Market Basket Analysis

Introduction

Market Basket Analysis is a data mining technique used to understand customer purchasing patterns and to uncover meaningful correlations between different entities according to their co-occurrence in the data set, and to extract significant knowledge about the unexpected trends and hidden patterns.

It is conceptually an efficient primal stone in marketing and sales research that may be used to handle strategic business decisions as its about understanding the consumer's behaviour patterns on his purchases. This overall technique led to find the affinities amongst the items in consumer's shopping baskets and the idea of it being able to affect homogeneous customers. Hence this technique has obtained an added value to recommendation systems. It opens a new opportunity for them to improve customer support thereby, decreasing their customer acquisition costs and increasing the customer retention rate.

In order to imply this Market Basket Analysis approach in various domains more organizations are discovering ways to gain useful insights into associations and hidden relationships. As industry leaders continue to explore the technique's value, a predictive version of market basket analysis is making in-roads across many sectors in an effort to identify sequential purchases. Yet lack of knowledge on individual's purchase, co-occurrences within their purchases are slightly neglected and it may results in the omit of mindsets behind their every purchase or transaction.

Abstract

Formal Definition of the Problem

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n binary attributes called items.

Let $D = t_1, t_2, \ldots, t_m$ be a set of transactions called the database.

Each element in D has a unique ID called Transaction ID and contains a subset of the elements of I

A rule is defined to be an implication relation of the form $X \implies Y$, where $X, Y \subseteq I$ and $X \cap Y = \emptyset$.

Measures

The following measures are used to get the statistical significance of an itemset.

Support

Support is a measure of how frequently the itemset appears in the dataset. The support of X which is restricted to T, where T is defined as the proportion of transactions in the dataset.

$$\text{Supp}(X) = \frac{\text{frequency}(X \subseteq T)}{\text{frequency}(T)}$$

Confidence

Confidence is a measure of how often the **rule** $X \implies Y$ has been found to be true.

$$\text{Conf}(X \implies Y) = \frac{\text{Supp}(X \cup Y)}{\text{Supp}(X)}$$

Lift

The problem with **confidence** is, it only considers the $\text{Supp}(X)$, not $\text{Supp}(Y)$. So we define another measure that also considers the popularity of Y.

$$\text{Lift}(X \implies Y) = \frac{\text{Supp}(X \cup Y)}{\text{Supp}(X) \times \text{Supp}(Y)}$$

Bottle Neck

But there is a problem here, since $X \subset I$ we might need to consider all the possible subset of I, and if I has n elements then, there are 2^n possible subset. Even if I has like 100 items, 2^{100} is more the than the number of atoms in the universe. If $n = 200$, it would take more than the **life of the universe** to compute all the implication relations among all of them. Since we can't compute the measures of each and every Subset, so we need to **make some tradeoffs** and cut down the number of subsets we need to compute. This is where the **cleverness** of Apirori Algorithm enters, and will be explained in the section below.

Algorithms

Apriori Algorithm

Although there are many algorithms out there to do market basket analysis. They are either not as efficient in terms of time or space as Apriori Algorithm. So we are going to use Apriori Algorithm to find the support of **some elements of the power set,** $\mathcal{P}(I)$. The following is pseudocode of the Apriori Algorithm.

```
1. Apriori(T, ε)
2.  L(1) ← {large 1 - itemsets}
3.  k ← 2

4.  while L(k-1) is not empty
5.      C(k) ← {c = Union(a, {b}) : a in L(k-1) and b not in a, {s ⊆ c : |s| = k - 1} ⊆ L(k-1)}

6.      for transactions t in T
7.          D(t) ← {c in C(k) : c ⊆ t}
```

```
8.          for candidates c in D(t)
9.              count[c] ← count[c] + 1

10.     L(k) ← {c in C(k) : count[c] ≥ ε}
11.     k ← k + 1

12. return Union(L(k))
```

k-itemset

A k-itemset is a set that contains k items

L_k

Set of large k-itemsets with minimum support.
Each member of this set has two fields:

1. itemset
2. support count

C_k

Set of candidate k-itemsets contains potentially large itemsets.

Each member of this set has two fields:

1. itemset
2. support count

Example

To understand the cleverness of the algorithm, we need to work out an example.

Consider the following transaction table:

TID	Milk	Bread	Butter	Beer	Diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

Let the minimum support count, "$\varepsilon = 1$".

According to line 2, we need to initiate L_1 to be large 1-itemset.

1-itemset	Support Count
Milk	2
Bread	3
Butter	2
Beer	1
Diapers	1

And in line 3, $k \leftarrow 2$
To find L_2

L_2:

1. L_1 is not empty, execute while loop in line 4
2. Generate C_2, we know from the line 5, C_2 is the subsets of all set that contains 2 elements of L_1, and those two elements are not equal.

C_2 = {{Milk, Bread}, {Milk, Butter}, {Milk, Beer}, {Milk, Diapers}, {Bread, Butter}, {Bread, Beer}, {Bread, Diapers}, {Butter, Beer}, {Butter, Diapers}, {Beer, Diapers}}

4. Now run the **For Loop**, to get the support count of each element.
5. If support count of each element in C_2.

C_2	Support Count
{Milk, Bread}	2
{Milk, Butter}	1
{Milk, Beer}	0
{Milk, Diapers}	0
{Bread, Butter}	1
{Bread, Beer}	0
{Bread, Diapers}	0

C_2	**Support Count**
{Butter, Beer}	0
{Butter, Diapers}	0
{Beer, Diapers}	1

6. Now, if the support count of each element in $C_2 \geq$ Minimum Support Count, i.e. $\varepsilon = 1$, add that element to L_2

L_2	**Support Count**
{Milk, Bread}	2
{Milk, Butter}	1
{Bread, Butter}	1
{Beer, Diapers}	1

8. Then, increment k, loop again.

It might seem like all the 5 elements are in L_2, no useless item was removed, but as we run the algorithm more times, you'll see!

L_3:
Running loop again we get

C_3	**Support Count**
{Milk, Bread, Butter}	1
{Milk, Bread, Beer}	0
{Milk, Bread, Diapers}	0
{Milk, Butter, Beer}	0
{Milk, Butter, Diapers}	0
{Milk, Beer, Diapers}	0
{Bread, Butter, Beer}	0
{Bread, Butter, Diapers}	0
{Bread, Beer, Diapers}	0
{Butter, Beer, Diapers}	0

So L_3 will be

L_3	**Support Count**
{Milk, Bread, Butter}	1

L_4:

To generate, L_4,

L_3 is not empty so run the while loop, but to generate C_4 we need need to choose 4 elements from {Milk, Bread, Butter}, but there are only 3 elements in there, so C_4 is empty, thus L_4 is empty.

While loop terminates, since L_4 is empty

> The cleverness of the apriori algorithm is,
> Instead of checking all the implication relations of subsets in the powerset $\mathcal{P}(I)$ of itemset I for support and confidence or lift, it removes all the elements of powerset that doesn't have the minimum support.
> Reason is, let minimum support be $\varepsilon > 0$.
>
> Suppose,
>
> $$\text{Supp}(X) < \varepsilon$$
>
> then the confidence,
>
> $$\text{Conf}(X \implies Y) = \frac{\text{Supp}(X \cup Y)}{\text{Supp}(X)}$$
>
> $$\text{Conf}(X \implies Y) = \frac{\text{Supp}(X \text{ and } Y)}{\text{Supp}(X)}$$
>
> and
>
> $$\text{Supp}(X \text{ and } Y) \leq \text{Supp}(X) < \varepsilon$$
>
> So,
>
> $$\text{Conf}(X \implies Y) < \varepsilon$$
>
> Similarly if our measure is **lift**. So, we don't need to calculate **confidence or lift** for all the subsets that contain X, this is how apirori algorithm cut down

> the number of subsets.

Generating Association Rules

To generation strong association rules from itemsets where strong association rules satisfy both minimum support and minimum confidence or minimum lift, do as follows:

For each frequent itemset l of L_{k-1}, generate all non-empty subsets of l.

For every non-empty subset s of l, output the rule "$s \implies (l-s)$" if $\text{measure}(s \implies (l-s)) \geq$ minimum measure, where measures are either confidence or lift.

For the above example.

The frequent item set are L_{k-1} which is L_3,

L_3 has only 1 item, and let confidence be our measure.

so , $l =$ {Milk, Bread, Butter}

let $p = s - l$

$$\text{Conf}(s \implies p) = \frac{\text{Supp}(s \cup p)}{\text{Supp}(s)}$$

First Let $s =$ {Milk}, so $p =$ {Bread, Butter}

s	p	Confidence
{Milk}	{Bread, Butter}	0.5
{Bread}	{Milk, Butter}	0.333..
{Butter}	{Milk, Bread}	0.5
{Milk, Bread}	{Butter}	0.5
{Milk, Butter}	{Bread}	1
{Bread, Butter}	{Milk}	1

Let minimum confidence = 1.

So, our strong association rules are

1. {Milk, Butter} $\implies$ {Bread}
2. {Bread, Butter} $\implies$ {Milk}

This not possible in real world application, because if minimum confidence is 1, then we saying we are 100% confident about this happening. Since it is a small dataset, we get these wired results. So, we need a big dataset. We went around the internet and found the biggest dataset we could find on the internet. It is so big, it took my computer 2-3 minutes to loadup data into RAM.

Data Collection

Data was collected from Instacart, an American company that operates a grocery delivery and pick up service in the United States and Canada. This anonymized dataset contains a sample of over 3 million grocery orders.

Source: https://www.kaggle.com/c/instacart-market-basket-analysis/data

---

Data Description

The dataset contains relational set of files describing customers' orders over time. For each user, 4 to 100 orders are provided with the sequence of products purchased in each order. The data of the order's week and hour of the day as well as a relative measure of time between orders is provided.

Data

```
os.listdir("../data")
```

The following are the files in the dataset.

```
['aisles.csv',
 'departments.csv',
 'orders.csv',
 'order_products__prior.csv',
 'order_products__train.csv',
 'products.csv',
 'sample_submission.csv']
```

```
aisles = pd.read_csv("../data/aisles.csv")
departments = pd.read_csv("../data/departments.csv")
orders = pd.read_csv("../data/orders.csv")
prior = pd.read_csv("../data/order_products__prior.csv")
```

```
train = pd.read_csv("../data/order_products__train.csv")
products = pd.read_csv("../data/products.csv")
```

alisles.csv

It contains all the alisles in the instacart website. although we don't use it in our implementation, some else might use to get some extra knowledge.\

```
len(aisles.aisle.unique())
```

```
134
```

```
aisles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134 entries, 0 to 133
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   aisle_id  134 non-null    int64 
 1   aisle     134 non-null    object
dtypes: int64(1), object(1)
memory usage: 2.2+ KB
```

```
aisles.aisle.unique()
```

```
array(['prepared soups salads', 'specialty cheeses', 'energy', granola bars', 'instant foods', 'marinades meat preparation',
'other', 'packaged meat', 'bakery desserts', 'pasta sauce', 'kitchen supplies', 'cold flu allergy', 'fresh pasta', 'prepared
meals', 'tofu meat alternatives', 'packaged seafood', 'fresh herbs', 'baking ingredients', 'bulk dried fruits vegetables',
'oils vinegars', 'oral hygiene', 'packaged cheese', 'hair care', 'popcorn jerky', 'fresh fruits', 'soap', 'coffee', 'beers
coolers','red wines', 'honeys syrups nectars', 'latino foods', 'refrigerated', 'packaged produce', 'kosher foods', 'frozen meat
seafood', 'poultry counter', 'butter', 'ice cream ice', 'frozen meals', 'seafood counter', 'dog food care', 'cat food care',
'frozen vegan vegetarian', 'buns rolls', 'eye ear care', 'candy chocolate', 'mint gum', 'vitamins supplements', 'breakfast bars
pastries', 'packaged poultry', 'fruit vegetable snacks', 'preserved dips spreads', 'frozen breakfast', 'cream', 'paper goods',
'shave needs', 'diapers wipes', 'granola', 'frozen breads doughs', 'canned meals beans', 'trash bags liners', 'cookies cakes',
'white wines', 'grains rice dried goods', 'energy sports drinks', 'protein meal replacements', 'asian foods', 'fresh dips
tapenades', 'bulk grains rice dried goods', 'soup broth bouillon', 'digestion', 'refrigerated pudding desserts', 'condiments',
'facial care', 'dish detergents', 'laundry', 'indian foods', 'soft drinks', 'crackers', 'frozen pizza', 'deodorants', 'canned
jarred vegetables', 'baby accessories', 'fresh vegetables', 'milk', 'food storage', 'eggs', 'more household', 'spreads', 'salad
dressing toppings', 'cocoa drink mixes', 'soy lactosefree', 'baby food formula', 'breakfast bakery', 'tea', 'canned meat
seafood', 'lunch meat', 'baking supplies decor', 'juice nectars', 'canned fruit applesauce', 'missing', 'air fresheners
candles', 'baby bath body care', 'ice cream toppings', 'spices seasonings', 'doughs gelatins bake mixes', 'hot dogs bacon
sausage', 'chips pretzels', 'other creams cheeses', 'skin care', 'pickled goods olives', 'plates bowls cups flatware', 'bread',
'frozen juice', 'cleaning products', 'water seltzer sparkling water', 'frozen produce', 'nuts seeds dried fruit', 'first aid',
'frozen dessert', 'yogurt', 'cereal', 'meat counter', 'packaged vegetables fruits', 'spirits', 'trail mix snack mix', 'feminine
care', 'body lotions soap', 'tortillas flat bread', 'frozen appetizers sides', 'hot cereal pancake mixes', 'dry pasta',
'beauty', 'muscles joints pain relief', 'specialty wines champagnes'],
    dtype=object)
```

It contains details about, 134 aisles. and they are as seen above.

departments.csv

It contains all the departments, although we don't use it in our implementation, some else might use to get some extra knowledge.

```
len(departments.department.unique())
```

```
21
```

The dataset contains contains 21 departments.

```
departments.department.unique()
```

```
array(['frozen', 'other', 'bakery', 'produce', 'alcohol', 'international','beverages', 'pets', 'dry goods pasta', 'bulk',
'personal care', 'meat seafood', 'pantry', 'breakfast', 'canned goods', 'dairy eggs', 'household', 'babies', 'snacks', 'deli',
'missing'],
dtype=object)
```

orders.csv

It contains all the orders

```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3421083 entries, 0 to 3421082
Data columns (total 7 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   order_id                int64  
 1   user_id                 int64  
 2   eval_set                object 
 3   order_number            int64  
 4   order_dow               int64  
 5   order_hour_of_day       int64  
 6   days_since_prior_order  float64
dtypes: float64(1), int64(5), object(1)
memory usage: 182.7+ MB
```

```
len(orders.order_id.unique())
```

```
3421083
```

```
len(orders.user_id.unique())
```

```
206209
```

order*products*.csv

This * represents prior and train.

These files give information about which products were ordered and in which order they were added in the cart. It also tells us that if the product was reordered or not.

```
prior.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32434489 entries, 0 to 32434488
Data columns (total 4 columns):
 #   Column             Dtype
---  ------             -----
 0   order_id           int64
 1   product_id         int64
 2   add_to_cart_order  int64
 3   reordered          int64
dtypes: int64(4)
memory usage: 989.8 MB
```

```
len(prior.order_id.unique())
```

```
3214874
```

```
len(prior.product_id.unique())
```

```
49677
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1384617 entries, 0 to 1384616
Data columns (total 4 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   order_id           1384617 non-null  int64
 1   product_id         1384617 non-null  int64
 2   add_to_cart_order  1384617 non-null  int64
```

```
 3   reordered          1384617 non-null  int64
dtypes: int64(4)
memory usage: 42.3 MB
```

```
len(train.order_id.unique())
```

```
131209
```

```
len(train.product_id.unique())
```

```
39123
```

products.csv

This file contains the list of total 49688 products and their aisle as well as department. The number of products in different aisles and different departments are different.

```
len(products.product_name.unique())
```

```
49688
```

```
len(products.aisle_id.unique())
```

```
134
```

```
len(products.department_id.unique())
```

```
21
```

Code and Results

```
# Import Packages

# Pandas for rading CSV files
import pandas as pd

# Numpy for arrays, array is a much more efficient data structure than builtin python list for our purpose
import numpy as np


# mlxtend is a machine learing library that contations alot of ml tools
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

```
# Import DataSets
order = pd.read_csv("../data/orders.csv")
product = pd.read_csv("../data/products.csv")
prior = pd.read_csv("../data/order_products__prior.csv")
train = pd.read_csv("../data/order_products__train.csv")

# Now we are merging prior and train datasets to get the complete order dataset.
train = train.append(prior,ignore_index = True)

train['reordered'] = 1
```

```
productCount = train.groupby("product_id",as_index = False)["order_id"].count()
```

```
# Top 100 most frequently purchased products
freq_product = 100

productCount = productCount.sort_values("order_id",ascending = False)
topProduct = productCount.iloc[0:freq_product,:]
topProduct = topProduct.merge(product,on = "product_id")
productId= topProduct.loc[:,["product_id"]]
```

```
# Here order_id is the count, so we need to sort the data frame with repect to order_id

df = train[0:0]
for i in range(0,99):
    pId = productId.iloc[i]['product_id']
    stDf = train[train.product_id == pId ]
    df = df.append(stDf,ignore_index = False)
```

```
# Here we are defining the basket to analysis
basket = df.groupby(['order_id', 'product_id'])['reordered'].sum().unstack().reset_index().fillna(0).set_index('order_id')
```

```
# This function is used to convert database into a boolean database as seen below

def encode(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
```

```
# here we encoding the basket df with our encode funtion
basket_sets = basket.applymap(encode)
```

```
# This is the size of the basket_sets
basket_sets.size
```

```
241667217
```

That is **over 240 million!**

```
# Get frequent itemsets using apriori
frequent_itemsets = apriori(basket_sets, min_support=0.001, use_colnames=True, low_memory = True)
```

```
# Print frequnt items sets
frequent_itemsets
```

	support	itemsets
0	0.015279	(196)
1	0.016088	(3957)
2	0.015144	(4210)
3	0.031514	(4605)
4	0.015439	(4799)
...	...	...
2518	0.001043	(40706, 47626, 47766)
2519	0.001013	(47626, 47766, 45007)
2520	0.001462	(47626, 49683, 47766)
2521	0.001439	(13176, 47209, 21137, 21903)
2522	0.001662	(13176, 47209, 21137, 27966)

2523 rows × 2 columns

```
# Now we generate rules:

rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Print rules
rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(5876)	(3957)	0.037381	0.016088	0.001157	0.030948	1.923701	0.000555	1.015335
1	(3957)	(5876)	0.016088	0.037381	0.001157	0.071911	1.923701	0.000555	1.037205

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(13176)	(3957)	0.161785	0.016088	0.003762	0.023252	1.445357	0.001159	1.007335
3	(3957)	(13176)	0.016088	0.161785	0.003762	0.233837	1.445357	0.001159	1.094043
4	(21137)	(3957)	0.112891	0.016088	0.002259	0.020013	1.243979	0.000443	1.004005
...	...	...	...	...	...	...	...	...	...
6417	(21137, 27966)	(13176, 47209)	0.014556	0.026530	0.001662	0.114147	4.302641	0.001275	1.098908
6418	(13176)	(47209, 21137, 27966)	0.161785	0.003389	0.001662	0.010270	3.030382	0.001113	1.006953
6419	(47209)	(13176, 21137, 27966)	0.090483	0.005011	0.001662	0.018363	3.664351	0.001208	1.013602
6420	(21137)	(13176, 47209, 27966)	0.112891	0.004891	0.001662	0.014718	3.009076	0.001109	1.009974
6421	(27966)	(13176, 47209, 21137)	0.058418	0.006463	0.001662	0.028443	4.401036	0.001284	1.022623

6422 rows × 9 columns

Conclusion

Since we have more than 6400 association rules, we want to filter out those unimportant rules, so we choose minimum confidence as 0.4 and minimum lift as 2.

```
# We can filter the dataframe using standard pandas code.

rules[ (rules['lift'] >= 2) & (rules['confidence'] >= 0.4) ]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
3319	(4605, 16797)	(24852)	0.002171	0.201259	0.001021	0.470377	2.337169	0.000584	1.508131
3403	(4920, 28204)	(24852)	0.002436	0.201259	0.001143	0.469055	2.330598	0.000652	1.504375
3408	(4920, 45066)	(24852)	0.002709	0.201259	0.001174	0.433238	2.152632	0.000628	1.409304
3421	(4920, 47766)	(24852)	0.003384	0.201259	0.001409	0.416414	2.069043	0.000728	1.368678
3426	(4920, 49683)	(24852)	0.002387	0.201259	0.001051	0.440268	2.187563	0.000570	1.427005
3446	(5876, 8277)	(13176)	0.002505	0.161785	0.001007	0.402060	2.485155	0.000602	1.401839
3482	(5876, 27966)	(13176)	0.004337	0.161785	0.001762	0.406309	2.511417	0.001061	1.411871
3572	(21137, 8174)	(13176)	0.003079	0.161785	0.001345	0.436876	2.700356	0.000847	1.488510
3583	(27966, 8174)	(13176)	0.002255	0.161785	0.001119	0.496094	3.066386	0.000754	1.663437
3590	(47209, 8174)	(13176)	0.003491	0.161785	0.001628	0.466385	2.882751	0.001064	1.570824
3614	(8277, 27966)	(13176)	0.004447	0.161785	0.001808	0.406541	2.512850	0.001088	1.412423
3620	(47209, 8277)	(13176)	0.006522	0.161785	0.002798	0.429020	2.651796	0.001743	1.468029
3679	(8424, 47766)	(24852)	0.003468	0.201259	0.001537	0.443303	2.202643	0.000839	1.434784
3806	(47209, 16759)	(13176)	0.002646	0.161785	0.001132	0.427930	2.645058	0.000704	1.465232
3852	(19057, 22935)	(13176)	0.002632	0.161785	0.001055	0.400996	2.478579	0.000630	1.399349
3858	(19057, 27966)	(13176)	0.003660	0.161785	0.001650	0.450811	2.786489	0.001058	1.526279
3864	(19057, 30391)	(13176)	0.002650	0.161785	0.001102	0.415611	2.568910	0.000673	1.434343
3870	(19057, 47209)	(13176)	0.005673	0.161785	0.002476	0.436421	2.697544	0.001558	1.487309
4284	(22825, 27966)	(13176)	0.002929	0.161785	0.001262	0.430709	2.662235	0.000788	1.472384
4290	(47209, 22825)	(13176)	0.003950	0.161785	0.001778	0.450218	2.782820	0.001139	1.524631
4452	(27966, 30391)	(13176)	0.003762	0.161785	0.001534	0.407819	2.520749	0.000926	1.415471
4475	(39928, 27966)	(13176)	0.003123	0.161785	0.001306	0.418153	2.584627	0.000801	1.440612
4488	(27966, 41950)	(13176)	0.002478	0.161785	0.001009	0.406942	2.515331	0.000608	1.413379
4494	(27966, 44359)	(13176)	0.002686	0.161785	0.001082	0.402776	2.489577	0.000647	1.403518
4506	(47209, 27966)	(13176)	0.010984	0.161785	0.004891	0.445323	2.752565	0.003114	1.511177
4560	(47209, 35951)	(13176)	0.003818	0.161785	0.001528	0.400322	2.474411	0.000911	1.397775
4566	(47209, 37646)	(13176)	0.004976	0.161785	0.002000	0.402025	2.484940	0.001195	1.401757
4582	(39928, 47209)	(13176)	0.004016	0.161785	0.001764	0.439208	2.714771	0.001114	1.494700
4682	(28204, 16797)	(24852)	0.003187	0.201259	0.001562	0.490231	2.435818	0.000921	1.566869
4693	(45066, 16797)	(24852)	0.003243	0.201259	0.001590	0.490399	2.436652	0.000938	1.567385
4703	(47626, 16797)	(24852)	0.004942	0.201259	0.002093	0.423574	2.104618	0.001099	1.385678
4710	(16797, 47766)	(24852)	0.005457	0.201259	0.002276	0.417161	2.072752	0.001178	1.370431
4715	(49683, 16797)	(24852)	0.003735	0.201259	0.001662	0.444834	2.210254	0.000910	1.438743
4785	(28842, 20114)	(26209)	0.002472	0.060080	0.001042	0.421611	7.017504	0.000894	1.625065
4792	(20114, 31717)	(26209)	0.002523	0.060080	0.001073	0.425463	7.081618	0.000922	1.635960

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
4798	(20114, 47626)	(26209)	0.002734	0.060080	0.001127	0.412108	6.859342	0.000962	1.598799
5150	(21137, 49683)	(24852)	0.005023	0.201259	0.002062	0.410570	2.040004	0.001051	1.355107
6049	(49683, 27845)	(24852)	0.002620	0.201259	0.001103	0.421044	2.092048	0.000576	1.379623
6067	(47626, 28204)	(24852)	0.003530	0.201259	0.001505	0.426433	2.118823	0.000795	1.392585
6073	(28204, 47766)	(24852)	0.003991	0.201259	0.001811	0.453911	2.255352	0.001008	1.462656
6079	(49683, 28204)	(24852)	0.002984	0.201259	0.001381	0.462859	2.299811	0.000780	1.487021
6126	(34969, 49683)	(24852)	0.002818	0.201259	0.001135	0.402675	2.000775	0.000568	1.337195
6173	(43961, 47766)	(24852)	0.002869	0.201259	0.001159	0.403769	2.006213	0.000581	1.339650
6190	(45066, 47626)	(24852)	0.004108	0.201259	0.001679	0.408697	2.030695	0.000852	1.350813
6197	(45066, 47766)	(24852)	0.004756	0.201259	0.002065	0.434318	2.158002	0.001108	1.411996
6202	(45066, 49683)	(24852)	0.002723	0.201259	0.001225	0.449759	2.234724	0.000677	1.451620
6251	(49683, 47766)	(24852)	0.006543	0.201259	0.002769	0.423214	2.102829	0.001452	1.384813
6293	(28842, 47626)	(26209)	0.002886	0.060080	0.001159	0.401561	6.683790	0.000986	1.570621
6300	(28842, 47766)	(26209)	0.003199	0.060080	0.001286	0.402152	6.693615	0.001094	1.572171
6312	(47626, 31717)	(26209)	0.003462	0.060080	0.001418	0.409586	6.817353	0.001210	1.591967
6397	(47209, 21137, 21903)	(13176)	0.003469	0.161785	0.001439	0.414738	2.563516	0.000877	1.432205
6411	(47209, 21137, 27966)	(13176)	0.003389	0.161785	0.001662	0.490270	3.030382	0.001113	1.644428

Future Remarks

Improving of Apriori Algorithm

Although Apriori is a significant break through historically, it suffers from a number of inefficiencies and trade-offs. Candidate generation generates larger number of subsets, so the algorithm scans the database too many times, so both the time and space complexity of this algorithms are very high, $O(2^{|D|})$, where $|D|$ is the total number of items present in the database. Although we many ways to solves, none were hopeful. All the method we tried, ended up at the crazy town of *np completeness*.

Calculating support of L_1 is easy, you can just calculate the all the support values and compare it to the minimum support. Now, Let's say, $A, B \in L_1$. We know that $\text{Supp}(A)$ is the just the probability of A happening in the database. Now to find the $\text{Supp}(A \cup B)$.

$$\begin{aligned}\text{Supp}(A \cup B) &\equiv P(A \cup B) \\ &= P(A) + P(B) - P(A \cap B)\end{aligned}$$

So, to get support value without scanning database again, we need to know $P(A \cap B)$.

Now to know $P(A \cap B)$, we need to know $P(A \cup B)$, Now to know $P(A \cap B)$ we need to know $P(A \cup B)$, unless we have some other equation, we can't get solutions of two variable from one equation. We don't have any other information. So the only way to get $P(A \cup B)$ is to scan database again. There are some clever work around for some specific cases, but there are none for general case.

If you get a thought like "Yeah, why don't we just calculate all the unions in just one scan of the dataset". Sure, you can and you will get a **constant time** algorithm, but the memory it would take to store all that information is astronomical. It would be unfair to this number to call it astronomical, because astronomical would be so tiny compared to this number with just small input sizes. We tried alot to improve this algorithm, we always ended up one of these extremes.

Unless someone with a giga-brain came up a super genius proof to show $p = np$, I don't think there is any general solution!!

Instacart Dataset

I think there is much more potential here, if we have some beefier hardware, we would be able to analysis this data more effectively, and may be we would be able to get some more useful data. We tried to implement our own apriori algorithm, memory management is quite complicated and out of the scope of this project, so we decided to use some standard machine learning library instead. Even with that low memory algorithm, it took like 8GB memory and 10 minutes of time. We can improve in this regard.