CS F469 Information Retrieval Project - Design Document

Group Members:

ID Number	Name
2017A7PS0075H	Prateek Agarwal
2017A7PS0093H	Shreeya Bharat Nelekar
2017AAPS0409H	Shriya Choudhary
2017AAPS1741H	Shubham Singhal

Plagiarism Detector using LSH Algorithm

Introduction:

We designed a plagiarism detector using LSH(Locality Sensitive Hashing) algorithm. The plagiarism of a given document is tested against a set of documents already specified by us. We specify the path to this set of documents in our code. So we can simply change the path to point to a different dataset for checking plagiarism against another set of documents. Also, we can change the similarity threshold to find if a document is plagiarised or not as needed.

Language used : Python3

Dataset For Plagiarism Detector:

The corpus contains 100 documents (95 answers provided by the 19 participants and the five Wikipedia source articles). For each learning task, there are 19 examples of each of the heavy revision, light revision and near copy levels and 38 non-plagiarised examples written independently from the Wikipedia source (there is an uneven spread in the number of answers across tasks and categories results from using a Latin-square arrangement to order the tasks carried out by the 19 participants). The answer texts contain 19,559 words in total (2,2230 unique tokens) and the Wikipedia pages total 14,242 words after conversion to plaintext using lynx -dump and removal of URL references. The average length of file in the corpus is 208 words (std dev. 64.91) and 113 unique tokens (std dev. 30.11). Overall, 59 (62%) of the files are written by native English speakers; the remaining 36 (38%) by non-native speakers.

LSH Algorithm:

The LocalitySensitiveHashing module is an implementation of the Locality Sensitive Hashing (LSH) algorithm for nearest neighbor search. The main idea in LSH is to avoid having to compare every pair of data samples in a large dataset in order to find the nearest similar neighbors for the different data samples. With LSH, one can expect a data sample and its closest similar neighbors to be hashed into the same bucket with a high probability. By treating the data samples placed in the same bucket as candidates for similarity checking, we significantly reduce the computational burden.

There are three main steps in lsh:

1. **Shingling** : Convert each document into a set of characters of length k. Represent each document as a set of k-shingles.
2. **Minhashing** : Step 1: Random permutation (π) of row index of document shingle matrix.
   Step 2: Hash function is the index of the first (in the permuted order) row in which column C has value 1. Do this several times (use different permutations) to create signature of a column.
   Property - The similarity of the signatures is the fraction of the min-hash functions (rows) in which they agree.
   Expected similarity of two signatures is equal to the Jaccard similarity of the columns.
   The longer the signatures, the lower the error.
3. **Locality Sensitive Hashing** : Specifically for min-hash signature matrix :-
   - Hash columns of signature matrix M using several hash functions
   - If 2 documents hash into the same bucket for at least one of the hash functions we can take the 2 documents as a candidate pair

   Procedure :
   - Divide the signature matrix into b bands, each band having r rows
   - For each band, hash its portion of each column to a hash table with k buckets
   - Candidate column pairs are those that hash to the same bucket for at least 1 band
   - Tune b and r to catch most similar pairs but few non similar pairs

Distance measures:

A distance measure is a function d(x,y) that takes two points in space as arguments and produces a real number, and satisfies the following axioms:

1. $d(x,y) \geq 0$ (no negative distances).
2. $d(x,y) = 0$ if and only if $x = y$ (distances are positive, except for the distance from a point to itself).
3. $d(x,y) = d(y,x)$ (distance is symmetric).
4. $d(x,y) \leq d(x,z) + d(z,y)$ (the triangle inequality).

Some of the distance measures we used are :

1. **Jaccard distance :** Jaccard distance = 1 - Jaccard similarity
   Jaccard similarity(J) = $| A \cap B | \, | A \cup B |$
2. **Hamming distance :** Hamming distance between two vectors is the number of components in which they differ.
3. **Edit distance** : The distance between two strings x and y is the smallest number of insertions and deletions of single characters that will convert x to y.
4. **Cosine distance** : For cosine distance, we do not distinguish between a vector and a multiple of that vector. The cosine distance between two points is the angle that the vectors to those points make. Given two vectors x and y, the cosine of the angle between them is the dot product x.y divided by the L2-norms of x and y

All these distances satisfy the mentioned axioms.

Precision

The fraction of relevant instances among the retrieved instances.

Recall

The fraction of the total amount of relevant instances that were actually retrieved.

Code Description

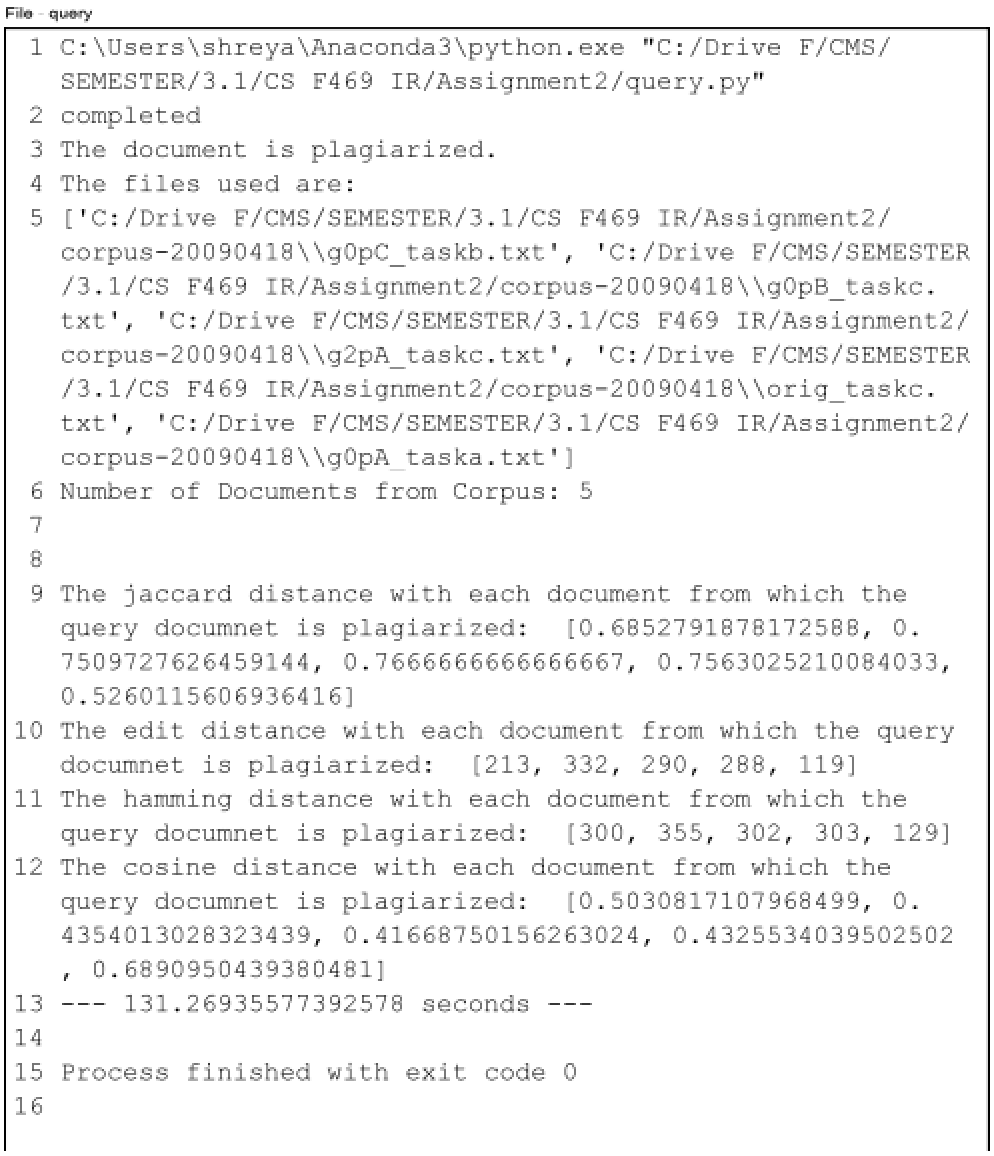
Utils.py - In this file, two functions are written, to read a document (for char) and to tokenize a document (for word).

lsh_Main.py - The purpose of this file is to apply LSH on the set of documents against which are query document will be tested later. We define a function with the path to the set of documents, the type(char or word) of minhashing and the threshold for similarity as the parameters. The function creates a MinHashLSH index using Redis as the storage layer and finally dumps the contents into a pickle file.

query.py - This file contains the code to apply LSH on the query document. The query document is minhashed according to the parameters specified in the previous file. If this document is found to be similar with other documents, a list of those documents is printed else a statement saying that the document is not plagiarised is printed. After getting the names of documents with which the query is similar, we find out the exact distance between the query and the list of documents using different distance measures such as Jaccard distance, Hamming distance, etc.

evaluation.py - In this file, functions are written to calculate the precision and recall of the plagiarism detector.

Sample Output:



```
C:\Users\shreya\Anaconda3\python.exe "C:/Drive F/CMS/SEMESTER/3.1/CS F469 IR/Assignment2/query.py"
completed
The document is plagiarized.
The files used are:
['C:/Drive F/CMS/SEMESTER/3.1/CS F469 IR/Assignment2/corpus-20090418\\g0pC_taskb.txt', 'C:/Drive F/CMS/SEMESTER/3.1/CS F469 IR/Assignment2/corpus-20090418\\g0pB_taskc.txt', 'C:/Drive F/CMS/SEMESTER/3.1/CS F469 IR/Assignment2/corpus-20090418\\g2pA_taskc.txt', 'C:/Drive F/CMS/SEMESTER/3.1/CS F469 IR/Assignment2/corpus-20090418\\orig_taskc.txt', 'C:/Drive F/CMS/SEMESTER/3.1/CS F469 IR/Assignment2/corpus-20090418\\g0pA_taska.txt']
Number of Documents from Corpus: 5


The jaccard distance with each document from which the query documnet is plagiarized:  [0.6852791878172588, 0.7509727626459144, 0.7666666666666667, 0.7563025210084033, 0.5260115606936416]
The edit distance with each document from which the query documnet is plagiarized:  [213, 332, 290, 288, 119]
The hamming distance with each document from which the query documnet is plagiarized:  [300, 355, 302, 303, 129]
The cosine distance with each document from which the query documnet is plagiarized:  [0.5030817107968499, 0.4354013028323439, 0.41668750156263024, 0.4325534039502502, 0.6890950439380481]
--- 131.26935577392578 seconds ---

Process finished with exit code 0

```