BITS Pilani Hyderabad Campus

CS F469 IR Assignment – 2

Locality Sensitive Hashing

Team Members:

1. Kushagra Gupta 2018A7PS0208H
2. Parveen 2018A7PS0623H
3. S. Sai Naga Shashank 2018AAPS0347H

Design Document

LSH works on the principle of grouping similar items from the corpus together by some means, so that while finding similar items for a target item we need not traverse each and every item in the corpus.

Dataset:

Kaggle Human DNA dataset.

It contains a single file with 4380 genome sequences.

Similarity Metric and LSH Family:

We implemented Jaccard distance as the similarity metric.

And its corresponding 'Min Hashing' technique to find the Signature Matrix

Tools/Packages Used:

- Python 3.8

Overview of the program flow:

1. Shingling: Break genome sequence into k-grams
2. Min-Hashing: Generating a smaller signature matrix
3. Locality-Sensitive Hashing: Grouping similar genome sequences

Implementation:

- **Shingling:** Every sequence is divided into k-grams.
  - If 'k' value becomes larger, then it can possibly lead to huge memory consumption, hence each shingle is converted into an equivalent hashed integer.
  - A 2D matrix (Document Matrix), where a cell represents the presence of a particular shingle in a particular sequence can be highly expensive in terms of space complexity because that matrix will be very sparse. Hence, we maintained sets for each sequence containing respective shingles.
  - 'k' value taken = 10.

- **Min-Hashing:** A 2D Signature Matrix has been created to reduce the memory consumption by the from the shingles set.
  - Signature Matrix has been created in such a way that it also preserves the similarity of the sequences.
  - This is done by generating hash functions of the form:
    - (a * x + b) % mod
    - a, b -> random numbers in the range [1, NO_OF_HASHES]
    - NO_OF_HASHES used are 200
    - x -> will be the row number of the Doc Matrix for which Min-Hashing is done.
    - mod -> no of total shingles from the corpus

  Algorithm used is same as taught in the class:

```
for each row r do begin
  for each hash function h_i do
      compute h_i(r);
  for each column c
      if c has 1 in row r
        for each hash function h_i do
            if h_i(r) is smaller than M(i, c) then
                M(i, c) := h_i(r);
end;
```

- **<u>Locality-Sensitive Hashing:</u>** Whole Signature Matrix is divided into **b** bands, where each band contains **r** rows.
    - No of bands used (**b**) = 10
    - Band Width or number of rows in each band (**r**) = 200/10 = 20
    - For each band, every sequence column with **r** entries is hashed and put into a bucket.
    - If any 2 sequences come together in the same bucket in any of the band, they are considered as similar / "Candidate Pairs".
    - Probability that 2 genome sequences will be the Candidate Pairs in at least one band = $1 - (1 - s^r)^b$, where s is the Jaccard similarity between 2 sequences

    - Since, the sequences in the dataset are more than 80% similar, we have taken b = 10, r = 20 such that only the sequences which are more than 95% similar will most probably be in the same bucket.
    - If s = 0.9, then P (Candidate Pairs) = 0.73
    - If s = 0.95, then P (Candidate Pairs) = 0.99

<u>Data Structures:</u>

- **b** maps, where each map contains the buckets of the sequences for a particular band

Query Calculation:

- All LSH Steps applied to the query as well, and all the sequences which are present in the buckets to which query has been put are returned.

Runtime:

K (Shingle length) ->	**5**	**8**	**10**
LSH Time on Dataset	120 seconds	270 seconds	740 seconds
Per Query Time	0.04 seconds	2 seconds	28 seconds

- ✓ This could have been improved by approx. 8-10 times by storing very sparse Document Matrix and efficiently applying Numpy operations on the matrix, but at the cost of memory need in GBs. Hence, we decided to go with space efficient optimization only.