MinHash based Locality Sensitive Hashing (LSH) for Document Matching

Raman Kumar Gupta

Locality Sensitive Hashing (LSH)

- LSH is technique used to cluster together similar items in a common bucket. LSH is used for Nearest Neighbour search for finding similar documents in a large corpus of documents.

- Instead of using the brute force approach of comparing all pairs of items, LSH instead hashes them into buckets, such that similar items are more likely to hash into the same buckets. As a result, for the problem of finding similar documents from a corpus, matching a given document, the search space and the number of comparisons needed is considerably reduced.

- Only the items which hash to the same bucket need to be compared.

- LSH differs from a cryptographic hash, because aim here is to maximise collisions to bucketize the similar documents.

- LSH has wide variety of use cases :: finding textually similar documents, eliminating near-duplicates among documents, collaborative filtering and dimensionality reduction.

- Practical examples include detecting Plagiarism, near duplicate removal of online articles in web search, recommendations, image search, matching fingerprints etc.

MinHash

- One of the methods in LSH family is MinHash or the min-wise independent permutations. **MinHash quickly estimates Jaccard Similarity** of two documents and thus finds degree of “similarity” of 2 documents. This degree of similarity can be used to rank nearest neighbours or match a document against a corpus of documents.

- MinHash LSH Technique is the focus of my presentation.

- MinHash LSH algorithm helps

1. Reduce document search space
2. Transforms from string comparison to machine word sized integer list comparison, which is much faster.

- As the size and number of documents grows the improvements in speed is more pronounced.

Jaccard Similarity

- Jaccard Similarity defines the degree of “similarity” of 2 sets. The Jaccard Similarity of two sets is defined as the ratio of size of their intersection and size their union.

- $J = |A \cap B| / |A \cup B|$ ➔ It is also known as *intersection over union (IOU)*.

- Here the similarity focus is on lexical similarity instead of semantics similarity i.e. character-level or word-level similarity, not “similar meaning” similarity. A very trivial example

- A = King of England B = King of Poland C = Queen and King

- Jaccard Similarity of A & B = 2/4 = 0.5 → A & B are more similar than other pairs, since J is highest
- Jaccard Similarity of A & C = 1/5 = 0.2
- Jaccard Similarity of B & C = 1/5 = 0.2

- The primary use case of Jaccard Similarity or MinHash is to find textually similar documents in a large corpus. Example detecting plagiarism, mirrored webpages etc.

How I used MinHash LSH Algorithm

- To match a given user JSON document against a large corpus of JSON documents to find the most matching document.
- MinHash LSH algorithm was complementary to main proprietary matching algorithm which had many assumptions built into for optimisation. MinHash LSH offered a more broader corpus search and led to overall improvement in matching rates.

Problem:

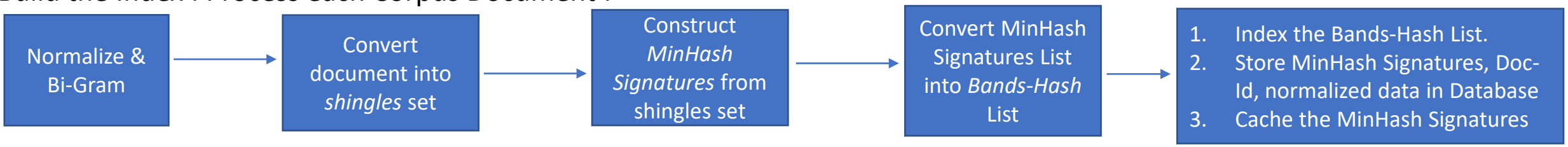
- A user provided sparse entertainment (movie/episodes) metadata JSON document consisting of fields like Title, Language, Genre, Cast, Director, Release Year, Duration, Production House, Description, Summary etc.
- This user JSON document is matched against a much larger corpus of proprietary *enriched* JSON documents. The corpus documents have more and enriched fields, has complete and correct data, provides summary, trailer & poster links etc.
- *The task is to match a simple user document against the corpus and returning the most matching document from the corpus back to the user.*

Solution:

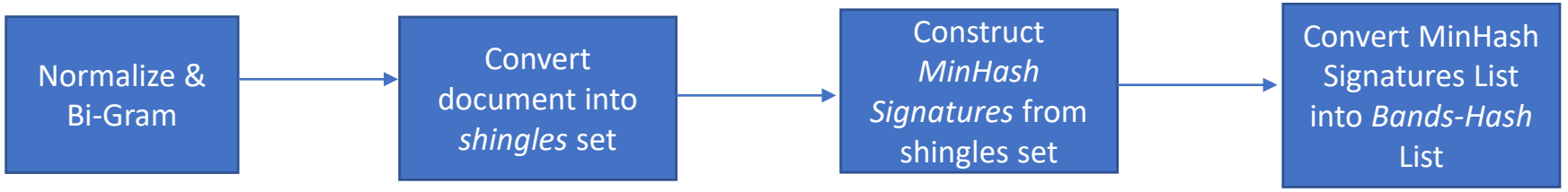
- MinHash LSH algorithm was used to match and then rank the matched corpus documents.
- A lower bound on Similarity Threshold was mathematically derived to define minimum similarity below which documents will not be considered for matching.
- This was not simply a Title database search because it handles inconsistent or wrong user input data, missing fields (like missing Title) in user input, handling user spelling mistakes, matches against dynamic daily updated corpus, scoring of matched results etc.

Details

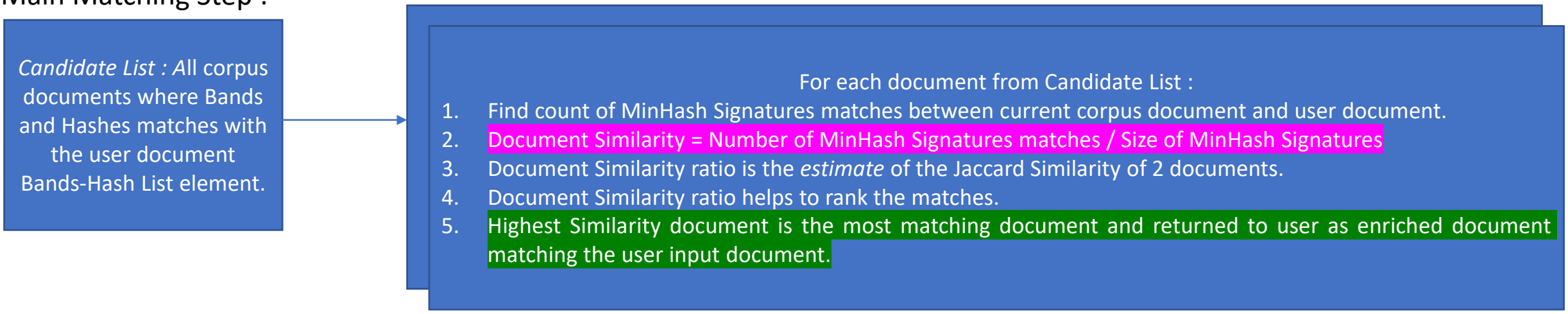
Build the Index : Process each Corpus Document :



Process User Input Document :



Main Matching Step :



Example Execution

Sample Corpus Documents

- **Titanic_1953** = {"Title" : "Titanic", "Id" : "Id1", "Type" : "movie", "ReleaseYear" : "1953", "Language" : "English", "Director" : "Jean Negulesco", "Cast" : "Clifton Webb, Barbara Stanwyck", "Description" : "Estranged couple sailing on the ill-fated maiden voyage of the RMS Titanic, which took place in April 1912.", "Duration" : "98" }

- **Titanic_1997** = {"Title": "Titanic", "Id": "Id2", "Type": "movie", "ReleaseYear": "1997", "Language: "English", "Director": "James Cameron", "Cast": "Leonardo DiCaprio, Kate Winslet", "Description": "A young rich girl falls in love with a kind but poor artist boy aboard the luxurious, ill-fated R.M.S. Titanic.", "Duration": "195"}

- **Godfather_Part1** = {"Title": "The Godfather", "Id": "Id3", "Type": "movie", "ReleaseYear": "1972", "Language": "English", "Director": "Francis Ford Coppola", "Cast": "Marlon Brando, Al Pacino", "Description": "An organized crime dynasty's aging patriarch transfers control of his clandestine empire to his reluctant son.", "Duration": "177"}

- **Slumdog_Millionaire** = {"Title" : "Slumdog Millionaire", "Id" : "Id4", "Type" : "movie", "ReleaseYear" : "2008", "Language" : "English", "Director" : "Danny Boyle", "Cast" : "Dev Patel, Frida Pinto"," Description": 'A Mumbai teenager reflects on his life after being accused of cheating on the Indian version of "Who Wants to be a Millionaire?"', "Duration" : "120"}

- **Pretty_Woman_English** = {"Title" : "Pretty Woman", "Id" : "Id5", "Type" : "movie", "ReleaseYear" : "1990", "Language" : "English", "Director" : "Garry Marshall", "Cast" : "Richard Gere, Julia Roberts", "Description": 'A man in a legal but hurtful business needs an escort for some social events, and hires a beautiful mistress only to fall in love with her', "Duration" : "120"}

- **Pretty_Woman_Spanish** = {"Title" : "Pretty Woman", "Id" : "Id6", "Type" : "movie", "ReleaseYear" : "1990", "Language" : "*Spanish*", "Director" : "Garry Marshall", "Cast" : "Richard Gere, Julia Roberts", "Description": 'A man in a legal but hurtful business needs an escort for some social events, and hires a beautiful mistress only to fall in love with her', "Duration" : "120"}

Sample Processing of Corpus Document Titanic_1997:

1. Normalize, Bigrams and convert into shingles => ['Ti', 'it', 'ta', 'an', 'ni', 'ic', 'James', 'Cameron', 'Leonardo', 'DiCaprio', 'Kate', 'Winslet', '1997', '195', 'English']
2. MinHash Signatures List of length 1000 => [51698105, 71268306093….]
3. Bands-Hash List of length 250 (b=250, r=4, b*r=1000) => [679105346, 307527197…]

Sample User Inputs:

User inputs with Typical mistakes in user data are also mentioned.

- **Incomplete Director name. No Cast.** Req1 = {"Title": "Titanic", "Id": "Req1", "Director": "Cameron"}
- **Incomplete Title Name. No Cast.** Req2 = {"Title": "Slumdog", "Id": "Req2", "ReleaseYear": "2008"}
- **Misspelled Title Name. No Director.** Req3 = {"Title": "god fathr", "Id": "Req3", "Cast": "Marlon Brando"}
- **No Director or Cast.** Req4 = {"Title": "Titanic", "Id": "Req4", "ReleaseYear": "1953"}
- **Spanish version of originally English language movie.** Req5 = {"Title": "Pretty Woman", "Id": "Req5", "Language": "Spanish"}

Sample Processing of User Document Req1

1. Normalize, Bigrams and convert into shingles => ['Ti', 'it', 'ta', 'an', 'ni', 'ic', 'Cameron']
2. MinHash Signatures List of length 1000 => [51698105, 71268306093….]
3. Bands-Hash List of length 250 (b=250, r=4, b*r=1000) => [679105346, 307527197…]

Main User Document Matching Step

- Candidate List: Select all corpus documents from database where band and hash matches that of the user document Bands-Hash List.
  - For **Req1** the Candidate List: [**Titanic_1953, Titanic_1997**]

- From this Candidate List:
  - Jaccard Similarity of **Req1** and **Titanic_1953, Titanic_1997** estimated from MinHash Signatures List => [0.421, 0.51] {Document Similarity = Number of MinHash Signatures matches / Size of MinHash Signatures}
  - The Actual Jaccard Similarity => [0.434, 0.543]
  - Jaccard Similarity estimated from MinHash Signatures List is nearly same.

- Highest Similarity score is of the corpus document **Titanic_1997,** and thus it is the most matching corpus document for user input **Req1**

- Complete Results

User Input Document Id	Matched Corpus Document
Req1	Titanic_1997
Req2	Slumdog Millionaire
Req3	Godfather_Part1
Req4	Titanic_1953
Req5	Pretty_Woman_Spanish

Benefits from Using LSH

1. Quantity of Matches Improvement:
   a. MinHash LSH led to about 15% overall improvement in matching rates.
   b. High confidence matching rates (matching score >= 0.70) improved by nearly 8%.
2. Quality of Matches Improvement:
   a. Similarity Threshold : A lower bound on Similarity Threshold was mathematically derived to define minimum similarity below which documents will not be considered matched. This led to removal of poor-quality matches passing through and improved quality of matches provided to user.
3. Matching Latency of about 300 ms was achieved by using Band-Hash List and caching of MinHash Signature List.

Improvements

1. Candidate List (Band - Hash List) :
   a. Use of Band - Hash List to *reduce search space* by creating a small Candidate List of similar corpus documents was a major improvement. Instead of finding Jaccard Similarity for all corpus documents, narrow it down to just the Candidate List of documents.
   b. The Band - Hash List is much smaller (typically 1/4 or 1/5) than MinHash Signature List, thereby further reducing comparisons needed to generate Candidate List.
   c. Corpus documents Band-Hash List is Indexed. The indexing improved the search for selecting the corpus documents whose Bands & Hashes matches with user document Band-Hash List elements. I stored it them a PostgreSQL table and created indexes on them.
2. Singles:
   a. The choice of 1 word shingles was reasonable given small document size. For larger documents shingle is typically 5-9 words long.
3. Others:
   a. Handle user spelling mistakes, partially wrong or missing data by use of Bi-Grams of Titles.
   b. Used Redis to cache the corpus MinHash Signature List so that getting them for each candidate is faster.