Plagiarism Detector

March 23, 2021

Overview

We have to make a plagiarism detector through which we could detect plagiarism of a file with some files in the corpse folder. We needed to define our own measure of similarity. We also needed time and space complexity for our approach.

Goals

1. **Define measure of similarity:** I have first weighed each of the terms appearing in the file using the TF-IDF method. After weighing I used a cosine vector product to determine similarity.
2. **Implement and do Complexity analysis:** Implemented a hashmap for storing and answering queries in O(N) time complexity.

Specifications

TF-IDF is short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use tf–idf.

Apart from that I have also implemented a hashmap so as to find and update the counting of terms present in the files in constant time. Hashmap is made using the standard djb2 hash function and used hashing with chaining to map the already existing elements.

Milestones

1. Implementation

   Implementation includes TF-IDF and cosine similarity product for evaluating the plagiarism percentage between any two documents.

   How I computed TF-IDF:

   TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more often in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

   TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

   IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

   IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

2. Complexity Calculations

Time Complexity:

I have used a hashmap for storing, retrieving terms and their count(number of occurrences). I have used hashing with chaining for the hashmap. I have used linked lists for chaining. Now Time Complexity.

Storing a term:- O(1+ a)

Retrieving a term:- O(1+a)

Retrieving their coun:- O(1+a)

Hence each of the updates required amortized complexity for each term is constant time. Hence overall it's constant time operation for each of the operations. So, the overall time complexity for the program should be amortized O(N).

Space Complexity:

It is O(Size of Table*(initial length of linked list assigned for chaining)). The initial size of table assigned is O(10000*500). So, It is constant with the assumption that no more than 500 words would be mapped to the same hashing index.

Instructions to use:

1. For changing the number of files in corpus files change the NUMBER_OF_FILES definition to ((number of files in corpus folder) + 1).
2. Currently it is set to 26. So, 25 documents should be in the corpus folder.
3. Input and output formats should be as specified. Add / at the end of the folder parameter.
4. For the current state of the folder following command on the terminal should run the plagiarism detector.

   Command on terminal: ./plagChecker catchmeifyoucan.txt corpus_files/

5. The above command should compare catchmeifyoucan.txt to all files in corpus folder.