Data mining ID 2222

Short report on homework 1

Finding similar datasets: Textually similar documents

Sofie Schnitzer and Adam Åström

November 11, 2024

Introduction

The programming language chosen for the assignment is Python, which is generally a good alternative for data driven programming. The main reasons for the choice in this case is python's various libraries with functionality, such as numpy and hashlib used in this assignment, and since we are working in different operating systems it was good as well.

We followed the steps in the assignment to find Jaccard similarity between documents.

Basic implementation

The first step was to create shingles of the text in the documents, which was done by selecting a value for K, where K is the substring length for each shingle. We used the K value 5.

```
    # function creates shingles of the data in the document

    def create_shingles(self, document) :
        shingles = set()
        # loop document extracting shingles of length k
        for i in range(len(document) - self.k + 1) :
            shingle = document[i:i + self.k]
            hashed_shingle = self.hash_shingle(shingle) # call hash function to
hash each shingle
            shingles.add(hashed_shingle)
        return shingles
```

Seen above is the source code for creating shingles (in the Shingling class) from one document taken in as a parameter. The shingles from the documents are returned as a set.

Once we have created shingles of a document, and stored these as a set, we have one problem. A set of shingles for a document is much bigger than the document itself and takes up much storage, so we need a better way of storing shingles. The solution is MinHashing sets of shingles to signatures for each set. A minhash signature of a set is a representation of the set (but much smaller) that gives us a comparable "fingerprint" for the document.

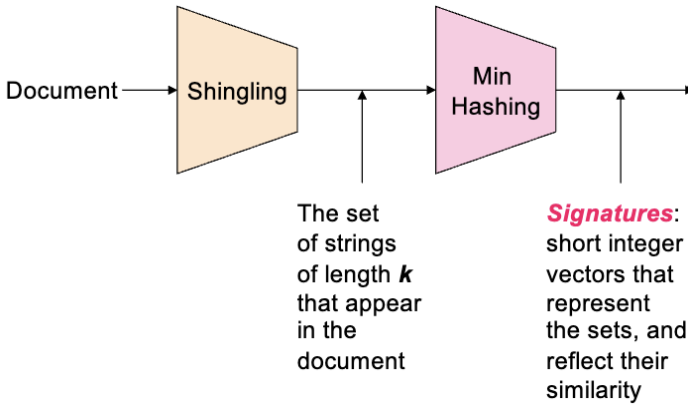
```
   def minhash_signature(self, shingle_set):

       shingles = np.array(list(shingle_set))

       hash_matrix = (self.hash_functions[0] * shingles[:, None] +
self.hash_functions[1]) % self.max_shingle_id

       signature = hash_matrix.min(axis=0)

       return signature.tolist()
```

The code above shows how we minhash (in MinHash class) the shingle sets to smaller signatures.

The comparison function takes in two signatures, check that they have the same length, it then checks each position in the signature against the same position in the other signature. For each match, 1 is added to the counter. The nr of matches is then divided by the length of the signature(s), which gives us the similarity.

Ex: `result = 4 / 6 = 0.6667 (if signature length is 6 and matches are 4)`

This far, we have implemented the steps below by a Shingling class and a CompareSets class (which would be enough to just compare the sets of shingles, if we were to add a 'main' or 'test' functionality as well). By adding a MinHashing class and CompareSignature class we can instead of comparing sets of shingles, compare the signatures to find out document similarity.



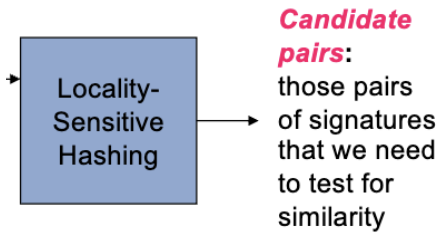
LSH implementation (for bonus points)

The following steps outline how to implement a class for **Locality Sensitive Hashing (LSH)**.

The LSH class would:

1. Take a collection of MinHash signatures and a similarity threshold t.
2. Divide each MinHash signature into bands.
3. Hash each band and place documents with the same hash value for any band into the same bucket.
4. Identify and return candidate document pairs where the MinHash signatures are similar by at least the fraction t.

This approach minimizes the number of similarity checks by only comparing signatures within the same bucket. The result is a set of candidate pairs—document pairs considered "similar enough" to potentially be a match. In this context, “similar enough” means that the Jaccard similarity between these document pairs is likely to meet or exceed the threshold t.



Building and Running

To run the project you need to have Python 3.7 or later and pip installed.

In order to build and run the program you need to firstly scrape the articles to populate the datasets. This is done in the news_scraper.py file.

Set the url for the article and the output names for the file (https://www.bbc.com/news/articles).

Run the script: python news_scraper.py

Make sure the scraped files are placed in a Data folder in the same directory as the scripts.

To run the main program: python tester.py

Command-line parameters:

Parameter	**Description**	**Default Value**
k	Length of shingles	5
num_hashes	Number of hash functions for MinHashing	125
num_bands	Number of bands for LSH	25
threshold	Similarity threshold for LSH	0.5

The output in the terminal of running the program can be explained by the following points:

1. Loaded Documents: The first 200 characters of each document
2. Execution Times: Time taken for MinHasing and LSH
3. LSH Candidate Pairs: Pairs of documents with similarity above the specified threshold
4. True Jaccard Similarities: Similarity scored for all document pairs

Results

Setup with unique documents and standard values:

- K = 5
- Num_hashes = 125
- Num_bands = 25
- Threshold = 0.8

Terminal Output:

```
MinHashing Execution Time: 1.1040 seconds
LSH Execution Time: 0.0017 seconds
Total Execution Time: 1.1057 seconds

LSH Candidate Pairs with Similarity >= 0.8:
No candidate pairs found.

Calculating True Jaccard Similarities for All Pairs...
Document 1 and Document 2 - True Jaccard Similarity: 0.0541
Document 1 and Document 3 - True Jaccard Similarity: 0.0684
Document 1 and Document 4 - True Jaccard Similarity: 0.1211
Document 1 and Document 5 - True Jaccard Similarity: 0.0517
```

More documents were compared; excluded in the screenshot due to space constraints in this report

Analysis:

Documents are very dissimilar, total execution time is moderate, and true jaccard similarities are low.

Setup with unique documents and the modified k values:

- K = 2
- Num_hashes = 125
- Num_bands = 25
- Threshold = 0.8

Terminal Output:

```
MinHashing Execution Time: 0.1578 seconds
LSH Execution Time: 0.0000 seconds
Total Execution Time: 0.1578 seconds

LSH Candidate Pairs with Similarity >= 0.8:
No candidate pairs found.

Calculating True Jaccard Similarities for All Pairs...
Document 1 and Document 2 - True Jaccard Similarity: 0.4513
Document 1 and Document 3 - True Jaccard Similarity: 0.4469
Document 1 and Document 4 - True Jaccard Similarity: 0.5636
Document 1 and Document 5 - True Jaccard Similarity: 0.4454
```

More documents were compared; excluded in the screenshot due to space constraints in this report

Analysis:

Documents are very dissimilar, total execution time is much faster, and true jaccard similarities are moderate.

Setup with unique documents and decreased threshold values:

- K = 5
- Num_hashes = 125
- Num_bands = 25
- Threshold = 0.3

Terminal Output:

```
MinHashing Execution Time: 1.1046 seconds
LSH Execution Time: 0.0000 seconds
Total Execution Time: 1.1046 seconds

LSH Candidate Pairs with Similarity >= 0.3:
No candidate pairs found.

Calculating True Jaccard Similarities for All Pairs...
Document 1 and Document 2 - True Jaccard Similarity: 0.0541
Document 1 and Document 3 - True Jaccard Similarity: 0.0684
Document 1 and Document 4 - True Jaccard Similarity: 0.1211
Document 1 and Document 5 - True Jaccard Similarity: 0.0517
```

More documents were compared; excluded in the screenshot due to space constraints in this report

Analysis:

Documents are very dissimilar, total execution time is moderate, and true jaccard similarities are low.

Setup with very similar documents and standard values:

- K = 5
- Num_hashes = 125
- Num_bands = 25
- Threshold = 0.8

Terminal Output:

```
MinHashing Execution Time: 1.1767 seconds
LSH Execution Time: 0.0000 seconds
Total Execution Time: 1.1767 seconds

LSH Candidate Pairs with Similarity >= 0.8:
Document 3 and Document 4 - Similarity: 1.0000
Document 1 and Document 3 - Similarity: 1.0000
Document 1 and Document 4 - Similarity: 1.0000

Calculating True Jaccard Similarities for All Pairs...
Document 1 and Document 2 - True Jaccard Similarity: 0.0541
Document 1 and Document 3 - True Jaccard Similarity: 1.0000
Document 1 and Document 4 - True Jaccard Similarity: 1.0000
Document 1 and Document 5 - True Jaccard Similarity: 0.0517
```

More documents were compared; excluded in the screenshot due to space constraints in this report

Analysis:

Documents are very similar, total execution time is moderate, and true Jaccard similarities are mostly low.