Information Retrieval Assignment 2

Abhishek Kumar, Gaurang Gupta, Shubham Asopa

Overview

Team

- Abhishek Kumar - 2018A4PS0653H
- Gaurang Gupta - 2018A7PS0225H
- Shubham Asopa - 2018A7PS0101H

On Quora, there are millions of questions asked every day and hence many of them might have already been asked and they have to be marked as duplicate. For the same, Quora uses Random Forest Model technique. This project is a different approach for the same problem using LSH.

The dataset contains a list of questions asked on Quora. The user will give a query in the form of a question string and our algorithm will fetch the questions which best match to the query.

Cleaning Of The Raw Data

The raw data was taken from "Kaggle Quora Questions Dataset" in the form of a csv file. The data was converted to a JSON file and each of the questions were assigned a unique id.

This was executed in clean.py file and the JSON file was stored as Questions.json. This process took around 40 seconds. The data is now cleaned for querying and we only had to run this once.

Shingling

In this step, we're making shingles from our data and assigning unique id's to the shingles and making the document shingle matrix.

From our data-set, we're making shingles of size 6 and making a set from them to avoid duplicate shingles. Around 850 thousand unique shingles were identified and were assigned unique id's and were stored in a JSON file for further use.

Now, for each document, we're finding the shingles which are present in the document and storing the set of the shingles corresponding to that document. All the documents and their corresponding shingles were stored in another JSON file creating the document shingle matrix.

This was executed in shingling.py file. The JSON file containing the unique shingles and their id's is shingles_id.json and the JSON file containing the document shingle matrix is doc_shingle_dense.json. This process took around 30 seconds. We only had to run this once and this will work for all queries.

Min-Hashing

In this step, we will be creating the signature matrix from the document shingle matrix using multiple hash functions.

First, we'll be generating 120 random hash functions in the form of $(ax + b)\%p$ where a, b and p can take values from 0 to number of shingles. We're also storing these hash functions in another JSON file so that we apply the same hash functions to our query.

Now, for each document, we'll be applying each hash function on the document and calculating the corresponding value of the cell in the signature matrix by applying the hash function on each shingle in the document and taking the minimum hash value. This process will give us the signature matrix which is stored in a JSON file for future use.

This was executed in min_hashing.py file. The JSON file containing the hash function values is hashFunctions.json and the JSON file containing the signature matrix is signature_matrix.json. This process took around 12 minutes. The time taken is huge as we have around 300 thousand douments and we're applying all hash functions on each of the documents which is time consuming. The signature matrix is now ready for LSH step.

LSH

In this step, we'll be applying Locality Sensitive Hashing technique to our signature matrix by hashing the values in the matrix to different buckets in different bands.

For each document, we're taking the corresponding values in the signature matrix and dividing it into bands and calculating the corresponding hash values using the inbuilt hash function in python and storing which bucket the corresponding document the document belongs to for that band. The result is now stored in another JSON file as it can be used multiple times for queries and hence is pre-processed.

We're using 30 bands with each band of 4 rows and 10000000097 buckets for each band.

This was executed in lsh.py file. The JSON file containing the band values and their corresponding bucket and document values is hashFunctions.json.. This process took around 50 seconds. Now we can query on this data.

Query

The user can now enter a query and find the corresponding documents which match the query.

After the user enters a query, shingling is done on the query and min hashing using the same hash functions which were used on the documents. LSH step on the query results in a set of documents which match with it in one or more bands i.e. they fall in the same bucket for a band. Then, we're calculating Jaccard similarity for the documents which were matched with the query and the user will see the top 5 documents with the highest Jaccard similarity with the query or an appropriate message if the document is not present. The user is also presented with an option if he/she wants to execute more queries or wants to quit.

This is processed in query.py and the average time taken by the query is even less than a second with 8 seconds for pre loading the corresponding JSON files. Some sample runs can be found here, here, here and here