BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(October 30, 2020)

Design Document

Locality Sensitive Hashing

SUBMITTED BY

ANUSHA AGARWAL 2018A3PS0032H

JUI PRADHAN 2018B3A70984H

KRITI JETHLIA 2018A7PS0223H

Under the supervision of

Prof. Aruna Malapati

1. Problem Statement

We have to implement Local Sensitive Hashing to find out duplicate or similar DNA sequences within the corpus. The steps involved are Shingling, Minhashing and Local Sensitive hashing. The

main idea is to hash similar documents into buckets and the documents in a particular bucket have high probability of being similar or duplicates.

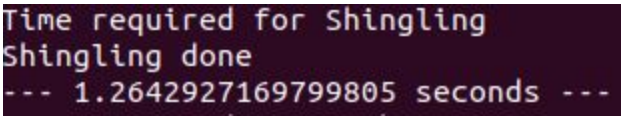
2. Algorithm

Our program takes a text query as input. In this case, it is a sequence of DNA. The output is the similar or duplicate sequences defined for a particular threshold.

The three major steps involved are

1. **Shingling**

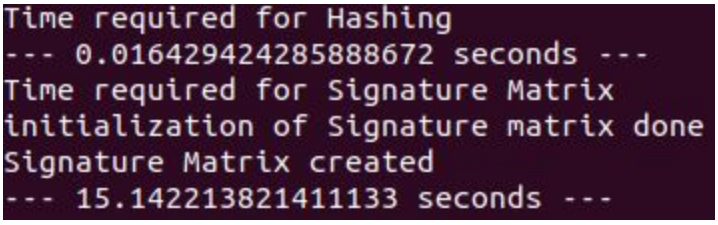
   A k shingle for a document is a sequence of k characters. It is similar to a k gram. Shingles are used to determine the similarity between two documents. We have used a dictionary to store shingles. The k value we have used is 5. We have also perform ed preprocessing and removal of unnecessary characters from our DNA dataset.



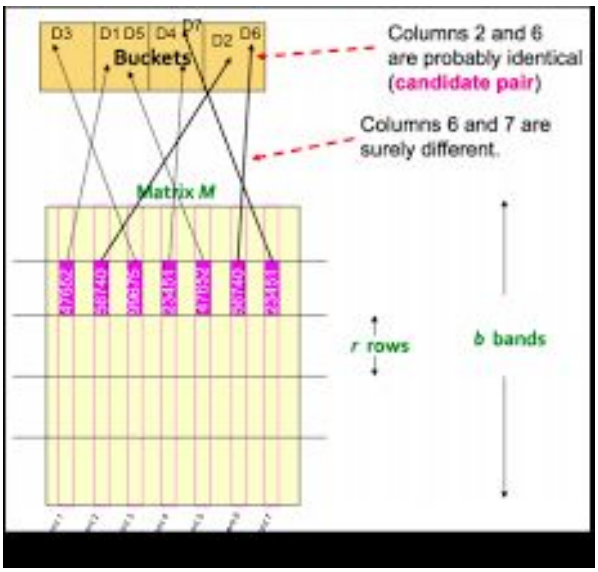
2. **Minhashing**

   The main goal of minhashing is to reduce the bigger matrix into a smaller signature matrix, while preserving the similarity between the documents. The hash functions we have used are of the form (ax+b)%c. We have used 100 hash functions in our implementation. We generate a signature matrix for these hash functions. The shape of the matrix is number of hash functions* number of documents in corpus.

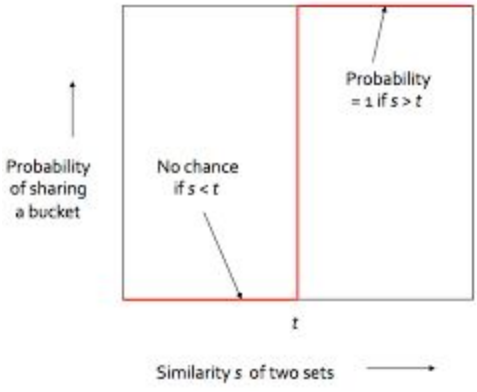
   The probability of the minhash function producing the same values determines the similarity between the documents.



3. **Local Sensitive hashing**



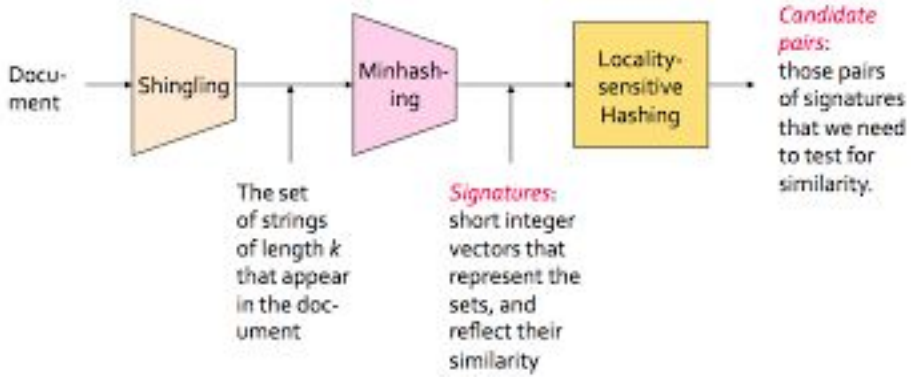
- The signature matrix is divided into b number of bands, where each band has a certain number of rows.
- The relationship between them is bands*rows=number of hash functions.
- The main idea is to put documents having similar hash values within a band into the same buckets.
- Similarity in one band conveys similarity throughout. Thus the documents in a particular bucket can be regarded as candidates for similarity and duplication.
- We can now apply any similarity metric to find the similarity within these candidates only. This considerably reduces the computations.
- We have used cosine similarity to find the probable similarity. Cosine similarity measures the angle between two vectors. Lower the angle, higher is the similarity. Thus higher the cosine values, more is the similarity.
- We also define a threshold value, above which, we consider two documents to be similar. The threshold value can be approximated by (1/bands) ^ (1/rows). If the probability value of sharing the bucket is greater than the threshold value, the items are similar.



- The choice of b depends on the fact lower b means higher similarity.As b increases, rows decrease and you get smaller signatures in a bucket, thereby increasing the number of false positives.
- We have taken our b value 5 and thus the number of rows is 20.

```
Banding Done
Time required for LSH
--- 0.10171151161193848 seconds ---
Time required for query time
--- 0.0035164356231689453 seconds ---
```

3. Flow Chart



4. Data structured used

- **Sets-** A set is an unordered and unindexed data structure. You cannot access its elements directly, a loop is required. We have used a set to store the candidates in a bucket. Also, we have used a set to store the document numbers containing a particular shingle.
- **Lists-** A list is a collection of ordered and mutable data. They are denoted by []. An item in a list can be accessed by index. We have used lists in various places like - storing the similar documents for a query along with their cosine values, storing the (a, b) coefficients of hash function, initialized signature matrix with lists containing infinity as elements.
- **Arrays**- We have used a numpy array to implement our signature matrix. Numpy arrays are 50 times more efficient than python lists because they are stored at a continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- **Dictionaries -** Dictionaries are used to store key and value pairs. We have created our buckets using an array of dictionaries, created the shingles using dictionaries.

5. Output Format

Document 21 is given as the input.

The program returns the documents in decreasing order of similarity. Here document 21 has the highest similarity.