Market Basket Analysis

Determining the best approach for Market Basket Analysis

Aparna Devi Akula
Data Science
CU Boulder

Aravindh Saravanan
Data Science
CU Boulder

Chanthrika Palanisamy
Data Science
CU Boulder

Siri Devarapalli
Data Science
CU Boulder

ABSTRACT

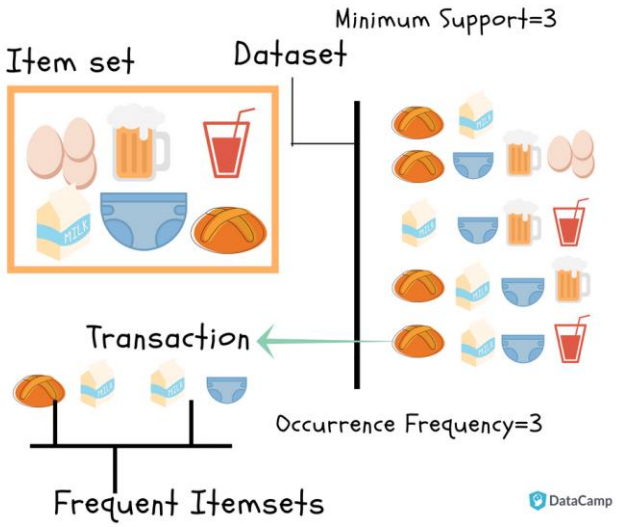
Data mining is a method to identify important patterns in Frequent Item Set Mining (FIM). Market basket analysis (i.e. Data mining technique in the field of marketing) is the method to find the associations between the items / item sets and based on those associations we can analyze the consumer behavior. Apriori is the part of the association rule used to determine the association relationship in a set of elements. But apriori has a major computational weakness because finding sets of items frequently has to iterate through the database for each combination of items. This study aims to find out different methods to find frequent itemsets and compare the algorithms in terms of time taken to perform the algorithm on the same dataset. We. will compare apriori, fp- growth, ECLAT and try to improve the fastest algorithm. Apriori turned out to be the fastest algorithm to our surprise. The main problem that was encountered by all the algorithms was insufficient memory space. We came accross a study that used generators to continuously output results instead of storing all the outputs and returning the entire result at once which causes memory insufficiency as a consequence. We tried to manipulate the apriori algorithm to improve the speed of algorithm. We worked on transactional data of instacart and compared the algorithms which will future studies to choose algorithms based on their datasets. We named the new algorithms as fast apriori and Super fast apriori.

1. INTRODUCTION

Market Basket Analysis is one of the key techniques used by large retailers to uncover associations between items. It works by looking for combinations of items that occur together frequently in transactions. To put it another way, it allows retailers to identify relationships between the items that people buy.

Association Rules are widely used to analyze retail basket or transaction data and are intended to identify strong rules discovered in transaction data using measures of interestingness, based on the concept of strong rules.

There are 3 algorithms that can be used to create association rules.



1.1. Apriori Algorithm

This algorithm is part of the association rule that is used to determine the associative relationship of an item combination. Apriori algorithm uses frequent itemsets to generate association rules. It is based on the concept that a subset of a frequent itemset must also be a frequent itemset. Frequent Itemset is an itemset whose support value is greater than a threshold value (support). Apriori algorithm will be suitable to be applied if there are several relationship items to be analyzed. Apriori algorithms are widely used by researchers for various aspects, among others, product arrangement, prediction. However, the

Apriori algorithm has a weakness of computational time which is quite high because the frequent itemset search process must scan the database repeatedly for each combination.

1.2. FP Growth

It is an improved version of the Apriori Algorithm which is widely used for frequent pattern mining. It is used as an analytical process that finds frequent patterns or associations from data sets. FP Growth is much faster than Apriori. The reason why FP Growth is so efficient is that it's a divide-and-conquer approach. And we know that an efficient algorithm must have leveraged some kind of data structure and advanced programming technique. It implemented tree, linked list, and the concept of depth-first search. It greatly reduces the size of the itemset in the database by one simple principle: If an itemset is frequent, then all of its subsets must also be frequent.

1.3. ECLAT Algorithm

This algorithm is used to generate item sets. It uses vertical databases as an input. The vertical nature contributes to the speed of the algorithm as it scans the database only once. The algorithm can start from, say, node and counts its support to determine whether it is frequent. If so, the algorithm progressively expands to the next level of nodes i.e ab, abc, and so on, until an infrequent node is reached, say abcd. It then backtracks to another branch, say abce, and continues the search from here. Eclat uses a purely vertical transaction representation. No subset tests and no subset generation are needed to compute the support. Generally, Transaction Id sets which are also called as tidsets are used to calculate the value of Support value. The support of item sets is determined by intersecting transaction lists. The main advantage of the vertical format is support for fast frequency counting via intersection operations on transaction ids (tids) and automatic pruning of irrelevant data.

1.4. FAST APRIORI

Fast apriori algorithm would pivot the table in such a way that the transactions are in columns and each row is an item. This helps apriori as it does not have to scan the database to find which transactions has that item. Thus, decreasing the number of times scans are required. It follows the same approach as apriori to generate frequent itemsets and association rules.

1.5. SUPER FAST APRIORI

Super-fast algorithm uses generators to decrease the memory required by the algorithm. It ends the result as soon as it is generated instead of storing all the results and sending them once all the results are computed. A generator is a special type of function that returns an iterable sequence of items. However, unlike regular functions which return all the values at once (eg: returning all the elements of a list), a generator yields one value at a time. To get the next value in the set, we must ask for it - either by explicitly calling the generator's built-in "next" method, or implicitly via a for loop. This is a great property of generators because it means that we don't have to store all of the values in memory at once. We can load and process one value at a time, discard when finished and move on to process the next value. This feature makes generators perfect for creating item pairs and counting their frequency of co-occurrence.

2. KEYWORDS

Market basket analysis, Apriori, ECLAT, K-Modes, Generators.

3. RELATED WORK

3.1. *Manpreet Kaur, Shivangi Kags. 2018. Market Basket Analysis: Identify the changing trends of market data using association rule mining. International Conference on Computational Modeling and Security (CMS 2016).*

In Manpreet Kaur and Shivangi kangs paper they tried to identify the changing trends of market data using association rule mining. Their proposed algorithm not only mines static data but also provides a new way to consider changes happening in data. This paper discusses the data mining technique i.e., association rule mining and provides a new algorithm which may help to examine the customer behavior and assists in increasing the sales. They used change modelling to understand the dynamics of data generation process by examining changes that have taken place in discovered patterns. Change modelling works on dynamic data and performs periodic mining.

In the paper evaluating the performance of apriori and predictive apriori algorithm to find new association rules based on the statistical measures of datasets by Mukesh sharma, Jyothi chowdary and Gunjan sharma the performance of two most important algorithm i.e., apriori and predictive apriori algorithm are chosen and compared based on the interesting measures using weka3.7.5 which is a java-based machine learning tool. Also, various statistical measures are calculated of different datasets and then based on the comparison of algorithms and statistical measures of

data, new rules are generated using see5 tool. They concluded that predictive apriori performs better based on predictive accuracy and recommends an algorithm by analyzing the mean, median and range of a dataset for finding the new association rules which are applied on the various datasets.

3.2. *Dr. M. Dhanabhakyam, Dr. M. Punithavalli. 2011. A Survey on Data Mining Algorithm for Market Basket Analysis. Global Journal of Computer Science and Technology Vol 11 Issue 11*

The paper, A Survey on Data Mining Algorithms for Analysis provides some of the existing data mining algorithms for market basket analysis. This survey is important because a lot of businesses are becoming concerned about mining association rules from their databases. For instance, the detection of interesting association relationships between enormous quantities of business transactions data can assist in catalog design, cross-marketing, loss-leader analysis, and various business-making processes. Mining Association rules are one of the most important fields of data mining. According to this rule, provided a set of customer transactions on items, the main intention is to determine the correlation among the sales of items.

The Association rule is **X => Y**, where X is referred to as the antecedent and Y is referred to as the consequent. X and Y are sets of items and the rule represents customers who purchase X are probable to purchase Y with probability %c where c is known as the confidence.

The problem discussed in this paper is that there are various difficulties faced by the Apriori algorithm. It scans the database a lot of times therefore it results in Super low efficiency. The drawbacks can be overcome by modifying the apriori algorithm effectively and using the fast apriori algorithm.

3.3. *Christian Borgelt. 2003. Effective Implementations of Apriori and Ecla*

The paper Efficient Implementations of Apriori and Eclat by Christian Borgelt compares apriori algorithm to ECLAT and concluded that for closed item sets the more efficient filtering gives Apriori a clear edge with respect to execution time, making it win on all five data sets. For maximal item sets the picture is less clear. If the number of maximal item sets is high, Apriori wins due to its more efficient filtering, while Eclat wins for a lower number of maximal items sets due to its more efficient search.

A Lot of algorithms have already been designed for generating frequent itemsets. One of the algorithms that we have used is the Eclat algorithm. Eclat algorithm uses vertical dataset and bottom-up approach for searching items in database. But eclat algorithm has some limitations. For example, an enormous number of iterations are required for processing the items and more escape time is required for finding frequent itemsets. Eclat algorithm uses bottom-up approach that is extraordinarily complex. We have improved eclat algorithm to reduce escape time and number of iterations by using top-down approach and transposing the original database.

3.4. *Sohaib Zafar Ansari. 2019. Market Basket Analysis: Trend Analysis of Association Rules in Different Time Periods. Dissertation*

This paper includes variability of time, because with the change in time the habits or behavior of the customer also changes. For eg: people wear warm clothes in the winter and light clothes in the summer. The paper attempts to discover association rules that display regular cyclic variation over time. The main aim of MBA is the detection and analysis of purchase behavior of the customers (items purchased together). Based on variability in demographics and difference in the segment of customers, every store will have different results. This research focuses on a single store and its transactions which will be giving a comparative analysis in different time
periods. This way we will be able to investigate the fluctuation of the consumer behaviors on purchases with the change in time.

Long Tail Effect often refers to products purchased in supermarkets describing their distribution as a long tail in which a small number of products is purchased more frequently whereas a large one is purchased less frequently. This phenomenon creates data sparsity problem and worsens, even more, their elaboration. For this research from the total number of transactions there were selected, those that included at least the purchase of 8 products to resolve the data sparsity problem and prevent the removal of any market basket product, which may present a risk of information loss.

4. MAIN METHODS

4.1 Apriori Algorithm

The work steps of Apriori algorithm can be described as follows:

1. Determine minimum support
2. Calculate items from support (transactions that contain all items) by scanning the database for 1-item set; Perform the 2-itemset combination from the previous k-itemset.
3. Then calculate support by scanning the database for 2-itemset.
4. Itemset that meets the minimum support will be the frequency of the candidate.
5. Set k-itemset value from support which meet minimum support from k-items.
6. Perform the process for the next iteration until there are no more k-itemset that meet the minimum support.

The Formulae to calculate Support, Confidence and Lift:

$$\text{Support (A)} = \frac{\text{Number of transaction in which A appears}}{\text{Total number of transactions}}$$

$$\text{Confidence (A}\longrightarrow\text{B)} = \frac{\text{Support(AUB)}}{\text{Support(A)}}$$

$$\text{Lift(A => B)} = \frac{\text{Confidence(A => B)}}{\text{P(B)}} = \frac{\text{P(A U B)}}{\text{P(A)P(B)}}$$

We tried running the apriori function which is a built-in function in mlxtend. The function takes a one- hot encoded data frame as input along with a criterion such as minimum support. We tried to run the function on the entire dataset. It crashed claiming insufficient memory space. Our professor suggested working with samples instead of the entire dataset. Once we started working with samples, we had to find which size was suitable and the threshold of the algorithm. For apriori the threshold we could find was 30,000 transactions. 50,000 transactions proved to be an overburden for the algorithm. The time-consuming functions were one hot encoding for the data and the apriori function was faster than one hot encoding.

4.2 FP Growth

FP – Growth is an improved version of Apriori. The main disadvantage of apriori is the fact that it scans the database multiple times to find each support value. Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree. Before constructing the FP-tree, reorder the transaction based on their item frequency. When start constructing the FP-tree, we need to create a header table that records the location of all item node with a link list. Each time a new node is added to the tree, it needs to be linked to the last node with the same item. The header table is essential to build conditional FP-tree in the later steps. Now we have scanned the database twice and build the FP-tree. All the transaction information is contained in the tree. The only thing left to be done is to recursively construct the conditional FP-tree to find all frequent itemsets with support count larger than 2.

For FP growth we used the mlxtend package. The benefit of using this package was the generation of association rules. We expected FP growth to perform better than apriori, but FP growth couldn't handle the entire dataset. So, we tried giving 50,000 transactions. It ran with a higher execution than apriori. This was shocking so we proceeded with the next algorithm.

4.3 ECLAT (Equivalence Class Clustering and Bottom-Up Lattice Traversal)

It is an algorithm for association rule mining. The basic idea is to use Transaction Id Sets(tidsets) intersections to compute the support value of a candidate and avoid the generation of subsets which do not exist in the prefix tree. In the first call of the function, all single items are used along with their tidsets. Then the function is called recursively and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs. This process is continued until no candidate item-tidset pairs can be combined.

We will use the pyECLAT package in Python. It is a very easy-to-use package. Once you've installed the package, you need a way to enter the data. In this case, we need to enter the data as a list of lists. Each transaction is represented as a list of products. The pyECLAT library

takes a data frame as input. We converted the list of transactions into a data frame and the pyECLAT package will take care of the rest. Now that we have the data, we need to specify several algorithm parameters. Firstly, we need to specify the smallest itemset size that you are interested in. In this case, we are interested in product associations, so we want to leave out individual (1-item) itemsets: the minimum size needs to be 2.

We also need to transcribe our minimum support value as a percentage.

Finally, the pyECLAT package wants us to specify a maximum size. We do not have a maximum size for the itemsets (we would be interested in large product associations as well). Therefore, we take the maximum transaction size. Now that we have all the necessary inputs, we can finally start using the pyECLAT package. There are two steps to this: we first instantiate, and then we fit the algorithm. The fit method returns two things: the so-called association rule indices and the so-called association rule supports.

So while performing the algorithm, we started with a sample that consists of 50,000 rows, it didn't work we ran into array out of bounds error. Later we took a sample of 5000 rows and formed a data frame called as transactions (contains all products that took in a single transaction) and gave this as the input for ECLAT function.

4.4 FAST APRIORI

At this point we understood that apriori algorithm is the fastest algorithm when compared to FP growth. We decided to explore apriori more. This is when we came across a slight improvement of the apriori algorithm

The work steps of Apriori algorithm can be described as follows:

1. Determine minimum support
2. Calculate items from support (transactions that contain all items) for 1- item set; Perform the 2-itemset combination from the previous k-itemset.
3. Then calculate support by scanning the database for 2-itemset.
4. Itemset that meets the minimum support will be the frequency of the candidate.
5. Set k-itemset value from support which meet minimum support from k-items.
6. Perform the process for the next iteration until there are no more k-itemset that meet the minimum support.

The basic idea was to improve the run time of apriori algorithm. Apriori takes so much time with large datasets because it must scan so many times to find the support of each item. We tried to pivot the table to decrease the number of scans required for the apriori algorithm. This reduced the execution time drastically. When we tried running the entire dataset it couldn't handle the dataset as it was trying to store all the results and then send the result as one data frame. This was consuming a lot of memory. This is a problem especially when we deal with datasets with categorical values as outputs and datasets with millions of rows.

4.5 SUPER FAST APRIORI

Apriori is an algorithm used to identify frequent item sets (in our case, item pairs). It does so using a "bottom up" approach, first identifying individual items that satisfy a minimum occurrence threshold. It then extends the item set, adding one item at a time and checking if the resulting item set still satisfies the specified threshold. The algorithm stops when there are no more items to add that meet the minimum occurrence requirement. Once the item sets have been generated using apriori, we can start mining association rules. Given that we are only looking at item sets of size 2, the association rules we will generate will be of the form {A} -> {B}. One common application of these rules is in the domain of recommender systems, where customers who purchased item A are recommended item B. Each sub task is made into different functions.

Given the size of the dataset which was around 32 million records and about 50k unique items, it becomes very difficult to perform any transformation or computation on the dataset. This meant that for even simple transformations like one hot encoding absurd amount of memory was needed.

To tackle this, we used Python Generators. A generator sssss a special type of a program that returns an iterable sequence of items. The key difference is that while a function returns all values at once, a generator returns one value at a time. This means that all the values are not stored in the memory at once and only one value is stored at a time, freeing a lot of memory. Memory We can load and process one value at a time discard when finished and move on to process the next value. This feature makes generators perfect for creating item pairs and counting their frequency of co-occurrence. Let us see generators in action:

Get all possible item pairs for a given order

eg: order 1: apple, egg, milk item pairs: {apple, egg}, {apple, milk}, {egg, milk}

order 2: egg, milk → item pairs: {egg, milk}

Count the number of times each item appears

eg: {apple, egg}: 1

{apple, milk}: 1

{egg, milk}:2

The generator function first creates all possible pairs for a given transaction, returns them one by one as a first step. The next step would be to count the number of times each itemset appears.

Once the item sets have been generated using apriori, we can start mining association rules. Given that we are only looking at item sets of size 2, the association rules we will generate will be of the form {A} -> {B}. One common application of these rules is in the domain of recommender systems, where customers who purchased item A are recommended item B.

5. EVALUATION

We can conclude our result by evaluating the metrics time taken to execute the algorithm and number of rows the algorithm can handle. We will compare the results made by Apriori, FP growth, ECLAT and two other different approaches to the apriori algorithm to find which algorithm is better based on two scenarios. If the algorithms provide similar outputs (frequent itemsets) then we will compare the algorithms based on time. If the algorithms generate different itemset then we compare the results based on interpretability, how the algorithm works with different data types.

The evaluation is also based on space complexity because some algorithms were not able to run more than certain rows of data.

The below table represents different algorithms, number of rows they can handle, and their execution times;

S.No	Algorithm	N rows	Execution Time
1.	Apriori	30000	2 mins 50 sec
2.	FP Growth	30000	3 mins 43 sec
3.	ECLAT	5000	42 mins 33 sec
4.	Fast Apriori	50000	48 sec
5. i	Super-Fast Apriori	50000	1 sec
5. ii	Super-Fast Apriori	32000000	6 mins 39 sec

It was surprising to us that the algorithm was able to handle such a large set of data hence we coined the term Super-Fast Apriori as apriori is basic function call for this algorithm.

6. ISSUES FACED

While discussing what project to do for the data mining we came across market basket analysis. It is a core problem which many companies like amazon have conquered. Understanding the algorithms involved and exploring ways to improve them to find the most optimal way to handle large datasets was the problem we took.

Finding the dataset which was compatible with most algorithms and had the columns we needed was hard. We searched on Kaggle, google datasets and generally on the web. We came across a Kaggle competition where the competition's goal was to analyze the Instacart dataset. Many participants had used apriori and FP growth to find frequent itemsets and consequently association rules to find out customer behavior patterns.

The next issue we faced was to get the data into a platform that we could use. Google colab was one of the top contenders. But, it could not get the data onto a drive. We tried using our local systems but that would mean that version control would be hard. We eventually settled for local systems because no online platform could handle the dataset as well as the computational power required to combine the data into one dataframe. We realised this when we tried to combine our data in the preprocessing stage.

We finalized our dataset because it had the items in one column. But our dataset was a combination of 6 datasets. Each dataset was a table which was linked to another table by one or more columns. We studied the dataset to understand that we did not need some tables sch as aisle. This reduced our dataset to 5 datasets. We first combined

departments and aisles to products on columns departmentID and aisleID respectively. Products dataset was combined to train data. Train data had productID and order ID. This dataset had each product in one row. This meant that we did not have 32 million rows but 32 million products purchased. This was a new insight that we had to address. We then combined the prior data. This data was products purchased by the same customer but prior to the train dataset.

Studying the dataset was an important part of the preprocessing and cleaning stage. In the preprocessing stage we checked for null values. Dealing with null values was different for categorical and numerical columns. The missing values in the numerical values were replaced with the values which were present in the most similar row. All rows with missing categorical values were deleted as they were almost negligible when compared to the entire dataset.

We visualized the data for preliminary analysis and to get an insight as to which rules might be generated. The most popular item was bananas from the produce department. This meant that bananas had to be one of the top association rules to be produced by any algorithm. We ran one hot encoding on the dataset as apriori and FP Growth needed a one hot encoded dataframe as input.

We started building Apriori algorithm using the entire dataset which consists of thirty-two million entries and got an error "ran out of space or array out of bounds". So, we tried using one millions rows which also didn't work, at last we nailed our dataset to thirty thousand rows and then Apriori algorithm worked on that.

Later we worked on FP Growth algorithm, taking into consideration that FP Growth is more efficient than Apriori we used one million entries worth dataset and started implementing it. Again, we ran into the same issue and got an error "ran out of space or array out of bounds". So, we were expecting at least fifty thousand entries would work but it didn't. So as in Apriori we took a dataset which has thirty thousand rows and then FP Growth algorithm was successfully implemented.

Next coming to the challenges faced while working on ECLAT algorithm, ECLAT is the algorithm that took the most of the time to figure out which approach to follow. Based on the previous experience we decide to start the algorithm only with thirty thousand entries. We wrote an algorithm using the data set which consists of all the variables, we were able to derive till frequent items but we couldn't derive association rules this is a biggest challenge. So again we started writing a different algorithm but not using the dataset which contains all the variables, instead we formed a dataset called as transactions which contains all the items that were in a particular transaction Ex: Transaction 1 contains ['Bag of Organic Bananas',' Organic Hass Avocado',' Cucumber Kirby',' Organic Whole String Cheese',' Organic Celery Hearts',' Organic 4% Milk Fat Whole Milk Cottage Cheese',' Lightly Smoked Sardines in Olive Oil',' Bulgarian Yogurt'], so with this dataset we were able to derive frequent items and association rules.

Next coming to the challenges faced while working on ECLAT algorithm, ECLAT is the algorithm that took the most of the time to figure out which approach to follow. Based on the previous experience we decide to start the algorithm only with thirty thousand entries. We wrote an algorithm using the data set which consists of all the variables, we were able to derive till frequent items but we couldn't derive association rules this is a biggest challenge. So again we started writing a different algorithm but not using the dataset which contains all the variables, instead we formed a dataset called as transactions which contains all the items that were in a particular transaction Ex: Transaction 1 contains ['Bag of Organic Bananas',' Organic Hass Avocado',' Cucumber Kirby',' Organic Whole String Cheese',' Organic Celery Hearts',' Organic 4% Milk Fat Whole Milk Cottage Cheese',' Lightly Smoked Sardines in Olive Oil',' Bulgarian Yogurt'], this transactions was a resultant of the original dataset but the size of the data set is five thousand rows only. Then with this transactions we were able to derive frequent items and association rules.

With the challenges faced, we decide to build an algorithm using apriori but in a different way and we named it as Fast Apriori. So we tried running this algorithm using fifty thousand entries in a dataset which took like 48 seconds this is really fast. As it took less time to compute the association rules for fifty thousand, we thought of using one million rows and compute the algorithm again. But we didn't get succeeded, we ran into an error saying "memory out of space".

7. RESULT

After evaluation, the result concludes that the Apriori algorithm is the best of all the other algorithms we took into consideration. We tried optimizing the Apriori algorithm and landed on two different approaches which

we named Fast Apriori and Super-Fast Apriori. The Fast Apriori algorithm was able to handle a larger dataset than the basic apriori algorithm and computed the rules faster than any other algorithm. But the fast apriori algorithm couldn't handle larger datasets therefore we further optimized the algorithm and named it Super-Fast Apriori Algorithm as it can handle over 32 million rows of data in less than 7 minutes.

Though the FP growth algorithm performed as we expected it took more time than the Apriori algorithm therefore we conclude that the Apriori algorithm is more efficient than FP growth algorithm. In contradiction, we thought that ECLAT algorithm is faster than all the other algorithms but turned out to be the slowest algorithm.

7.1. Apriori Algorithm

Below image shows the basic apriori algorithm which works pretty fast on small data sets.

In this project we sampled twenty thousand rows from the original 32 million rows dataset

```
from mlxtend.frequent_patterns import apriori
frequent_itemsets = apriori(df_3,
                            min_support = 0.01,
                            max_len = 2,
                            use_colnames = True)
print(frequent_itemsets.head())

frequent_itemsets.shape

rules_1 = association_rules(frequent_itemsets,
                            metric="confidence", min_threshold=0)
```

7.2. FP Growth Algorithm

The below image shows the FP algorithm which also works only on small data sets.

While executing FP growth algorithm we faced time complexity as well as space complexity.

```
frequent_itemsets_fp=fpgrowth(df_3, min_support=0.01,
                                    snuse_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp,
           metric="confidence", min_threshold=0)
```

7.3. ECLAT Algorithm

The below image shows the basic ECLAT algorithm which works only on extremely small data set(has less than five thousand rows).

```
min_support = 1/100

# start from transactions containing at least 2 items
min_combination = 2

# up to maximum items per transaction
max_combination = 2

rule_indices, rule_supports = eclat.fit(min_support=min_support,
                                        min_combination=min_combination,
                                        max_combination=max_combination,
                                        separator=' & ',verbose=True)
```

7.4 Fast Apriori Algorithm

As Apriori is the fastest of all the above three algorithms we wanted to increase its efficiency and computational ability and the Fast apriori algorithm is the result of it.

```
def fast_apriori(df_FA):
  frequent = apriori(df_FA,min_support=0.01,
                     use_colnames=True)
  rules_eclat = association_rules(frequent,metric='support',
                    support_only=True,min_threshold=0.004)
  output = rules_eclat[['antecedents','consequents','support']]
  output['antecedents'].duplicated().sum()
  cols = ['antecedents','consequents']
  output[cols] = output[cols].applymap(lambda x: tuple(x))
  output = output.explode('antecedents').reset_index(drop=True).
                    explode('consequents').reset_index(drop=True)
  output.sort_values('support',ascending=False)
  print(output)
fast_apriori(df_fa)
```

7.5. Super-Fast Apriori Algorithm

The above algorithm still wasn't able to handle datasets that had more than fifty thousand rows. The super-Fast algorithm is the most efficient and fastest algorithm that can handle both and time and space complexity.

```
def association_rules(order_item, min_support):

    print("Starting order_item: {:22d}".format(len(order_item)))

    # Calculate item frequency and support
    item_stats             = freq(order_item).to_frame("freq")
    item_stats['support']  = item_stats['freq'] / order_count(order_item) * 100

    # Filter from order_item items below min support
    qualifying_items       = item_stats[item_stats['support'] >= min_support].index
    order_item             = order_item[order_item.isin(qualifying_items)]

    print("Items with support >= {}: {:15d}".format(min_support, len(qualifying_items)))
    print("Remaining order_item: {:21d}".format(len(order_item)))

    # Filter from order_item orders with less than 2 items
    order_size             = freq(order_item.index)
    qualifying_orders      = order_size[order_size >= 2].index
    order_item             = order_item[order_item.index.isin(qualifying_orders)]

    print("Remaining orders with 2+ items: {:11d}".format(len(qualifying_orders)))
    print("Remaining order_item: {:21d}".format(len(order_item)))

    # Recalculate item frequency and support
    item_stats             = freq(order_item).to_frame("freq")
    item_stats['support']  = item_stats['freq'] / order_count(order_item) * 100

    # Get item pairs generator
    item_pair_gen          = get_item_pairs(order_item)

    # Calculate item pair frequency and support
    item_pairs              = freq(item_pair_gen).to_frame("freqAB")
    item_pairs['supportAB'] = item_pairs['freqAB'] / len(qualifying_orders) * 100

    print("Item pairs: {:31d}".format(len(item_pairs)))
```

Fortunately, all the algorithms gave out same set of association rules.

itemA	itemB
Organic Strawberries	Bag of Organic Bananas
Banana	Limes
Banana	Cucumber Kirby
Organic Hass Avocado	Organic Strawberries
Banana	Organic Fuji Apple
Organic Hass Avocado	Bag of Organic Bananas
Bag of Organic Bananas	Organic Raspberries
Banana	Large Lemon
Banana	Strawberries
Bag of Organic Bananas	Organic Baby Spinach
Banana	Organic Avocado
Banana	Organic Baby Spinach
Bag of Organic Bananas	Organic Hass Avocado
Banana	Organic Strawberries
Bag of Organic Bananas	Organic Strawberries

The above is the association rules generated by all the algorithms. Item A represents Antecedent, and Item B represents Consequent

8. CONCLUSION

In conclusion for categorical output data:

- When we deal with datasets which have around 30,000 transactions, Apriori is a tried and tested method whose implementation is straight forward.
- When we deal with slightly bigger datasets fast apriori algorithm is preferred. This algorithm simply pivots the table decreasing the number of scans required for the algorithm to calculate support for each item.
- When we deal with large datasets where the number of transactions is in the terms of millions Super fast apriori algorithm is the best algorithm. This is not an algorithm which is inbuilt into python.

All these observations have been made in the assumption that the data is a transactional dataset with categorical output. If the application has huge amounts of data which is categorical in nature, then Super fast apriori is the best but we must write the code on our own which will raise different problems such as optimization and debugging. The apriori algorithm which we have used is already optimized by machine learning engineers and peer reviewed by coders all over the world.

9. FUTURE WORK

Future work may include the following:

1. Adding analysis such as K nearest neighbors to study customer behavior.
2. Including time analysis to study customer trends over time (hourly, daily, monthly and based on season). Customer trends along with product trends would vary based on season as each season has different demands in terms of food.
3. We can also extend the analysis to include different types of data. Retail data is an industry which would be highly insightful.
4. Predictions based on the rules generated.
5. Analyse prior data of customers to understand patterns of customers buying products repeatedly.

10. REFERENCES

1. https://medium.com/geekculture/the-k-modes-as-clustering-algorithm-for-categorical-data-type-bcde8f95efd7
2. https://www.kaggle.com/code/datatheque/association-rules-mining-market-basket-analysis
3. https://kalpanileo1996.medium.com/prefix-span-algorithm-with-pyspark-483ab3494373
4. https://hands-on.cloud/implementation-of-eclat-algorithm-using-python/
5. https://pbpython.com/market-basket-analysis.html
6. https://towardsdatascience.com/understand-and-build-fp-growth-algorithm-in-python-d8b989bab342#:~:text=What%20is%20FP%2DGrowth,or%20associations%20from%20data%20sets.
7. ://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html
8. http://hanj.cs.illinois.edu/pdf/span01.pdf
9. https://pypi.org/project/prefixspan/
10. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.fpm.PrefixSpan
11. https://www.geeksforgeeks.org/implementing-apriori-algorithm-in-python/
12. http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/
13. https://intellipaat.com/blog/data-science-apriori-algorithm/
14. https://moam.info/spade-an-efficient-algorithm-for-mining-frequent-_599edae21723dd0c4031e814.html
15. https://towardsdatascience.com/the-eclat-algorithm-8ae3276d2d17
16. https://github.com/msimao96/Eclat_model_on_Market_Basket_Optimisation/blob/main/Eclat_model_on_Market_Basket_Optimisation.R