Search the Bible using LSH - DESIGN

Data set:

Our dataset consists of 10000 documents. Each document has the Citation, Book name, Chapter number, verse number and the verse. This dataset was taken as CSV(Comma Separated Values) file.

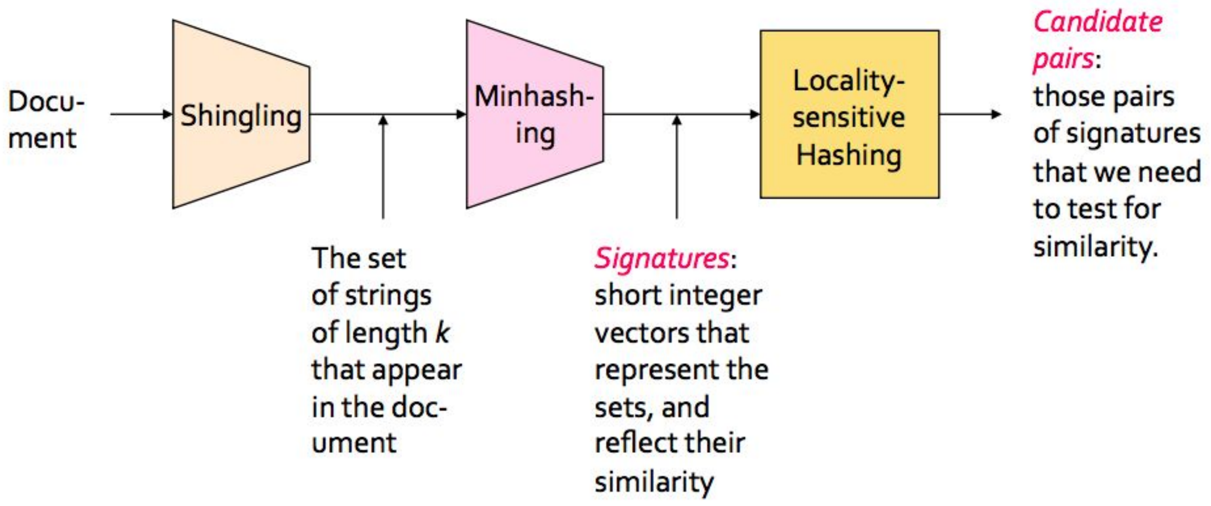
Technologies:

Python
Flask Web framework
HTML
BootStrap 4(CSS + JS)

Implementation:

We used Locality Sensitive Hashing to create a search engine for the Bible dataset. The steps involved are Shingling, Minhashing and Locality Sensitive Hashing.

The dataset is read using python - CSV library. The text part in each document is tokenized and removed stop-words using python - nltk package.


Docu-ment
Shingling
Minhash-ing
Locality-sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity.
The set of strings of length k that appear in the doc-ument
Signatures: short integer vectors that represent the sets, and reflect their similarity

Step 1 - Shingling

- We parsed all the documents and for each document we consider all the words present in that document.
- We remove stop words like the, for, themselves, etc. We then converted all uppercase letters in each word to lowercase letters.
- After these changes were done, we considered shingles of size 3 for a continuous stream of the sentence. For example - the sentence ‘to be or not’ gives us the shingles - tob, obe, beo, eor etc. We created a list of all the shingles present in each document.
- For each of the document, we create a vector space model, which represents the shingles present in the document out of all the possible shingles. We constructed the shingling matrix for the entire dataset.
- The same was done for query too.

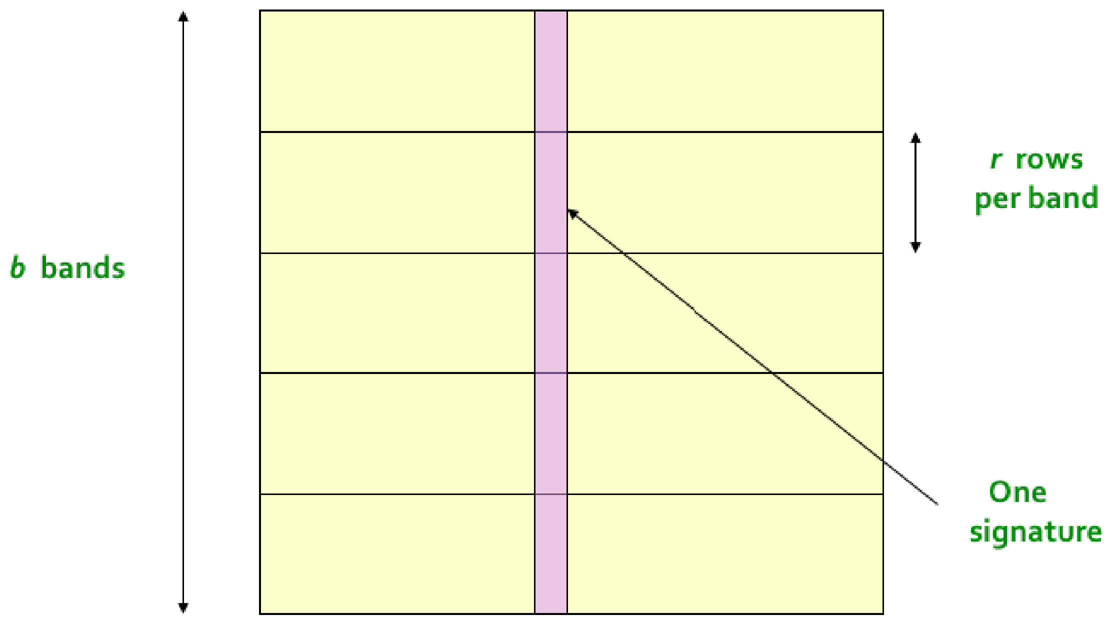
Step 2 - Minhashing

- We considered ten hashing functions to hash our shingling matrix into a smaller signature matrix. The hash functions were generated at random.
- The rows were permuted according to the hash functions, each of the row containing the first row where we could find 1.
- The final signature matrix has 10 rows and 10,000 columns.
- The same procedure was applied to the query resulting in 10 rows and 1 column.

Step 3 - Locality Sensitive Hashing

- We divide the signature matrix into b bands, each band having r rows.
- For each band, hash its portion of each column to a hash table with k buckets.

- Candidate column pairs are those that hash to the same bucket for at least 1 band.
- If 2 documents hash into the same bucket for at least one of the hash functions we can take the 2 documents as a candidate pair.



Signature matrix *M*

Step 4 - Extracting Documents:

- **Candidate Docs:**

After getting the 3-D bucket matrix and the query bucket matrix, we go through each band and in turn to each bucket and extract the documents in which the query got hashed. Now we have all the candidate documents.

- **Filtered Docs:**

After getting the candidate docs, we find the similarity of each of the candidate docs with the query doc using different similarity measures and extract the documents with most similarity. The different similarity measures used are mentioned below:

- **Distance measures used:**
    - Euclidean Distance

- Jaccard Distance
- Hamming Distance
- Cosine Distance
- Edit distance

Running times:

For 1000 documents, 100 hash functions:

Getting shingles:	0.67 secs
Making Shingles matrix:	0.77 secs
Making Signature matrix:	1.93 secs

For 5000 documents, 100 hash functions:

Getting shingles:	4.46 secs
Making Shingles matrix:	5.40 secs
Making Signature matrix:	8.67 secs

For 10000 documents, 100 hash functions:

Getting shingles:	8.60 secs
Making Shingles matrix:	14.1 secs
Making Signature matrix:	20.1 secs

Query Search time for 10000 documents and 100 hash functions:

On an average:	0.70 secs