**Indexing and Retrieval**

**Small Search Engine: Setup, Compilation, Execution and Output Guidelines**

This is an implementation code designed in python 2.7 for small search engine

For this code to be executed on any terminal, we need to have below pre-requisites met:

1. Python 2.7 installed

Go to https://www.python.org/downloads/ -> install as per operating system

Required Python Packages

from math import log  
 import operator  
 import sys  
 import re

from collections import defaultdict

from itertools import izip

from itertools import groupby

This search engine implementation is done in two phases, one for inverted index creation and one for BM25 ranking algorithm.

* Phase one : Simple Inverted Indexer

Requirement:

File name: indexer.py

No. of Arguments: Two

1. tccorpus.txt (Input)

This file in the input argument containing already stemmed document collection

1. index.out (Output)

This is the output file generated which stores the inverted index in the form of,

Term frequencies (tf) are stored in inverted lists: word -> (docid, tf), (docid,tf), ...

Steps to be done:

Place the given python file i. e. “indexer.py” and given corpus text file “tccorpus.txt” to any particular location (File path)

Compilation and Run Python code:

Go to terminal

Type python. This will start the python shell terminal

Navigate to the required path and run .py file as below->

->FilePath/ indexer.py tccorpus.txt index.out

Expected Output:

File Output:

File index.out is created storing inverted indexes for all words in the given document corpus (Excludes token that contains only digit 0-9)

* Phase two: (BM25 Implementation for Ranking)

Requirement:

File name: bm25.py

No. of Arguments: Three

1. index.out (Input)

This is the inverted index for the given corpus. (Output file for indexer.py)

1. queries.txt (Input)

This is the input argument containing stemmed test queries

1. 100 (Input)

Input argument that indicates the maximum number of document results for each query

Steps to be done:

Place the given python file i. e. “bm25.py”, “index.out”, “queries.txt” to a particular file path

Compilation and Run Python code:

Go to terminal

Type python. This will start the python shell terminal

Navigate to the required path and run .py file as below->

->FilePath/bm25.py index.out queries.txt 100

bm25.py index.out queries.txt 100 > results.eval

Expected Output:

Based on the BM25 implementation, an output file “results.eval” is generated as the file path (Same result is displayed on screen)

Run name: SARITA\_RUN (SYSTEM\_NAME)

Output Format:

query\_id Q0 doc\_id rank BM25\_score system\_name

That is for each query with query-id ranging from 1-7, it displays the top 100 documents along with document id and ranking score (Sorted in decreasing score)

References:

1) <http://www.tutorialspoint.com/>

2) <http://stackoverflow.com/>

3) <http://pythonguru.com> for examples on reading writing dictionary

#############################################################

Implementation

##############################################################

This small search engine is implemented in python in two steps based on the received corpus (with total stemmed documents: 3204)

1. Building an Inverted Index:

* No of inputs required : Corpus
* No of outputs : A file storing term frequencies in the form word->(docid,tf) for all words in the corpus excluding the tokens containing only (0-9)
* The file is retrieved and then split based on the ‘#’ tag found in stemmed documents, as the document following it, is a document id
* Use of python functions like : lambda, filter and group by are used to achieve the same.
* These document based on doc-id and document content is stored in a dictionary.
* Then for each key in the formed dictionary , we iterate for all words in the document and adds the frequency of each word i.e

frequencies[each] = 1 *# First occurrence*

**else**:  
frequencies[each] += 1 *# Repeated*

* We use an intermediate list word\_docid\_freq for storing the doc along with calculated frequency.
* Inverted Index, index is a dictionary storing the inverted list for all terms in the document. (Index length : 10873)

1. BM25 Ranking Algorithm : (Ranking algorithm based on binary independence model)

* No. of inputs : 1 (index.out -> inverted index) 2 (queries.txt -> stores stemmed queries) 3 (max number of document results)
* No. of output : top 100 documents for each given query based on the ranking generated using BM25 algorithm (results.eval)
* The input inverted index is retrieved via split and stored in dictionary indexes, Query results as required in the assignment is stored in results list

docids\_length is the dictionary storing document id along with its calculated length

K = calculate\_K\_value (dl,avdl) -> Smoothing factor calculation

avdl -> average document length -> 68.8667290886 (length of all documents/total number of document)

* BM25 is given as

BM25(Q,D) =

term1 = log(((r + 0.5) / (R - r + 0.5)) / ((n - r + 0.5) / (N - n - R + r + 0.5))  
term2 = ((k1 + 1) \* f) / (K + f)  
term3 = ((k2 + 1) \* qf) / (k2 + qf)  
**return** term1 \* term2 \* term3

where, k1=1.2, b= 0.75, k2= 100,(R and r are zero as no relevance information provided)

*n = no if docs containing the term  
 f = frequency of a term in a particular document  
 qf = query frequency of the term  
 r = relevant documents containing the term  
 R = relevance information  
 dl = length of the document  
 avdl = average length of the documents  
 N = Total no of documents in the collection*

* Thus, for each query in query\_lists BM25 score for documents is calculated for all the terms in the query.
* The resultant score for each document is the summation of all the BM25 scores calculated for each individual term of the query.
* For each query, the top 100 documents depending on the BM25 score is displayed and printed out on the console.

----------------------------------------------END---------------------------------------------------------