**Personalized Product Recommendation System**

***Team Name: Trojan Horses***

**Created By:**

Rishabh Jain

Mayank Shukla

Guneesh Vats

Github link: <https://github.com/rishabhj2912/Personalized_product_recommendation>

Link to the video: <https://drive.google.com/file/d/1ojZ7Zkke2T3uy_TVU0iMEsf-vt26oKsC/view?usp=sharing>

**1 Data Understanding**

**1.1 Data Source**

The original datasets, which contain 5 months (Oct 2019 - Feb 2020) behavior data among 1639358 customers and 53904 unique products from a medium online cosmetics store, were obtained from Kaggle (https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-cosmetics-shop). Each data instance represents an event (view, add to cart, remove from cart, or purchase) between a user and a product during a visiting session to the website. 98.25% of the product category textual information and 41.95% of the product brand information is missing. Other than the pricing information, we are only left with categorical ids, which are not easy to make sense of without corresponding textual explanation.

**2 Data Preparation**

**2.1 Data Cleaning**

The five monthly datasets (Oct 2019 - Feb 2020) were merged together and duplicated rows were removed as they might have been collected due to some technical errors. Products with non-positive prices are filtered out as these products are probably some complimentary gifts, which are some temporary products that we can’t recommend to customers. Only users who brought at least 10 different products are kept to construct our user-product matrix. The rows of the matrix represent unique users and columns represent products. The (i,j)-entry represents the number of times that user i purchased product j.

When constructing the user-brand matrix, we filtered out rows with missing brand information. Since 41.95% of the product brand information is missing in the original datasets, we didn’t treat them all as a single“no name” brand as this “no name” brand would be the dominant brand with the most popularity. The user-brand matrix was constructed with users who have purchased at least 5 different brands. The rows of this matrix represent unique users and columns represent unique brands. The (i,j)-entry represents the number of times that user i purchased brand j.

We filtered the data to include only users with enough purchase records so that our recommender systems will have enough signals to learn from.

**2.2 Train, Validation, and Test split**

For a given user in our matrices, 20% of his or her purchase history is randomly selected to be in the test set, 20% of the purchase history in the validation set, and the remaining 60% of the purchase history in the training set. Therefore, the training, validation, and testing matrices have the same size and the entries of one matrix are zeroed out in the other two matrices.

**2.3 Target Variable**

To build the recommender system, we would like to know people’s preferences towards the products and brands. Since the datasets don’t have any explicit feedback such as users’ ratings, we decided to rely on the variable `event\_type` which conveys implicit feedback through users’ actions of view, add to cart, purchase and remove from cart. We consider `purchase` as the signal of preference towards certain products and brands. Therefore, we defined the target variable as the number of purchases for both products and brands.

**2.4 Feature Engineering**

To generate a numerical quantification of users’ preferences on certain brands or products from the raw browsing history of customers, we firstly combined all browsing data from October 2019 to February 2020 together and then grouped the whole dataframe by unique *user\_id*s.

For user-brand relationships data, we summed the purchase counts for each user-brand pair that had at least one ‘purchase’ event associated with them in the combined dataframe. Numerical *brand\_id* is also generated for corresponding *brand* information. See Table 3. In Appendix for a snippet of the sample data.

The user-product relationships data was generated in the same way as the above mentioned user-brand relationships data. See Table 4. In Appendix for a snippet of the sample data.

After generating user-brand and user-product relationships data, we then generate the user-brand and user-product matrix where each entry indicates how many times the user has purchased that brand or product from October 2019 to February 2020.

**3 Modeling & Evaluation**

**3.1 Evaluation Metrics**

The recommendation systems output a sorted list of items for every customer. Our goal is to have the sorted list accurately match customers’ preferences. Hence, Precision at K is an appropriate evaluation metric for model performances, where K is the number of products/brands to recommend for a specific user. For a given user:

To evaluate model performance, the metric is averaged over all customers. The number K is determined by business constraints. Since we do not know what would be an appropriate K for this particular online store, we calculated the mean Precision@K for a list of K values that we think are reasonable (K ∈ [1, 30]).

**4. Model : Neural Network**

With deep learning frameworks from Keras, we built a collaborative filtering recommendation system with neural networks. This approach leverages the embedding layers to understand the interactions between the users and products or brands. Incorporating deep learning techniques help to tackle some drawbacks of traditional collaborative filtering. This approach also incorporates the ideas from matrix factorization by using inner product on the latent features of users and products.

**4.1 Structure of Neural Networks**

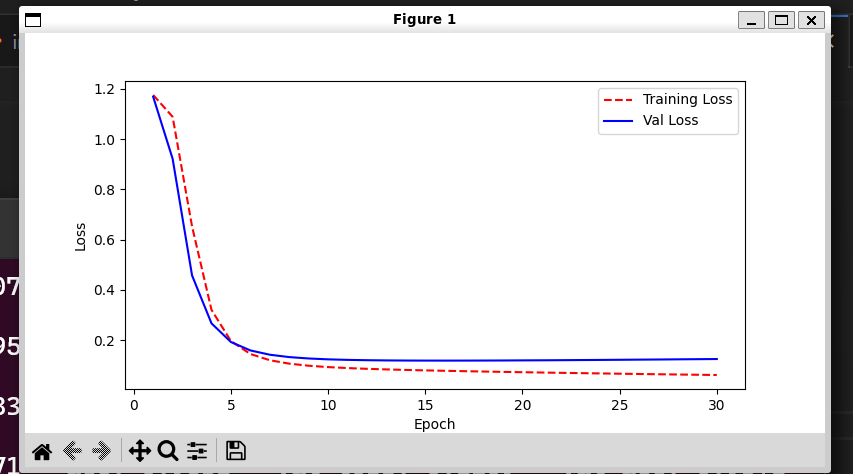
Our neural network consists of input layers for both users and items, embedding layers and reshape layers for users and items, and a dot layer combining the user embedding and item embedding via dot product. The embedding layers are important in recommender system implementation because they are used to map categorical objects such as user IDs with similarities.

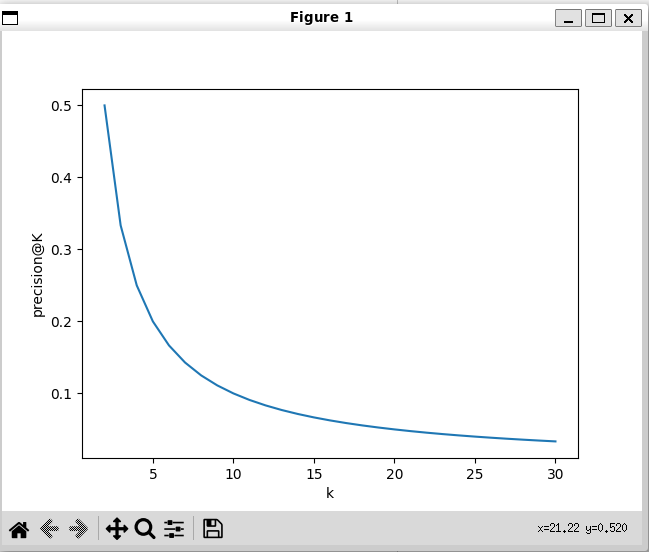
**4.2 Tuning hyperparameters**

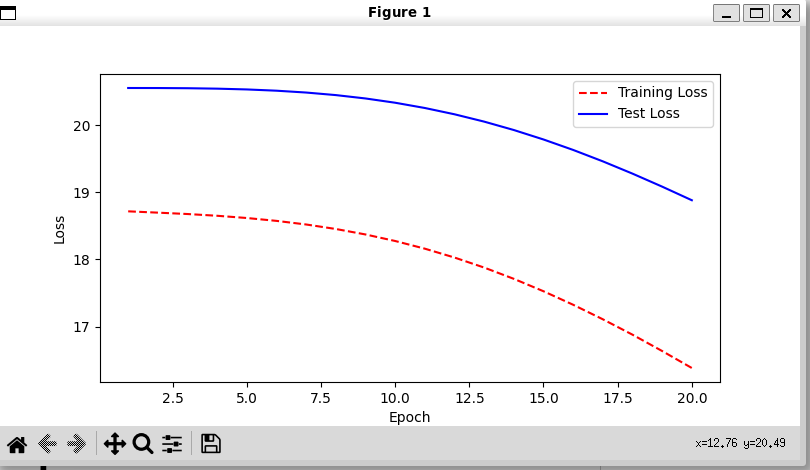
Within the embedding layers, there are many hyperparameters such as the number of epochs, learning rates, batch sizes, dimensions of embedding layers and number of latent factors that can be tuned for better performance. The graph below shows the learning curve of loss against the number of epochs. The optimal number of epochs is the elbow point of the learning curve. Besides tuning parameters, model performance can also be improved by changing optimizers and adding dropout layers. See Figure. 10. in Appendix for validation metric RMSE performance.

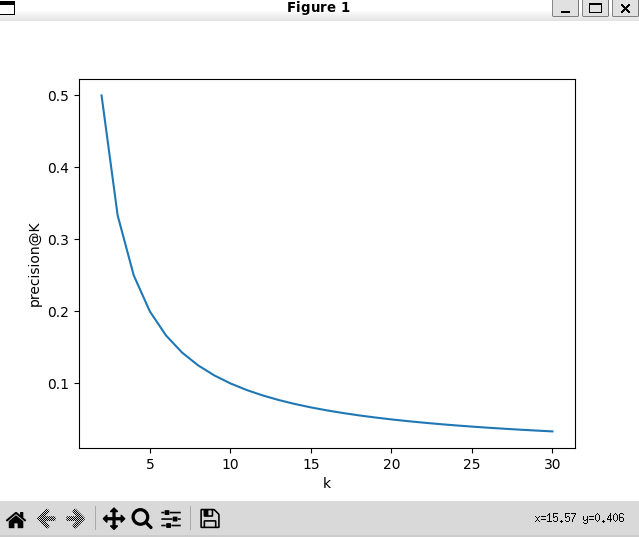
**4.3 Model Results (performance on test data)**

With our selected model, we combined the training data with the validation data, and retained the model on the combined dataset. After predicting on the test set, we got a test precision@k, which is 0.10327541, which is lower than the precision@k on the training dataset.

As the neural network model has a relatively high precision@k, the product recommendations are more likely to satisfy customer needs. Also, deep learning techniques are quite flexible and generally perform better on the larger dataset, so as the firm keeps operating and attracting more new customers, the dataset will continue growing, and, therefore, the neural network model will become more sophisticated to make personalized recommendations. However, we need to be aware of the overfitting issues and control it through regularization.





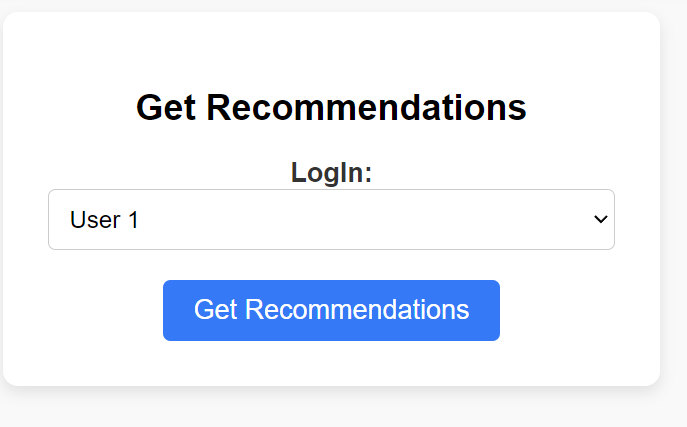


**5. Web Application:**

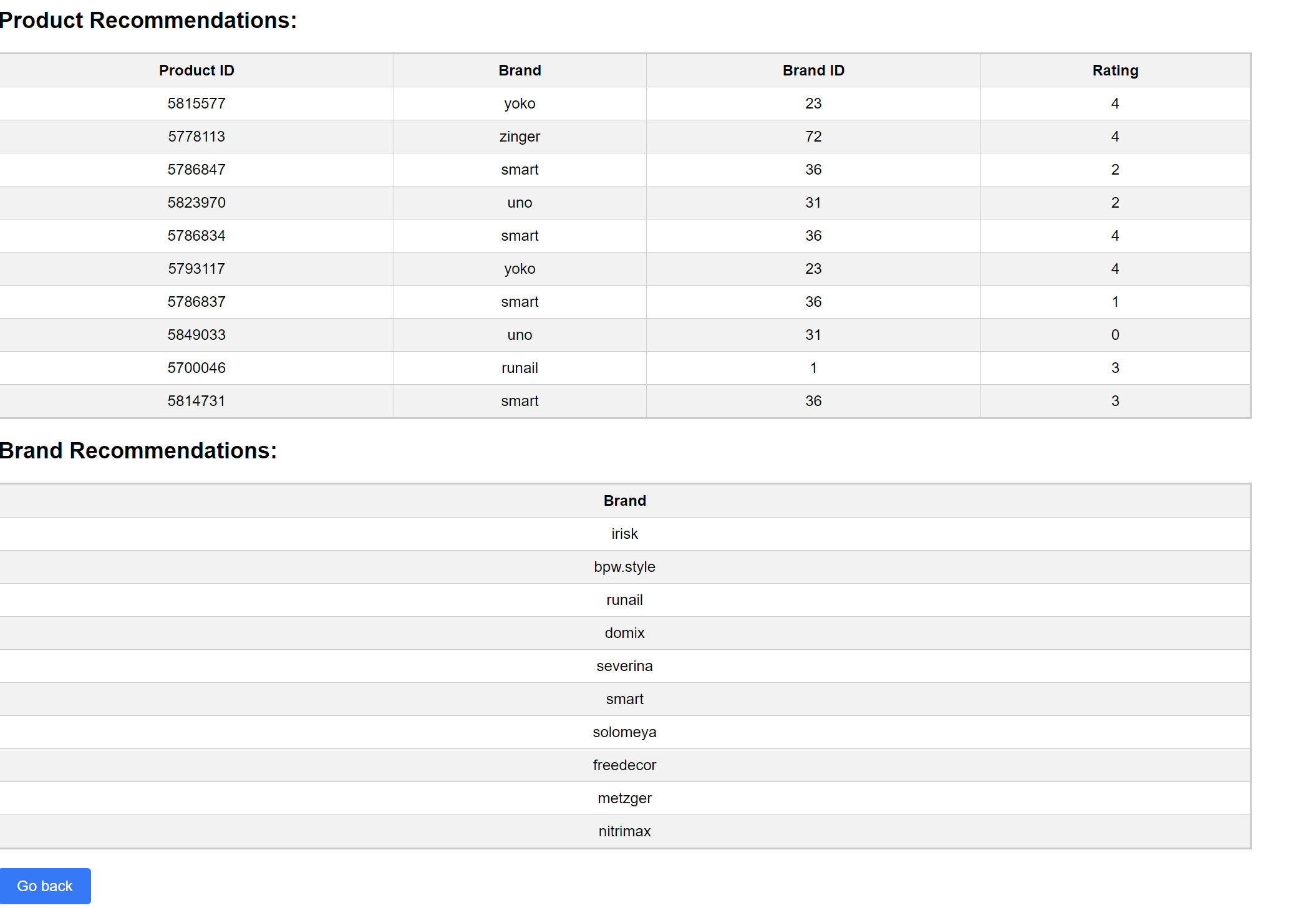
**5.1 Recommendation of product based on User LogIn.**

We gave a drop-down list of some users who are already present in existing database and on logIn of user we recommend the products to that user based on his purchase history.

LogIn page:



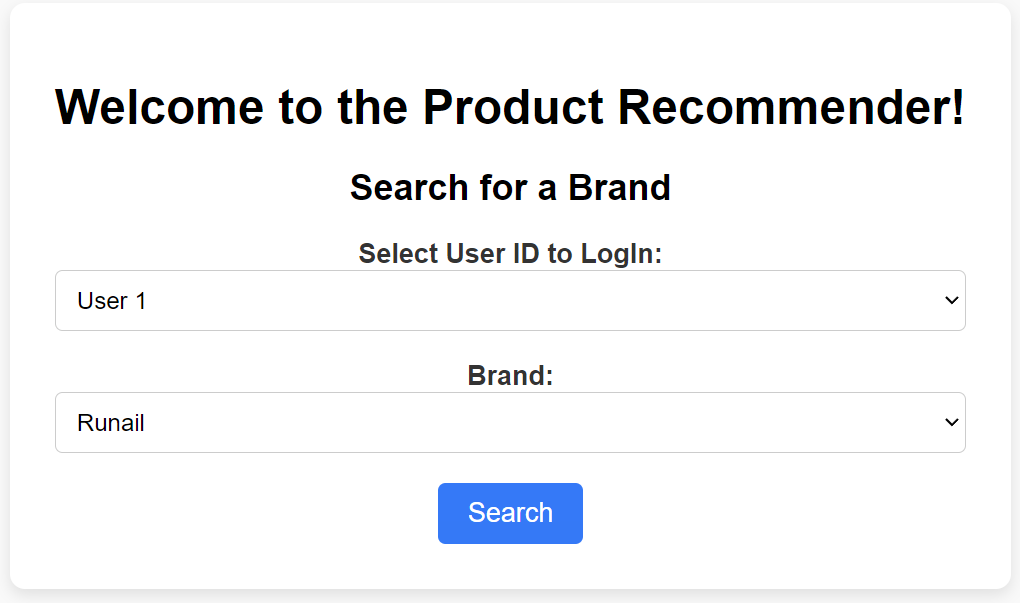
Recommendation Response:



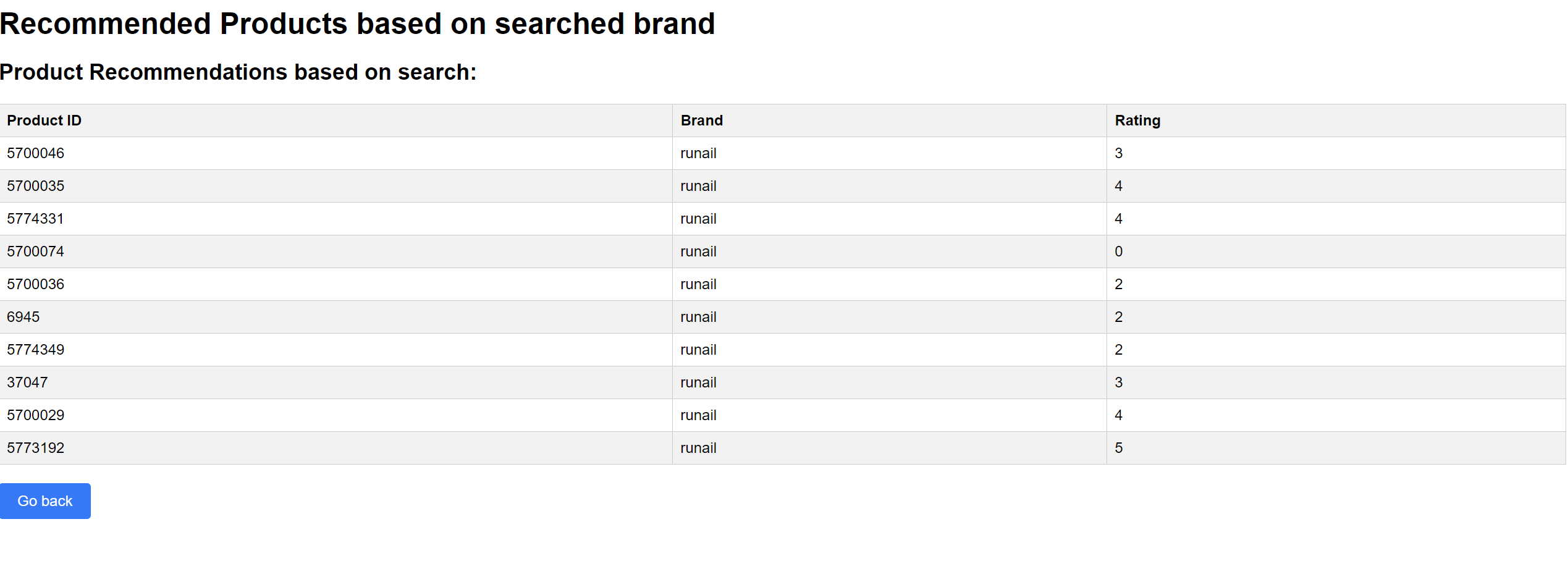
**5.2 Recommendation of product based on Search:**

When user logIn and Search for a brand, we recommend the top 10 products related to that brand.

Search:



Recommendation Response:



Future Work:

1. Dynamic Database

A) When hosted we can use the newer purchases of users to update the recommendations.

1. Selective Attention

A) Current Model equal attention for Purchased and Items in Cart & 0 Attention to items removed from

cart or just viewed.

B) But in future model we can adjust the attention

C) Purchased items > Added to Cart > Viewed Items > Removed from Cart (Negative Attention)

1. Negative Sampling
2. Improved Dataset

A) Categories based on type of product

B) Description of Product (Name, use etc.)

i) We can use BERT embeddings of this description to calculate similarities between 2 products for better recommendations.

Recommender systems builders may firstly group the customers according to explicit or implicit user information and then fine tune models for different groups of customers. RFM (Recency, Frequency, Monetary) analysis is a popular customer analysis technique for user segmentation. Even with only implicit purchasing behaviors data, recommender systems builders can analyze the recency of a customer’s purchasing behavior, the frequency of a customer’s purchasing history, and the monetary value of a customer’s spending.

Furthermore, recommender systems builders can combine explicit user information (if available) with the RFM analysis results to cluster customers into appropriate number of groups for more elaborate model training and evaluation.

Link to the video: https://drive.google.com/file/d/1ojZ7Zkke2T3uy\_TVU0iMEsf-vt26oKsC/view?usp=sharing