VRIJE
UNIVERSITEIT
BRUSSEL

NETFLIX CHALLENGE

Information Retrieval and Data Mining

Students:
Robin De Haes - 0547560
Mathias Mallinus - 0556721

Academic year 2021-2022

Sciences and Bio-Engineering Sciences: Computer Science

1 Loading the Dataset

To gain further insight, we first give an overview of the dataset composition in Table 1. There are 480 189 customers, called users from here on out, and 17 770 movies. This would mean there are 8 532 958 530 possible user-movie combinations (or ratings) while there are only 100 480 507 known ratings. This means we have a matrix density of only $\frac{8532958530-10048050}{8532958530} = 0.012$ or around 1%. Therefore, we created a sparse matrix to store entries in as it is more memory-preserving and scalable.

	Minimum	**Maximum**	**# Distinct Entries**
Movie ID	1	17770	17770
User ID	1	2649429	480189
Rating	1	5	100480507

Table 1: Overview about data entries in the Netflix challenge dataset.

Our parsing pipeline is shown in Figure 1. We tried to make this parsing process quite general, which means some steps might look redundant but have been included to allow easier modifiability and scalability if necessary. First, we read in the text files line per line to prevent any possible issues with reading in files that would be too large to keep in memory as a whole. If such a line would contain a movie ID, we store this movie ID as the one we will now be parsing ratings for. Conversely, if a line does not represent a movie ID, we split the line and extract both the user and rating from it. We then store the current movie ID, user ID and rating on the same index in three separate lists that we subsequently output. From these three lists two Scipy sparse matrices are made. One matrix of the form $USERS \times MOVIES$ and one matrix of the form $MOVIES \times USERS$. Each non-zero entry in this matrix indicates the rating that a user gave for that particular movie. Zero entries indicate absence of ratings.

There are three remarks that should be made here. The first one is that since both the movie IDs and user IDs are integers and we are working with a sparse matrix, we could create the matrices directly using these IDs as indices. To ensure a sparse matrix of the right size is made and more easily allow IDs that are not integers, we chose to explicitly map each ID to an incrementing index and used these indices to generate our sparse matrix. We then output two sparse matrices and two dictionaries that can be used for bidirectional mapping. The movie dictionary allows mapping from a movie ID to its corresponding index in the sparse matrices and vice versa. The user dictionary works analogously. Secondly, at this point we could have outputted the lists of movies, users and ratings separately but extracting them from the sparse matrices can happen relatively quickly so we chose to do this when needed. Giving a sparse matrix as argument to functions is less involved than having to give a sparse matrix and/or three lists. The additional time that is needed for this extraction seems negligible. Lastly, we would like to note that our pipeline actually also supports subsetting the original dataset by randomly including ratings with a certain probability. This was mainly used for development purposes and is therefore not described in this report.

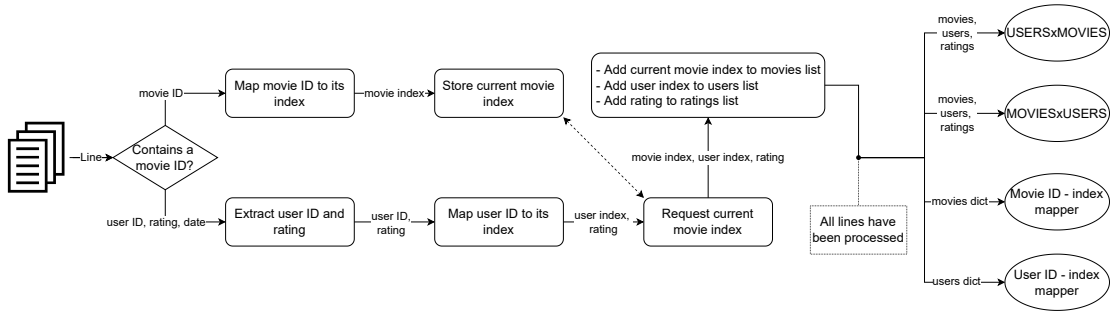

Figure 1: Schematic overview of the parsing pipeline used for the Netflix challenge dataset.

2 DIMSUM

Singular Value Decomposition (SVD) is a technique often used to factorize a matrix A into three smaller matrices U, Σ, V such that $A = U\Sigma V^T$. However, SVD requires the computation of A^TA which is infeasible to perform as standard matrix multiplication in case A is too large. DIMSUM (Zadeh and Carlsson, 2013) allows a distributed approximation of A^TA. DIMSUM's complexity is independent of the number of matrix rows which makes it feasible for tall skinny matrices. Harming the singular values too much is prevented by devising a good importance sampling scheme based on cosine similarities. These similarities can be exploited since highly popular columns should be sampled less as a small sample of them can already hold enough information. In this report, we will focus on the impact of the parameter γ in DIMSUM on the accuracy of the approximation of A^TA for the $USERS \times MOVIES$ matrix since this is a tall skinny matrix. This parameter γ modulates the sampling behavior. Higher values for γ will lead to more emissions which should preserve singular values more, but will also lead to higher communication costs and more complexity. We analyze the impact of γ by computing the Mean Squared Error (MSE) over all the entries in the approximation and exact computation of A^TA for different values of γ.

Before analyzing the impact of γ two remarks should be made. First, we implemented mapper, reducer and combiner functions for the DIMSUM algorithm but without the actual use of Hadoop. This lack of distribution makes our implementation of the algorithm rather slow. Therefore, a smaller subset of the data was used to perform our analysis. Our used subset was based on the probe test file, provided by the Netflix dataset itself. Nevertheless, the gained insights should be extendable to the full dataset. Secondly, Zadeh and Carlsson (2013) assume entries of A have been scaled to be in [-1,1] using the largest magnitude element. Therefore, we perform this operation as well. However, this does not seem to affect the general trend that is visible in the impact of γ although our computed error was lower.

The maximum value of gamma that is useful to test would be the square of the largest column norm. For this value emission probability is 100% and we actually perform an exact computation of A^TA instead of an approximation. Figure 2 shows a plot of the MSE of 20 different γ values, uniformly distributed over the range $[\frac{\gamma_{max}}{20}; \gamma_{max}]$ with γ_{max} being the maximum useful value mentioned before. This experiment was repeated 10 times in order to obtain a more reliable average and standard deviations.

There is a clear downward trend in the MSE or upward trend in the DIMSUM performance when γ rises. This is to be expected, since emission probability increases for higher γ and higher sampling will generally lead to better approximations of the singular values. This has been formalized for DIMSUM in Probably Approximately Correct (PAC) guarantees stating $\gamma = \Omega(n/\epsilon^2)$, with n the number of columns of A and ϵ the error we allow. Concretely, higher γ is associated with a lower ϵ, which is what we observe. As expected, if γ reaches a certain threshold emission probability is 100% and we even obtain an MSE of 0 as we are performing an exact computation of A^TA instead of an approximation. We can see that higher sampling also leads to less variance in the results.

It should also be noted that the MSE might seem rather low in general, even for low values of γ. This is caused by two factors. First, the preliminary rescaling we performed can have this effect. Secondly, in this specific scenario our implementation will always have a correct approximation for any zero entry. Since the probe test matrix has a very low density this leads to a large amount of values being correctly approximated. In practice, this leads to an MSE that is lowered by a constant factor $\frac{\#non-zero\ entries}{\#entries}$. For reference sake, we included Figure 3 showing the same trend in MSE when it is only being computed on the non-zero entries.

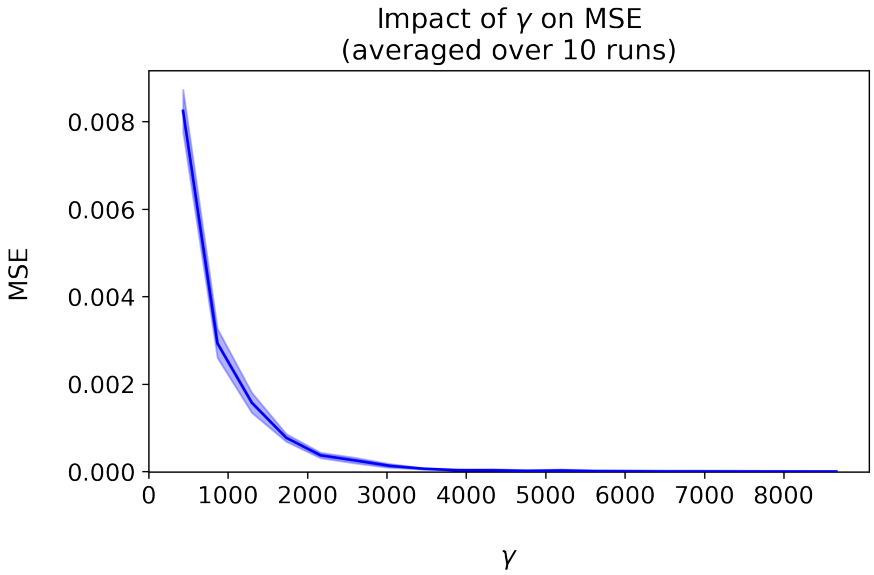

Figure 2: Impact of γ on the MSE between the exact $A^T A$ and the variant approximated by DIMSUM.

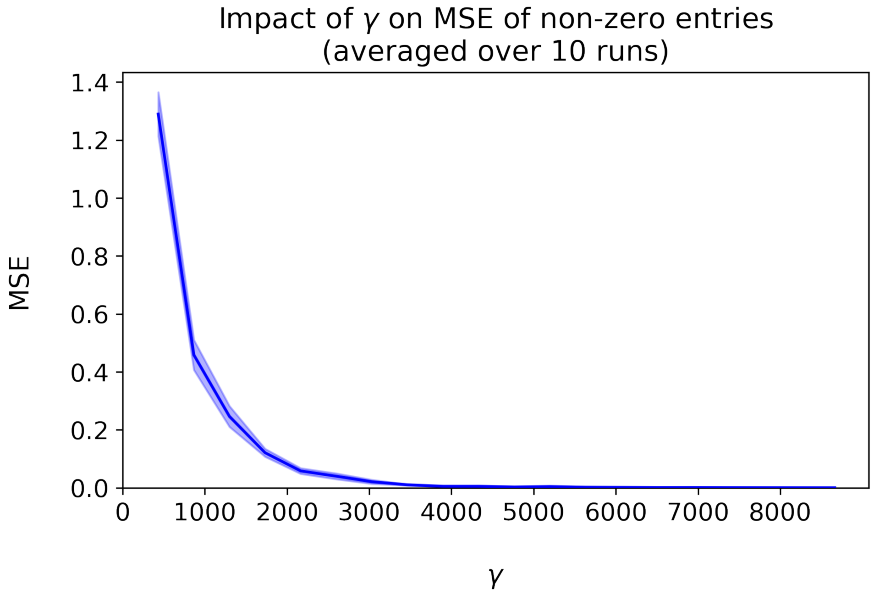

Figure 3: Impact of γ on the MSE, computed only on non-zero entries, between the exact $A^T A$ and the variant approximated by DIMSUM.

3 Gradient Descent with Latent Factors

In this section we will make use of built-in functions to compute the K largest latent factors of the SVD decomposition of the wide short sparse matrix $MOVIES \times USERS$. The summarizing components U, Σ and V^T are multiplied to construct the matrices Q and P^T with $Q = U$ encoding latent information about the rows (movies) and $P^T = \Sigma V^T$ encoding information about the columns (users). These latent factors Q and P^T can in effect be used to predict ratings of A. However, A is a sparse matrix meaning a lot of values are missing which causes the latent factors to be inexact and in turn might lead to bad generalization of the predictions. This can be mitigated through a Gradient Descent approach which will iteratively refine both Q and P^T based on an objective function using prediction error and regularization.

The analysis presented in this section will focus on two main aspects. First, the influence of the hyperparameter K on the performance of our system will be investigated. This parameter K indicates how many singular values are computed in the SVD decomposition. Since the largest singular values represent the most significant factors, they can therefore be seen as a more manageable compressed representation of A. Changing the "compression rate" can of course be expected to influence the generalization of our predictions. Secondly, we will compare the evolution of in-sample and out-of-sample errors for two gradient descent approaches to get more insight into the evolution of the performance of our system throughout the learning process. The first approach is stochastic gradient descent (SGD) that randomly shuffles the samples each epoch and performs frequent updates, namely one after each sample. The second approach is batch gradient descent (BGD) that performs only one update per epoch, but this update is larger and more consistent as it considers all samples. The test set used in this analysis was based on the provided probe text file and the RMSE was used as accuracy metric. For more information we refer to section 4.

Before delving deeper into the analysis, we want to remark that other hyperparameters, besides K, will play a role in the performance of our system as well. The learning rate η, regularization parameter λ and even the number of epochs will interact with K in the sense that different values for them might lead to another value of K performing the best. Since we mostly wanted to focus on K, we performed some initial smaller experiments on multiple subsets of the data to find good values for these other hyperparameters and kept them fixed during the main experiments. In these experiments we took a random subset of the data, performed an 80-20 test split, trained our model with multiple combinations of hyperparameter values on 80% of the subset and validated it on the other 20% of the subset. We repeated this with five different subsets. Due to the size of the dataset and even the subsets, a more elaborate cross-validation approach was not feasible with our limited resources but this seemed to be an adequate alternative that gave good results.

Impact of K The impact of K has been investigated by performing two epochs of stochastic gradient descent with $\lambda = 0.01$ and $\eta = 10^{-5}$, with K varying from 1 to 100. For each value of K the RMSE after learning was obtained for a training set and a validation set. The training and validation set were obtained by randomly splitting up the probe training set into a training and validation set according to an 80-20 split. The results are shown in Figure 4.

The results clearly show that both the in-sample and out-of-sample error initially decrease when increasing the number of singular values that are used. This is to be expected as extracting too few latent factors generally leads to having too little information to be able to learn a useful pattern in the training data. However, it is also clear that extracting too many latent factors leads to overfitting and an increasing out-of-sample error. Noise from the training data is learned which leads to bad generalization. This is clearly seen by the divergence of the RMSE on the

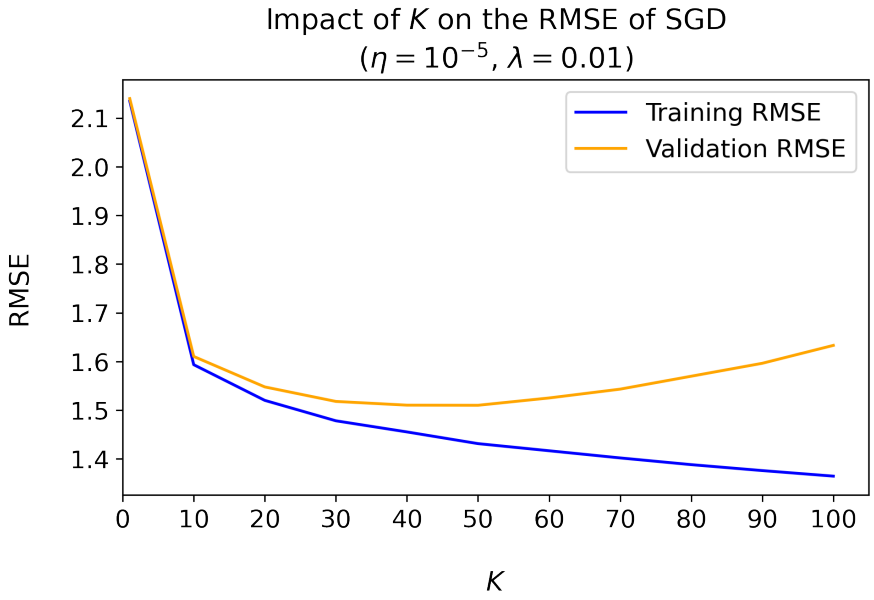

Figure 4: Impact of K on the training and validation RMSE obtained by performing two epochs of stochastic gradient descent with $\eta = 10^{-5}$ and $\lambda = 0.01$.

training and the validation set, with the validation RMSE starting to increase when $K > 50$ while the training RMSE keeps decreasing further.

As mentioned before, other hyperparameters will also have an influence on which value of K performs well. If the complexity of our model increases due to higher K, we could for example increase regularization to mitigate the risk of overfitting.

Comparison of Performance of SGD and BGD To compare the performance of SGD and BGD, we first performed some fine-tuning to get reasonable results for both approaches. Hyperparameters K and λ could be kept the same, i.e., $K = 50$ and $\lambda = 0.01$, but a different η was needed. The results of performing five epochs of fine-tuned SGD and BGD are shown in Figure 5.

As was noticeable in our preliminary experiments, SGD generally performs better with smaller learning rates. Since the gradient estimate is based on only one sample, occasional updates in the "wrong direction" are plausible. Although this stochasticity can actually help the model to escape local minima, a smaller learning rate is generally still preferred to prevent too significant deviations in the learning process. Therefore, we used a learning rate $\eta = 10^{-5}$ for SGD. Despite the relatively small η we see both the training and test error decrease quickly, indicating successful learning. However, after only a couple of epochs the errors seem to stagnate with further epochs only marginally improving the results. The frequent updates allow the model to reach a good approximation while only having to go over all samples a couple of times. On a final note, we want to mention that we performed random shuffling of the input data in the case of SGD which seemed to generally improve results as well.

In the case of BGD the gradient is not an estimate, which should lead to more stable updates. However, these updates are much less frequent as the model is only updated once after going over all samples. Because these updates are less frequent BGD generally benefits from using

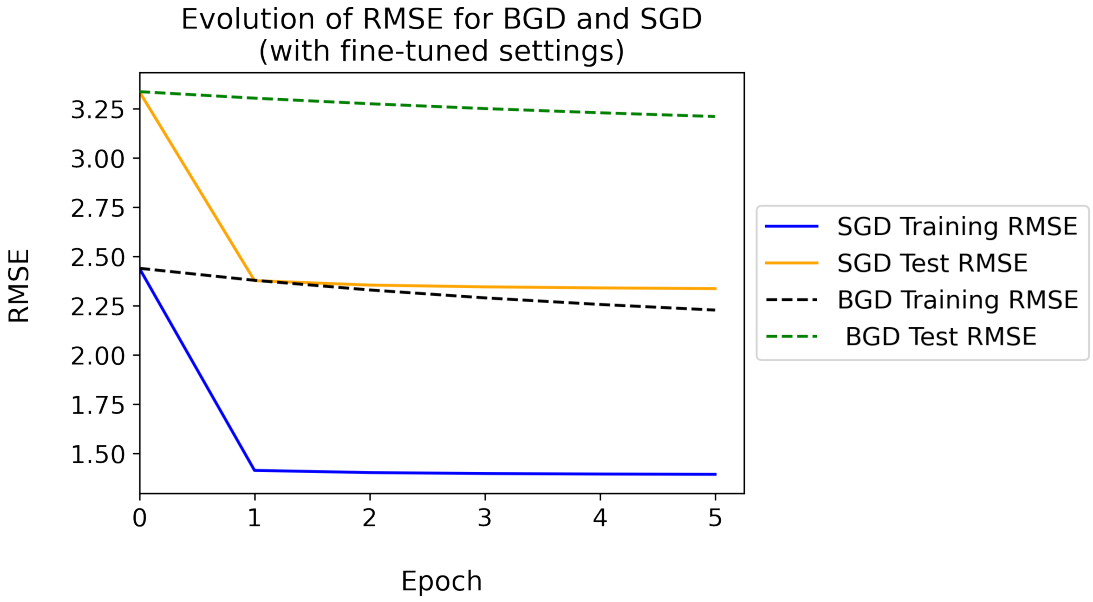

Figure 5: Evolution of the training and test RMSE throughout 5 epochs for SGD and BGD with $K = 50$, $\lambda = 0.01$ and η respectively being 10^{-5} and $\frac{0.1}{|training\ set|}$.

a larger η. Nonetheless, performing BGD with $\eta = 0.1$ and $\eta = 10^{-5}$ both led to very large steps for the used dataset even after trying to scale the input data. This could not only lead to overshooting or oscillating around the minimum, but in our case it seemed to actually lead to subsequent large overcorrecting steps that cause divergence. Since the size of the dataset can influence the size of the total gradient sum, we tried to mitigate the issue by taking the average gradient instead of the gradient sum. This indeed led to more stable learning, albeit rather slow learning. Using the average instead of the sum comes down to using a lower η of around 10^{-8} since we simply divide the sum by a constant. Again, we see both the training and test error decrease indicating successful learning. The rate of decrease, however, is much lower than for SGD. In this case, this is most likely caused by the lower η. Nevertheless, a fine-tuned version of SGD seems to outperform a fine-tuned version of BGD for the Netflix Challenge for the same number of epochs. Even without the need for an averaged gradient in BGD, this would be the expected outcome. The fact that multiple updates can occur in one pass over the training data for SGD generally leads to quicker convergence to good predictions for the same number of total passes. This makes SGD a better candidate for learning in a setting with large datasets. Finally, it is noteworthy that there is a difference visible between in-sample and out-of-sample error that is similar for both approaches.

4 Accuracy

To be able to measure whether our system is correctly learning, some evaluation metric should be used. We chose to compute the root mean-squared error (RMSE) between made predictions and the actual ratings as a way to estimate the accuracy of the gradient descent approaches mentioned in section 3.

Two problems had to be taken into account when evaluating our system. First, the prediction matrix would have the same dimensions as our original input matrix which would lead to the same scalability and memory problems as mentioned in section 1. Therefore, we will sequentially go over all the non-zero ratings in the matrix with the actual values and predict a single rating to compute the squared error on. This prediction can be made by multiplying the i-th row of Q with the j-th column of P^T. These squared errors are then aggregated to obtain the RMSE. Secondly, we have only one dataset and in-sample accuracy cannot reliably be used to obtain generalization. Therefore, we split our data into a training and a test set so out-of-sample accuracy could be measured as well.

Two approaches have been used to perform the aforementioned train-test split. In both cases we split up a sparse matrix into two sparse matrices of the same size as the original sparse matrix, but with non-overlapping non-zero entries and possibly with a different density. The first approach is the traditional approach where we randomly select a certain percentage of the original matrix to obtain a testing matrix and only use the rest as a training matrix. For reproducibility, we provide the option to provide the seed used in the pseudo-random sampling operation. The second approach uses the probe text file included in the Netflix dataset. This file contains movie ID and user ID entries in a similar format as the full dataset. We can parse this probe file, using the same mappers that were obtained when parsing the full dataset, to get the indices that indicate which ratings from the full sparse matrix should be used in a test matrix. Once we have these, we can simply split off these specific entries from the full sparse matrix to obtain a test set and use the rest of the full matrix as training set.

Most results presented in this report were obtained using the second approach, based on the probe text file. We give a small overview about this training-test split in Table 2. This overview shows that the test set obtained this way is relatively small, being only 1.4% of the original dataset while the most common split ratio is said to be 80-20. However, considering the test set still contains more than a million samples, has a broad coverage of Movie IDs and Users IDs, and has been suggested by the dataset creators we decided to rely on the probe file.

	# Movie IDs	**# User IDs**	**# Ratings**	**% of Full Dataset**
Training	17770	480189	99072112	98.6
Test	16938	462858	1408395	1.4
Full	17770	480189	100480507	100

Table 2: Overview of the data entries in the train-test split for the Netflix challenge based on the provided probe text file.

References

Zadeh, R. B., & Carlsson, G. (2013). Dimension independent matrix square using mapreduce. *arXiv preprint arXiv:1304.1467*.