MATS: Market Basket Analysis for Targeted Sales

Silas Swarnakanth Kati(5654085); Shashank Magdi(5735673); Seetharam Ramireddy(5733705);
Raj Vaibhav Gude(5748350); Navanshu Khare(5746283)

Github Link: SilasKati/Market-Basket-Analysis-for-Targeted-Sales (github.com)

1. Introduction

1.1 Problem Statement and Goal

The goal of our project is to predict which products will be in a user's next order. To achieve this we first try and explore the basic information about the dataset given. Then we try different Association analysis techniques on that data. Finally, we try different classification techniques for prediction.

1.2 Importance

The purpose behind this is to arm businesses with the tools they need to boost sales by better understanding how people buy things. In today's retail environment, gaining a competitive edge requires analyzing transactions that frequently occur together and using these important observed insights to construct a personalized and efficient target marketing approach.

1.3 Expected Results

We intend to find which association rule mining works better for this dataset and which classifier produces the best results in predicting the user's next order.

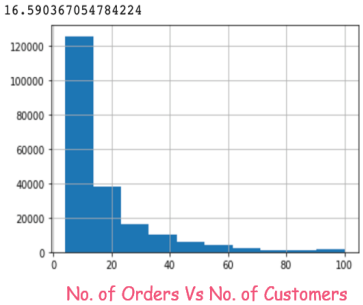
1.4 Data Description

The Instacart Online Grocery Shopping Dataset 2017 consists of a group of files that describe customers' orders over time. It comprises 6 CSV files containing information about 3.4 million grocery orders from more than 200,000 Instacart members.

Orders between 4 to 100 for each customer is described in the dataset, and to provide an avenue for prediction of reorders, the dataset has been devised in a way that the last order of users have been taken out and then divided into training and testing datasets. These can be used to run the classification algorithms(if any) and possibly predict the products that will be re-ordered for a given user id.

The remaining information regarding all the prior orders are present in a separate prior_orders file which we can use to run various algorithms to generate association rules and perform customer segmentation accordingly.

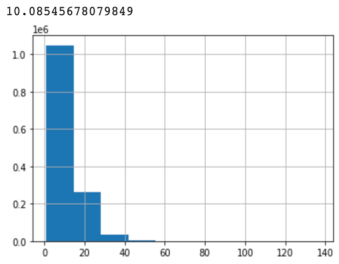
2. Exploratory Data Analysis:



No. of Orders Vs No. of Customers

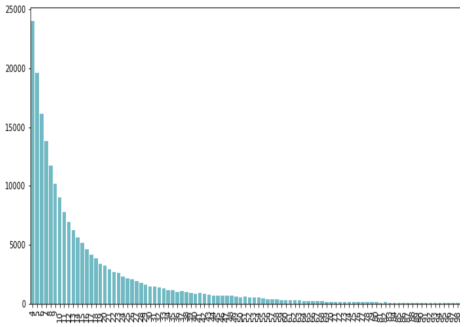
This is the histogram we get when we group orders by user_id. Analyzing the orders file, we can see that the average number of

orders per person remains at 16.5, while the claim that the maximum number of orders of 100 per customer is clearly validated here.

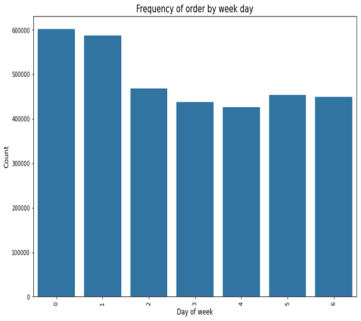


No. of Products in each order vs No. of Orders

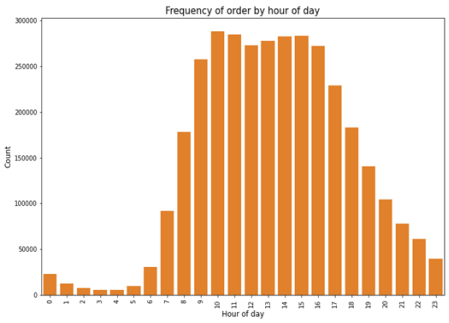
When the prior ordered products are grouped by the order id, we get this. It was important to understand the number of items within each order, the average number of items per order, was 10.8. The major key insight using grouping and defining a function to return unique values: we identified the total number of customers as 206,209 and we selected 75000 for the prediction case(to predict the next order) and the rest(131,209) assigned to train.


Maximum order number.

Now, by grouping the file with user id and order number, clearly, the customers with orders of 4 have the highest number of occurrences, while the maximum number of orders of a customer given is 100 as mentioned in the description.

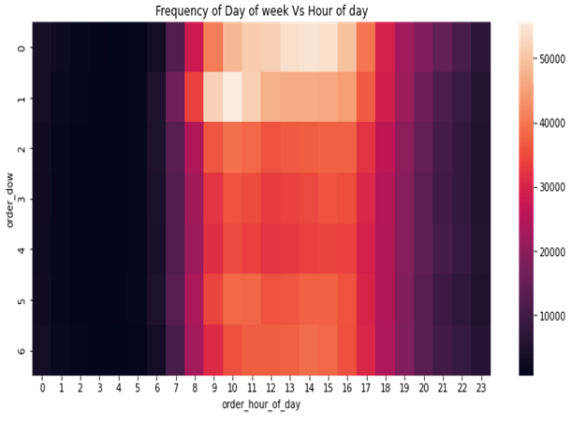


The trend depicting which days had the most orders was successfully understood: The highest orders grossed remain between Saturday(0) and Sunday(1), while the middle of the week (Wednesday).



It was not only important to analyze which days had the most frequent orders but also which time of the day and clearly the popularity among the orders lies during the daytime when the distribution is analyzed w.r.t the time of the day.

Further, when we merge both the days of the week and also the hour, from that distribution the heatmap



says that Saturday evenings and Sunday mornings offer the favorable times to place orders.

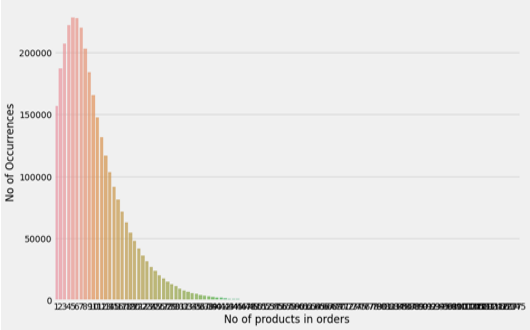


Fig: Count of items across orders vs their occurrences in the training dataset

Findings:

- 94% of all the orders made have less than 25 products in them. Meaning 94% of all the carts have less than 25 items.
- The distribution is a right-tailed distribution that starts rapidly declining after reaching the peak.
- The highest occurrence is 5 products per order in both datasets.

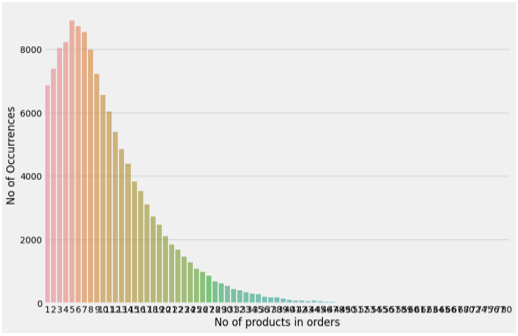
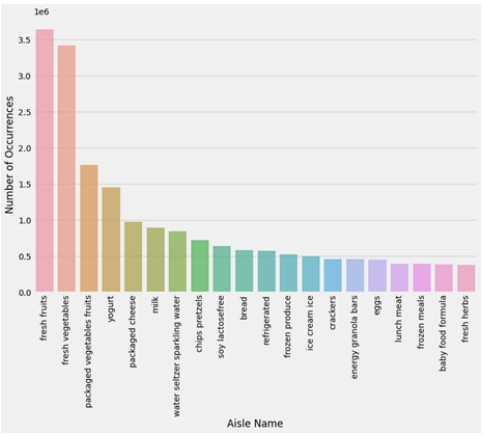


Fig: Count of items across orders vs their occurrences in the prior dataset



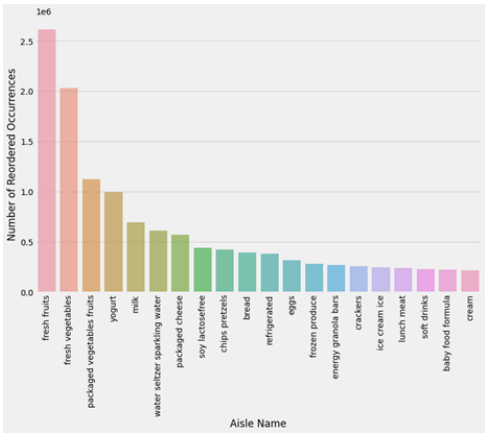
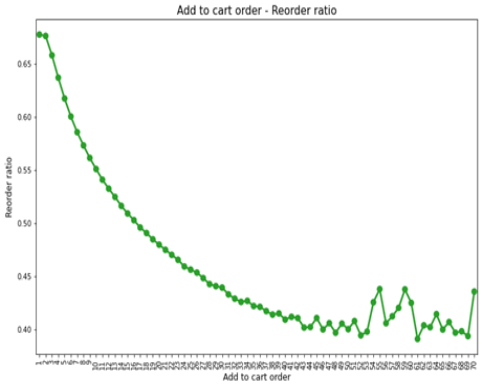


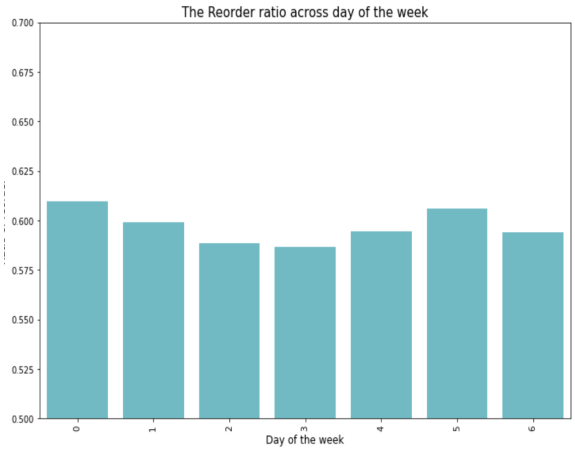
Fig: (a) The number of orders across each aisle. Fig: (b) Number of reorders across each aisle
Findings:

- 46% of all products sold belong to produce and dairy eggs departments (Distribution: Produce – 29% and Dairy Eggs – 16%).
- The aisles containing Fresh Fruits and Fresh Vegetables have the highest orders. The reorder ranks of the top most commonly purchased aisle products remain mostly constant. Meaning, that organic food aisles remain the leaders in orders and reorder.



Having a look at this distribution of departments, the largest department definitely produces, while dairy eggs and snacks also remain heavily popular. Reorders are an important aspect to understand trends and help in rule mining. When the reorder ratio was computed for each department, the dairy eggs had the highest reorder ratio among all the items, while produce too had a pretty high return, but personal care products showcased a poor reorder ratio.

Many times, we add products to the cart but do not end up buying them, therefore understanding when the product was added to the cart is vital. This is why we felt that determining the relationship between adding to cart and reorder ratio was important after finding out which items had a high reorder ratio. This graph clearly shows the customer is more likely to reorder the item if he/she placed that item in the cart originally when contrasted with the ones that were added later on.



While we saw the frequency of orders across each day of the week, here we show the reorder ratio. Sundays again remain very popular to place the reorders, while Thursdays also offered good returns.

3. Association Rules Mining

After understanding the data and observing the trends and patterns from EDA, the findings and insights were used for association rule mining across various granularities and filtered data subsets.

3.1. Apriori Algorithm

Initially, the algorithm has been implemented on the entire dataset of 32 million records containing all the 3.4 million transactions. The minimum support threshold was first set to 10%

where 1772 frequent items were generated with a total of 1560 itemset pairs. However, none of the generated rules had the lift greater than 1, showing that the pairs are negatively correlated. Then, alternate threshold values were tested for, varying between 0.1% to 10% with the number of frequent items and itempairs generated increasing for a decreasing minsup threshold value.

3.1.1. Setting the thresholds

A minsup of 1% was chosen after many iterations and this generated 10,906 frequent items with which 48,751 rules were generated. 208 of these rules had their lift greater than 1, but however most of their confidence was very low. So another check was added to fetch only the rules with minimum confidence threshold of 20% of which only 47 rules satisfied both the conditions.

3.1.2. Observations from the rules

One observation was that a majority of these rules contained the same products in the pair but of different flavours. For instance, 'Strawberry Yogurt' was often bought with 'Blueberry Yogurt' with a lift value of 5.3. Similarly grapefruit ginger sparkling yerba mate was often bought with Cranberry Pomegranate sparkling Yerba Mate'. This pattern has been observed across many categories like cottage cheese, cat food, baby food, essence water, smoothie, spoons and forks, and chocolate bars etc. Also, the specifications of ingredients/mixtures in many product pairs are also the same – for instance, 0% yogurt of flavour A is bought with 0% yogurt of flavour B. Similarly whole milk products are bought with whole milk products of differing flavours or types. There is very little cross purchase.

Another observation from an uber level is that a majority of the rules are amongst yogurts when compared to all other product categories. This goes to say that people often tend to buy more than one yogurt and definitely not the same flavour. They always seem to try multiple flavours. This observation can be very useful in store shelf planning and website marketing recommendations.

3.1.3. Association mining on subsets of the data by day of the week

After finding the above mentioned insights, the algorithm was again implemented on weekend and weekday transactions. This was done because, from the EDA we see that a majority of the transactions occur over the weekend. The goal of this approach was to observe the unique patterns or rules which appear on either parts of the week, if any. The data was split as follows – Monday through Friday was considered as the weekday and Saturday – Sunday as the weekend. After being able to separate the data, the algorithm was run on both the datasets and the outputs were analysed.

3.1.4. Weekday rule mining observations

A total of 41211 rules were mined with a minimum support threshold of 1%, of which only 242 rules qualified as interesting with respect to the lift value to be greater than 1. The threshold of 1% was chosen to keep it consistent across the analysis. Of these 242, a total of 50 associations were identified to be unique and appear only on the weekdays and not on the weekends. This includes spoons and forks, ales and ciders, marshmallows and candy bars.

3.1.5. Weekend Rule mining observations

Similarly for the weekends, a total of 56993 rules were mined with a minimum support threshold of 1% of which only 187 rules qualified as interesting with respect to the lift value. Of these total 187 rules, 19 associations were observed to be uniquely present only for the weekends and not the weekdays. These include Chicken sausage burritos and Turkey sausage burritos, Fiesta salad with BBQ chopped salad and Quinoa salads, Chicken dinner with Vegetable Turkey dinner.

3.1.6. Association mining on subsets of the data by hour of the day

Likewise the algorithm was also run on a partition of the dataset by hour of the day. This was done as the EDA shows that a larger number of orders are placed forenoon for all days. The dataset was again divided as before noon and fore noon by partitioning before and after 12pm. The algorithm was run on both the categories and the results were as follows –

3.1.7. Forenoon Rule mining observations

For forenoon, a total of 46,693 rules were mined of which 14 rules were identified to be unique as they appear only in the afternoon orders and that cross the threshold value of lift >1 proving to be interesting.

3.1.8. Association Mining to observe lifts across aisles

After observing and noting all these findings, association rules were also mined across aisles taking the aisle as the granularity to observe the correlation between aisles. The data was transformed from the product level to aisle level and then the algorithm was applied. The idea was to check if aisle A lifts aisle B. This step however hasn't been very fruitful as none of the rules crossed the lift value of 1. However, by observing confidence, we could see that certain aisles often appeared together like fresh fruits and fresh vegetables, seafood and fresh fruits.

3.1.9. Overall Conclusion

One common observation made across all these runs of the algorithm on different subsets of the data was that, the algorithm was quick to generate the frequent items but took a lot of time to

generate pairs, likely because the apriori scans the entire database to generate each pair and this iteration costs a considerable amount of time.

itemA	itemB	freqAB	supportAB	freqA	supportA	freqB	supportB	nfidenceA	nfidenceB	lift
Organic Strawberry Chia Lowfat 2% Cottage Cheese	Organic Cottage Cheese Blueberry Acai Chia	306	0.010155	1163	0.038595	839	0.027843	0.263113	0.36472	9.449868
Grain Free Chicken Formula Cat Food	Grain Free Turkey Formula Cat Food	318	0.010553	1809	0.060033	879	0.02917	0.175788	0.361775	6.026229
Organic Fruit Yogurt Smoothie Mixed Berry	Apple Blueberry Fruit Yogurt Smoothie	349	0.011582	1518	0.050376	1249	0.041449	0.229908	0.279424	5.546732
Nonfat Strawberry With Fruit On The Bottom Greek Yog	0% Greek, Blueberry on the Bottom Yogurt	409	0.013573	1666	0.055288	1391	0.046162	0.245498	0.294033	5.31823

Fig. The top 4 rules generated by running the algorithm on all the records

3.2. FP Growth Algorithm

Frequency Pattern Mining or the FP Growth algorithm is another one famous for generating association between multiple products purchased by the various customers. This algorithm's results are a set of rules with different parameters(based on our requirement), which the target organizations can use to make future decisions.

3.2.1 Why FP growth when we already have Apriori?

This algorithm was chosen as a measure of comparison against the Apriori algorithm and to understand which one is more viable to use under their respective circumstances. The biggest advantage of FP growth is that this algorithm needs to scan the database only twice the entire duration and also that too without any candidate generation while the Apriori scans the transactions at every iteration and generates several candidate item sets.

3.2.2 Results

The size of our dataset is massive as already mentioned with a size of over 3.4 million grocery orders accumulated from more than 200,000 Instacart members. We have tried implementing the FP growth algorithm on the entire dataset. We tried running the algorithm on Jupyter notebooks using the local systems, then on Google Colab(although it has heavy resource constraints with the 12.7GB RAM allotment option), then on Google Colab Pro(with a 25 GB RAM allowance) and also on the various powerful UMN CSE desktops but soon realized this was not a feasible task because of the following snippet :

```
[ ] trans_encoder = TransactionEncoder() # Instanciate the encoder
    trans_encoder_matrix = trans_encoder.fit(all_transactions).transform(all_transactions)
    trans_encoder_matrix = pd.DataFrame(trans_encoder_matrix, columns=trans_encoder.columns_)

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-13-74cc51da232b> in <module>
      1 trans_encoder = TransactionEncoder() # Instanciate the encoder
----> 2 trans_encoder_matrix = trans_encoder.fit(all_transactions).transform(all_transactions)
      3 trans_encoder_matrix = pd.DataFrame(trans_encoder_matrix, columns=trans_encoder.columns_)

~/.local/lib/python3.8/site-packages/mlxtend/preprocessing/transactionencoder.py in transform(self, X, sparse)
    123                                dtype=bool)
    124         else:
--> 125             array = np.zeros((len(X), len(self.columns_)), dtype=bool)
    126             for row_idx, transaction in enumerate(X):
    127                 for item in transaction:

MemoryError: Unable to allocate 149. GiB for an array with shape (3214874, 49677) and data type bool

SEARCH STACK OVERFLOW
```

To summarize, for us to successfully run the algorithm on the entire dataset(after we troubleshoot multiple errors too) it requires a humongous 149 GB of RAM. We soon realized an adoption of another strategy to test our hypothesis was required due to such unrealistic resource constraints.

```
trans_encoder_matrix.head()
```

	#2 Coffee Filters	#2 Cone White Coffee Filters	#2 Mechanical Pencils	#4 Natural Brown Coffee Filters	& Go! Hazelnut Spread + Pretzel Sticks	(70% Juice!) Mountain Raspberry Juice Squeeze	+Energy Black Cherry Vegetable & Fruit Juice	.5\" Waterproof Tape	0 Calorie Acai Raspberry Water Beverage	0 Calorie Fuji Apple Pear Water Beverage	...	with Twist Ties Sandwich & Storage Bags	with Xylitol 18 Sticks Sugar Free Gum	with Xylitol Island Berry Lime 18 Sticks Sugar Free Gum	with Xylitol Minty Sweet Twist 18 Sticks Sugar Free Gum	with Xylitol Original Flavor 18 Sticks Sugar Free Gum	Un C Fl F
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	

5 rows x 45139 columns

Our approach and why this was causing such a huge memory issue: The structure of the dataset required to run the This above code snippet depicts how the data was meant to be transformed for us to work with the FP growth algorithm on the entire dataset.

3.2.3 Alternative pathway

We soon realized, for us to compare and understand the performance of both the Apriori and FP-growth algorithms, we must obtain results on a similar sized dataset to avoid skew in results, which is why we concluded our findings through Apriori on the dataset and documented the results, after which we decided to run both Apriori and F-growth on a portion of the dataset consisting of comparatively fewer number of transactions, but still high enough to understand the trends and comparisons between these algorithms. This was on the Training_dataset CSV file from the same Kaggle Instacart Dataset.

	A	B	C	D	E	F	G	H	I
1	ntecedent	onsequent	cedent sup	equent sup	support	confidence	lift	leverage	conviction
2	frozenset({'	frozenset({'	0.055583	0.11798	0.018444	0.331825	2.81256	0.011886	1.320044
3	frozenset({'	frozenset({'	0.11798	0.055583	0.018444	0.156331	2.81256	0.011886	1.119416
4	frozenset({'	frozenset({'	0.055583	0.083028	0.011729	0.211024	2.541609	0.007114	1.162231
5	frozenset({'	frozenset({'	0.083028	0.055583	0.011729	0.14127	2.541609	0.007114	1.099784
6	frozenset({'	frozenset({'	0.11798	0.042268	0.013566	0.114987	2.7204	0.008579	1.082167
7	frozenset({'	frozenset({'	0.042268	0.11798	0.013566	0.320952	2.7204	0.008579	1.298907
8	frozenset({'	frozenset({'	0.083028	0.042268	0.012728	0.153295	3.62671	0.009218	1.131128
9	frozenset({'	frozenset({'	0.042268	0.083028	0.012728	0.301118	3.62671	0.009218	1.312056
10	frozenset({'	frozenset({'	0.11798	0.083028	0.023428	0.198579	2.391714	0.013633	1.144183
11	frozenset({'	frozenset({'	0.083028	0.11798	0.023428	0.282174	2.391714	0.013633	1.228738
12	frozenset({'	frozenset({'	0.142719	0.083028	0.016569	0.116095	1.398269	0.004719	1.037411
13	frozenset({'	frozenset({'	0.083028	0.142719	0.016569	0.199559	1.398269	0.004719	1.071012
14	frozenset({'	frozenset({'	0.049494	0.142719	0.014847	0.299969	2.101819	0.007783	1.224633
15	frozenset({'	frozenset({'	0.142719	0.049494	0.014847	0.104026	2.101819	0.007783	1.060864
16	frozenset({'	frozenset({'	0.142719	0.062	0.016447	0.115241	1.858714	0.007598	1.060175
17	frozenset({'	frozenset({'	0.062	0.142719	0.016447	0.265274	1.858714	0.007598	1.166803
18	frozenset({'	frozenset({'	0.074568	0.142719	0.015243	0.204415	1.432294	0.004601	1.077549
19	frozenset({'	frozenset({'	0.142719	0.074568	0.015243	0.106803	1.432294	0.004601	1.03609
20	frozenset({'	frozenset({'	0.074568	0.083028	0.012492	0.167518	2.017617	0.0063	1.101492
21	frozenset({'	frozenset({'	0.083028	0.074568	0.012492	0.15045	2.017617	0.0063	1.08932
22	frozenset({'	frozenset({'	0.074568	0.11798	0.017042	0.228536	1.937082	0.008244	1.143308
23	frozenset({'	frozenset({'	0.11798	0.074568	0.017042	0.144444	1.937082	0.008244	1.081674
24	frozenset({'	frozenset({'	0.04598	0.142719	0.010144	0.22062	1.545836	0.003582	1.099953
25	frozenset({'	frozenset({'	0.142719	0.04598	0.010144	0.071078	1.545836	0.003582	1.027018
26	frozenset({'	frozenset({'	0.04598	0.062	0.012156	0.264379	4.264159	0.009305	1.275113
27	frozenset({'	frozenset({'	0.062	0.04598	0.012156	0.196066	4.264159	0.009305	1.18669
28	frozenset({'	frozenset({'	0.056467	0.074568	0.010685	0.189229	2.537673	0.006475	1.141423
29	frozenset({'	frozenset({'	0.074568	0.056467	0.010685	0.143295	2.537673	0.006475	1.101351
30	frozenset({'	frozenset({'	0.056467	0.142719	0.016889	0.299096	2.095698	0.00883	1.223107
31	frozenset({'	frozenset({'	0.142719	0.056467	0.016889	0.118338	2.095698	0.00883	1.070175
32	frozenset({'	frozenset({'	0.056467	0.062	0.010281	0.182076	2.936692	0.00678	1.146805
33	frozenset({'	frozenset({'	0.062	0.056467	0.010281	0.165827	2.936692	0.00678	1.131099

Results of FP growth on Training dataset (With a Minsup threshold of 1%) : By setting a minimum support threshold of 1%, we obtained a total of 33 Association rules with not a lot of actionable insights that can be derived from these and also too many rules in general. Also to point out that when we tested it with a minimum support threshold of 0.1%, a huge number of rules were obtained, which is why we changed it to 1%.

1	ntecedent	onsequent	cedent sup	equent sup	support	confidence	lift	leverage	conviction
2	frozenset({'	frozenset({'	0.055583	0.11798	0.018444	0.331825	2.81256	0.011886	1.320044
3	frozenset({'	frozenset({'	0.055583	0.083028	0.011729	0.211024	2.541609	0.007114	1.162231
4	frozenset({'	frozenset({'	0.042268	0.11798	0.013566	0.320952	2.7204	0.008579	1.298907
5	frozenset({'	frozenset({'	0.042268	0.083028	0.012728	0.301118	3.62671	0.009218	1.312056
6	frozenset({'	frozenset({'	0.083028	0.11798	0.023428	0.282174	2.391714	0.013633	1.228738
7	frozenset({'	frozenset({'	0.049494	0.142719	0.014847	0.299969	2.101819	0.007783	1.224633
8	frozenset({'	frozenset({'	0.062	0.142719	0.016447	0.265274	1.858714	0.007598	1.166803
9	frozenset({'	frozenset({'	0.074568	0.142719	0.015243	0.204415	1.432294	0.004601	1.077549
10	frozenset({'	frozenset({'	0.074568	0.11798	0.017042	0.228536	1.937082	0.008244	1.143308
11	frozenset({'	frozenset({'	0.04598	0.142719	0.010144	0.22062	1.545836	0.003582	1.099953
12	frozenset({'	frozenset({'	0.04598	0.062	0.012156	0.264379	4.264159	0.009305	1.275113
13	frozenset({'	frozenset({'	0.056467	0.142719	0.016889	0.299096	2.095698	0.00883	1.223107

Next, we set a minimum confidence threshold as 20% to filter out the refined rules:

```
apripri_rule = compute_association_rule(apriori_matrix, metric="confidence", min_thresh=0.2)
apripri_rule.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Organic Baby Spinach)	(Bag of Organic Bananas)	0.075232	0.117912	0.015623	0.207664	1.761175	0.006752	1.113275
1	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.066010	0.117912	0.019065	0.288816	2.449414	0.011281	1.240308
2	(Organic Raspberries)	(Bag of Organic Bananas)	0.042166	0.117912	0.012363	0.293191	2.486523	0.007391	1.247987
3	(Organic Strawberries)	(Bag of Organic Bananas)	0.082374	0.117912	0.019184	0.232893	1.975136	0.009471	1.149888
4	(Cucumber Kirby)	(Banana)	0.030143	0.147169	0.010058	0.333677	2.267308	0.005622	1.279906

For example, here we can infer that, using the .head function, retrieve the first few rows to visualize the different rules and metric for those rules. This is the one for Apriori.

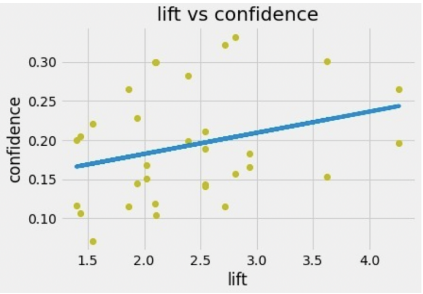
```
fp_growth_rule = compute_association_rule(fpgrowth_matrix, metric="confidence", min_thresh=0.2)
fp_growth_rule.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
0	(Organic Baby Spinach)	(Banana)	0.075232	0.147169	0.016027	0.213029	1.447518	0.00
1	(Organic Baby Spinach)	(Bag of Organic Bananas)	0.075232	0.117912	0.015623	0.207664	1.761175	0.00
2	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.066010	0.117912	0.019065	0.288816	2.449414	0.01
3	(Organic Raspberries)	(Bag of Organic Bananas)	0.042166	0.117912	0.012363	0.293191	2.486523	0.00
4	(Organic Raspberries)	(Organic Strawberries)	0.042166	0.082374	0.010435	0.247479	3.004339	0.00

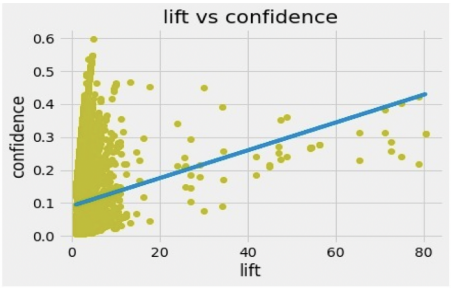
This case is for the preview of a few rules obtained using the FP-growth algorithm.

A lift greater than 1 proposes that the presence of antecedent increases the chances that the consequent will occur in the transaction. Also lift lower than 1 means purchasing the antecedent reduces the chances of purchasing the consequent in the same transaction.

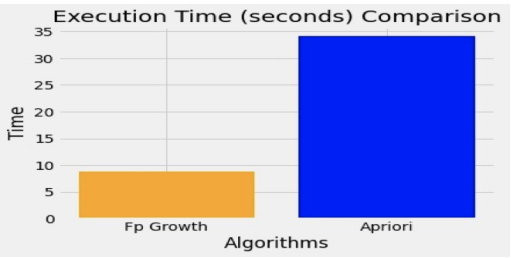
For Minsup = 1%



For Minsup = 0.1%



These following metrics relationship graphs visualizes how lift vs confidence varies w.r.t minimum support percent's of 1% and 0.1%. Clearly on the right we can see that a lot of items possess high confidence and low lift from which we can infer that we know these items are bought together frequently but we don't know if one item is causing the other item to be bought. We can make use of a few arbitrary points in the first graph whether both the confidence and lift is high and try selling the items together and then evaluate if the decision has paid off.



The following visual shows us that the FP growth's performance in terms of execution time is way better than that of Apriori although FP growth is more expensive to build and consumes more memory as we have seen in our case.

3.2.4 Other alternative(s)

1) We have discovered that the FP-growth algorithm can utilise the Apache Spark Machine Learning Library(MLib) in tandem with the Scala API on Databricks.. Databricks offers a distributed platform to quickly generate association rules on such large real life datasets as this.
2) Instead of using the Databricks platform we can set up Spark on our environment we are working on and try . Apache Spark is an open Source unified analytics engine and has based its emphasis on speed and concepts of distributed systems. Because of the size of the data at hand Spark would have been the best alternative for faster data analysis.
3) There are other algorithms we can try such as the **ECLAT** which also has lesser execution time than the Apriori but uses a vertical Data storage format against the Horizontal one which both Apriori and FP-growth use, so some data preparation will be needed.

Final Note: Although FP growth takes less time to execute and has no candidate generation, FP growth is cumbersome and harder to build than the Apriori and also the algorithm might not fit in the shared memory when the database we are using is large.

4. Customer Segmentation using Clustering

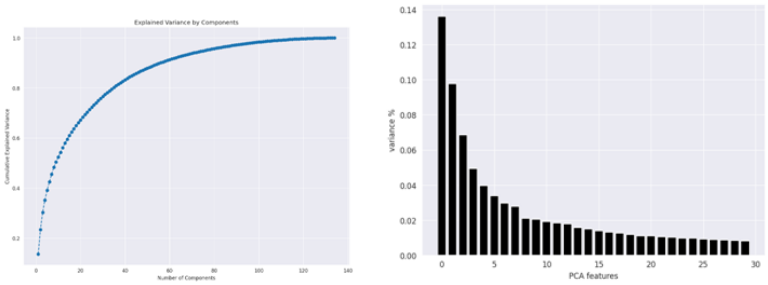
Clustering is the grouping of a particular set of objects based on their characteristics, aggregating them according to their similarities. This methodology partitions the data implementing a specific join algorithm, most suitable for the desired information analysis.

4.1 K-means clustering

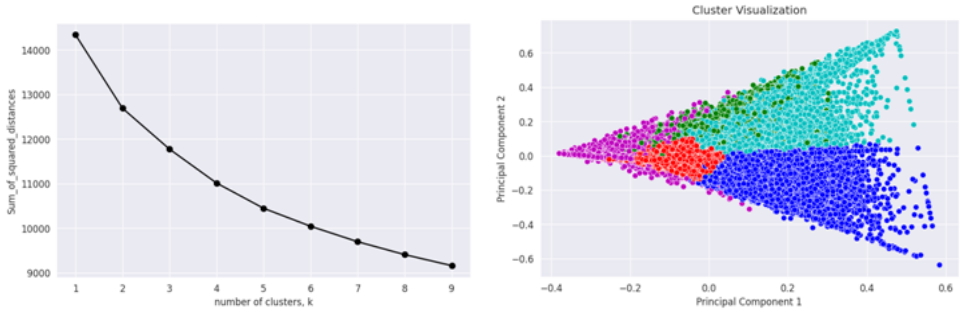
4.1.1 Clustering users based on aisles

Approach: We perform K means clustering for clustering between data points We took user_id and aisles from the Instakart data and converted it into a contingency table where contingency. We have 205773 rows and 134 columns .

Since we have 134 columns in the dataset and Clustering doesn't perform well with High dimensionality data, we are using PCA to reduce the dimensionality of the data. PCA is a method used to reduce the number of variables in your data by extracting an important one from a large We plotted a No of components vs. Cumulative Explained Variance graph.



It can be interpreted from this graph that How many components of Data explain how much variance of the data. Where 30 components itself explains more than 75% of the variability.
Another reason to choose to perform PCA is because Clustering on the large dataset works with only cosine similarity and PCA uses Cosine similarity, to be more specific Eigenvalues and EigenVectors to reduce the dimensionality of the data. Row 0 is explaining around 14% of the variability of the data and the second row is explaining around 6% of variance of data. Once We have reduced the dimensionality, Next step is to determine No. of clusters for K means clustering. Generally best approach is to use Elbow method, which uses the sum of square distances from each point to its assigned center and run multiple iteration to Plot the curve according to the number of clusters k and .The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

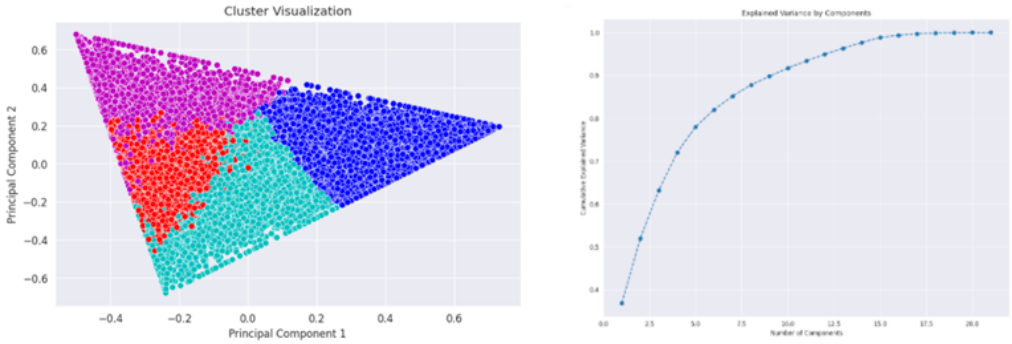


After determining No. of clusters to be 5 from the Plot. Clustering algorithm is implemented and 5 clusters with centroids with 30 dimensions are generated. Graph has been drawn from feature 1 and features 2 in 2 dimensional spaces to show the cluster location. After the graph has been plotted we can see that there are 5 different clusters where some clusters are overlapping suggesting we can reduce the value of K. The points represent the user_id and each color represents the part of a cluster user_id is, For instance cluster_1 is represented by red color and so user 123 lies in cluster_1. Now it's time to look for meaningful insights from the clustering. Therefore we assigned user_id with its respective cluster point.
Observation : It has been observed that for cluster 1, Fresh vegetables is the most desirable product followed by fresh fruits. And in Cluster 4 , users like dairy products most. Observation deduced from clusters can be used for designing specific marketing campaigns for specific clusters or this can be enhanced by merging association analysis to form recommendation engines.

	fresh fruits	fresh vegetables	packaged vegetables fruits	yogurt	packaged cheese	milk	water seltzer sparkling water	chips pretzels
0	23.766015	40.486209	13.117776	6.711818	5.457100	4.391670	3.307547	2.761865
1	10.008169	2.830990	3.674280	6.603774	2.620367	3.932953	66.412532	3.916936
2	40.587399	16.242973	13.260757	10.586787	5.505520	6.490945	3.946979	3.378641
3	20.626118	15.689237	12.534093	14.919470	10.070035	8.722170	8.377277	9.061601
4	43.069151	12.729290	16.978433	2.928351	6.332550	6.570474	6.906777	4.484974

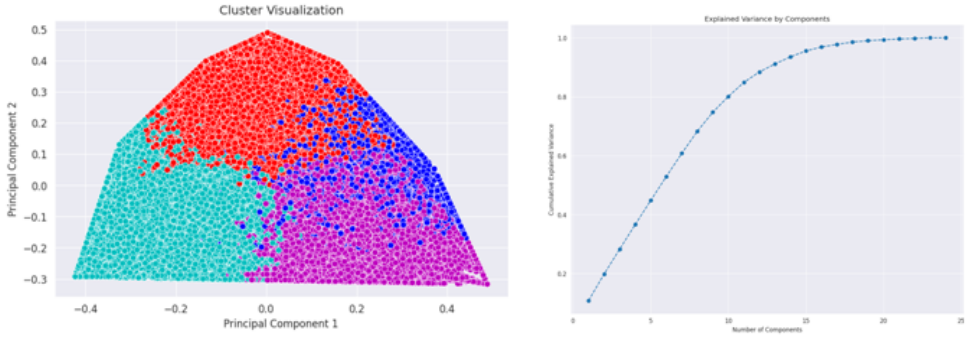
4.1.2 Clustering users based on department

Similar approach has been used to cluster users based on departments from which they purchase products. Here we have 21 departments which have been divided into 4 clusters. From the result it is clear that users from cluster 0 shop around 84% times from produce department and cluster 1 users buy most beverages.



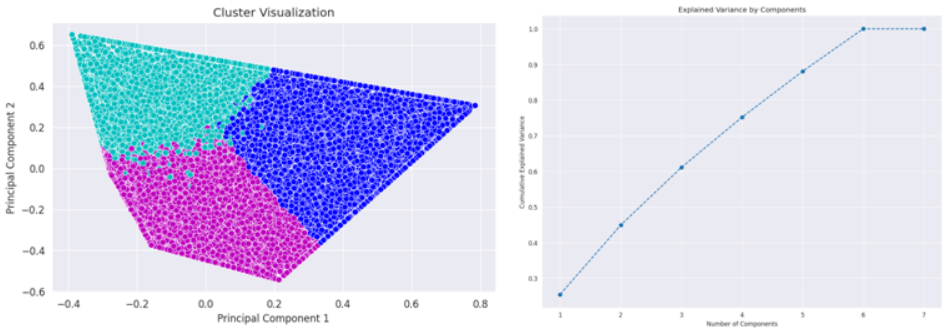
4.1.3 Clustering users based on hour of the day

For 3rd clustering , users based on the hour of the day they have purchased products have been chosen. 24 columns are generated in the contingency table and are divided in 4 clusters. From PCA it is clear that 12 PCA are explaining 85% of the variance in the dataset. From the result it is clear that users from cluster 0mostly shop around 11 AM whereas cluster 4 users mostly shop between 12 PM to 3 PM.



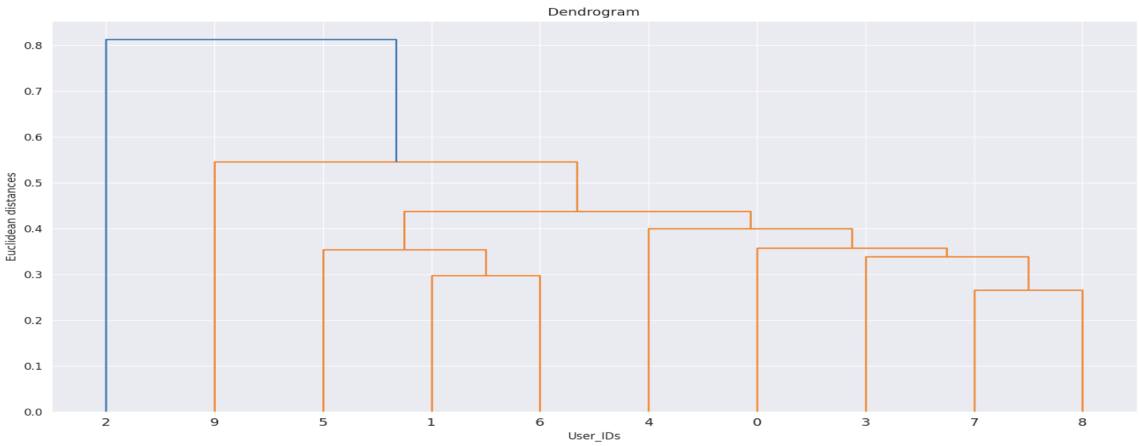
4.1.4 Clustering users based on day of the week

Lastly, Users are clustered based on the Day of the week they has purchased the product. Contingency table has 8 columns and 205773 rows. PCA is not necessary here as dimensionality is low, but PCA is still used to point out an interesting note that out of 7 PCA features, 6 features are explaining 100% of the variance and the 7th feature is not contributing in variance. From the result it is clear that Users from cluster 1 purchase most on Saturday and users from cluster 3 on Sunday. Weekend is favorable for purchasing products.

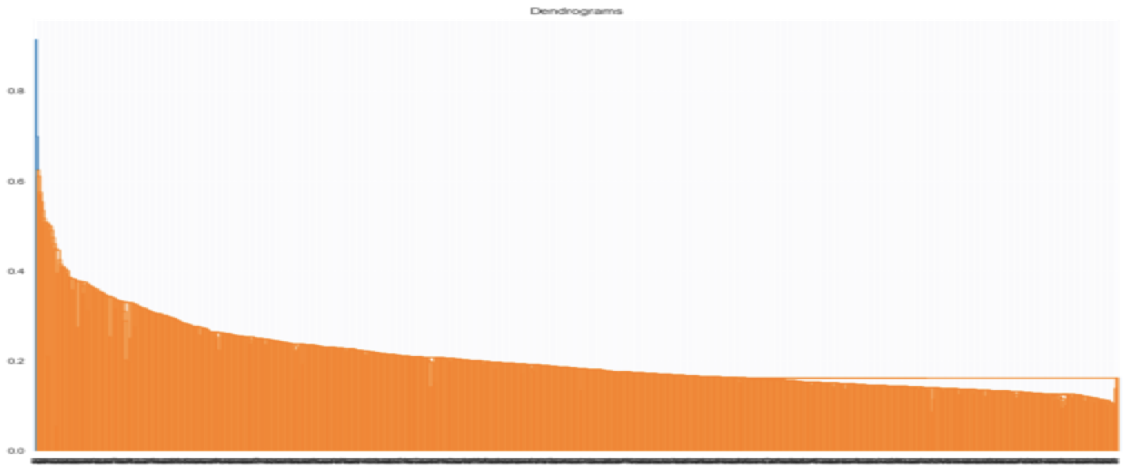


4.2 Hierarchical clustering

For hierarchical clustering, We are clustering users based on Department, and after applying PCA with 12 features to the dataset. Users are plotted in 12 Dimensional space and are clustered together based on Euclidean distance, Closest points will be clustered first and farthest points will be Clustered in the last. A Dendrogram is generated based on clustered users.



This diagram shows the dendrogram for 10 users using the MIN algorithm.



This diagram shows the dendrogram for 1000 users using the MIN algorithm. Deriving insights from is difficult, which is why the partitional clustering algorithms would seem to be a good fit.

4.3 Challenges faced using Clustering algorithms

1. Choosing k value. We used the elbow method and kneed library to give us the best k value.
2. The dimensionality issue. Used PCA to reduce the dimensions to a favourable number.
3. The clusters that we got were of points too densely packed so the silhouette score was very low and was not an efficient way of measuring the cluster validity.
4. We also tried achieving the same results by performing the hierarchical clustering. The single link algorithm worked but because the points were so close to each other, at each level of the dendrogram, only one point was added to the cluster. That meant that if we were looking for 4 cluster groups in hierarchical clustering, cluster 1 would have 1 point, cluster 2 would have 1 point, cluster 3 would have 1 point and then cluster 4 would have all the remaining points.
5. Correlation metric also did not work well either as there was only one large cluster and the others had only 1 point in them.
6. Complete link and average link methods did not run at all as they were requiring huge memory.

4.4 Conclusions

Cosine Similarity is Used to resolve the curse of dimensionality in a given problem .Clustering. Results can be used for designing marketing campaigns or design recommendation Engines. Products can be estimated and stored based on order habits of users.

5. User Product Recommendation

5.1 Modelling Strategy

With the dataset present at hand, we can have two approaches in preparing the data for our models.

Training Data: The features are built using prior_orders_data. Then, for each user, we use the Order information from train_orders_data to label the dependent variable, i.e. reordered. We then predict the product reorder likelihood and select the most likely products with a high reorder probability.

Test Data: We generate it from orders.csv with eval == ‘test’.

5.2 Feature Engineering

5.2.1 Generating Product features

1. product_reorder_rate : How frequently the product was reordered regardless of the user preference?

2. average_pos_incart : Average position of product in the cart?
3. fp_reduced_feat_1 : column 1 from NMF output (Non-Negative Matrix Factorization, to reduce sparsity)
4. p_reduced_feat_2 : column 2 from NMF output
5. p_reduced_feat_3 : column 3 from NMF output
6. aisle_reorder_rate : How frequently a product is reordered from the aisle to which this product belongs?
7. department_reorder_rate : How frequently a product is reordered from the department to which this product belongs?

5.2.2 Generating User features

1. user_reorder_rate : What is the average reorder rate on orders placed by a user?
2. user_unique_products : What is the count of distinct products ordered by a user?
3. user_total_products : Count of all products ordered by a user?
4. user_avg_cart_size : Average products per order by a user?
5. user_avg_days_between_orders : Average number of days between 2 orders by a user?
6. user_reordered_products_ratio : Number of unique products reordered / number of unique products ordered.

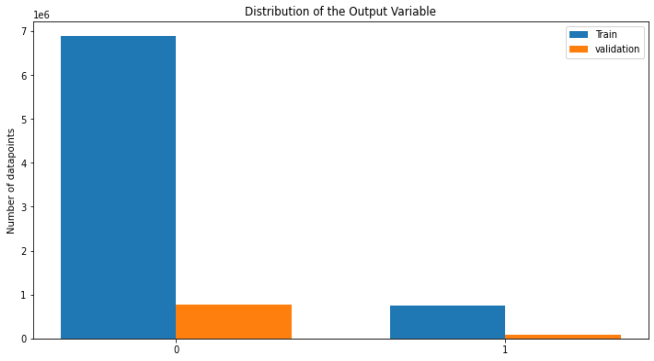
5.2.3 Generating User Product features (creating features based on how the user interacts with a product)

1. u_p_order_rate : How frequently did the user order the product?
2. u_p_reorder_rate : How frequently users reordered the product?
3. u_p_avg_position : What is the average position of the product in the cart on orders placed by the user?
4. u_p_orders_since_last : What's the number of orders placed since the product was last ordered?
5. max_streak : Number of orders where the user continuously brought a product without a miss.

5.2.4 Miscellaneous Features

1. Product features based on time: Reorder frequency of a product given any hour of the day.
2. Product features based on day of week: What is the reorder frequency of any product given any day of week?
3. Product features based on difference between 2 orders: How frequently a product was reordered given a difference between 2 orders (days) containing the product?
4. User feature based on difference between 2 orders: How frequently a user reorders any product given a difference between 2 orders (days)?
5. Product reorder rate based on difference between 2 orders: How frequently a user reordered a product given the difference between 2 orders (days)?

5.3 Generating Training and Test Data



To construct train and test data, we combined all of the features into a single dataframe. By modifying the default data types of columns, we were able to shrink the size of our dataframe. Changing the data types of columns from default to lower range reduced the size by nearly three times. We then used Hierarchical Data Format (HDF5) file formatting to store the data as this kind of file formatting is used for storing large amounts of data.

Since our dataset is large, we went with a 9:1 ratio for test and validation. So, this gives us training data of 7.63 million records and validation data of 850 thousand records.

5.4 Training Models

We trained our datasets on the following models as they were faster to train on our personal computers considering our large dataset.

1) Logistic Regression 2) Decision Tree 3) Random Forest 4) Multilayer Perceptron

5.4.1 Performance Comparison

Confusion Matrices

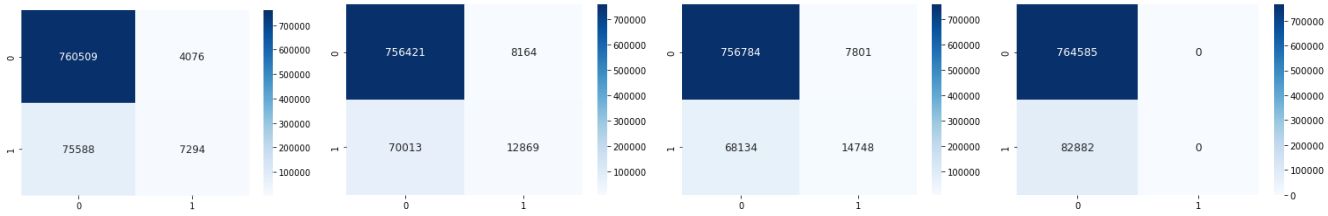


Fig: (a) Logistic Regression (b) Decision Tree (c) Random Forest (d) Multilayer Perceptron

Metrics

Model	Accuracy	Precision	F-score	Odds Ratio	LogLoss
Logistic Regression	90.6%	64.1%	0.950231	18.004584	0.2551
Decision Tree	90.7%	61.2%	0.950864	17.030472	0.2508
Random Forest	91%	65.4%	0.952227	20.998627	0.2518
Multilayer Perceptron	90.7%	0%	0.948586	9.224982	0.2536

5.5 Results

Based on our tests and the comparisons performed, we feel that the random forest classifier worked best for our dataset of the models we tested. The second model that was close was the decision tree model.
We believe that the large part of why Random Forest worked the best was because it improves the generalization performance by constructing an ensemble of decorrelated decision trees. Also, since our data has a lot of redundant attributes, decision tree and random forest algorithms gave us better performance.
The ordering of the performance of models in our tests would be:
Random Forest > Decision Tree > Logistic Regression > Multilayer Perceptron

5.6 Challenges faced while performing classification

1) Class imbalance problem: In our training dataset we had 7.6 million people who had not reordered and 830,000 people who had reordered. So, the challenge we faced was dealing with this and we were not able to figure out how to solve this and oversample or undersample our data.
2) The Multilayer perceptron always ended up learning features that were of non-reordered people and was not able to classify any of the reordered customers on the validation dataset.
3) We can see the same effect of the class imbalance on all the other models too.

6. Conclusion

For generating apriori rules we were successfully able to apply the apriori algorithm and generated unique rules based on certain categories. For the FP-growth algorithm, we faced memory limitations hence ran the algorithm on a subset of the dataset and got some interesting rules.
For customer segmentation, we were able to cluster customers based on their shopping habits using k-means clustering. Hierarchical clustering did not run well due to large data size and memory requirements.
On the user product recommendation part, we tried evaluating a few classification models and were able to get product recommendations of existing and new users. Random forest model seemed to work the best out of all others.

7. References

1. Instacart Market Basket Analysis : Part 2 (FE & Modelling) | by Arun Sagar | Medium

2. In depth understanding of the dataset
3. Feature Engineering structure by Symeon Kokovidis ‘s kernel
4. F1- Maximization implementation taken from Faron’s kernel
5. Dataset reduction technique from Eryk Lewinson’s blog
6. Comparing Apriori vs FP growth using a different dataset called the Groceries Dataset
7. Understanding the pros and cons of FP-growth and Aprioi
8. FP Growth Algoithm basic understanding
9. Simplifying market basket analysis using Data Bricks
10. Comparative analysis between Apriori and FP growth by Zoumana on Kaggle:
11. Research paper based on FP Growth : IJCSNS International Journal of Computer Science and Network Security: May 2020 103 : Association Rule Mining on Customer’s Data using Frequent Pattern Algorithm Khaled H. Alyoubi
12. An alternative approach to use FP growth: Using Spark:
13. Data Analysis notebook on Instacart
14. Clustering segmentation using PCA
15. An end to end comprehensive guide for pca by Analytics Vidhya
16. PCA and partitional clustering
17. Clustering K-means in python by realPython
18. Hierarchical Clustering reference using Instacart dataset
19. Apriori algorithm implementation in python