Design Document

Made By- Mudit Chaturvedi (2018A7PS0248H), Sristi Sharma (2018A7PS0299H), Tanay Gupta (2018AAPS0343H)

Making Shingles -

- Shingle size is selected according to the minimum size of documents in our corpus.
- To store the documents in which a shingle appears we have used a pandas DataFrame, as it has many vector operations which reduces the time needed for MinHashing and LSH step.
- The function takes in the path of the database, the size of shingles and a query as input and creates this table.
- We used a dictionary while creating this table due to its dynamic size and fast access. This is later converted to a pandas DataFrame.
- The time complexity of this function is O(mn), where m is the number of documents in our database and n is the length of each document.
- This file is saved in pickle format for later use.

Min Hashing -

- Given the shingle to document matrix, we need to create a signature matrix.
- We took 100 permutations of the original shingle order as 100 permutations gave us the best results in a reasonable amount of time, also this allowed us to experiment with different sizes of buckets and rows in the LSH step.
- The function takes in the path of the shingle document matrix and the number of permutations to be generated.
- The process to generate one row of this signature matrix is as follows:-
    - Original order of the shingles is shuffled.
    - The first non zero row is then seeked out, for every document in the corpus in this shuffled matrix. This step is performed using a numpy mask which allows us to vectorize the process which gives us around 40 times boost in terms of speed in comparison to looping through the matrix.
- This process is repeated 100 times to generate the signature matrix, of dimension (100x number of documents in the corpus+1(for query)) .
- The time complexity of this process is O(kmn) where m is the number of documents, n is the number of shingles and k is the number of permutations.
- The above step takes around 2 seconds whereas looping through the entire shingle document matrix to generate the signature matrix takes more than 80 seconds.

- This signature matrix is stored in the form of a pandas DataFrame to take advantage of its many vector operations.

Locality Sensitive Hashing -

- From the signature matrix returned by the MinHash step, we proceed to divide the matrix into ‘b’ bands with ‘r’ rows each for every column (document in the corpus and the query document).
- We take the number of buckets for each band to be 1,00,000 . Note this value can be increased further but it will lead to more sparse and irrelevant computations in the buckets and any value close to the number of documents runs the risk of being overly crowded in one bucket while being sparse in others, thereby defeating the purpose of minhashing.
- Taking the number of bands to be 20, with a size of 5 rows in each, we calculate the sum of the portion of each column in each band.
- For each band, we hash the sum of each column’s portion in the band to the buckets.
- A ‘bucket’ dictionary is created which uses the band number as key and the buckets of the band as value. The buckets for each band are dictionaries with sum hash as the key and set of documents as value. Bucket creation is completed in O(bn), where b is the number of bands and n is the number of documents.
- Having created this bucket dictionary we now store all the documents that are hashed to the same bucket as our query, for all bands in a list.
- This list contains all our candidate pairs and for each document in this list, we check the Hamming Similarity between the document and query. Checking the Hamming Similarity between two documents takes O(b) time where b is the number of bands. Doing this for all the ‘m’ candidate pairs, time complexity increases to O(bm).
- If the similarity is more than the threshold then we store the document id in a dictionary ‘ranked_res’, corresponding to its similarity value.
- ‘ranked_res’ uses similarity value as key and list of documents as value, as output of the LSH step, the dictionary is sorted in descending order of similarity scores and documents corresponding to the top k similarity scores are returned.
- Top k results are returned in O(k) time.
- Overall time complexity for the LSH step is O(bn), where b is the number of bands and n is the number of documents. [Since n>>m, if an ideal number of bands and rows are used].

Analyzing Results -

- Since the entire concept of Locality Sensitive Hashing relies on random generation of permutations, there is a possibility that for the same query, the top result may be different each time, for example, say if the query given is most similar to documents 2,4,6,7,8 and 15. Then for different permutations, different signature matrix and hash buckets are obtained and the relative ordering (by similarity) of these documents can be different.[For one run, the ordering on descending similarity can be 4,7,2,8,6,15 while for some other run it can be 2,4,6,7,8,15]
- However, this difference in results is within an acceptable range as the similarity scores are always higher than the threshold (almost always ranging from 0.93 to 1.0).
- The results are dependent on the length of shingles, if shingle length is too large the matrix will be too sparse and calculations redundant while if it is too small then the matrix will be too dense and there won't be any point in using LSH..
- If the query length is too small then an empty set will be returned as the similarity with all the documents is less than the threshold.
- The minimum length for a query to give relevant results depends on the size of the document which it is similar to but it is observed that a query of length less than 200 sequences gives empty results almost all the time.
- The whole process from creating shingles to returning similar documents is completed under 4 seconds.