CS F469: Information Retrieval

Open-Source Search Engine [LSH]

1. Ujjwal Raizada	2017A7PS1398H
2. Satyam Mani	2017A7PS0277H
3. Daksh Yashlaha	2017A7PS0218H
4. Pranjal Gupta	2017A7PS0124H

Locality-Sensitive Hashing Based Search Engine

Design Document

This search engine uses **locality-sensitive hashing function or nearest neighbour search** in higher dimensional space which is based on some kind of “distance” between the points.

The general idea of LSH is to use a family of functions (we call them LSH families) to hash data points into buckets, so that the data points which are close to each other are in the same buckets with high probability, while data points that are far away from each other are very likely in different buckets.

Here are the commonly used LSH family and their metric space[1]:

- Euclidean Distance - Bucketed Random Projection
- Jaccard-Distance - MinHashing
- Cosine-Distance - Random Hyperplanes
- Hamming-Distance - Random Bit Sampling

In this assignment, we have implemented three different Non-Euclidean measures namely, Jaccard-distance, cosine-distance and hamming distance.

Tools used for implementation

- Python3 is the language of choice because of rich in-built data structures and libraries for data pre-processing.
- NLTK library is used for data pre-processing and tokenization.

1

Components

Jaccard Distance Measure

The Jaccard Similarity of two sets is the size of their intersection divided by the size of their union.

Basic Implementation Overview:

1. Shingling : Converting documents into a collection of k-grams
2. Min-Hashing : Converting large documents set into equivalent signature matrix, preserving similarity.
3. Locality-Sensitive Hashing : Find pair of documents whose signatures are likely to be from similar documents.

Implementation Details:

1. Shingling:

9-gram shingle was used for making set of shingles, which was generated by extracting a sequence of 9 tokens appearing in the document. After extracting the set of shingles it is compressed to be stored such that it occupies less space. Compression is done using hashing shingle token into an equivalent integer hash.

Data structure used :

- Matrix_list : 2-D Array, used for storing the characteristic matrix.

2. Min-Hashing:

Each document is represented as the set of boolean vector having set of shingles as corresponding dimensions.

Each document vector is then hashed to corresponding equivalent signature using a hash function such that hashed document's signature fits in memory and similarity between signature of two functions is approximately the same as the original document vectors.

Hash function for Jaccard similarity is min-hash.

- We *randomly* pick 100 for hash functions.
- Hash function used: **(a * x + b) mod p,** where ***p*** is a prime number
- Ordering under different hash function gives a random row permutation

Data Structures used:

- **signature_matrix [ ] [ ] -** list of lists that store the signature matrix which is compressed version of characteristic matrix.

Algorithm details:

1. Compute $h_1(r)$, $h_2(r)$, . . . , $h_n(r)$.
2. For each column c, do the following:
    - A. If c has 0 in row r, do nothing.
    - B. However, if c has 1 in row r, then for each i= 1,2, . . . , n set SIG(i, c) to the smaller of the current value of SIG(i, c) and $h_i(r)$.

3. Locality-Sensitive Hashing

Columns of signature matrix is hashed several times so that similar items fall to the same bucket with probability being very high for similar items.

Algorithms details:

- Signature matrix is divided into ***b*** bands of ***r*** rows.
- Each column vector in a band is hashed into large no of buckets, buckets being different for every band.
- Columns that hash to the same buckets in different band are called *candidate pairs.*
- b and r are adjusted to minimise error and to get most similar pairs.
- Candidate pairs having similarity measure more than a threshold value *t*, are returned as similar items.

Prob(at least 1 band is identical) = $(1 - t^r)^b$

Note: b and r are tuned to get best S-curve.

Data Structure used:

- **Bucket** - A dictionary with band numbers as it's key and has another dictionary as its value pair(valueDictionary)
- **ValueDictionary** - It is a dictionary with hash value generated by hash function used in the band and value being set of documents in the corpus.

- **Bucket** : { key, value }
  - Key : band number
  - Value : **valueDictionary**

- **valueDictionary :** { key, value }
  - Key : hash value generated by hash function
  - Value : set of documents in the corpus

Cosine Distance Measure

Cosine distance between two points in space is defined as the angle made between vectors of these points.

Implementation Details:

- The hashing technique used for cosine distance is called random hyperplanes.

- In random hyperplanes technique, we choose random vectors from the set of all possible vectors whose components can only take two values (+1 or -1). We assume these random vectors as normal vectors to the hyperplane in the space containing the document vectors.

- We try to divide the space into two groups which is determined by the dot product (1 if positive, else 0) of the document vector with all the normal vectors.

- This process is repeated several times to get a signature matrix which represents the actual compressed characteristic matrix.

Signature for cosine similarity is called sketches.

Data structure used:

- norm _vector_list [ ] : list of random hyperplane vectors
- signature_matrix : signature matrix generated by following the said theory

Hamming Distance Measure

Hamming distance between two points is defined as the no of components in which they differ.

Implementation Details

- We randomly took n (say, 100) dimensions from the characteristic matrix.

- The probability of including any random dimension (say m) is 1/d, d is the number of dimensions in the characteristic matrix.

Data structure used:

Signature_matrix [ ] [ ] : saves the ith dimension of the document vectors in the jth row where 1 <= j <= number and i is chosen randomly.

Performance comparison

Performance:

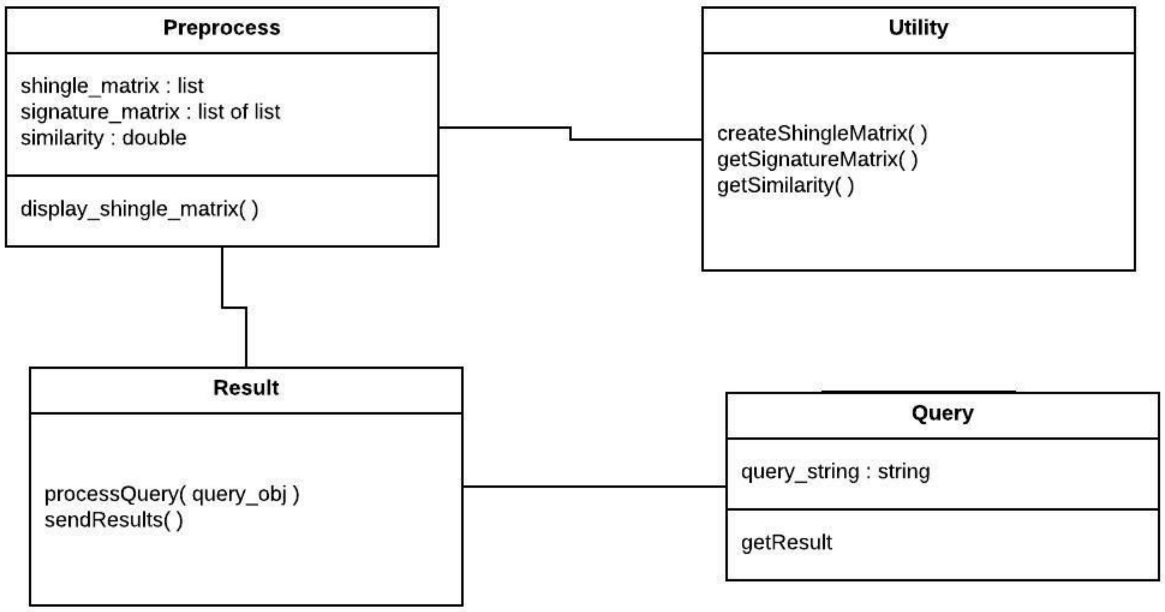
The performance of a model is judged on the basis of precision as well as recall it provides for different queries. We took 10 labeled documents and ran different queries and compared the results with the expected results, then took the average of the result..

- **Precision:** Number of actually relevant documents among the returned result.
- **Recall:** Number of relevant documents returned among the expected documents.

Note: Automatic calculation was not possible because our data (github README) is not labelled.

	Cosine Similarity	Jaccard Similarity	Hamming
Precision	0.8147	0.86288	0.87933
Recall	0.7911	0.7625	0.7892

Class Diagram


Preprocess
shingle_matrix : list
signature_matrix : list of list
similarity : double
display_shingle_matrix( )
Utility
createShingleMatrix( )
getSignatureMatrix( )
getSimilarity( )
Result
processQuery(query_obj)
sendResults( )
Query
query_string : string
getResult