**LOCALITY SENSITIVE HASHING**

**The data set:**

We used the Musixmatch million songs dataset to train our retrieval model. It consisted of formatted data of 27143 different songs with their unique track IDs, term frequencies of the overall top 5000 terms, that constituted roughly 92% of all occurrences in the entire corpus of lyrics available. The dataset was taken as a TXT file.

**Technologies used:**

Programming Language- Python 3.7

Libraries- Porter2stemmer (for normalization), Pandas(for reading files), NumPy (for working with arrays), CSV (for creating file that stores tf-idf scores), Math (for logarithmic and other calculations), Random(used for creation of hash functions).

**Implementation:**

1. The data is taken in form of a text file and is converted into a binary term coincidence matrix where every 1 represents the shingle being present(here we are using 1-shingles,i.e.,the terms themselves as shingles) and 0 represents the shingle being absent in the song. This matrix is stored as a CSV file and for further applications is read and used as the same.
2. Minhashing: We have chosen to reduce the dimensions of our term coincidence matrix to half so as to not lose a lot of latent information which would be used for calculating similarities. So, number of permutations= Number of shingles/2 is done and those many signatures are generated for each document(song in this case). The signatures are stored in a dictionary with key as track ID and value as the list of its signature values for all the permutations.
3. Documents are further sliced into bands of same length which would be used for creating buckets of candidate items.
4. Euclidean distance: The Euclidean hash family is generated by the formula H(x)=u.x/w +b, where w is number of bands, u is a random vector with iid values taken from a normal distribution N(0,1), b is again a random bias from a normal distribution N(0,w). The hash functions are kept same for each band, however they are randomly selected for all bands. These hash functions create buckets of similar entities in each band.
5. Hamming distance: The hamming hash family simply maps each document slice of a band to one of its dimensions. These dimensions are again selected as random between (0,r) where r is number of rows in each band. In this manner buckets for each band are generated and stored in a dictionary.
6. Angle distance: The angle distance hash family is given by H(x)=signum(u.x) where u is a random vector of dimension r, its values again chosen from a normal distribution. These hash functions map the document slices in each band to respective buckets.
7. Query is pre-processed in a similar fashion as assignment 1. It is treated as a document and its minhashes are generated and its slices are hashed in the same way as described above. OR-LSH technique is used to find the candidate items that are used for testing for similarity. Only the top 10 similar items are returned.
8. Comparative analysis:

On giving input query for all three distance measures separately and comparing the outputs, we found that Euclidean LSH gives the best output, followed by Hamming and then angle distance LSH. One possible reason is that in both Euclidean and Hamming distances the actual latent information is not entirely lost but in angle distance we are only taking the signs of each signature as the mapped value, so the similarity information is somewhat lost.

**Data Structures used:**

Dictionary: Used for storing the dataset in the form document ID: string containing term frequencies of the respective terms, for storing

List: Used for storing buckets corresponding to each band.

Dictionary of lists: For storing signatures in the form document ID: list of signatures generated using different permutations.