COP290

Plagiarism checker

Vishal Bindal
2018CS50425

To Run

1. Run ‘ `make` ’. It will create obj files and an executable ‘plagChecker’
2. Run `./plagChecker <LOCATION_OF_TEST_FILE> <LOCATION_OF_CORPUS_FOLDER>`
   `e.g. ./plagChecker corpus_files corpus_files/catchmeifyoucan.txt`

If there are M documents in the corpus folder, the output will consist of M lines with each line of the form:
`<DOC_NAME> <SIMILARITY_%>`

Project structure

main directory
|---------src
        |----------main.c
        |----------handle_input.c
        |----------similarity.c
|------headers
        |---------handle_input.h
        |---------similarity.h
|-----Makefile

handle_input.c contains functions to read all files in the corpus folder, and to get all words in a given file in a char** array.

similarity.c contains functions to implement the logic of computing similarity through the metric of cosine similarity
main.c uses functions from handle_input.c, similarity.c through the headers handle_input.h and similarity.h

Metric and algorithm used

I've implemented **cosine similarity,** and **TF-IDF vectorisation** for the assignment.

Source referred for the idea: (slides by Intel on text similarity)
https://www.slideshare.net/ankit_ppt/text-similarity-measures

Algorithm:

(Here n = no of total words across all files (corpus folder + target file),
M = no of documents)

1. **Obtain a list of all unique words in all the documents, the 'vocabulary' of the corpus folder+target file**
   char** vocabulary = <All unique words in corpus folder and target file>
   This is done by
   a. Making a char** array by appending list of words in all the files, in O(n) time
   b. Sorting the above array in O(nlogn) time
   c. Removing duplicates from the array, in O(n) time
   Let v be the length of vocabulary. v<=n

2. Create a matrix of size (M*v). `matrix[i][j]` will denote the **word count of word `vocabulary[j]` in document i**. `0<=i<M, 0<=j<v`

3. Apply **TF-IDF (Term frequency-inverse document frequency)** to the above matrix
   a. Normalise each element of matrix by total word count in that document. i.e.
   `matrix[i][j] = matrix[i][j]/ word_count[i]`
   where `word_count[i]` is no of words in document i
   b. Multiply each element of matrix by a term `log((1+M)/(1+k))` where k is the number of documents in which that specific word arrives

4. **Calculate cos(theta) between the vector of target document, with the vector of each document** in the corpus folder one by one. Multiply by 100 to get the percentage.

Metric:

Represent documents in form of vectors
a[0…..v-1] and b[0….v-1]
where v = no of unique words in vocabulary

```
a[i] = ((word count of word vocabulary[i] in the document) / (no of total words in document)) * (log((total no of documents + 1) / (no of documents word vocabulary[i] appears in + 1)))
```

The **normalisation by total no of words** is done so to **ensure similarity % is not affected by size of document**.
The log term is multiplied so to **increase the effect of 'rare' words, than common words** like 'the' which are found in most files. Sharing 'rare' words implies higher similarity than sharing common words.
The (+1) in log term is to ensure division by a non-zero number
'Log' is done of the quantity to ensure the effect of enhancing effectiveness of rare words does not increase 'too much'

Then the similarity is given by
`100 * (dot product of a and b) / ((norm of a) * (norm of b))`

This is just the cos(angle between 2 vectors) multiplied by 100. **Smaller angle means vector of both documents is close by, implying documents are similar.** Cos value is large for small angle, so similarity value will be high for small angles and similar documents.

Complexity:

n = no of words across all corpus documents + target document
M = no of documents

Time complexity : `O(n*log(n))`
Bottleneck step is using quick sort on the vocabulary array (O (nlog(n))), and using binary search to create the word count matrix (n words * O(log(n)) = O(nlog(n)))

Space complexity : `O(n*M)`
The matrix has size M*v, where v is no of unique words. Since v <= n, size of matrix = O(M*n)

Output for sample test case

ecu201.txt 39.47%
sra42.txt 3.32%
esv254.txt 3.86%
sra107.txt 1.58%
edo20.txt 4.20%
prz100.txt 2.89%
hal10.txt 41.88%
erk185.txt 2.77%
hte42.txt 1.34%
edo14.txt 1.13%
ckh80.txt 2.63%
edo26.txt 2.77%
bgt221.txt 2.06%
bef1121.txt 2.77%
bmu5.txt 2.56%
jrf1109.txt 1.71%
sra31.txt 1.69%
sra126.txt 2.13%
ehc229.txt 2.79%
tyc12.txt 58.10%
bwa248.txt 3.83%
sra119.txt 2.70%
catchmeifyoucan.txt 100.00%
abf70402.txt 3.07%
abf0704.txt 4.40%

As expected, % match with ecu201.txt, hal10.txt and tyc12.txt is large, will it is <5% for others.