COP290 – Design Practices
Assignment : 7
Plagiarism Checker

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

AIM :

In this assignment, given a set of text documents , we need to determine whether or not, or the degree to which, any pair of documents contains heavily copied portions of text. If one document shares an inordinate number of phrases in common with another, it's likely the document is plagiarized.

Approach For Text Similarity Detection :

I used cosine similarity to determine the similarity measurements of a document with corpus files. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size.
Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.
each of the documents are converted to n-dimensional vector, the query is also converted to n-dimensional vector, these vectors are then normalized, and the TF*IDF is calculated for each of them, the advantage of using TF*IDF is that common words across the documents are weighted down, i.e. they are assigned a lower score, and the unique words are weighted more, using logarithmic, now that we have a vector for each of the document and also the query, it can simply be compared using Dot Product, which is

$$Cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\,\|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2}\,\sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a}\cdot\vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

Here **a** and **b** are term frequency vectors.

Inverse Document Frequency : The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Here:

* N : total number of documents in the corpus N = |D|
* denominator is the number of documents where the term *t* appears (i.e. tf(t,d) != 0). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator increasing by 1.

Implementation Approach :

self implemented hash-map has been used to store term frequency of each word. Let the document of which we need to check the plagiarism is *input file* and total number of files in corpus folder are *k*.

1. Read all files (including input document, i.e. k+1) and sanitize their words (decapitalizing them, removing commas, brackets, semi colons etc.) and store them in a map with their term frequency.

2. Normalize the hashmaps of all files(*k*+1) by dividing all term frequencies by (*total no. of words in that file*).

3. Calculate *idf* (inverse document frequency) of each word for each hashmap by the above given formula. Inverse term frequency can be negative.

4. Do multiplication of term frequency of each word of each hashmap with its respective *inverse document frequency (idf).*

5. Now we have (*k*+1) vectors for each file. Take hashmap of *input file* and find its cosine with corpus files one by one using the above formula. The shorter the value of the angle, the more value of its cosine and therefore, more plagiarism.

6. cosθ will output a value from [-1,1], but for our implementation it won't output a value less than 0, hence the output of Cosine Similarity will be from [0,1], this can be mapped from [0,100] to get a percentage of potential plagiarisation.

Now we can assume that the threshold similarity upto 50%. i.e. above 50% is highly plagiarised.

Map Implementation :

struct map :

a struct that contains three arrays.

1. string array *words*: it stores the words of the file on the respective index.
2 float array *freq*: it stores the term frequency and later tf*idf of the corresponding word of respective word.
3. int array *indics*: it stores all the indices that are non-empty.

Time complexity :

1. Add an element in map : avg : O(1) and worst case : O(n)
2. get the value of a key : avg : O(1) and worst case : O(n)
3. bool contains : avg : O(1) and worst case : O(n)
4. normalization : O(n)
5. update_tfidf : O(k*n)
6. readFile : O(total characters in file)

OVERALL TIME COMPLEXITY :

$$O(s+k * n)$$

Where

s : default hashmap capacity
k : total number of files in corpus folder
n : total number of words in all files

Submitted By :
Manoj Kumar
2018CS50411