ID2222 Assignment 1 Finding Similar Items

Chenyang Ding and Yilai Chen
November 11, 2024

1 Introduction and Solution

In this assignment, we implement the stages of finding textually similar documents based on Jaccard similarity using the *shingling*, *minhashing*, and *locality-sensitive hashing* (LSH) techniques and corresponding algorithms. To test and evaluate our implementation, we write a python program to uses our implementation to find similar documents among thousands of text documents.

Below, we implement example classes for the different stages of finding text-similar documents. They are `CompareSets`, `CompareSignatures`, `LSH`, `MinHashing`, `Shingling`.

1.1 CompareSets

```python
class CompareSets:
    @staticmethod
    def jaccard_similarity(set1, set2):
        intersection = set1.intersection(set2)
        union = set1.union(set2)
        return len(intersection) / len(union) if len(union) > 0 else 0
```

The process of simply calculating Jaccard similarity is to divide the intersection by the union.

1.2 CompareSignatures

```python
class CompareSignatures:
    @staticmethod
    def signature_similarity(sig1, sig2):
        match_count = sum(1 for i in range(len(sig1)) if sig1[i] == sig2[i])
        return match_count / len(sig1)
```

This method calculates the similarity between two signature vectors by counting the number of matching positions and dividing by the length of the vectors.

The result represents the fraction of matching elements, providing an approximate similarity score between the two signatures.

1.3 LSH

```python
class LSH:
    def __init__(self, bands, rows):
        self.bands = bands # bands count
        self.rows = rows # rows of each bands
    def hash_band(self, band):
        ...
    def find_similar_pairs(self, signatures, threshold):
        ...
```

An implementation class of the LSH algorithm. First, the class is initialized according to the number of bands and the number of rows in each band, followed by the method of hashing the bands and the method of finding similar document pairs.

1.4 MinHashing

```
class MinHashing:
    def __init__(self, num_hashes):
        self.num_hashes = num_hashes # num of hash functions
        self.hash_functions = [self._create_hash_function() for _ in
range(num_hashes)]
    def _create_hash_function(self):
        ...
    def generate_signature(self, hashed_shingles):
        ...
```

A MinHashing implementation class that first initializes the class by setting the number of hash functions, and then has methods for generating several hash functions and corresponding signatures.

1.5 Shingling

```
class Shingling:
    def __init__(self, k):
        self.k = k # length
    def generate_shingles(self, document):
        ...
    def hash_shingles(self, shingles):
        ...
```

A Shingling method to slice text and get the hash value

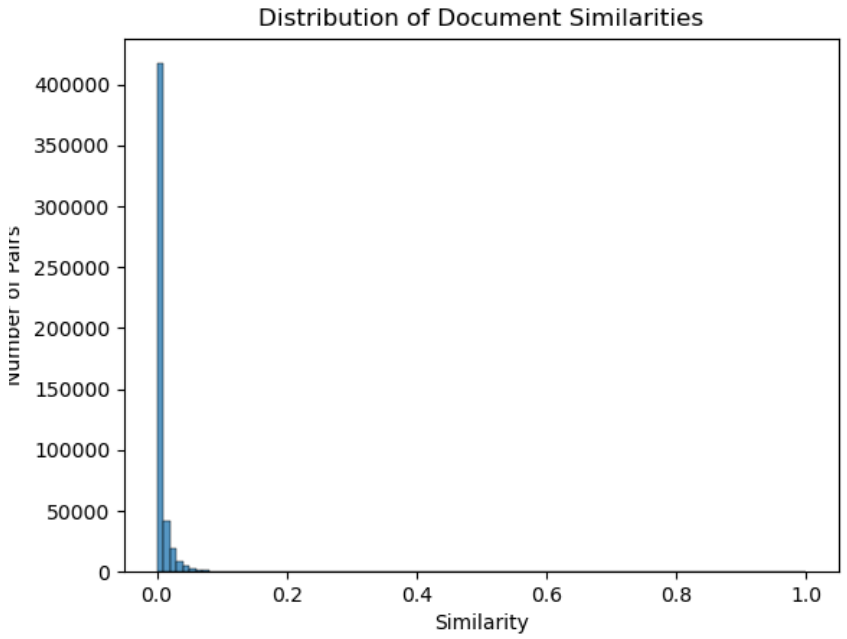
2 Experiments

The experiments were conducted on a dataset from Kaggle1. We obtained a series of news headline datasets, and we performed a similarity evaluation by comparing these headlines.

> https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset

2.1 Histogram of similarity distribution

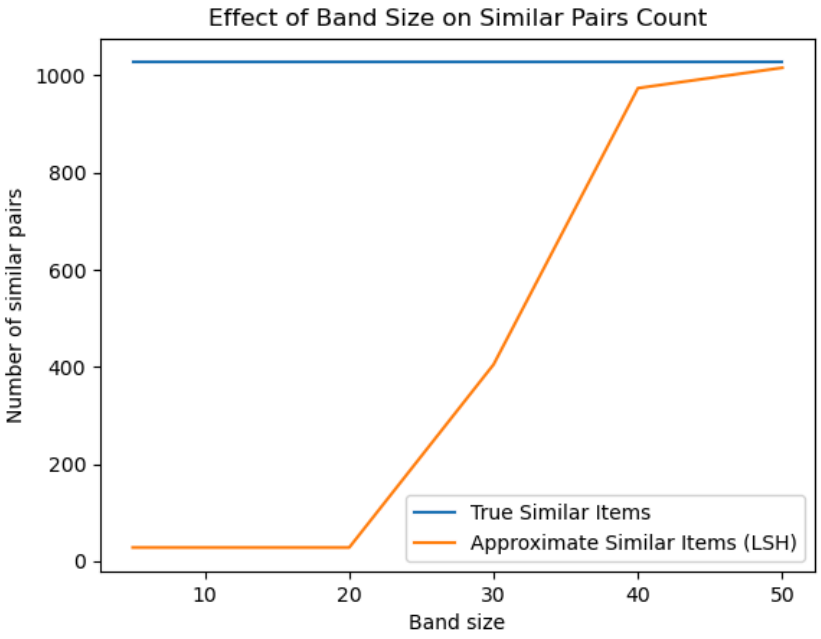
- In this experiment, **1000 documents** are randomly selected from a given document collection and **shingles(k=5)** are generated for each document.
- **Jaccard similarity** is calculated between each pair of documents.
- Similarity values of all document pairs are stored in a list and a histogram is plotted to show the distribution of similarity.



The histogram shows the distribution of document similarity. It can be seen that the similarity of most document pairs is close to zero, indicating that in this document collection, the similarity between most documents is very low.
This also indicates that there is less duplication or greater content difference between documents, so in practical applications, we may focus on a small number of document pairs with high similarity.

2.2 The impact of Band Size on the number of similar pairs

- In the second experiment, the **LSH** algorithm uses **different band sizes** to find similar document pairs.
  To implement LSH, MinHash signatures are generated for the document collection, and **100 hash** value signatures are generated for each document. Threshold = 0.1.
- **Different band sizes** are set to **5, 10, 20, 30, 40, and 50**. The number of rows in each band is set according to the formula **rows** = NUM_HASHES // bands.
- The number of similar document pairs estimated using LSH and the number of true similar document pairs calculated by the exact Jaccard similarity are calculated for each band size.



By increasing the number of bands, the LSH method can improve the approximation effect of the real similar pairs, thereby reducing missed detections. However, too many bands may increase the false alarm rate (i.e., mistaking non-similar pairs for similar pairs).
The trend in the figure shows that after the number of bands is appropriately increased, the LSH method can more effectively approach the number of real similar pairs, thus achieving a balance that ensures both the recall rate and the false alarm rate.