ECommerce Recommender System

Venkata Naga Sri Sai Pranavi Kolipaka
University of South Carolina
kolipakv@email.sc.edu

Sai Krishna Revanth Vuruma
University of South Carolina
svuruma@email.sc.edu

***Abstract*—Recommender Systems (or RecSys as they are widely known) can be found in many modern day applications ranging from e-commerce websites like Amazon to music platforms like Spotify. Through this work, we fine-tuned a BERT4Rec [1] model that can leverage user behavioral data to make similar or relevant product recommendations for the user to consider. We used a publicly available dataset [2] to train the model and then evaluated it on popular evaluation metrics [3] used to quantify recommender system performance. On the test dataset, our fine-tuned BERT4Rec model achieved Recall@10 of around 60% and an NDCG@10 of 45%.**

I. Introduction

Recommender Systems learn the experiences and opinions from their customers' behaviors and recommend the items or products that they will find the most relevant among the possible results. They leverage machine learning algorithms to give users a list of items that are relevant to the item that they are currently looking for. Recommender systems have been utilized in many fields, like e-commerce, health, social networks, industry, e-learning, music, Internet of Things (IoT), food and nutritional information system, and marketing. They provide the facilities to enhance the adaption of applications to each user.

II. Background & Related Work

A recommender system leverages machine learning algorithms to give users a list of items that are relevant to the item that they are currently looking for. Popular systems also use behavioral data from other users that have shown similar tendencies as the current user to make more personalized recommendations. They use different criteria such as past purchases, demographic information and behavioral data among others.

Shin [4] developed a ShopperBERT model to extract a universal user representation pre-trained from large-scale user behaviors for recommender systems in an e-commerce platform. This model utilizes the framework of BERT in learning general user representations, combined with the two augmentation methods. It invloves two types of pre-training methods: ShopperBERT-MP and ShopperBERT-CLS. ShopperBERT-MP performs the pretext tasks predicting masked information of the [MASK] token with the final hidden vector of that token. "MP" stands for "mean pooling" method, which averages the final hidden vectors of the behavior tokens which will be used as a user representation. On the other hand, ShopperBERT-CLS puts a [CLS] token in front of a sequence of purchase logs, and then predicts the hidden information of the [MASK] tokens by using the final hidden layer of the [CLS] token. The learned embedding of the [CLS] token is used for the user representations.

Islet [5] proposed a hierarchical recommendation system to increase the performance of the e-commerce recommendation system. It is a DeepIDRS approach that has a two-level hierarchical structure: (1) The first level uses bidirectional encoder representations to represent textual information of an item (title, description, and a subset of item reviews), efficiently and accurately; (2) The second level is an attention-based sequential recommendation model that uses item embeddings derived from the first level of the hierarchical structure. This recommendation system not only uses item title and description, but also item's reviews when generating item embedding vectors with bidirectional encoder representations. This increases recommendation accuracy. This hierarchical structure provides a personalized, agile and explainable recommendation system which uses bidirectional encoder representations to provide more accurate recommendation results.

Chu [6] proposed a novel recommender system for eCommerce in which Word2Vec is adopted to extract information from the comments users have made on the items bought. Then dimensionality reduction is applied using PCA to project the acquired data into a lower dimension space. A clustering algorithm is then used to group the involved items to a small number of clusters. Finally, ICRRS is applied to make item recommendations to the users and the recommendation results are generated for each user.

Loukili [7] developed a Machine Learning based recommender algorithm to suggest personal recommendations to customers using association rules via the FrequentPatternGrowth algorithm. An association rule based recommendation system, via the FP-Growth algorithm, provides high accuracy while being easy to implement and explain, hence it is adopted in this model. The limitation of this work is that some of the evaluation characteristics of a recommender system, such as diversity and explainability, are difficult to define.

III. Dataset

A. eCommerce Dataset

The dataset is made publicly available on Kaggle [2] by the REES46 Marketing Platform [8]. It contains behavioral data of users from a large eCommerce store collected over a period of seven months from Oct 2019 - Apr 2020 spanning around 285 million user events on the platform. We chose to focus on the Oct 2019 sample that consists of 42 million rows of

recorded user-interactions. The schema consists of 9 columns with the following information:

- **Event:** timestamp, type
- **Product:** id, category, category_code, brand, price
- **User:** user_id, user_session_id

There are three possible event types: view, cart, and purchase that indicate how the user interacted with the product in a session.

B. Data Analysis

Upon examining the raw data from the sample, we had five columns that were object data (or strings) such as event_time, event_type, category_code, brand and user_session; three integer columns namely product_id, category_id and user_id; and one floating-point column, price.

```
df1.describe()
```

	product_id	category_id	price	user_id
count	42,448,764.0	42,448,764.0	42,448,764.0	42,448,764.0
mean	10,549,932.375842676	2.057404237940798 7e+18	290.3236606849143	533,537,147.50816935
std	11,881,906.970608113	1.843926466140415e+16	358.26915533940263	18,523,738.17465412
min	1,000,978.0	2.0530135522261076e+18	0.0	33,869,381.0
25%	1,005,157.0	2.0530135554641106e+18	65.98	515,904,318.0
50%	5,000,470.0	2.0530135556318828e+18	162.93	529,696,452.0
75%	16,000,305.0	2.0530135634248998e+18	358.57	551,578,838.25
max	60,500,010.0	2.1754195950939676e+18	2,574.07	566,280,860.0

Fig. 1. October Data

We can infer from Figure 1, that the standard deviation value of product_id and price columns is higher than their respective mean values. This indicates that there is a high variation between values, and abnormal distribution for data - meaning we have a diverse range of products and product costs to play with.

As shown in 12, the category_code column has about 25% NULL values, while the brand column has around 13%. The user_session column has 2 NULL values. These values need to be backfilled or dropped during data cleaning. After dropping the duplicate values, we visualized the total number of users who viewed, carted or purchased an item. Around 96% of the records in the raw dataset pertain to users viewing a product while the remaining 4% is split amongst carting and purchasing products. This indicates a great imbalance in the dataset that we must address. The results for the same can be found in Figure 2.

From Figure 3, we can see that the average number of users is highest during the weekend, i.e., Friday, Saturday, and Sunday compared to all the other days with the highest number of users on the platform during Sunday.

The sum of users on each day over October is more than 1 million. This implies that October is a busy month for ecommerce applications. This is visible in Figure 13. Figure 14 shows the activity of users on all days. From this, we can understand that most users visited the shop on Sunday (depicted as 6) and least number of users on Monday (depicted as 0). The average number of users during 4 PM are the highest during both months every day. This can be seen in Figures 15.

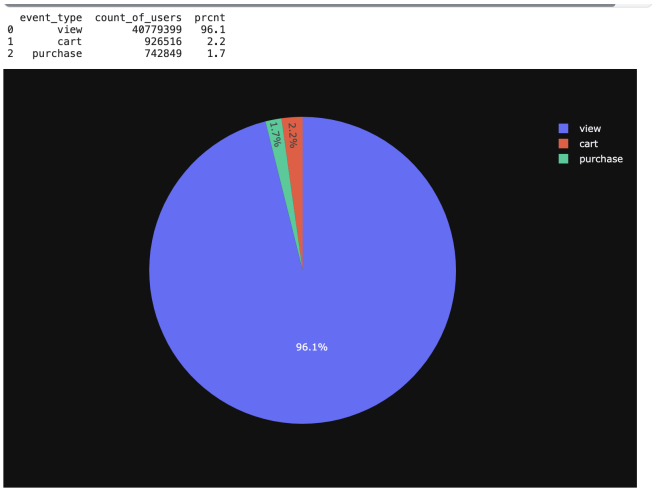


Fig. 2. Event Type Breakdown

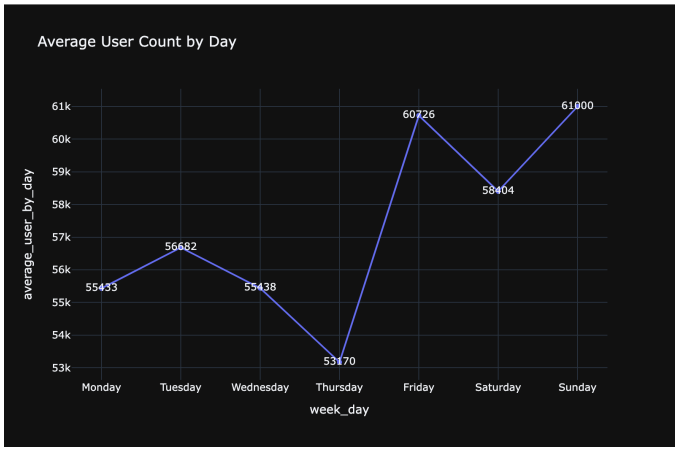


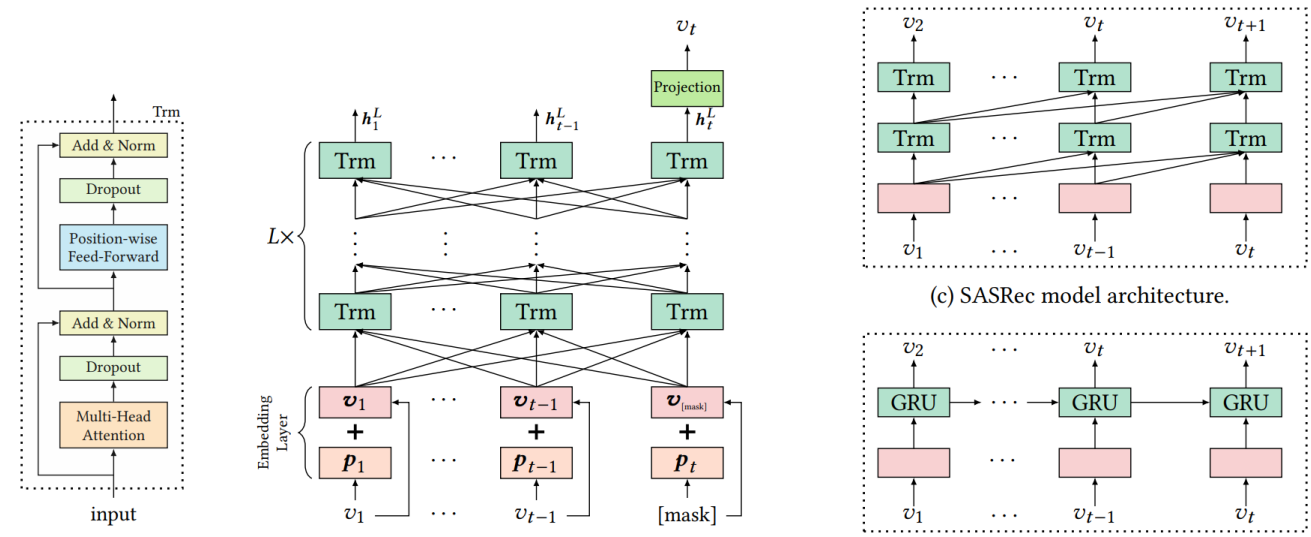
Fig. 3. Average Daily User count per weekday

C. Data Cleaning

Given the size of the dataset being too large, we only kept records that pertained to the user making a purchase and dropped all the other event_types. In addition, any rows with NULL or NaN values and duplicates were also dropped from the dataset.

The data in the event_time column was set to be string (or object) data type. This was converted into a datetime data type so that we can sort the records by timestamp for creating the sequence. The category_code column used a dot(.) separator to concatenate multiple sub-categories for a product. We replaced that with a space to make it easier for the recommender.

In the end, we had a cleaned dataset of more than 500K records of user purchases made on the platform in Oct 2019. This will be the sample that we feed to the model.



(a) Transformer Layer. (b) BERT4Rec model architecture. (c) SASRec model architecture. (d) RNN based sequential recommendation methods.

Fig. 4. Differences in sequential recommendation model architectures. BERT4Rec learns a bidirectional model via Cloze task, while SASRec and RNN based methods are all left-to-right unidirectional model which predict next item sequentially [1].

IV. Model

A. Sequential Recommendation

Sequential recommendation is a sophisticated approach to providing personalized suggestions by analyzing users' historical interactions in a sequential manner. Unlike traditional recommendation systems, which consider items in isolation, sequential recommendation takes into account the temporal order of user actions.

The idea of sequential recommendation tasks is to identify patterns in user interactions and suggest similar products for the user to interact with. For example, if a user has interacted with products such as running shoes, joggers and sweat bands, it is very likely that they are interested in athletic products. They are more likely to buy a smart watch to track their fitness statistics as opposed to something like a new monitor. Through sequential recommendation systems we can recognize such product pairings based on previous user interactions and recommend them for new users looking at similar products.

B. BERT4Rec

BERT4Rec [1] is a variant of the popular BERT model that is trained and designed for Recommendation Systems. Building on the strong points for the original BERT model such as the multi-headed self-attention layer, BERT4Rec adopts transformers to sequential recommendation tasks. Thanks to the self-attention mechanism, BERT4Rec can directly capture dependencies in any distances. In contrast to other recommender systems, BERT4Rec uses bidirectional self-attention to model users' behavior sequences to improve quality of recommendations. The model architecture can be found in Figure 4.

The is fine-tuned for recommendation tasks by modifying the recommendation task into a masked language modeling problem. The idea is to predict masked items in a sequence of items a user has interacted with, given the other items in the sequence. This adaptation allows BERT4Rec to capture sequential patterns in user-item interactions, that can be used for tasks like next-item recommendation or session-based recommendation. The BERT4Rec model performed best on benchmark datasets such as Amazon Beauty, Steam and MovieLens with a performance increment of 11.03% NDCG@10 against the strongest baselines.

Input: $[v_1, v_2, v_3, v_4, v_5] \xrightarrow{\text{randomly mask}} [v_1, [\text{mask}]_1, v_3, [\text{mask}]_2, v_5]$
Labels: $[\text{mask}]_1 = v_2, \quad [\text{mask}]_2 = v_4$

Fig. 5. Token Masking

The input to the model is a sequence of tokens i.e. products in our case. Each user's purchases are grouped together based on user_session and user_id and ordered by the timestamp. So the products that the user purchased earlier appear first in the sequence before the products purchased later. This sequence of products is then passed to the BERT4Rec model for training. As shown in Figure 5, the model then masks some tokens in the sequence at random and has to train itself to predict what those masked tokens would be. For our use case, the masked tokens will be product recommendations.

A major difference between other sequential recommenders and BERT4Rec is that it uses bidirectional attention, so the masked tokens don't need to be at the end of the sequence. For sequences where the masked tokens appear in the middle, BERT4Rec uses its patented multi-headed bidirectional attention to understand relations between tokens both before and after the masked token i.e. product.

C. Data Preparation

As mentioned in the previous section, each user's purchases are grouped together based on user_session and user_id and ordered by the timestamp. Then we drop columns such as the session_id and timestamp which no longer have use in training. The product sequence is then converted to a list object to pass to the model.

We have set limits on the sequence length to better train the model. Any sequence with length outside of [3, 50] is dropped and only those in the range will be passed to the model.

Once the sequences are ready, we mask some tokens in the sequence as depicted in Figure 5 with a probability of 25%. The dataset was split into train, validation and test using a 70:10:20 split.

D. Training

The BERT4Rec model is available on RecBole [9], a library that is developed based on Python and PyTorch for reproducing and developing recommendation algorithms in a unified, comprehensive and efficient framework for research purpose.

The model was trained over 50 epochs with the following parameters:

- embedding size of 64
- hidden layer size of 128 and an inner layer size of 256 with two transformer layers and a dropout layer (prob=0.3)
- two attention heads (n=2) for multi-headed bidirectional attention on the data
- batch_size of 4096 for training and validation
- token masking probability of 25%
- 'adam' learner with a learning rate of 0.01
- 'gelu' activation function
- 'cross entropy' loss function

The loss function value during training is shown in Figure 6.

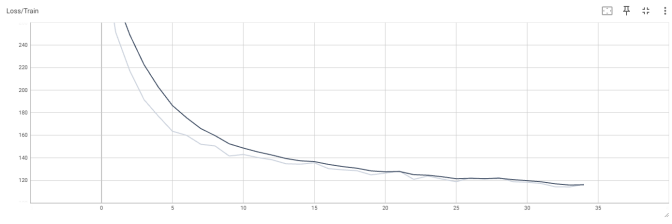


Fig. 6. Training Loss vs Epoch

V. Evaluation

A. Metrics

While evaluating recommender systems it is important that we don't look to calculate scores on individual predictions as that could skew the model. Instead, we consider the set of predictions as a whole. For evaluating the performance of our recommender system, we will use popular metrics [10] in the space such as:

- **Precision @K, Recall @K:** Similar to how precision and recall are usually calculated, but adjusted for the top-K number of predictions made by the model.
- **Item Coverage @K:** It measures the proportion of items that a recommender system can recommend, and the measure increases as the size of the recommendation list increases.
- **Normalized Discounted Cumulative Gain @K:** Also known as NDCG, this metric can be used to calculate a cumulative score of an ordered set of items. Items of higher relevance appearing at lower ranks are penalized thus resulting in a more grounded score.
- **Mean Reciprocal Ratio:** Also known as MRR, it measures the average of the reciprocals of the ranks of the first relevant item in the list of recommendations.

B. Results

As described in the previous section, we evaluated the model on cummulative predictions as opposed to individual predictions. So each metric will be calculated for the top-k recommendations made by the model.

For example, if the model recommended products p1, p2, p3, p4 and p5 for a user, the top-3 recommendations will be p1, p2 and p3. We used k=10 as the baseline for evaluating our model, so all reported metrics will be for the top-10 recommendations made by the model.

On the k=10 baseline, our model's performance is as follows on these metrics:

- recall@10 = 58.03%
- mrr@10 = 41.34%
- ndcg@10 = 45.34%
- hit@10 = 58.03%
- precision@10 = 5%
- itemcoverage@10 = 26.91%
- giniindex@10 = 97.17%

In addition, we have evaluated certain metrics against different values of k to understand how the model is doing with a varying top-k value. Figure 8 shows recall against k, while figure 7, figure 9, figure 11 and figure 10 do the same for precision, NDCG, MRR and itemcoverage respectively.

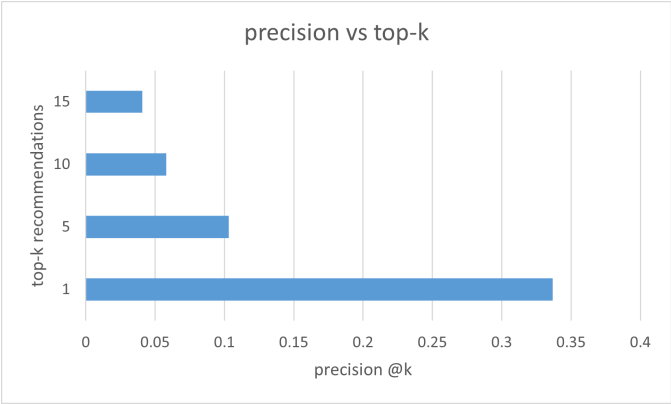


Fig. 7. Precision vs Top-k

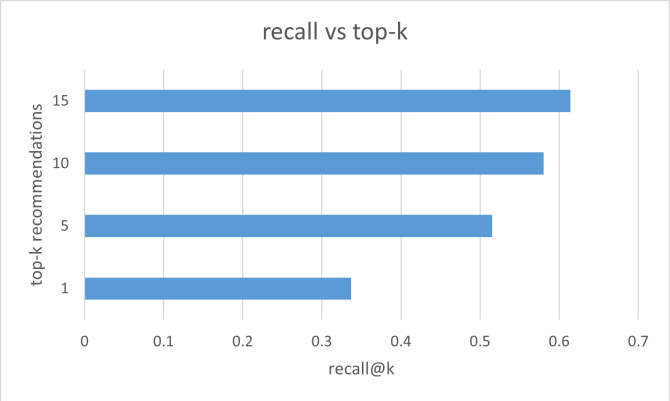


Fig. 8. Recall vs Top-k

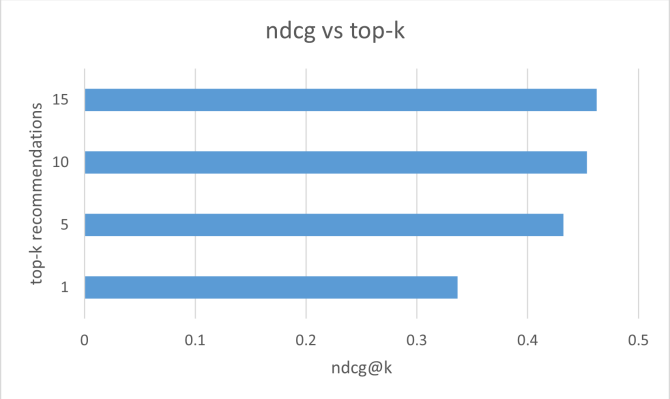


Fig. 9. NDCG vs Top-k

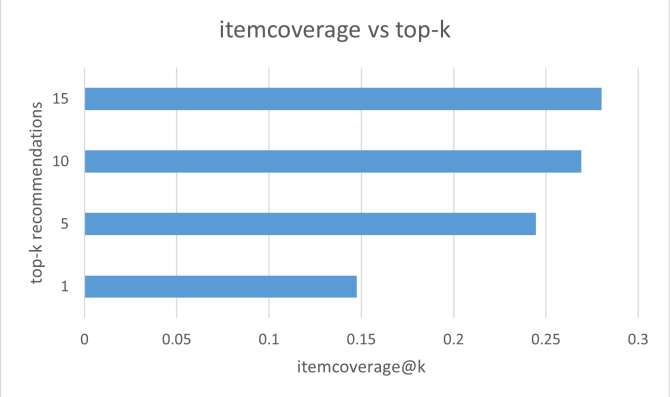


Fig. 10. Item Coverage vs Top-k

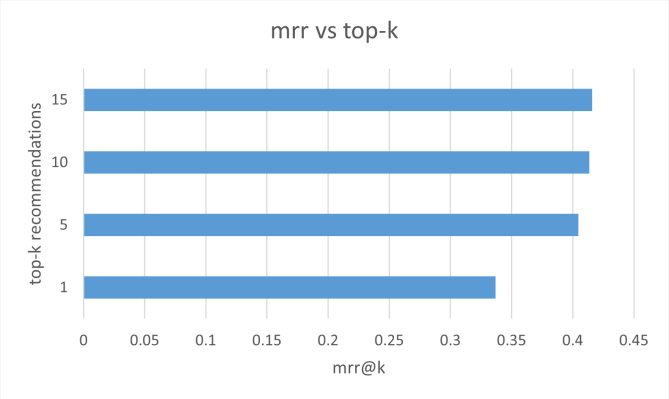


Fig. 11. Mean Reciprocal Ratio vs Top-k

C. Comparison

Although the model has a decent recall value on the test dataset, there are areas in which it can improve:

- The precision value is really low (below 5%) as the k-value increases which indicates that the model is having trouble identifying more than one relevant product.
- The NDCG value is consistent with differing k-values which is ideal, but it's value should preferably be above 50% indicating that it is at least making half of the top-k recommendations relevant to the user.
- The ideal MRR value is 1 which indicates that the first recommendation made by the model for each user is relevant. Our MRR currently stands around 0.4.

VI. Conclusion & Future Scope

We built a BERT4Rec model for sequentially recommending products on a ecommerce dataset with the following performance metrics: Recall@10 of about 60%, NDCG@10 of 45% and Itemcoverage@10 of 27%.

In the future, this work can be expanded to include data from the other product-specific columns such as price, brand, etc. for contextual recommendation. Other sequential recommender models such as GRU4Rec, etc. can be used to compare their performance against BERT4Rec. With better computing infrastructure we can leverage the full dataset and train more intelligent models.

VII. Contributions

Both the authors have contributed equally towards the project. Pranavi focused on gathering the data and performing data analysis, preparation tasks. While Sai trained the BERT4Rec model and compiled the results.

Our code is available on GitHub at https://github.com/sai-vuruma/ecommerce-recommender-system.

References

[1] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," 2019.

[2] M. Kechinov. (2024) ecommerce behavior data from multi category store. Accessed on March 09, 2024. [Online]. Available: https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store
[3] G. Shani and A. Gunawardana, "Evaluating recommendation systems," *Recommender systems handbook*, pp. 257–297, 2011.
[4] K. Shin, H. Kwak, K.-M. Kim, M. Kim, Y.-J. Park, J. Jeong, and S. Jung, "One4all user representation for recommender systems in e-commerce," *arXiv preprint arXiv:2106.00573*, 2021.
[5] I. Islek and S. G. Oguducu, "A hierarchical recommendation system for e-commerce using online user reviews," *Electronic Commerce Research and Applications*, vol. 52, p. 101131, 2022.
[6] P.-M. Chu and S.-J. Lee, "A novel recommender system for e-commerce," in *2017 10th international congress on image and signal processing, biomedical engineering and informatics (CISP-BMEI)*. IEEE, 2017, pp. 1–5.
[7] M. Loukili, F. Messaoudi, and M. El Ghazi, "Machine learning based recommender system for e-commerce," *IAES International Journal of Artificial Intelligence*, vol. 12, no. 4, pp. 1803–1811, 2023.
[8] REES46. (2024) About rees46. Accessed on March 10, 2024. [Online]. Available: https://rees46.com/
[9] L. Xu, Z. Tian, G. Zhang, J. Zhang, L. Wang, B. Zheng, Y. Li, J. Tang, Z. Zhang, Y. Hou, X. Pan, W. X. Zhao, X. Chen, and J. Wen, "Towards a more user-friendly and easy-to-use benchmark library for recommender systems," in *SIGIR*. ACM, 2023, pp. 2837–2847.
[10] M. Bhattacharyya. (2024) Metrics of recommender systems: An overview. Accessed on March 10, 2024. [Online]. Available: https://towardsdatascience.com/metrics-of-recommender-systems-cde64042127a

SUPPLEMENTARY MATERIALS

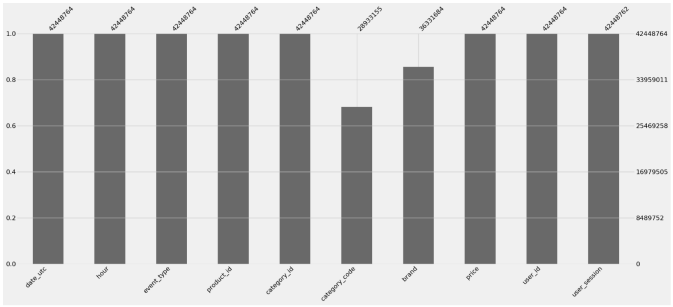


Fig. 12. Null Value count in Raw Data

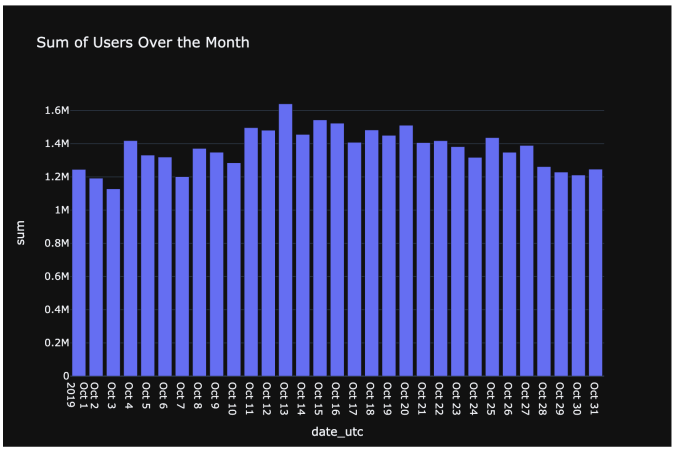


Fig. 13. Users Over October

	date_utc	hour	count	day
count	744	744.0	744.0	744.0
mean	2019-10-16 00:00:00	11.5	57,054.790322580644	2.903225806451613
min	2019-10-01 00:00:00	0.0	121.0	0.0
25%	2019-10-08 00:00:00	5.75	28,130.5	1.0
50%	2019-10-16 00:00:00	11.5	65,650.0	3.0
75%	2019-10-24 00:00:00	17.25	78,779.5	5.0
max	2019-10-31 00:00:00	23.0	113,215.0	6.0
std	NaN	6.926843254305482	29,612.354270867734	1.9416218394723965

Fig. 14. Users Over All Days in October

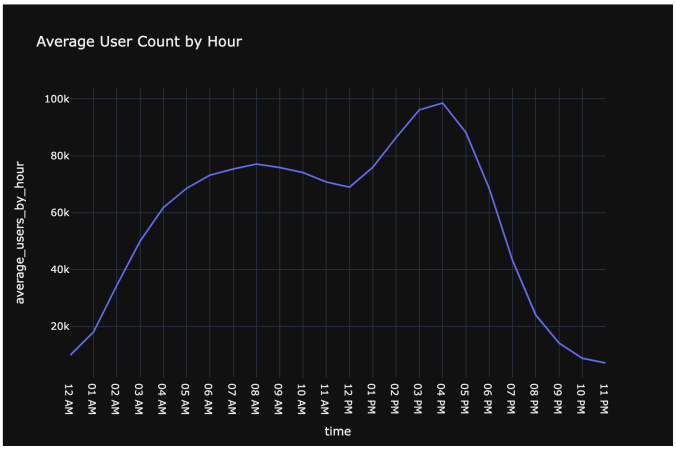


Fig. 15. Users Over October per Hour