Birla Institute of Technology and Science-Pilani, Hyderabad Campus

First Semester 2018-2019

Information Retrieval (CS F469)

Design Document

Assignment 2

Locality Sensitive Hashing

Rohith Saranga (2017A7PS0034H)
Kasuba Badri Vishal (2017A7PS0270H)
Rikil Gajarla (2017A7PS0202H)
JSNS Rahul (2017A7PS0262H)

Under the guidance of
Dr. Aruna Malapati

Abstract:

This project aims to build a plagiarism detection software using a technique called Locality Sensitive Hashing. In this technique, similar items are hashed in to same buckets. Items that end up in the same bucket have high probability of being similar. This algorithm involves three steps.

1. Shingling
2. Minhashing
3. Locality Sensitive Hashing

Any substring of length k is defined as a k-shingle. The text in all documents is used to form k-shingles. Generally, a shingle size of around 9 is chosen for long research articles. An incidence matrix is built with these shingles and Minhashing is applied on it to get a signature matrix. Minhashing compresses the document vectors to give signatures that have lesser number of rows. In the lsh part, this signature matrix is divided into bands with a certain number of rows in each band. Each band is hashed into buckets so that similar documents have more probability of ending up in same bucket. Now documents in same bucket can be checked using appropriate similarity metric to find similarity.

Architecture:

Programming Language: python

Dataset: https://ir.shef.ac.uk/cloughie/resources/plagiarism_corpus.html

The dataset is a set of 100 documents of which 95 documents are short answers given by 19 participants to 5 questions related to Computer Science. These answers were copy pasted from given Wikipedia articles by the 19 participants. The other 5 documents are original answers.

Some of the libraries used are pandas, numpy, pickle.

Data Structures used are:

1. pandas.DataFrame for storing incidence matrix and signature matrix
2. python list for storing similar files to be returned to the user
3. python dictionary for buckets
4. python set for finding jaccard similarity

Modules:

The project has been divided into the following modules.

1. shingling
2. minhashing
3. lsh
4. statistics
5. main

Folder structure:

```
└── Locality-Sensitive-Hashing
    ├── lsh.py
    ├── main.py
    ├── minhashing.py
    ├── shingling.py
    └── statistics.py
```

Go to this link for full documentation of all modules and functions used.
docs_build\html\index.html

Working Description:

Pre-processing:

The shingling module takes the text documents and performs pre-processing like case normalization, removal of \n, \r and other unimportant characters. Then shingles from each document are created and added to the incidence matrix. The incidence matrix is a dataframe with a shape of no of shingles x no of documents. It has 1 when a shingle is present in a document and 0 if not at a respective position of the dataframe.

Minhashing technique:

h hash random hash functions are generated. Hashes[k] is the kth hash function. Each hash function will return the value as $(\boldsymbol{ax}+\boldsymbol{b})\%\boldsymbol{c}$ where a and b are >0 and c is equal to number of shingles.

Signature matrix of shape (no hash functions x no of documents) is initialized to infinity.

For each shingle (row) in incidence matrix, if a document has 1 in that row then the corresponding column in signature matrix is filled with hash values of the indices only if they are less than the values already present. In this way number of rows is decreased from number of shingles to h (no of hash functions). The reason this is done is because

The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets.

Locality Sensitive Hashing:

The signature matrix generated by minhashing is divided into b bands with r rows in each band. If m is the number of hash functions used for signature matrix.

$$m = br$$

A hash function is used to hash each band. While hashing a particular band, parts of each document are hashed into a set of buckets. For different bands different set of buckets are used. Each set of buckets is a dictionary having hash values as the keys and list of document id's as values. So, documents with exactly same signature in a band end up in the same bucket of that band. These documents are considered as candidate pairs and similarity between them is calculated using an appropriate metric.

Results:

incidence matrix: 81 sec (1.5 min approx)

k: 4

size: 11640 x 100

signature matrix: 1636 sec (27 min approx)

hash functions: 50

size: 50 x 100

signature matrix: 2471 sec (41 min approx)

hash functions: 100

size: 100 x 100

Output:

```
Enter file id: 99
Enter threshold: .3
Given file: corpus\orig_taske.txt
Output:
corpus\g4pC_taske.txt    0.565377532228361
corpus\g0pB_taske.txt    0.3624085537422622
corpus\g4pE_taske.txt    0.19554110543427775
corpus\g1pD_taskb.txt    0.14316892186817984
```

```
corpus\g1pA_taskd.txt   0.13278388278388278
corpus\g1pB_taskb.txt   0.12838170191834727
corpus\g4pE_taskb.txt   0.12265446224256293
corpus\g1pB_taskd.txt   0.1160846040334481
corpus\g3pB_taskd.txt   0.08470353761833582

Precision: 0.2222222222222222
Recall: 0.2222222222222222


Enter file id: 85
Enter threshold: .3
Given file: corpus\g4pD_taska.txt
Output:
corpus\g2pE_taska.txt   0.346875
corpus\g1pD_taska.txt   0.30541141586360265
corpus\g4pD_taskc.txt   0.15516062884483936
corpus\g4pD_taskb.txt   0.14521679284239505
corpus\g0pB_taske.txt   0.1386410432395333
corpus\orig_taskb.txt   0.1363840287124271
corpus\g0pA_taskd.txt   0.12889210716871832
corpus\g2pC_taskc.txt   0.12337662337662338

Precision: 0.25
Recall: 0.4
```