Comparative Analysis of Association Rules and Graph Neural Networks in Product Recommendation Systems

Diogo Terrantez
up202308526@edu.fe.up.pt

University of Porto
Portugal

Abstract

This paper presents a comparative analysis of two product recommendation systems, one based on Association Rule Mining and the other on a Graph Neural Network. Utilizing a comprehensive dataset from the UCI Machine Learning Repository, the study explores the effectiveness of these systems in a real-world e-commerce context. The ARM system is evaluated for its traditional rule-based approach, while the GNN system is examined for its advanced, deep learning-based capabilities. The paper includes detailed methodology, experiments, and results, highlighting the strengths and limitations of each system, and offers insights into their applicability for modern e-commerce platforms.

1 Introduction

Recommendation systems are algorithms that analyze user preferences and behaviors to provide personalized suggestions for products, content, or services. They are crucial in enhancing user experience and engagement across various online platforms (e.g., e-commerce sites, streaming services, etc.). This paper describes the development process and comparison of two distinct types of recommendation systems: one utilizing association rules and another based on a Graph Neural Network (GNN). Both systems are trained from a real-world online retail transaction dataset containing two years of purchases.

1.1 Association rules

Association rules are a key but relatively old concept in data mining and machine learning, focusing on discovering interesting relationships between variables in large databases. Association Rule Mining (ARM) in a large transaction dataset was solved over twenty years ago [1]. However, ARM has stood the test of time and is being used to this day for e-commerce personalization, fraud detection, video streaming services, etc.

This project applies the ARM concept to a large retail transaction dataset, building a system to recommend products based on a customer's shopping basket. This system will serve as a baseline for comparison with a more recent approach that employs GNN models for recommendations.

1.2 Graph Neural Networks

GNNs represent a cutting-edge subset of deep learning technologies designed to analyze data structured as graphs. They have emerged as a transformative approach for product recommendation systems, harnessing the intricacies of the relationships between products and users to deliver highly personalized suggestions. GNNs leverage the natural graph structures inherent in user-item interactions. This structure encapsulates not just the direct interactions between users and products, such as purchases or ratings, but also the complex network of relationships, including product similarity, social connections among users, and contextual information that influences purchasing decisions [3].

Product recommendation with GNNs can be seen as a link prediction task, using a model to predict the likelihood of a user forming a new interaction with an item (i.e., purchase). Figure 1 is a sample of the knowledge graph created for this project. There are two types of nodes: Customers in the center and Products around. Each edge represents a purchase, and multiple users may purchase the same item.

For a model to represent and learn from a heterogeneous graph (multiple types of nodes/edges), it must be adapted for that purpose.

A Heterogeneous Graph Neural Network (HGNN) is a type of neural network specifically designed to work with heterogeneous graphs [7]. These networks are instrumental in scenarios where the data is naturally represented as a heterogeneous graph, such as recommendation systems

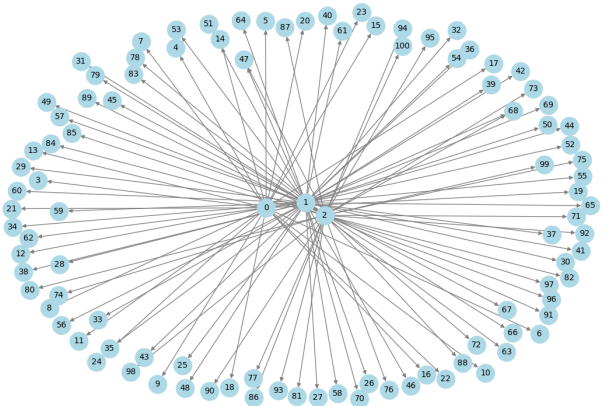


Figure 1: Sample knowledge graph created from the transaction dataset

(users, items, interactions), knowledge graphs (entities, relations), and social networks (various types of users and interactions).

This project uses an HGNN to learn the complex representations in the knowledge graph and predict products a customer may purchase.

2 Method

This section describes the method employed to implement both recommender systems, starting from data preparation. The tool of choice was Python throughout the entire project.

2.1 Dataset

The dataset for this project is "Online Retail II"[1], available in the UC Irvine machine learning repository. It represents all transactions over two years from a UK-based online retailer. The company mostly sells all-occasion giftware, and many of the customers are wholesalers. Table 1 contains a dataset sample. Each row is a single-item transaction. Multiple sequential rows are the list of items a customer purchased simultaneously, with the same invoice number, date, and customer. Some invoices include postage fees (row 5), signify item returns (row 6), or discounts (row 8). There are also some rows with missing values (row 7). Stock codes differ for products with multiple variations (rows 1, 2).

The dataset was loaded using the Python Data Analysis Library[2] (Pandas). Using pandas to check the dataset features reveals a shape of 1067371 rows by eight columns, 4382 with no description, and 243007 rows with no customer ID. It also indicates that there are rows with negative quantity and price values (indicating returns).

2.1.1 Data Cleaning

Some cleaning steps are necessary before using the dataset for the recommendation systems. First, to handle the missing values, all the rows with no customer ID were dropped. A transaction with no customer ID cannot be used as there is no customer to attribute the purchase to. Attributing the transactions to a random customer would interfere with the GNN learning, so these rows must be dropped. Rows with no description are relatively low and seem to match with a missing customer ID, so dropping those is not a problem. The final number of rows after this step is 824364.

[1] archive.ics.uci.edu/dataset/502/online+retail+ii
[2] pandas.pydata.org

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
536365	84029G	KNITTED UNION FLAG ...	6	01/12/2010 08:26	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE ...	6	01/12/2010 08:26	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	01/12/2010 08:26	7.65	17850	United Kingdom
536370	22492	MINI PAINT SET VINTAGE	36	01/12/2010 08:45	0.65	12583	France
536370	POST	POSTAGE	3	01/12/2010 08:45	18	12583	France
C536383	35004C	SET OF 3 COLOURED DUCKS	-1	01/12/2010 09:49	4.65	15311	United Kingdom
536414	22139		56	01/12/2010 11:52	0		United Kingdom
C536379	D	Discount	-1	01/12/2010 09:41	27.5	14527	United Kingdom

Table 1: Sample from initial dataset

Invoice	StockCode	Quantity	InvoiceDate	Price	Customer ID	Country	label
489434	21232	24	01/12/2009 07:45	1.25	13085	United Kingdom	1
489434	20867	12	01/12/2009 07:45	1.25	13085	United Kingdom	0
489434	21523	10	01/12/2009 07:45	5.95	13085	United Kingdom	1
489434	22467	1	01/12/2009 07:45	2.55	13085	United Kingdom	0
489434	21871	24	01/12/2009 07:45	1.25	13085	United Kingdom	1
489434	21714	6	01/12/2009 07:45	1.25	13085	United Kingdom	0

Table 2: Sample from final dataset

Then, all the rows with negative item quantities were also dropped. These rows indicate returns, which are not relevant for this project. The final number of rows after this step is 805620.

Finally, to have a uniform dataset, all the different product variations were merged. To do that, the letters were first removed from all stock codes. Then, the rows with the same stock code merged for each invoice, summing the quantities. All other values in the same transaction are always equal, including price, which is the same for all product variations. The final number of rows after this step is 747400.

The entire description column was also dropped to save on memory since item descriptions will not be used in the recommendation systems.

2.1.2 Data Explorations and Processing

The dataset is simple in nature, not having many variables that are relevant to the recommendation systems. Only one plot was considered relevant for the recommendation task: a scatterplot of individual products and the total quantity bought by each user. Customers who buy too many items will heavily skew the recommendation results, so they should be removed as outliers. In this process, the users above the 95th percentile in the number of unique items or total quantities bought were removed from the dataset, along with their transactions. The final number of rows after this step is 425233. Customer distribution plots are available in Figure 2

The top percentages of transactions per country were also noted: United Kingdom - 91.5%, Germany - 2.1%, France - 1.4% (Portugal - 0.46%).

The final processing step consists of augmenting the dataset before creating the knowledge graph for training (the ARM system will not use this dataset). In this process, a new binary column is added to the dataset, "Label". All the existing transactions (rows) are labeled with "1". Then, for each individual transaction, a new, fake one will be added. The new transaction will be labeled with "0" and have all the same values as the previous one, except: **StockCode** - a random product the customer did not buy; **Quantity** - a random quantity from the pool of all quantities the random product has been purchased in; **Price** - a random price from the pool of all prices the random product has been sold at.

The fake transactions are intercalated with the real ones. This step ensures all customers have a proportional number of real and fake transactions and also doubles the number of transactions to 850466. Table 2 represents a sample from the final dataset (with the outliers already removed).

2.2 Association Rule Mining

ARM in Python is very straightforward. The recommender system will recommend items for a customer based on a shopping list. The first step to make such a system is generating an invoice/item matrix. This matrix represents, for each invoice (shopping list), the products that have been purchased from the list of total products.

Then, the FP-growth algorithm (from mlxtend[3] library) is applied to the matrix, generating 624 frequent item sets (min support = 0.01).

The association rule list, along with all relevant metrics, is also generated (also using mlxtend). Rule generation uses "lift > 1" as a threshold.

Lift is a measure used to evaluate the performance and relevance of an association rule. It is a metric that compares the observed frequency of A and B occurring together with the frequency that would be expected if A and B were independent. A lift greater than 1 implies that the presence of A increases the likelihood of B occurring in a transaction. In the context of a recommendation system, lift is used to find rules that not only have a high confidence (i.e., when A happens, B also happens often) but also are not explained by chance, and hence have a lift significantly higher than 1. This helps to filter out rules that might have high confidence simply because the consequent item B is very common.

Table 3 shows a sample of the 270 rules generated.

2.2.1 Recomender System

The recommender system receives the list of products in the customer's hypothetical shopping basket. If they contain any antecedent set from the rules, the consequent will be added to the recommendation list.

2.3 Knowledge Graph

Preparing a knowledge graph is the first step to building the GNN-based recommender system.

The graph for this project has two types of nodes: customers and products, and one type of edge: purchase. **Customer** nodes are defined by their total spending from all purchases, average spending per invoice, and average quantity purchased per invoice. **Product** nodes are defined by their average sale price and total sales volume. **Edges** are defined by the total quantity of that item the customer purchased, the average sale price, the total number of purchase instances, and the label indicating the the purchase is real or fake (from the data processing step).

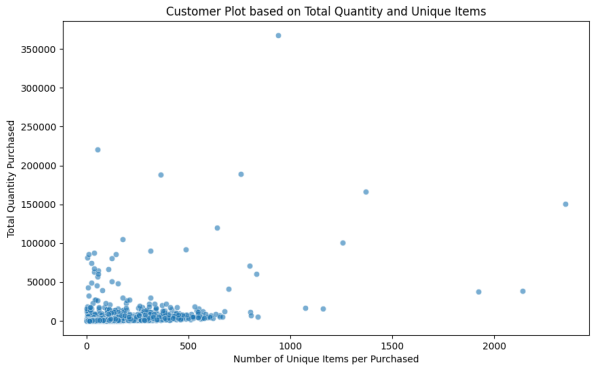
This modeling results in a heterogeneous graph, in which each customer is connected to every item they have bought (real or fake). Figure 1 is a sample of the knowledge graph used for this project. The complete graph has 5392 customer nodes, 3801 product nodes, and 735101 edges.

The graph instantiation used the Deep Graph Library[4] (DGL) for Python.

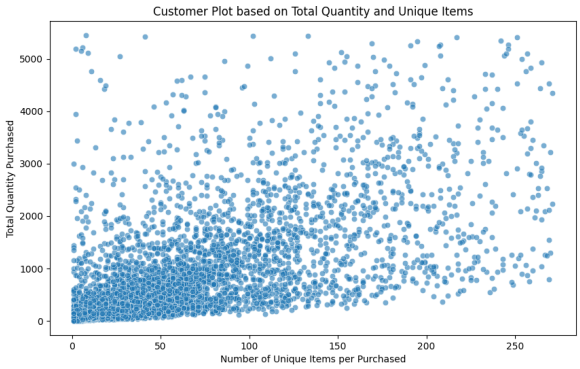
2.4 GNN Model

The GNN created for this project is designed to work with heterogeneous graphs for the purpose of product recommendation. It consists of two main components: A heterogeneous convolution layer and a fully connected layer.

[3] rasbt.github.io/mlxtend
[4] dgl.ai



(Original plot)



(No outliers)

Figure 2: Customer distribution plots

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
22745	22748	0.013020722522445854	0.014297263946215055	0.010978256244415131	0.8431372549019607	58.97193043884219	0.010792095537741296
22748	22745	0.014297263946215055	0.013020722522445854	0.010978256244415131	0.7678571428571428	58.97193043884219	0.010792095537741296
21122	21124	0.017616271648014978	0.015361048466022723	0.012297	0.6980676328502415	45.44400952801531	0.012026744646400859
21124	21122	0.015361048466022723	0.017616271648014978	0.012297	0.8005540166204986	45.44400952801531	0.012026744646400859
22698	22697, 22699	0.017360963363261137	0.016425	0.012169694906599719	0.7009803921568628	42.67808340952962	0.011884543982885035
22697, 22699	22698	0.016425	0.017360963363261137	0.012169694906599719	0.7409326424870466	42.67808340952961	0.011884543982885035
22698, 22699	22697	0.013488787711161227	0.021616	0.012169694906599719	0.9022082018927444	41.73778533992399	0.011878
22697	22698, 22699	0.021616	0.013488787711161227	0.012169694906599719	0.562992	41.73778533992399	0.011878
22698	22697	0.017360963363261137	0.021616	0.014637674992553508	0.8431372549019608	39.00505635324996	0.014262398647353871

Table 3: Sample association rules (sorted by lift)

The HeteroGraphConvLayer is responsible for processing node features of different types in the graph (customer and product). It contains two linear layers, each transforming the features for one of the nodes. To keep the project simple, the features from the edges are being disregarded.

ProductRecommendationGNN is the GNN model proper. It contains an instance of HeteroGraphConvLayer and a fully connected layer that makes the final predictions.

The GNN performs a link prediction task, predicting the likelihood of a customer buying a product. It is built using the DGL and Pytorch[5] libraries.

2.4.1 Training

The GNN model is trained using the knowledge graph generated from the labeled dataset. For training, the dataset must be split into training and validation parts. Splitting a graph consists of keeping all of the original nodes while removing part of the edges. The training and validation data consist of two subgraphs created from the original, keeping 80% and 20% of the (randomly selected) edges, respectively.

During the training loop, the loss is calculated using the edge prediction from the model and the corresponding label (real or false). The loss is calculated for the training and validation data in every training step (the model has an evaluation mode to prevent fitting to validation data). An early stop logic was implemented to finish training when the validation loss does not decrease for a given number of steps. The optimizer (Adaptive Moment Estimation) and loss function (Binary Cross-Entropy Loss) are provided by Pytorch.

Adaptive Moment Estimation is an extension of stochastic gradient descent and is commonly used to train deep-learning models. Binary Cross-Entropy Loss is a loss function commonly used in binary classification problems.

2.4.2 Recomender System

The GNN-based recommender system utilizes the trained model and a knowledge graph to predict the items a customer may buy. When making predictions for a customer, the original knowledge graph must first be modified. It originally contained edges for each real and fake purchase attributed to the customer. The neural network output is the likelihood of a certain edge occurring, meaning it only generates probabilities for existing edges. The first step for recommendation is linking the customer to all the products available. This is done by adding an edge from the customer node to all the product nodes that are not already connected to it. Since the model does not use edge features (mentioned in the graph creation step), these can be added to the graph naively. The neural network will then use the updated graph to calculate the likelihood of each edge activating (the customer purchasing the product). The recommendations consist of the sorted list of edge probabilities. Edges with a probability above 50% are considered a recommendation.

[5] pytorch.org

3 Experiments

When evaluating a recommendation system, one would typically rely on post-deployment metrics, such as:

Click-Through Rate: The rate at which users click on the recommended items.

Conversion Rate: The rate at which recommendations lead to a purchase.

Revenue Increase or Sales Uplift: The increase in sales attributable to the recommendation system.

Customer Satisfaction: Assessed through user surveys.

These metrics, however, are outside the scope of the project. Instead, we must rely solely on offline evaluation methods (using historical data from the dataset), such as:

Precision and Recall (F1 Score): The proportion of relevant recommended items balanced with the proportion of recommended items.

Scalability: The impact on the system's performance as the number of users or items grows.

A/B Testing: Comparing recommendations from both systems in a similar setting

To evaluate the ARM recommender, the association rules will be generated using 80% on the dataset and the other 20% to generate recommendations. All experiments used the same seed for the dataset splitting and Pytorch functions with random components, meaning results will be identical when repeating the experiments.

3.1 F1 Score

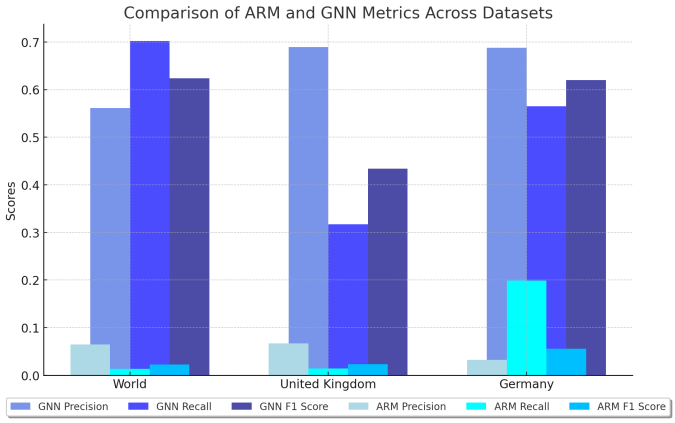
To calculate the F1 score, the first step is to define how to classify a recommendation:

True Positive (TP): Items that the association rule recommended and were bought.

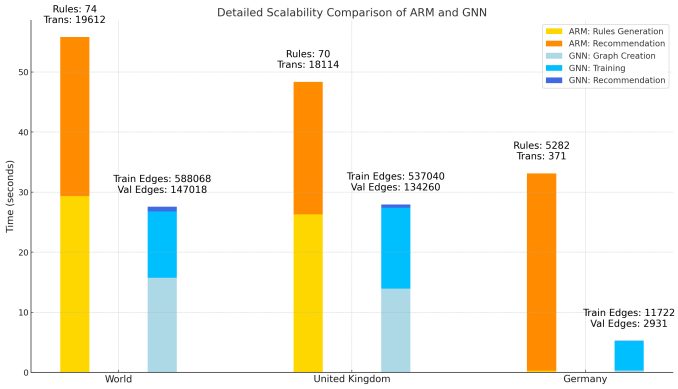
False Positive (FP): Items that the association rule recommended but were not bought.

False Negative (FN): Items that were bought but not recommended by the association rule.

F1 Score is the harmonic mean of Precision and Recall. Precision is the ratio of correctly recommended items to the total recommended items.



(F1 Score)



(Scalability)

Figure 3: Evaluation metrics

Recall is the ratio of correctly recommended items to the total items that should have been recommended.

$$Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}, F1score = 2 * \frac{Precision*Recall}{Precision+Recall}$$

3.2 Scalability

The scalability for both systems was calculated simultaneously with the F1 score. The steps involved in evaluating systems, starting from the same dataset, are: association rule generation and product recommendation for the ARM system; knowledge graph creation, model training, and product recommendation for the GNN system.

Each of the steps takes an amount of time that varies depending on the dataset size. For this reason, the F1 score was calculated three times for each system, measuring the duration of each step and the size of the relevant features from the dataset, which was filtered differently for each run. The filtering criterion is the country of origin of the users. The first run includes all the users, the second only users from the United Kingdom (91.5% of users), and, finally, only users from Germany (2.1% of users).

3.3 A/B Testing

The final experiment consists of generating a list of recommendations for two customers, using both systems. The customers chosen are the one with the most unique purchases and one with the mean number or purchases.

Their recommended items will be obtained differently from each system. For the ARM recommender, all the unique items purchased will be used as the antecedents, and the output will be the recommendations. For the GNN recommender, all edges for that customer with a probability > 0.95 will be a recommendations.

3.4 Results

The results from the experiments are shown in Figure 3. The GNN-based recommendation system scores a lot higher in all metrics, even though the model has a small number of features. This difference in scoring, however, is not only because of good GNN performance, but also a very poor performance from the ARM system. This could mean poor threshold values used to define rules, but experiments changing these values resulted in either a very low number of rules, or a high number of weak rules that further decreased the score.

The more likely motive for poor ARM performance is the nature of the data used. The dataset represents sales from a retailer that sells giftware. Purchases of this type of products do not follow meaningful rules that could be captured with this approach.

The scalability plot details the time spent in each step leading to a recommendation, along with the variables that cause the variations. The GNN-based system once again took the lead, though results are not as comparable due to implementation details. GNNs are designed with scalability in mind, benefiting from advancements in deep learning frameworks and hardware accelerators. They can scale to handle larger graphs, as the graph shows, a more than 40-fold increase in the training edges from the Germany dataset to the United Kingdom's, which equates to an increase of 2.2x in training time. The decrease in training time is due to the early stop condition triggering earlier in the training cycle. Neural network training is random in nature and using different seeds considerably impacts the duration of this step, as well as model performance.

The evaluation time is very fast, taking 0.7 seconds to evaluate all 147 thousand edges in the world dataset. The time taken in the Germany dataset was too small to plot in the graphic (0.015 seconds). This step cannot be properly evaluated as such small durations are very susceptible to the operating system and hardware behavior.

The ARM system recommendation time scales one-to-one with the number of rules and transactions (invoices), as the implementation follows a double for-loop, explaining the longer recommendation duration for the Germany dataset. The number of rules is much higher due to the data characteristics. This filtered dataset generates 1812 frequent item sets, apart from the 408 with the full world data. A data analysis confirms that German customers only bought 1696/3081 unique products.

3.5 A/B Testing

The maximum and mean number of unique items bought are 271 (customer ID 12836) and 59 (customer ID 18112). Table 4 shows the amount of recommendations for each customer.

Customer ID	ARM	GNN	Intersection
12836	63	217	17
18112	5	908	1

Table 4: Number of customer recommendations

From the number of recommendations, it may seem surprising that the customer with fewer unique items purchased got a lot more recommendations from the GNN. However, that means the model has not learned the customer preferences very well and will recommend more items. That is expected since the training graph has a lot more training examples for customer 12836 (the knowledge graph has an edge for every customer purchase).

The ARM recommender has, as expected, more recommendations for a larger number of antecedents. The best recommendations will likely be the products from the intersection of both systems' recommendations.

Appendix A lists the intersected recommendations for customer 12836. for the other one, it is: "WOODEN STAR CHRISTMAS SCANDINAVIAN".

4 Related Work

The field of product recommendation systems has seen significant advancements, with numerous studies exploring various methodologies.

4.1 Association Rules in Recommendation Systems

Early works in this area laid the foundation for using these techniques in product recommendations [1]. Further studies have expanded on this by exploring different algorithms for rule mining and their application in e-commerce settings [4].

4.2 Graph Neural Networks (GNN) in Recommendations

The adaptation of GNNs for recommendation systems is a relatively recent development. The work of Ying *et al*. [6] on Graph Convolutional Networks for web-scale recommender systems greatly advances this field. They demonstrated the effectiveness of GNNs in capturing complex user-item interactions.

4.3 Comparative Studies

Several studies have compared traditional methods like association rules with more advanced techniques. For instance, the work of Lian *et al*. [5] offers insights into the comparative performance of classic machine learning models and deep learning approaches in recommendation accuracy.

4.4 Hybrid Models

The potential of combining different approaches, such as rule-based systems with machine learning models, has also been explored [2]. These hybrid models aim to leverage the strengths of each approach to enhance recommendation performance.

5 Conclusion

This project consists of a comparative analysis of two distinct product recommendation systems: one based on Association Rule Mining (ARM) and the other utilizing a Graph Neural Network (GNN). The evaluation, using a robust dataset from the UCI repository, has provided some insights on how these algorithms work.

The GNN model outperformed the ARM system in all metrics. This superior performance of the GNN can be attributed to its advanced capability to learn complex patterns and relationships within the data. However, the ARM's lower performance in this context points to its limitations in handling datasets characteristic of some e-commerce environments, where purchasing patterns may not always follow predictable rules. The scalability analysis highlighted GNN's inherent advantage in handling large datasets, a crucial feature for modern e-commerce platforms.

In conclusion, this project underscores the potential of GNNs in revolutionizing product recommendation systems, offering more accuracy and efficiency, particularly for large-scale e-commerce datasets. Future research could explore hybrid models, combining the interpretability of ARM with the predictive power of GNNs, to create even more effective recommendation systems.

References

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, jun 1993. ISSN 0163-5808. doi: 10.1145/170036.170072. URL https://doi.org/10.1145/170036.170072.

[2] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12:331–370, 2002.

[3] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, WWW '19, page 417–426, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313488. URL https://doi.org/10.1145/3308558.3313488.

[4] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, may 2000. ISSN 0163-5808. doi: 10.1145/335191.335372. URL https://doi.org/10.1145/335191.335372.

[5] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1754–1763, 2018.

[6] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219890. URL https://doi.org/10.1145/3219819.3219890.

[7] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 793–803, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330961. URL https://doi.org/10.1145/3292500.3330961.

A Recommended product list

StockCode	Description
20724	RED RETROSPOT CHARLOTTE BAG
20725	LUNCH BAG RED RETROSPOT
20726	LUNCH BAG WOODLAND
20727	LUNCH BAG BLACK SKULL
21136	PAINTED METAL PEARS ASSORTED
21166	COOK WITH WINE METAL SIGN
21175	GIN + TONIC DIET METAL SIGN
21232	STRAWBERRY CERAMIC TRINKET BOX
22384	LUNCH BAG PINK POLKADOT
22469	HEART OF WICKER SMALL
22470	HEART OF WICKER LARGE
22554	PLASTERS IN TIN WOODLAND ANIMALS
22556	PLASTERS IN TIN CIRCUS PARADE
22910	PAPER CHAIN KIT VINTAGE CHRISTMAS
47566	PARTY BUNTING
84836	ZINC METAL HEART DECORATION
84970	SINGLE HEART ZINC T-LIGHT HOLDER

Table 5: Product recommendations for customer 12836