Homework 1 - ID2222 Data Mining

Group 71 - Nour Alga & Karim Haque

10th November 2025

1 Goal of this Lab and Data used

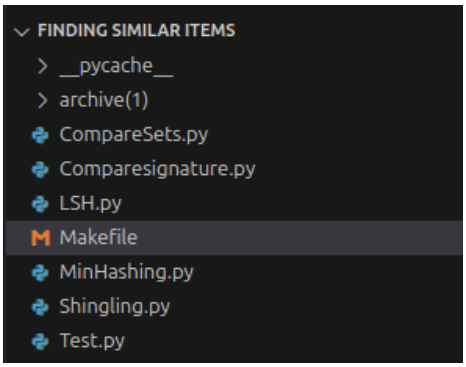
The goal of this lab is to implement and evaluate methods for identifying textually similar documents using Jaccard similarity through shingling, MinHashing, and Locality-Sensitive Hashing (LSH). Our experiments were carried out using Python. We used the textual dataset from `https://www.kaggle.com/datasets/asad1m9a9h6mood/news-articles`, consisting of 2,581 news articles published between 2015 and 2017:

- Article: Text containing the news article and the place of publication.
- Heading: Text containing the heading of the news article.
- Date
- NewsType: Type of article, i.e., business or sports.

2 How to Run the Code

The zip file contains one folder :

- Finding similar items: This includes the Python files for the different classes and a Test file and a MAkefile for visualising and plotting the figures in the report
- Firstly upload the dataset from `https://www.kaggle.com/datasets/asad1m9a9h6mood/news-articles` add a new folder called archive(1) where the is the Dataset Articles.cvs it should look like this



or just upload the dataset as a zip file and change the file path on the Test.py

```
# to reduce the randomness of shinging ..
random.seed(42)

# Loading data that's on the file archive(1) if you change the fold
file_path = "archive(1)/Articles.csv"
df = pd.read_csv(file_path, encoding='latin1')
```

- Open a terminal in the project folder. Run the following command: **Make run**

3 Solution

In order to compute the document similarity, five classes have been implemented:

3.1 Shingling

By shingling, you can break down each document into overlapping sequences of k characters or words. Each document is then represented as a set of these shingles:

- **Character shingles:** The method `create_shingles_char` generates all contiguous substrings of length k (e.g., for T = "hello" and $k = 3$, the shingles are "hel", "ell", "llo").
- **Word shingles:** The method `create_shingles_word` generates contiguous sequences of k words from T (e.g., for T = "i am right here" and $k = 2$, the shingles are "i am", "am right", "right here").

- **Hashing shingles:** Each shingle $s \in S$ is converted to a fixed-size integer for efficient comparison (e.g., "hel" $\rightarrow$ 1234567890, "ell" $\rightarrow$ 987654321, "llo" $\rightarrow$ 192837465):

$$h(s) = \text{hash}(s) \bmod 2^{32}.$$

3.2 CompareSets

Calculates the Jaccard similarity between two sets of shingles. This is done by dividing the intersection by the union of the collections:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

3.3 MinHashing

In the `MinHashing` class, compact signatures for sets of hashed shingles are generated to approximate Jaccard similarity. Initialised with the method `num_hashes`, it creates random coefficients (a_i, b_i) for each function. With the help of the method `compute_signature`, a signature is generated for a set of hashed shingles. For each hash function, it computes the hash values of all shingles and records the smallest one, capturing the "minimum hash" for that function. After repeating this for all hash functions, the result is a vector of minimum values, the MinHash signature, that can be used to estimate the Jaccard similarity between sets without comparing them directly:

$$h_i(s) = (a_i \cdot s + b_i) \bmod p,$$

and the MinHash signature for set S is

$$\text{signature}[i] = \min_{s \in S} h_i(s).$$

3.4 CompareSignatures

The `CompareSignatures` class compares two signature vectors sig_A and sig_B by computing the fraction of positions where they match, which approximates the Jaccard similarity:

$$\text{sim}(A, B) \approx \frac{|\text{sig}_A \cap \text{sig}_B|}{|\text{sig}_A|}.$$

This approach reduces computation compared to directly comparing full shingle sets.

Test of Jaccard similarity on our Data

To visualize the Jaccard similarity heatmap, we first processed our dataset by keeping only the Article column and working with the first ten business documents. We chose articles of the same type (business) to increase the chances of

finding textual similarities between them. However, with k = 1, we noticed that even though all documents belong to the same category, some of them still have a Jaccard similarity of 0, as they discuss different business topics or concepts.

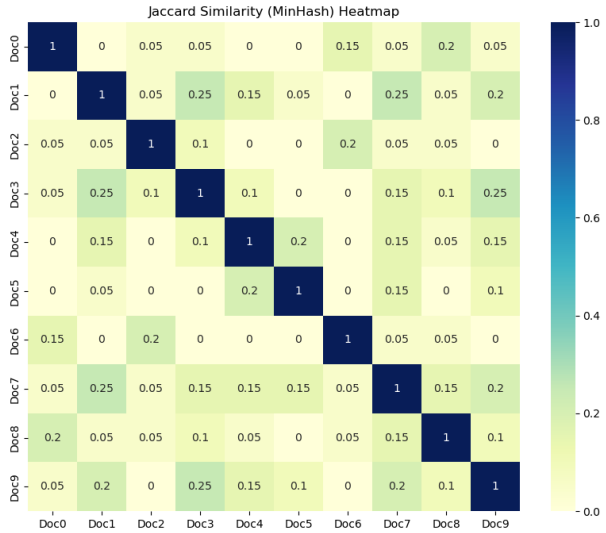
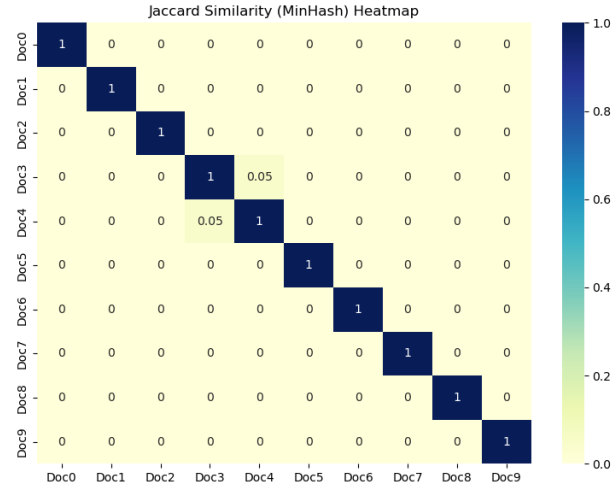


Figure 1: Jaccard similarity using k = 1

Importance of K

Let's look at the Jaccard similarity map for a larger value of k: 4



The parameter K in shingling plays a crucial role in determining how docu-

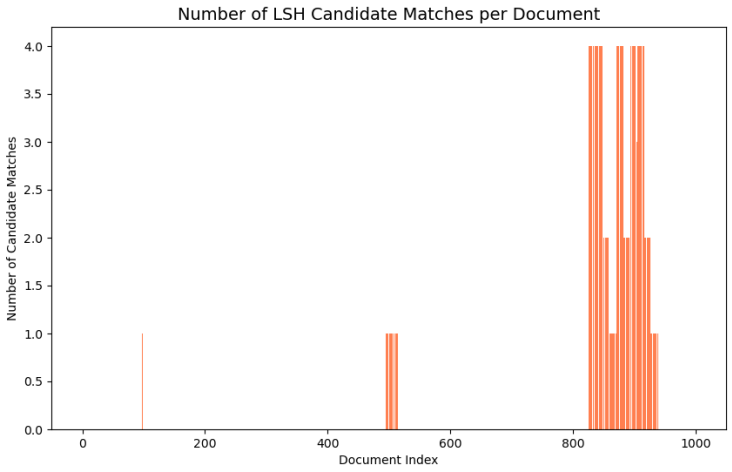
ments are compared. When we increased the value to K = 4 for our set of ten documents, we observed that almost all documents had a Jaccard similarity of 0. This happens because with larger K, the generated shingles (sequences of words) become more specific and appear less frequently across different documents — reducing the overlap between them. For short documents, using a small K (such as 1 or 2) is recommended, since it captures more shared patterns and increases the chance of finding similarities. In contrast, for longer documents, a larger K (like 3 to 5) can be more effective, as it avoids counting very common short sequences that do not contribute much to meaningful similarity.

3.5 LSH

In our implementation, the LSH class is designed to identify candidate pairs of similar documents based on their MinHash signatures. The function `split bandes()` divides each signature into a fixed number of equal parts called bands. The `get_candidates()` function then hashes each band and groups together documents that share the same hash value within at least one band, marking them as candidate pairs. The `filter_candidates()` function refines these candidates by computing the actual similarity between their signatures and keeping only those above a chosen threshold. Finally, the run() function combines all these steps, providing the complete LSH process from banding to final filtering.

Test

To visualise our Candidate pairs that are similars we use 1000 documents and choose for parameters k = 2, number of hashes: 20, number of bands = 5 we plot this figure



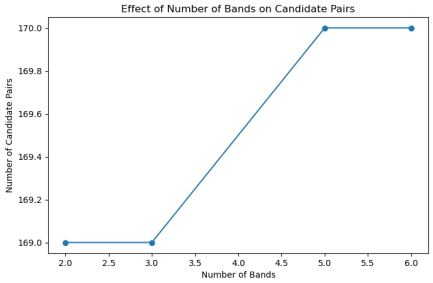
From the figure, we can see that most documents have very few or no candi-

date matches, meaning they were not grouped with any other document during the LSH process. Only a few documents have multiple candidate matches, which suggests that these texts share similar content patterns and are likely related or repetitive.

These results are influenced by the chosen parameters k = 2 (word shingles), hashes = 20, bands = 5, and a threshold of 0.8. Such settings can affect the sensitivity of LSH in detecting similarities: strict configurations might lead to fewer matches, while looser ones might group unrelated documents. This naturally leads us to question how these parameters impact our results, which will be explored in the following experiments.

Effect of number of bands

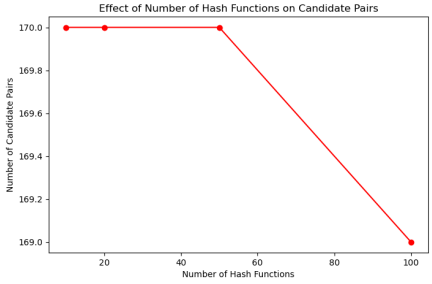
To visualize the effect of the number of bands on candidate pair detection, we fixed the number of hash functions at 20 and the threshold at 0.8. As shown in the figure, increasing the number of bands leads to more candidate pairs being identified. This occurs because with more bands, each band contains fewer rows (signature elements), making it easier for documents to match in at least one band.



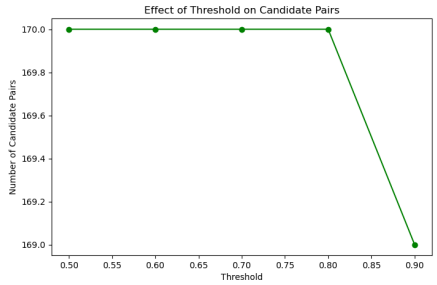
Important Note: It is critical to ensure that the number of bands does not exceed the number of hash functions. When testing with 20 bands and 20 hash functions, each band contained only 1 row, resulting in an excessive number of false positive candidates (95,588 pairs) due to random hash collisions. This demonstrates the importance of choosing appropriate parameter combinations: the number of rows per band (numhashes/numbands) should be at least 2 to 3 to maintain meaningful similarity detection. For our final analysis, we limited testing to a maximum of 10 bands to ensure reliable results.

Effect of threshold and number of hash functions

For numbandes = 5 and threshold = 0.8

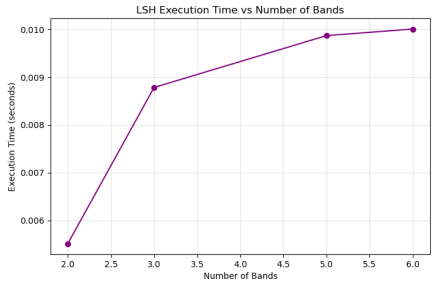


With more hash functions, the signature size grows, making it less likely for two documents to match across all rows within a band.
For number of bandes 5 and number of hash function: 20



A higher threshold means we only retain pairs with greater similarity, filtering out more candidates.

Execution Time



As the number of bands increases, execution time grows because the algorithm must iterate through more bands and process a larger number of candidate pairs generated by the increased sensitivity to matches.

Limitations

During our experiments, it was difficult to control the randomness of the plots generated by different runs. Because random seeds were not fully fixed across all components, some figures may vary slightly when the code is re-executed. However, the overall patterns and interpretations remain consistent.

4 Conclusion

This homework explores document similarity using shingling, MinHashing, and Locality-Sensitive Hashing (LSH). We observed that the shingle size k significantly affects similarity detection. Smaller k values capture broader overlaps, while larger ones naturally make matches rarer. MinHashing efficiently approximates Jaccard similarity with reduced computation, and LSH further improves scalability by focusing on likely candidate pairs. Overall, MinHashing and LSH together provide an effective and scalable approach for identifying similar documents.