MATH80600A project report

A Comparative Analysis of Recommendation Systems between Matrix Factorization & Deep Learning Techniques

Yahya Abdul Aziz
HEC
`yahya.abdul-aziz@hec.ca`

Franck Benichou
HEC
`franck.benichou@sciencespo.fr`

Yichen Lin
University of Montreal
`yichen.lin@umontreal.ca`

Abstract

With the explosive growth of e-commerce and social media, recommendation system has become crucial tools for many businesses. In recent years, deep learning has gained pervasive influence on multiple fields, including recommendation systems. This project aims to provide a comprehensive review of recommender systems. More concretely, we carry out a comparative experiment on the traditional and state-of-the-art deep learning recommendation methods. We used matrix factorization, Autoencoders(Autorec) and GNN(IGMC) methods. Our results showed that deep learning methods outperform the traditional methods particularly in larger datasets. Our codes are available at https://github.com/YichenLin/MATH-80600A-Project.

1 Introduction

With the ever-increasing amount of online information, recommender systems have become a go-to strategy to overcome information overload. Recommender systems facilitate personalization through e-commerce and have grown significantly in terms of public awareness since its rise in the nineties, in parallel to the Web debut. Indeed, many conferences and workshops are exclusively dedicated to this topic, such as the ACM Conference on Recommender Systems (RecSys). The most classical methods in recommender systems include collaborative filtering methods, knowledge-based methods, and content-based methods. These methods evolved to fit various scenarios, data domains, and contexts, such as time, location (ie. gps signal) and social media information. Numerous breakthroughs have been found such as the application of Matrix Factorization (MF), for instance with the movie recommendation during the 2006 Netflix contest. Conventional methods such as MF are considered state of the art and have been favored over content-based methods, due to dealing better with sparsity, scalability, and predictions. However, they still face several challenges such as the cold start problem, relatively high data sparsity, scalability issues with O(m*n) complexity in MF, difficulty to boost diversity of the recommended items, and the obstacles to learn latent implicit features. The last decade has witnessed the Deep Learning (DL) revolution with powerful applications in computer vision, speech recognition, and natural language processing. Indeed, DL approaches have overcome some of the obstacles faced by conventional recommender systems. These approaches can capture the non-trivial and non-linear user-item relationships, and they allow for the representation learning of more complex abstractions as data representations in the higher layers. It is to be noted that DL recommenders also face the following challenges: lack of interpretability, large data requirements, extensive hyperparameter tuning. As a result, we intend to compare the strengths and limitations of Matrix Factorization (MF), one of the state-of-the-art conventional recommender systems, with DL approaches such as AutoRec and GNN. Other DL algorithms have been studied but we limited our analysis to these two for brevity. First, we will undertake a survey of the literature and explain these methods. Then, we will briefly describe the datasets we used. Third we will describe our experiments with MF, Autorec, and GNC with the goal to optimize prediction rating in user-item context. Finally, we will discuss the results before touching upon the limitations.

2 Related Works and methodologies

2.1 Matrix Factorization

Matrix Factorization (MF) is one of the Collaborative Filtering approaches characterized by the reliance on both items and users' vectors as opposed to Neighborhood based approach [5] that relies on the preferences of the user's neighbor. The items and users' vectors are used to infer latent factors of item ratings. MF represents the relation between items and users and forms two low rank matrices. Finally, the multiplication of these two matrices helps us estimate the user's future preferences [2]. Mathematically, the rating prediction is:

$$\hat{r}_{ui} = x_u^T y_i = \sum_k x_{uk} y_{ki}$$

For prediction, the loss function is as follows with regularization terms on the users and items vectors:

$$L = \sum_{u,i \in S} r_{u,i} - x_u^T y_i + \lambda_x \sum_u \|x_u\|^2 + \lambda_y \sum_u \|y_i\|^2$$

A variety of measures exist to assess the rating prediction strength of MF techniques and other recommenders in general. There exist three types of metrics to evaluate the power of an algorithm: classification, prediction, and rank accuracy. The classification accuracy assesses the quality of the algorithm to differentiate good items from bad ones. Examples are Precision, Recall, ROC, AUC [6]. The prediction accuracy evaluates the difference between predicted rating and actual rating is the type of metric we chose. Typical prediction metrics are the MAE, normalized MAE, MSE, and RMSE. RMSE has been chosen across our experiments. Rank accuracy measures whether the algorithm sorts the recommended items like the user. Research in MF and Context Aware MF are abundant. Some commonly used MF techniques are Singular Value Decomposition [23], PCA, Probabilistic Matrix Factorization [22], and Non-Negative Matrix Factorization [21]. In specific settings, it was showed the SVD could be improved by adding biases to users and items. SVD++ was proposed by Koren et al. to use implicit feedback. It demonstrated a high accuracy but expensive computational costs. Tensor Factorization extends the two-dimensional MF problem into an n-dimensional version by including contextual information [9]. The multi-dimensional matrix is factored into a lower-dimensional one, where each contextual dimension, the item, and the user are represented with a lower dimensional feature vector [4]. Karatzoglou et al suggested a multiverse recommendation system by using CF methods on Tensor Factorization. Comparative analysis of Boolean Matrix Factorization (BMF) with SVD is undertaken by Akhmatnurov et al [1]. Experiments with the Boolean matrix dot product of binary matrices with or without contextual information with an SVD showed a higher precision for the BMF where the number of user neighbors is not high. Tensor Factorization based on a fuzzy mapping between the latent and contextual factors is proposed by Fang et al [7]. In their paper, movie tags and release time were used as contextual variables and led to a higher RMSE and HLU while diminishing the number of iterations by 25%.

2.2 Autoencoder

Despite its commendable performance, matrix factorization still suffers from data sparsity and cold start issues. Many approaches have been proposed to solve these problems but most of them fall short.This is because those approaches fail to capture user preferences and item features, which are not readily available in the data set and must be implicitly learned. Given that it is an unsupervised task with very high dimension data, autoencoder was introduced in the scenario. A vanilla autoencoder consists of an input layer, a hidden layer and an output layer [3]. In particular the input layer and successively restrictive hidden layers formed the encoder. We have an encoded representation of the input data that can be in a lower dimensional form or in a different form as we will see later. The second half of the autoencoder is the decoder. It is the reverse of the encoder. It reconstructs the original input data. Below is an illustration of autoencoder:

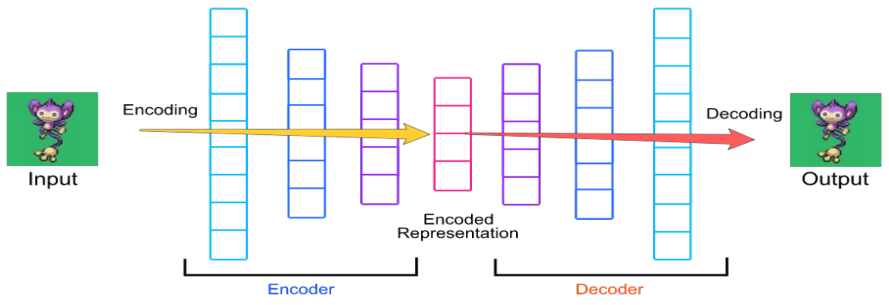

Figure 1: Autoencoder

There is a linear weighting at each layer followed by nonlinear activation function neuron. For simplicity we show the transformation for the encoder and decoder as

$$h = f(x) = s_f(Wx + b)$$
$$x' = g(h) = s_g(W'h + b')$$

In this scenario, the cost function with regularization is

$$RE_{AE(x,x')} = (\sum_x SE|CE(x, x')) + \lambda * Regularization$$

With the same purpose, [11] proposed a deep autoencoder architecture, where the encoder and the decoder are multi-layer deep networks.

An implementation of autoencoders for recommender systems called Autorec tackles the collaborative filtering problem [26]. In this form the autoencoder takes as input a matrix of user item interactions with the ratings filled in with the dimension of M by N. Thus it takes the user ratings and items as input and reduces the dimensionality by forcing it through a bottleneck. Then it reconstructs the data into the output layer. It can be done in two ways: the first is item based and the other is user based. In the user based method, users a ranked by similarity and items by similar users are recommended to the current user. In the item based methods, similar items to those purchased by the user is recommended . A variational autoencoder (VAE) assumes the prior of the latent variable zz is from some parametric family of distributions $p\theta(z)p\theta(z)$. The observed data is generated from the posterior distribution $p\theta(x|z)p\theta(x|z)$. The true posterior of zz is $p\theta(z|x)p\theta(z|x)$, but is intractable. To get $p\theta(z|x)p\theta(z|x)$, we use another parametric family $q\phi(z|x)q\phi(z|x)$, which is tractable, to approximate it.[15] This is an unsupervised latent variable model that instead of learning a low dimensional representation of the input data, encodes the original input as a probability distribution in the encoder portion. In the decoder portion, it samples from that probability distribution to rebuild the original input data.

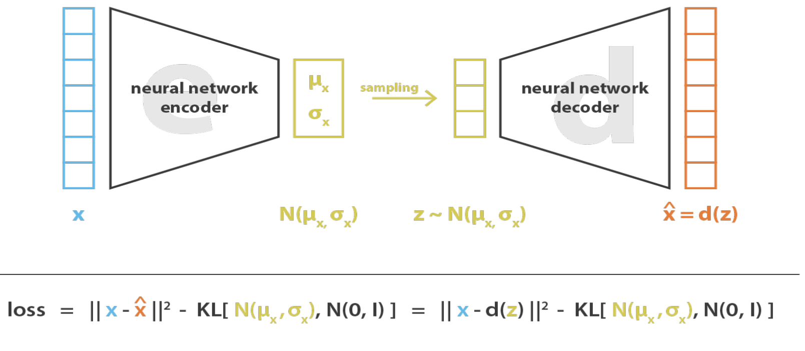
neural network encoder
μ_x
σ_x
sampling
neural network decoder
x
$N(\mu_x, \sigma_x)$
$z \sim N(\mu_x, \sigma_x)$
$\hat{x} = d(z)$
loss = $\|x - \hat{x}\|^2$ - KL[$N(\mu_x, \sigma_x)$, N(0, I)] = $\|x - d(z)\|^2$ - KL[$N(\mu_x, \sigma_x)$, N(0, I)]

Figure 2: Variantional autoencoder

2.3 GNN

Recently, graph neural network (GNN) which can integrate node information and topological structure have been proven quite useful for recommender systems, since most of the information has graph structure essentially. We first briefly summarize four typical GNN frameworks adopted in the field of recommendation [28], then we give a brief introduction to the IGMC we will be using in our experiment.

2.3.1 GNN frameworks

- **GCN** [16] Graph Convolutional Networks is a type of convolutional neural network that can work directly on graphs and take advantage of their structural information. It shares filter parameters over all locations in the graph. GCNs can be categorized into 2 major algorithms, Spatial Graph Convolutional Networks and Spectral Graph Convolutional Networks.
- **GraphSage** [10] is an inductive framework that leverages node feature information to efficiently generate node embeddings for previously unseen data. It is used to generate low-dimensional vector representations for nodes, and is especially useful for graphs that have rich node attribute information.
- **GAT** [27] Graph attention network used masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions. By stacking layers in which nodes are able to attend over their neighborhoods' features, GAT enables specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation (such as inversion) or depending on knowing the graph structure upfront.
- **GGNN** [17] Gated Graph Sequence Neural Networks adopts a gated recurrent unit (GRU) in the update step. It can incorporate the full graph structure without loss of information.

2.3.2 IGMC

IGMC (Inductive Graph-based Matrix Completion)[29] trains a GNN based on 1-hop subgraphs around (user, item) pairs generated from the rating matrix and maps these subgraphs to their corresponding ratings. IGMC is inductive, which means it can generalize to users/items unseen during the training, and can even transfer to new tasks.
IGMC adopts the relational graph convolutional operator (R-GCN) [25] as GNN's message passing layers, which has following form:

$$x_i^{l+1} = W_0^l x_i^l + \sum_{r\in R} \sum_{j\in N_r(i)} \frac{1}{|N_r(i)|} W_r^l X_j^l$$

where x_i^l denotes node i's feature vector at layer l, W_0^l and $\{W_r^l | r \in R\}$ are learnable parameter matrices. We stack L message passing layers with tanh activations between two layers. Next, we pool the node representations into a graph-level feature vector. After getting the final graph representation, we use an MLP to output the predicted rating:

$$\hat{r} = w^T \sigma(Wg)$$

where W and w are parameters for the MLP which map the graph representation g to scalar rating $\hat{r}$. And σ is ReLU activation function.

3 Datasets

3.1 MovieLens

The first dataset is the movielens set, it contains three kinds of information: 5-star rating on movies by users; demographic characteristics of users and descriptive characteristics of the movie. We plan to use MovieLens Latest Datasets(Small version for prototyping and Full version for training/testing). The dataset contains 6 files: genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv and tags.csv. For our experiments on the MovieLens data, we restricted our investigation on the ratings set and did not add contextual information. The 100k, 1 Million and the 25 Million (later downsized to 2.5 Million) movielens datasets were used to compare Matrix Factorization performance with Deep Learning approaches such as AutoRec and GNN.

3.2 Amazon

For the second dataset, we have opted for a product recommendation and in particular we have chosen to go with the Amazon Product dataset. The dataset is simplistic and clean with no metadata regarding the products included. In particular we are going with the magazine subscriptions as it is of a moderate size (89689 reviews) which allows to do adequate testing with our limited computational capacity. The data has two forms, one which includes information regarding the reviewer and the textual review and another which only has the user id, the product id, the rating and a timestamp. For our purposes we are only interested in the user, the product and the rating assigned which is on a 1-5 likert scale. Furthermore, magazine subscriptions is an intuitive and easily human explainable domain with many repeat customers as compared to other fields in which most buy a single product.

4 Experiment

4.1 Evaluation Metric

For all our experiments we minimize the root mean squared error (RMSE) between the predictions and the ground truth ratings. RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum_{(i,j)\in\tau|X(i,j)-\hat{X}(i,j)|}}{|\tau|}}$$

where τ denotes the set of ratings we try to predict, $X(i,j)$ denotes the rating user i gave to item j and $\hat{X}(i,j)$ denotes the predicted rating from u_i to v_j.

4.2 Data Exploration

4.2.1 Movielens dataset

Before deep diving in the prediction problem, we quickly explored the 100k movie lens datasets to better understand the ratings, items, and users. We focused on this dataset because it shows movie, user, and ratings trends that are like the trends of the bigger datasets. It is worth noticing that the average age of the users is 42 years old with the youngest being 7 and the oldest being 73. The most popular genres are drama and comedy. Also most of the releases occurred in the 1990's and there is incomplete data at the end of the decade. We also observed that most movies got released on Fridays and week-end which is logical.
68% of the movie raters are male. They are family oriented with 62% of them being either between 30 to 60 or less than 15. There is no significant difference in the ratings distribution given by Male or Female.

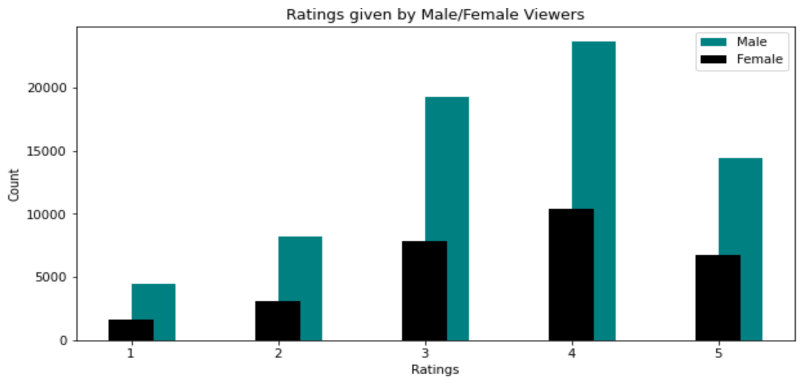

Figure 3: MovieLens Ratings distribution per gender

4.2.2 Amazon dataset

A quick exploratory data analysis of the Magazine Subscription reviews on Amazon reveals that a majority of the reviews are positive 4-5 with a mean of 4.04. There is however a bump at a rating of

1. This is intuitive as a regular customer only leaves a review if they either really liked or disliked the product. We also observe that the majority of products only have a small number of reviews which increases the sparsity of our data. The great majority of items have very few ratings.

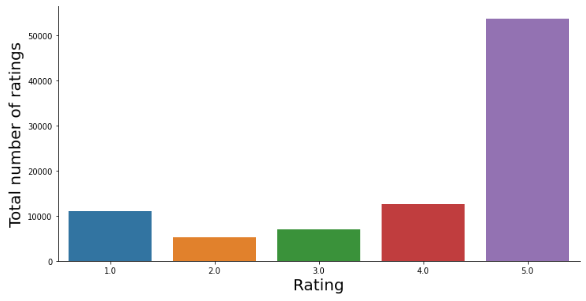

Figure 4: Ratings distribution

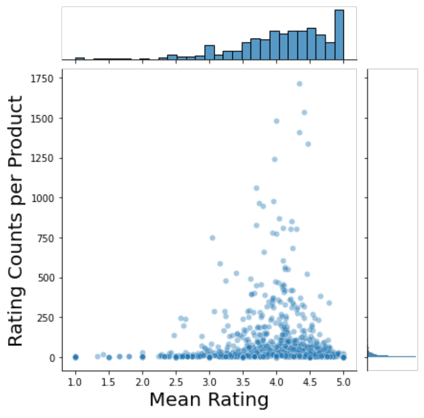

Figure 5: Ratings distribution

4.3 Matrix factorization

4.3.1 Movielens

We implemented from scratch the Alternating Least Squares MF technique along with an SGD based MF across the 100k and 1 million movie lens. We did not implement MF techniques on the 2.5 million dataset because it too computationally expensive on GCP. Indeed, fine tuning both MF techniques would take around 1 hour on the 1 million dataset. Across the 2 datasets, every user has rated at least 20 movies, leading to a sparsity level of 6.3% in the 100k dataset. We built the ratings matrix with users in the rows and the movies in the columns, and the elements in the matrix being the ratings. We split the data between training and validation set by removing 10 ratings per user from the training and placing them in the validation set. We implemented the proper dot product of the two user and item vectors (for prediction), the ALS loss function with L2 regularization terms on the users/item vectors, and the ALS derivation. Then we fine tuned the right number of iterations, latent factors k, and the regularization terms through a grid search to minimize the validation RMSE. For the SGD based MF, we implemented the SGD derivation and an SGD Loss function and optimized the MF SGD model hyperparameters through a grid search to minimize the validation RMSE. The MF SGD hyperparameters are: number of iterations, learning rates, number of latent factors, bias terms, and L2 regularizations.

4.3.2 Amazon

We preprocess the dataset by removing all the extra metadata such as the textual review and reviewers' names. Only information such as userIds, productids and ratings are kept.
KNN algorithm searches for the nearest items to the current user trends, thus it will recommend items similar to what the user has consumed in the past. It does not compare the items consumed by one user with another user.
We set hyperparameter k=5 to make sure we get items that are close to the target. As most data has few reviews, setting k to 5 also prevents our model from underfitting. We also use the pearson baseline as our measure of similarity between items, since the shrinking coefficient also helps with avoiding overfitting.

4.4 Autoencoder

4.4.1 Movielens

Like for the MF techniques, we preprocessed the ratings, movies, and users movie lens datasets, across the 100k, 1 Million, and 25 million restricted to 2.5 million (due to out of memory error) to construct the appropriate rating matrix with the users in the rows and the movies in the columns and the ratings in the matrix. The implementation was done with the tensor flow package with GPU activated in the GCP virtual machine. The input layer is of size 3005. There is 1 encoder layer with 500 hidden neurons, 1 decoder layer with 500 hidden neurons, and 1 latent space with 128 neurons. The training/validation split across the 3 datasets was 75%/25%. We tweaked the following hyperparameters and applied them across the 3 datasets because fine tuning was quite computationally expensive and lengthy : number of layers, optimizer, learning rate, batch size, the activation function, the number of epochs and the number of hidden neurons. Code referenced from [13].

4.4.2 Amazon

For the deep learning portion of our analysis. We have fit an autoencoder with an input layer of size 2015, 1 encoder layer with 512 hidden neurons, a latent space layer with 256 neurons, a dropout layer with probility of 0.8, 1 decoder layer with size 512 and finally a dense output layer of size 2015. Furthermore, the use of a dropout layer with a probability of 0.8 is in order to increase the generalizability of the model as the potential to overfit is large. There is a large amount of parameters to fit in the model (2328799) despite its inherent simplicity. However with a dataset of this size, the training times are reasonable at about 10 minutes on the free version of Colab. It should be noted that the chosen activation function for the encoder, decoder and latent space is selu as it was found to have the best overall performance. Code referenced from [18] [12] [19] [24].

4.5 GNN

We conduct experiment of GNN on MovieLens-100K dataset. The data was randomly splitted into 90% and 10% train/test sets. IGMC model was implemented base on pytorch_geometric [8]. For the user-item graph, we used 1-hop enclosing subgraphs and achieved good performance. Though 2 or more hop subgraphs can slightly increase the performance, they also take much longer to train, according to the experiment from [29].
The final architecture of our IGMC uses 4 R-GCN layers with 32,32,32,32 hidden dimensions. The final MLP has 128 hidden units with a dropout rate of 0.5. The model was trained using Adam optimizer [14] with a batch size of 32 and learning rate of 1e-3.
The training time of GNN on MovieLens-100K took about 8 hours on a single 8G GPU. We expect the training time on MovieLens-1M to take much longer time. It would only be feasible to use multiple GPUs to achieve reasonable training time. Due to computation limitations, we only tested our IGMC model on MovieLens-100K. Code referenced from [30][20].

5 Results and Discussion

Below table 1 is the final results table of all the recommendation methods we experimented, evaluated in RMSE. Noted that the text in the brackets are the different methods we implemented the matrix factorization technique on. Table 2 is the hyperparameter table for all our models.

	MovieLens 100K	**MovieLens 1M**	**MovieLens 2.5M**	**Amazon 89K**
Matrix Factorization 1	2.95(ALS)	2.99(ALS)	NA	1.34(KNN)
Matrix Factorization 2	0.96(SGD)	0.95(SGD)	NA	NA
Autoencoder(AutoRec)	1.02	0.90	0.78	0.0847
GNN(IGMC)	0.9041	NA	NA	NA

Table 1: Final Results(RMSE)

	MovieLens100K	MovieLens1M	MovieLens2.5M	Amazon89K
Matrix Factorization 1	Regularization = 100, Latent Factors = 80, 25 iterations	Regularization = 100, Latent Factors = 80, 50 iterations	NA	K=5 Metric: Pearson_baseline Type: Item_based
Matrix Factorization 2	α = 0.001, Latent Factors = 40, 200 iterations, regularization = 0.01	α = 0.01, Latent Factors = 40, 200 iterations, regularization = 0.01	NA	NA
Autoencoder(AutoRec)	Adam Optimizer, Layer = 1, α = 0.001, batch size = 512, λ = 1, hidden neurons = 500, epochs = 500, activation = sigmoid			Adam Optimizer, α = 0.0001, Activation: SELU, Dropout=0.8, Layers = 4
GNN(IGMC)	Adam Optimizer 4 R-GCN(32,32,32,32) 1 MLP(128) 1-hop enclosing subgraph			

Table 2: Hyperparameter table

5.1 Movielens

We observed that the MF ALS was consistently outperformed by the MF SGD across the 100k and 1 million movie lens datasets. The MF techniques do not tend to improve their performance with bigger datasets. The optimal hyperparameters for each technique are available in the summary table. In terms of RMSE, the AutoRec method is at par with MF SGD for the 100k movie lens dataset but beats MF SGD on the 1 million with a RMSE of 0.90 vs 0.95 for MF SGD. Further, the more data, the better the validation RMSE of the AutoRec. In terms of computational time, we noticed that MF was computationally expensive in terms of fine tuning by taking 1 hour to fine tune (ALS & SGD) on the 1 million dataset. Further, the more the data the more the computational training time for Autorec : 27 minutes on the 100k dataset, 1 hour and 27 minutes on the 1 million dataset and 8 and 13 hours and 39 minutes on the 2.5 million dataset. Finally, we conclude that MF SGD provides the best performance amongst MF but is increasingly computationally intensive with larger datasets while Autorec provides better performance in larger datasets, due to the implicit learning of latent features.

The GNN and AutoRec both outperforms MF for MovieLens 100K dataset, with RMSEs at 0.9041 and 1.02. This is because these deep learning models are able to capture implicit features without using metadata and further feature vectors. While MF learns a linear latent representation, DL models can learn a nonlinear latent representation through activation function. This allows the model to have much bigger capacity. Besides, the IGMC(GNN) utilizes the graph properties in the user-item data by enabling inductive matrix completion, it increases generalization and performance. Thus the IGMC achived the lowest RMSE among all models.

5.2 Amazon

The autoencoder shows a performance orders of magnitude better than the KNN. This gap only widens with the increasing size of the dataset as the neural network learns more accurate implicit features in the data in order to make accurate rating predictions. Furthermore, the knn is non parametric and does not have a learning portion and thus has a relatively static performance which is after a threshold dataset size ceases to improve. With a larger dataset, the memory requirements of the KNN become prohibitively large as opposed to the autorec whose number of parameters does not change significantly (only the input layer) and thus with a rising dataset size and in the absence of accurate and descriptive feature vectors the autorec becomes the more attractive choice.

6 Conclusion and Future work

In this project, we tried different recommendation methods on two popular datasets and compared their performances. Particularly, we used matrix factorization as the traditional recommendation method, and Autorec and GNN(IGMC) as the deep learning methods. Our experiments reveal that the deep learning methods have much better performance. This is because these methods are efficient in capturing meaningful latent features such as collaborative signals and sequential patterns.
Currently our models only incorporates the primary data into recommendation, while real-world datasets are associated with rich side information on users and items. Going further, we could explore more complex feature interactions with user-item pairs.

References

[1] Marat Akhmatnurov and D. Ignatov. Context-aware recommender system based on boolean matrix factorisation. In *CLA*, 2015.

[2] Marharyta Aleksandrova, Armelle Brun, Anne Boyer, and Oleg Chertov. Identifying representative users in matrix factorization-based recommender systems: Application to solving the content-less new item cold-start problem. *J. Intell. Inf. Syst.*, 48(2), April 2017.

[3] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[4] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, page 301–304, New York, NY, USA, 2011. Association for Computing Machinery.

[5] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering, 2013.

[6] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Trans. Web*, 5(1), February 2011.

[7] Yaoning Fang and Yunfei Guo. A context-aware matrix factorization recommender algorithm. In *2013 IEEE 4th International Conference on Software Engineering and Service Science*, pages 914–918, 2013.

[8] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[9] Anjali Gautam, Parila Chaudhary, Kunal Sindhwani, and Punam Bedi. Cbcars: Content boosted context-aware recommendations using tensor factorization. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 75–81, 2016.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[11] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[12] Shashank Kapadia. Recommendation system in python: Lightfm, Feb 2020.

[13] khanhnamle1994. https://github.com/khanhnamle1994/metarec/tree/master/autoencoders-experiments/autorec-tensorflow.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[17] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2017.

[18] mandeep147. mandeep147/amazon-product-recommender-system.

[19] marlesson. https://github.com/marlesson/recsys_autoencoders/blob/master/recommender.py.

[20] muhanzhang. https://github.com/muhanzhang/igmc.

[21] Roozbeh Rajabi, Mahdi Khodadadzadeh, and Hassan Ghassemian. Graph regularized nonnegative matrix factorization for hyperspectral data unmixing, 2011.

[22] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 880–887, New York, NY, USA, 2008. Association for Computing Machinery.

[23] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system - a case study. 2000.

[24] saurav9786. Recommender system using amazon reviews, Nov 2020.

[25] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[26] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[28] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. Graph neural networks in recommender systems: A survey, 2021.

[29] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks, 2020.

[30] zhoujf620. https://github.com/zhoujf620/motif-based-inductive-gnn-training.