Build a Recommender for Instacart

Caixuan Sun

1. Introduction

How customers make their purchase decisions are quite different today than years ago. In today's digital world, every single customer can be provided with personalized service due to the increasing amount of information and data. Recommender systems are shaping our behavior everyday and nearly everywhere. When we are reading articles, we will be given recommendations to other articles we might like. We will even be given recommendations to products based on our reading history. When we are shopping online, we will be recommended with similar products, products often bought together, or products bought by other similar customers. Recommenders are one of the most successful machine learning technologies in business. Intelligent recommendation engines will increase businesses' ability to analyze client behavior and then deliver the right product or the right offer to the right customer at the right time which will probability enhance sales per transaction. McKinsey reports that 35% of Amazon's revenue is generated by its recommendation engine.

For the second capstone project, I would like to delve into this interesting field to build a recommender system for Instacart, the online grocery ordering and delivery company. Useful insights about customer's preference could be extracted based on

what they have consumed previously and then we can infer what products they might be interested in.

2. Data Wrangling and EDA

2.1 Data Description

The dataset used for this project is from open sourced dataset provided by Instacart[1] . It includes a set of six relational files describing customers' orders over time. Generally speaking, the dataset contains a sample of over 3 million orders from more than 200,000 users.

Specifically, three of the data files are information about aisles, departments and products. IDs and names of aisles, departments and products are provided. There are 134 aisles, 21 departments and 49,688 products. Which aisle and department each product belongs to are also given.

orders.csv contains 3.4 million rows for 206,209 users. Each row represents an unique order. For each order, the associated user_id, the order sequence number for this user, the day of the week and the hour of the day the order was placed, and days since the last order are provided.

order_products__prior.csv contains around 3.2 million orders which are all the orders prior to the users' most recent order. This is a item level dataset. It show us

[1] *The Instacart Online Grocery Shopping Dataset 2017", Accessed from https://www.instacart.com/datasets/grocery-shopping-2017 on 01/28/2019.*

what products were bought in each order, the order each product was added to cart, and if this product was reordered or not.

We have all the prior orders in the above dataset. While all the most recent orders for each user are split into training and test data. Test data is reserved by Instacart for competitions. Training set is in *order_products__train.csv* which contains 131,209 most recent orders. The other 75,000 most recent orders are in the hold out test data.

Dataset is clean. Missing values only exist in the column of days_since_prior. All the users' first order with Instacart are NAs. I filled those NAs with zero since they're the first orders, days since their prior orders are zero.

2.2 EDA

Let's start our EDA by exploring users' grocery shopping habits on Instacart. When do users buy groceries? During which day of week and what hour of day?

Figure 1. Distribution of Orders Over the Week

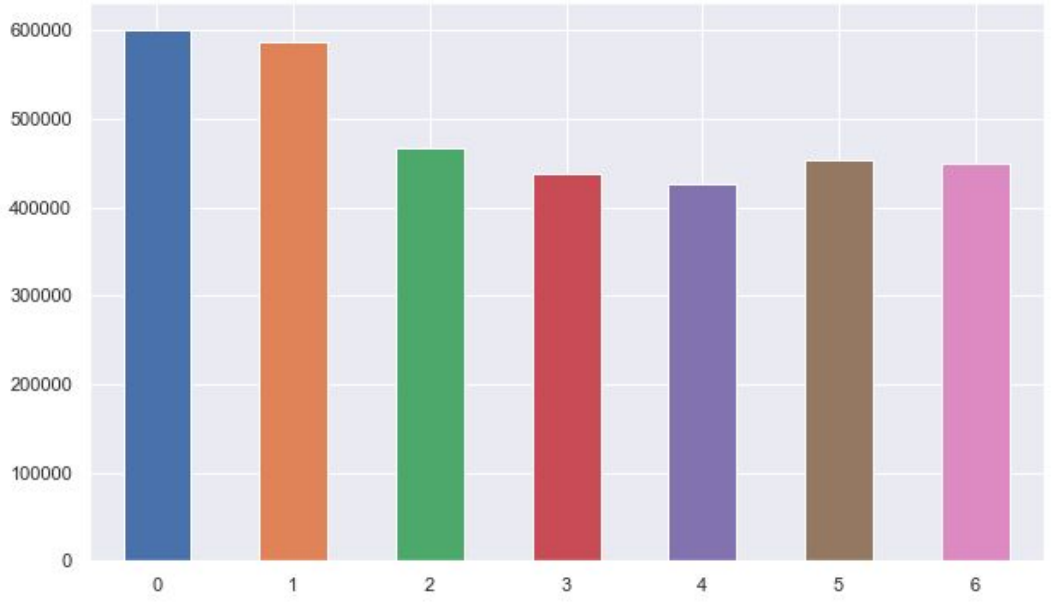
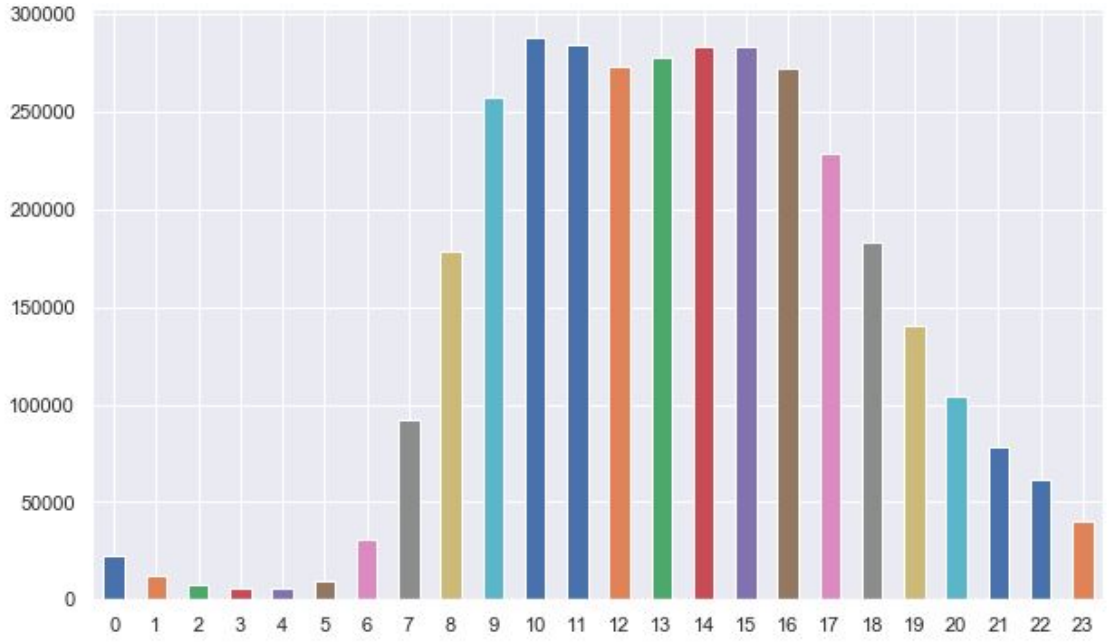


Figure 1 presents a bar plot for a column named order_dow which indicates what day of week the order was place on. But how the day of the week corresponds to the number (0-6) in this column is not provided. Based on the bar plot, my guess will be that 0 indicates Saturday, 1 indicates Sunday, 2 indicates Monday and so on. Since people are doing more grocery shopping during weekends, and less during weekdays. If this is the case, we can see that lowest order numbers happen on Wednesdays.

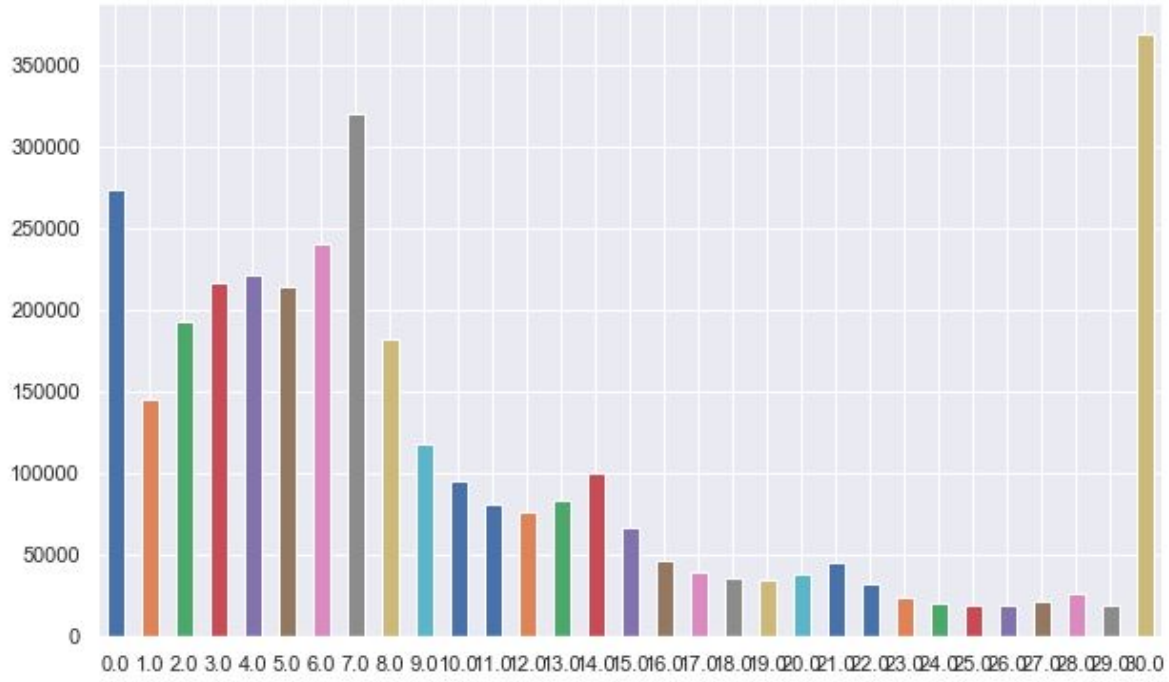
Figure 2. Order Distribution Over the Day



We can find from the figure above that majority of the orders are placed during daytime.

Next, let us see how ofter users will order again. By checking the summary statistics for column days_since_prior_order, we find that maximum is 30 days. 50% of all the orders are either first orders or placed within 7 days since their prior order.

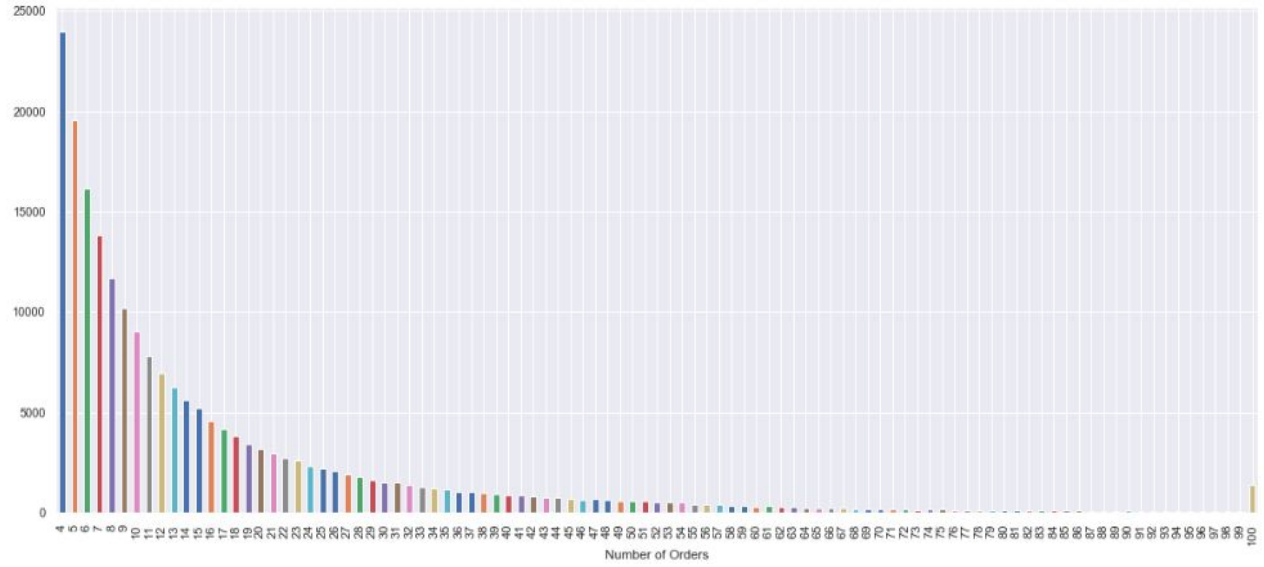
Figure 3. Distribution of Days Since Prior Order



The majority of orders are placed within 8 days since their prior orders. People tends to do grocery shopping at least once per week. The distribution peaks at 7 days. One thing needs to be noticed is that according to the plot a large number of orders are placed exactly 30 days after their prior order which may not make sense. One reason might be that it's censored, i.e. all values greater than 30 days are converted to 30 days.

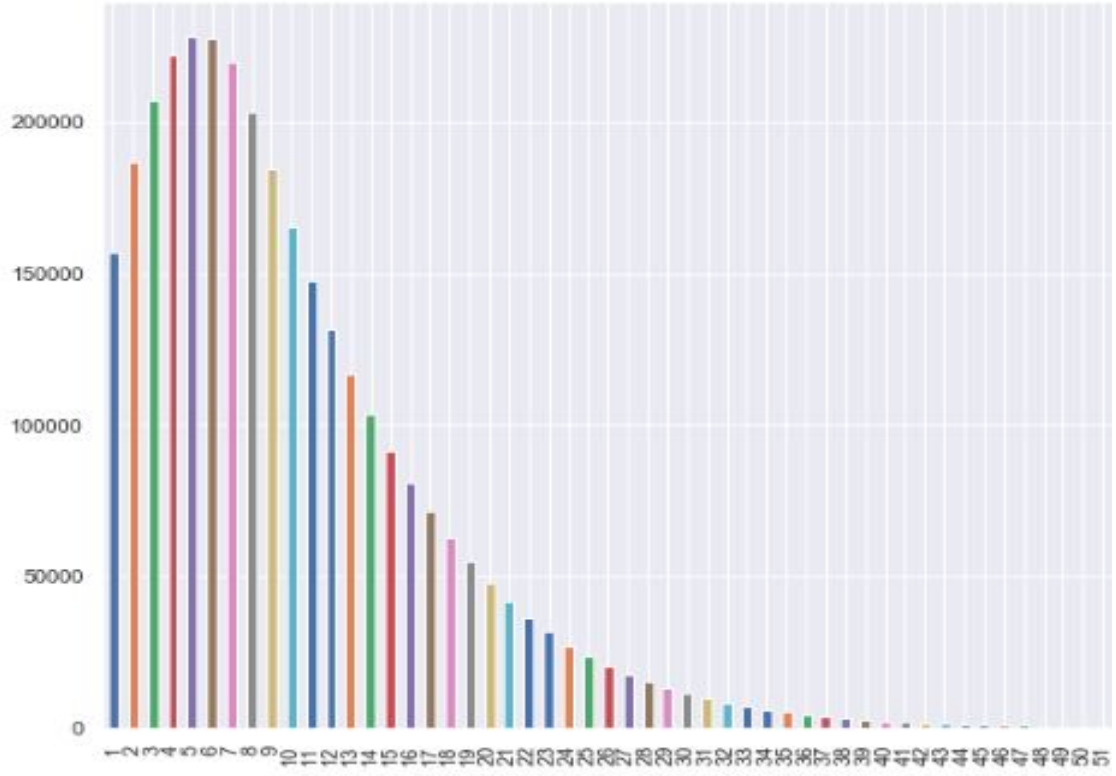
How many orders have each user placed on Instacart? From the following figure, we can see that every user in the provided dataset have ordered more than 4 times with Instacart. While the number of orders placed is capped at 100 which is not normal thus I suspect that data for this feature is also censored.

Figure 4. Number of Orders Distribution



Now let’s move to products. We’re wondering about how many products were purchased in each order.

Figure 5. Products per Order Distribution



From Figure 5, we can see that most frequently, users would buy 5 products in each order. For visibility reason, I cut off the figure at 51, while the maximum is 145.

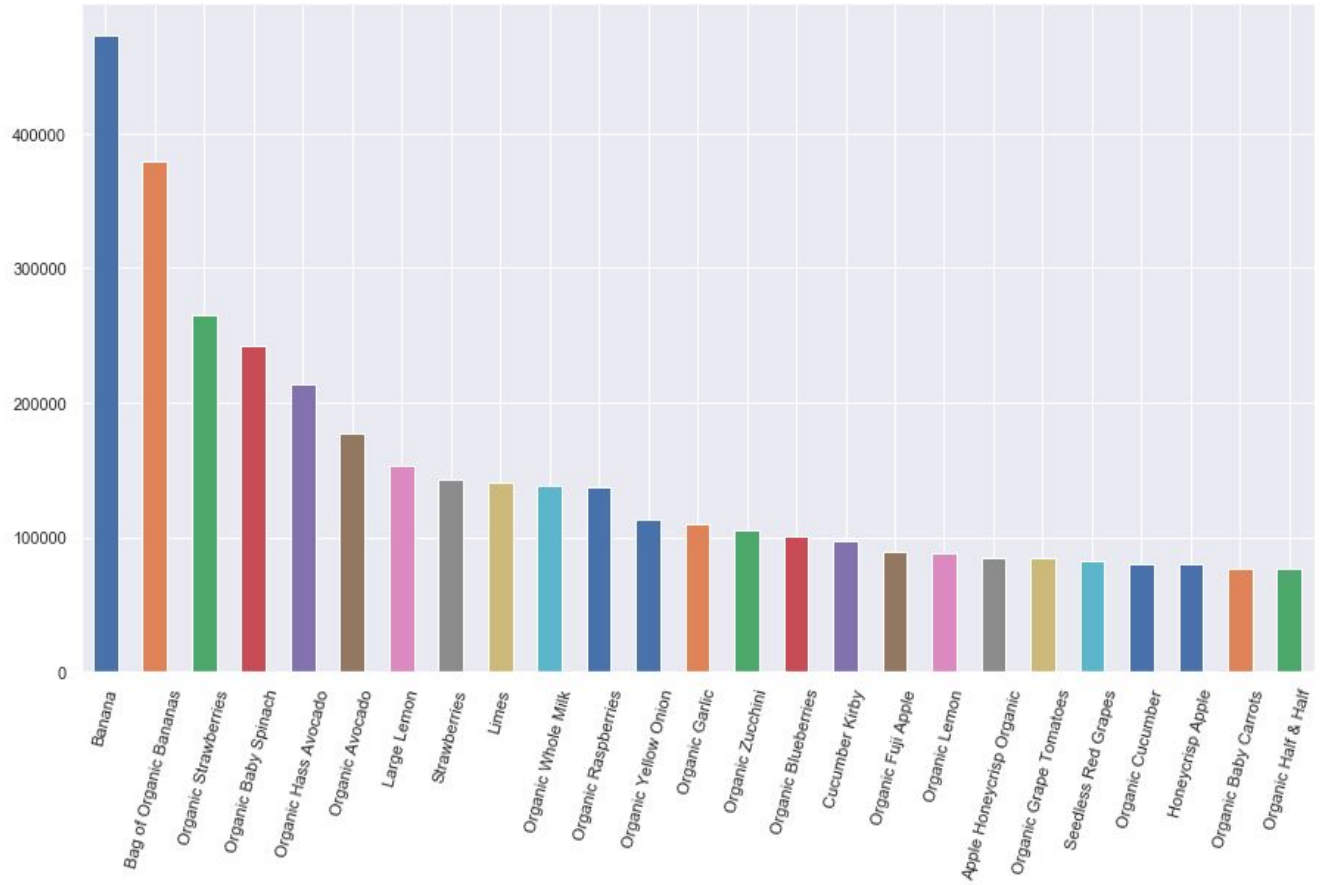
The dataset also provides us with a column named add_to_cart_order which tells us in what order each product was added to the cart in each order. Let's see which products are added to cart first in each order. Here I present the top 25 products that were added to cart first in the order those products were bought.

Banana	110916
Bag of Organic Bananas	78988
Organic Whole Milk	30927
Organic Strawberries	27975
Organic Hass Avocado	24116
Organic Baby Spinach	23543
Organic Avocado	22398
Spring Water	16822
Strawberries	16366
Organic Raspberries	14393
Sparkling Water Grapefruit	13733
Organic Half & Half	12676
Large Lemon	12316
Soda	11770
Organic Reduced Fat Milk	9885
Limes	9719
Half & Half	9528
Hass Avocados	9500
Organic Reduced Fat 2% Milk	9338
Raspberries	8885
Organic Fuji Apple	8762
Organic Blueberries	8740
Apple Honeycrisp Organic	8730
Organic Yellow Onion	8548
Unsweetened Almondmilk	8535

In 110,916 orders, banana was first put into their cart. Banana and bag of organic bananas are the items users often put into cart first. Users are quite certain that they need bananas, thus the first thing to do is to add them to shopping cart.

Now we know which products were most often added to cart first. How about the most popular products? What are they for grocery shopping?

Figure 6. Best Sellers



Banana	472565
Bag of Organic Bananas	379450
Organic Strawberries	264683
Organic Baby Spinach	241921
Organic Hass Avocado	213584
Organic Avocado	176815
Large Lemon	152657
Strawberries	142951
Limes	140627
Organic Whole Milk	137905
Organic Raspberries	137057
Organic Yellow Onion	113426
Organic Garlic	109778
Organic Zucchini	104823
Organic Blueberries	100060
Cucumber Kirby	97315
Organic Fuji Apple	89632
Organic Lemon	87746
Apple Honeycrisp Organic	85020
Organic Grape Tomatoes	84255
Seedless Red Grapes	82689
Organic Cucumber	80392
Honeycrisp Apple	79769
Organic Baby Carrots	76896
Organic Half & Half	76360

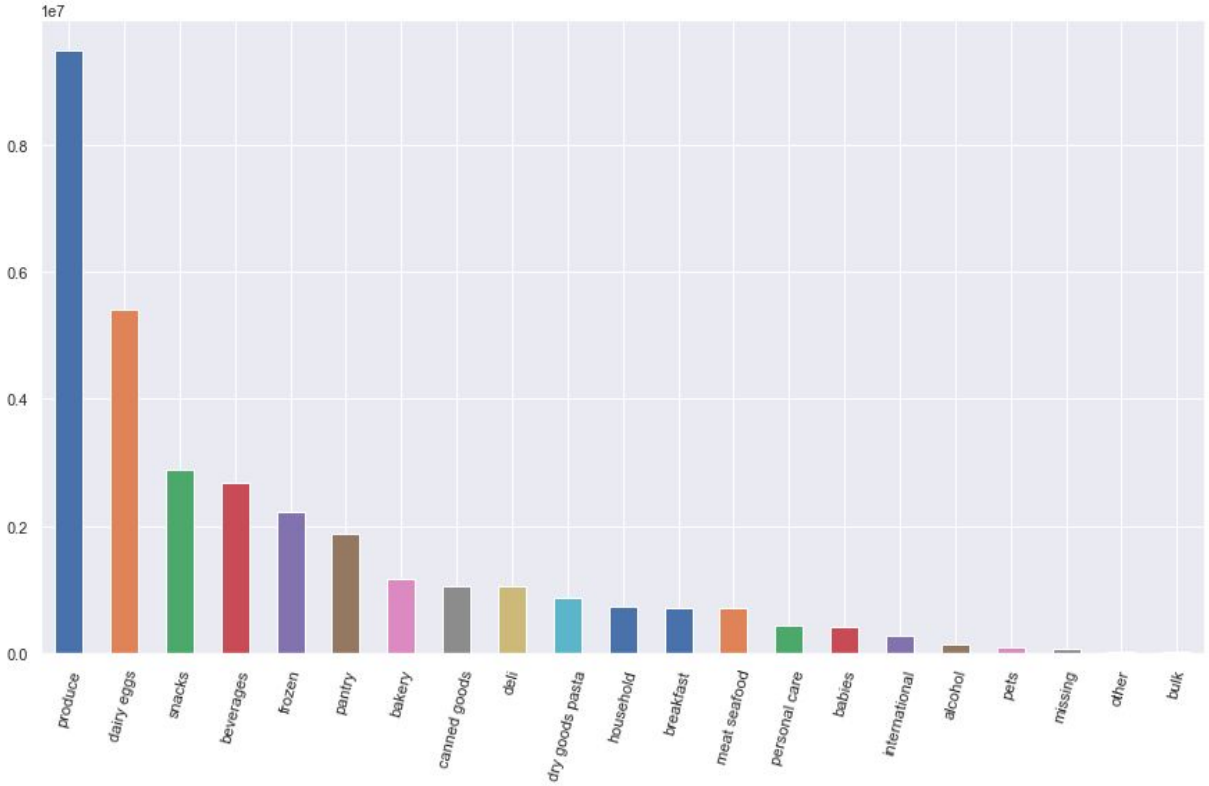
Most of the top sellers are organic fruit and vegetables. Users on Instacart are living a healthy life.

Figure 6. Best Sellers by Aisles

fresh fruits	3642188
fresh vegetables	3418021
packaged vegetables fruits	1765313
yogurt	1452343
packaged cheese	979763
milk	891015

As can be seen from the top sellers, the best sold products are from aisles of fresh fruits, fresh vegetables.

Figure 7. Best Sellers by Departments

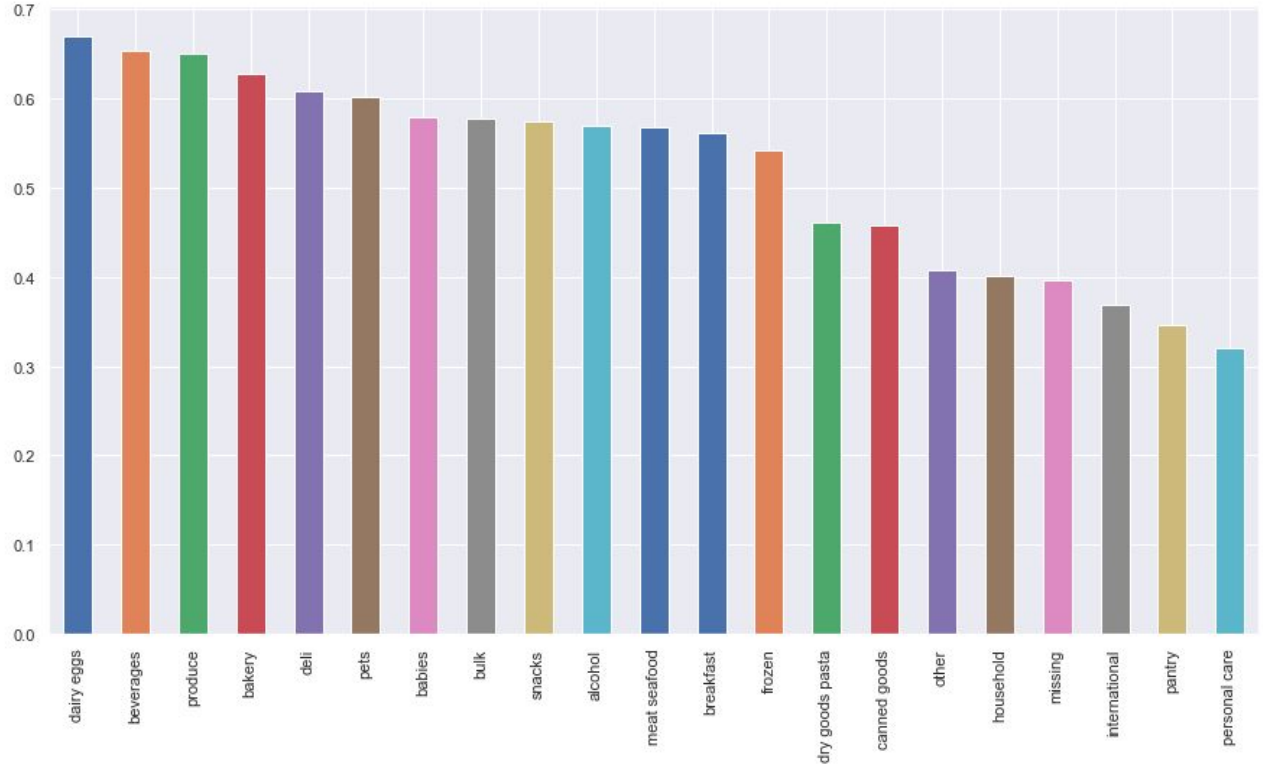


Produce is at the top. Bulk is at the bottom since people don't need to buy bulk items frequently as fresh fruit and vegetables.

I also checked the best sellers for reordered products, what aisles they belong to and what department they are in. The result is pretty similar.

Since we are given if the product in each order is reordered or not. We can delve into reorder realm a little more. Let's first see the reorder ratio by deparment. We will know products from which department were most got reordered.

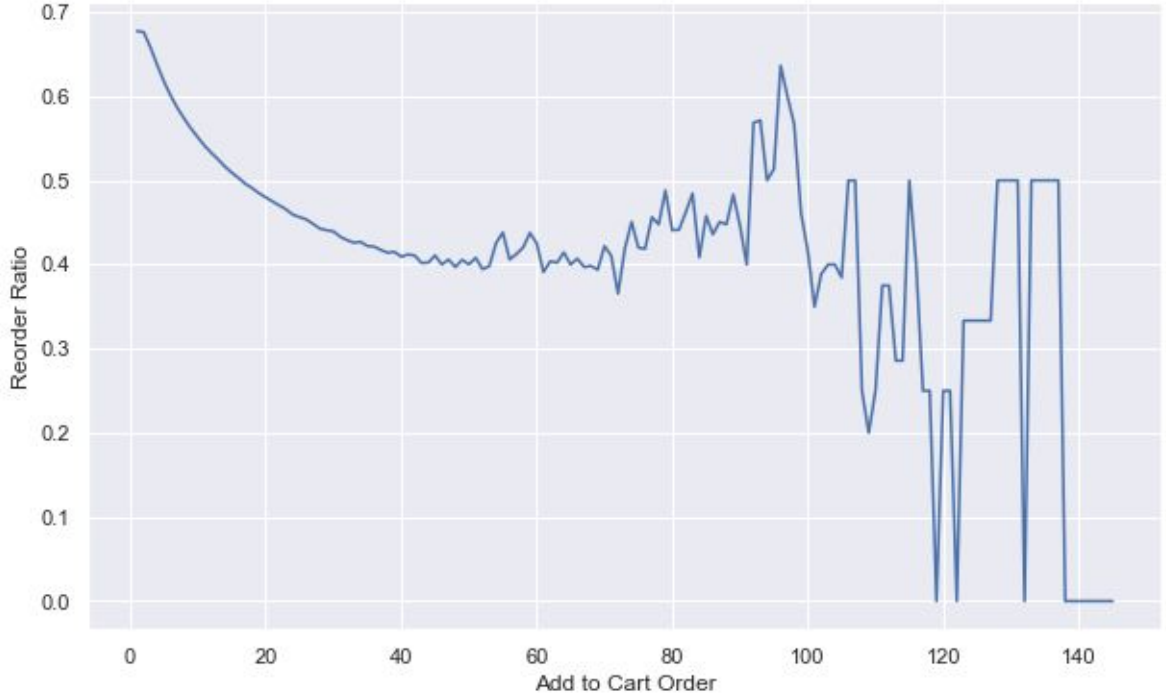
Figure 8. Reorder Ratio by Department



Dairy eggs is the department with highest reorder ratio. Users tend to reorder items from this department most frequently. By contrast, personal care gets the lowest reorder ratio.

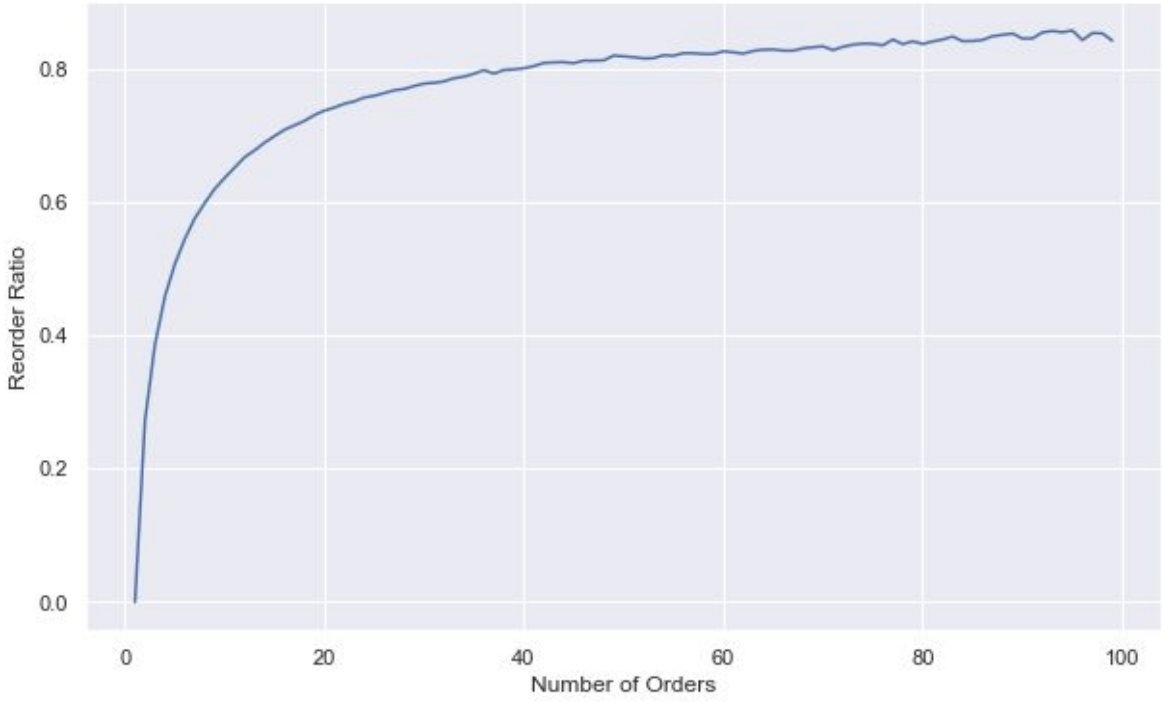
We have found that the most reorder products are quite similar to the best sellers. What other factors are associated with reorder ratio? Does reorder ratio relate to the order of that product added to cart?

Figure 9. Reorder Ratio VS Add_to_Cart_Order



We can see a clear declining in reorder ratio while add to cart order increases from 0 to 40. Users usually like to add things already in their mind to cart first, among which many will be items they need to replenish in a regular period of time. Products added to cart later might be some new product they bought for the first time. The reorder rate for new products will not be that high. After that users would like to explore new things to buy. After 40 sequences, the line is getting weird since there are only a few orders bought more than 60 items, result is becoming extreme with few data point.

Figure 10. Reorder Ratio VS Number of Orders



The line will start from (1, 0) since all items in the first order are considered as newly ordered rather than reordered. Reorder ratio rises steeply for the first ten orders, then keeps rising diminishingly. The positive relationship between these two features are reasonable as more orders placed, bigger probability of ordering same items as before. There is a ceiling for reorder ratio. No matter how many orders have been placed by the users before, users will still buy something new.

3. Build the Recommender

3.1 Introduction to Recommender Systems

The most common recommendation systems are content based and collaborative filtering recommender systems. Content-based filtering will make recommendations

based on users' preferences for items' features. It will recommend items which are similar with the ones being positively rated by the user. A major drawback of content-based filtering is that it is restricted to recommending items that are of the same type. To improve on this, we have the collaborative filtering which is to derive individual preferences based on community behavior. Collaborative filtering can be implemented using different models. I will use matrix factorization and neighborhood-based model. Let's first discuss our evaluation metric and the benchmark model.

3.2 Evaluation Metric

Evaluation metric I choose for the recommender of this case is recall@k. k is the number of products recommended. For our case, the feedback is implicit, there's no explicit rating scores. It is more appropriate to use classification accuracy metrics. Precision as we know represents the proportion of recommended items that actually bought by the users. It is not that appropriate since we do not know users' reaction to the recommended items. Besides, precision will give us the same result for two users who both bought 2 items from the recommendation list with 10 items, but one of them bought a total of 5 items while the other only bought a total of those 2 items. The performance of the recommender should not be the same which is what the precision score gives us. Recall is more appropriate which represents the proportion of actual bought items that are from the recommended list.

From the EDA part, we know that reorder ratio is quite high. Since recommender is not only about recommending items that users have already purchased before, but also about recommending items that users may be interested

in but are unaware of them. Therefore, I would like to build another evaluation metric, still using recall, but eliminating reordered items from test data to check the proportion of first time ordered products that are from the recommended products. To sum up, we have mean recall@k score with all products included and another mean recall@k score without those reordered products.

3.3 Benchmark: Non-Personalized Popularity Model

For the benchmark model, I choose the non-personalized popularity model. Given the number of products we want to recommend, k, we can simply get the top-k bought products. The benchmark model is non personalized since it is recommending the same item list to all the users. Here I present the performance of our benchmark model given k=[10, 20, 50] based on the two evaluation metrics describe above.

Table 1. Performance of Benchmark Model

	with reordered products	without reordered products
recall@10	0.070	0.027
recall@20	0.096	0.044
recall@50	0.154	0.080

We can see that recall score increases with the number of products recommended increases. It makes sense cause as we recommend more, more actually purchased items will fall into the recommended list.

The popularity model is not performing that well in recommending people to try new products they've never bought before. This is because as we know from the

EDA the top popular products are similar as those most reordered items. Thus when we get rid of the reordered items from the test data, the interaction of those two lists would shrink.

Let’s see what the user with id 100 actually bought in their most recent order and what the popularity model recommended to this user.

User_id 100 actually bought:

	product_id	product_name
21136	21137	Organic Strawberries
21615	21616	Organic Baby Arugula
24851	24852	Banana
26368	26369	Organic Roma Tomato
27343	27344	Uncured Genoa Salami
38546	38547	Bubblegum Flavor Natural Chewing Gum
38688	38689	Organic Reduced Fat Milk
48627	48628	Organic Whole Wheat Bread

User_id 100 got recommended:

	product_id	product_name
13175	13176	Bag of Organic Bananas
16796	16797	Strawberries
21136	21137	Organic Strawberries
21902	21903	Organic Baby Spinach
24851	24852	Banana
26208	26209	Limes
27844	27845	Organic Whole Milk
47208	47209	Organic Hass Avocado
47625	47626	Large Lemon
47765	47766	Organic Avocado

user 100 bought two of the products from the recommended list: organic strawberries and banana.

3.4 Matrix Factorization using ALS

In this section, I implemented matrix factorization using ALS algorithm. The algorithm is based on research by Yifan Hu et.al [2]. The basic idea of matrix factorization is to decompose the user-item interaction matrix into a product of two matrices with lower dimensions. One matrix can be regarded as the user matrix where rows represent users, and columns are latent factors. The other matrix is the item matrix where rows are latent factors and columns are items. Matrix factorization is a better solution for sparse data problem. I chose ALS algorithm to solve for the optimization of matrix factorization problem since it is less computationally intense than SVD.

3.4.1 Data Preprocessing

Since the test data is held out, I will use all the prior orders as my whole dataset then split it into training set with all the orders prior to the most recent orders, and test set with all the most recent orders. I will train the recommender using all the prior orders then test the performance of the recommender by comparing the most recent orders with the recommended list of products.

After splitting the dataset, I first converted the test set to a dataframe in the form of only two columns: first column is user_id, second column is a list of products purchased by each user in their most recent order which looks like this:

[2] Yifan Hu, Yehuda Koren, and Chris Volinsky, *Collaborative Filtering for Implicit Feedback Datasets*. http://yifanhu.net/PUB/cf.pdf

	user_id	products
0	1	[196, 46149, 39657, 38928, 25133, 10258, 35951...
1	2	[24852, 16589, 1559, 19156, 18523, 22825, 2741...
2	3	[39190, 18599, 23650, 21903, 47766, 24810]
3	4	[26576, 25623, 21573]
4	5	[27344, 24535, 43693, 40706, 16168, 21413, 139...

Next step is to transform the training set to the form required by the model. We need a sparse user-item matrix. The matrix represents the interactions between user and item in term of preferences. Each row represents a user, each column represents an item and the value in the matrix is the preferences. As we said before, we do not have explicit ratings, we only have implicit feedback. Thus I use the number of purchases to represent the preferences. After transforming the training set to a sparse user-item interaction matrix, we have to do one more step cause the input of fitting the model is not user-item matrix, but a matrix of confidences for the like items. You can learn more about this from the paper in footnote 2. To get the confidence matrix, we need a parameter called alpha and then multiply it with the user-item matrix. For the section below where we are fitting the default model, I chose a alpha=10. Later I will tune this parameter along with other parameters in the model.

3.4.2 Fitting the Defalut Model

Default parameters:

factors = 100: the number of latent factors.

regularization = 0.01: regularization parameter to prevent overfitting.

iterations = 15: the number of ALS iterations to use when fitting data.

The recall@10 for the default model is 0.021 when reordered items are included, and 0.044 when reordered items are excluded. The recall score is less than the popularity model with all products, but it's better than the popularity model without reordered products. The Collaborative Filtering model using matrix factorization is better at recommending new products to customers than the benchmark model.

3.4.3 Tuning the Model

Through the same process, I will build the training and validation set from the training set.

Parameters to be tuned and values to be searched:

alpha: for the confidence matrix [10, 15]

factors: [30, 40, 50]

regularization: [0.01, 0.1, 1.0]

iterations: [25, 50]

The best values are in red.

Hyperparameter tuning takes a lot of time, thus I only searched once and used the results.

3.4.4 Model Evaluation

Table 2. Evaluation of the Tuned Matrix Factorization Model

	with reordered products	without reordered products
recall@10	0.021	0.043
recall@20	0.035	0.072
recall@50	0.063	0.131

By comparison, all recall@k with reordered products are lower than the benchmark popularity model. While the ALS model is performing better after we get rid of all the reordered products. So it's doing a better job in recommending new stuff to consumers. I will put all the evaluation results for the three models together in the conclusion section for us to easily make comparisons.

3.5 Neighborhood-Based Model

There are basically three steps for neighborhood-base models. First is to make a choice for the entity to compare either users or items. Second is to choose a similarity measure to find nearest neighbors. Last is to define a strategy to make recommendations to the user out of his nearest neighbors. The central to most neighborhood-based models is similarity measure between users or items. In this project, I use the item-item similarity algorithm provided by Turi Create and choose two different similarity measures which are cosine and jaccard.

3.5.1 Data Preprocessing

First step of preprocessing data is to split the dataset into training and test set, same as matrix factorization model. Next is also to convert the test set to the dataframe with two columns. For the training data, the preprocessing is different. We do not need the user-item interaction matrix. Although we do need a dataframe showing us how many purchases each user have made for each products according to their prior order history. There are three columns, user_id, product_id and quantity indicating number of purchases. Then final step is to transform this dataframe to SFrame as required by Turi Create. Here is what the training data looks like:

user_id	product_id	quantity
1	196	9
1	10258	8
1	10326	1
1	12427	9
1	13032	2
1	13176	2
1	14084	1
1	17122	1
1	25133	7
1	26088	2

[12422758 rows x 3 columns]
Note: Only the head of the SFrame is printed.

With training data ready, we can now fit the model and evaluate the performance of the models.

3.5.2 Model Evaluation

Two similarity measures are adopted: cosine and Jaccard.

Table 3. Evaluation of the Neighborhood-Based Models

	Cosine		Jaccard	
	with reordered products	without reordered products	with reordered products	without reordered products
recall@10	0.018	0.039	0.018	0.037
recall@20	0.029	0.062	0.030	0.061
recall@50	0.05	0.105	0.053	0.110

Just compare between these two models, we can see that if we use the metric without reordered products, Cosine is performing better only when 10 products were recommended. Jaccard is doing better when we are recommending 20 or 50 products.

3.6 Conclusion

In this project, I build recommender systems for Instacart using two models: matrix factorization model and neighborhood-based model. If we include all the products in the evaluation step, we can see that the non-personalized popularity model is the best due to the high reorder ratio and the high similarity between reordered products and most popular products. We focus more on the evaluation metric excluding the reordered products since we want to see how our recommender systems are doing when recommending new products to consumers. As can be seen from the following table, both of the MF model and the neighborhood-based model are performing better than the benchmark model. Matrix factorization is the best. Matrix factorization is better at solving for the sparsity problem.The sparsity ratio of our user-item matrix is 99.88% which means 99.88% of entries in the matrix are zeroes.

Table 4. Summary of Evaluation for All the Models

	Benchmark		MF		Cosine		Jaccard	
	include	exclude	include	exclude	include	exclude	include	exclude
recall @10	0.070	0.027	0.021	0.043	0.018	0.039	0.018	0.037
recall @20	0.096	0.044	0.035	0.072	0.029	0.062	0.030	0.061
recall @50	0.154	0.080	0.063	0.131	0.05	0.105	0.053	0.110