Recommender System for Amazon Electronics Products using Machine Learning and Deep Learning Techniques

Team: Jingxuan Bao, Yaoqi Deng, Guanlin Huang
Repositroy Link: https://github.com/Jingxuan-Bao/Amazon_Product_Recommendation
Project Mentor TA: Crescent

Abstract

We developed an electronics product recommender system using machine learning and deep learning techniques. Our work is based on previous Kaggle works with innovation in utilizing the time stamp feature and exploring the use of deep learning in developing the recommender system. We trained and tested our models, including KNN, SVD, LSTM, and neural network, on an extensive dataset of Amazon electronics product reviews. By optimizing the KNN and SVD model parameters, we achieved promising prediction accuracy, while our results in the more sophisticated LSTM and neural network showed less accurate results due to overfitting, highlighting the importance of tuning the parameters for a better fit. Future work can explore the use of season-based timing to capture users' seasonal preferences and minimize ethical concerns related to user privacy by encoding personal information. Furthermore, training the models on a small subset of the data can improve speed and reduce computational costs.

1) Introduction

In this work, we focus on the development of an Amazon electronics product recommender system using machine learning and deep learning tools. We trained and tested our model on an extensive dataset of Amazon electronics product reviews comprising 7,824,482 entries, which include user IDs, product IDs, ratings, and timestamps. The dataset can be found on Kaggle at https://www.kaggle.com/datasets/pritech/ratings-electronics. To address the issue of data sparsity, we examine the distribution of user and product ratings in the dataset. Based on our observations, we decide to preprocess the data by filtering it to only include users with at least 5 ratings and products with at least 10 ratings. This filtering process not only helps alleviate the sparsity problem but also ensures the generation of high-quality recommendations. Additionally, we conduct exploratory data analysis (EDA) on the filtered data to examine the distribution of product/user average ratings and visualize the dataset's time series features.

Our implementation contributions involve building a collaborative filtering recommender system for Amazon electronics products using a variety of algorithms, including SVD, SVD++, KNN, and KNNBaseline. For these basic models, we perform a grid search on a 10% subset of the data to find the optimal hyperparameter patterns. In addition to these basic models, we also experiment with more complex model architectures, such as neural networks [1] and Long short-term memory (LSTM) [2], with the latter capturing the time-related features in the dataset to further enhance recommendation quality.

For evaluation, we assess the performance of all the models using two accuracy measures: Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics help us determine the differences between the predicted ratings and actual ratings in the test set. By comparing the error metrics, we can gauge the effectiveness of each model in generating accurate recommendations. In terms of LSTM and neural networks, we utilized the learning curve to visualize the progression of the training with each epoch. We have also attempted tuning the parameters using the outcome of the learning curve.

2) How We Have Addressed Feedback From the Proposal Evaluations

We first focused on filtering the dataset to address data sparsity and improve recommendation quality. We included users with at least 5 ratings and products with at least 10 ratings and justified our choices by examining the distribution of user and product ratings using distribution graphs and relevant statistics. After filtering the dataset, we proceeded with the grid search for various algorithms, including SVD, SVD++, KNN, and KNNBaseline. Because of the computational cost limitation, we used a 10% subset of the data for the grid search to optimize the hyperparameter pattern. Upon obtaining satisfactory results, we trained our models with the optimal parameters on the full dataset.

For innovation, we experimented with neural networks (NN) and Long Short-Term Memory (LSTM) models to improve our recommender system's performance. Our LSTM model, in particular, captures the time-related features in the dataset, showcasing our commitment to exploring innovative approaches. We used Keras from Tensorflow to develop and evaluate the LSTM model.

3) Background

A. Kaggle submission: "Amazon Electronics Products Recommender System". This Kaggle kernel provides an example of a recommender system using the Amazon electronics dataset and implements user-based collaborative filtering with the KNNWithMeans algorithm.

B. Kaggle submission: “Product Recommendation Systems”.
This Kaggle submission provides a more detailed implementation for SVD, KNN, and KNNWithMeans.

Regarding the mentioned Kaggle submissions, most recommender system work on Kaggle focuses on traditional models such as SVD, KNN, and recommendations based on item popularity. Although these approaches have already achieved good performance, there are still areas worth exploring. For instance, the KNN algorithm is constrained by the massive computational requirements of sparse matrices, resulting in a substantial computational load. Furthermore, most projects have not made good use of the timestamp feature, which is often crucial in real-world recommender systems.

Therefore, we still want to investigate the potential application of more complex models for these types of problems. In our project, we introduced neural networks and LSTM models. The previous work done on this dataset filtered out the time information for their models. However, timestamp can be an important factor in customers' reviews. But the previous work was ignored. A user's interests and preferences are not static and can evolve over time. Therefore, considering the time stamp of past interactions can help the recommender system to better capture a user's current interests and predict their future behavior.

4) Summary of Our Contributions

Implementation contribution(s): We began by filtering the dataset to address data sparsity issues and improve the overall quality of our recommendations, including users with at least 5 ratings and products with at least 10 ratings. We justified these filtering criteria by analyzing the distribution of user and product ratings using distribution graphs and relevant statistics. We implemented a variety of algorithms, such as SVD, SVD++, KNN, and KNNBaseline, and introduced more complex models like neural networks and LSTM. Our LSTM model, specifically, is designed to capture the time-related features in the dataset, demonstrating our dedication to exploring innovative approaches. We also optimized the hyperparameter tuning process by conducting a grid search on a 10% subset of the data, which allowed us to reduce computational costs while achieving satisfactory results.

Evaluation contributions: We performed cross-validation for SVD, SVD++, KNN, and KNN baseline models to ensure their generalizability and robustness. We compared their performance by plotting

the results, which provided a clear visual representation of their respective strengths and weaknesses. For the neural network and LSTM models, we computed the mean absolute error (MAE) and root mean square error (RMSE) to assess their performance. We also created learning curves for these two methods to analyze their training and validation performance over time, helping to identify potential overfitting or underfitting issues.

5) Detailed Description of Contributions

5.1 Implementation Contributions

Data Preprocessing and Filtering:

To better understand the data distribution, we calculated the quantiles for users and products at various percentiles and visualized them in Appendix Figure 1. Our analysis revealed that more than 60% of users provided only one rating, while approximately 70% of products received fewer than five ratings from users. This finding indicated that the dataset was indeed sparse, characterized by a large number of users with few ratings and many products with a limited number of ratings.

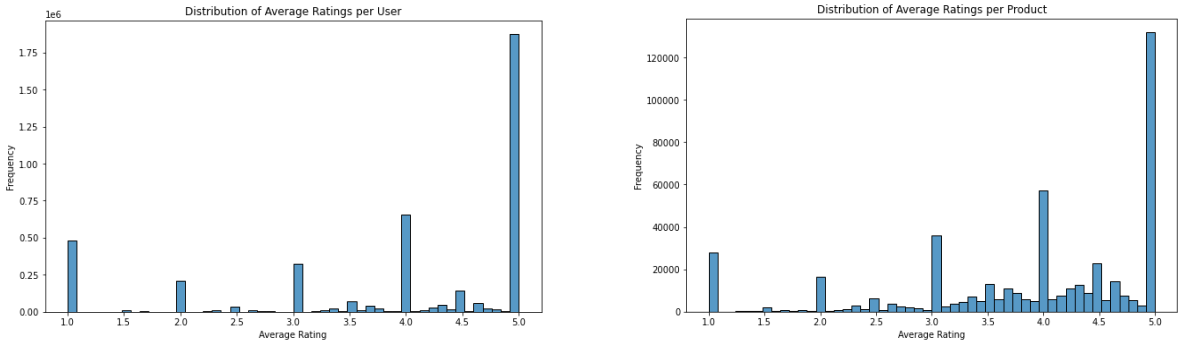


fig 1

Considering this insight, we further examined the average ratings per user and average ratings per product, as depicted in Figure 1 above. We observed a significant number of users and products with an average rating of 1.0. For a recommender system, such a distribution may negatively impact its effectiveness. To address this issue and filter out potentially unreasonable data, we established thresholds requiring users to have at least five ratings and products to have at least ten ratings. These values were chosen to balance reducing data sparsity and retaining enough data for meaningful analysis. The filtered data came to shape (1991560, 4), about 75% of data were filtered.

By examining the EAD of the filtered data, it's important to note that the time series characteristic corresponds with Appendix Figure 2, making it straightforward to observe fluctuations in rating numbers across time stamps. This observation has motivated us to utilize LSTM to capture this feature for our project.

Baseline models implementation:

We defined a parameter grid for SVD and SVD++ models, which included 'n_epochs' with values [5, 10, 15, 20], 'lr_all' with values [0.002, 0.005, 0.008], and 'reg_all' with values [0.2, 0.4, 0.6]. For KNNBasic and KNNBaseline models, we used a parameter grid that included 'k' with values [10, 30, 50]. We conducted GridSearchCV on a 10% subset of the data for all four models using their respective parameter grids, due to computational constraints. For the KNNBasic model, we trained it with a cosine similarity metric and an item-based approach using the optimal ‘k’ value found in the grid search. This configuration allowed the KNNBasic algorithm to capture similarities between items and generate recommendations accordingly. In the case of the KNNBaseline model, we also trained it

using the best 'k' value from the grid search. KNNBaseline improves upon KNNBasic by considering baseline estimates for each user-item pair, accounting for systematic tendencies in user ratings.

LSTM implementation:

For the LSTM model, we first preprocessed the data on the timestamp column. We normalized the timestamp feature based on relative time since the first time that user leaves a rating. Since the timestamp we normalized is in second, we use MinMaxScaler to scale it between 0 and 1. The last step of the data preprocessing is to reshape the input variables since LSTM models expect a specific input shape, which is a 3D tensor including the number of samples in each batch, time steps, and input dimension. We defined the LSTM layer with 128 units and a dropout layer of 0.2 to help prevent overfitting. Since the LSTM layer is a sequence of hidden states, to predict the user rating we need to reduce the sequence to a single output value. Hence, a fully connected dense layer with 64 units and ReLU activation is also added in order to introduce non-linearity into the model. Lastly, we added output layers with a single unit to predict the user rating. The model is then compiled with loss function using mean squared error and adam optimizer for updating weights in the model during training.

The model is then trained with a function named EarlyStopping to reduce computation effort. EarlyStopping is a callback that stops the training process if the validation loss does not improve after 5 epochs and restores the weights of the best epoch. Lastly the model is trained with 10 epochs, a batch size of 64 and the mse and mae are recorded for each epoch.

Neural Network implementation:

For the Neural Network model, we used the same data preprocessing as the LSTM model. We defined the model with the Keras API from TensorFlow. It included two embedding layers with output dimension of 64, one for user IDs and one for product IDs. We concatenated the flattened embedding layers and added two fully connected layers with 64 and 32 units, respectively, and ReLU activation. The output layer had one neuron to predict rating. The model was compiled with the MSE loss function, Adam optimizer, and MAE as the evaluation metric.

The model was trained on preprocessed data with a split of 0.2 and 0.8 for validation. User and product IDs were passed as input and rating as output. We trained the model for 10 epochs with a batch size of 128, and evaluated it on the validation set after each epoch. The trained model was saved for later use. To evaluate the model, we loaded the saved model and used the evaluate method on the test data. We printed the test loss and MAE and plotted the learning curve for each epoch to visualize the training and validation losses over time.

5.2 Evaluation Contribution

Baseline models evaluation:

Upon analyzing the results obtained from the 5-fold cross-validation, we can draw insights into the performance of each model in figure 2.

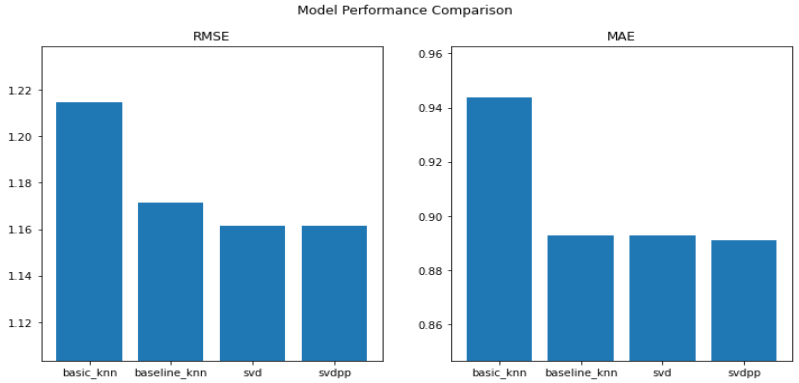


fig 2

SVD++ outperforms the others, achieving the lowest RMSE (1.1615) and MAE (0.8912) values, mainly due to its ability to incorporate implicit feedback from user interactions. This leads to more accurate predictions and better recommendations.SVD comes in second place, with marginally higher error values. Both SVD and SVD++ are matrix factorization methods effective at uncovering latent factors, but SVD does not account for implicit feedback information from users. KNNBaseline falls between the performance levels of KNNBasic and matrix factorization-based methods. It improves upon KNNBasic by incorporating baseline estimates for each user-item pair, reducing biases in the raw data and enhancing accuracy. KNNBasic has the highest RMSE (1.2144) and MAE (0.9439) values. As a neighborhood-based method, it relies on similarity measures, making it more interpretable but potentially less accurate in sparse data scenarios.

LSTM evaluation:

For evaluating the LSTM model, we computed the RMSE of 1.451 and MAE score of 0.943.Compared to the score from other models, basic_knn, baseline_knn, svd, and svd++ all outperformed the accuracy of LSTM model. One reason might be the time feature does not have any actual effect on the user's rating. The dataset we used might not be comprehensive enough to include information on temporal tendency changes. Additionally, the time feature we defined is based on the relative time since the user's first ever rating, which in this case the time feature only takes users' evolving preferences but does not take into consideration users' seasonal preferences. Another reason might be the fact that our LSTM did not include a parameter optimization step due to the limited resources while the models above have this extra step. Those models may be simpler and more interpretable for result evaluation and optimizing their parameters.

To better understand the training progression, we also developed the learning curve for LSTM model. The result is shown in figure 3 below. According to the result, the training curve progresses with better accuracy as the epochs increase, while the validation curve does not improve over epochs. This result suggests that the LSTM model is overfitting. The model is quickly learning to fit the training data, but is not able to improve its performance on new data. This result is expected since overfitting occurs when a model learns the specific characteristics of the training data too well, to the point where it performs poorly on new data. LSTM added a new feature and contains more parameters compare to the other models, making it more prone to overfitting.

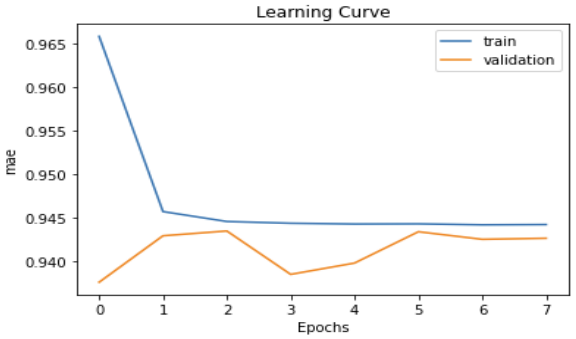


fig 3

Neural Network evaluation:

Slimilarly, to evaluate the neural network model, we again used the mean absolute error (MAE) as the performance metric. The model achieved a validation MAE of 1.0429 after 10 epochs of training. While the validation loss did not decrease much after the first epoch, the training loss continued to decrease with each epoch. This suggests that the model may be overfitting to the training data. To reduce overfitting, we added a dropout regularization layer to the fully connected layers with a rate of 0.2. However, further tuning of hyperparameters may be necessary to improve the model's

performance. In comparison to other models, we do not have the information to determine how the neural network model's accuracy compares. Nonetheless, the neural network model uses embeddings to represent user and product IDs and concatenates them with a timestamp feature, which may provide more detailed information about the data. However, the model's reliance on this temporal information may have limitations in predicting user ratings accurately.

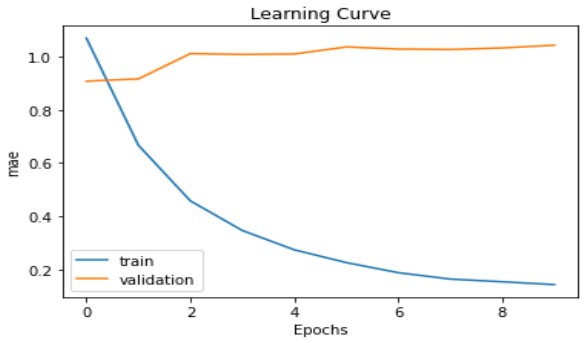


Fig 4

6) Compute/Other Resources Used

We used ml.m5d.12xlarge and a machine that could hold more than 36 GB RAM to implement the baseline models, LSTM, and Neural Network training because of the high computational and memory request for the deep learning of LSTM that involves large neural networks while baseline models with KNN algorithm require less computation.

7) Conclusions

In this project, we developed an Amazon electronics product recommender system using machine learning and deep learning tools. Our work is based on previous Kaggle works with innovation in utilizing the time stamp feature and explored using deep learning in developing the recommender system. We trained and tested our models including KNN, SVD, LSTM, and neural network on an extensive dataset of Amazon electronics product reviews. We optimized the KNN and SVD model parameters and obtain a promising prediction accuracy, while our results in the more sophisticated LSTM and neural network are showing less accurate results. According to the learning curve in LSTM and neural networks, these two models seem to be overfitting due to the lack of parameter optimization.

For the LSTM and neural network, future work can focus on tuning the parameters for a better fit. Additionally, relative time can be converted to season-based timing when adding the temporal feature to train the model to explore the effect on users' seasonal preferences. Training these models requires a significant amount of computation which might cause environmental effects. For subsequent work, training the model can utilize a small subset of the data to improve speed and reduce computation work. User privacy might be an ethical problem in developing the system. Future work can minimize this effect by encoding the categorical features that contain personal information including userId and productId to prevent exposing sensitive private information.

8) Roles of team members:

Jingxuan Bao: Data Process & Filter, SVD, SVD++, KNN, KNN baseline training & evaluation, report writing

Yaoqi Deng: LSTM training & evaluation, report writing

Guanlin Huang: Neural Network training & evaluation, report writing

Other Prior Work / References:

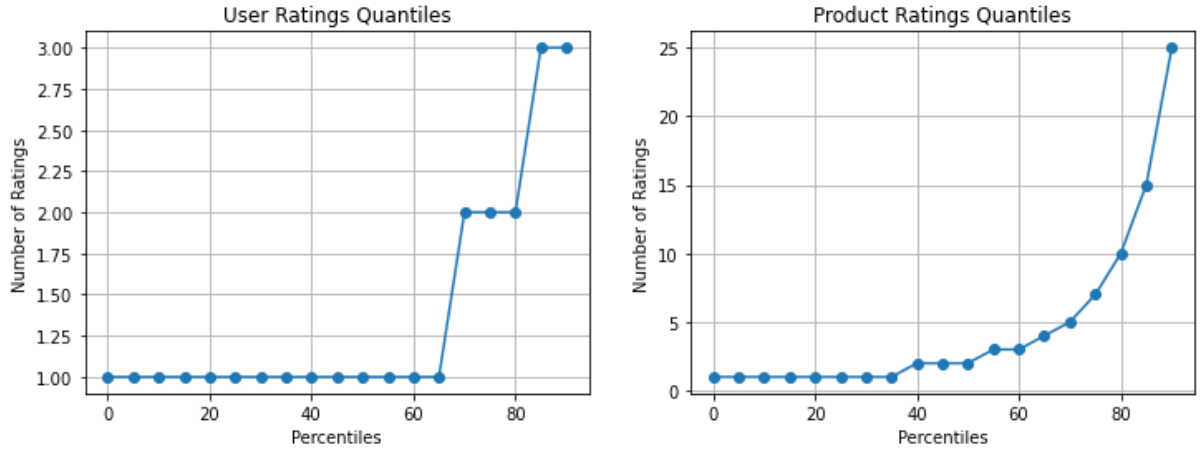
[1] Neural Collaborative Filtering for Deep Learning Based Recommendation Systems | Architecture Breakdown & Business Use Case | Width.ai. (n.d.). Retrieved April 4, 2023, from https://www.width.ai/post/neural-collaborative-filtering

[2] Zhao, C., You, J., Wen, X., & Li, X. (2020). Deep Bi-LSTM networks for sequential recommendation. Entropy, 22(8). https://doi.org/10.3390/E22080870

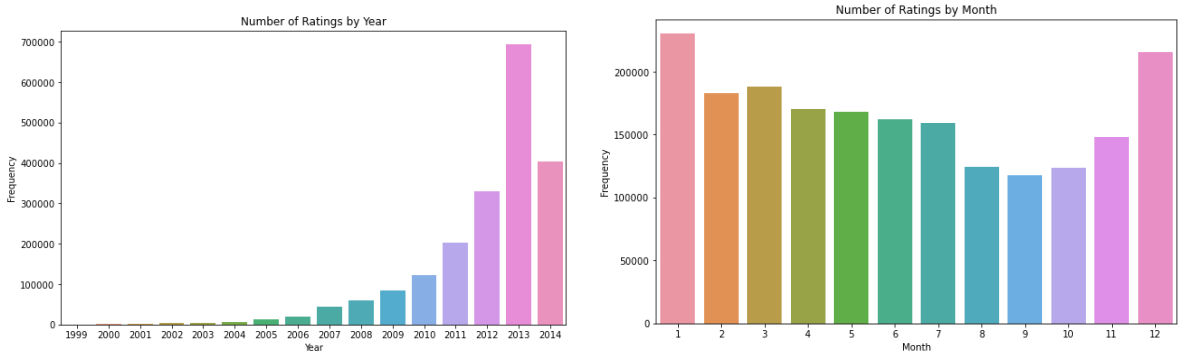
Appendix:

A. Kaggle submission: "Amazon Electronics Products Recommender System".
URL: https://www.kaggle.com/saurav9786/recommender-system-using-amazon-reviews. Code link: https://www.kaggle.com/saurav9786/recommender-system-using-amazon-reviews/code.

B. Kaggle submission: “Product Recommendation Systems”.
URL:https://www.kaggle.com/code/pritech/product-recommendation-systems.
Code link: https://www.kaggle.com/code/pritech/product-recommendation-systems/code.



Appendix fig 1



Appendix fig 2