Next Product Recommendation and Prediction Amazon KDD Cup '23

Written by DSI Students[1]*
Jiaying Liang[†], **Sovann Chang**[†]
Data Science Institute at Vanderbilt University
1400 18th Ave S Sony Building, Suite 2000
Nashville, TN 37212 USA
jiaying.liang@vanderbilt.edu sovann.d.chang@vanderbilt.edu
https://github.com/Liang-Jiaying/SNA-Product-Recommendation

Abstract

With this project, we aim to predict the next item that a user will click on in a browsing session. To make this prediction, we intend to take advantage of three types of data. The first is a list of products that the user has previously clicked on in the session. The second is a graph generated from historical clicking habits from other users. The third is information about each product in our dataset, including title, price, brand, color, size, and description. All of this data was taken from the Amazon KDD Cup Multilingual Shopping Session Dataset, but only the data from United Kingdom sessions and products was used. To learn from this data, we create a graph neural network with three convolutional layers and one fully connected layer. The results from our model show that it performs significantly better than random predictions, indicating that it has learned something from the training data. We use these results as evidence that this model could be used to accurately recommend products to users if given sufficient time and data.

Introduction

With the rapid growth of e-commerce platforms, personalized product recommendation systems have become increasingly important for enhancing user experience and driving sales. In this project, we aim to develop a next product recommendation and prediction model using the Amazon KDD Cup Multilingual Shopping Session Dataset.

The dataset consists of two main components: a product catalog containing attributes such as title, brand, description, color, and size, and a record of user sessions, which contains the sequence of products viewed by a user during a particular browsing session. The primary goal of this project is to predict the next product that a user is likely to click on or view, given the sequence of products previously visited within the same session.

To tackle this challenge, we propose a hybrid approach that combines graph neural networks (GNNs) and natural language processing (NLP) techniques. GNNs are well-suited for modeling the structural relationships between products, as they can capture the underlying graph structure of user-product interactions. Simultaneously, NLP models are employed to process textual data, such as product titles and descriptions, enabling the incorporation of latent semantic information into the recommendation process.

By integrating these two powerful techniques, we aim to develop a recommendation system that not only considers the sequential patterns in user browsing behavior but also leverages the rich semantic content of product information. If given enough training data, compute, and memory, this approach has the potential to provide more accurate and relevant recommendations, enhancing the overall user experience on e-commerce platforms.

The following sections of this paper will delve into the details of our methodology, including data pre-processing, model architecture, and training procedures. We will also present the results of our experiments, discuss the findings, and outline potential future research directions.

Data

Data Cleaning

To improve the effectiveness and reliability of our recommendation model, we performed several data pre-processing steps on the Amazon KDD Cup Multilingual Shopping Session Dataset. The dataset initially contained product and session information from multiple countries. However, we limited our scope to data originating from the United Kingdom (UK) for many reasons, including limiting the size of the data and ensuring that the data is not complicated with multiple languages.

Below are the steps for cleaning two original datasets.

For product data

1. **Locale Filtering**: We filtered the product dataset to include only entries with the 'UK' locale, discarding products from other countries.

*With help from the DSI-Social Network Analysis class instructor Dr. Tyler Derr.

†These authors contributed equally.

2. **Price Filtering**: We observed a large number of products with unrealistically high prices (e.g., $40,000,000). To mitigate the impact of such outliers, we removed products with prices exceeding $500 from our dataset.
3. **Brand Consolidation**: The product catalog contained over 76,000 unique brand names, many of which had fewer than five associated products. To reduce sparsity and improve model performance, we filtered out rare brands which are brands have less than 50 associated products.
4. **Feature Engineering**: We explored the potential of incorporating additional features, such as product colors and product materials, into our model. However, after analyzing the data, we decided to combine all textual features, including product titles, brands, colors, models, materials, and descriptions, into a single text field. We dropped the author column since most of the data is missing this information, and the author of a product may not be particularly indicative of future click habits.

For training session data

1. **Locale Filtering**: Like the product data, we filtered the session data to include only UK-based sessions.
2. **Previous Items Sequence Cleaning**: We cleaned the previous item sequence in user sessions, which was initially represented as a string of product IDs. To make the data consistent with our modeling approach, we converted this string into a list of strings, with each element representing a product ID. This involved parsing the string, extracting individual product IDs, and organizing them into a structured list format.
3. **Session Length Feature**: To incorporate session length information into our model, we created a new feature that captured the number of previous items in each session.
4. **Session Filtering**: We filtered sessions with less than three previous item interactions to improve sequential pattern learning. This step ensured that the model had an adequate dataset for accurate predictions. Sessions meeting this criterion were retained for further analysis.
5. **Session in Product List**: We filtered sessions to include only those containing products filtered by locale, price, and brands as per the product data cleaning process. This ensured that the sessions aligned with the cleaned product dataset, maintaining consistency in the data used for analysis and modeling. Sessions containing products meeting these criteria were retained for subsequent processing.
6. **Further Session Trimming**: After the filtering in steps 4 and 5, we were left with around 180,000 training sessions. In order to make this more manageable, we trimmed the data to 20,000 rows.
7. **Previous Items Trimming**: We trimmed the previous item sequence in sessions to focus on the most recent interactions. Recognizing that a lengthy sequence may not significantly contribute to predicting the next item compared to the last few interactions, we retained only the last five items. This trimming process aimed to improve the model's ability to capture recent user behavior for more accurate predictions.

After this round of data cleaning, our dataset became significantly smaller compared to the original data. We revisited the items in the filtered sessions, which contained fewer products, and applied another round of filtering to the product data. This step ensured that we only kept products included in the newly processed sessions.

After cleaning the training dataset, we applied the same cleaning process to the test data. The test data was provided separately on the competition website. Initially, we filtered it based on the valid products criteria. Subsequently, we trimmed it down to retain only the last five previous items, mirroring the approach used for the training sessions data. Where the training data was trimmed to 20,000 rows, the testing data was trimmed to 5,000 rows

By implementing these pre-processing steps, our goal was to extract a high-quality dataset that would enable our recommendation model to learn meaningful patterns and provide accurate predictions while maintaining computational efficiency.

Feature Engineering

After reducing the size of the data, it was still not ready to be used to train the Graph Neural Network model. We needed to convert the text column in the products data to be numeric. We also needed to generate a graph representation of the sessions data. The following feature engineering steps were taken to further prepare the data.

For the products data

1. **Text Pre-processing**: We utilized a small BERT model named "all-MiniLM-L6-v2" to produce 384-dimensional embeddings for the text field created during the previous data cleaning phase.
2. **Price Descaling**: To normalize the price column, we divided it by the maximum price in the column.

For the sessions data

Edge List: From the training sessions obtained during data cleaning, we divided each session into a dataset comprising the previous item and the next item. This resulted in a new dataset with two columns: previous item and next item. Subsequently, we grouped these columns to determine the frequency of each combination, representing the weight of the corresponding edge. The final dataframe contained three columns, representing the previous item, the next item, and the frequency - how many times a user went from previous item to next item across all the training data.

Changing Product IDs to Index Numbers Up to this point, the original product IDs were still in use. To ensure compatibility with the GNN model, we replaced these IDs with their respective index numbers in the sorted product ID list.

For the products Data

Index Column: Created an index column by resetting the index of the products data.

For the sessions Data

1. **Edge Index List**: Earlier in the feature engineering phase, we represented the previous item and the next item in the edge index list using product IDs. We subsequently replaced these product IDs with the index numbers assigned in the previous step.
2. **Training Test Sessions**: The sessions consist of product IDs, which we replaced with their respective index numbers. During model training, this simplifies input requirements as we only need to provide the list of product index numbers in each session. The model then accesses the corresponding features in the product list based on these index numbers.

Building the Graph Representation

Using our edge index list, we created a networkx graph to represent each product (node) and each click from one product to another (directed edge). This graph was put into adjacency matrix format, then represented in coordinate (COO) format. The rows and columns were then concatenated and converted to a PyTorch tensor, forming a representation of the edge indices.

Dimensionality Reduction of Feature Embeddings

In the products data, there were 385 features (1 feature for price, 384 features for text description embeddings), which was too large for our computational resources. To address this, we opted to reduce these 385 features to 32 features using a single forward pass in a Graph Convolutional Network. The GCN is shown below, both in code and as a high-level overview, so that the reader can choose what to digest.

The code is shown below.

```
class GCN(torch.nn.Module):
    def __init__(self, num_features,
                 hidden_channels, output_dim):
        super().__init__()
        self.conv1 = GCNConv(num_features,
                             hidden_channels)
        self.conv2 = GCNConv(hidden_channels,
                             output_dim)
        self.double()

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)
```

Algorithm 1: The architecture of the GCN to reduce our feature dimensions:

Algorithm 1: Graph Convolutional Network (GCN) for Feature Embedding

Input:
num_features - Number of input features
hidden_channels - Number of hidden channels
output_dim - Dimension of output features
Parameter: GCNConv - Graph Convolutional layer
Output: Log-softmax output of the GCN

1: Initialize GCN model with parameters num_features, hidden_channels, output_dim.
2: Initialize GCNConv layers: conv1 and conv2.
3: Set model to double precision for numerical stability.
4: Define the forward pass function with input features x and edge_index.
5: Perform first graph convolution with conv1.
6: Apply ReLU activation function.
7: Perform second graph convolution with conv2.
8: Apply log-softmax function along dimension 1.
9: **return** Log-softmax output x.

After completing the detailed data cleaning and processing steps, we were ready to proceed to the modeling phase.

Methods

This methodology section delves into the strategies employed to achieve our goals. Specifically, we crafted a second Graph Neural Network (GNN) model tailored for training on the sessions, product features, and graph. This section explains the model architecture construction and training methodologies aimed at maximizing system performance.

Modeling Architecture

The GNN's architecture consists of a flexible number of convolutional layers (minimum 2) followed by a fully connected layer. The forward pass accepts our three inputs: the list of previous items (prev_items), the feature matrix (x), and the graph, represented in edge index format (edge_index). The output of the GNN is 15,386 dimensions. The truth labels are one-hot encoded, with 15,386 unique products represented by the one-hot encoding. The GNN is shown below, both in code and as a high-level overview, so that the reader can choose what to digest.
The code is shown below.

```
class GNNPredictor(torch.nn.Module):
    def __init__(self, hidden_dim, output_dim,
                       num_layers):
        input_dim = 32
        fc_input_dim = input_dim * 5

        # Want at least 2 layers
        if num_layers < 2:
            num_layers = 2

        super(GNNPredictor, self).__init__()
        self.conv_layers = torch.nn.ModuleList(
                        [GCNConv(input_dim, hidden_dim)])
```

```
        if num_layers > 2:
            for _ in range(num_layers - 2):
                self.conv_layers.append(
                        GCNConv(hidden_dim, hidden_dim))

        self.conv_layers.append(
                        GCNConv(hidden_dim, input_dim))

        self.fc = torch.nn.Linear(fc_input_dim, output_dim)

    def forward(self, prev_items, x, edge_index):
        # Apply graph convolution layers
        for conv in self.conv_layers:
            x = F.relu(conv(x, edge_index))

        # Get updated representation of previous items
        new_prev_items = []
        for prev_list in prev_items:
            new_prev_list = []

            for item in prev_list:
                new_prev_list.extend(x[item].tolist())

            new_prev_items.append(new_prev_list)

        # Apply fully connected layer
        new_prev_items = self.fc(
                            torch.tensor(new_prev_items))
        return F.softmax(new_prev_items , dim=1)
```

Algorithm 2: The architecture of the GNN, built to be trained and predict the next product in a sequence

Algorithm 2: GNNPredictor Class

Input:
hidden_dim - Dimension of hidden layers
output_dim - Dimension of output features
num_layers - Number of graph convolution layers
input_dim - Dimension of input features
prev_items - List of previous items
x - Input features
edge_index - Edge index of the graph
Parameter: GCNConv - Graph Convolutional layer
Output: Softmax output for predictions

1: Initialize GCN model with parameters hidden_dim, output_dim, num_layers, input_dim
2: Initialize GCNConv one layer
3: Add more GCNConv layer as hidden layers
4: Apply a fully connect linear layer
5: Apply ReLU activation function on each conv layers
6: Get updated representation of previous items
7: Apply the fully connect layers to the updated previous items
8: Apply softmax function.

Using the GNN architecture shown above, we passed in our 20,000 training samples so that the neural network could learn the patterns in the data. We ran the entire batch through at once rather than using stochastic or mini-batch descent. We chose 50 epochs of training, which took around 30 minutes.

Results

After training the model, we set the model to evaluation mode and passed in the 5,000 testing sessions. For each session, the output was one 15,386-dimensional softmax, with each element of the softmax representing an approximate probability that the next item is a unique product.

Because there are over 15,000 potential next products, the probability of predicting the exact item is very low, so we did not feel that the best way to measure success was to simply count the number of next products that were assigned the highest probability. Instead, we arranged the product probabilities in descending order and located the zero-indexed position of the actual next product. This means that if the true next product had the highest probability of any product, it would be ranked at position 0. If the true next product had the 10th highest probability, it would be ranked at position 9. We recorded the ranking for each true next item, and used that to estimate our graph neural network's performance.

Below is a histogram of the 5,000 product ranks, split into 100 bins. In addition to the histogram, we measure the number of times that the actual next product was given the highest probability or was in the top 10, 25, 50, and 100 most likely items.

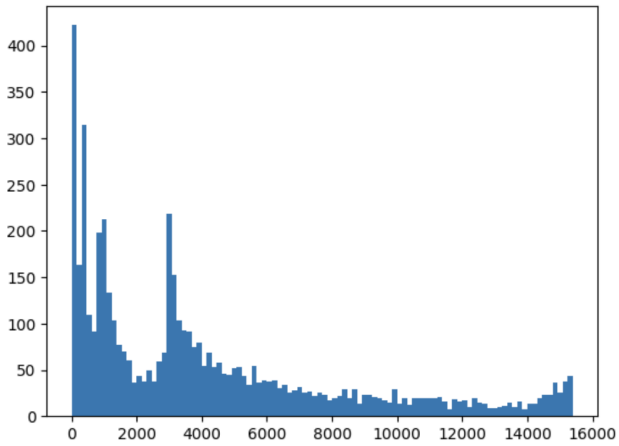

Figure 1: Rankings of the next product in the probabilities of all products. A 0 represents the top probability, a 1 represents the second highest probability, etc.

Most likely	Top 10	Top 25	Top 50	Top 100
8	64	124	203	315

Table 1: The number of times that the actual next product was in the top X most likely next products

Note that the numbers in the table are not exclusive. That is, if the true next product was given the a probability within the top 25, it also contributes to the counts for top 50 and top 100.

Discussion and Conclusion

Due to the sheer volume of potential next products and fairly limited size of our training and testing sets, we did not believe that evaluating the model based on pure accuracy of the single most likely next product was the most appropriate metric. To understand why, we think about the odds of successful predictions using a model that predicts a random product each time. With 15,386 unique products and only 5,000 test sessions, the expected number of correct next product predictions is zero. In fact, there is only around a 1/3 chance of predicting even one next product correctly. An expected accuracy of zero from a random model makes an evaluation of pure accuracy not particularly useful.

This is the reason we turned to a more subjective analysis of the model's performance, comparing it to a random model without official statistical analyses. Beginning with the histogram, we can clearly see that the model has identified something useful in the data. A random model would produce a flat histogram, with each of the 100 bins containing around 50 samples. Our GNN is significantly right-skewed for a statistic where a smaller value indicates better performance. The first bucket contains over 400 samples, and the majority of the data falls within the top 4,000 predicted items.

Turning to the table, we can see a similar indication of good performance. We show the same results as above, this time compared to the expected number of true next products that would fall within the top X products by a random model.

	Top 1	Top 10	Top 25	Top 50	Top 100
GNN	8	64	124	203	315
Random	0	3	8	16	32

Table 2: The number of times that the actual next product was in the top X most likely next products compared to the expected results from a random model

As we can see, our GNN significantly outperforms a random model. Of course, some improvement is to be expected, as a random model is the absolute baseline. However, we believe that our GNN is not simply better, but multiple volumes better than a random model. Our model has more than 20 times as many rankings of the true next item within the top 10 as the average random model and nearly 10 times as many in the top 100. Given these results, we believe our model to be a success.

Limitations and Future Work

We acknowledge that much of our model design and data cleaning was done in the interest of reducing the time and memory required. As students, we do not have access to the resources required to store tensors representing the full dataset or an un-reduced feature matrix. Given additional resources, we believe our model could have performed even better.

With more time, we could have attempted some different methods that are now reserved for potential future work. We did not investigate the performance of mini batch or stochastic gradient descent. We had hoped to investigate the best hyperparameter values, from number of convolutional layers to hidden dimension size to number of epochs of training. Although we do believe that the model could be further improved by changing some of these methods and hyperparameters, we are content with its performance as is.

References

Our train and test datasets can be found at the following link: https://www.aicrowd.com/challenges/amazon-kdd-cup-23-multilingual-recommendation-challenge/problems/task-1-next-product-recommendation/dataset_files