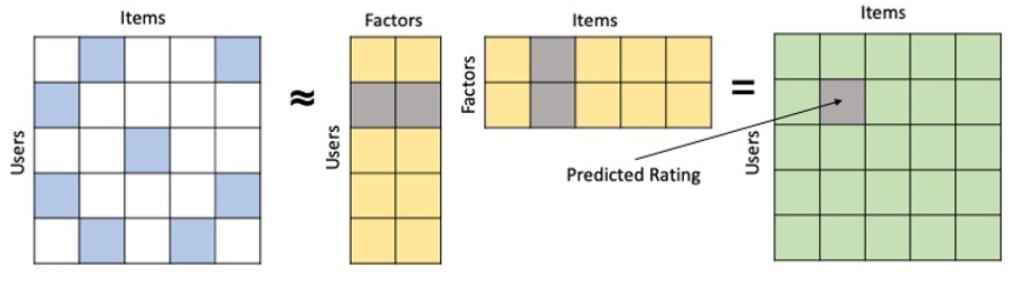
**Problem**

With an increase in online shopping activity, it becomes necessary to develop personalized recommendation systems to optimize shopping experiences [1]. Based on customers’ online purchasing activity, session-level and customer-product journey interactions can be used to target communications and personalize shopping experience strategies for vendors and customers [1]. For example, relevant items can be recommended to each user based on purchasing behavior to encourage users to purchase more items which increase revenue for the vendor. This can be done for new and existing customers. It is reported that 35% of Amazon’s revenue comes from its recommendation engine [2]. Given a multitude of items to choose from and limited customer attention lifespan and real estate space, it is essential to recommend relevant items that a user will be more likely to purchase. In addition, accurate purchasing predictions can ensure adequate product inventory based on customer engagement, which can lead to directed nudge models [1].

**Existing Solutions**

For recommendation systems, an important criteria is to have feedback data that is used as an indicator of whether a user likes an item. There are two types of feedback data, explicit and implicit. Explicit data directly informs a user’s preference, for example rating from a scale of 1 to 5 [3]. However, such data is not always available and oftentimes, the user's preference from an online shopping activity has to be inferred based on the number of sessions taken to purchase the item, whether the item results in a purchase, as well as the number of views, additions to carts, and removals from carts. Implicit data takes such feedback to measure a user's preference for an item indirectly [3]. Collaborative filtering analyzes relationships between users and interdependencies among products to identify new-user item associations [3]. Given a user-item interaction matrix, matrix factorization is applied to estimate user factors and item factors. User and item factors are then multiplied to predict the score of an item for a user [4].



Typical matrix factorization algorithms learn user and item factors by minimizing squared errors between the predicted and actual ratings. Such optimization is not optimized to recommend items to users based on the top scoring items that the model predicts [5]. Instead, we propose a weighted approximate rank pairwise loss function (WARP) that optimizes ranking of items to recommend to users.

For user conversion, we are replicating the findings from Sohini’s paper [1].

**Data**

We are using public data from Kaggle and Groupby internal data.

Kaggle data

****

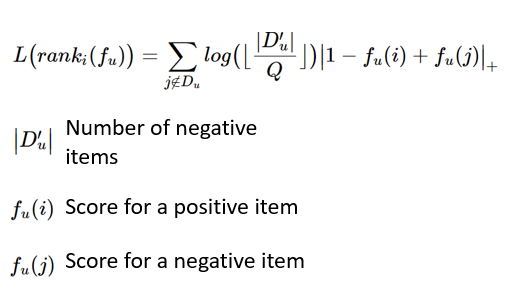
|  |  |
| --- | --- |
| Columns | Description |
| user\_id | Permanent user ID |
| user\_session | Temporary user's session ID. Same for each user's session. Is changed every time user comes back to online store from a long pause. |
| product\_id | ID of a product |
| category\_id | Product’s category ID |
| event\_type | View, cart, remove from cart, purchase |
| price | Price of product |

Groupby data

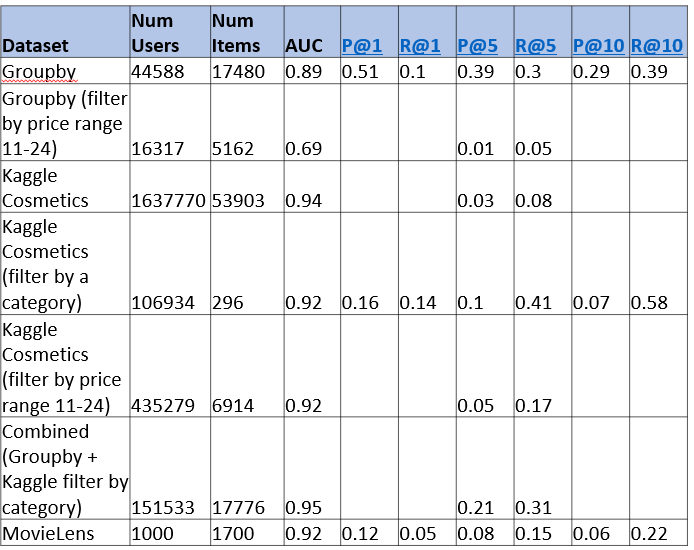
|  |  |
| --- | --- |
| Columns | Description |
| customerVisitorId | UUID for a given visitor that is unique to a device/browser |
| customerSessionId | UUID for a given session, usually expires after 30 minutes of inactivity |
| product.ID | Product’s category ID |
| product.Name | View, cart, remove from cart, purchase |
| product.Price | Price of product |
| product.Collection | Collection of the item |
| totals.totalOrderQuantity | Number of items purchased |

**Our Solution and Findings**

For the recommendation system, we are focusing on the customer journey interaction where we aggregate multiple session IDs for a customer. WARP was introduced to rank image intonation labels [6]. For each user, there is a pair of positive and negative items. We define a positive and negative item as a binary metric where 0 indicates that the user has not purchased the item and 1 where the user has purchased the item. The loss function only updates when the rank of a negative item exceeds that of a positive item. This approximates a form of active learning that yields a more informative gradient update, where the model samples the number of negative items Q times until the rank of a negative item exceeds that of a positive item. The loss function is described below [6].

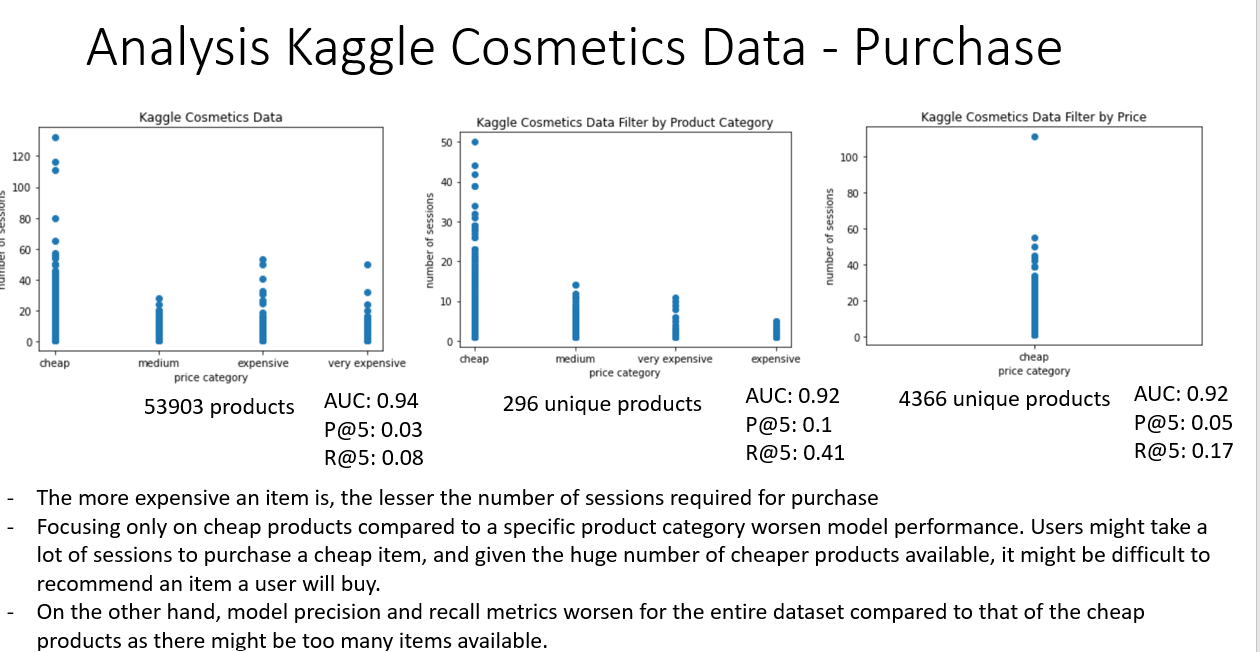


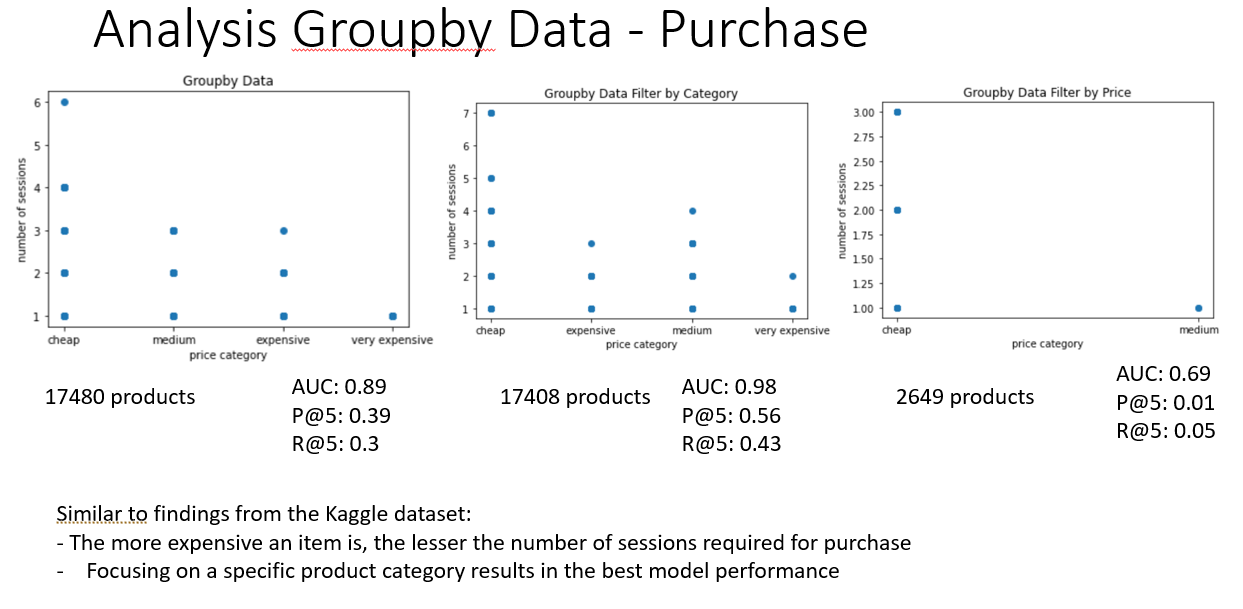
We randomly split the users into train and test sets, 80% and 20% respectively. Based on the random split, we might face a cold start problem where the users in the test set might not be in the train set, which is a situation we might face with new users. Since we are interested in recommending top ranked items, we will evaluate our model performance by comparing the items that are recommended against those that are purchased via precision@k and recall@k, where k represents the number of items recommended. We also use AUC to evaluate how well a randomly chosen positive item will rank higher than a negative item. We achieved the following results for the recommendation system:



* The performance on the Groupby full dataset is superior to the Kaggle Cosmetics dataset while the performance on Kaggle Cosmetics dataset matches that of MovieLens benchmark dataset.
* As the number of items recommended increases, precision decreases while recall increases. Recommending 5 items strikes a balance between precision and recall.

Further analysis on price category and number of sessions to purchase reveals the following insights:



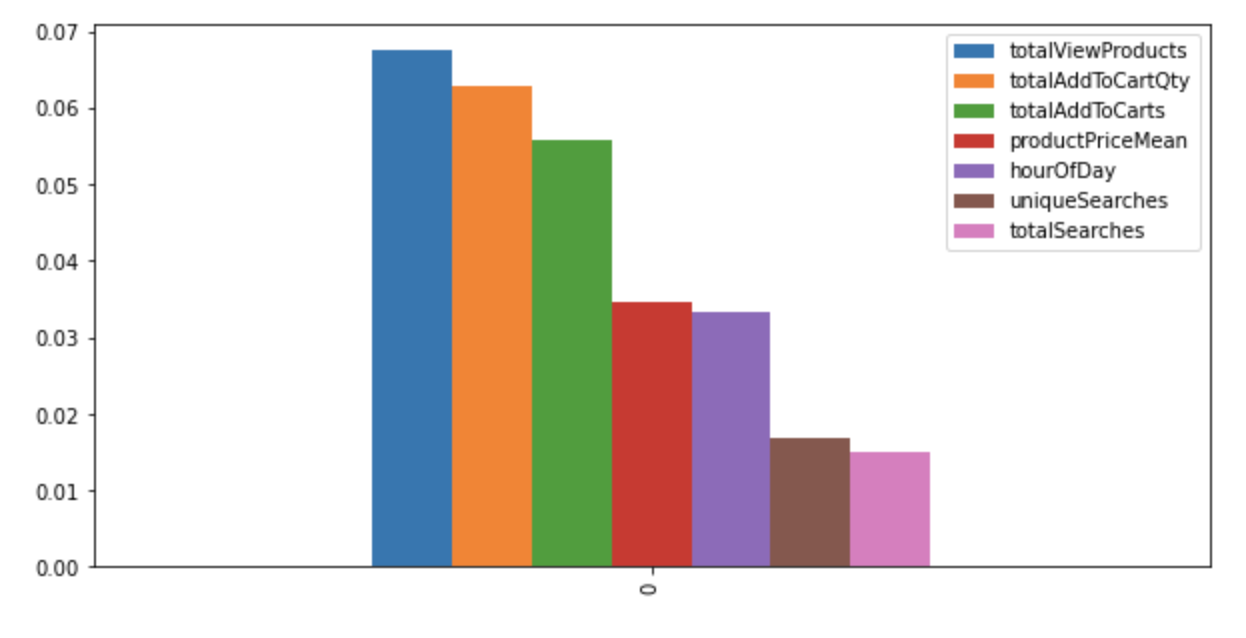


On the Kaggle dataset, using the whole dataset worsens model performance while on the Groupby dataset, the opposite finding is observed. This might be due to the number of sessions to purchase. In the groupby dataset, there are fewer sessions (1 day of data) and less overlap between the number of sessions to purchase across products of different prices. While on the Kaggle dataset, there are more sessions and items (5 months of data) and more overlap, which makes it harder for a user to purchase an item that is recommended.

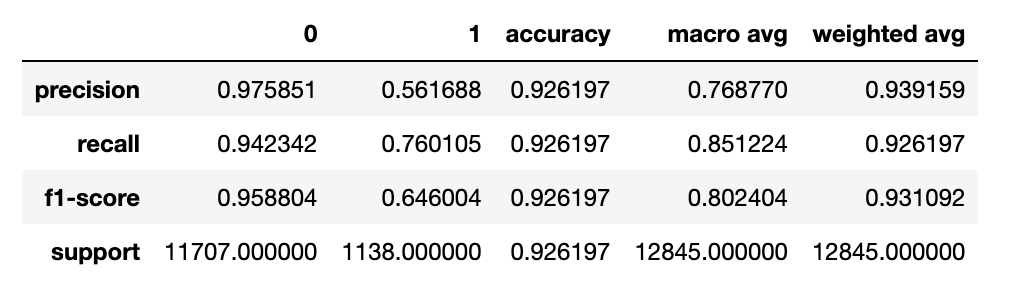
There is also a balance between recommending items to users that are brand specific and brand agnostic. Recommending items that are brand specific might increase the chance that a user will purchase the items while recommending items that are brand agnostic might increase brand awareness at the expense of purchase.

For user conversion, we worked on the Groupby data and looked at the session-level data and implemented 2 binary classification models. We selected the following features by looking at the correlations and also by calculating the permutation importance:

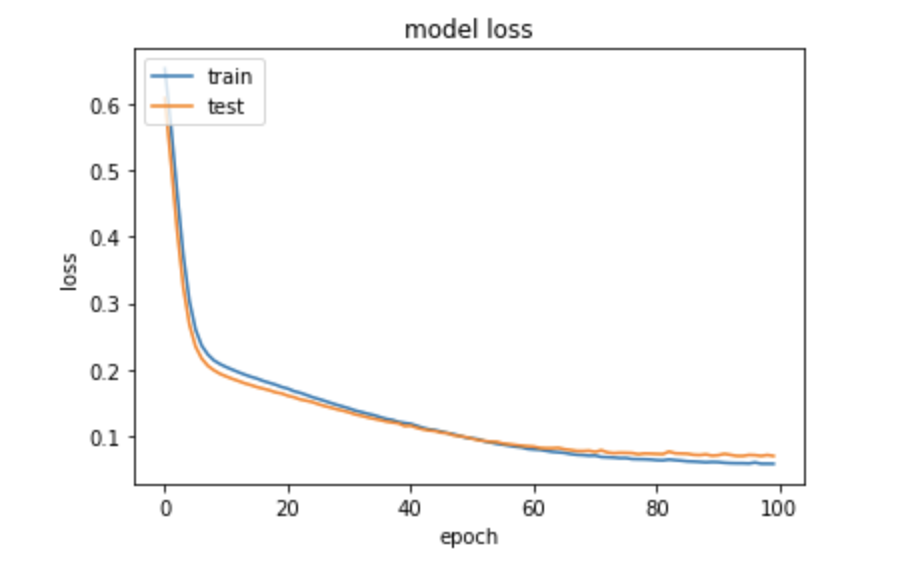
|  |  |
| --- | --- |
| **Importance scores** | **feature** |
| 0.067646 | Total View Products |
| 0.062860 | Total Add to Cart Quantity |
| 0.055818 | Total Add to Carts |
| 0.034540 | Viewed Mean Product Price |
| 0.033196 | Hour of Day |
| 0.016931 | Unique Searches |
| 0.014983 | Total Searches |



For Modeling, first, we use AutoML TPOT library to train a training data set of 7 most important features in the session-level data. We have found that the best model is from an ensemble stacking Classifier: GaussianNB(DecisionTreeClassifier(input\_matrix, criterion=gini, max\_depth=3, min\_samples\_leaf=16, min\_samples\_split=16)). The following are the metrics of this model. Note that accuracy in this case cannot be used as a metrics for determining whether this is a good model or not. This is because the label is highly imbalanced (8.65% conversion rate).



Second, we implemented an LSTM model by training on sequences of events (eg. view\_product, order, search, etc.) which we have extracted from the session-level data. We found that most sessions have less than 50 events. Therefore, we decided to train only on the data with 40 or less events. We have also compared the model performance with different probability thresholds. We found that a threshold 0.9 has the best F1 score. The best F1 score we found is ~0.884915. When compared with the model performance for cosmetic data [1], it seems that we are close to the optimal performance as we have relatively less data in GroupBy dataset (only 1 day of data). Also, the conversion rate in GroupBy data is about 8%, which is lower than the rate in the cosmetic data [1]. The following are the metrics after 100 epoch:



Accuracy: 0.982315

Precision: 0.928778

Recall: 0.845009

F1 score: 0.884915

**Next Steps**

For recommendation system:

* Analyze differences in model performance for different datasets with respect to price and number of sessions
* Evaluate model performance based on F-1 score
* Visualize model performance with respect to the number of latent features
* Recommend items for new and existing users, deploy model as API and serve using Flask on AWS

For user conversion:

* Deploy models using Flask, Elastic Beanstalk. Build UI for demonstrating modeling scoring with real time sequence of user click events.
* Explore if transfer learning will help the sequence model
* Explore if it is possible to identify customer churn

**Systems Design**

We are adapting to AWS services for training, delivering, scoring and storing the models and metrics data. For data preprocessing, feature engineering and training, we are using Sagemaker. We read data from S3 through MLFlow, preprocess the raw data base upon research findings, train and save the model and its metrics back to S3 through MLFlow. For model delivery, we are using AWS Elastic Beanstalk to host a Flask application, which retrieves and the model from S3 through an API call to MLFlow.



**Ethical Considerations**

Possible things to do for data collection and preparation tasks

* Collect buying history of users, calculate propensity score. For customers with lower scores we can email them incentives. For high score customers we try to retain them.
* Collect product search history of users. We can infer user’s favorite brands, prices, and categories.
* Segment users into different groups based upon categories they viewed and purchased. Collect user retention information.
* Cluster users into different types: old users, new visitors, churn users.
* Collect demographics data about the customers.

Considerations

When predicting customer conversion from viewing purchasing, we can see viewing statistics which tie to personally identifiable information (ie. user id). This will potentially have privacy concerns. Therefore, we will have to

1. Store such information in a secured manner.
2. Make sure customers are not able to access other customers’ viewing activities.
3. Notify users that we are collecting such information, and provide ways for users to opt out. We will need to consider the retention period of the data according to regional laws and comply with CCPA, GDPR regulations.

There are many ways to personalize the shopping experience, for example: Recommend products that users may like; change the user interface of the website making it more convenient for the users to get to the products which they are likely to navigate to.

1. When personalizing the experience for customers, we need to be careful to not be too aggressive in recommending products. We do not want to confine customers to certain products, limiting their choices.
2. We need to be transparent to the users what data we are collecting and what we use the data for.
3. We need to be careful when recommending products based upon demographics data.

**Timeplan**

|  |  |
| --- | --- |
| Week | Deliverables |
| 14 | * Finalize analysis of model performance * Deploy model as API and serve using Flask on AWS |
| 15 | * Prepare for project presentation day * Finalize demo * Finish technical report * Complete code on GitHub |

**References**

[1]. [2102.01625.pdf (arxiv.org)](https://arxiv.org/pdf/2102.01625.pdf)

[2]. <https://rejoiner.com/resources/amazon-recommendations-secret-selling-online/>

[3]. http://yifanhu.net/PUB/cf.pdf

[4].<https://towardsdatascience.com/factorization-machines-for-item-recommendation-with-implicit-feedback-data-5655a7c749db>

[5]. https://dl.acm.org/doi/pdf/10.1145/2507157.2507210

[6]. http://ethen8181.github.io/machine-learning/recsys/5\_warp.html#Reference