Total Variation Regularisation for Item KNN Collaborative Filtering: Performance Analysis

Lars van Blokland
Supervisor(s): Elvin Isufi, Maosheng Yang
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

Algorithms that recommend items to users are known as recommender systems and have become an important part of online ecosystems. These systems calculate the similarity between given users or items based on the ratings they have received. Such similarities can be modeled as a graph where users or items (nodes) have connections (edges) to other users or items if the ratings they possess are alike. Using a graphical representation allows for the use of graph regularisers. These techniques predict ratings for a given user or item by using the ratings of users or items that are connected to the target in the graph. For the Total Variation Regulariser, much is still unclear concerning its performance when applied to Item KNN Collaborative Filtering. This report will show why Item Total Variation was unable to out-perform more traditional recommender systems in the conducted experiments. The findings indicate how this poor performance could in part be attributed to the choice in dataset. Additional results also convey that Total Variation may be predisposed to performing worse for a certain type of metric.

1 Introduction

As internet usage became more prolific, so to did the development of systems that help us navigate the endless content [1]. These recommender systems use vast amounts of information to assist users in finding items that will fit their needs. Initial research into recommender systems focused on collaborative filtering [2], where users are recommended items that similar users have also chosen. Almost all new research into recommender systems has been centered around machine learning [3]. Examples such as [4] make use of a complex neural network to boost the performance of a recommender system. This increased popularity of neural methods has caused them to fall under scrutiny from others. Papers such as [5, 6] suggest that positive results booked by machine learning methods are often the result of comparisons to weak baselines, or a lack of proper empirical rigor. Others have noted that papers on machine learning recommender systems are frequently difficult to reproduce and rarely outperform standard techniques [3]. These findings suggest that there is some merit to the continued development of non-machine learning methods.

A potential alternative is the idea of a graph-based recommender system. Indeed, nodes connected by weighted edges bear a close resemblance to the notion of users and the similarities between them used by collaborative filtering [7]. Using graphs to benefit recommender systems is not a novel idea [8, 9], however, the concept of applying a graph signal regulariser [10] to estimate unknown ratings is still relatively unexplored. In the field of graph signal processing these regularisers haven proven to be effective ways of reconstructing missing signal information [11]. In the context of recommender systems, regularisers can be utilized to estimate unknown ratings for a user based on the ratings of users similar to them.

The goal of this research is to evaluate the performance of the Total Variation [11] regulariser when applied to item KNN collaborative filtering. The accuracy of predicted ratings made with the Total Variation method will be compared with those made by a number of different baseline recommender systems. By applying both the Total Variation and the chosen baselines in a variety of different settings, and measuring their performance with a diverse set of metrics, the goal is to determine the effectiveness of a graph-based, total variation recommender system.

2 Methodology

This section aims to outline how results were achieved by explaining the manner in which experiments were conducted. It describes the process of acquiring and preparing the target dataset for the use in experimentation.

2.1 Acquiring the data

The dataset used for the evaluations done in this project is the MovieLens 100K (ML100K) [12] dataset. ML100K contains 100.000 movie ratings, making it a good choice for research on recommender systems. One challenging characteristic of the ML100K dataset is its rating sparsity. The 943 users whose ratings comprise ML100K have on average only rated 106 of 1682 available movies. This sparsity can make estimating predictions difficult, as it decreases the overlap between any two users or any two movies. By representing the data as a graph it becomes possible to use regularization techniques to predict missing entries. These regularization methods have proven to work effectively even when the data is extremely sparse. [10]

2.2 Preparing the data

The ML100K dataset guarantees that each user present in the dataset will have rated at least 20 items, which ensures that users contain enough information to be compared to each other. Unfortunately this guarantee does not extend to items, with some items having only a single rating. In [12] it is described that collaborative filters have historically under performed when trained on data where users or items possessed fewer than 20 ratings. This has motivated the decision to remove all items with fewer than 20 ratings from the dataset before the creation of any train or test set. Statistics for this change are visible in Table 1.

2.3 Training and testing

Evaluating the performance of the Total Variation method was done with the help of training and testing sets. Each train and test split comprises 80 and 20 percent respectively of the filtered ML100K dataset. For all ratings of a single item, 80 percent went to the train set, and 20 percent went to the test set. Whilst this method of splitting the data is random on a local scale, the resulting splits are different than dividing all ratings at the same time. The motivation behind this decision was to ensure that each item possessed a sufficient number of both training and testing examples. A global 80 - 20 split could have assigned all ratings for a single item to either the train or test sets. This would lead to either training on data that would never be tested, or being tested on data that was never trained for.

	Number of Ratings	Mean User	Mean Item	Median User	Median Item
Unfiltered Database	100000	106	59	65	27
Filtered Database	94728	100	102	62	70

Table 1: Comparison between statistics of the filtered and unfiltered ML100K dataset. Mean and Median columns indicate the number of ratings as counted by the specified method.

3 Approach

This section describes how each process used in the creation of rating predictions operates. It details which metrics are applied to evaluate results, and outlines which baselines are used and how they function.

3.1 Creating the graph

Rating data is represented as a matrix where row i contains the ratings for user i and column j contains the ratings for item j. If $cell_{ij}$ contains a rating, this value is representative of the rating given by user i to item j. Empty cells indicate that the given user has not rated the given item.

$$\mathrm{R} = \begin{array}{c} \begin{matrix} \mathrm{item}_1 & \mathrm{item}_2 & \cdots & \mathrm{item}_j \end{matrix} \\ \begin{pmatrix} 1 & 5 & \cdots & 2 \\ 2 & 4 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 3 & 3 & \cdots & 1 \end{pmatrix} \end{array} \begin{matrix} \\ \mathrm{user}_1 \\ \mathrm{user}_2 \\ \vdots \\ \mathrm{user}_i \end{matrix}$$

(1)

Before nodes can be connected to create a graph, it is necessary to calculate the similarities between all items. To this end, each column of the rating matrix is compared to each other column using the Pearson similarity method [7]. The result of this operation is an N by N matrix, where N is the number of items in the original rating matrix. The value in $cell_{ij}$ represents the similarity between the ratings of item i and item j. These values range from 1, indicating a perfect correlation, to -1, indicating a perfect inverse correlation.

$$S = \begin{array}{c} \begin{matrix} item_1 & item_2 & \cdots & item_n \end{matrix} \\ \begin{pmatrix} 1 & 0.543 & \cdots & 0.221 \\ 0.543 & 1 & \cdots & 0.783 \\ \vdots & \vdots & \ddots & \vdots \\ 0.221 & 0.783 & \cdots & 1 \end{pmatrix} \end{array} \begin{matrix} \\ item_1 \\ item_2 \\ \vdots \\ item_n \end{matrix}$$

(2)

The next step in creating the graph involves removing similarity values until each node is only connected to its K most similar neighbours. Firstly, all values along the major diagonal are set to 0. This ensures that nodes are not connected to themselves with an edge. The second step is to remove all but the K highest similarity values for each node. To guarantee

that the resulting matrix is undirected, removal of the unwanted similarities is done in a row wise fashion. Starting at the top row of the matrix similarities are deleted until only the K largest remain. The remaining similarities are indicative of edges connecting the current node to its closest neighbours. For each of these neighbouring nodes the current node is marked as one of their K neighbours. As this process continues it is possible that nodes are encountered that have already been assigned K neighbours. Once each row of the matrix has exactly K similarities left, the resulting matrix is known as the adjacency matrix. In this matrix a non-zero value x at $cell_{ij}$ represents an edge with weight x between nodes i and j.

$$A = \begin{array}{c} \begin{array}{cccc} item_1 & item_2 & \cdots & item_n \end{array} \\ \begin{pmatrix} 0.0 & 0.543 & \cdots & 0.0 \\ 0.543 & 0.0 & \cdots & 0.783 \\ \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.783 & \cdots & 0.0 \end{pmatrix} \end{array} \begin{array}{c} item_1 \\ item_2 \\ \vdots \\ item_n \end{array}$$

(3)

Before the adjacency matrix can be used for Total Variation, the similarity values need to be normalized. Normalization of the adjacency matrix is achieved through division of all similarity values by the largest eigenvalue of the adjacency matrix [10]:

$$\mathbf{A}_n = \frac{1}{|\lambda_{max}|}\mathbf{A} \tag{4}$$

3.2 Applying Total Variation

Total Variation is a method of graph signal regularisation. In signal regularisation the signals of nodes are compared to the signals belonging to their neighbours. The resulting value indicates the degree to which the signal of the target node is similar to the signals of its neighbours. In the context of recommender systems, Total Variation can be used to calculate the values of missing ratings for a given user or item. To calculate these missing ratings the training set is iterated over in a row-wise fashion, where each row contains all known ratings for a given user. Together with the aforementioned normalized adjacency matrix, and a hyper-parameter μ, these rows comprise the input needed to perform Total Variation.

The formula describing Total Variation is as follows:

$$\min_{\mathbf{x}_i \varepsilon R^U} \|\mathbf{y}_i - \mathbf{x}_i\|_2^2 + \mu \|\mathbf{x}_i - \mathbf{A}_n \mathbf{x}_i\|_1 \tag{5}$$

Where:

- $\mathbf{y}_i$ is a vector containing the known ratings of user i.
- $\mathbf{x}_i$ is a vector, describing the current state of the optimization parameter.

- μ is a parameter that determines how much the ratings of neighbouring nodes influence the prediction.
- $\mathbf{A}_n\mathbf{x}_i$ is the dot product of the normalized adjacency matrix and the optimization parameter.

The function can be split into three parts:

- $\|\mathbf{y}_i - \mathbf{x}_i\|_2^2$ the fitting term: assigns a penalty if the optimization parameter $\mathbf{x}_i$ is too different from the known ratings $\mathbf{y}_i$
- $\|\mathbf{x}_i - \mathbf{A}_n\mathbf{x}_i\|_1$ the regulariser: assigns a penalty if the values contained in the optimization parameter are not similar enough to the values of their neighbours.
- μ: determines the weight of the regulariser.

To calculate the missing ratings, the Total Variation function is solved using the CVXPY package implementation of Lasso Regression [13]. This algorithm finds an allocation of values for $\mathbf{x}_i$ that minimizes the result of the Total Variation function. $\mathbf{x}_i$ is initially filled with random values. At each iteration of the algorithm $\mathbf{x}_i$ is updated in a manner that further lowers the result of the Total Variation function. This procedure terminates when the difference between two consecutive results is sufficiently small, or a maximum number of iterations has been reached. When either of these options happen, the optimization parameter $\mathbf{x}_i$ is returned as the vector of predicted ratings for user i. When each row in the original rating matrix has been processed, the returned vectors can be combined to construct a full matrix of predicted ratings.

3.3 Measuring Performance

Performance of the Total Variation Collaborative Filter, and of the baselines, was measured with the following four methods:

Root Mean Squared Error measures the difference between predicted and true ratings.

Normalized Discounted Cumulative Gain at K measures the quality of rankings within the predicted values for the K highest ratings. [7].

Precision at K measures how many of the K highest predictions are relevant. Here relevant means possessing a value larger than the average value for a given user or item.

Recall at K measures how many of K relevant true ratings have corresponding predicted ratings that are also relevant.

Lower RMSE values are more desirable. NDCG, Recall, and Precision values should be as close to one as possible.

3.4 Baselines

To evaluate the performance of the rating predictions made using Total Variation, error values of these predictions are compared to error values of predictions made by a number of baseline recommender systems. The following methods of rating prediction were used as baselines for this research:

Item KNN: A classic method of collaborative filtering that uses item-item similarities to construct a neighbourhood of similar items [7]. Ratings are then estimated by taking

weighted averages of the ratings possessed by items in the neighbourhood. This baseline is influenced by a single hyper-parameter K which determines the number of items included in the neighbourhood.

User KNN: Identical to Item KNN, except that the user-user similarities are used instead of the item-item ones.

User Mean Rating: A very simple method where the predicted rating is the mean of all ratings made by other users for the target item.

Item Mean Rating: Similar to User Mean Rating, except that the mean of ratings made by the target user for all other items is taken instead.

4 Experimental Setup and Results

This section describes in what manner experiments were conducted, and what the results of these experiments were.

4.1 Setup

To evaluate the performance of the Total Variation Collaborative Filter, rating predictions were calculated for a range of hyper-parameter values. These hyper-parameters include K, determining the number of neighbours that are connected to each node, and μ, which controls how much influence these neighbours have. For K the available values were $[10, 20, 30, 40, 50]$, and for μ the available values were $[0.0, 0.25, 0.5, 0.75, 1.0]$. These ranges enable 25 distinct experiments

The KNN baseline filters only possess a single hyper-parameter K. It indicates how many ratings of other items are considered when calculating the weighted average. To ensure that results form both filters were comparable, the range for K used by the baseline filter is identical to the one used by the Total Variation filter.

Each experiment was executed five times on randomly generated train-test splits. These results were then averaged and taken as the final values.

4.2 Results

Table 2 gives an overview of results achieved by the Total Variation Collaborative Filter and the baselines. The TV Collaborative Filter performed best when K and μ were 10 and 1.0 respectively. The decision for these values is motivated in section 4.3.

An interesting trend present in the results is that the baselines utilizing user-user relations have consistently better metric scores than their item-based counterparts. A potential explanation for this behaviour is the distribution of ratings within the ML100K dataset. Table 1 shows that the filtering process described in 2.2 makes the mean and median number of ratings approximately equal for users and items. However, the mean and median do not entirely describe how the ratings are distributed amongst users and items.

Figure 1 shows the distribution of ratings across users and items. It indicates that users possess more rating weight than items for the left side of the figure. This shows that user ratings are more optimally distributed than item ratings. A rating given to a user or item with few ratings will better improve the quality of information than when this rating is given to a user or item with a large number of ratings. Whilst the rating weight of users and items

	RMSE	Rec@5	Rec@10	Rec@20	Prec@5	Prec@10	Prec@20	NDCG@5	NDCG@10	NDCG@20
Item KNN CF	1.077	0.468	0.677	0.832	0.582	0.573	0.556	0.841	0.872	0.900
User KNN CF	1.035	0.515	0.711	0.863	0.671	0.628	0.589	0.882	0.906	0.927
Item Mean	1.034	0.472	0.673	0.835	0.582	0.569	0.558	0.805	0.845	0.880
User Mean	1.011	0.522	0.716	0.865	0.683	0.634	0.591	0.887	0.910	0.930
Item TV CF	1.025	0.468	0.676	0.832	0.582	0.580	0.561	0.823	0.867	0.891

Table 2: Performance metric scores achieved by the Item Total Variation Collaborative Filter (Item TV CF), the Item and User Collaborative Filters (Item KNN CF, User KNN CF) and the Item and User means. Both the TV and KNN Collaborative Filters had a K of 10. The TV Collaborative Filter had a μ of 1.0.

is both concentrated at lower Rating per User or Item values, items have more weight at higher values, indicating a worse distribution of information.

This distribution could explain why the Total Variation CF was unable to perform better than the Mean User baseline. What this does not account for however is the performance of the Total Variation CF when compared to item-based baselines. Only in RMSE does Total Variation CF stand out. In the three remaining metrics it rarely performs better than the baselines. Since Recall, Precision and NDCG are all concerned with the quality of numerically high ratings, worse performance in these metrics could indicate that Total Variation CF is not adept at predicting high ratings.

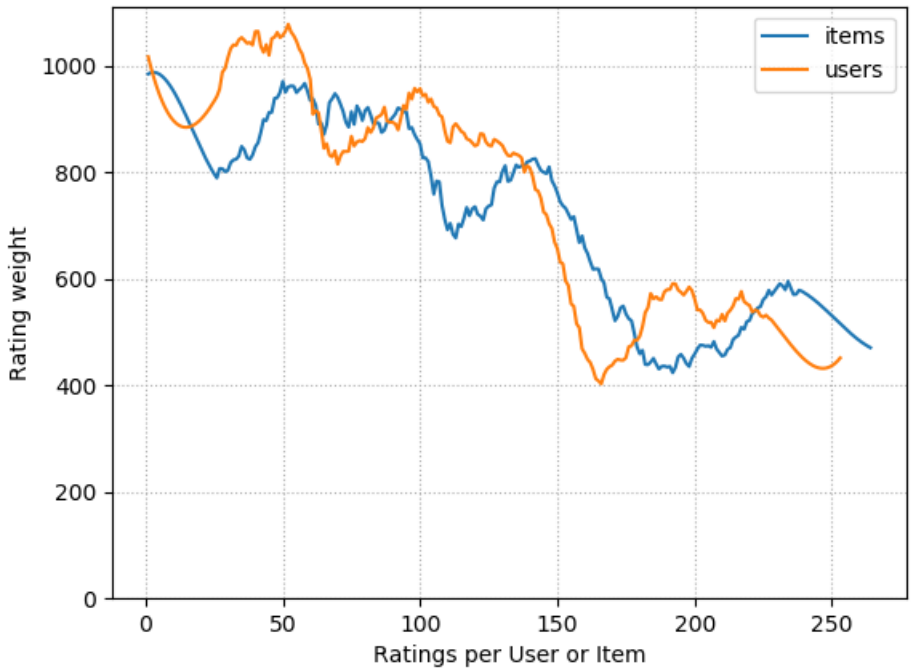
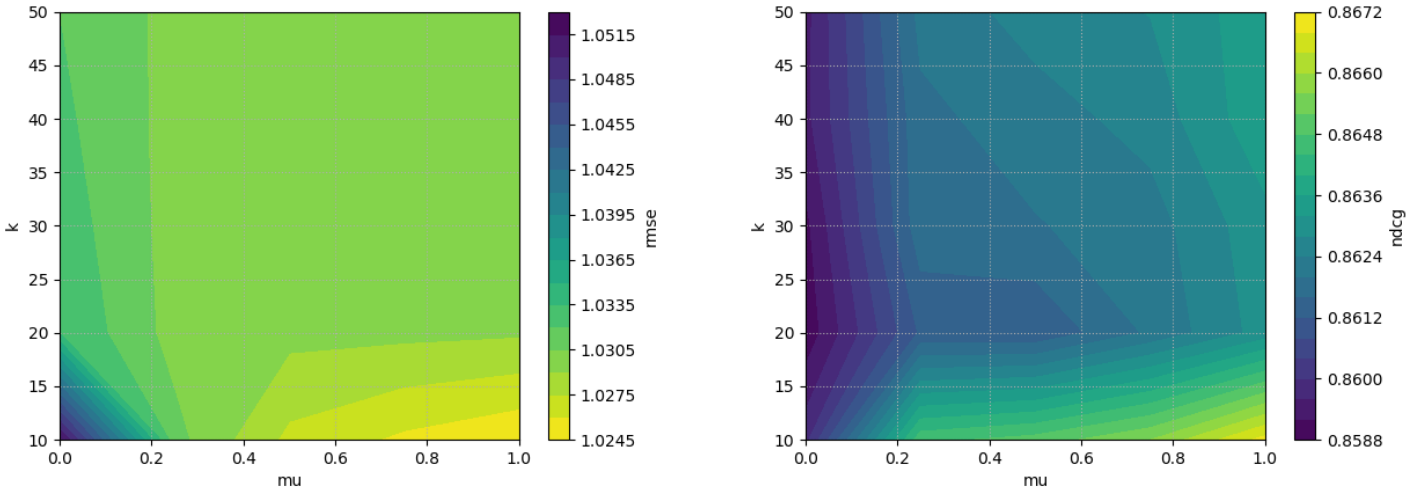


Figure 1: Comparison of the distribution of rating weight of users and items for the filtered ML100K dataset. Rating weight is described as the number of ratings given to a specific user or item, multiplied by how many users or items received that number of ratings.

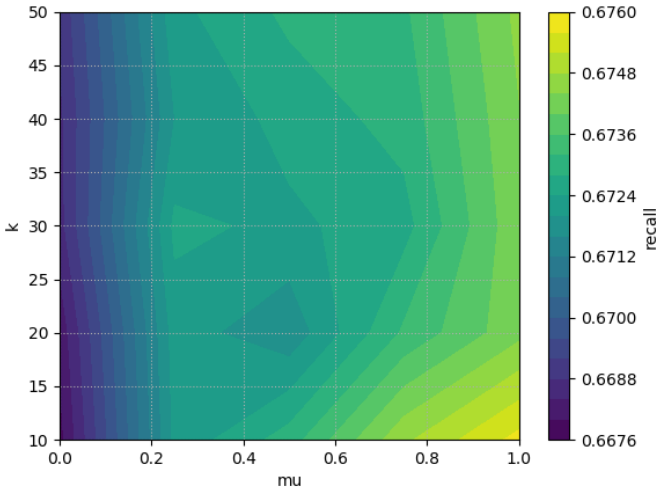
4.3 Parameter Optimization

To find ideal parameter values for the Total Variation CF, 25 distinct experiments were run. Each experiment had a different combination of values for K and μ. The results of these trials are visualized in Figure 2. Extracting the ideal parameter values from the figure is not difficult. For all metrics, the Total Variation CF performs best when K is 10 and μ is 1.0. Results being optimal when K is 10 indicates that including too many neighbours decreases the ability of Total Variation CF to accurately predict ratings. Increasing K dilutes the neighbourhood with items that will be increasingly dissimilar to the target item, making their information less valuable. For μ, Figure 2 shows that favouring the regularizing term over the fitting term when performing Total Variation is beneficial to performance. For all metrics except RMSE, an increase in μ constituted an increase in the metric performance, regardless of the current value assigned to K. This seems to indicate that increasing the influence of neighbours is always positive, irrespective of how many neighbours there are.

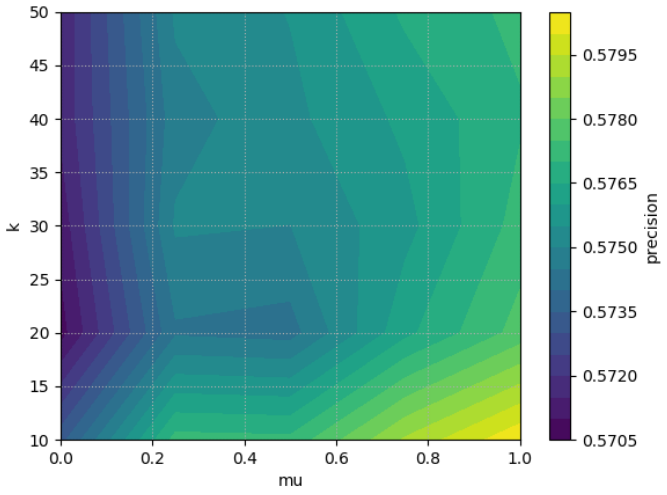


(a)

(b)



(c)



(d)

Figure 2: Performance of the Total Variation CF for RMSE (a), NDCG (b), Recall (c) and Precision (d). Yellow colour hues indicate a better performance, whilst blue hues indicate a worse performance.

5 Responsible Research

To ensure proper reproducibility of this research, only publicly available resources were used to conduct the experiments. The dataset used for all measurements, ML100K, is available for free online [12]. This is also true for the Lasso Regression solver [13] used to find solutions for Total Variation. All operations mentioned in sections 2 and 3 can be replicated using the Python programming language in tandem with packages such as Pandas, Numpy and Scipy [14, 15, 16]. Figures and visualisations in this research were made with Matplotlib [17]. This report and the resources mentioned above should allow for the reproduction of presented results.

6 Discussion and Conclusion

This research has measured the performance of the Total Variation Item KNN CF. Analysis of the results has shown that the Total Variation CF performs worse than some common baseline recommender systems. User-based baselines in particular generated better predictions across all performance measures. A possible explanation for this was found in a difference of rating distribution amongst users and items. Another point of interest was that the Total Variation CF was only able to outperform the item-based baselines in RMSE score. This finding indicated that the Total Variation CF struggles to accurately predict numerically high ratings, and as a result only has average scores for the NDCG, Recall and Precision metrics.

An attempt can be made to verify these claims by comparing the results of the Item Total Variation CF with results from a User-based Total Variation CF. Table 3 presents results found by [18], wherein the performance of a User Total Variation CF is measured. A notable takeaway of these results is the superior RMSE score when compared to the Item Total Variation CF. This could indicate that the ML100K dataset has higher quality information when applying a user based approach, as discussed in section 4.2. However the presence of factors such as differing values for K and μ, other methods of dataset preparation and potentially better implementations of Total Variation could all have contributed to a better RMSE score.

The results in this research have painted a limited overview of the Total Variation CF and its performance. Any future work would focus on providing a more in-depth analysis. This work should ascertain why Total Variation CF scored badly on metrics such as NDCG, Recall and Precision when compared to other item-based techniques. Further points of interest include testing on multiple datasets, utilising more sophisticated baselines and further experimentation on ways to set up Total Variation.

	RMSE	Rec@5	Rec@10	Rec@20	Prec@5	Prec@10	Prec@20
User TV CF	0.958	0.518	0.713	0.862	0.683	0.634	0.591
Item TV CF	1.025	0.468	0.676	0.832	0.582	0.580	0.561

Table 3: Comparison between the metric scores of the Item Total Variation CF discussed in this research, and the User Total Variation CF discussed in [18]. Results for the User TV CF were predicted using a K of 40 and an average μ of 0.7. Furthermore, results for the User TV CF did not include evaluation by the NDCG metric, warranting its exclusion in the table.

References

[1] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pp. 158–167, 2000.

[3] M. F. Dacrema, P. Cremonesi, and D. Jannach, "Are we really making much progress? a worrying analysis of recent neural recommendation approaches," in *Proceedings of the 13th ACM conference on recommender systems*, pp. 101–109, 2019.

[4] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

[5] J. Lin, "The neural hype and comparisons against weak baselines," in *ACM SIGIR Forum*, vol. 52, pp. 40–51, ACM New York, NY, USA, 2019.

[6] D. Sculley, J. Snoek, A. Wiltschko, and A. Rahimi, "Winner's curse? on pace, progress, and empirical rigor," 2018.

[7] C. C. Aggarwal *et al.*, *Recommender systems*, vol. 1. Springer, 2016.

[8] Z. Huang, W. Chung, and H. Chen, "A graph model for e-commerce recommender systems," *Journal of the American Society for information science and technology*, vol. 55, no. 3, pp. 259–274, 2004.

[9] K. Lee and K. Lee, "Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items," *Expert Systems with Applications*, vol. 42, no. 10, pp. 4851–4858, 2015.

[10] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3042–3054, 2014.

[11] S. Chen, A. Sandryhaila, J. M. Moura, and J. Kovačević, "Signal recovery on graphs: Variation minimization," *IEEE Transactions on Signal Processing*, vol. 63, no. 17, pp. 4609–4624, 2015.

[12] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[13] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[14] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[15] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E.

Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[16] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[17] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[18] K.Mariunas, “The performance of total variation regularizer for user collaborative filtering,” 2022.