Advances in Data Mining: assignment 2

Sjoerd Hermes & Miltiades Violaris

November 19, 2020

1 Introduction

The aim of this report is to present three different similarity measures which can be used to find similar users, based on the movies they watched. Instead of using a brute force method where all possible pairs of users are computed over all movies they watched, a technique called locality-sensitive hashing (LSH) was implemented, as the aim is to find a substantial amount of similar users within a relatively short time (30 minutes). Whilst using a brute force approach might be possible on a small data set, on a data set containing 103703 users and 17770 movies this is not feasible. To assess whether a user is similar to another user, three different similarity measures are proposed: the Jaccard similarity, the cosine similarity, and the discrete cosine similarity. As similar users are obtained in different ways from the three different similarity measures, the various model parameters needed to be tuned in order to optimize the results. To this end, the report contains both a theoretical overview of the different similarity measures, the signature matrix, the locality sensitivity hashing method, as well as the actual application on the Netflix data, where the parameter choices made are elaborated on, including the effect of increasing/decreasing a parameter value, and on the results obtained.

Netflix data

The data used for this report is a reduced form of the so-called "Netflix data". This data set contains the ratings that clients of Netflix (called users) gave to various movies. The data contains 65225506 ratings $r \in \{1, 2, 3, 4, 5\}$ from a total of 103703 unique users u and 17770 unique movies m. This data set was transformed into a sparse matrix with the users representing the columns and the movies representing the rows, such that the matrix was 17770 (m) $\times$103703 (u) dimensional. The entries of this sparse matrix were then either 0 for non-rated movies (which make up to majority of the entries), 1 for movies that a user rated in the case of the Jaccard and discrete cosine similarity measures, or the actual rating $r \in \{1, 2, 3, 4, 5\}$ in the case of the cosine similarity. This results in two different sparse matrices, where the Jaccard and discrete cosine similarity measures share the same sparse matrix, but the cosine similarity operates on a different sparse matrix. Aside from the difference in rating values, the sparse matrices were formatted to either a Compressed Sparse Column (CSC) format or a Compressed Sparse Row (CSR) format. A sparse matrix that is of CSC format is faster in column slicing, whereas a sparse matrix of CSR format is faster in row slicing. Whether the sparse matrices used to compute the different similarity measures were of the CSC or CSR form will be elaborated on in section 2.2, as the computation of the signature matrix has a large impact on how to sparse matrix should be formatted in order to achieve optimal results.

Remarks

Before moving on to the actual report, a few remarks should be made. First off all, the computers on which the various methods were tested both had 16GB of RAM, which influences the execution time. As the computer on which the code will be run for the assessment contains 8GB of RAM, the execution time probably differs. This fact influenced the choice for the optimal combination of parameters, as a combination of parameters that results in an acceptable time on the test computers with 16GB of RAM, might not result in an acceptable time on a computer with 8GB of RAM. Additionally, to ensure stability of the results, the combinations of parameters were tested over 5 runs using 5 different seeds: 2020, 20, 100, 1 and 999. The results were averaged, and the choice for the best combination of parameters was based on these mean results. This was done as both the number of similar users found and the computing time differ significantly between different seeds, especially for the cases with a relatively short signature length (< 100). Therefore, multiple runs were executed to somewhat take into account this variability. Ideally, a much larger sample should be taken to get more reliable results, but the final choices for the parameter combinations seemed to lead to stable results. Finally, the code contained in the main.py file is different from the code used during the parameter testing. This is because the code in the main.py file appends pairs of similar users one by one to a .txt file, whereas the code used in the testing phase wrote all pairs of similar users at once, at the end of the code. There are no other differences between the files, and therefore, the chosen combination of parameters should also give good results when running the main.py file.

2 Theory and application

2.1 Similarity Methods

As stated above in the introduction, three different similarity measures were applied in the Netflix data case-study. In this section, all three methods will be explained, including the value (called threshold t here) for which two users are deemed similar.

The first of the three similarity measures is the Jaccard similarity, which is defined as:

$$\text{Sim}(u_i, u_j) = \frac{|u_i \cap u_j|}{|u_i \cup u_j|}, \tag{1}$$

where u_i and u_j are some users such that $i \neq j$. Using the Jaccard similarity, two users u_i, u_j are deemed simlar if $\text{Sim}(u_i, u_j) > 0.5$. The difference between the cosine and discrete cosine similarities is not in their similarity measure. In fact, the similarity between two users using the cosine and discrete cosine methods are identical. First, the cosine distance is computed as follows:

$$\text{Distance}(u_i, u_j) = \theta = \arccos\left(\frac{u_i \cdot u_j}{|u_i||u_j|}\right), \tag{2}$$

with this distance measure θ, the cosine similarity can be computed as:

$$\text{Sim}(u_i, u_j) = 1 - \frac{\theta}{180}. \tag{3}$$

Rather than using the cosine distance as the similarity measure, we first transformed this value before using it, such that the resulting quantity is the cosine similarity. The reasoning behind this is that the cosine distance takes values in $[-1, 1]$, whereas the cosine similarity takes values in $[0, 1]$, which is desirable for a similarity measure. In this case, two users u_i, u_j are deemed similar if $\text{Sim}(u_i, u_j) > 0.73$. This holds for both the cosine and discrete cosine case.

What is immediately striking is that the threshold for the Jaccard similarity seems a lot less stringent than that of the cosine and discrete cosine similarity measures. However, two users tend to have a higher value for the cosine similarity than they do for the Jaccard similarity. Therefore, in order to avoid ending up with tens of thousands of similar pairs of users for the cosine and discrete cosine similarity measures, the threshold of cosine and discrete cosine similarity measures was set higher compared to that of the Jaccard similarity.

2.2 Signature matrix

With the three similarity measures defined in section 2.1, the next concept can be explained: the signature matrix. A signature matrix S is a $h \times u$ dimensional matrix which is an essential part of the LSH algorithm. The matrix contains in some sense condensed information of the original sparse matrix, which can then be used to find similar users. The Jaccard similarity measure uses a different method to derive the signature matrix from the cosine and discrete cosine similarity measure. In this section, both methods will be explained.

To derive the signature matrix S for the Jaccard similarity, a technique called Minhashing was used. Minhashing permutes the rows of the sparse matrix, in order to subsequently determine what the first non-zero entry is for each user. The index number of the row of the first non-zero entry is saved for each user. Subsequently, when each user has a value for their first non-zero entry obtained from the permuted rows of the sparse matrix, the resulting vector becomes the first row of the signature matrix S. This process is repeated h times, where h is the signature length and is user-specified, resulting in a signature matrix that is $h \times u$ dimensional. Note that for each new row of S, the sparse matrix is permuted again, otherwise all entries for a column of S would be identical over the h rows. In order to derive S, h iterations over the rows of the sparse matrix needed to be done. As `Scipy` allows for the sparse matrix to be of CSR format, which drastically reduces computation time when iterating over rows of a sparse matrix, the sparse matrix for the Jaccard similarity was of CSR format.

Rather than using Minhashing to derive the signature matrix S, in the case of the cosine and discrete cosine similarities, the sparse matrix was converted to CSC format right away, rather than first using a CSR format to derive the signature matrix. Since signature matrix S used for the Jaccard similarity was derived by iterating over the rows of the sparse matrix, using a CSR format saved a great deal of time. However, the signature matrix for the cosine and discrete cosine similarities are not derived from iterating over the rows of the sparse matrix, but instead by a simple matrix multiplication. To understand why matrix multiplication instead of row-based iteration could be used, it is important to note that instead of Minhashing, a method called "random projection" was used in order to derive the signature matrix. This works as follows: h random hyperplanes of length m, where $m =$ the amount of movies in the sparse matrix (17770), were generated from a Gaussian distribution, such that a matrix V with random hyperplanes of $m \times h$ dimensionality was constructed. The way in which the signature matrix is then constructed is by taking a user vector x, and taking a random hyperplane v (say the first hyperplane), then if $v \cdot x > 0$, the value of that user for the first row of the signature matrix is 1, and if $v \cdot x < 0$, the value of that user for the first row of the signature matrix is -1. This process should be repeated for all users, over all h hyperplanes. The result is a $h \times u$ signature matrix S, which can easily be computed by matrix multiplication, as

$$V^T_{h\times m} \cdot Sparse_{m\times u} = S_{h\times u}. \tag{4}$$

However, the values of S are not yet in $\{-1, 1\}$, therefore the sign function was applied: $sgn(S)$, resulting in the desired signature matrix.

2.3 Locality-sensitive hashing

As computing the similarity of two users, over all movies that the users watched, for each possible pair of users is too time consuming, a technique called locality-sensitive hashing (LSH) was applied. LSH, as was applied in this report, takes a signature matrix S, consisting of h rows and u columns, where h is the chosen signature length and u are the unique users in the data, 103703 in this case, and divides this into b bands, consisting of r rows, such that $b \times r = h$. Once the partitioning of S has been done, each user in a band in S will be hashed to some bucket. Two users that have the exact same rows in one band will be hashed to the same bucket, otherwise they will be hashed to different buckets. It is possible, that some users will randomly be hashed to the same bucket, even though not all rows of the two users share the same value. In order to make the chance of this happening very small, a large number of buckets should be chosen. The users that reside in the same bucket are called "candidate pairs", as no real measure of similarity has been computed yet. Ideally for each band the number of buckets should be taken to be equal to all possible combinations of values that a user can take within a band. But in this way the number of buckets will be a huge number which is computationally infeasible. Instead, for each band a dictionary was created with all the unique combinations of values of the users and then hashed the users to their corresponding bucket. Additionally, for more efficiency the buckets containing at least one user were saved in order to avoid looping over single user buckets.

2.4 Advanced selection

Once the candidate pairs are gathered, the calculation of the similarity between the users is up next. Even though the number of candidate pairs is significantly lower than the number of all possible potential user pairs $\binom{103703}{2} = 5377104253$, this task is still considered very time consuming since the similarity has to be computed from the sparse matrix between 17770 movies. One possible solution to this is choosing a large number of rows per band so the number of candidate pairs is reduced even more, but with the drawback of having many false negative pairs, meaning many similar pairs of users are lost during the LSH algorithm. Instead, an alternative direction was chosen so that more similar pairs of users would be found in the 30 minute time limit. Selecting a small number of rows with many candidate pairs and reducing the false negatives, the similarity of a pair of users was calculated from the signature matrix instead of the sparse matrix, a process which is many times faster since the signature matrix has only a fraction of the 17770 rows the sparse matrix has. The similarity from the signature matrix is only an approximation and was used to further 'filter' the similar pairs of users. The similarity between two users on the sparse matrix was only calculated if the similarity from the signature matrix for those two users exceeded the so-called signature threshold t'. For each similarity measure, t' was chosen close to the desired threshold t with the purpose of selecting only pairs that have an actual similarity close to, or higher than, t. In many of the most promising combinations of parameters, $t' \neq t$. This was because in some cases, t' underestimates the actual similarity of a pair of users, such that reducing this value would lead to potentially more candidate pairs. More on this can be found in section 3. Finally all pairs exceeding both thresholds were added to the final text file. Overall, this resulting in having to measure the similarity over the sparse matrix for only the truly similar users, thereby greatly reducing the computation time required.

3 Parameter properties

Before moving on to choosing the most optimal combination of parameters based on the results, some of the properties of the parameters ought to be discussed.

Signature threshold *t'*

The first parameter to be discussed is the signature threshold t'. This threshold differs from the actual threshold used to compute the similarities. Instead, the signature threshold is only used to compute the similarity of two users using the signature matrix S, whilst the threshold used to compute the similarity of two users on the sparse data does not deviate from the values specified in section 2.1. The reasoning behind changing this parameter is to obtain more signature pairs, and as a result more actual pairs. This occurs in the following way: by lowering the signature threshold t', it is easier for two users to be assessed as "similar" than by using a more stringent threshold. This will result in more signature pairs. Note that, in general, the similarity of two users is overestimated on the signature matrix when compared to their actual similarity, such that more signature pairs than actual pairs are obtained, even for the same threshold. However, there are also cases where the similarity of two users is underestimated over the signature matrix. In such cases, having a lower signature threshold, would result in "capturing" some of those pairs which have an actual similarity larger than t, but which would not be captured if $t' = t$. One downside of lowering the signature threshold is that most of the additional signature pairs obtained do not satisfy the similarity criterion on the sparse matrix, and as such, some time is lost on checking the similarity of additional unsatisfactory pairs, which would not be obtained by maintaining the same signature threshold as the actual threshold.

Signature length *h*

The next parameter to be discussed is the signature length h. Ceteris paribus, a longer signature length seems to lead to more pairs obtained at the cost of a longer computation time. This is not such a surprise, considering a longer signature length means that more bands are included in the signature matrix, over which similar users can be found. As such, the computation time increases, because the algorithm needs to evaluate more bands. The question is whether increasing the signature length is worth it, as the additional band(s) from a longer signature need not lead to many new similar pairs of users. Suppose that for some signature length h, we obtain b bands. If we then increase the signature length to $h + \delta$ and obtain $b + 1$ bands, the first b bands will be exactly the same as when using signature length h. Therefore, the similar users obtained from the first b bands will also be the same. If the additional $b + 1^{th}$ band will result in new similar users (so not duplicates of users that were already found in the first b bands), such that it makes up for the additional computation time by the increase of the signature length by δ, then the signature length should be increased.

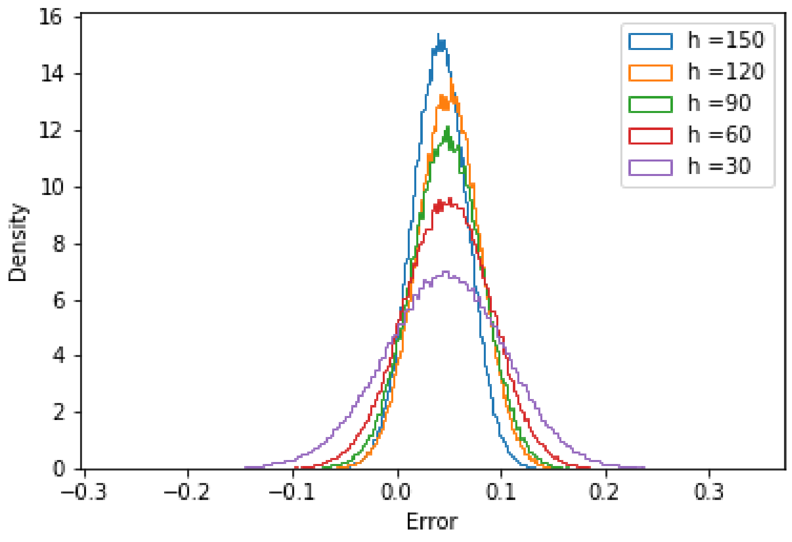

Figure 1: Difference between the signature similarity and the actual similarity for various signature lengths.

The advantage of having longer signature matrices can be observed in figure 1 above. The figure shows the density of the error between the signature similarity and the actual similarity for various signature lengths, where the similarity measure was the Cosine similarity and $error = actual\ similarity - signature\ similarity$. The longer the signature length, the higher the error density is around 0. This implies that longer signature lengths results in a smaller deviation of the signature similarities from the actual similarities, all similarities were computed from the same pairs from the Netflix data. In other words, all else equal, longer signature lengths have a better accuracy such that more candidate pairs are obtained but less approved pairs overall. This causes for an important balancing act when choosing the signature length, as can be seen in the results section.

Rows r and bands b

The last two tuning parameters in the model can be discussed simultaneously, as changing one also changes the other. As mentioned in section 2.3, for signature length h, one has b bands and r rows, such that $b \times r = h$. Therefore, an increase in r results in a decrease of b, except for cases where the number of rows in each band is not the same. From experimenting, it becomes apparent that one needs to find a balance in the number of rows used in the algorithm, as very few rows per band leads to an explosion in the computation time, whereas many rows per band leads to a situation where (almost) no similar pairs of users are obtained. Both these cases will be explained here. Suppose we divide the signature matrix up in r rows and b bands, were r is some small number, such that b becomes large. In that case many users end up in the same bucket, as users need to have the same value on only a few rows of the signature matrix. Furthermore, due to the high amount of bands, even if two users do not have the same values for all rows in one band, they might have the same values for all rows on the next band. This results in a huge amount of candidate pairs, over which two similarities need to be computed: first the similarity of two users over the values of the signature matrix corresponding to those users, in order to filter out many users that are not similar, and then as a post-processing step the similarity of two users is computed on the sparse matrix for the users that were deemed similar enough by the previous step. If the number of candidate pairs per bucket is huge, computing these similarities takes a long time. In contrast, if the number of rows per band is very large, two users need to have the exact same value on many rows per band in order to be hashed to the same bucket. As such, most buckets contain few, if any, candidate pairs which results to (almost) no actual pairs. Of course, the opposite also holds such that increasing the number of rows per band decreases the computation time.

4 Results

With these notions in mind, the LSH algorithm was run various times, using many different combinations of parameters. The time of the LSH algorithm was tracked as well during evaluations. A mixture of methods was applied during testing process in order to find good parameter combinations: theoretical knowledge from section 3 was used, in conjunction with trial and error. The initial tests were run to "test the waters": combinations of parameters that resulted in very slow execution time of the algorithm, or resulted in a small amount of similar pairs were related to the theory in section 3, and subsequently altered to allow for a different combination of parameters giving a less extreme result (so not a very long execution time, and a higher amount of similar user pairs). Because of this, even more "extreme" parameter combinations did not require testing, which reduced the amount of tests needed. This can be clarified through the following example: suppose that when testing the Jaccard similarity, the following parameter values are used: $t' = 0.5, h = 150, r = 50$ and $b = 3$, which results in no pairs of similar users. Because the theory of section 3 notes that everything else equal, increasing the number of rows per band decreases the execution time, but also decreases the amount of pairs of similar users found, there was no need to run tests where t' and h retained their value, and r increased and b as a consequence decreased. In total, approximately 50 tests of unique parameter combinations were run, with multiple random seeds ,described in section 1, to account for variability in the results. The combinations of parameters that gave the most promising results were averaged across the five runs with different random seeds and can be seen in tables 1, 2 and 3. As the tables only contain the most promising results, parameter combinations that are not included lead to unsatisfactory results: they either took too long to compute, even on computers with 16GB of ram, or the amount of similar users obtained by the algorithm was too low. The parameter combinations shown in the tables lead to good results. Furthermore, the variability between the parameter values is quite small, especially the variability for the number of rows and bands. Each of three tables will be discussed in order of appearance.

Table 1: Combination of the signature threshold, the signature length, the number of rows and bands resulting in the signature and actual numbers of pairs, including the execution time in seconds, using the Jaccard similarity with $t = 0.5$ for 5 different seeds.

t'	h	r	b	**Mean signature pairs (10^3)**	**Mean actual pairs**	**Range actual pairs**	**Mean time**
0.47	150	5	30	116.8	778	706-821	1990
0.48	150	5	30	62.4	733	666-796	1763
0.49	150	5	30	45.0	697	635-769	1697
0.50	150	5	30	25.3	622	556-707	1609
0.44	150	6	25	148.0	479	406-544	806

The results for the Jaccard similarity measure, observed in table 1, seem very promising. For all displayed combinations of parameters, the resulting amount of pairs exceeds the amount of pairs one should expect according to the assignment document. Even the lowest amount of pairs obtained (479) for the combination of $t' = 0.44, h = 150, r = 6$ and $b = 25$ is more than double the expected amount stated in the assignment document. On the other hand, the computation time does seem to be on the high end, since many of the combinations take almost half an hour to run on computers with 16GB of RAM. Note that the signature threshold t' does not deviate too much from the actual threshold t. This is because there is a limit to how much decreasing the value of the signature threshold aids in discovering new pairs. Setting the value of the signature threshold lower than 0.44 did not result in more new pairs compared to higher values of the signature threshold, but it did increase the computation time. The most promising results all have a signature length of 150. Increasing this value did (in certain cases) increase the amount of pairs obtained, but the computation time increased to such an extend that those parameter combinations were not deemed viable, as most of the choices with $h = 150$ are time consuming already. Since the computer on which the algorithm will be assessed has 8GB of RAM, and the algorithm is thus expected to take even longer, those choices were not seen as promising. On the other hand, decreasing the signature length generally decreased computation time, but the amount of pairs obtained decreased as well. When drawing comparisons, the decrease in computation time did not weigh up against the loss in obtained pairs. The same story holds for the combination of rows and bands. The optimal value was assessed to be 5 (and 6 in the case of a low value of t'), since a further reduction of the amount of rows per band caused the algorithm to be too slow, whereas a higher amount of rows per band resulted in a sub-optimal amount of pairs obtained. The exception to this is the case where the amount of rows = 6. When lowering the signature threshold to the useful minimum (0.44), with 6 rows per band, the algorithm still managed to find a decent amount of pairs whilst being efficient. However, the amount of pairs obtained using this parameter combination still results in the lowest amount of pairs obtained from all the promising parameter combinations. With all this in mind, the parameter combination that was chosen for the assessment of the algorithm was the combination: $t' = 0.48, h = 150, r = 5$ and $b = 30$. When the algorithm is run using this combination of parameters, a good amount of pairs is obtained without taking too long on a computer with 16GB of RAM, and is expected to preform reasonable on a computer with 8GB of RAM too. Even if the algorithm is still running at the 30 minute mark, plenty of pairs should have been appended to the .txt file already.

Table 2: Combination of the signature threshold, the signature length, the number of rows and bands resulting in the signature and actual numbers of pairs, including the execution time in seconds, using the Cosine similarity with $t = 0.73$ for 5 different seeds.

t'	h	r	b	**Mean signature pairs (10^3)**	**Mean actual pairs**	**Range actual pairs**	**Mean time**
0.66	112	14	8	296	260	158-315	610
0.67	140	14	10	652	356	151-660	1101
0.68	140	14	10	272	267	94-549	871
0.69	140	14	10	108	178	54-423	752
0.70	140	14	10	45	102	31-270	707
0.66	210	14	15	483	522	108-326	1374
0.67	210	14	15	160	383	207-506	1165
0.68	210	14	15	38	226	108-326	1085
0.67	130	13	10	460	327	273-464	595
0.67	143	13	11	1016	426	327-640	1585

In table 2, the results for the cosine similarity measure can be observed. What is immediately striking is that the amount of pairs obtained is significantly lower compared to the amount of pairs obtained using the Jaccard similarity. As has been stated in section 2.1, it is easier to find similar pairs of users using the cosine similarity than when using the Jaccard similarity, given that the same threshold t is used. However, the threshold for the cosine similarity is much more stringent in this case than that of the Jaccard similarity, such that only the best case of the cosine similarity $t' = 0.66, h = 210, r = 14$ and $b = 15$ resulting in an average of 522 actual pairs barely exceeds the worst result obtained from the Jaccard similarity $t' = 0.44, h = 150, r = 6$ and $b = 25$ resulting in an average of 479 actual pairs. On the other hand, the computation time is significantly lower for most parameter combinations of the cosine similarity. Various other parameter values were tried for the cosine similarity to get results closer to those of the Jaccard similarity, but no better results were achieved than those in table 2. In the case of the cosine similarity measure, the most promising results were obtained by setting the signature threshold lower than the actual threshold, as the time increase could be "afforded": the LSH algorithm with cosine similarity is optimized to such an extend that setting the lowest usable signature threshold 0.66 (as reducing this value further did not result in more pairs, but did result in an increase in the computation time) still did not cause the algorithm to take a long time. As has been stated, a longer signature length also increases computation time, but does give more similar pairs as well, this effect can be seen in parameter combinations $t' = 0.66, h = 210, r = 14$ with $b = 15$ and $t' = 0.67, h = 143, r = 13$ with $b = 11$, resulting in an average of 522 and 426 pairs of users over 5 runs respectively where the average computation time was about 1374 and 1585 seconds respectively. The choice for the final parameter combination seems difficult. However, some facts can aid this decision: the combinations consisting of 13 rows will not be used here, as in both cases, the amount of candidate pairs often exceeded 60 million in various runs. In some cases, it even went up to 100 million. Whilst the computers used to run the tests can handle such a large amount of candidate pairs, the assessment computer containing 8GB of RAM might not even get to the point of computing the actual similarity, and therefore no similar pairs would be added to the .txt file. This leaves the options containing 14 rows per band. From these options, combinations with a relatively low amount of actual similar pairs found can also be eliminated. In the end, a clear winner seems to emerge as the best combination of parameters: $t' = 0.66, h = 210, r = 14$ and $b = 15$. This combination has the highest amount of pairs whilst still being within the acceptable time limit.

Table 3: Combination of the signature threshold, the signature length, the number of rows and bands resulting in the signature and actual numbers of pairs, including the execution time in seconds, using the Discrete cosine similarity with $t = 0.73$ for 5 different seeds.

t'	h	r	b	**Mean signature pairs (10^3)**	**Mean actual pairs**	**Range actual pairs**	**Mean time**
0.67	140	14	10	359	177	70-287	1059
0.68	140	14	10	164	129	45-236	864
0.69	140	14	10	52	86	23-184	752
0.70	140	14	10	18	50	8-116	718
0.66	210	14	15	399	283	152-373	1311
0.67	210	14	15	126	201	88-287	1137
0.68	210	14	15	32	111	44-181	1078
0.67	130	13	10	482	221	103-276	305
0.67	143	13	11	921	194	112-331	1571

The discrete cosine results in table 3 are not as impressive as those of the other two similarity measures in terms of similar pairs obtained. Only three of the most promising combinations of parameters resulted in an average amount of pairs of over 200. However, when relating this to the expectations from the assignment document, the results are still within line of what can be expected. The best results were again obtained by decreasing the signature threshold, allowing for more signature pairs and thus actual pairs as well. The signature length seems to have varying results. For $t' = 0.66$ a long signature length leads to many similar pairs, in fact the highest amongst the potential parameter combinations (283 pairs), whereas a long signature length for $t' = 0.68$ leads to one of the worst results out of all the potential parameter combinations (111 pairs). Like with the cosine similarity, the options with 13 rows are not feasible if the computation power of the assessment computer is taken into account. Therefore, for the discrete cosine similarity too, the final parameter combination consisted of a "safe" option to avoid extremes of not finding any pairs due to a huge amount of candidate pairs (resulting in termination of the algorithm before writing pairs to the .txt file), or finding too little pairs. This still leaves many options, however, from the observed results in table 2, as well as the behavior of the algorithm during the testing runs, one particular combination of parameters stands out: $t' = 0.66, h = 210, r = 14$ and b.

This combination leads to the highest amount of similar pairs out of all potential combinations in table 2. Furthermore, as the average time over the five different runs is not close to half an hour, the expectation is that even on a computer with 8GB of RAM, the half hour limit will not be exceeded either. Therefore, the combination $t' = 0.66, h = 210, r = 14$ and b was chosen as the final combination of parameters when using the discrete signature similarity.

5 Conclusion

In this report, three measures of similarity were applied on the Netflix data using locality-sensitive hashing (LSH) in order to find similar users. The similarity measures are the Jaccard similarity, the cosine similarity and the discrete cosine similarity, each with their own threshold for which a pair of users if deemed similar if they exceed this threshold. The Netflix data was transformed to a sparse matrix, which was then formatted to either Compressed Sparse Column (CSC) format or Compressed Sparse Row (CSR) format in order to optimize finding similar users. The reasoning behind the application of LSH is to avoid looping over all combinations of 103703 users with the 17770 movies in the Netflix data set. By using a signature matrix, either created through Minhashing in the case of the Jaccard similarity, or through random projections in the case of the cosine and discrete cosine similarities, LSH sends users to buckets by dividing the signature matrix into rows and bands. Users that reside in a bucket are so-called "candidate" pairs, over which the similarity is computed through one of the aforementioned measures. This is done over the values of their signature matrix. Then, if two users exceed a threshold, they are saved, only to be checked again for similarity in a post-processing procedure. However, this time the similarities are computed over the actual entries in the sparse matrix, rather than the signature matrix. The pairs that exceed the threshold for the corresponding similarity measure are saved, as these pairs satisfy the similarity criterion. To find the greatest amount of similar users within 30 minutes per similarity measure, the model parameters t', h, r and b, denoting the signature threshold, the signature length, the rows and the bands respectively, needed to be tuned. Finding the "optimal" combination of parameters was done through various tests with all kinds of parameter combinations. In the end, for each similarity measure, the following parameter combinations were assessed as giving the best results, while keeping in mind the difference in computing power between the computers that the algorithm was tested on, and the computer used for the assessment of the algorithm:

- Jaccard similarity: $t' = 0.48, h = 150, r = 5$ and $b = 30$
- Cosine similarity: $t' = 0.66, h = 210, r = 14$ and $b = 15$
- Discrete cosine similarity: $t' = 0.66, h = 210, r = 14$ and $b = 15$

Using the above parameter combinations, for their respective thresholds, running the algorithm with the Jaccard similarity resulted in more similar pairs of users than with the cosine and discrete cosine similarities. However, using the Jaccard similarity as the similarity measure also took longer than the other methods. Different parameter values were applied to the cosine and discrete cosine methods to get results more similar to that of the Jaccard similarity, but to no avail.

In the end, the LSH method provides a useful way to get hundreds (depending on the similarity measure and threshold) of pairs of similar users in less than half an hour.