Shopper Behavior Exploration and Market Basket Analysis using Apache Spark

Rahul Bhasin
Apr 12 · 10 min read

In this blog, I will talk about my Shopper Behavior Exploration on a real Instacart Dataset of 3MM+ records. I will discuss how you can quickly run your market basket analysis using Apache Spark ML `FP-growth` algorithm on Databricks.

Market Basket Analysis is a technique used by large retailers to discover associations between their items. It works by looking for combinations of items that are bought together frequently, providing information to understand the purchase behavior. Association Rules Mining is one of the very important concepts of machine learning being used in Market Basket Analysis. Let's understand what it is.

Association Rule Mining

Association rules can be thought of as an 'If-Then' relationship. In simple words, if a customer has an item **A** in a basket, the Association Rules help in identifying the other item **B** that the customer is most likely to buy. For

instance, if a customer buys bread then it is likely that it will buy butter as well.

Supermarkets with thousands of different products in their store can boost their revenues and profits by accessing these untapped opportunities for identifying the relationship between the items purchased. They can do so by placing items frequently bought together in the same aisle, optimizing their catalog design, cross-selling items with collective discounts on their websites and analyzing customer behavior analysis, etc.

Two important metrics used in defining association rules are **Support** and **Confidence**. Every association rule should have minimum Confidence and minimum Support at the same time and these are usually user-specified.

Now, let's look at what are Support, Confidence, and Lift metrics with 2 items (suppose) ***X*** and ***Y***.

Support:- Support is defined as the frequency with which items ***X*** and ***Y*** are purchased together over the total number of transactions. This metric tells us how frequent an itemset is in all the transactions.

Confidence:- Confidence is the frequency with which **X** and ***Y*** are purchased together over the frequency with which ***X*** is purchased alone.

Lift:- Lift is defined as the Support over the Support for ***X*** times the Support for *Y*. Think of lift as the rise in the probability of having ***Y*** on the cart with the knowledge of ***X*** being present over the probability of having ***Y*** on the cart without any knowledge about the presence of ***X***.

I would be using these metrics later while implementing `FP-growth` algorithm.

. . .

Dataset

In this blog, the dataset I am using is **Instacart's real dataset** which was released on Kaggle in 2017.

This is an anonymized dataset containing a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, there are about 4 to 100 orders, with the sequence of products purchased in each order.

The dataset can be downloaded from here.

Platform used

I would be using Apache Spark for the implementation of this project and using **PySpark**, **SparkSql** and later **Scala** programming languages for coding.

Note that you can try Apache Spark on the Databricks cloud for free from this link.

Ok, enough for the theory, let's get to the code.

. . .

Importing all the available files into the Spark DataFrame and creating temporary tables

The dataset comes with six different .csv files that can be merged together to perform analyses. To begin with, I imported all the 6 files in the Spark environment and created temporary tables to run Spark SQL queries on them.

```
aisles = spark.read.csv("/FileStore/tables/aisles.csv", header=True, inferSchema=True)
departments = spark.read.csv("/FileStore/tables/departments.csv", header=True, inferSche
order_products_prior = spark.read.csv("/FileStore/tables/order_products__prior.csv", hea
order_products_train = spark.read.csv("/FileStore/tables/order_products__train.csv", hea
orders = spark.read.csv("/FileStore/tables/orders.csv", header=True, inferSchema=True)
products = spark.read.csv("/FileStore/tables/products.csv", header=True, inferSchema=Tru

aisles.createOrReplaceTempView("aisles")
departments.createOrReplaceTempView("departments")
order_products_prior.createOrReplaceTempView("order_products_prior")
order_products_train.createOrReplaceTempView("order_products_train")
orders.createOrReplaceTempView("orders")
products.createOrReplaceTempView("products")
```

importing_files.py hosted with ❤ by GitHub view raw

Files import into Spark DataFrame

Let's take a look at the **top 5 rows** of each of the imported file along with their **Data Dictionary**:

> `orders` (3.4m rows, 206k users):

- `order_id` : order identifier
- `user_id` : customer identifier
- `eval_set` : which evaluation set this order belongs to
- `order_number` : the order sequence number for this user (1 = first, n = nth)
- `order_dow` : the day of the week the order was placed on
- `order_hour_of_day` : the hour of the day the order was placed on
- `days_since_prior` : days since the last order, capped at 30 (with NAs for `order_number` = 1)

```
orders.show(n=5)
```


> `products` (50k rows):

- `product_id` : product identifier
- `product_name` : name of the product
- `aisle_id` : foreign key
- `department_id` : foreign key

```
products.show(n=5)
```


> `aisles` (134 rows):

- `aisle_id` : aisle identifier
- `aisle` : the name of the aisle

```
aisles.show(n=5)
```


> `deptartments` (21 rows):

- `department_id` : department identifier
- `department` : the name of the department

```
departments.show(n=5)
```


> `order_products_train` (131K+ rows):

Training data supplied to participants of Kaggle

```
order_products_train.show(n=5)
```


> `order_products_prior` (32MM+ rows):

Orders data prior to that user's most recent order

```
order_products_prior.show(n=5)
```


Now that we have created DataFrames and taken a glimpse at the top 5 rows of each file, let's proceed with **EDA using Spark SQL** to find insights and patterns from the Instacart dataset.

. . .

Exploratory Data Analysis

What time of day do customers purchase?

```
df = sqlContext.sql("select count(order_id) as total_orders,
order_hour_of_day as hour
from orders
group by order_hour_of_day
order by order_hour_of_day")
df.show()
```

> Note: With Databricks notebooks, we can use the `%sql` to execute SQL code within a new cell in the same Python notebook. For instance, the equivalent code of the above code snippet would be as below.

```
%sql
select count(order_id) as total_orders, order_hour_of_day as hour
from orders
group by order_hour_of_day
order by order_hour_of_day
```

Note: Databricks supports various types of DataFrame and SQL visualizations within notebook cells. You can learn more about them here.

▶ Above line graph shows that the customers are more likely to place an order between 9 am to 6 pm

How often do customers place orders?

```
%sql
select days_since_prior_order,count(order_id) as total_orders
from orders
group by days_since_prior_order
order by days_since_prior_order
```

▶ It appears that most of the customers order once a week since the majority of records are concentrated between 0 to 7 days

▶Also, a large number of customer place their order 30 days or later days since because ‘days_since_prior’ column is capped at 30

On which day of the week customers purchase the most?

```
%sql
select count(order_id) as total_orders,
  (case
     when order_dow = '0' then 'Sunday'
     when order_dow = '1' then 'Monday'
     when order_dow = '2' then 'Tuesday'
     when order_dow = '3' then 'Wednesday'
     when order_dow = '4' then 'Thursday'
     when order_dow = '5' then 'Friday'
     when order_dow = '6' then 'Saturday'
   end) as day_of_week
  from orders
 group by order_dow
 order by total_orders desc
```

▶ Sunday and Monday have the most orders, while Thursday has the least orders in a week

How many items do customers purchase in an order?

Let's create a Master Dataset by merging together products, departments, order_products_train, and order_products_prior datasets together and run the query on top of that.

```
%sql
create table master_table as
(select op.*,p.product_name,p.aisle_id,p.department_id,d.department
from
 (select * from order_products_train
 union
 select * from order_products_prior) as op
 inner join products as p
 on op.product_id = p.product_id
 inner join departments as d
 on p.department_id = d.department_id)
```

```
%sql
select order_id,count(product_id) as total_items
from master_table
group by order_id
```

▶ The above bar chart depicts that the most common number of items purchased in order by customers is 4

▶ Majority of customers prefer to purchase between 1 to 15 items per order

Which are the top departments from which orders are placed?

```
%sql
select department, count(*) as orders_count from master_table
group by department
order by orders_count desc
limit 10
```

▶ If we take a look at top 10 departments from which most items are purchased, we would infer that almost 50% of the items purchased belong from just 2 departments which are '*produce*' and '*dairy eggs*'

Which are the most purchased items?

```
%sql
select product_name, count(*) as orders_count from master_table
group by product_name
order by orders_count desc
limit 200
```


▶ These are the top 8 items bought by Instacart customers in their orders. Banana seems to be most bought commonly bought item in baskets followed by *strawberries*, *baby spinach*, *avocado*, etc.

Let's also make a word cloud of the top 200 items bought by Instacart customers.

```
# !pip install wordcloud
from wordcloud import WordCloud
import matplotlib.pyplot as plt

df = sqlContext.sql("SELECT product_name FROM (select product_name,
count(*) as orders_count from master_table group by product_name
order by orders_count desc limit 200)")
df2 = df.rdd.flatMap(lambda x: x).collect()
fullStr = ' '.join(df2)

wordcloud = WordCloud(background_color="white").generate(fullStr)

# Display the generated image:
plt.figure(figsize=(14, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
display()
```

From the word cloud, it appears that Americans buy ***organic food*** and ***veggies*** a lot as words like *Organic, Milk, Water, Apple, Sparkling, Egg, Green, Cheese*, etc. are getting highlighted the most.

. . .

FP-Growth Algorithm

The FP-growth algorithm is described in the paper Han et al., Mining frequent patterns without candidate generation, where “FP” stands for frequent pattern. Given a dataset of transactions, the first step of FP-growth is to calculate item frequencies and identify frequent items. Different from Apriori-like algorithms designed for the same purpose, the second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly, which are usually expensive to generate. After the second step, the frequent itemsets can be extracted from the FP-tree.

> *You can learn more about the **steps involved in the FP-growth Algorithm** here.*

Organizing the data by the shopping basket

For implementing FP-growth, first, we would be creating baskets of each order in our dataset. We would do so by creating a *baskets* data frame having 2 columns: first, the *order_id* and second, the list of items bought in that particular order.

```
from pyspark.sql.functions import collect_set, col, count
rawData = spark.sql("select p.product_name, o.order_id from products p
                    inner join order_products_train o
                    where o.product_id = p.product_id")
baskets = rawData.groupBy('order_id').agg(collect_set('product_name').alias('items'))
baskets.createOrReplaceTempView('baskets')
rawData.show(5)
baskets.show(5)
display(baskets)
```

create_baskets.py hosted with ❤ by GitHub view raw

Below is a glimpse of the top 5 rows of the *baskets* data frame, to be fed into the FP-growth algorithm.

Implementation of FP-growth algorithm using Scala

Here, we would be using `spark.ml`'s FP-growth package for implementation.

Let's begin with taking minimum support value as 0.001 and minimum confidence value as 0. Both of these thresholds are user-defined. Minimum support value is selected such that there is a minimum 1 item in our itemsets and it does not take much time for computation (lower minimum support value results in more computation time).

```
%scala
import org.apache.spark.ml.fpm.FPGrowth

// Extract out the items
val baskets_ds = spark.sql("select items from baskets").as[Array[String]].toDF("items")

// Use FPGrowth
val fpgrowth = new FPGrowth().setItemsCol("items").setMinSupport(0.001).setMinConfidence(
val model = fpgrowth.fit(baskets_ds)
```

FPGrowth.py hosted with ❤ by GitHub view raw

Generating **frequent item-sets** and building **association rules**

```
// Display frequent itemsets
val mostPopularItemInABasket = model.freqItemsets
mostPopularItemInABasket.createOrReplaceTempView("mostPopularItemInABasket")

// Display generated association rules.
val ifThen = model.associationRules
ifThen.createOrReplaceTempView("ifThen")
```

frequent_itemsets_scala.py hosted with ❤ by GitHub view raw

Now, let us explore the frequent item-sets we generated above.

```
%sql
select items, freq from mostPopularItemInABasket where size(items) >
2 order by freq desc limit 20
```

The most frequent items of baskets comprise of *organic avocado, organic strawberries*, and *organic bananas* together.

A good way to think about *association rules* is that model determines that if you purchased something (i.e. the *antecedent*), then you will purchase this other thing (i.e. the *consequent*) with the following confidence.

```
%sql
select antecedent as `antecedent (if)`, consequent as `consequent
(then)`, confidence from ifThen order by confidence desc limit 20
```


The above results show that if a customer has *organic raspberries, organic avocados*, and *organic strawberries* in its basket, then it may make sense to recommend organic bananas as well. Surprisingly, the top 10 purchase recommendations either organic bananas or bananas.

The metric *lift* defines how independent are the *antecedent* and *consequent* in the itemsets. *Lift >> 1* means items are highly dependent on each other, *lift = 1* means items are independent of each other and l*ift <<1* means items are substitutes of each other (this means that presence of one item has a negative effect on the presence of other item and vice versa).

```
%sql
select * from ifThen where lift > 1 order by lift desc
```

As we can see in the above results, which has the *association rules* in decreasing the value of the *lift* values that if someone buys *Strawberry Rhubarb Yoghurt* then there is a very high chance of buying *Blueberry Yoghurt* as well.

Implementation of FP-growth algorithm using PySpark

The FP-growth algorithm we implemented above using Scala can also be implemented using PySpark with equivalent code shown below:

```
from pyspark.ml.fpm import FPGrowth

fpGrowth = FPGrowth(itemsCol="items", minSupport=0.001, minConfidence=0)
model = fpGrowth.fit(baskets)

model.freqItemsets.show() # Display frequent itemsets

model.associationRules.show() # Display generated association rules

model.transform(baskets).show() # transform examines the input items against all the ass
```

FPGrowth_pyspark.py hosted with ❤ by GitHub view raw

Summary

In this blog, we analyzed customer shopping behavior and performed Market Basket Analysis using Apache Spark on a huge dataset. If you want to implement the Market Basket Analysis in other environments, you can use *apyori* library in Python and *arules* library in R for the *Apriori algorithm*.

Well, that is all for this article. I hope you guys have enjoyed reading it, please share your suggestions/views/questions in the comment section.

Thanks for reading !!!

References: -

i) https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html

ii) https://s3.us-east-2.amazonaws.com/databricks-dennylee/notebooks/Market+Basket+Analysis+using+Instacart+Online+Grocery+Dataset.html

iii) https://towardsdatascience.com/association-rules-2-aa9a77241654

Market Basket Analysis Spark Data Science Machine Learning Association Rule