Report 1: Homework Report Template

Georg Zsolnai, Oscar Reina

November 10, 2024

1 Introduction

This report describes the implementation of a system used for identifying textually similar documents using Jaccard similarity. The solution is made by using several techniques:

- Shingling
- MinHashing
- Locality-Sensitive Hashing (LSH)

First, documents are broken into k-shingles, and their Jaccard similarity is approximated using MinHash signatures. The CompareSets class calculates the similarity between the sets of shingles, and CompareSignatures estimates the similarity between the MinHash signatures. Finally, LSH is used to find candidate document pairs based on their similarities. This system is tested on a small corpus of 5 to 10 documents to evaluate its performance, via using the similarity threshold for identifying similar documents.

2 Dataset

The dataset used in this project has been extracted from UC Irvine Machine Learning Repository and boasts a collection of SMS messages labeled as either "ham" (non-spam) or "spam." The messages come in different formats, offering contexts such as normal conversations or offers. The "ham" label represents messages that entail everyday communication and "spam" labeled messages involve advertisements or offers. As an example, a personal messages could look like "Go until jurong point, crazy.." (ham) and an offer could look like "WINNER!! As a valued network customer you have been selected to receive a £900 prize reward!" (spam). These labeled messages allow us to test Jaccard similarity and the MinHashing technique in identifying similar documents. The dataset can be accessed through this url `https://archive.ics.uci.edu/dataset/228/sms+spam+collection`.

Once downloaded, the file should be turned into *.txt* format and placed within the */data* folder.

Importantly, for the sections below we used a small subset of the data to make the outputs not only legible but also easily explainable for the brevity of the report.

3 Results

3.1 Shingling

The objective of the shingling stage was to convert each document into a set of hashed k-shingles. For this dataset we used a shingle size of one word. Generally, each document showed to contain hundred of unique shingles, but it depended on the length of the document.

This shingling process can be shown in both Figures 1 and 2 for un-hashed and hashed shingles respectively, with their corresponding execution time.

```
CurrTime: 2024-11-09 14:30:57.672928, Starting SimItems.....
Time elapsed after shingling (in seconds): 0:00:00.000540
{'£500!', 'with', 'details', 'card', 'to', 'claim', 'your', 'worth', 'Reply', 'today.', 'prize', 'won', 'a', "You've", 'now!', 'Claim', 'free', 'gift'}
{'iPhone', 'Win', 'online', 'by', 'out!', "Don't", 'a', 'now', 'our', 'exclusive', 'miss', 'contest.', 'free', 'entering', '12'}
{'she', 'said', 'She', 'late', 'will', 'here', 'should', 'be', 'soon.', 'but'}
{'Can', 'you', 'send', 'urgent.', 'by', '5', 'the', 'me', 'report', 'today?', 'PM', "It's"}
{'u', 'lar...', 'wif', 'Joking', 'oni...', 'Ok'}
{'Ok', 'Joking', 'lar...'}
```

Figure 1: Un-hashed shingles

```
CurrTime: 2024-11-09 14:32:01.414140, Starting SimItems.....
Time elapsed after shingling (in seconds): 0:00:00.000568
{-8873858523169545691, 42572093782032694 75, 8084439476985186116, -2573591727440662226, -8332343559045234575, -9131361048946477860}
{-9131361048946477860, -2573591727440662226, 8084439476985186116}
{-3883247587123173 11, -6179359418267203740, -8216243566359789785, 4750755508836653123, 3197989230922105892, 1432951981606540601, -2682558228901777850, 5871176951187521000, -2485611073480303
984, -5921426806262204367, 1334768291038335794, -1627988433712004171, -5835512296701270056, 8654645464919503731, 9118587871963644085, 4644417185603328019, -4422636686680868514, -142584048855
79108450}{-7000824736666010711, -7937436625041838102, 4210123047040592904, -6062966693469466 2, -2622373649492792275, -7061716237950470543, -6770510092543054063, -2649405227494141838, 141622
3721975626770, -8874249117850517859}
{-8818174098744739421, -5100386580024209 25, 3197989230922105892, 7565499614543703 3, 7904374652313883242, -7226568936257476010, -2402108446404228396, -2199361415933925036, -4510888157261634377, 7921288849574974903, -2437202116764580739, 8760191346887689850}
```

Figure 2: hashed shingles

3.1.1 MinHashing

The objective of the MinHashing stage was to approximate the Jaccard Similarity between sets by creating more compact matrix signatures. This were created using minimum hash values so that we can later on quickly estimate the similarity without comparing entire sets directly.

We tested the results depending on the number of hash functions on Jaccard similarity using four sentences:

- Sentence 0: "Ok lar... Joking wif u oni..." (ham)

- Sentence 1: "Ok Joking lar..." (ham)
- Sentence 15: "We should meet up for coffee sometime next week. Let me know when you're free." (ham)
- Sentence 16: "sometime coffee next free week. up when Let know meet" (ham)

As shown in Figure 3, having 10 hash functions makes the results less reliable outputting higher values due to the presence of false positives. This is because it captures fewer characteristics of the documents, resulting in higher changes of documents with different content receiving similar signatures. For example:

```
CurrTime: 2024-11-09 14:11:19.249736, Starting SimItems.....
Time elapsed after shingling (in seconds): 0:00:00.000633
Time elapsed after MinHash (in seconds): 0:00:00.166064
Threshold is set to: 0.4472135954999579
Comparing documents:  0  and  1 . Similarity of  0.7
Comparing documents:  15  and  16 . Similarity of  0.8
```

Figure 3: MinHash with 10 hash functions

However, as seen in Figure 4, when we have more hash functions, the result gives better approximations of Jaccard similarity (showing a more precise measurement) . However, it is also worth noting the increase in the computational cost based on the time elapsed to complete the execution.

```
CurrTime: 2024-11-09 14:12:22.409002, Starting SimItems.....
Time elapsed after shingling (in seconds): 0:00:00.000675
Time elapsed after MinHash (in seconds): 0:00:00.167732
Threshold is set to: 0.4472135954999579
Comparing documents:  0  and  1 . Similarity of  0.55
Comparing documents:  15  and  16 . Similarity of  0.53
```

Figure 4: MinHash with 100 hash functions

3.1.2 LSH

LSH's main objective is to reduce the complexity of doing documents' pairwise comparisons by grouping similar documents together into buckets. These buckets identify candidate document pairs that are likely to be similar. This improves the computational times needed to directly compare similarities between all documents.

In this assignment, LHS is used in the MinHash signature matrix by partitioning the generted signatures into several bands. The bands, which

are divided into further rows, represent a candidate pair of documents. If two documents happen to share the same hash value in at least one of the bands, then they are considered potential candidates. This is later verified by checking their Jaccard Similarity of the entire signature and checking whether they surpass the minimum similarity threshold t. This threshold is set according to the following formula; (n = number of bands; r = rows per band).

$$t = \left(\frac{1}{n}\right)^{\frac{1}{r}}$$

In order to experiment the results according to the number of bands and rows, we varied their numbers and observed how these variations affected the similarity results.

As shown in Figure 5, if we increase the number of bands (e.g 50) and decrease the number of rows per band (e.g 2), the number of final candidate pairs and the number of similarities identified increases. This happens because (1) the similarity threshold is going to be smaller and (2) having more bands allows for a better partition of the signature matrix, improving the possibilities of a match being found across the bands. This additionally has the effect that we have less data to compare with thus making it easier to compare with more bands.

```
CurrTime: 2024-11-09 15:05:13.345821, Starting SimItems.....
Time elapsed after shingling (in seconds): 0:00:00.000520
Time elapsed after MinHash (in seconds): 0:00:00.169523
Threshold is set to: 0.1414213562373095
Comparing documents:  0  and  1 . Similarity of  0.55
Comparing documents:  2  and  14 . Similarity of  0.05
Comparing documents:  13  and  14 . Similarity of  0.08
Comparing documents:  4  and  13 . Similarity of  0.07
Comparing documents:  6  and  14 . Similarity of  0.08
Comparing documents:  15  and  16 . Similarity of  0.53
Comparing documents:  12  and  15 . Similarity of  0.06
Candidates: {(0, 1), (15, 16)}
Time elapsed after all functions (in seconds): 0:00:00.170336
```

Figure 5: LSH candidates execution with 50 bands and 2 rows per band

Nevertheless, if we decrease the number of bands and increase the number of rows per band, this would lead to the opposite effect. The similarity threshold will increase (which will make it harder for similarities scores to overcome) and it will be harder to find a match between two similar items. This is because, there are fewer changes to detect similar hash signatures. This can be further seen in Figure 6

```
CurrTime: 2024-11-09 15:08:07.549267, Starting SimItems.....
Time elapsed after shingling (in seconds): 0:00:00.000476
Time elapsed after MinHash (in seconds): 0:00:00.176994
Threshold is set to: 0.4472135954999579
Comparing documents:  0  and  1 . Similarity of  0.55
Comparing documents:  15  and  16 . Similarity of  0.53
Candidates: {(0, 1), (15, 16)}
Time elapsed after all functions (in seconds): 0:00:00.177471
```

Figure 6: LSH candidates execution with 25 bands and 4 rows per band

4 Setup and execution

To build and run this project, please follow these steps:

1. Ensure you have Python installed.
2. Make sure you are running a virtual environment to be able to install pip packages (or install pip packages globally at your own risk)
3. Install PySpark by running:

```
pip install pyspark
```

4. Download the dataset provided in Section 2 and place it within the */data* directory that you have to create in the root directory of the repository.
5. Once the dataset is downloaded and dependencies are installed, simply execute the following command:

```
python main.py
```