# Information Retrieval Project - Plagiarism Detection

Jesper van Stee - 20205019
Stefan Chiru - 20202124

30 January 2020

## 1 Introduction

In order to detect plagiarism in news articles we created a solution based on an existing framework for MinHash and Locality Sensitive Hashing, called `datasketch`[1]. In this report we will present our implementation of getting our ground truth data and Locality Sensitivity Hashing, and the results of some experiments of both.

## 2 Data

Two data sets were provided containing a set of news articles, `news_articles_small.csv` and `news_articles_large.csv`. Both data sets have two columns each: `News_ID`, `article`. Every row is thus an article and an ID. The small dataset is used for the experiments, the large dataset is used to gather final results presented in `results.csv`.

## 3 Method

In this section we discuss the implementation of both the ground truth and the Locality Sensitive Hashing.

### 3.1 Ground truth

To generate the ground truth we need to calculate the Jaccard Similarity of each pair of articles. In order to do this we write two functions to help us. One to calculate the Jaccard Similarity, and one to generate n-gram shingles. The formula for Jaccard Similarity can be rewritten as

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

This is implemented in python by dividing the length of the intersection part by the length of the union part. The intersection part was done by taking the AND (&) operator on the set of a and set of b, then taking the length of the list of that. The union part was done by taking the length adding the length of and and length of b, then subtracting the intersection part. Finally divide the intersection and union parts.

First we load the data set. We take the entire article string in every row, and remove all special characters using a regular expression, then bring it to lowercase and create a list splitting at spaces. Next create n-gram shingles of every article, this is done by using a simple function taking the entire set and combing all the consecutive combinations of words in the list.

Next create all possible pairs by taking the product of the dataset with itself, then only keeping the combinations that are not a product of an article with the same article.

Finally keep a list of all similarities of all pairs, then create a histogram of this list with a bin size of 0.005.

## 3.2 Locality Sensitive Hashing used the

The Locality Sensitive Hashing method uses the `datasketch` framework. We start by reading the data set again, then do the same preprocessing by removing special characters with a regular expression. Next we create a list of MinHashes. This is done by a simple for loop over the data that creates MinHashes of every article with a set number of permutations. For every article this MinHash is appended to the list of MinHashes.

Then use the `MinHashLSH` function with a set threshold and the same number of permutations used for the MinHashes. In this part there are two options, we provide the number of band and rows, or let the framework calculate the optimal.

The optimal value for the number of bands b and rows r is calculated by two nested for loops. The outer loop goes over every value between 1 and the number of permutations. The inner loop over every value between 1 and the maximum number of rows, which is the total number of permutations divided by the value for b in the current loop. In the inner loop calculate the false positive and false negative probability using the threshold, b and r. Then calculate the error which is $FP \cdot W_{FP} + FN \cdot W_{FN}$ with $W_{FP}$ and $W_{FN}$ being some weight given in the constructor to prioritize minimizing false positives or false negatives, by default 0.5 for both.

Next we insert every MinHash with the docID and the MinHash of the doc. Then loop over the list of MinHashes, then query the LSH with every MinHash in the list to find all duplicates for all articles in the dataset. If there is more than 1 result, since when querying the actual document will always be above the similarity threshold, we increment a counter and add the duplicates to a dataframe. Finally we print the results.

## 4 Results

First we generate the ground truth, calculated as described in the method section. We plot the histogram for 1-gram, 2-gram and 3-gram shingles, with bins of size 0.005, or half a percentage. For the 1-gram shingles the resulting histogram is plotted in Figure 1. We observe that there are no documents above the 0.8 threshold. The maximum similarity lies around 0.6.

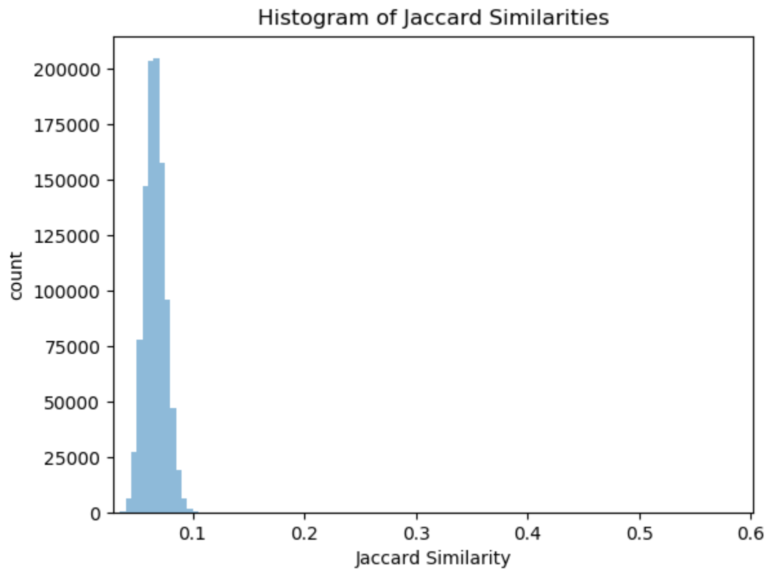

Figure 1: 1-gram shingle ground truth histogram

The 2-gram shingle histogram is plotted in Figure 2. This time there are values all the way up to 1.0, so zoomed in on the section 0.8 to 1.0 can be seen in Figure 3. We see that there are 20 counts of queries with a Jaccard Similarity of more than 0.8, hence there are 10 plagiarized articles in the data set, as there will be two counts for every pair of documents.

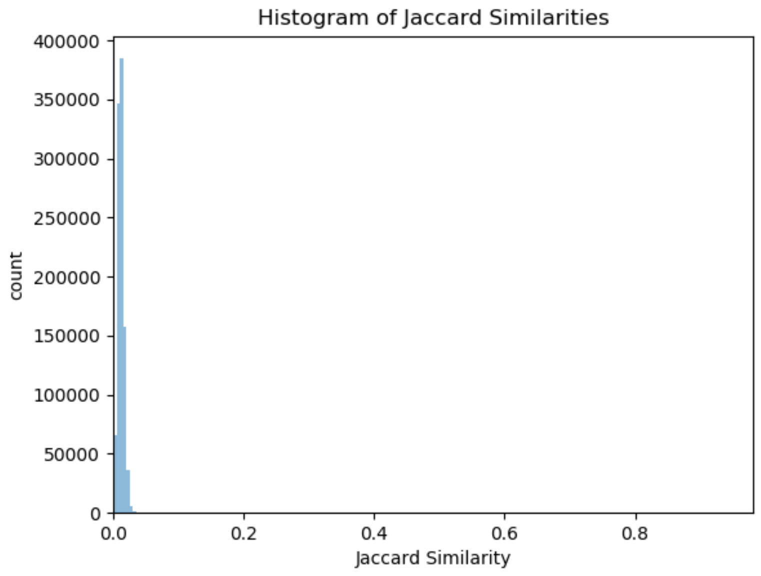

Figure 2: 2-gram shingle ground truth histogram

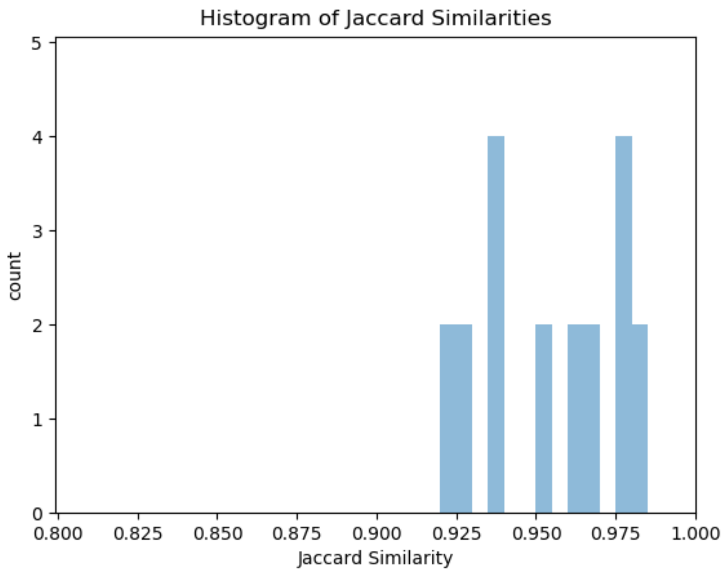

Figure 3: zoomed in 2-gram shingle ground truth histogram

Lastly for 3-gram shingles plotted in Figure 4 and zoomed in on the 0.8 to 1.0 range in Figure 5. We observe a difference in the major peaks near the 0.0 mark, they shift even more to the left, so documents are deemed even less similar. Again, there are 20 counts of queries with a Jaccard Similarity of more than 0.8.

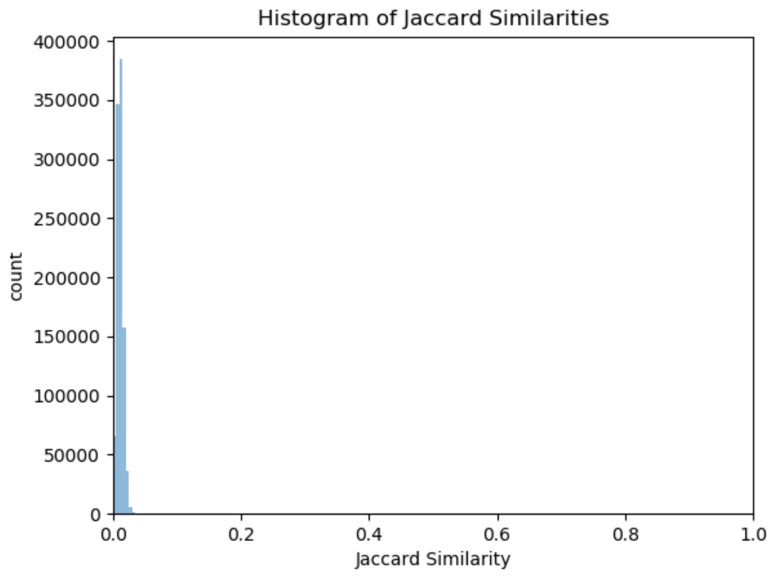

Figure 4: 3-gram shingle ground truth histogram

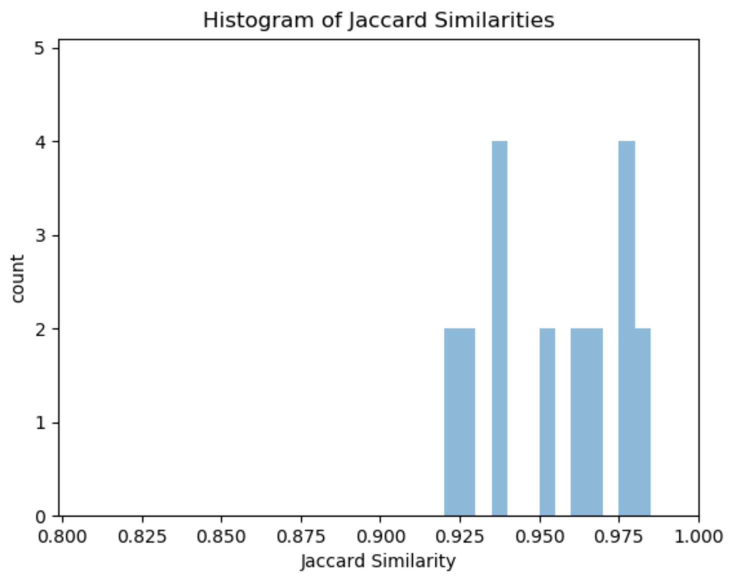

Figure 5: zoomed in 3-gram shingle ground truth histogram

In the first experiment we analyse the effect of different sizes of the signature length, the number of bands and number of rows. These results can be found in Table 1. The first number is the calculated optimal number of rows and bands for the given similarity threshold. How this is calculated is discussed in the method section. In the case of this experiment the threshold was 0.8.

| **Signatures** | **Bands** | **Rows** | **Count** |
|---|---|---|---|
| 256 | **17** | **15** | **20** |
| | 256 | 1 | 999000 |
| | 128 | 2 | 591388 |
| | 64 | 4 | 2424 |
| | 32 | 8 | 20 |
| | 16 | 16 | 20 |
| | 8 | 32 | 20 |
| | 4 | 64 | 20 |
| | 2 | 128 | 18 |
| | 1 | 256 | 6 |
| 128 | **9** | **13** | **20** |
| | 128 | 1 | 998864 |
| | 64 | 2 | 231728 |
| | 32 | 4 | 544 |
| | 16 | 8 | 20 |
| | 8 | 16 | 20 |
| | 4 | 32 | 20 |
| | 2 | 64 | 18 |
| | 1 | 128 | 14 |
| 64 | **5** | **11** | **20** |
| | 64 | 1 | 986122 |
| | 32 | 2 | 102776 |
| | 16 | 3 | 154 |
| | 8 | 8 | 20 |
| | 4 | 16 | 20 |
| | 2 | 32 | 20 |
| | 1 | 64 | 18 |
| 32 | **3** | **10** | **20** |
| | 32 | 1 | 927090 |
| | 16 | 2 | 67844 |
| | 8 | 4 | 106 |
| | 4 | 8 | 20 |
| | 2 | 16 | 20 |
| | 1 | 32 | 18 |
| 16 | **2** | **8** | **20** |
| | 16 | 1 | 744670 |
| | 8 | 2 | 32638 |
| | 4 | 4 | 66 |
| | **2** | **8** | **20** |
| | 1 | 16 | 20 |
| 8 | **1** | **7** | **20** |
| | 8 | 1 | 580538 |
| | 4 | 2 | 18394 |
| | 2 | 4 | 44 |
| | 1 | 8 | 20 |

Table 1: Results with different sizes of signature length, bands, and rows.

In the final experiment we use some different similarity thresholds, this gives us the calculated optimal bands and rows for the number of permutations and the count of plagiarized documents. We use 256 number of permutations in this experiment. We see that the thresholds between 0.6 and 0.95 result in the correct amount of near duplicates found, given the ground truth. Every given threshold has different number of bands and rows that are calculated as optimal.

| **Threshold** | **Bands** | **Rows** | **Count** |
|---|---|---|---|
| 0.99 | 1 | 166 | 10 |
| 0.95 | 5 | 51 | 20 |
| 0.9 | 9 | 28 | 20 |
| 0.8 | 17 | 15 | 20 |
| 0.7 | 25 | 10 | 20 |
| 0.6 | 32 | 8 | 20 |
| 0.5 | 42 | 6 | 38 |
| 0.4 | 51 | 5 | 472 |

Table 2: Different thresholds resulting in the optimal number of bands and rows, and the count of pairs above threshold.

### 4.1 Large dataset

The results of the large data set can be found in the `results.csv` file. We find, using a similarity threshold of 0.8 and 256 number of permutations, a total of 164 duplicates. Since there are 2 duplicates for each plagiarized article, divide this number by two thus 82 plagiarized documents in the large data set. Comparing these articles manually does indeed show that these articles are very similar.

## 5 Conclusion

We successfully find near duplicate articles in the data sets by implementing a MinHash Locality Sensitive Hashing algorithm using the `datasketch` framework. Using different number of signatures, bands, and rows shows that the number of duplicates found vastly differs. The framework provides a way to calculate the optimal value for the number of bands and rows given a similarity threshold, using these values returns us the correct number of near duplicate articles given our ground truth.

## References

[1] Eric Zhu. datasketch: Big Data Looks Small `http://ekzhu.com/datasketch` [Retrieved January 2021]