**DESIGN DOCUMENT**

CS- F469

ASSIGNMENT-2

PLAGIARISM DETECTOR USING LSH

Piyush Mishra – 2016B2AA0633H

Abhishek Singh – 2016A7PS0093H

Nimit Kanani – 2016B3A70464H

1. INTRODUCTION

In this project, we investigate the similarities between documents from a corpus consisting of various news articles related to sports domain.

For the entire project, we use python 3.6 with a few external libraries such as:

1. NLTK for text preprocessing
2. Binascii for converting shingles into integers
3. DATASET & PRE-PROCESSING

The dataset used in the project was obtained from Kaggle.

For pre-processing we used a regular expression (see the code) to remove all punctuations such as full stops, commas, exclamation points, semi colons, etc. Also all the text was converted into lowercase.

1. K-SHINGLING

Each document was split into k-shingles, where k is a positive integer input by the user. Number of shingles in a document of d words is (d-k+1). We used list data structure to keep track of all shingles present in a particular document. Finally, to obtain the list of all unique shingles across all documents in the corpus, we used set data structure. As the information we need is whether a shingle appears in a particular document or not, and not how many times a particular shingle appears in a particular document, we hashed the shingles into integers using the binascii.crc32() of the binascii module. This was done because cost of checking presence of a particular integer is smaller as compared to a shingle.

Time taken for constructing the shingling matrix for the corpus is O(N\*(d-k+1)), where N = number of documents in the corpus, d = number of words per document, k = length of shingle.

1. MINHASHING & SIGNATURE MATRIX

Instead of taking the shingle sets of each document and comparing it to every other document, we can just compare their signatures. We provide the user with the option to choose the number of hash functions for this purpose. All the hash functions output a random permutation of numbers from 1 to n where n is the number of hash functions input by the user.

Finally, to create the signature matrix that was aimed for, we initialized all cells of a n by D dimensional matrix to sys.maxsize. Then for a particular combination of hash function and a document, we obtained it’s signature using the method described in class. In this way we ended up with the signature matrix.

1. LOCALITY SENSITIVE HASHING

Having obtained the signature matrix, next step was to find the candidate pairs for testing similarity. For this we used the banding technique as described in the text book. We divided the hash functions into a b bands, each containing r rows, such that b\*r = n (number of hash functions). We then took vote from each band. Even if a single band bins two documents into same bucket, we deem them eligible for test of similarity, which is comparing the original shingle sets of the two documents.

Cost of computation of LSH:

Supposing our corpus has N documents, number of hash functions is K and band size is b, we end up doing O(N\*b) comparisons. Since b is O(K), thus the total complexity of computing LSH is O(N\*K).

1. DISTANCE MEASURES

By default, our implementation uses Jaccard Similarity of signatures as the primary distance measure to measure similarity between two documents. We have also included implementation of Hamming and Cosine distance measures in the project so that the results can be compared with each other. In the tests we performed, Cosine distance metric seemed to give best results followed by Jaccard similarity and Hamming distance. We also compare the results with the true similarity measure, which is the Jaccard similarity between the shingle vectors of the documents.

1. PRESENTING RESULTS

We ask the user for a particular document ID and return m most similar documents to it, along with the calculated Jaccard similarities between the document signatures and the true Jaccard similarities between the shingle pairs.

We also provide user with options to test the results with other distance measures such as Hamming distance and Cosine distance.