# IRWS 2019

# Assignment 1

**Introduction:**

Solve the problems detailed below.

You should submit an archive (zip/rar/gzip/tarball/7zip) containing all your code together with a README.txt . **DO NOT submit a project folder from an IDE such as eclipse!**

Your README.txt should describe how to compile and execute each of your programs.

All of your programs should be console applications, **WITHOUT** any user interface. Make sure that your code compiles on any machine (if you use a language or languages that need to compiled).

Your applications will be tested on a test system with the following specification:

Ubuntu Linux

2 gb ram

NO internet connection

NO windows manager or GUI, command line only (BASH)

**Rules for language choice**:

You may use any language that can be executed on a Linux environment and does not require an IDE to be installed. If any installation is required, you MUST provide instruction and links to the necessary resources.

Your applications or scripts must not use fixed paths.

If I have to edit it to get it to work, then your script/application is incomplete.

note: you cannot use R. (sorry, its too easy **http://www.dataperspective.info/2017/11/information-retrieval-document-search-using-vector-space-model-in-r.html** )

**Task 1**: (20 marks)

Write a script or a program that reads a text file, pre-processes it and saves

the results into a new file. The text file contains documents, one document per line.

Each document is one or several sentences.

Your program should take three parameters: input file name, output file name, stopword list

It should pre-process documents so that they can be later used to create an inverted index. Basic pre-processing should consider:

- punctuation
- tokenization
- lower-casing/upper-casing / punctuation / numbers
- stop word removal (sample list will be provided, comma separated list of words)
- stemming must use one of the Porter stemmer libraries you can find here: https://tartarus.org/martin/PorterStemmer/index.html the library must be used or you can write your own as long as it gets the same result. the page also has a link to a sample vocab and output to test your algorithms. (I'll put it up on moodle)

Your program should write the pre-processed documents into the output file.

**sample input file**:

D1 This is a sample document

D2 This is a second document. This one has more sentences.

D3 The name of the document is spaced by a TAB character! all docs are on a single line

**sample output file**

D1 sample document

D2 second document more sentence

D3 name document space tab character doc single line

**Task 2:** (20 points)

Write a script or a program that reads a text file of documents (you can

assume these are pre-processed already), creates an inverted index and saves to file.

Your program should take two parameters: **input file name and output file name**.

Input file is made of documents, Each document will have an Identifier/Title separated from the content by a TAB character.

Each line of the output file should contain the term and title of documents that the term occurs in. Each column must be separated by a TAB character.

sample RUN: script2 <infile> <outfile>

**sample input file:**

D1 sample document

D2 second sample document

**sample output file:**

sample D1 D2

document D1 D2

second D2

The output file must be created in the same location as the input file.

**Task 3: (20 marks)**

Write a script or a program that reads a text file containing an inverted index, creates the TF-IDF weights matrix and saves it in a file.

Your program should take two parameters: input file name out file name.

Input file can be assumed is a representation of an inverted index where each line contains the term and identifiers of documents where that term occurs in.

Your output file should contain the weights matrix, document identifiers as the header of each column and terms as the first column.

sample in file: (columns will be TAB delimited)

| | | |
|---|---|---|
| sample | D1 | D2 |
| document | D1 | D2 |
| second | D2 | |

sample out file (columns must be TAB delimited)

| | | |
|---|---|---|
| | D1 | D2 |
| sample | 0 | 0 |
| document | 0 | 0 |
| second | 0 | 0.301 |

you MUST round to three decimal places as soon as the calculation is made

**Task 4 (20 marks)**

Write a script or a program that reads a text file containing a TF-IDF weights matrix

defined in the previous task, and two additional parameters that are documents' identifiers. Your program should return the cosine similarity value of those two documents.

sample command to run: script weightfile D1 D2

sample output: 0.6

**Task 5 (20 marks)**

Write a script or program that launches from the command line and takes in two arguments.

Argument 1 is an input file that contains documents, 1 per line with an identifier separated by a TAB character. Each document may be one or more sentences.

Argument 2 is a query that contains proximity instructions

sample launch: script corpus.txt "{sample /2 document} AND second"

The script must output a list of documents in the corpus that matches the query. The query language works as follows:

clauses are separated by brackets **{** and **}** phrases inside the brackets should be resolved first

After brackets , starting from left to right, AND clauses should be resolved

Next OR clauses should be resolved

Last BUT NOT / AND NOT clauses should be resolved

term1 /# term 2 indicates term1 WITHIN # words of term 2 , proximity clauses will ONLY be presented within { }

sample query:

{A /2 B} AND C BUT NOT {D OR E}

this is resolved as

**Z** = {**A** within 2 words of **B**} **Y** = {**D** OR **E**}

{**Z** AND **C**} AND NOT **Y**

**Q** = {**Z** AND **C**}

**Q** NOT **Y**