. *The Electronic Library, 36*(1), 55-70.
- Marijan, R., & Leskovar, R. (2015). A library's information retrieval system (In)effectiveness: Case study. *Library Hi Tech, 33*(3), 369-386.

**Opening Exercise (0 points)**

**Discussion (25 points)**

**Mastery Exercise (10 points)**

**Portfolio Project (300 points)**

**OPTION #1: Term Weighting with TF-IDF**

TF-IDF (term frequency-inverse document frequency) is a way of determining which terms in a document should be weighted most heavily when trying to understand what the document is about. The term **frequency** reflects how often a given term appears in the document of interest. The **document frequency** is measured with respect to a **corpus** of other documents. It tells you how often the term appears in your corpus overall. The terms that are most informative about a particular text have a high term frequency and a low document frequency.

The TF-IDF for a term is the product of its term frequency and the scaled inverse of its document frequency. **Stopwords** are those words that occur so frequently in the language that they rarely convey information about the meaning of a document.

In this project, you will construct a "Term Weighting with TF-IDF" calculator using Brown's corpus from Python's nltk, or one of your choosing. Select a document from the corpus and calculate the following:

1. Term frequency;
2. Raw document frequency;
3. Inverse document frequency; and,

4. TF-IDF.

Wrap your computations in the form of a graphical user interface that depicts your results in a graphical manner. Note that you can use Python to perform the computations and another programming language, such as Java, for your user interface and outputs. You should be able to call Python scripts from Java. Alternatively, you can use just Python to write out the text with the stopwords redacted and Excel charts render the remaining computations. If you use Excel, write your computational result to a spreadsheet using the Python xlrd package.

The following illustrations show how such an interface may appear:

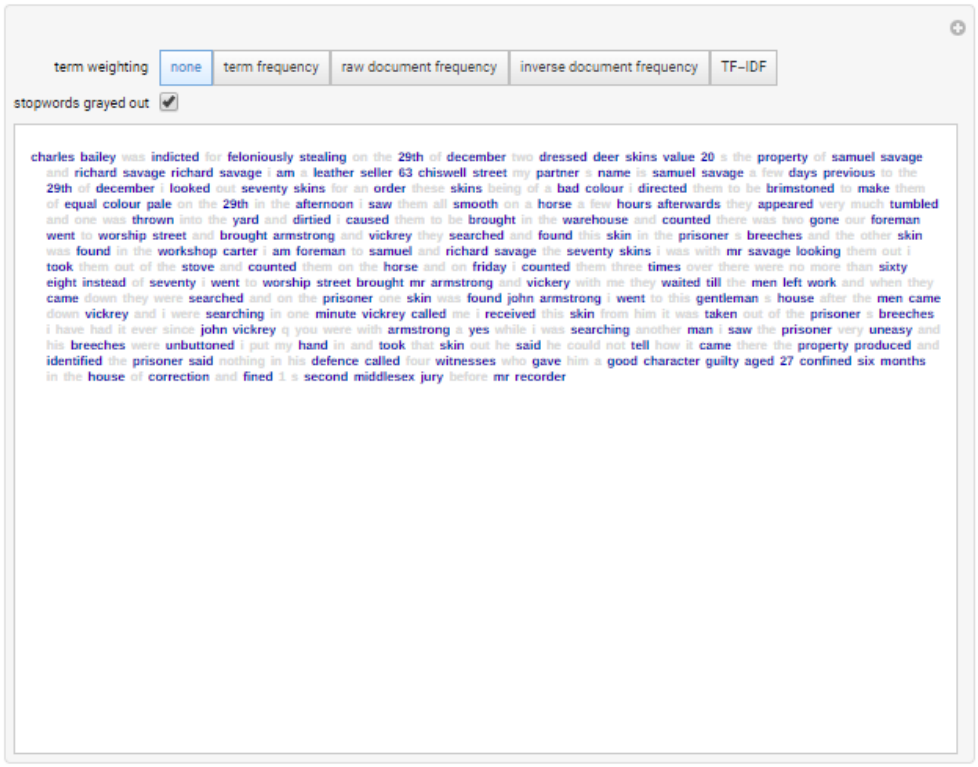



**Figure 1: Panel 1, Stopwords greyed-out**

The first panel displays the original text from the corpus with font colors of the stopwords greyed-out.

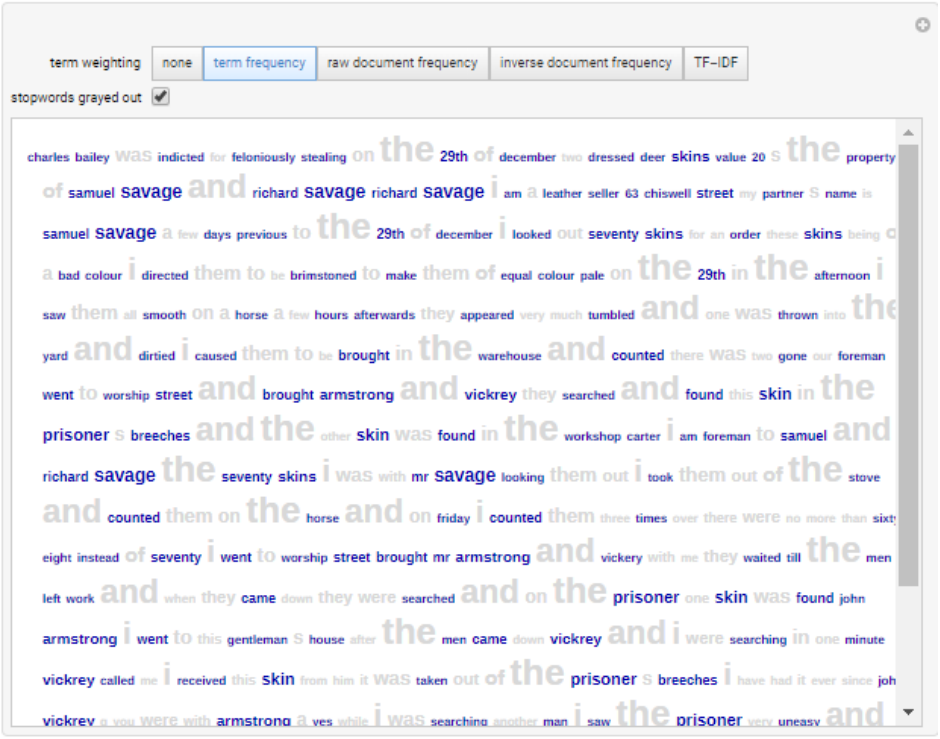



**Figure 2: Panel 2, Frequency counts of words**

The second panel displays the term frequencies from the document in a fashion that the more frequent a term occurs, the larger its font.


**Figure 3: Panel 3, Document frequency of terms**

The third panel displays the terms' raw document frequencies by scaling a term's fonts based on its ubiquity with documents of the corpus.


**Figure 4: Panel 4, Inverse document frequency of terms**

The fourth panel displays the terms' inverse raw document frequencies by scaling a term's fonts based on the inverse of their raw document frequencies that were computed from scores depicted Panel 3.


**Figure 5: Panel 5, TF-IDF of terms**

Finally, the fifth panel displays the terms' tf-idf score by scaling a term's fonts based on that score.

Submit your source files and results in the form of a zip file.

**Option #2: Building an IR Using Lucene**

In this Portfolio Project option, you will implement a user interface for an information retrieval system built using Lucene. You will use the Java programming language and the Eclipse IDE. Work through the installation notes and examples found here:

- https://www.tutorialspoint.com/lucene/lucene_first_application.htm
- http://www.lucenetutorial.com/lucene-in-5-minutes.html.

Use the corpus from Python's nltk Brown corpus.

Write a user interface to your Lucene project that allows the user to enter a different corpus and number of different types of queries. The types of queries your program should process (against the corpus) include the following types:

1. A term query;
2. A term range query that is used when a range of textual terms are to be searched;
3. A prefix query that is used to match documents whose index starts with a specified string;
4. A Boolean query that is used to search documents which are the results of multiple queries using AND, OR, or NOT operators;
5. A phrase query that is used to search documents which contain a sequence of terms;

6. A wildcard query which is used to search documents using wildcards like ‘*’ for any character sequence and ‘?’ matching a single character;
7. A fuzzy query that is used to search documents using fuzzy implementation that is an approximate search based on the edit distance algorithm; and,
8. A match all docs query that matches all the documents.

The user interface will present the user with a textbox in which they enter a query against a corpus. The user will then select a query type from a menu and the output of the query results will be written to a textbox on the user interface.

## COURSE POLICIES

| Grading Scale | |
|---|---|
| A | 95.0 – 100 |
| A- | 90.0 – 94.9 |
| B+ | 86.7 – 89.9 |
| B | 83.3 – 86.6 |
| B- | 80.0 – 83.2 |
| C+ | 75.0 – 79.9 |
| C | 70.0 – 74.9 |
| D | 60.0 – 69.9 |
| F | 59.9 or below |

**Course Grading**

20% Discussion Participation
0% Live Classroom
8% Mastery Exercises
37% Critical Thinking Assignments
35% Final Portfolio Project

## IN-CLASSROOM POLICIES

For information on late work and incomplete grade policies, please refer to our **In-Classroom Student Policies and Guidelines** or the Academic Catalog for comprehensive documentation of CSU-Global institutional policies.

**Academic Integrity**
Students must assume responsibility for maintaining honesty in all work submitted for credit and in any other work designated by the instructor of the course. Academic dishonesty includes cheating, fabrication, facilitating academic dishonesty, plagiarism, reusing / repurposing your own work (see the CSU-Global Guide to Writing & APA for percentage of repurposed work that can be used in an assignment), unauthorized possession of academic materials, and unauthorized collaboration. The CSU-Global Library provides information on how students can avoid plagiarism by understanding what it is and how to use the library and internet resources.

**Citing Sources with APA Style**
All students are expected to follow the CSU-Global Guide to Writing & APA when citing in APA (based on the most recent APA style manual) for all assignments. A link to this guide should also be provided within most assignment descriptions in your course.

**Disability Services Statement**
CSU-Global is committed to providing reasonable accommodations for all persons with disabilities. Any student with a documented disability requesting academic accommodations should contact the Disability Resource Coordinator at 720-279-0650 and/or email ada@CSUGlobal.edu for additional information to coordinate reasonable accommodations.

**Netiquette**
Respect the diversity of opinions among the instructor and classmates and engage with them in a courteous, respectful, and professional manner. All posts and classroom communication must be conducted in accordance with the student code of conduct. Think before you push the Send button. Did you say just what you meant? How will the person on the other end read the words?

Maintain an environment free of harassment, stalking, threats, abuse, insults, or humiliation toward the instructor and classmates. This includes, but is not limited to, demeaning written or oral comments of an ethnic, religious, age, disability, sexist (or sexual orientation), or racist nature; and the unwanted sexual advances or intimidations by email, or on discussion boards and other postings within or connected to the online classroom. If you have concerns about something that has been said, please let your instructor know.