IRWS

Assignment (40% of the module)

Introduction

- This is a group project. **TWO** members per group.
- There are two tasks to be achieved in this assignment. Solve them both.
- You should submit an archive (zip) 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 (two separate files one for each of the two tasks).
- 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 windows system, with NO internet connection, only using command prompt.
- Your applications or scripts must not use fixed paths i.e. it should run from my system without any changes.
- If I have to edit it to get it to work, then your script/application is incomplete.
- Accepted languages are python and node.js. You cannot use R
- You should submit a **video presentation** for each of the tasks, both members should present and demonstrate how the systems works.

Task 1 (50 %)

In this task you are required to write a series of scripts to be able to return the proximity of a query, the following milestones should be achieved in order:

Preprocessing:

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 and 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.

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

Inverted index

Write a script or a program that reads a text file of documents (you can assume these are pre-processed already in previous milestone), 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. The output file **must** be created in the same location as the input file.
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

TF-IDF

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 to be a representation of an inverted index where each line contains the term and identifiers of documents where that term occurs in (milestone 2).
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

Cosine similarity

Write a script or a program that reads a text file containing a TF-IDF weights matrix defined in milestone 3, 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

IR System

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

sample launch: script corpus.txt “Sample second”

Your output should be the ordered list of documents that matches the query with the score, example output would be
D2 0.9
D1 0.4

Task 2 (50 %)

You are required to write an application/program or script that will generate a fused results set from a set of historic results using the Probfuse fusion method.

a. Ask the user for a set of historical results in csv format where each result set Starts with the IR engine Letter and the Query Number. So A2, means this is from IR engine A and is the second query . see below.
b. Ask the user for input : how many segments to use
c. Ask the user for the name of the live results file. In CSV format with each IR engine identified as the first entry (see below)
d. Ask the user to state what results they want fused out of A, B, C or D
e. Output the fused set of results to a file (user should not be asked for an output file name)

Historical data file:

- Each line is a new result set
- Each line starts with the name of the IR engine (A, B , C or D) and the number of the query (1, 2, 3) and is separated from the result set by a comma (,)

- Each result set contains R for a relevant result and N for a non-relevant result. There are no unknown results, this is a FULLY JUDGED CORPUS.
- Each result is separated from the next by a comma (,)
- Each line ends in a CR and a LF character to indicate end of line

Live Results file:

- The live results file is a CSV file
- Each line contains one set of live results
- Each line starts with the IR Engine name (A. B , C or D) , the name is separated for the result set by a comma (,)
- The result set consists of a set of document numbers (integer values only) separated by a comma (,)
- Each line ends with a CR and a LF character to indicate the end of a line

General Notes

Grading

Your assignment will be marked on:

- Quality of the readme file
- Quality of the program in completing the task (your document showing the operation and your application source code)
- Your application code (*must* be clearly commented so a reader can follow the function of the code)
- Following your readme the user can run your code on a windows machine on a test data set that is not the same as the sample data on moodle
- Ease of use, options and efficient execution
  (options could be to run another fusion exercise using the same files but with a different number of segments or with a different combination of IR engines, a method of ensuring subsequent runs do not overwrite the output of previous runs, etc. anything you want you are a masters student, use your imagination. However, marking of this is based on the user opinion o f how useful it is)

Readme file:

your readme file should contain

- Name of your script/code/program
- List of system requirements
- Language and version that it is written in
- List of files in the application or script
- Instructions to run the application including any installation or compilation instructions (note: these will be tested with NO IDE so your code MUST be executable without an IDE installed)
- Expected input required and format of that input
- Expected output and format of that output

Operation:

- All input and output files should be assumed to be in the same directory as your application, program or script.
- All output should be to a file(s) in the same directory as your program or script.
- You are NOT expected to create a GUI. Command line input is preferred either during the application run or as starting parameters.

Application Code:

- you must submit your application code in a format that can be read.
- Your code must be commented with useful comments – eg

```
//this prints the result to the screen
```

- This is NOT a useful comment. It states exactly what the application is doing

```
//this function takes the file name as input and assigns it to a variable which is used to open the file and is used again later in the calculation function to read in the options selected by the user. The filename does not require an extension to be added as this is handled by the library (x) imported above. If there is more than one file with the same name, this function will select the first one listed in the directory.//
```

- This is a useful comment as it explains how the block of code it relates to functions within the overall application and informs the reader about how it works and what the exceptions or possible problems could be when it is used.
- You do NOT need to comment every line. You only need to comment a block that is important to how your application/script runs.