



COMMUNICATION SCIENCE ET TECHNOLOGIE
Vol 23, N°01, pp 23, 35
ENPO – Maurice AUDIN
ASJP/ ESSN 2773-3483



Heterogeneous Graph Neural Networks for Product Recommendation on Transactional Retail Data

Imad Eddine Khiloun ^{*1}, Karima Belmabrouk ², Latifa Dekhici ^{2,3}, Christoph Bergmeir ⁴

¹*S.I.M.P.A laboratory, Department of Computer Science, University of Science and Technology of Oran Mohamed Boudiaf*

²*Department of Computer Science, University of Science and Technology of Oran Mohamed Boudiaf*

³*LDREI laboratory, Department of electrical engineering, Higher School of Electrical and Energy Engineering of Oran*

⁴*DaSCI, Department of Computer Science and AI, University of Granada, Spain*
Address

imadeddine.khiloun@univ-usto.dz

karima.belmabrouk@univ-usto.dz

latifa.dekhici@univ-usto.dz

bergmeir@ugr.es

Abstract— Personalized product recommendation is crucial for enhancing user experience and driving sales in e-commerce, yet effectively leveraging sparse, implicit feedback from transactional data remains challenging. This paper investigates the application of heterogeneous Graph Neural Networks (GNNs) for product recommendation on the widely used Online Retail dataset. We frame the task as link prediction between customers and products, constructing a heterogeneous graph from processed transactional records. We employ a GNN model based on GraphSAGE, adapted for heterogeneity, using learnable embeddings derived solely from the interaction structure. Our experiments demonstrate the model's effectiveness, achieving a ROC AUC of 0.853 and an F1-Score of 0.750 (with 0.937 Recall) on the held-out test set using an optimized configuration (negative sampling ratio 1.0, classification threshold -0.5). We analyze the significant impact of the negative sampling ratio during training on the final precision-recall trade-off, highlighting the importance of aligning training parameters with desired recommendation goals. Our findings confirm the viability of heterogeneous GNNs for modeling implicit feedback and providing effective recommendations in a retail context. This paper is organized as follows: Section I introduces the problem and our approach. Section II reviews related work in recommendation systems and graph-based methods. Section III details our methodology, including dataset description, data preparation, graph construction, and the GNN model architecture. Section IV describes the experimental setup and evaluation metrics. Section V presents the quantitative results, including the analysis of parameters like negative sampling. Section VI discusses the findings and limitations. Finally, Section VII concludes the paper.

Keywords— Recommendation Systems, Graph Neural Networks (GNNs), Link Prediction, Implicit Feedback, Heterogeneous Graphs, Online Retail, E-commerce, GraphSAGE, Node Embeddings.

I. INTRODUCTION

Personalized recommendation systems are integral to modern online platforms, enhancing user engagement and facilitating navigation through vast product catalogs [1]. A primary challenge lies in leveraging sparse, implicit user feedback (e.g., purchase histories, clicks), which only indirectly signals preferences [2]. Effectively modeling the underlying complex user-product interaction patterns from such data is crucial for generating relevant suggestions but remains non-trivial.

Traditional methods like Matrix Factorization, while foundational [3], can struggle with the extreme sparsity often present in real-world implicit datasets and may not fully capture higher-order collaborative signals embedded within the interaction structure [4]. To overcome these limitations, Graph Neural Networks (GNNs) have gained significant traction as a powerful tool for recommendation systems [5]–[6]. GNNs naturally operate on graph-structured data, allowing them to explicitly model user-product interactions and learn rich node representations by aggregating information from their network neighborhoods. This capability is particularly promising for uncovering intricate patterns in implicit feedback data.

Representing user-product interactions within a heterogeneous graph framework, distinguishing between user nodes and product nodes, further allows GNNs to learn type-specific representations and interaction patterns [7]. This work explores the application of heterogeneous GNNs for recommendation based on implicit feedback, framed specifically as a link prediction task: predicting the likelihood of a future interaction between a user and a product.

Our contributions are:

- A methodology for processing raw, implicit transactional data into a structured heterogeneous graph suitable for GNN modeling.

- The implementation and evaluation of a heterogeneous GNN model, utilizing the GraphSAGE architecture [8] adapted via PyTorch Geometric's to_hetero capabilities [9].
- An empirical analysis of the model's effectiveness in predicting user-product interactions using standard evaluation metrics.

The remainder of this paper details our methodology, experimental setup, results, and discussion, followed by concluding remarks.

II. RELATED WORK

Recommendation systems have progressed significantly beyond early collaborative filtering. While foundational Matrix Factorization (MF) techniques [10] are well-understood, they often grapple with the high sparsity typical of implicit feedback datasets, sometimes failing to capture complex higher-order user-product relations [2]–[4]. Methods adapting MF for implicit data, such as Weighted MF [10], address the lack of negative signals but fundamentally model pairwise interactions [4].

Graph Neural Networks (GNNs) represent a more recent advancement, directly leveraging the user-product interaction graph [5]. State-of-the-art models like LightGCN [4] learn user and product embeddings via simplified graph convolutions, proving highly effective for recommendation by capturing collaborative signals embedded in the graph structure, explicitly addressing limitations of earlier methods. This underscores the power of neighborhood aggregation in GNNs for implicit feedback scenarios.

Furthermore, the inherent heterogeneity of user-product interaction graphs (distinguishing between user and product node types) motivates the use of Heterogeneous GNNs (HGNNs) [11]. Models tailored for heterogeneous graphs, such as HAN [7], or general GNN frameworks adapted using tools like PyTorch Geometric [9], allow for type-specific message passing, potentially yielding more refined user and product representations compared to treating the graph homogeneously.

Our research utilizes these GNN principles. We employ a heterogeneous adaptation of the inductive GraphSAGE architecture [8], implemented using PyTorch Geometric [9], to learn node embeddings directly from the implicit feedback graph structure. A key focus is the practical preprocessing of noisy transactional data to enable GNN application and the use of neighbor sampling for efficient training on large-scale graphs.

III. METHODOLOGY

This section details our approach, from formulating the recommendation problem to constructing the graph and defining the GNN model architecture.

A. Problem Formulation

We formulate the product recommendation task as a link prediction problem on a heterogeneous graph. Let $G = (V, E)$ represent the interaction graph, where $V = V_{\text{customer}} \cup V_{\text{product}}$ is the set of nodes comprising unique customers (V_{customer}) and unique products (V_{product}). E represents the set of observed 'buy' interactions, forming edges between customers and products they have purchased. The goal is to train a model that, given the graph G , can predict the probability $P(\text{edge}(u, v) = 1 \mid G)$ of a link (a 'buy' interaction) existing between an arbitrary customer $u \in V_{\text{customer}}$ and product $v \in V_{\text{product}}$, particularly for pairs (u, v) not present in the observed edges E .

B. Dataset Description

We utilize the standard Online Retail dataset, originally sourced from the UCI Machine Learning Repository [12] and widely available on platforms like Kaggle. This dataset consists of 541,909 transactional records generated by a UK-based non-store online retail company between December 1st, 2010, and December 9th, 2011. The company primarily sells unique all-occasion gifts, with a significant wholesale customer base.

Each record in the dataset represents a line item within an invoice and contains the following key attributes:

InvoiceNo: A 6-digit integral number uniquely assigned to each transaction. If this code starts with the letter 'C', it indicates a cancellation.

StockCode: A 5-digit integral number uniquely assigned to each distinct product.

Description: The product name (textual).

Quantity: The quantity of each product per transaction (numeric).

InvoiceDate: The day and time when each transaction was generated (timestamp).

UnitPrice: The price per unit of the product in sterling (numeric).

CustomerID: A 5-digit integral number uniquely assigned to each customer. Notably, this field contains a significant number of missing values.

Country: The name of the country where each customer resides (textual).

The presence of missing CustomerIDs and the inclusion of cancellation records are important characteristics that necessitate preprocessing before the data can be effectively used for graph-based modeling, as detailed in the next subsection.

C. Data Preparation

The initial step in our methodology involved preparing the raw Online Retail dataset for graph construction. This required addressing several data quality aspects and filtering records not suitable for our link prediction task:

1) Handling Missing Customer IDs: A significant characteristic of this dataset is the presence of transactions without an associated CustomerID. As our graph model requires distinct customer nodes for interaction modeling, all rows where the CustomerID was missing were removed. This reduced the dataset from the original 541,909 rows to 406,829 records containing identifiable customer information.

2) Filtering Cancellations: The dataset includes records corresponding to cancelled orders, marked by an InvoiceNo starting with 'C'. Since our goal is to model and predict successful purchases, these cancellation records were filtered out, further refining the dataset to 397,924 rows representing valid, non-cancelled transaction line items.

It is important to note that even after these filtering steps, the number of rows (397,924) is substantially larger than the number of unique invoices present in the data (18536 unique non-cancelled invoices). This is because the dataset is structured at the line-item level, meaning each row corresponds to a specific product (StockCode) purchased within a single invoice (InvoiceNo). Therefore, a single invoice containing multiple different products will contribute multiple rows to the dataset.

Fig 1 provides an overview of the data characteristics after these initial preparation steps, illustrating the number of unique values present in each relevant column of the cleaned 397,924-row dataset. This highlights the cardinality of key identifiers like StockCode and CustomerID, which informs the subsequent graph construction phase.

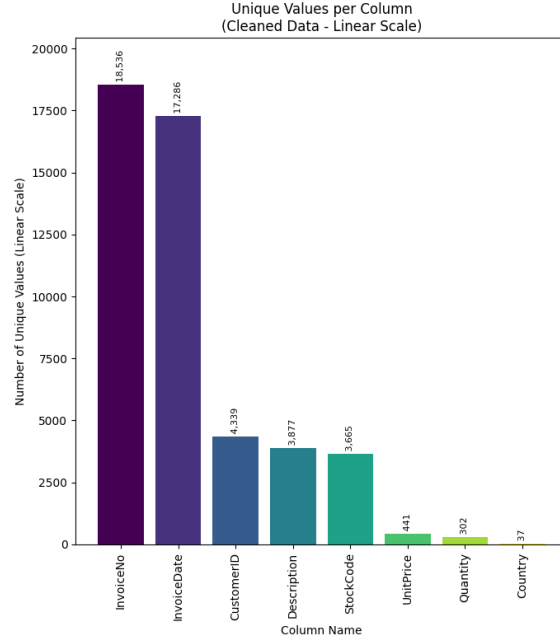


Fig. 1 The plot showing unique values per column, generated from the DataFrame containing 397,924 rows

D. Graph Construction

With the cleaned dataset containing 397,924 valid transaction line items, we proceeded to construct the heterogeneous graph representation required by our GNN model:

1) Node ID Mapping:

We first established unique, consecutive integer identifiers for all distinct customers and products present in the cleaned data. Each unique CustomerID was mapped to an index in the range $[0, \dots, N_c-1]$, and each unique StockCode was mapped to an index in the range $[0, \dots, N_p-1]$. N_c and N_p represent the total number of unique customers and products, respectively, forming the nodes of our graph. These mappings ensure consistent indexing for embedding lookups and graph operations. This resulted in $N_c = 4,339$ customer nodes and $N_p = 3,665$ product nodes.

2) Edge List Creation and Deduplication:

Using the mapped IDs, we generated an edge list representing the 'buy' interactions. For each row in the cleaned dataset, we created a potential edge from the corresponding mapped customer ID to the mapped product ID. As our objective is link prediction (determining if an interaction exists), and the data contains multiple entries for the same customer-product pair (due to repeat purchases), we performed a critical deduplication step. We retained only the unique (customer_id, product_id) pairs from the generated list. This ensures that the final graph used for modeling is a simple graph, where at most one directed edge exists from any customer u to any product v , signifying that at least one purchase occurred. This process yielded N_e unique directed 'buy' interaction edges. This resulted in $N_e = 266,802$ unique 'buy' edges.

3) Heterogeneous Graph Object:

The identified nodes and the deduplicated edges were then organized into a HeteroData object using the PyTorch Geometric library [9]. This object explicitly defines

the graph's structure with two node types: "customer" (size N_c) and "product" (size N_p). The N_e unique 'buy' edges were stored under the specific edge type ("customer", "buy", "product") using PyTorch Geometric's edge_index format (a tensor of shape $[2, N_e]$).

4) Adding Reverse Edges:

To enable bidirectional information flow essential for effective message passing in many GNN architectures, we utilized the `T.ToUndirected()` transform from PyTorch Geometric. Applied to our HeteroData object, this transform automatically created and added the corresponding reverse edges for the ("customer", "buy", "product") type. These reverse edges were assigned the type ("product", "rev_buy", "customer") and added to the HeteroData object, completing the graph structure for GNN input.

This graph construction process resulted in a heterogeneous graph capturing the unique customer-product purchase relationships, ready for the GNN model detailed next, Table I summarizes the graph characteristics.

TABLE I
GRAPH CHARACTERISTICS

Property	Value
Customer Nodes (N_c)	4,339
Product Nodes (N_p)	3,665
'Buy' Edges (N_e)	266,802
'Rev-Buy' Edges	266,802
Graph Density	1.68%

E. Model Architecture

Our proposed model for predicting customer-product interactions leverages a heterogeneous Graph Neural Network structure, designed to learn meaningful representations from the interaction graph. The architecture consists of embedding layers, a GNN encoder, and a link classifier.

GNNs are a class of deep learning models designed to operate directly on graph data [5]. They learn node representations (embeddings) by iteratively aggregating information from a node's local neighborhood through a process often called message passing [14]. This allows GNNs to capture both node features (if available) and, crucially for our task, the complex topological structure of the interaction graph.

1) Input Node Embeddings:

Given the absence of explicit node features in the Online Retail dataset, we employ learnable embedding layers to represent each node type. We initialize two `torch.nn.Embedding` layers: one for customers (`customer_emb`) with dimension (N_c , `hidden_channels`), and one for products (`product_emb`) with dimension (N_p , `hidden_channels`). N_c and N_p are the number of unique customers and products identified during graph construction, and `hidden_channels` is a hyperparameter defining the dimensionality of the embeddings.

- Justification:

Learnable embeddings are essential when rich node attributes are unavailable, allowing the model to learn representations directly from the graph structure and interaction patterns [2]–[3]. The dimensionality, `hidden_channels`, was set to 64. This is a common choice in GNN research, striking a balance between model capacity (allowing for sufficiently complex representations) and computational cost/risk of overfitting associated with very high dimensions.

2) *Heterogeneous GNN Encoder:*

We employ a GNN encoder to propagate information across the graph and refine the initial embeddings based on neighborhood context.

The core architecture is based on GraphSAGE [8], implemented using SAGEConv layers.

GraphSAGE is an influential inductive GNN framework designed to efficiently generate embeddings for nodes in large graphs. Instead of using the full graph Laplacian (like some earlier GCN variants), GraphSAGE learns aggregation functions (e.g., mean, max-pooling) that gather information from a fixed-size, uniformly sampled set of a node's local neighbors. This aggregated neighborhood information is then combined with the node's own current representation and passed through transformation layers to generate the node's embedding for the next layer [8]. This sampling approach enhances scalability and inductive capability.

GraphSAGE is well-suited for this task due to its inductive capabilities and its effectiveness in generating representations by sampling and aggregating neighbor information. We utilize a two-layer GNN structure.

- Justification:

A 2-layer GNN allows each node's final embedding to capture information from its 2-hop neighborhood. This typically provides a good balance between capturing sufficient local structure (direct interactions and interactions of neighbors) and computational complexity [4]. Deeper GNNs can sometimes suffer from over-smoothing, where node embeddings become indistinguishable [13]. Two layers represent a standard and often effective depth for many link prediction tasks.

A ReLU activation function is applied after the first SAGEConv layer to introduce non-linearity, enabling the model to learn more complex functions.

This base GNN is adapted for heterogeneity using PyTorch Geometric's `to_hetero` wrapper [9], informed by the graph's metadata. This crucial step allows the model to learn distinct message-passing parameters for (customer, buy, product) and (product, rev_buy, customer) edge types, respecting the different roles of nodes and interaction directions.

3) *Link Prediction Classifier:*

To generate a prediction score for a potential link between a customer u and a product v , we use their final embeddings (z_u, z_v) generated by the GNN encoder. We employ a dot product mechanism: $\text{score}(u, v) = z_u \cdot z_v$.

- Justification:

The dot product is a simple, efficient, and commonly used similarity function in latent factor models and embedding-based recommendation [3]. It effectively measures the alignment or compatibility between the learned customer and product vectors in the

shared latent space, with higher values indicating stronger predicted affinity. Its simplicity also helps reduce the number of additional parameters in the model.

The overall Model integrates these justified components, processing graph batches to generate link prediction logits.

IV. EXPERIMENTS

This section outlines the experimental methodology employed to train our heterogeneous GNN model and evaluate its performance on the task of predicting customer-product purchase interactions using the graph constructed in the previous section.

A. Experimental Setup

1) Data Splitting:

The interaction edges ("customer", "buy", "product") were partitioned into training (80%), validation (10%), and test (10%) sets using T.RandomLinkSplit [9]. Negative sampling was enabled to generate non-interacting pairs for training supervision. The primary experimental setup, presented first, utilized a negative sampling ratio of 2.0 (two negative samples per positive training edge). For comparative analysis (presented in Section V), setups using ratios of 1.0 and 4.0 were also trained and evaluated. The splitting procedure ensured appropriate handling of reverse edges and included negative samples in the validation and test sets for evaluation.

2) Mini-Batching and Neighbor Sampling:

Training and evaluation were performed using mini-batches generated via LinkNeighborLoader [9] to handle the graph size efficiently.

- *Neighbor Sampling:*

For each target supervision edge in a batch, a 2-hop computation subgraph was sampled using num_neighbors=[20, 10]. This configuration samples up to 20 neighbors for the source/destination nodes (1-hop) and up to 10 neighbors for each of those nodes (2-hop), providing local context for the GNN while maintaining computational feasibility [8].

- *Batch Size:*

A batch_size of 128 supervision edges was used for training, balancing gradient estimation quality and memory usage. Validation and testing used a larger batch size for efficiency.

- *Data Shuffling:*

The training loader employed shuffling (shuffle=True) for each epoch, while validation and test loaders did not (shuffle=False) to ensure consistent evaluation.

3) Model Training Details:

- *Model Configuration:*

The heterogeneous GNN model detailed in the previous section was used, configured with a hidden embedding dimension (hidden_channels) of 64.

- *Optimizer:*

The Adam optimizer was employed with a learning rate of $lr=0.001$. Adam is a standard choice for deep learning models, known for its adaptive learning rates and generally robust performance.

- *Loss Function:*

We utilized the binary cross-entropy loss with logits. This loss is appropriate for binary link prediction tasks where the model outputs raw logits, providing numerical stability by combining the sigmoid activation and loss calculation.

- *Training Duration:*

The model was trained for 50 epochs. This duration was determined based on preliminary observations where both the training loss and key validation metrics showed significant stabilization and diminishing returns beyond this point, suggesting reasonable convergence for comparative purposes.

B. Evaluation Metrics

To assess the performance of our GNN model on the link prediction task, we employed several standard evaluation metrics, calculated on the validation and test sets:

Confusion Matrix: A table summarizing the counts of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

Area Under the Receiver Operating Characteristic Curve (AUC / ROC AUC): This metric evaluates the model's ability to correctly rank positive instances higher than negative instances, irrespective of a specific classification threshold. It plots the True Positive Rate (Recall) against the False Positive Rate at various thresholds. An AUC score of 1.0 represents perfect ranking, while 0.5 signifies random chance performance. AUC is calculated using the raw prediction scores (logits) from the model.

Accuracy: The proportion of all predictions (both positive and negative links) that the model classified correctly. It is calculated as $(\text{True Positives} + \text{True Negatives}) / (\text{Total Predictions})$.

Precision: The proportion of predicted positive links that were actually correct. It measures the relevance of the positive predictions and is calculated as $\text{True Positives} / (\text{True Positives} + \text{False Positives})$.

Recall (True Positive Rate): The proportion of actual positive links that the model correctly identified. It measures the model's ability to find all relevant instances and is calculated as $\text{True Positives} / (\text{True Positives} + \text{False Negatives})$.

F1-Score: The harmonic mean of Precision and Recall, providing a single score that balances both metrics. It is calculated as $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$.

Threshold Dependency: Metrics 2 through 6 (Accuracy, Precision, Recall, F1-Score, Confusion Matrix) require converting the model's continuous output scores (logits) into discrete binary predictions (0 or 1). This conversion depends on a chosen classification threshold. Based on preliminary analysis balancing precision and recall (detailed in

Section V), a threshold of -0.5 was selected for reporting these metrics. AUC is calculated independently of this threshold using the raw scores.

V. RESULTS

This section presents the evaluation results of our heterogeneous GNN model trained using the experimental setup described previously. We first report the performance of the baseline configuration and then analyze the impact of varying key hyperparameters.

A. Baseline Model Performance

Our baseline model was configured using the parameters detailed in the previous section: Adam optimizer ($\text{lr}=0.001$), a negative sampling ratio of 2.0 during training split generation, and neighbor sampling parameters $\text{num_neighbors}=[20, 10]$. The model was trained for 50 epochs. Performance was evaluated on the held-out validation set.

Initial evaluation compared classification thresholds of 0.0 and -0.5 on the validation set. While $\text{threshold}=0.0$ yielded higher precision (0.8257), $\text{threshold}=-0.5$ achieved substantially higher Recall (0.7268 vs 0.4784) and a better overall F1-Score (0.7085 vs 0.6058). Given the goal of identifying potential purchase interactions, the improved Recall and F1-Score at threshold -0.5 led us to select it as the primary threshold for reporting classification metrics in subsequent analyses.

The quantitative results for the link prediction task, using different classification threshold are presented in Table II. The corresponding confusion matrix ($\text{threshold} = -0.5$) is visualized in Fig 2.

TABLE II

BASELINE MODEL PERFORMANCE ON VALIDATION SET

Metric	Value (threshold=0.0)	Value (threshold=-0.5)
ROC AUC	0.8591	0.8592
Accuracy	0.7925	0.8007
Precision	0.8257	0.6912
Recall	0.4784	0.7268
F1-Score	0.6058	0.7085

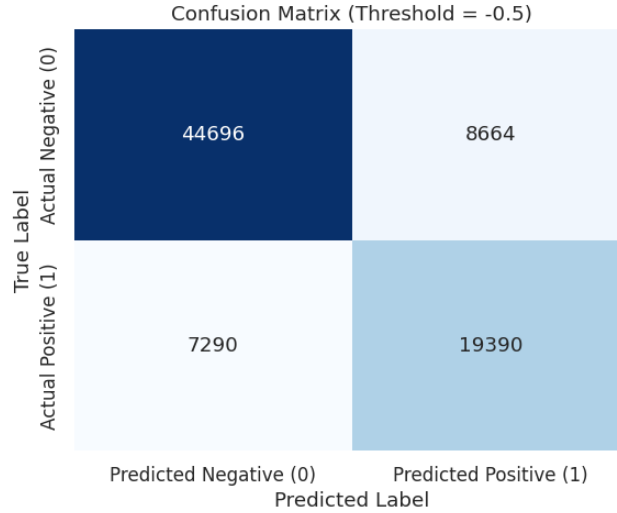


Fig. 2 Confusion Matrix for the baseline model on the validation set, using a classification threshold of -0.5

Interpretation of Baseline: The baseline model achieves a strong ROC AUC score of 0.8592, demonstrating good ranking ability. With the chosen operating threshold of -0.5, the model balances precision and recall reasonably well, achieving an F1-Score of 0.7085 and correctly identifying approximately 73% of the actual positive links (Recall = 0.7268). Fig 2 details the specific counts of true/false positives and negatives at this threshold.

B.Impact of Negative Sampling Ratio

To assess the sensitivity of the model to the proportion of negative examples encountered during training, we trained two additional models, identical to the baseline except for the negative sampling ratio used during the RandomLinkSplit stage: one with a ratio of 1.0 (equal positive and negative training supervision edges) and one with a ratio of 4.0 (four times as many negative edges). Performance was evaluated on the validation set using the same -0.5 threshold.

The results are compared in Table III.

TABLE III

PERFORMANCE COMPARISON FOR DIFFERENT NEGATIVE SAMPLING RATIOS (VALIDATION SET, THRESHOLD = -0.5)

Neg. Sampling Ratio	ROC AUC	Accuracy	Precision	Recall	F1-Score
1.0	0.8563	0.6968	0.6332	0.9352	0.7551
2.0	0.8592	0.8007	0.6912	0.7268	0.7085
4.0	0.8518	0.8351	0.8293	0.2213	0.3494

- Analysis:

The negative sampling ratio significantly influenced the model's predictive behavior, particularly the precision-recall balance, even when evaluated at the fixed -0.5 threshold (Table III).

Using a ratio of 1.0 resulted in the highest Recall (0.9352) and F1-Score (0.7551). This suggests that with fewer negative examples during training, the model learned a representation space or decision boundary that, when combined with the -0.5 threshold, was highly effective at identifying positive instances, albeit at the cost of lower Precision (0.6332) and Accuracy (0.6968).

Increasing the ratio to 2.0 (baseline) improved Precision and Accuracy compared to 1.0, but substantially reduced Recall and F1-Score.

Further increasing the ratio to 4.0 drastically shifted the balance towards Precision (0.8293) and Accuracy (0.8351) but resulted in very poor Recall (0.2213) and the lowest F1-Score. This indicates that training with a large excess of negative samples pushed the model towards being overly conservative in predicting positive links at this specific evaluation threshold.

Interestingly, the ROC AUC score remained relatively stable across the ratios (0.85-0.86), peaking slightly at 2.0. This suggests the model's overall ranking ability wasn't dramatically altered, but the calibration of its scores (affecting performance at a fixed threshold) was highly sensitive to the negative sampling ratio.

Based on these validation results, the model trained with a negative sampling ratio of 1.0 yielded the best F1-Score and Recall at our chosen operating threshold of -0.5. While the ratio of 2.0 served as our initial baseline configuration, we select the model trained with the 1.0 ratio as the preferred configuration due to its superior performance on these key metrics. Therefore, the final evaluation on the held-out test set will utilize the model trained with a negative sampling ratio of 1.0.

C. Test Set Performance

Following the analysis on the validation set, the model configuration trained with a negative sampling ratio of 1.0 was selected as the preferred model due to its superior F1-Score and Recall. To obtain an unbiased estimate of its generalization performance, this final model was evaluated on the held-out test set, using the same classification threshold of -0.5 determined during validation.

The performance metrics achieved on the test set are presented in Table IV.

TABLE IV

FINAL MODEL PERFORMANCE ON TEST SET (NEG. RATIO = 1.0, THRESHOLD = -0.5)

Metric	ROC AUC	Accuracy	Precision	Recall	F1-Score
Value	0.8531	0.6876	0.6251	0.9372	0.7500

- Analysis:

The results on the unseen test set confirm the effectiveness of the selected model configuration. The performance is largely consistent with the validation results,

achieving a high ROC AUC of 0.8531 and an F1-Score of 0.7500. Notably, the model maintained its excellent Recall of 0.9372, successfully identifying the vast majority of positive interactions within the test data at the chosen threshold. This consistency between validation and test performance indicates good generalization and suggests that the model selection process did not significantly overfit to the validation data.

VI. DISCUSSION

Our experiments demonstrate the viability of employing a heterogeneous Graph Neural Network, specifically an adapted GraphSAGE model, for link prediction on the implicit feedback data characteristic of the Online Retail dataset. The model achieved strong performance, particularly in identifying potential customer-product interactions, as evidenced by the high ROC AUC and Recall scores obtained on the test set.

A key finding relates to the significant impact of the negative sampling ratio used during training setup. While the overall ranking ability (AUC) remained relatively stable, the ratio drastically influenced the precision-recall trade-off at our chosen operating threshold (-0.5). Training with fewer negative samples (ratio 1.0) resulted in a model optimized for high Recall and F1-score, suggesting it learned representations particularly effective at identifying positive links when evaluated at a lower decision threshold. Conversely, increasing the negative samples (ratio 4.0) pushed the model towards higher precision but severely hampered its ability to recall actual positive interactions. This highlights the sensitivity of threshold-dependent metrics to this training parameter and underscores the importance of tuning it based on the specific application's goals (e.g., prioritizing recall in recommendation discovery). Our selection of the 1.0 ratio model reflects a prioritization of finding most relevant items (high Recall/F1).

The successful application of learnable embeddings without explicit node features confirms that GNNs can effectively leverage purely structural information from interaction graphs for recommendation tasks. The 2-layer GraphSAGE architecture provided sufficient neighborhood context, achieving good results without introducing the potential complexities or over-smoothing issues of deeper models. The use of a simple dot product classifier proved effective in translating learned embeddings into meaningful link prediction scores.

Limitations of this study include the reliance solely on interaction data, incorporating product or customer features could potentially enhance performance. The analysis focused on a static snapshot of the data, neglecting temporal dynamics which could influence purchase behavior. The cold-start problem for new users or products was not explicitly addressed in this link prediction framework.

Despite these limitations, our findings reinforce the potential of heterogeneous GNNs for implicit feedback recommendation, particularly demonstrating how training configurations like negative sampling can be tuned to align model behavior with desired operational characteristics like high recall.

VII. CONCLUSION

In this paper, we addressed the task of product recommendation from implicit feedback using the standard Online Retail dataset. We presented a methodology encompassing data preprocessing tailored for transactional records, construction of a heterogeneous user-item graph, and the application of a GraphSAGE-based Graph Neural

Network adapted for heterogeneity. By utilizing learnable embeddings derived solely from the graph structure and framing the problem as link prediction, our approach demonstrated considerable effectiveness.

Our experiments yielded strong performance, notably achieving a ROC AUC score of 0.8531 and an F1-score of 0.7500 on the held-out test set with our selected configuration (negative sampling ratio 1.0, threshold -0.5). We particularly highlighted the significant impact of the negative sampling ratio during training setup on the final precision-recall balance, emphasizing the need to align this parameter with specific recommendation goals. These results confirm the viability and potential of applying heterogeneous GNNs to effectively model sparse, implicit interaction data common in real-world retail scenarios.

REFERENCES

- Zhang, S., Yao, L., Sun, A., & Tay, Y., "Deep learning based recommender system: A survey and new perspectives.", *ACM Computing Surveys (CSUR)*, 52(1), 1-38, 2019.
- Rendle, S., Krichene, W., Zhang, L., & Anderson, J., "Neural collaborative filtering vs. matrix factorization revisited.", *Proceedings of the 14th ACM Conference on Recommender Systems*. 2020.
- Koren, Y., Bell, R., & Volinsky, C., "Matrix factorization techniques for recommender systems.", *Computer*, 42(8), 30-37, 2009.
- He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M., "Lightgcn: Simplifying and powering graph convolution network for recommendation.", *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020.
- Wu, S., Sun, Y., Zhang, W., Xie, X., & Cui, B., "Graph neural networks in recommender systems: a survey.", *ACM Computing Surveys (CSUR)*, 55(5), 1-37, 2022.
- Gao, C., Li, Y., et al, "Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions.", *arXiv preprint arXiv:2207.12204*, 2022.
- Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., & Yu, P. S., "Heterogeneous graph attention network.", *Proceedings of The World Wide Web Conference*, 2019.
- Hamilton, W. L., Ying, R., & Leskovec, J., "Inductive representation learning on large graphs.", *Advances in neural information processing systems*, 30, 2017.
- Fey, M., & Lenssen, J. E., "Fast graph representation learning with PyTorch Geometric.", *arXiv preprint arXiv:1903.02428*, 2019.
- Hu, Y., Koren, Y., & Volinsky, C., "Collaborative filtering for implicit feedback datasets." *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, 2008.
- Yang, C., Xiao, Y., Zhang, Y., Sun, Y., & Han, J., "Heterogeneous graph representation learning: A survey.", *arXiv preprint arXiv:2011.09861*, 2020.
- Chen, D., Sain, S. L., & Guo, K., "Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining.", *Journal of Database Marketing & Customer Strategy Management*, 19(3), 197-208, 2012.
- Li, Q., Han, Z., & Wu, X. M., "Deeper insights into graph convolutional networks for semi-supervised learning.", *Proceedings of the AAAI conference on artificial intelligence*, 32(1), 2018.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E., "Neural message passing for quantum chemistry.", *International conference on machine learning*, PMLR, 2017.