ID2222 – Data Mining

Assignment 1 - Finding Similar Items: Textually Similar Documents

Course Coordinator:
Vladimar Vlassov & Sarunas Girdzijauskas

Teaching assistants:
Edward Tjörnhammer
Kambiz Ghoorchian
Mohamed Gabr

Group 2

Óttar Guðmundsson

Örn Arnar Karlsson

2018-11-11

Description

Comparison of data is essential in multiple problems. One of these problems is text comparison, to check for similarities of documents if plagiarism is suspected. This can easily be done by comparing them word for word, but will not provide good results if the plagiarised document has changed the words using few letters. However, morphing the documents into shingles and dividing the mutual shingles of both documents with all possible shingles will give us a metric, Jaccard similarity, that defines how related they are. This is a quite expensive computation and comparing all over 1.000 documents with one another will take a really long time.

In this exercise, we implement a method called MinHashing to construct signatures, that is an estimation of the similarity. selecting only signatures that are above certain threshold using Local Sensitive Hashing. Then, documents that are above this threshold are compared using the Jaccard similarity.

How to run

We wrote our solution in Scala using a Jupyter notebook with Spark Kernel. A guide to installing Jupyter Notebook, install Spark Scala and connect the kernel to Jupyter can be done using Toree (guide here)

Open the notebook and click “Run all cells” on the toolbar. This will create shingles for all documents found in the data folder, compute MinHash for all of them and print out similar documents and time taken to compute the results. Our solution is implemented using a collection of functions, properly commented.

Note: In some cases, we encountered dead kernels in the Notebook. Attached is the file assign1.scala that has all of the code in a pipeline, computing all of our calculations.

Solution

We completed all of the tasks in this assignment as well as the bonus task. In this chapter, we will explain our solution.

Shingling

For this task, we created a function that takes a document and an integer k, constructs k-shingles and returns them in a set. A set in scala makes sure that there are no duplicates. Then we create an integer hash value for each string in the set using the MD5 hashing method.

Compare Sets

Two sorted immutable sets of Integers are compared by taking their intersection size and dividing it with their union size. That value is the similarity of both sets, rounded to two decimals and printed out to the console. The distance is also compared by subtracting it from 1, which is also rounded and printed.

MinHashing

For this problem, we created two helper methods. One that creates k random hash functions for a given integer k. The hash functions follow the formula (a*x+b) mod c. Where a and b are random integers from 0 to k and c is the number of elements in the shingles set. The second helper functions then create a signature matrix for the matrix representation of the shingles set where each element is the maximum integer value.

Then we created a MinHash function that uses the algorithm given in lecture nodes to create to turn the maximum integer matrix to the signature matrix for the shingles set.

CompareSignatures

In this task, we created a triangular matrix of similarities between sets. All sets are compared for every hash function and if they have the same value in as in our signature matrix, their index in the matrix is incremented. Finally, the whole matrix is divided by the number of hash functions used creating a correlational matrix of all sets ranging from 0 to 1.

LSH (Bonus)

Here we constructed the method *arrToBandCan* that takes the signature matrix and returns a two-dimensional array where each element is a hashed string, where each string is concatted of all elements of the row for each band. That matrix is the pushed into *find_for_t* that calculates the appearance of candidate pairs for each band. Finally, the candidate pairs appearance is then compared to a given threshold, returning all candidate pairs of signatures that agree on at least fraction t of their components.

Results

We noticed that for our MinHash implementation, it seemed that the estimation was way higher than the actual similarity. For 16 hotel reviews found on the internet, we noticed that the estimation was sometimes around 35% while their actual similarity was closer to 20%.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_illinois_ch	usa_illinois_ch	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci	usa_new york ci
usa_new york ci	-	0.43 vs 0.201	0.39 vs 0.198	0.43 vs 0.193	0.37 vs 0.182	0.24 vs 0.15	0.4 vs 0.198	0.38 vs 0.197	0.06 vs 0.03	0.07 vs 0.049	0.38 vs 0.184	0.4 vs 0.192	0.36 vs 0.192	0.1 vs 0.033	0.34 vs 0.192	0.28 vs 0.185
usa_new york ci	0.43 vs 0.201	-	0.46 vs 0.191	0.38 vs 0.207	0.38 vs 0.195	0.21 vs 0.142	0.35 vs 0.204	0.38 vs 0.203	0.03 vs 0.027	0.04 vs 0.046	0.39 vs 0.191	0.37 vs 0.187	0.39 vs 0.2	0.1 vs 0.028	0.31 vs 0.198	0.31 vs 0.183
usa_new york ci	0.39 vs 0.198	0.46 vs 0.191	-	0.38 vs 0.204	0.41 vs 0.195	0.23 vs 0.148	0.4 vs 0.212	0.35 vs 0.201	0.03 vs 0.027	0.06 vs 0.046	0.4 vs 0.194	0.34 vs 0.191	0.39 vs 0.206	0.1 vs 0.031	0.31 vs 0.196	0.31 vs 0.185
usa_new york ci	0.43 vs 0.193	0.38 vs 0.207	0.38 vs 0.204	-	0.45 vs 0.238	0.22 vs 0.132	0.39 vs 0.239	0.41 vs 0.225	0.04 vs 0.02	0.05 vs 0.034	0.43 vs 0.227	0.31 vs 0.193	0.41 vs 0.239	0.13 vs 0.022	0.38 vs 0.225	0.4 vs 0.179
usa_new york ci	0.37 vs 0.182	0.38 vs 0.195	0.41 vs 0.195	0.45 vs 0.238	-	0.16 vs 0.119	0.43 vs 0.239	0.44 vs 0.226	0.03 vs 0.016	0.04 vs 0.03	0.53 vs 0.238	0.37 vs 0.186	0.47 vs 0.241	0.11 vs 0.018	0.45 vs 0.228	0.31 vs 0.166
usa_new york ci	0.24 vs 0.15	0.21 vs 0.142	0.23 vs 0.148	0.22 vs 0.132	0.16 vs 0.119	-	0.21 vs 0.133	0.19 vs 0.14	0.06 vs 0.034	0.06 vs 0.059	0.19 vs 0.126	0.25 vs 0.153	0.2 vs 0.129	0.07 vs 0.038	0.2 vs 0.135	0.21 vs 0.156
usa_new york ci	0.4 vs 0.198	0.35 vs 0.204	0.4 vs 0.212	0.39 vs 0.239	0.43 vs 0.239	0.21 vs 0.133	-	0.37 vs 0.227	0.03 vs 0.021	0.05 vs 0.036	0.44 vs 0.23	0.36 vs 0.197	0.41 vs 0.241	0.12 vs 0.023	0.38 vs 0.224	0.29 vs 0.182
usa_new york ci	0.38 vs 0.197	0.38 vs 0.203	0.35 vs 0.201	0.41 vs 0.225	0.44 vs 0.226	0.19 vs 0.14	0.37 vs 0.227	-	0.03 vs 0.021	0.03 vs 0.037	0.34 vs 0.226	0.42 vs 0.201	0.36 vs 0.225	0.09 vs 0.023	0.41 vs 0.227	0.32 vs 0.188
usa_new york ci	0.06 vs 0.03	0.03 vs 0.027	0.03 vs 0.027	0.04 vs 0.02	0.03 vs 0.016	0.06 vs 0.034	0.03 vs 0.021	0.03 vs 0.021	-	0.09 vs 0.059	0.05 vs 0.018	0.04 vs 0.027	0.03 vs 0.019	0.03 vs 0.056	0.04 vs 0.02	0.05 vs 0.031
usa_illinois_ch	0.07 vs 0.049	0.04 vs 0.046	0.06 vs 0.046	0.05 vs 0.034	0.04 vs 0.03	0.06 vs 0.059	0.05 vs 0.036	0.03 vs 0.037	0.09 vs 0.059	-	0.06 vs 0.032	0.07 vs 0.047	0.02 vs 0.033	0.04 vs 0.055	0.05 vs 0.035	0.05 vs 0.051
usa_illinois_ch	0.38 vs 0.184	0.39 vs 0.191	0.4 vs 0.194	0.43 vs 0.227	0.53 vs 0.238	0.19 vs 0.126	0.44 vs 0.23	0.34 vs 0.226	0.05 vs 0.018	0.06 vs 0.032	-	0.38 vs 0.191	0.41 vs 0.228	0.1 vs 0.019	0.37 vs 0.226	0.24 vs 0.172
usa_new york ci	0.4 vs 0.192	0.37 vs 0.187	0.34 vs 0.191	0.31 vs 0.193	0.37 vs 0.186	0.25 vs 0.153	0.36 vs 0.197	0.42 vs 0.201	0.04 vs 0.027	0.07 vs 0.047	0.38 vs 0.191	-	0.39 vs 0.191	0.1 vs 0.029	0.32 vs 0.197	0.28 vs 0.191
usa_new york ci	0.36 vs 0.192	0.39 vs 0.2	0.39 vs 0.206	0.41 vs 0.239	0.47 vs 0.241	0.2 vs 0.129	0.41 vs 0.241	0.36 vs 0.225	0.03 vs 0.019	0.02 vs 0.033	0.41 vs 0.228	0.39 vs 0.191	-	0.11 vs 0.022	0.33 vs 0.221	0.33 vs 0.178
usa_new york ci	0.1 vs 0.033	0.1 vs 0.028	0.1 vs 0.031	0.13 vs 0.022	0.11 vs 0.018	0.07 vs 0.038	0.12 vs 0.023	0.09 vs 0.023	0.03 vs 0.056	0.04 vs 0.055	0.1 vs 0.019	0.1 vs 0.029	0.11 vs 0.022	-	0.1 vs 0.022	0.09 vs 0.034
usa_new york ci	0.34 vs 0.192	0.31 vs 0.198	0.31 vs 0.196	0.38 vs 0.225	0.45 vs 0.228	0.2 vs 0.135	0.38 vs 0.224	0.41 vs 0.227	0.04 vs 0.02	0.05 vs 0.035	0.37 vs 0.226	0.32 vs 0.197	0.33 vs 0.221	0.1 vs 0.022	-	0.32 vs 0.18
usa_new york ci	0.28 vs 0.185	0.31 vs 0.183	0.31 vs 0.185	0.4 vs 0.179	0.31 vs 0.166	0.21 vs 0.156	0.29 vs 0.182	0.32 vs 0.188	0.05 vs 0.031	0.05 vs 0.051	0.24 vs 0.172	0.28 vs 0.191	0.33 vs 0.178	0.09 vs 0.034	0.32 vs 0.18	-

Figure 1: Corpus of documents related to hotel reviews, comparing Jaccard similarity vs estimated similarity using MinHash

We did find another dataset online of newspaper articles that had a plagiarized version in them as well. Noticeably, our code managed to spot all of them.

```
Similarities found between ./data/data_7.txt and ./data/data_0.txt
Estimated similarity is 94.0%
Actual Jaccard similarity is 98.0%
Distance is 0.02

Similarities found between ./data/data_40.txt and ./data/data_5.txt
Estimated similarity is 100.0%
Actual Jaccard similarity is 98.3%
Distance is 0.017

Similarities found between ./data/data_4.txt and ./data/data_31.txt
Estimated similarity is 100.0%
Actual Jaccard similarity is 98.1%
Distance is 0.019

Similarities found between ./data/data_32.txt and ./data/data_1.txt
Estimated similarity is 100.0%
Actual Jaccard similarity is 98.3%
Distance is 0.017

Similarities found between ./data/data_18.txt and ./data/data_6.txt
Estimated similarity is 99.0%
Actual Jaccard similarity is 97.8%
Distance is 0.022
```

Figure 2: All plagiarized documents found

Furthermore, we implemented the extra function to compare all sets using MinHash and using the Jaccard Similarity. It's worth mentioning that comparing 16 documents is twice as fast using the MinHash and nearly three times faster with 100 documents (39060 ms vs 99921 ms). This clearly demonstrates the usefulness of the function.