Apriori algorithm for Market Basket Analysis

Ayoola Bashir Idris-Animashaun
Scalable Systems Programming
M.Sc. Data Analytics
National College of Ireland
Dublin, Ireland
x20103689@student.ncirl.ie

***Abstract*—Apriori algorithm is a common technique used to find the frequent items in a data set.The algorithm uses a support measure to evict items that do not meet a set threshold penned for analysis. The algorithm is often used in the market basket analysis, although it has found other applications as well. In this paper, market basket and apriori are discussed and implemented on an e-commerce data set. A set of associative rules is given in the results showing pairs of antecedents and consequent with the confidence measure for their rule.**

Keywords- apriori, market basket analysis, hadoop, mapreduce

I. Introduction

Today, there is an explosion of technology and we can tell by the amount of hardware devices hitting the market every quarter. This inevitable leads to an even bigger explosion of interaction which means data is produced in even larger quantities and will continue to do so. Tech giants like Google have come up with open-source frameworks that allows for unique, scalable and reliable ways to handle large amounts of data. The framework used in this paper is Hadoop, re-introduced by Google. Commonly deployed on distributed environments where the volume of data is big, Hadoop comprises of two units that work to split, shuffle, sort data quicker than commercial data applications. The two units, or functions, called Mapper and Reducer, work by breaking down a data set line by line and performing some operation on each line in a distributed fashion. The process produces (key, value) pairs, which can be used during data mining processes.

Data mining can be said to be the distillation of data-sets into valuable. usable, insightful visualizations of information. MapReduce can play a part of this process by carrying out data transformation or analysis on huge data-sets. One type of data-set that could exist is a list of transactions in a store. Suppose a store has a record of all orders taking place and wishes to know what items are bought together. We can apply MapReduce to work on the massive dataset we would be working with. We will be using MapReduce due to its flexible, fast, resilient, scalable and cost-effective nature. The framework supports parallel programming and offers security and authentication. The hadoop file system has good read/write characteristics thus processing data faster.

Many algorithms are available to know what items are bought together, in this paper, we select to use the apriori algorithm for its simple, reliable application to the task. In finding out items that are bought together, we look to establish associations between items such that, we can say, with certain confidence, that buying an item A will likely mean an item B will also be purchased in the same market basket. The market basket analysis uses the apriori algorithm to find items that are purchased together. The principle can be applied in other areas like drug creation, and text analysis.

Objectives: In this paper, we explore the implementation of apriori on a dataset of transactions and extract items that sell together and items that are frequently bought cumulatively. although, our dataset has over 100,000 rows, it is small compared to what MapReduce can handle, we will however utilize the dataset to show how MapReduce can achieve a better execution time for the processing og big data sets. The dataset we will be using is the Olist Brazilian E-commerce website. The website serves as a market place where sellers are connected to buyers. Olist acts as a distribution center that routes orders to the nearest seller who can fulfill the order.

II. Related Works

[2] proposed a Transaction Reduction MapReduce Apriori algorithm to address the repeated processing of the traditional apriori algorithm. On its own, the apriori algorithm finds candidate items by sweeping over the dataset as many times as it needs to. This puts a drag on computing resources, the TRM algorithm works by removing items with support values lower than a set threshold. The remaining candidate go on to become pairs and so on. The basic idea is that only items which are frequent individually that can be frequent together. In [3], the deployment of MapReduce on AWS cloud is researched. This study exposes some of the limitations of MapReduce like limitations on parallelism, incompatibility with tools and algorithms. (key, value) data representation presents as loss of data features. It is also discovered that performance increases with increase in nodes up to a certain point. The apriori algorithm is also used in this study to see the effect of cloud platforms when processing data with map reduce. In [4], the research focuses on MapReduce's performance based on number of EC2 nodes in a AWS environment. The research, used Hadoop, writtenin Java and deployed on a AWS cluster to run experiments on the performance on Hadoop with varying sizes of data and number of records. Lastly, in [1], a survey exposing the perspective of big data's application in association rule mining was done with the objective to painting the role of

cloud computing in data processing. Some of the algorithms surveyed include the Hybrid Frequent Itemset Mining (HFIM) in Spark, a framework built to make Hadoop faster, uses a vertical and horizontal layout to find the association in the dataset. Privacy focused algorithms like the privacy-preserving utility mining algorithm (PPUM), which generates frequent items by using a minimum utility threshold is also surveyed in this literature. The literature mentions the TRMR-Apriori proposed by 5a as possessing the strength to find frequent items from massive datasets hosted in the cloud.

III. Methodology

A. Data attributes

The Olist ecommerce dataset that we will be working with has 7 files within it relating to domains within the store like products, orders, sellers, customers, geolocation, payments and reviews. The files are related using unique attributes like $order\_id, customer\_id, product\_id$. For this paper, we will be making use of the $olist\_order\_items\_dataset$. It contains the record of all orders taken within a certain period in time. The orders are given unique IDs and each row in the data set corresponds to one product bought under a given unique order ID. For example, if product A and product B were bought under order A, there will be a row showing the details for product A under order A and another row showing the details for product B under order A. The attributes collected for each product bought under an order, are $order\_id$, $order\_item_id$, $product\_id$, $seller\_id$, $shipping\_limit_data, price, freight\_value$. To find our frequent item sets, we will need to arrange our data by putting all purchased products belonging to an order together. We will use this as the input to our apriori algorithm powered by MapReduce.

B. Data loading

The olist dataset is available on Kaggle and is allowed for research purposes under the Kaggle data privacy policy. The zip file containing the records is downloaded into the local environment set up for this paper.

C. EDA

In exploring our data, we take an initial look using a Jupyter notebook. For the analysis in this paper, the columns we need for the market basket analysis do not have missing values, as shown in Figure 1, where we check for missing values.

The data is not missing any values, and there are over 95,000 unique orders to work with. This is small compared to what Hadoop can manage. We also see that we have over 30,000 unique products being sold by 3,000+ entrepreneurs in Brazil.

We can add the English translation of the product categories to give context. We take a look at the top 20 product categories for the store. We expect to see our frequent items coming from these categories.

```
olistdata.info()
#No missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112650 entries, 0 to 112649
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   order_id             112650 non-null  object
 1   order_item_id        112650 non-null  int64
 2   product_id           112650 non-null  object
 3   seller_id            112650 non-null  object
 4   shipping_limit_date  112650 non-null  object
 5   price                112650 non-null  float64
 6   freight_value        112650 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 6.0+ MB
```

Fig. 1: Data info

```
#We look at how many unique orders there are
olist = olistdata.copy(deep=True)
olist['order_id'].nunique()
```

```
98666
```

```
#We can see that we have more rows than unique order_id,
#Let's see how many products are available in Olist
olist['product_id'].nunique()
```

```
32951
```

```
#How many sellers
olist['seller_id'].nunique()
```

```
3095
```

Fig. 2: Data features

```
#We can expect to see our frequent items come from these categories

import seaborn as sns

ttp = pd.value_counts( new_olist['product_category_name_english'])
#top_cat_plot = sns.barplot(data = top_cat)
top_cat_plot = ttp[len(ttp)-20:len(ttp)].plot.barh()
top_cat_plot
```

```
<AxesSubplot:>
```

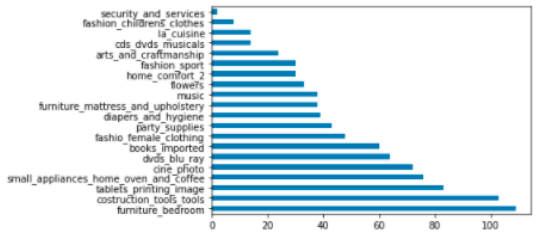

Fig. 3: Top product categories

D. Understanding Support & Confidence

a) : **Support:** The support is a measure of how many times or how frequent a particular item (or item sets) is in a given list of transactions. It is basically counting how many times the item (or item sets) comes up over the total count of transactions. Supposing we have two items, suitcasessuitcases and milkmilk, sold by a particular store. One would expect that the number of times that milk is bought is higher than

than the number of times that suitcases are bought. We would say that milk has a higher support than suitcases.

$$Support(milk) > Support(suitcases)$$

Now, imagine another scenario, where we have suitcases and gift box as an item set $suitcases, gift\ box$ and shoes and gift box shoes,gift box as another item set. One would expect that we have more sales of shoes and gift boxes than suitcases and gift boxes.

$$Support(shoes, box) > Support(suitcases, gift\ box)$$

$$Support(A) = \frac{N}{T} \tag{1}$$

The support is mathematically represented as in equation 1, where we have a fraction of the number of times a particular item (or item sets) comes up in a given list of transactions over the total number of transactions. In the example given above, if 5 is the total number of times milk is bought in a list of 30 transaction, then the support of milk will be as follows;

$Support(A) = \frac{5}{30}$ which is a support of 16%.

If this support level exceeds the given threshold for a certain market basket analysis, then milk will be counted as a frequent item and we can find what other items are bought frequently with milk. To be able to say, that the presence of milk and another product are frequently bought together, such that, we can estimate that when a consumer buys one, they buy the other, we calculate a confidence measure for both items.

b) : **Confidence** When we talk of confidence, it is a measure of how likely a particular item will be selected given that some other items have been selected already. Continuing with our example from above, if a show is selected and added to a cart, we can estimate that a gift box could be selected. We could also estimate that instead of a gift box, a pair of trousers will be selected instead or even a belt. We can look at past transactions to build an associative rule that tells us with a level of confidence, the items that could be bought or selected to be bought, if a shoe has been chosen to be bought already.

In this case, the shoe is the antecedent and the other item that we want to know is the consequent. We have a conditional probability of the occurrence of the consequent given the presence of a particular antecedent. From our example, we will have
{shoes} —¿ {gift box}
{shoes} —¿ {pair of trousers}
{shoes} —¿ {belt}

The confidence level of each rule will tell us how likely the occurrence of each scenario is based on the frequency of each item set occurring a given number of transactions. The confidence can be derived from equation 2, which shows the support of the transactions containing the antecedent and the consequent together over the support of the antecedent alone.

$$Confidence(A - -- > C) = \frac{Support(AUC)}{Support(A)} \tag{2}$$

E. Architecture of Mapreduce workflow

The architecture for our workflow is shown in Figure 2, we have two MRJob steps. The first step has a mapper that generates all the products belonging to a particular order_id and a reducer that extracts the products into (key, [value]) tuples where the key is the order_id and the value is a list of products that was purchased in that order_id. The next step is to initialize a list that will be used to store the frequent items on each pass of the apriori algorithm. Another mapper is used in this step to generate the items, while a combiner and reducer function sorts and aggregates the count for each item. The output is a (key, value) line where the key is the item and the value is the number of times it occurs within our transaction record.

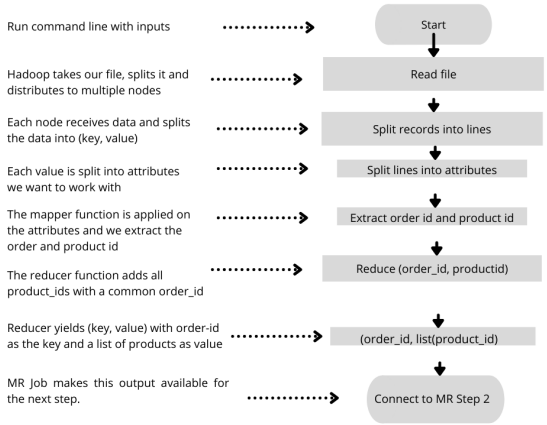

Fig. 4: MR Step 1

On the first iteration of the algorithm, all items that meet a certain support threshold are kept for the next iteration. These items are written to a text file to await the next iteration. Since we are looking for items that are frequently bought together, it stands to reason that each of the items itself will be a frequently bought item. As a result, we start by establishing the list of items that are frequently purchased.

From this list, we raise a new list by pairing each item on our list with another and seeing how many times the pair is bought together. We use our support threshold once again to eliminate items that do not meet up. Lastly, we calculate a confidence for our chosen antecedent and consequent pairs. Please note that the antecedent could contain of more than one item. The confidence helps us know the exact combination

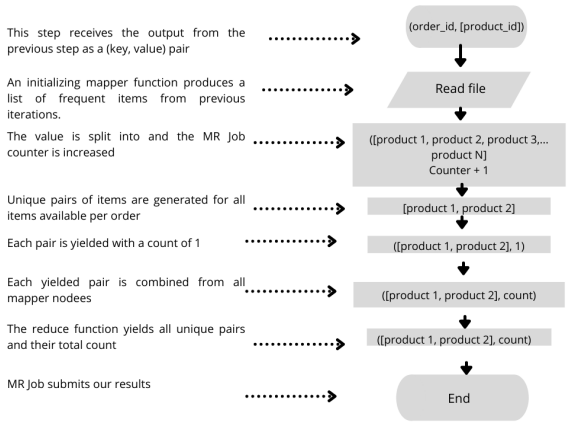

Fig. 5: MR Step 2

of items that influence one another. This is where we get associative rules, where we can say, if product A and product B is bought, we can predict that a certain item, product D will be bought as well.

F. Justification for data processes involved

Due to the volume of data, running apriori algorithm on a single node machine can be tedious and slow. Why not use MapReduce where the data can be split across multiple nodes, processed and items generated. In the first step, we see the power of MapReduce in quickly sorting the purchases according to the $order\_ids$. In the second step, by eliminating the items that do not meet our support threshold during the iteration, we save time on sifting through the data to remove this low support items. The MapReduce brings tremendous throughput to our performance in this regard. Lastly, when we calculate the confidence for each frequent item set we have, we are able to get the associative rules for our products.

G. Ethics

The data we are using does not have private information in it. The customers data-set included uses unique $'customer\_id' and 'customer\_unique\_id'$ to preserve privacy for its customers. Lastly, the data-set is used under the Kaggle privacy policy and we make no attempts to disclose or find out the real customers in the data-set.

RESULTS

From our experiments, and as seen in Figure 6, we find the most number of associative rules for a $k = 3$ frequent item set at a support level of 0.0001/0.00011, a confidence threshold of 0.4/0.45, and a decay rate of -0.7/-0.65/-0.6. The tables below show results for different levels of support, confidence level, and decay rate, all with a fixed number of frequent items set at 3.

```
1 946344697156947d846d27fe0d503033,b0c89945c034268074f5f80b362bda34: --->
  ad0a798e7941f3a5a2fb8139cb62ad78 : support = 2.0270407232481302e-05 : confidence =
  0.6666666666666667
2 b0c89945c034268074f5f80b362bda34,ad0a798e7941f3a5a2fb8139cb62ad78: --->
  946344697156947d846d27fe0d503033 : support = 2.0270407232481302e-05 : confidence =
  0.6666666666666667
3 b9900407a55cb2b306ae612415c3340e,e2cac69b319c0f8a21dbf04b925121bf: --->
  55bfa0307d7a46bed72c492259921231 : support = 3.0405610848721 95e-05 : confidence = 1.0
4 b9900407a55cb2b306ae612415c3340e,55bfa0307d7a46bed72c492259921231: --->
  e2cac69b319c0f8a21dbf04b925121bf : support = 3.0405610848721 95e-05 : confidence = 1.0
5 e2cac69b319c0f8a21dbf04b925121bf,55bfa0307d7a46bed72c492259921231: --->
  b9900407a55cb2b306ae612415c3340e : support = 3.0405610848721 95e-05 : confidence = 1.0
6 fc5dd987f12a7b823a76a44aa1ba88f6,07ffc018eaf23e086370dea42c74077b: --->
  7eaf66aca1ed54df09b3c864b7416110 : support = 2.0270407232481302e-05 : confidence =
  0.6666666666666667
7 fc5dd987f12a7b823a76a44aa1ba88f6,7eaf66aca1ed54df09b3c864b7416110: --->
  07ffc018eaf23e086370dea42c74077b : support = 2.0270407232481302e-05 : confidence =
  0.6666666666666667
8 35afc973633aaeb6b877ff57b2793310,f2e53dd1670f3c376518263b3f71424d: --->
  99a4788cb24856965c36a24e339b6058 : support = 2.0270407232481302e-05 : confidence =
  0.6666666666666667
9 f4d705aa95ccca448e5b0deb6e5290ba,c211ff3068fcd2f8898192976d8b3a32: --->
  4025ee582ef6b8c478af3b44cf89054b : support = 2.0270407232481302e-05 : confidence = 0.5
```

Fig. 6: Associative rules

a) : **Unexpected findings** One unexpected finding was the low support threshold that was needed to derive the items mentioned above. One future task to undertake is to find all products bought by a customer and build an associative rule for finding out what products a customer could buy given past products in their order history.

Another unexpected find is that we got good rules only when we use negative decays.

K	Support	Confidence	Decay	Results
3	0.0001	0.4	-0.7	7
3	0.00011	0.4	-0.7	9
3	0.00012	0.4	-0.7	8
3	0.00013	0.4	-0.7	2
3	0.00014	0.4	-0.7	1
3	0.00015	0.4	-0.7	1

TABLE I: Support effect on results

K	Support	Confidence	Decay	Results
3	0.0001	0.4	-0.7	7
3	0.0001	0.45	-0.7	9
3	0.0001	0.5	-0.7	6
3	0.0001	0.55	-0.7	6
3	0.0001	0.6	-0.7	6
3	0.0001	0.65	-0.7	3

TABLE II: Confidence effect on results

K	Support	Confidence	Decay	Results
3	0.0001	0.4	-0.7	7
3	0.0001	0.4	-0.65	9
3	0.0001	0.4	-0.6	9
3	0.0001	0.4	-0.55	3
3	0.0001	0.4	-0.5	0

TABLE III: Decay effect on results

In Table I, we see that as support threshold increases, the results we get decreases. This shows that the number of individual items that are larger than the support threshold we select, reduces. Since they do not come up on the single pass, they will not be paired with other frequent items on further iterations.

Table II refers to the effect the confidence level has on the associative results we get. As confidence increases, the results drop. This means, at higher confidence level, there are fewer pairs of antecedent and consequent that we can be sure about.

Table III refers to the effect of decay on the support, which affects the results in return. A decay function is added to get the best number of items on each run of the apriori process. If the support was static, less frequent items would be found and in some situations, there might be no item-sets found. To achieve the goal of the market basket analysis, an exponential decay function is used to calculate the support on each iteration.

b) : **Relevance of results** Our results show us the items that are frequently purchased together. The relevance of this for our e-commerce website is in packaging these items in such a way as to booster sales for the sellers on the platform. They could present these findings to the sellers to encourage partnership, discount sales and more goodies for their customers to enjoy from.

IV. Conclusion

In this paper, we have carried out a market basket analysis using the apriori algorithm to find the frequent items that are bought together from our records of transaction. Our approach used a multi step MR procedure to pre-process our data and input it into our apriori algorithm. From our analysis, we derived some associative rules for items that are bought together. The associative rules are supported with a 40% confidence between the antecedent and the consequent. We ran several hadoop jobs to get the best support level as well as decay level.

For future works, an experiment to find all items bought by a customer and using this to predict what items a customer would buy next can be carried out. We can also look at deploying another algorithm, the PCY algorithm in finding creating associative rules from the list of transactions.

References

[1] A. Kumar and D. Hariprasad. "A Survey on Association Rule Mining Algorithm with Hadoop MapReduce and Spark : A Big Data Perspective". In: 2019.

[2] Sayeth Saabith, Elankovan Sundararajan, and Azuraliza Abu Bakar. "A Parallel Apriori-Transaction Reduction Algorithm Using Hadoop-Mapreduce in Cloud". In: *Asian Journal of Computer Science and Information Technology* 1 (Apr. 2018). DOI: 10.9734/AJRCOS/2018/41045.

[3] Jongwook Woo. *Market Basket Analysis Algorithm on Map/Reduce in AWS EC2*.

[4] Jongwook Woo and Yuhang Xu. "Market Basket Analysis Algorithm with Map/Reduce of Cloud Computing". In: (Apr. 2012).