

Recommendation Systems: Collaborative Filtering using Matrix Factorization

CSE 575: Statistical Machine Learning

Fall '22

Project Report: Group 13

([GitHub Repo](#))

Name	ASU ID	Email-ID
Visaj Nirav Shah	1225369210	vshah47@asu.edu
Darsh Vaidya	1225514966	dvaidya5@asu.edu
Malvi Mungalpara	1225184103	mmungalp@asu.edu
Meet Kishorbhai Lakhani	1225636854	mlakhan3@asu.edu
Vibavari Gurunathan	1225412305	vnolas25@asu.edu
Rohit Rajesh Rughwani	1223264705	rrughwan@asu.edu

Under the able guidance of

Course Instructor: Prof. Yingzhen Yang

Teaching Assistant: Mr. Utkarsh Nath

Introduction	2
Problem Description	3
Related Works	3
Methodology	4
Data Ingestion: About Dataset	4
Data Pre-Processing and Aggregation	5
The Math	6
The Model	7
Temporal Folds: Custom GridSearchCV	8
Results	9
Conclusion and Future Works	12
Future Work	12
References	13

Introduction

Data is growing at an exponential rate and so are the choices available online. Humans are faced with endless information on a daily basis, therefore it is essential that users find relevant results and are not lost in an overwhelming sea of data. A large quantity of data can be difficult to analyze and make sense of, especially if it is not organized or structured in a meaningful way. This can make it difficult to extract valuable insights and knowledge from the data. Not finding appropriate information time and again can discourage users from using a particular platform and lead to frustration. Thus arises the need for some system that prioritizes and delivers pertinent data.

A recommendation system is a tool that is used to predict what a user may be interested in based on their past behavior. These systems are commonly used by online retailers and streaming services to suggest products or content that a user may like. They work by analyzing the user's past interactions with the system, such as the items they have viewed or purchased, and using that information to make personalized recommendations. Some recommendation systems also incorporate data from other sources, such as a user's social media activity or demographics, to make their recommendations more accurate.

Recommender systems deal with information overload and give personalized suggestions to the users by analyzing their current interests and product selection pattern from a large amount of dynamically generated data according to the user's behavior with respect to various items, and help them access the content they're interested in and sometimes would never have searched for. Recommender systems are being used in abundance in eCommerce and entertainment industries like Netflix, Spotify, Amazon, etc. to enhance customer experience, gain competitive advantage, retain customers and drive sales. The use of recommender systems reduces the browsing time of the user and improves user satisfaction and engagement with the product. Since recommender systems work in real-time, the recommendations can change according to the changing habits of the consumer.

Problem Description

The three primary categories of recommender systems are content-based, collaborative filtering, and hybrid approaches. Content-based filtering operates by building profiles of products and people and making product recommendations based on prior user experiences. The similarity between different users is the basis for collaborative filtering without requiring explicit profiling of users or products. It attempts to provide product or item recommendations based on data from a group of people who had previously shown an affinity for the same things. The hybrid technique combines two or more techniques, improving results and getting beyond each model's drawbacks. Although recommendation systems are widely used today, they have a number of drawbacks, including a cold start, scalability, sparsity, and many others.

The fundamental principle behind collaborative filtering, one of the most popular and comprehensive strategies in recommendation systems, is to anticipate which products a user would be interested in based on their preferences. When sufficient information is available, recommendation systems using collaborative filtering can provide an accurate forecast because this method is based on human preference. In the past, user-based collaborative filtering has shown great success in predicting consumer behavior as the most crucial component of the recommendation system. However, as the number of people and products has constantly increased, their broad use has exposed certain genuine issues, such as data sparsity and data scalability.

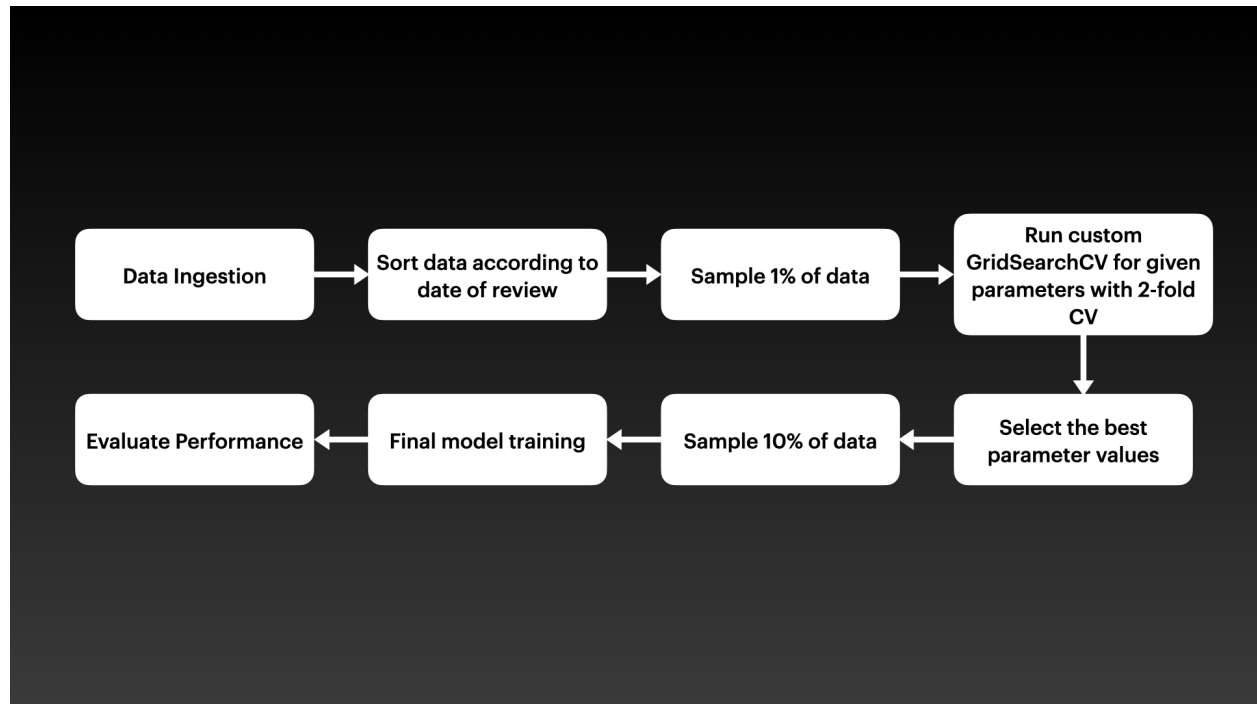
Matrix factorization is one of the most widely used models in recommender systems. It falls within the category of a latent factor model[18] using collaborative filtering, which searches for latent dynamics and structures in the observable ratings. Our main goal for this project is to build a recommendation system using matrix factorization methods as it can detect latent features. Taking advantage of this property, we aim to improve recommendation systems.

Related Works

Recommender systems were first mentioned in a technical report as a "digital bookshelf" in 1990 by Jussi Karlgren at Columbia University[1] and implemented at scale and worked through in technical reports and publications from 1994 onwards by Jussi Karlgren, then at SICS[2][3] and research groups led by Pattie Maes at MIT[4] Will Hill at Bellcore[5] and Paul Resnick, also at MIT[6][7] whose work with Group Lens was awarded the 2010 ACM Software Systems Award. Montaner provided the first overview of recommender systems from an intelligent agent perspective[8] Adomavicius provided a new, alternate overview of recommender systems[9] Herlocker provided an additional overview of evaluation techniques for recommender systems[10] and Beel et al. discussed the problems of offline evaluations[11] Beel et al. have also provided literature surveys on available research paper recommender systems and existing challenges[12][13][14].

Methodology

The pipeline built for the project is summarized below:-



Each of the steps followed is explained in detail in the following subsections:-

Data Ingestion: About Dataset

We worked with the Netflix Prize Competition dataset. The dataset was released by Netflix in 2006 for a competition they were hosting. The participants were required to build a recommendation system using their dataset, and lower the RMSE by 10% compared to Netflix's RMSE at that time. The winner would get \$1,000,000 as an award. Ever since the competition was announced, the dataset has become very popular. It is one of the gold standard datasets for training a recommendation system model because it contains millions of real-world data regarding customer preferences and reviews. The dataset is currently available on Kaggle.

Talking about the dataset in a little more detail, it contains 100,480,507 reviews (rows) which encompass 17,770 movies and 480,189 users. With an abundance of data at this scale, it was a clear choice for us to use this dataset to build one of the best-performing recommendation systems. The dataset consists of four attributes:-

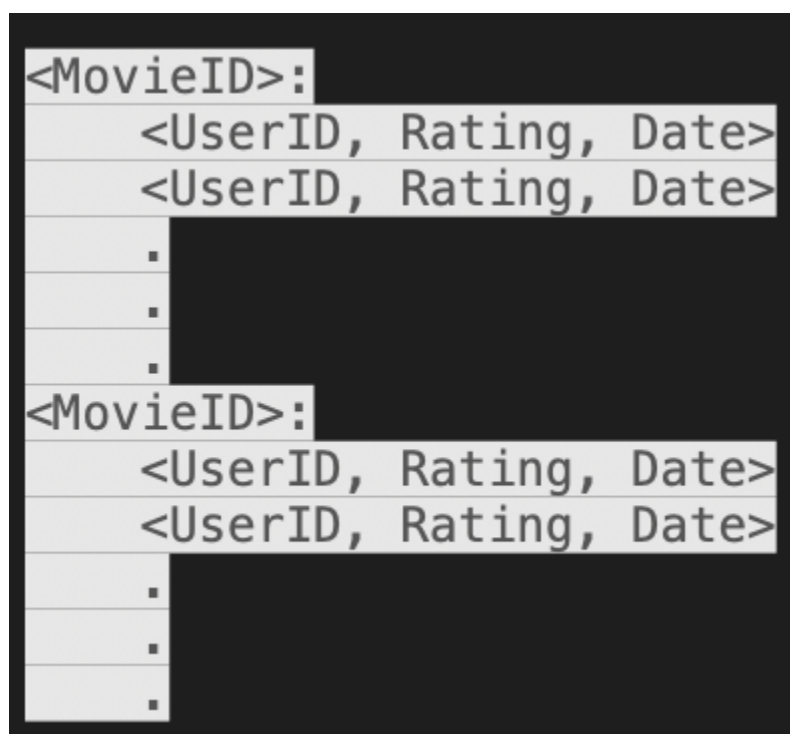
- UserID: unique key identifying the user who logged the review
- MovieID: unique key identifying the movie about whom the review is logged
- Date: the date when the review was logged
- Rating: the rating given by UserID to MovieID on Date. The value of this attribute belongs to {1, 2, 3, 4, 5}.

Given the size of the dataset, and the computing capabilities of our personal computers, we are working with 1% of the dataset for hyperparameter tuning and 10% of the dataset for the final model training.

Data Pre-Processing and Aggregation

The dataset provided by Netflix is not in directly usable form. Many preprocessing steps need to be followed to aggregate the data into a final usable dataset.

The first step is to resolve the hierarchical structure of data. In the original dataset, the reviews are collated according to the MovieID. So, all reviews for a particular movie are placed together.



<MovieID>:
<UserID, Rating, Date>
<UserID, Rating, Date>
▪
▪
▪
<MovieID>:
<UserID, Rating, Date>
<UserID, Rating, Date>
▪
▪
▪

We need all the attributes in a single row and no such hierarchy. Essentially, we need to flatten the dataset. To resolve this, we wrote a Python function that generates data files in the required format. The given dataset is split across 4 TXT files, so we merged all the flattened data into a single CSV file for ease of use.

The dataset also needs to be sorted according to the Date feature. It is so because all the rows share a temporal relationship which will be further elaborated upon in the Temporal Folds: Custom GridSearchCV section. This sorting is performed using the `sort_values()` method provided by Pandas.

In addition to the four features, we will need a fifth derived feature to easily create folds for cross-validation. This fifth feature is Year, which is extracted from the Date.

Features
UserID
MovieID
Ratings
Date
Year

The Math

Matrix factorization[17] is one of the most ground-breaking techniques applied to recommendation systems. It is deeply rooted in linear algebra, and hence to understand how the model works, it is quintessential to understand the math behind the model.

For each user i , we create a vector p that contains a quantified value of i 's interest in a particular quality of a product. This quantified value is derived from i 's reviews. Similarly, for each product j , we create a vector q that contains a quantified value of different qualities (genres, cast, or any distinguishing factor for that matter), the same qualities using which the user vectors are created.

The product $q^T p$ gives us the rating matrix, r . The Netflix dataset already represents this r . The goal is to create p and q vectors for all users and products, such that their $q^T p$ which gives us r' is as close to r as possible. We need to minimize the difference between r' and r .

The Matrix Factorization method is a latent factor method because it helps us find values for quantities that were not measured earlier. For example, if a user has never seen a mystery movie before, we do not know the user's preference for that genre. But, with this method, we find the value for all qualities for all users and products. Hence, we can even predict how much a certain user will prefer a product with a certain quality even if the user has not rated for that particular quality.

The intuition behind this method is similar to that of Single Value Decomposition (SVD). For example, consider the famous mathematical problem of splitting a large number into its two

prime factors. The solution to this problem has large implications for many modern security systems, especially in finance and banking.

What happens in case we do not have a perfect r ? Often in real-world data, there are many missing values. One solution is to perform data imputation. Numerous techniques like replacement with mean, median, or even 0, can be used. But, modern recommendation systems rely on regularization, instead of imputing missing values. Considering regularization, the final equation becomes,

$$\min_{q^*, p^*} \sum_{(u,i) \in \kappa} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2)$$

The Model

We built our model using the PyTorch framework[15]. The framework provides convenient functions and objects to build an Embedding model, which is what we need here. The vectors are represented as embeddings for building the model. An embedding layer is created using these embeddings.

We have selected Mean Squared Error as our criterion for the model. It is calculated by taking the average of the squared difference between actual and predicted values. PyTorch provides an implementation for this criteria, *torch.nn.MSELoss()*, hence it was convenient to include it in our model.

The learning algorithm we have used to optimize the model is Stochastic Gradient Descent (SGD). The advantage of this method is that it is one of the faster Gradient Descent (GD) algorithms, and it brings down the computation time for what is already a large training dataset. SGD is faster than other variations of GD algorithms because it randomly selects only 1 sample for a single iteration, whereas other variations like Batch GD select more samples per iteration which increases their compute time.

For hyperparameter tuning, we have set the following search space:-

```
parameters = {
    'batch size': [1024, 2048, 4096, 8192],
    'epoch': [10, 50, 100],
    'lr': [0.01, 0.1]
}
```

We are fine-tuning three hyperparameters:

- Batch Size: The number of samples that will be passed through the model at a time. The training data is divided into chunks where each chunk contains samples equal to the batch size.

- Number of Epochs: The number of times the model will iterate through the entire training data
- Learning Rate: Controls how much the model parameters (weights) are impacted with respect to the loss gradient.

Temporal Folds: Custom GridSearchCV

GridSearchCV is a standard technique used for determining the best hyperparameters for a given model, chosen from a fixed search space. GridSearchCV consists of two components - GridSearch and Cross Validation (CV). GridSearch tries out all the different possible combinations of hyperparameter values using cross-validation to determine the performance of each combination.

CV involves dividing the entire training dataset into groups called folds. A k-fold CV means the training dataset is divided into k groups (folds). Then, the model is trained on k - 1 folds, and the kth fold is used to determine the performance of the model. Each fold becomes the kth fold once, i.e., it is not trained on. The average of these k runs is then considered the overall performance of that particular combination. This process is repeated for each combination, and the combination that gives the best results is chosen for the final model training.

The problem with the standard GridSearchCV method provided by scikit-learn is that it is very difficult to control how the folds are created. For problems like recommendation systems, random sampling for creating the folds is not an option. It is so because we need to take into account the temporal relationship between rows. It does not make sense to use reviews logged at a later date to predict ratings for an earlier date. This is also the reason why we had to sort the data according to Date in the Data Pre-Processing and Aggregation part. Hence, we created our custom GridSearchCV function which creates temporal folds based on our logic.

We are making folds based on the Year feature, which is why we had it extracted earlier. Our folds are rolling in nature, i.e., for the first run we used *trainingYears* = [1999, 2000] and *testingYear* = 2001. In the next run, the *TestingYear* is subsumed in *trainingYears*, so *trainingYears* = [2000, 2001] and *testingYear* = 2002. The rolling continues so on. We built our function for 4-fold CV, but considering the compute time, we have used 2-fold CV for the results.

1999	2000	2001				
	2000	2001	2002			

Training

Testing

Results

Root mean squared error (RMSE) is a metric used to evaluate the performance of a model. It is a measure of how well the model fits the data and how close the predicted values are to the observed values. The smaller the RMSE, the better the model is at predicting the data. The RMSE is calculated by taking the difference between the predicted and observed values, squaring them, and then taking the square root of the average of these values. RMSE is a good judge of model performance because it is relatively easy to interpret.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

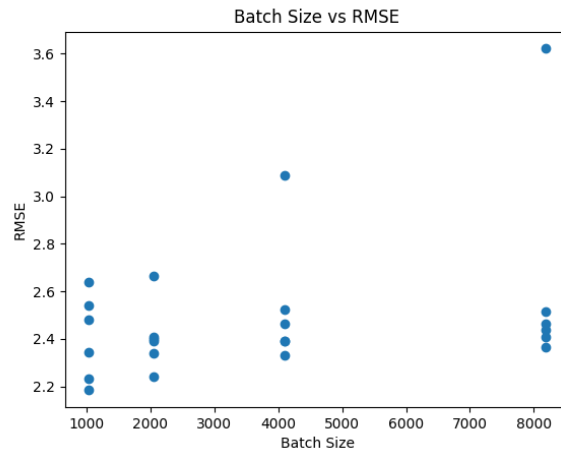
Netflix's RMSE	0.9514
Required RMSE for Grand Prize	0.8563
Authors' RMSE	0.8614
Our RMSE	1.3339

Netflix's RMSE back in 2006 was 0.9514. The competition required participants to bring it down to 0.8563. The authors whose paper we have followed were able to bring it down to 0.8614 at the time of publishing. Our RMSE in this project is 1.3339.

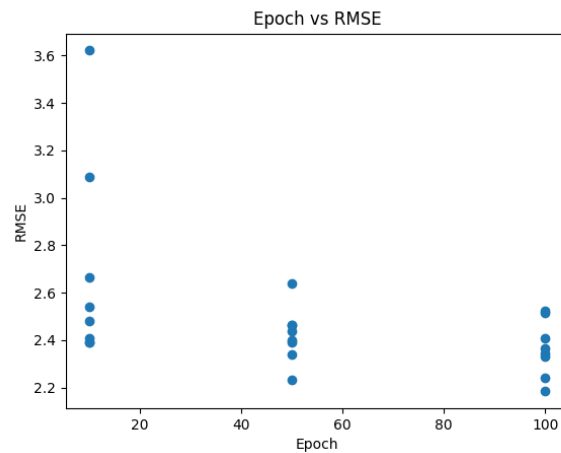
The reason for a higher RMSE in this project is very clear. Because it is such a large dataset with more than 100 million rows, we could only utilize 10% of it. We strongly believe that if we can utilize the entire dataset, we can bring down our RMSE drastically.

For each combination tested in GridSearchCV, we plotted the RMSE against different hyperparameters.

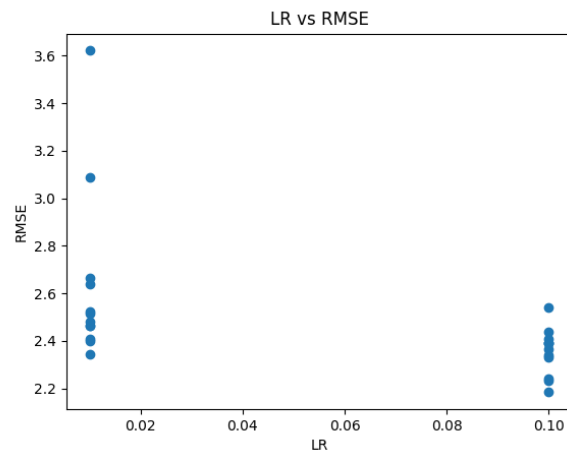
From the Batch Size vs. RMSE graph, Batch Size = 1024 has the minimum average RMSE.



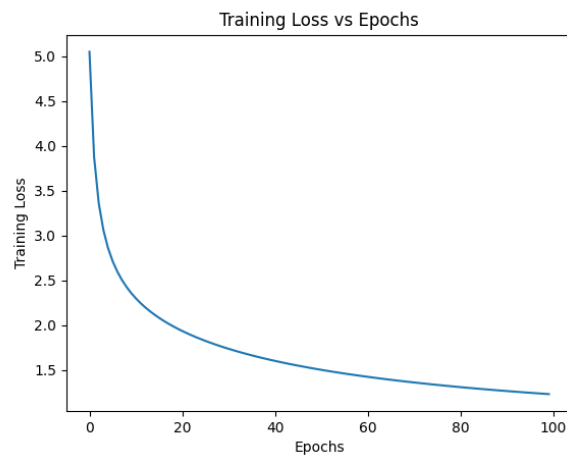
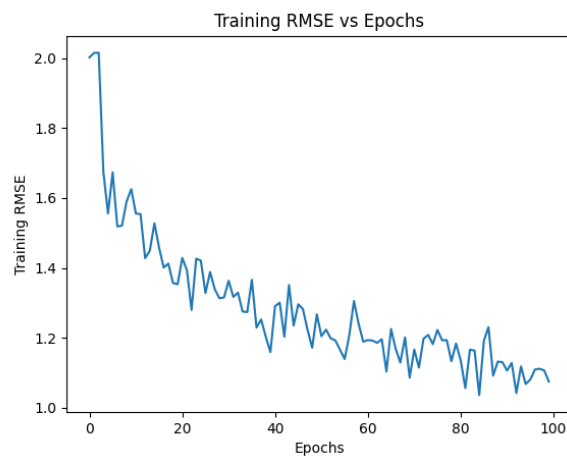
Running a model for a reasonable number of epochs is crucial for getting the best performance. Here, we could not try beyond 100 epochs (which is very small for such a large dataset) because it is extremely time-consuming on a personal laptop. Hence, we think that the RMSE curve will still go down if we run for more than 100 epochs.



A learning rate of 0.1 gives a lower RMSE, hence that is selected.



Tracking the Training RMSE and Training Loss with each epoch clearly shows a descent that will continue if trained for more epochs. Hence, our model will do a much better job with access to advanced computational resources like GPUs.



Conclusions

Matrix factorization is a powerful technique for building a recommendation system that can accurately predict user preferences and generate useful recommendations. This technique has been widely applied in a variety of applications, from movie recommendation systems to music streaming services. Matrix factorization is also an important component of many machine learning algorithms, such as collaborative filtering and deep learning. With the continuing development of artificial intelligence and machine learning, matrix factorization will continue to be an important tool for building powerful and accurate recommendation systems. Recommendation systems create immense value for a variety of businesses because of their satisfactory predictions.

Matrix factorization is also very efficient for such large datasets. The process often involves decomposing the data into two matrices, each representing one set of variables. From these matrices, relationships between the variables are identified and used to generate recommendations. This makes it easier to personalize the user experience and provide tailored suggestions. Furthermore, it is easily scalable, i.e., factorization to be run on multiple machines in parallel, which can significantly reduce the time it takes to complete the factorization. Additionally, by using a distributed system, the amount of data that can be handled is not limited by a single machine's memory or processing power.

Finally, by leveraging existing infrastructure, such as AWS or Google Cloud, additional scalability can be achieved by dynamically adding or removing resources as needed. Moreover, even when data is sparse, matrix factorization can still be used to identify patterns in the data and to gain insights. This is because matrix factorization is based on linear algebra, which can identify relationships between the elements of the matrix even when some of the data is missing.

Future Works

We can improve our custom GridSearchCV function's performance by executing code more efficiently using parallel computing. Initiating multiple threads in the same processor unleashes the constraint of sharing data between the jobs more effectively. Here, thread-based parallelism can decrease our workload in the background especially while dealing with big data. Parallel processing will save time, and money, solve more complex or bigger problems and leverage remote resources.

For this project, we have used only 10% of the total dataset available as using just the CPU is not going to cope with such massive data. Disregarding the limitations of personal computers and using GPU will allow us to deal with larger datasets resulting in reduced RMSE. Parallel programming and GPU[16] will also allow us to integrate metadata in matrix factorization which will result in better recommendations.

The sequential, dynamic user-system interaction and long-term user involvement can both be handled by Reinforcement Learning. Although the concept of using Reinforcement Learning for recommendations is not new and has been around for over 20 years, it was not very practical, mostly due to issues with classic Reinforcement Learning algorithms' scalability. Since the invention of deep reinforcement learning (DRL), which made it possible to apply RL to the recommendation issue with enormous state and action spaces, a new trend has, however, developed in the field.

References

- [1] Karlgren, Jussi. 1990. "An Algebra for Recommendations." Syslab Working Paper 179 (1990).
- [2] Karlgren, Jussi. "Newsgroup Clustering Based On User Behavior-Recommendation Algebra." SICS Research Report (1994).
- [3] Karlgren, Jussi (October 2017)A digital bookshelf: original work on recommender systems". Retrieved 27 October 2017.
- [4] Shardanand, Upendra, and Pattie Maes. "Social information filtering: algorithms for automating “word of mouth”." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 210-217. ACM Press/Addison-Wesley Publishing Co., 1995.
- [5] Hill, Will, Larry Stead, Mark Rosenstein, and George Furnas. "Recommending and evaluating choices in a virtual community of use." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 194-201. ACM Press/Addison-Wesley Publishing Co., 1995.
- [6] Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergström, and John Riedl. "GroupLens: an open architecture for collaborative filtering of netnews." In Proceedings of the 1994 ACM conference on Computer supported cooperative work, pp. 175-186. ACM, 1994.
- [7] Resnick, Paul, and Hal R. Varian. "Recommender systems." Communications of the ACM 40, no. 3 (1997): 56-58.
- [8] Montaner, M.; Lopez, B.; de la Rosa, J. L. (June 2003). "A Taxonomy of Recommender Agents on the Internet". Artificial Intelligence Review. 19 (4): 285–330. Doi:10.1023/ A:1022850703159..
- [9] Adomavicius, G.; Tuzhilin, A. (June 2005). "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions". IEEE Transactions on Knowledge and Data Engineering.17 (6): 734–749. CiteSeerX 10.1.1.107.2790.doi:10.1109/TKDE.2005.99..
- [10] Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; Riedl, J. T. (January 2004). "Evaluating collaborative filtering recommender systems". ACM Trans. Inf. Syst. 22 (1): 5–53. CiteSeerX 10.1.1.78.8384.doi:10.1145/963770.963772..
- [11] Beel, J.; Genzmehr, M.; Gipp, B. (October 2013). "A Comparative Analysis of Offline and Online Evaluations and Discussion of Research Paper Recommender System

Evaluation"(PDF).Proceedings of the Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys) at the ACM Recommender System Conference (RecSys).

[12] Beel, J.; Langer, S.; Genzmehr, M.; Gipp, B.; Breiting, C. (October 2013). "Research Paper Recommender System Evaluation: A Quantitative Literature Survey(PDF). Proceedings of the Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys) at the ACM Recommender System Conference (RecSys).

[13] Beel, J.; Gipp, B.; Langer, S.; Breiting, C. (26 July 2015). "Research Paper Recommender Systems: A Literature Survey". International Journal on Digital Libraries.17 (4): 305– 338. doi:10.1007/s00799-015-0156-0.

[14] Waila, P.; Singh, V.; Singh, M. (26 April 2016). "A Sciento metric Analysis of Research in Recommender Systems" (PDF). Journal of Scientometric Research.5: 71–84. doi:10 .5530/jscires.5.1.10.

[15] Parts of the code were adapted from Minsang Yu's implementation. <https://github.com/FloweryK/Matrix-Factorization-Techniques-for-Recommender-Systems>

[16] Keckler, Stephen & Dally, William & Khailany, Bruce & Garland, Michael & Glasco, David. (2011). GPUs and the Future of Parallel Computing. Micro, IEEE. 31. 7 - 17. 10.1109/MM.2011.89.

[17] Y. Koren, R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009, doi: 10.1109/MC.2009.263.

[18] Latent Factor Models and Matrix Factorizations. In: Sammut, C., Webb, G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_887