Caden Weiner

4 May 2021

Course Project Report

Computer Science 315

Frequent Pattern Analysis on Three Million Instacart Market Baskets

Introduction

Patterns appear everywhere, however patterns cannot always be easily interpreted to make quick predictions without additional tools. With the use of computers however, it is possible to examine large quantities of data and make accurate predictions based on previous behavioral patterns. Frequent pattern mining works extremely well with observing trends in user behavior, as well as being able to make judgements based on the rules generated. Being able to understand previous behavior is extremely interesting because it is only possible to make predictions about the future once past behavior is understood. Frequent pattern mining can be applied to many different fields such as bioinformatics, website navigation, decision making and analyzing purchasing history to create recommendation systems.

I chose to investigate patterns in Instacart market basket data because this is information attached to daily life, and something that might be able to help us understand the behavior of people as they go about their lives. Not to mention that this data can be used to help benefit both the customer and the business by recommending items that the user might want and has previously wanted. This can help facilitate the symbiotic relationship between customer's and businesses by providing them the products that they want. From my personal experience, it is very easy to forget to buy something that you may have been planning on or realizing that there is something you needed after you leave the store, and most people may end up stopping at one

of the many other stores that is closest instead. This both wastes the customers time and causes the company to miss out on potential sales. By using association rules, it is possible to recommend to users the items they commonly need based off previous history, as well as based off patterns found in other similar users. Frequent pattern mining can produce great benefits for both business and customers. The goal of this project was to better understand the relationship between customers and their individual behavior, as well as to investigate trends that occur throughout the orders of all users.

I set out to do three things; observe the differences between patterns gathered from individual users' behavior and the behaviors of the collective orders, observe the differences between the apriori basket mining technique and the frequent pattern growth technique, and to come up with ideas about how to apply these observations. I had also set out to make different observations based off the data, such as determining what were common sizes of orders, and which order sizes were most frequent, along with which days users primarily placed their orders on. I also observed the frequencies of different products being order by analyzing all the items that were ordered.

Investigating these questions brought along many challenges and as I began to learn more about the data, tools and results, the more I realized that there was so much more that I wanted to investigate. The first challenge was creating a structure for the program and figuring out how to plan the project. I decided that the best order would be to observe and understand the data set, then I would think of different ways that I could apply the frequent pattern mining techniques. Figuring out how to apply the techniques first required me to transform the data into baskets as the data provided was not in a format that was very easy to read in. It took me a while to understand what I was doing wrong, because my techniques were not working due to memory

issues causing the program to take days to execute. To resolve these issues, I built a system that would allow me to only basketize the data for a specific user which allowed me to test it and understand how the data could be manipulated, this piece of my function could further be reused down the line for investigating trends in specific users. However, I still had to deal with the problem of it taking far too long to go through the algorithm as it constantly had to access a data frame with 30,000,000 items. My solution to this, was to loop through the data frame in segments and that way there would be less usage of main memory, and this drastically reduced the run time allowing an operation that took several days to run withing 70 minutes. However, 70 minutes is still far too slow to run every time and as such, I determined that the best way to deal with this would be to store the baskets in a file and reading from it when the baskets have already been generated. This allowed the program to generate the baskets for 3,000,000 orders within 20 seconds. I also ran into problems figuring out how to use the two algorithms which were implemented in the mlxtend library as I ran into several issues with memory, however I was able to solve them by implementing and understanding several techniques that were discussed in lectures.

In my findings, I found that the data gathered by looking at a particular user was much more effective at predicting their behavior than looking at the system. I initially believed that there may be more patterns such as the diapers and beer discovery, however there were not many rules that showed such distinctions with very high supports. This is due to the wide variety of choices that are available to the users, when there are 632 bread related products, and 811 types of chips, it becomes very difficult to find patterns due to the large variety of options. However, if you observe the behavior of specific users, you can notice obvious patterns, such as when a user that buys many alcohols related products, those products end up being highly correlated and have

strong lift, support and confidence scores as the user follows their own patterns and routines and buys those products frequently. I believe that in order to better understand frequent patterns, it is likely that it will be necessary to find a more precise way to categorize products, so that we can more accurately make predictions based off general user behavior and be able to advertise a wider range of products to users.

Data Mining Task

In order to solve this problem, it was necessary to transform the input data into baskets that can be taken as input by the mlxtend fpgrowth and apriori algorithms as well as the apriori algorithm from the apyori library. The first step of this is to convert the input data into a two-dimensional list of baskets. Our input data consists of products.csv, order.csv, order_products_train.csv and order_products_prior.csv. The file products.csv contains information about all the items such as id, name and information where the product can be found such as aisle and department. Both order_product_train.csv and order_product_prior.csv contain information about all the items that were ordered, such as their product ids and the ids of the orders the products are associated. Prior is a set that contains 32,434,490 and train contains 1,384,618 items. The file that contains the associations between users and orders is orders.csv and it contains information about the orders such as the times of ordering and which user id is associated with what order id and can be used to search for all the orders associated with a user and then using those orders to create the baskets of items. I loaded all this data as data frames, and I associated each of the product ids with its corresponding name so that, when generating the rules, I wanted the orders to be represented by their product name rather than their id so the patterns and rules could be understood much easier. I converted the data frames containing order information into baskets for each order and store it in a file delimited by the symbol `|` rather

than a comma since some names contain commas and other symbols, however none contain the `|` symbol so it can be used for delimiting.

Creating the association between orders and their items is very time consuming and as such, I found storing it and reading it from files when that file had already been generated to be far faster. Initially a key challenge was dealing with the large size of the data frame and the slow time to perform lookups. This was initially causing the data frame to take several days in order to turn all the orders into their basket representation. However, I was able to find a solution to this by segmenting the data into smaller portions and looping through the different portions of the data frame. This allowed me to reduce the usage of main memory required and ended up increasing the speed of generating the baskets for all the orders in seventy minutes rather than taking days. By only generating these new baskets when the file doesn't exist and reading from the constructed baskets when it does it reduces the burden of creating the baskets and allows for the baskets to be read from the file in under twenty seconds which drastically improved my ability to test and experiment on the data.

The data that I generated for the basket orders was in the format of a two-dimensional list, which was the same format as when I generated the baskets solely based on the orders of a single user. I went with storing the data in this format because of several reasons. Firstly, having the data in a two-dimensional list, it was able to be read into the apyori implementation of apriori quickly, and secondly, it was possible to construct the two-dimensional list into a data frame that would be taken as an input for fpgrowth and apriori from the mlxtend library. Converting the list to a data frame required me to first create a transaction encoder object, and then using the fit and transform functions it provides on the list. From there that transaction encoded array would be used as the input to create a pandas data frame and it would use the te.columns_ feature (product

names) as the data frame columns. This transformation was applied to the data for all the orders as well as to the data for individual users' orders. Then by using the association rules function after running apriori and fpgrowth, it generated the association rules, which was a data frame containing information about antecedents and consequences, as well as support, lift and confidence values as well as several other metrics that I did not focus on in my experiments. For the rules of individual users, I printed the top 2 rules by the metrics of lift, confidence and support to a file. However, with the data for all the baskets I sampled different portions of the data. For example, the first 500,000 orders and the last 2,000,000 orders. This allowed for me to notice that items that were frequent in a sample that was greater than $1/10^{th}$ of the data, was highly likely to be frequent in the whole data set as we learned during class. However, with the data generated by the apyori algorithm, it was very difficult to print the rules in a format that was easily understandable due to the way its rules were generated and as such I mainly focused on the other approaches. However, it was able to still print out the necessary rules that were accurate when I checked them against the rules generated by the mlxtend's apriori library function.

The key challenges to solve this task included, memory issues, understanding how to implement the algorithms, and runtime constraints. Memory issues were highly related to runtime being very slow, and this had made it very difficult to test, and it was very taxing on my computer. I has several memory issues, the first one was that trying to perform searches on a data frame that contained over thirty million items in each column was very slow, however I was able to solve this by breaking it into smaller portions of the data frame which drastically reduced memory issues and runtime. Another memory issue that I had was with mlxtend's apriori and fpgrowth functions as they would crash due to not having enough memory when run on the three million baskets. However, I was able to solve this by making use of the knowledge that I could

use parts of the data instead of needing to use all the data, as well as by using the low memory field provided by the apriori algorithm and making the generated data frame sparse.

Once I solved these issues runtime was much faster, and I was able to make observations on what was working in the program and observing the generated rules and making judgements based on them. Once I understood what the needed inputs were for the algorithms, and how they generated data, it was much easier to understand what I needed to do with them.

The key questions that I wanted to observe were as follows:

1. How do frequent patterns very when applied to the orders of specific users verses all the orders of many different users?
2. Are there any noticeable trends for the number of orders for each user, the times when items are ordered, and the day of the week items are ordered? How could we observe changes in the rules generated by these groups?
3. What are the main differences between the FP Growth and Apriori in terms of performance?

Technical Approach

In order to solve this problem, I had to perform several operations on the data in order to prepare it to be used by the algorithms. The first step was to load in the data from all the .csv files into pandas data frames. From there, I dropped the unnecessary columns to help limit the size of the data frame. I then had a function to add an additional field to the orders data frame so that each item in an order had both the name and id associated with. Then, I check to see if the file that contains the orders that we need to process to run the two algorithms. If the file doesn't already exist, we need to generate it which is a slow process as we must constantly search through the data frame, by only generating the information when it doesn't exist it drastically

sped up the run time of the project. In order to generate it needs to break the data frame into smaller chunks so that it can run much faster by utilizing less memory. Now that we have all the data that we preprocess we can run our frequent pattern mining algorithms on the data. I keep display the times that each segment of the program takes to run.

I then ran each of the frequent pattern algorithms on the order histories of 10 random users and the set of all 3,000,000 orders. The results of the algorithm are printed to their respective files, however only the top two rules for the random users are printed. I ran tests by running algorithms on approximately 1,000 random users and tested it by retaking the top rules by support, confidence and lift. Then I proceeded to analyze the frequencies of some of the metrics found in the data such as the dates and times and number of orders and created plots for them. I implemented the fpgrowth and apriori functions from the mlxtend python library.

FP Growth

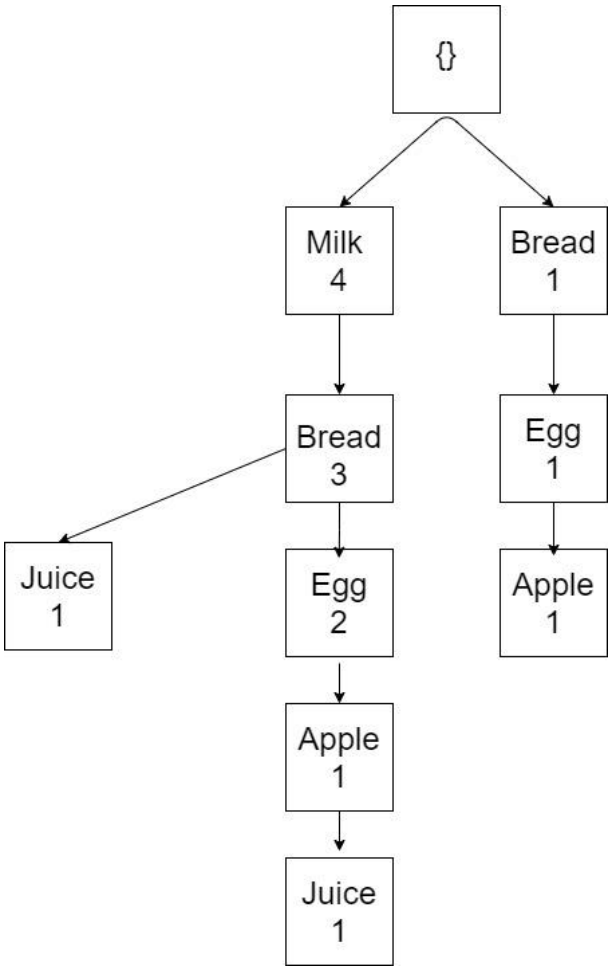
Algorithmic Steps:

1. Scan baskets and find frequent singular items.
2. Sort in descending order based off frequency.
3. Filter out the infrequent items and sort each basket's items by frequency in descending order. This is known as a basket's f-list.
4. Scan the baskets again and construct the fp tree.
5. To construct the fp tree, we scan through the baskets and start at the root each time. The nodes under the root will be generated by constructing trees. This will be demonstrated on a tree below. This step makes use of the principle of the first node having the highest frequency, and as such for those patterns it will not contain any nodes of a higher frequency. In these situations, if a pattern beyond a certain depth of the tree is missing, a

new branch and node is added, otherwise the count of the nodes that match the pattern is increased.

ID	Ordered Frequent Items
1	{milk, bread, egg, apple, juice}
2	{bread, egg, apple}
3	{milk, bread, juice}
4	{milk, bread, egg}
5	{milk}

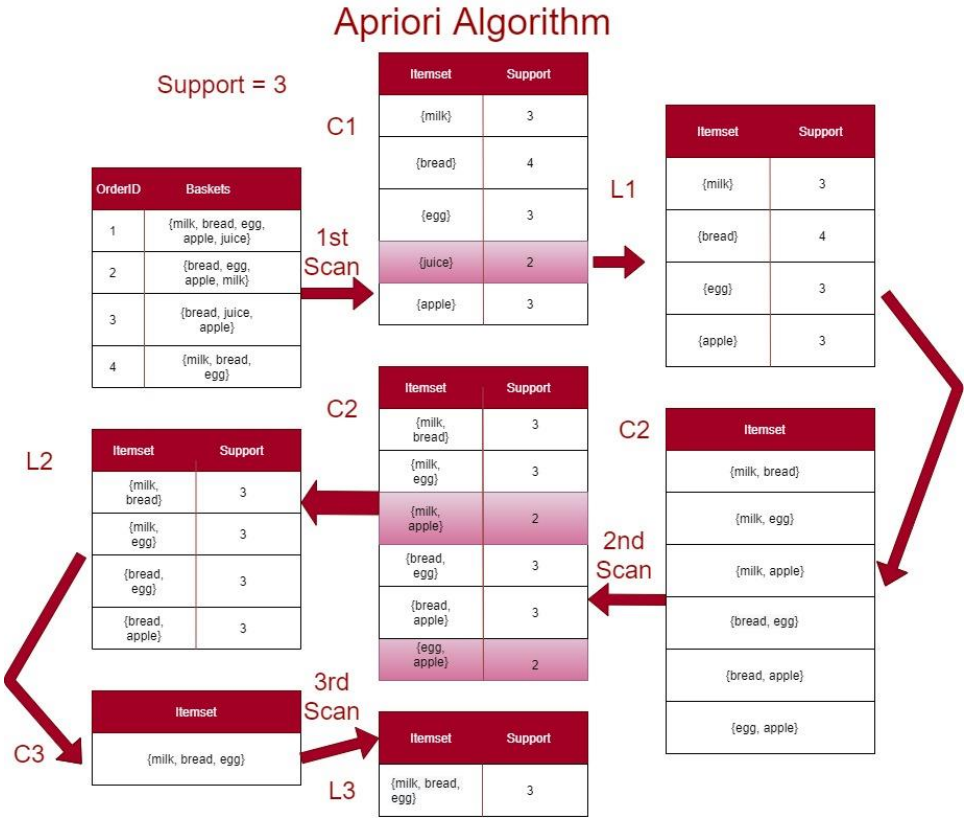
Item	Frequencies
milk	4
bread	4
egg	3
apple	2
juice	2



Apriori

1. Scan the baskets and calculate which individual items pass the support threshold. Where n = 1. This step will help speed the program up greatly by reducing the items it scans.
2. Filter out any items that are not frequent.
3. Produce all combinations of products of size n = n + 1.
4. Scan through the baskets of size n and determine which sets of items are frequent.

5. Repeat steps 2-4 until there are no more items that can pass through the support threshold. Keep track of the supports of all item sets as the algorithm progresses.



Evaluation Methodology

Due to the nature of frequent pattern mining, it is difficult to classify the accuracy of the outputs, as the outputs are specific to the dataset they are mined on, and it is difficult to find a metric to classify the accuracy of the rules that are observed. They are three main metrics that are used to evaluate frequent pattern rules are support, confidence and lift. They all are quite important but often provide different information about the rules. These are the best ways to indicate the accuracy of each individual rule and can be verified by looking at the datasets that ended up generating them. We can also make rational judgements to see more about whether too products are related through correlation or causation. Rules that have lower lift, but a high support, tend to appear, especially in the data set that is made up of all the orders.

The lowest rule by lift generated by this data had a lift of 2.76 and the antecedent was (organic strawberries, Kale) and the consequent was (Banana). These items are all very frequent as well as generic items and they have a strong correlation in our data. However, when we look at the top rules by lift, we see a drastic difference, with a lift of 77.09 with (Non-Fat Acai & Mixed Berries Yogurt) and with (Icelandic Style Skyr Blueberry Non-fat Yogurt) as its consequent. This rule clearly has two products that are highly related and different flavors of the same exact type of product. These two products are clearly similar as they are both nonfat yogurts, and while this rule has a lower support than the rule with lower lift, it has both has higher lift and confidence. I tried taking random samples of user's orders and constantly it seemed like the top rules by lift and confidence were much more accurate than the rules where support was taken as the most important factor. It is also important to note that the rules with higher confidence often had much higher lift values.

It appears, based off my observations of the data that lift is a more important factor in indicating correlation than support is. Also, while confidence can be a good indicator of the importance of a rule, lift is much more important for classifying how dependent two products are on each other. These observations are also backed up by the technical reasoning behind the lift metric, which is to account for the popularity of the antecedent when predicting the likelihood of purchasing the consequent. This was the main reason for choosing lift as the threshold value that I placed the most importance on, and my tests on the dataset helped to verify this. This helps to consider how given rules with high confidence could potentially be a fluke due to the popularity of an item, where items with lower confidence might be highly connected with another item. Another thing that is important to note is that it is also possible to only generate rules for items that pass a certain threshold, in the case of my experiments I chose to limit the items that did not

pass a confidence threshold of .4 on the data set of three million items. It ended up being a sweet spot as it helped limit the rules to the most important ones.

It was extremely important to note how the parameters used on the orders of specific users varied greatly compared the data set that was compiled out of all the user's orders. For individual users, I found the rules to be highly accurate, and highly tailored to the specific user. When evaluating which methods to use I ran into difficulties as a result of user's order behavior being very different than looking at a data set that contained many orders from different users. The order histories of customers would often produce results with high support, confidence and lift. This is the outcome that I had initially expected, however it came at the cost of run time, since they may generate many rules that are important, but due to the quantity of the rules they are consequentially less important. Thus, I ended up choosing a confidence threshold of .65 which allowed for me to prevent many unnecessary rules from being generated. One of the hiccups I ran into in this phase was caused by not being able to choose one threshold that worked well for all users in terms of confidence and support. I had to choose rules for the support based off the number of orders a user had since users with few orders would end up with all their items being frequent, thus not allowing them to exploit monotonicity. Therefore, I choose .3 as the support value for users of that had at most 5 orders, and then I reduced it as the number of baskets a user had increased. I also had to limit how many items could be generated for a rule and chose 5 as my threshold for max length, thus the longest possible antecedent or consequent would be (a,b,c,d,e). This allowed for me to prevent the program from running out of control, while still allowing it to generate larger and more specific rules.

In an industry setting its often the case that they only care about the highly relevant results, so it is important to have a metric that can be used as a strong indicator of product rules

as many places sell thousands of products. For example, in our dataset there is 49688 items and there are so many possible combinations that the number takes up an entire page. It is extremely important to only consider the most relevant combinations due to the sheer number of combinations, and therefore in these algorithms it is extremely important to strive for monotonicity. By having a support threshold, we can eliminate all the infrequent items that do not pass the support threshold, and this leads to monotonicity and limits the number of combinations that are needed to check. We can further limit the combinations we care about by filtering out all the results that don't match the metrics that we are searching for. These thresholds may need to be changed depending on the data and the understanding of that data. For example, in the set of all orders, there are only 47 item combinations that pass our support threshold of .001, which indicates that they data generates many different results due to all the possible items that the user can choose from leading to all the frequent pattern rules having very low support values. I chose this specific number in order to generate a substantial number of rules while at the same time keeping monotonicity to improve runtime. Decreasing the support threshold too much will generate many more rules, and while there may be information gained from these rules, there are tradeoffs to be made when choosing parameters.

There are also several other considerations that are necessary to consider such as time and memory. For example, I had to limit the search of frequent patterns above a certain size, in my case I chose the maximum items in a rule to be five. This was due to memory issues where there would be baskets of very large sizes and it would continue to generate more and more rules, and as it was not able to drop any of the rules it would not obtain monotonicity thus causing the program to run very slowly on individual users. These baskets generated had lower support due to the support parameters chosen for user baskets as having a support of 10% means that an order

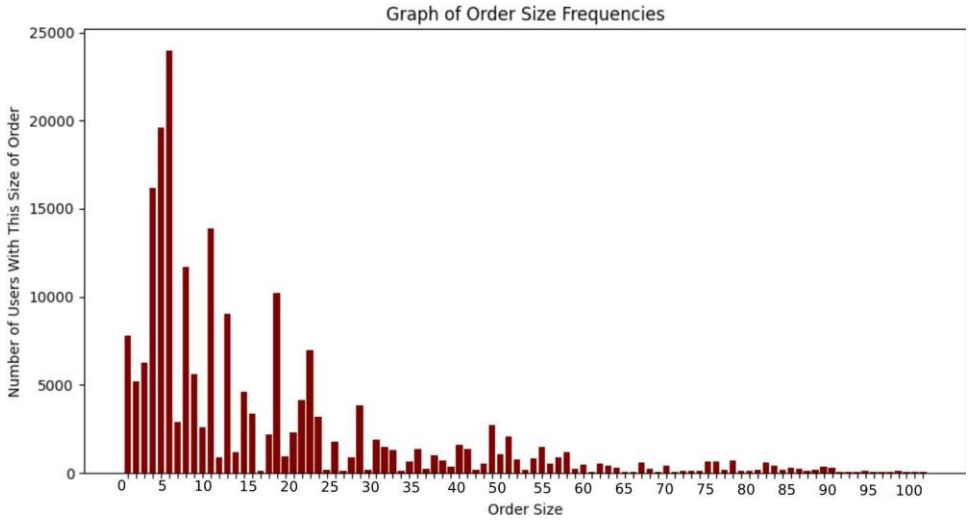
that appeared once in a user's order history would cause all the items to be frequent. I adjusted the algorithm to help handle that, however as orders tend to have high tendency to repeat different item purchases, many items would still pass these thresholds. By limiting the size of rules that could be generated, I may have lost some information about rules with lower support, confidence and lift, but gained improved efficiency.

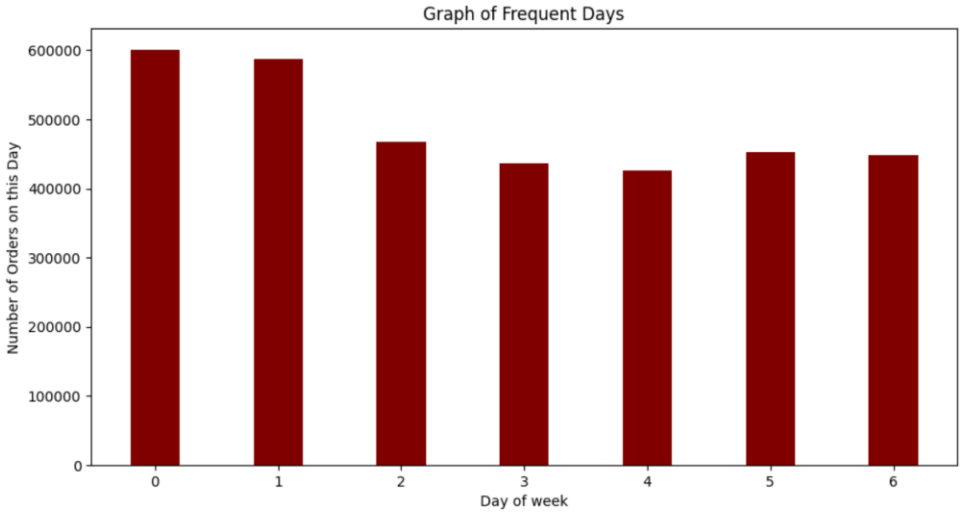
The main challenge of the data was the size of the data set, as this caused it to use up a large amount of main memory and made accessing the data frame costly. I was able to solve this problem by having the data frame break up the data into smaller slices and then store all of the data in a file with each order being represented by a line and delimited with the `|` symbol, which worked best based on the product names. This allowed the generation of the baskets to take place using a preprocessing technique and allowing the results of creating the orders to be accessed within seconds rather than hour. This same problem was not solvable for individual users and using the data frame structure to create orders for individual users was more efficient.

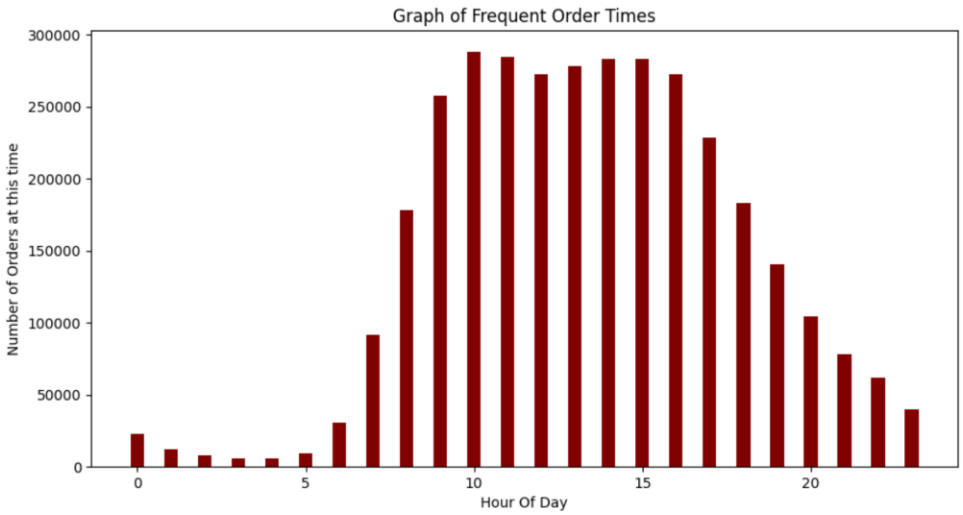
Results and Discussion

Through my investigations, I also tried to investigate how the information about the times of orders might be used to investigate the frequent patterns differently. Some of the assumptions that I had were that I might have seen a different trend in the type of groceries ordered on the weekend. In order to observe where possible differences in the order contents might occur, I plotted the frequencies of the times of orders, days of orders and the frequencies of the number of orders a user had. I was not able to find any apparent trends as I had expected, however, upon research I discovered that most people shop on the weekends, but it is not necessarily related to what they purchase. These fields were investigated to get a better understanding of the data

rather than being the goal of my experiment. I believe that continuing to do more investigation into how these fields could be used to observe new patterns based on user behavior.







Throughout this project, I discovered important information about how rules are reliant on the variation in the types of products. While a user may consistently choose a small subset of products, when there are many users, they often make many different choices about the products they are consuming. This can be visualized through the rules we generated for individual users when compared to the rules generated for the whole dataset. Rules for a specific user were specific and understandable. These rules also had much higher support, confidence and lift values on average. I also discovered that in terms of the importance of features, lift is more important than confidence which is more significant than support through viewing the rules and research the topics.

As I dealt with memory issues when running the apriori and fpgrowth algorithms, I realized that I could make use of several important principles. The key principle was that we can determine frequent items over the whole data set by taking a random sample of it. In my experiments, I ran it on sizes of 500,000 – 2,000,000 as there were memory issues when I exceeded 2,000,000 items, however I found that they all shared the same rules, with some variations of the rules in portions of the dataset but not in others. This was a solid indicator of what the trends were in all the data and the observations made by the data were understandable,

as many of the rules contained frequently purchased and generic products such as fruits and vegetables. Other rules that were generated in this data set were highly related and the rule with the highest lift was two kinds of low-fat yogurts. It makes sense that two variants of a product would be purchased together as people commonly like to buy similar versions of the same products to have different flavors. I believe that these observations worked very well to categorize those frequent patterns in the data, however I believe that for an online retail company such as Instacart, these patterns are far less important than rules for specific users. Individual users tend to follow their own patterns and while the trends of a group can be insightful, they are not as helpful when it comes to making predictions for a specific user. The rules generated for specific users followed clear patterns most of the time, such as a customer who bought several kinds of alcohol being likely to purchase another type of alcohol together with it. Frequent patterns generated for the entire set requires products to be popular for all users, and with 49668 products, it is only able to generate rules for very popular items with fewer alternatives. For example, it is hard to generate a rule with milk for the whole set because there are around 1,000 products that contain milk. With so many choices it is highly unlikely to generate a rule with high support for a milk product despite many products falling under the category of milk products. Due to this, I don't think using the whole dataset to generate rules is the correct approach, but instead I believe that generating rules based off specific users order history will generate the best possible rules.

Another goal in my investigation was to determine the differences between fpgrowth and apriori. I found that the two algorithms were able to generate the same rules with the same values for support lift and confidence, however the differences occurred when it came to runtime and memory usage. The algorithm of fpgrowth is much more memory intensive, however as a

tradeoff to this, its runtime performance is much quicker than apriori. While apriori is less memory intensive it runs slower than fpgrowth. When running apriori on the 2,000,000 orders it took 2852.8 seconds, whereas fpgrowth took only 1448.8 seconds. Clearly fpgrowth is much faster than apriori when there is sufficient memory.

There are several folders in the project which contain .txt files that store the rules generated. This worked well for storing the results for the mlxtend functions but the reason that I moved away from focusing the apyori library's function was because it did not present the data in a way that was easily accessible, understandable and storable. This likely has a way to get it to store the results properly, however I could not find that information in the documentation of the library.

The mlxtend functions were the two that I had thought worked best because they were able to quickly provide the important rules and their data structures were understood more clearly as data frames. As well as the fact that both functions belonged to the same library. However, one of the hurdles to dealing with them was memory issues, which I was able to account for by making the data sparse and using the low memory mode that their functions provided. While this did cause the code to run slower, it was only used for the larger datasets and as both functions utilized the feature their efficiencies were still comparable.

Lessons Learned

In this project I learned the importance of combining a variety of datamining techniques. I utilized preprocessing, limiting memory usage as well as wrote efficient code without redundancy. It was also necessary for me to learn about understanding different libraries that are available to use and how their documentation is written. I also learned about the fpgrowth technique which is a frequent pattern mining technique that I was unfamiliar with despite its high

popularity. Through these experiments I was able to get more familiar with pandas' data frame objects and I was able to learn about how the data that the techniques are used upon is just as important as the technique itself. If the data set contains many possible variations, products that are popular win out, despite not necessarily being the indicators of trends in specific user behavior. Thus, I learned that it is important to understand the data, and how different samples of data can have different outcomes depending on where they are gathered from. This also played into my understanding about how these types of systems can be biased. For example, in an area where certain specialty products are popular, we might see certain rules appear, however if we tried to apply those same rules to a different area, they likely would not be successful.

In hindsight, I would have started doing a bit more research at the beginning and I also would have tried to figure out the issues that were causing slow runtimes much faster. Due to how slow the runtimes were, it was very hard to find the flaws in my program or what was causing the issues. I eventually figured out that it was a problem caused by memory and was able to fix it, however if I was able to understand that memory was an issue sooner, I would have been able to move on to researching different questions that I could answer related to the data. I really wanted to investigate how different factors such as time, day and the number of orders a customer currently had could affect the outcome, however, there just was not enough time to investigate all the details of these scenarios. The main decision that I think could have helped to resolve the memory and runtime issues would have been to utilize a database system such as SQL. Although I am not super familiar with SQL, I have some experience working with SQLite and I believe that using a system like this would be a more efficient way of storing the data than text files.

Important Formulas

Lift = Lift (a,b) = Support({a,b})/(Support({a}) * Support({b}))

Confidence = Confidence (a, b) = Support({a,b})/Support({a})

Support = Support(a) = baskets containing {a} / total baskets

Acknowledgements

1. Chonyy. “FP Growth: Frequent Pattern Generation in Data Mining with Python Implementation.” *Medium*, Towards Data Science, 10 Nov. 2020, towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3.
2. D, Kavyashri. “Mining Frequent Itemsets - Apriori Algorithm.” *DWgeek.com*, 15 Mar. 2018, dwgeek.com/mining-frequent-itemsets-apriori-algorithm.html/.
3. Moffitt, Chris. “Introduction to Market Basket Analysis in Python.” *Practical Business Python Atom*, 3 July 2017, pbpython.com/market-basket-analysis.html.
4. Ng, Annalyn. “Association Rules and the Apriori Algorithm: A Tutorial.” *KDnuggets*, www.kdnuggets.com/2016/04/association-rules-apriori-algorithm-tutorial.html.
5. Rasbt. “Supporting out-of-Core Processing in the Apriori Function for DataFrames That Don't Fit into Memory · Issue #298 · Rasbt/Mlxtend.” *GitHub*, github.com/rasbt/mlxtend/issues/298.
6. Raschka, Sebastian. *Association Rules - Mlxtend*, rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/.
7. Piush. Vaish. “Apriori Algorithm (Python 3.0).” *A Data Analyst*, 1 Jan. 2019, adataanalyst.com/machine-learning/apriori-algorithm-python-3-0/.
8. Unknown Author: “FP-Growth Algorithm.” *KOPO.COM*, www.kopo.com/?sid=9&lid=18.