# Link Prediction: Product Recommendation on Instacart

Course project for CSE 6240: Web Search and Text Mining, Spring 2023

Kien Tran
Georgia Institute of Technology
Atlanta, Georgia, USA
ktran332@gatech.edu

Tanmay Kenjale
Georgia Institute of Technology
Atlanta, Georgia, USA
tkenjale3@gatech.edu

Yu-Ching Chen
Georgia Institute of Technology
Atlanta, Georgia, USA
ychen3317@gatech.edu

Chase Harrington
Georgia Institute of Technology
Atlanta, Georgia, USA
charrington30@gatech.edu

## ABSTRACT

Product recommendation systems are one of the most popular applications of link prediction. This project reviews various approaches to the link prediction task, which can be broadly described in two steps: creating embeddings/feature vectors to represent each node and using the node embedding to predict the probability of a link between two nodes. To best apply link prediction tasks to real-world data, we use the Instacart Market Basket Analysis dataset from Kaggle. In this paper, we implement four established link prediction methods to recommend products to users in the Instacart dataset: two basline approaches (Matrix Factorization and Node2vec), as well as two more novel approaches (GraphSAGE and Graph Attention Network). We find that GraphSAGE performs the best of the four approaches, achieving an Area-Under-the-Curve score of 0.952.

## 1 INTRODUCTION

In this project, we seek to use the Instacart dataset to construct a tool that can predict which food products will be purchased by users. We frame this problem as a bipartite graph, with one cluster of nodes being users and the other cluster of nodes being products. Edges connect users to products if that product was purchased by that user. Our goal becomes a link prediction problem; we aim to use the structure of the graph to predict which products users are most likely to purchase.

We employ four existing methods to predict links between users and products: Matrix Factorization, Node2vec, GraphSAGE, and Graph Attention Networks. Our experiment results display that the four approaches can achieve decent performance while graph convolutional methods outperformed the two baseline methods, matrix factorization and Node2vec. By successfully creating a link prediction algorithm using the Instacart dataset, users will be able to receive more customized and applicable product recommendations in their Instacart app. More accurate recommendations will increase user satisfaction and potentially increase usage of the app.

## 2 LITERATURE SURVEY / BASELINES

Approaches to the link prediction task can broadly be described in two steps:

(1) Create some embeddings/feature vectors to represent each node (encode)
(2) Use the node embedding to predict the probability of a link between two nodes

Li and Chen [4] proposed a graph kernel framework for a product recommendation problem between users and items and conducted experiments on three real-world data sets. The proposed approach includes constructing the user-item graph from transaction histories, defining the pairwise similarities using random walks path, classifying the positive transaction with one-class SVM algorithm, and ranking the top 10 recommendations for each users. The performance of the graph kernel method is slightly better than other existing approaches, comparing precision (0.0286), recall (0.1461), and f-measure (0.0049). Although the graph kernel procedure made an improvement in link prediction for user-item bipartite graph, we would like to figure out other up-to-date methods with higher accuracy.

Within the domain of recommendation systems, Matrix factorization (MF) is one of the most notable approaches. MF creates embeddings by factorizing the adjacency matrix and using a simple cross-product operation to predict links between user nodes and item nodes. Menon and Elkan [5] implement such an approach and is able to achieve an AUC score of as high as 0.812, where AUC is defined as the Area under the ROC Curve. This value indicates high discriminative ability, and it is suggestive that a matrix factorization approach can be highly accurate for link prediction.

There are multiple approaches to create unsupervised or self-supervised embeddings for link prediction task. One popular approach proposed by Grover and Leskovec [2] is Node2vec. This method utilizes a biased random walk strategy to generate node embeddings. The random walk is controlled by two parameters, which allows the random walk to balance between bread-first search and depth-first search. The results and paths of the repeated random walks are fed through a neural network to output embeddings with structural information for each node. The experiment of link prediction with Node2vec embeddings was also demonstrated in the study [2]. The result, AUC scores, is improved compared with the other feature learning algorithms. However, Node2vec is sensitive to the random walk parameters and it can only consider structural features without taking other node features into account. Despite of these limitations, this approach will be a decent baseline to compare against our other methods because of its popularity and simplicity.

On the front of Graph-based supervised methods, multiple advances have been made to create end-to-end models that can be

trained on supervised tasks. Kipf and Welling [8] proposed the use of graph convolutional networks to aggregate and process information from the embeddings of nodes in the local neighborhood. Hamilton et al. [9] generalize this approach with different (including learnable) aggregation methods and make the task inductive, which means it can be used on new nodes or even new graphs.

Hamilton et al. [3] introduces a framework, GraphSAGE, which can learn a better node embedding by aggregating information from local neighbors of nodes. Also, Saxena et al. [7] built up a recommendation system for financial products, such as funds and ETF. It deployed the GraphSAGE model for learning representations and for predicting the probability that a holder would be interested a financial product (link prediction). The scheme could be applied to our task by predicting the probabilities that a user would buy a product on Instacart. With significant improvements from the model in [7], we expect to build up an effective product recommendation system by implementing GraphSAGE on our dataset. We will specify the details of this method in the Methods section.

Velickovic et al. [6] incorporates the attention mechanism to help a node attend to its neighbors' features and improve performance further. The approach introduced the Graph Attention Network (GAT), which utilizes graph attention layers in its network. The layer inputs node features and performs masked attention, which allows nodes to attend to just the nodes within a local neighborhood. This preserves structural information. The resulting attention coefficients are linearly combined with the input features and a weight matrix to output a new feature set for each node. This can be extended with multi-head attention, which stabilizes model learning. The graph attention layer can be used for prediction or its outputs can be fed into a separate model. The GAT architecture was shown to increase performance in inductive learning by about 27% over GraphSAGE.

Most of the literature on Graph-based link prediction does not specifically address bipartite graphs, which is relevant for recommendation systems. From our survey, we find that researchers [7] usually use the standard version of Graph Neural Network (GNN) architectures for bipartite graphs without major modification.

## 3 DATASET DESCRIPTION

The dataset that we are using for this project is the Instacart Market Basket Analysis dataset from Kaggle [1]. The Kaggle dataset contains seven .csv data files; the two that we are concerned with are 'orders.csv,' which contains order IDs and user IDs among other features, and 'order_products_*.csv,' which contains order IDs and product IDs. Before constructing our graph, we filter out users who have fewer than 10 orders to reduce the amount of noise in the graph.

The resulting dataset consists of user and product nodes. There are a total of 49,443 product nodes and 101,696 user nodes in the graph. The products are organized in 21 departments, each of which corresponds to different types of products such as produce, baked goods, frozen food, etc. There are 9,337,086 edges between the users and products, which correspond to whether or not the user purchased the product. After calculating the degree and eigenvector centrality of the nodes in the graph, we found that the products with the highest centrality measures were in the produce and dairy/eggs departments. This tells us which product categories are the most purchased.

The dataset included multiple features for each purchase, including the order day of week, order hour of day, number of days elapsed since the last order, the sequence of the product purchase in the order, and whether the product has been purchased by the user before. We calculated a weight feature between 0 and 1 for each edge, with a higher weight meaning that the user purchased the product more times than if it was a lower weight. We did this by dividing the number of times a user purchased a product by the number of orders the user made.

We also generated features for each product by converted the product description into text embedding vectors with a pretrained sentence transformer. Additionally, we created users features with 22 dimensions by aggregating the frequency, recency, and monetary value of orders. The product features, and user features created in this step will be called "baseline features" from now on and they will be utilized in the GraphSAGE and Graph Attention models.

## 4 DESCRIBE THE EXPERIMENTAL SETTINGS

We formulate the problem as a link prediction task between a user and an item. Positive samples are created with existing user-item links. Negative samples are generated with sample rate = 1. Thus, we have a balanced dataset for the binary classification task.

We employ temporal splitting at a user-level, described by Yang et al. [10], to create 3 datasets: Train, Validation, and Test. To be more specific, we first created the **positive label edges** for each user by using the new products in the most recent basket (t) for positive test labels, the second most recent basket (t-1) for positive validation labels, and the third most recent basket (t-2) for positive train labels. Then, we constructed the **graph edges** for each set (train/validation/test) using all products bought before the basket that was used to construct the label sets. Lastly, for each positive label in the label set, we generated a **negative sample** between the same user with another random product, thus creating a balanced dataset for the link prediction model. We chose to only use new products to increase the difficulty of the prediction problem since re-purchase is a popular behavior with grocery products.

In terms of evaluation metrics, we used ROC-AUC scores as the main metric to evaluate all models' ability to rank products in the test set. In addition, we also calculate classification metrics such as F1-score, Precision, and Recall for additional context. Traditional ranking metrics such as MRR and MAP were not used for evaluation due to the high complexity of predicting and ranking all 49,443 products for each of the 101,696 users.

Regarding computational resources, we used use High-RAM, standard GPU instances in Google Colab to train our Node2Vec and GraphSAGE models, and premium GPU instances (40GB GPU RAM) to train Graph Attention Network models due to their high-memory requirement.

## 5 METHODS

We implemented 4 methods and compared the results to understand their strengths and weaknesses. Below are the methods we plan on implementing for our recommendation problem:

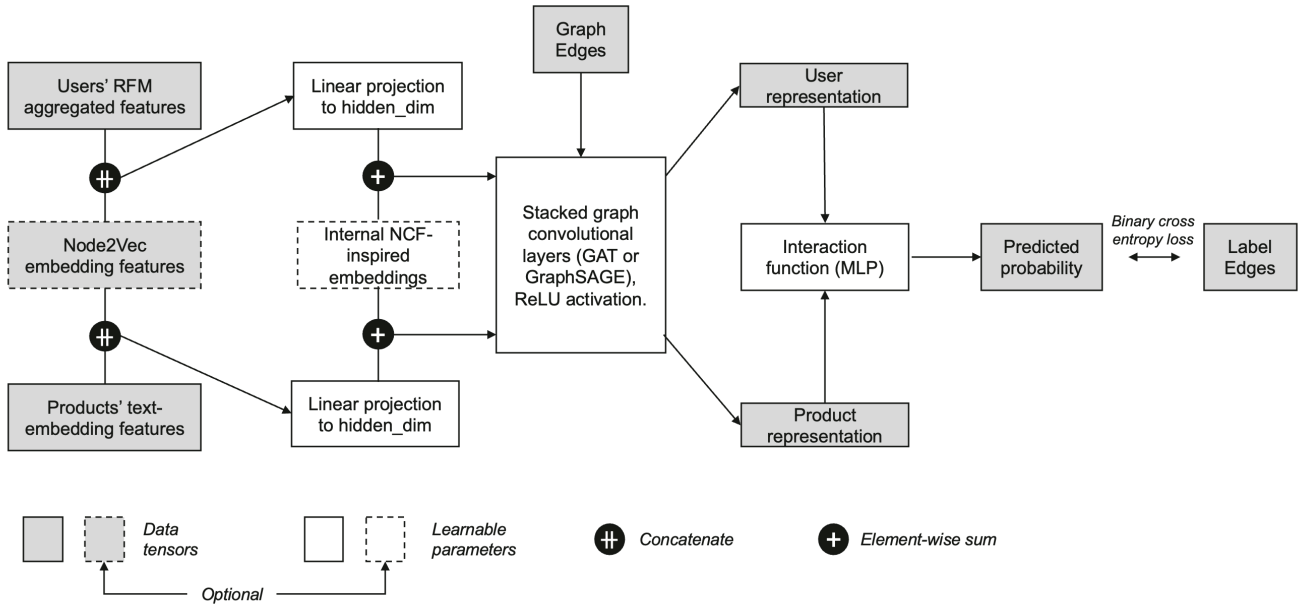


**Figure 1: Link prediction model architecture for GraphSAGE and Graph Attention Network**

## 5.1 Matrix Factorization

As a baseline, we implemented a simple matrix factorization approach to generate embeddings for nodes [5]. We begin by constructing the adjacency matrix for our graph, which is bipartite and sparse. We implement a Stochastic Gradient Descent (SGD) matrix factorization algorithm that introduces bias terms in order to help reduce sparsity and account for the popularity of certain individual items. Once the matrix is factorized, we obtain embeddings for each user and product, and we perform a simple cross-product operation on test set user-product pairs to predict links between them. A high cross-product score indicates a high likelihood of a link existing. This score was used to rank the products in the test set for evaluation. For classification metrics, we define and tune a threshold value; if the embedding cross-product is higher than this threshold, then we predict a link, and if the embedding cross-product is less than this threshold, we do not predict a link.

## 5.2 Node2vec + MLP

We also used Node2vec as a baseline model in our comparison [2]. We used PyTorch Geometric's implementation of the method. We imported all training edges into PyTorch and trained a Node2vec model on the graph with p=1 and q=2 because we regard the problem as breadth-first search as the users would be more interested in neighbor items. Then, we obtained embeddings for each node from the trained Node2vec model. Furthermore, we concatenated the node embeddings as edge embeddings for positive and negative links in training set. Finally, we trained a shallow Multi-Layer Perceptron classifier (MLP) using the popular scikit-learn library with the concatenated embedding of user-item pairs as input to retrieve the probability of link existence. We evaluated the model similarly as described above and present the results in the following section. Also, the Node2vec embeddings will be used as advanced features in the GraphSAGE and GAT models.

## 5.3 GraphSAGE

Figure 1 shows the model architecture that we used to adapt GraphSAGE and Graph Attention Network for the link prediction problem. The node features consisted of the user and item features that we generated as mentioned in section 3. We then linearly projected these features to a hidden dimension. We inputted the hidden state into our SAGEConv layers with mean aggregation to output final representations for users and items. These representations pass through an interaction function to output a probability that a user will purchase an item in the future. Our model was trained with Binary Cross Entropy Loss. During our testing, we experimented with various parameters and techniques. One test was concatenating our initial user and item features with Node2vec embeddings to see if this provided better results. Another test we conducted was adding additional user and item embeddings to the hidden state similar to Neural Collaborative Filtering to potentially alleviate the cold-start problem. We also tested model depths of 2 and 3 and we tested hidden layer dimensions of 128, 256, and 512.

## 5.4 Graph Attention Network

We also implemented a GAT-based model as discussed in [6] and in the literature review. While the paper tested GAT on node classification tasks, we apply it to link prediction on the Instacart dataset by simply modifying the output layer of the model. Our procedure of implementing and testing is very similar to our GraphSAGE procedure mentioned above and simply switches the SAGEConv layers with PyTorch GATv2Conv layers. We tested different hidden layer sizes (128, 256, 512) and number of attention heads (1, 2, 4, 8) to

see how this affects model performance. We also use the Node2vec embedding concatenation technique as used in GraphSAGE. We will evaluate model performance in the same manner as discussed above.

## 6 EXPERIMENTS AND RESULTS

We summarize our experiment results for link prediction on Instacart dataset in Table 1.

**Table 1: Comparing Precision, Recall, F1-measure, and AUC scores between Matrix Factorization, Node2vec, GraphSAGE, and Graph Attention Network.**

| Models | Precision | Recall | F1-measure | AUC scores |
|---|---|---|---|---|
| MF | 0.825 | 0.819 | 0.822 | 0.822 |
| Node2vec + MLP | 0.860 | 0.864 | 0.862 | 0.935 |
| GraphSAGE | **0.863** | **0.912** | **0.889** | **0.952** |
| GAT-based | 0.854 | 0.905 | 0.879 | 0.947 |

### 6.1 Matrix Factorization

The matrix factorization approach performed the worst of the four models that we developed, with an AUC score of 0.822. This is partially to be expected, as matrix factorization cannot always capture the complexity of graphs as well as more advanced methods. Nonetheless, this result is still adequate, and it serves as a high baseline for methods to follow.

### 6.2 Node2vec + MLP

On the other hand, the Node2vec + MLP approach achieved good performance with an AUC score of 0.935. This result shows that the Node2vec embedding incorporates a lot of useful information about the graphical structure of the data for the link prediction task. We use these baseline embeddings for our future models.

### 6.3 GraphSAGE

For the GraphSAGE model, we conducted several combinations of settings described in the previous section. The model that performed the best had 2 layers, used Node2vec embeddings concatenated with the initial features (advanced features), used 256 neurons in each hidden layer, and used mean as the aggregation function. We achieved an AUC score of 0.952.

Our worst performer was the vanilla GraphSAGE without advanced features, showing the importance of this initialization. We also found that adding NCF-inspired embeddings tended to fit very well to the training data quickly, but they led to overfitting. It is expected that GraphSAGE performs better than the baseline models because it is able to incorporate node features and because it can capture more complex relationships with more layers of graph convolution. It also has a good initialization by using Node2vec embeddings, which already capture the graph's structural information.

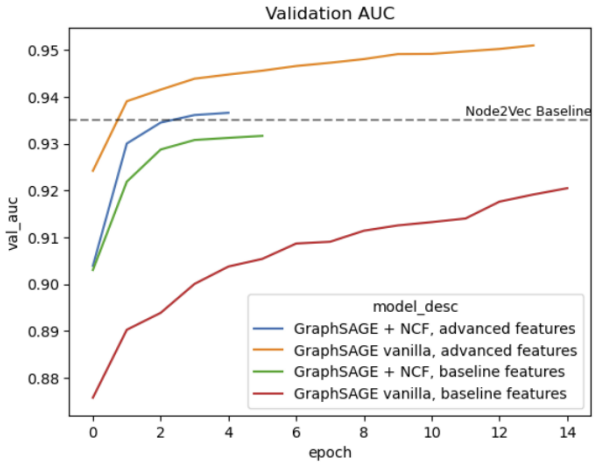


**Figure 2: GraphSAGE comparison between baseline features and advanced features and between regular embeddings and NCF embeddings**

### 6.4 Graph Attention Network

Finally, we conducted experiments with GAT with the same settings as explained in the GraphSAGE experiments. The best GAT model we got during our experiment used 4 attention head and 512 neurons in each hidden layer. We achieved the an AUC score of 0.947. This is lower than that of our best GraphSAGE model, which goes against our research and expectations.

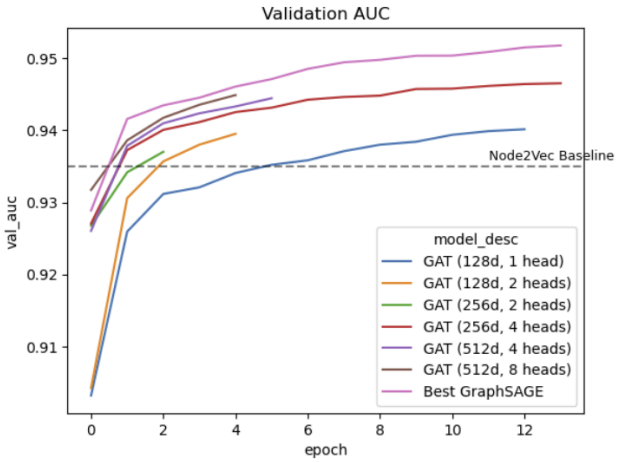


**Figure 3: Graph Attention Network comparison between hidden dimension sizes and number of attention heads**

## 7 CONCLUSION AND FUTURE WORK

In conclusion, our GraphSAGE model performed better on Instacart product recommendation than our Graph Attention Network, Node2vec, and Matrix Factorization models. We expected our GNN-based models to perform better than our baselines, and this was confirmed by our results.

Matrix Factorization was limited by the sparsity of the graph adjacency matrix. When the matrix is extremely sparse, capturing the structure of the graph is difficult, particularly when compounded with temporal factors. Factorizing is also extremely computationally expensive. The performance of the model is the weakest of the four models, although it is by no means weak when standing alone.

Node2vec performed significantly better than Matrix Factorization. However, this method only includes the graph structure of as input and cannot consider node features. This limits its performance, but the extracted embeddings proved useful for our GNN-based models.

Our GAT model performed worse than our GraphSAGE model. We propose that additional experimentation and hyperparameter tuning should be done on GAT to achieve a better product recommendation performance, since we were limited by time and compute resources. We did find that using Node2vec features helped both our GraphSAGE and GAT models greatly. We hypothesize that since Node2vec captures long-distance, graphical information in its random walks, it is able to provide a good feature set for GNN-based models. It is worth exploring other model architectures and techniques that can better capture local and long-distance graphical information for more accurate product recommendations. We also found that the user and item features we engineered had a strong impact on model performance. This leads us to believe that more extensive feature engineering should be tested for better link prediction.

Further research into link prediction for product recommendation will increase user satisfaction, engagement, and increase our understanding of customer behavior.

## 8 CONTRIBUTION

All team members have contributed a similar amount of effort.

## REFERENCES


[1] 2017. "The Instacart Online Grocery Shopping Dataset 2017", Access from https://www.kaggle.com/c/instacart-market-basket-analysis on Feb 25, 2023.

[2] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. arXiv:1607.00653 [cs.SI]

[3] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[4] Xin Li and Hsinchun Chen. 2013. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems* 54, 2 (2013), 880–890.

[5] A.K. Menon and C Elkan. 2011. Link Prediction via Matrix Factorization. *Machine Learning and Knowledge Discovery in Databases* (2011).

[6] Arantxa Casanova Adriana Romero Pietro Liò Yoshua Bengio Petar Veličković, Guillem Cucurull. 2017. Graph Attention Networks. *arxiv preprint arXiv:1710.10903* (2017).

[7] Rachna Saxena, Abhijeet Kumar, and Mridul Mishra. 2022. Holder Recommendations using Graph Representation Learning & Link Prediction. *arXiv preprint arXiv:2212.09624* (2022).

[8] Max Welling Thomas N. Kipf. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arxiv preprint arXiv:1609.02907* (2016).

[9] Jure Leskovec William L. Hamilton, Rex Ying. 2017. Inductive Representation Learning on Large Graphs. *arxiv preprint arXiv:1706.02216* (2017).

[10] Nitesh V. Chawla Yang Yang, Ryan N. Lichtenwalter. 2014. Evaluating link prediction methods. *DOI 10.1007/s10115-014-0789-0* (2014).