Association Rule Mining for a UK Online Retail Company

Alex Fung   Viswesh Krishnamurthy   Tony Lee   Patrick Osborne

Feb 26, 2020

Abstract

The retail industry is one that has changed far beyond recognition with the advent of internet. From once having to make a list, physically travel to a store, buy & haul your purchase yourself, to now simply ordering your needs online and getting it delivered in less than 24 hours, retail buying has become far more convenient. This buying convenience has also introduced challeges on the part of the sellers. The once obvious buying patterns are no longer obvious and requires complex analyses to understand customer preferences. In this project, we attempt to help a UK based Online Retail store understand their customers' buying patterns.

Background

One of the most powerful tools in online retail is a recommender system. Such a system helps sellers mine through their sales and unearth important associations between their products. In turn, such associations can be presented to customers as recommendations. Our client, the online retail store wishes to build a long term strategy based on the understanding this project gives them.

Objective

The objective of our analysis is to develop an usupervised, prediction model using Machine Learning techniques and the CRISP-DM framework (cite textbook) on the available transaction data to optimally place product that are frequently purchased together, or to suggest other items to purchase when certain items are added to the online shopping cart. The intent of this is to increase sales at this retailer by making it more convenient for the purchaser to find products related to the ones they intend to purchase and to upsell at the cashier or online checkout

Data Analysis

The original data set, "Online Retail.csv" is sourced from the UCI Machine Learning Repository. It is a transnational data set which contains all the transactions occurring between 01\12\2010 and 09\12\2011. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers. The link to the Online Retail Data Set from the UCI Machine Learning Repository can be found here: https://archive.ics.uci.edu/ml/datasets/online+retail.

Data dictionary

The following data dictionary of the dataset is provided. The columns of interest in this specific case of association rule mining are columns `InvoiceNo`, which acts as a unique ID for each transaction, and `Description`, which provides the unabridged name of the item.

Table 1: Data Dictionary - Online retail store

Feature	Feature.Description
InvoiceNo	Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter ’c’, it indicates a cancellation.
StockCode	Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
Description	Product (item) name. Nominal.
Quantity	The quantities of each product (item) per transaction. Numeric.
InvoiceDate	Invice Date and time. Numeric, the day and time when each transaction was generated.
UnitPrice	Unit price. Numeric, Product price per unit in sterling.
CustomerID	Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
Country	Country name. Nominal, the name of the country where each customer resides.

Initial Data Exploration & Cleaning

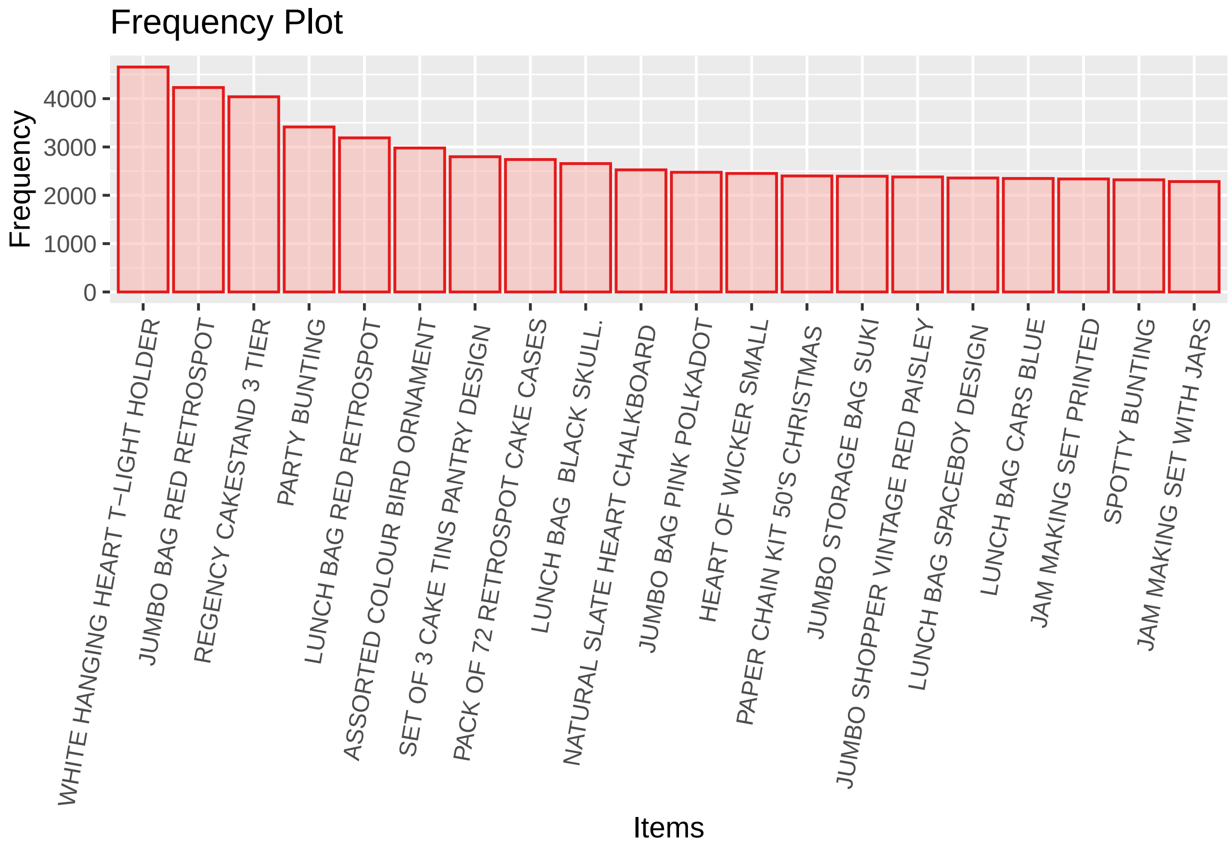
The original data set is relatively clean, and for the Apriori and ECLAT algorithms, will not require much preparation before running the association rule mining algorithms. For the FP Growth algorithm, further manipulation of the dataset will be needed, and is covered under the FP Growth Algorithm section in detail.

A quick look at the header of the data set is provided below.

X	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850	United Kingdom
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850	United Kingdom
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/10 8:26	7.65	17850	United Kingdom

We learn from the original data that all invoices that start with a “C” are cancelled orders and therefore are removed from the data. Also, the patterns will be interpreted based on the product descriptions and hence all rows with empty descriptions are deleted too.

DOTCOM POSTAGE: Lines with description “DOTCOM POSTAGE” refer to postage charges and don’t contribute anything meaningful to the problem. Hence those lines are removed as well. This now leaves us with a dataset that has all valid invoices and items



Models

Association Rule Learning Algorithms

From the onset, it was clear that an Association Rule Learning model will provide a data solution to a business problem. Association Rule Learning will allow the discovery of potentially hidden, or interesting relationships on the patterns of online purchases by customers. By using various algorithms, it may be possible to uncover what customers typically purchase given one or more specific items, and then recommend those items to the customer. Doing so will allow the retailer to maximize its sales, and the customer will also be happy with their additional purchases.

For this dataset, it was decided to use the following models for association rule mining as a starting point:

- Apriori algorithm
- FP Growth algorithm
- ECLAT algorithm

These algorithms were chosen for their ease of understanding, ease of explanation, and performance. Their descriptions, and attributes are covered more in detail in their own sections below.

Model Parameters

All association rule mining algorithms have 3 very important parameters:

- Support

- Confidence
- Lift

Support Support is the proportion that an item represents in the total transaction dataset. For example, *support(PINK REGENCY TEACUP AND SAUCER) => (Trasanctions featuring (PINK REGENCY TEACUP AND SAUCER))/Total Transactions In regards to this dataset, a high support denotes that the item is frequently purchased, whereas a low support denotes the item is rarely purchased in transactions.

Confidence Confidence is defined as the probability that an item combination was bought. For example, *confidence(PINK REGENCY TEACUP AND SAUCER & GREEN REGENCY TEACUP AND SAUCER) => (Total transactions with both PINK & GREEN TEACUP & SAUCER)/Total Transactions In regards to this dataset, a high confidence denotes that the association rule for this itemset was not found to be true for a high percentage, whereas a low confidence denotes that the association rule for this itemset was not found to be true for a high percentage.

Lift Lift is defined as the increase in the chance that an item combination is bought when a single item in that combination is bought. For example, *Lift(PINK & GREEN REGENCY TEACUP & SAUCER) = Confidence(PINK & GREEN REGENCY TEACUP & SAUCER)/support(GREEN REGENCY TEACUP & SAUCER) In regards to this dataset, a high lift where the score is greater than 1, denotes that the items within the itemset are highly dependent on one another. An average lift where the score is approximately equal to 1, denotes that the items within the itemset are independent of one another. A low lift where the score is less than 1, denotes that the items within the dataset are negatively dependent of one another; in other words, these items are never purchased together.

Evidently, for a rule generated by the association rule mining algorithms to be useful, the Support, Confidence, and Lift parameters must be specified in such a way that they provide some business value. Therefore, for the Apriori, FP Growth, and Eclat algorithms, a minimum Support of 0.01 was chosen. This meant that only rules where at least 1% of all transactions had to featur the specified rule. Any generated rules with less than 1% support were not of interest, as the online retailer would not gain minimal sales from rules that have a low support. In addition, a minimum confidence of 75% was chosen, because only itemsets with a medium of high probability of being fulfilled were interesting.

Apriori algorithm

Description

The apriori algorithm is one that is custom built for mining for association rules in a dataset. This algorithm works on datasets that are transactions. In our case, we have already converted the dataset into “transactions” prior to running the model on. The apriori algorithm, as the name suggests works on the basis of previously mined information. It is a breadth first algorithm, where it mines for frequent subsets by traversing across the breadth of each level of transaction.It starts by creating a “candidate set” which is a table of count of each unique item in the dataset. In the next step, it expands by counting each frequently appearing pair and then three items in the subsequent step and so on and so forth. At each step and finally, the stop criterion for the algorithm is decided by the “support” metric explained above

Running Apriori Algorithm

```r
#apriori algorithm
apriori.rules <- apriori(trans, parameter = list(supp=0.01, conf=0.75,minlen = 3, maxlen=5))
apriori.rules_df<- data.frame(lhs = labels(lhs(apriori.rules)), rhs = labels(rhs(apriori.rules)),
                              apriori.rules@quality)
```

Apriori Association Rules Result

The results of the apriori algorithm are association rules which are directly seen in R Console. For ease of understanding, the rules have been converted into a data frame and the head of the data frame is shown below.

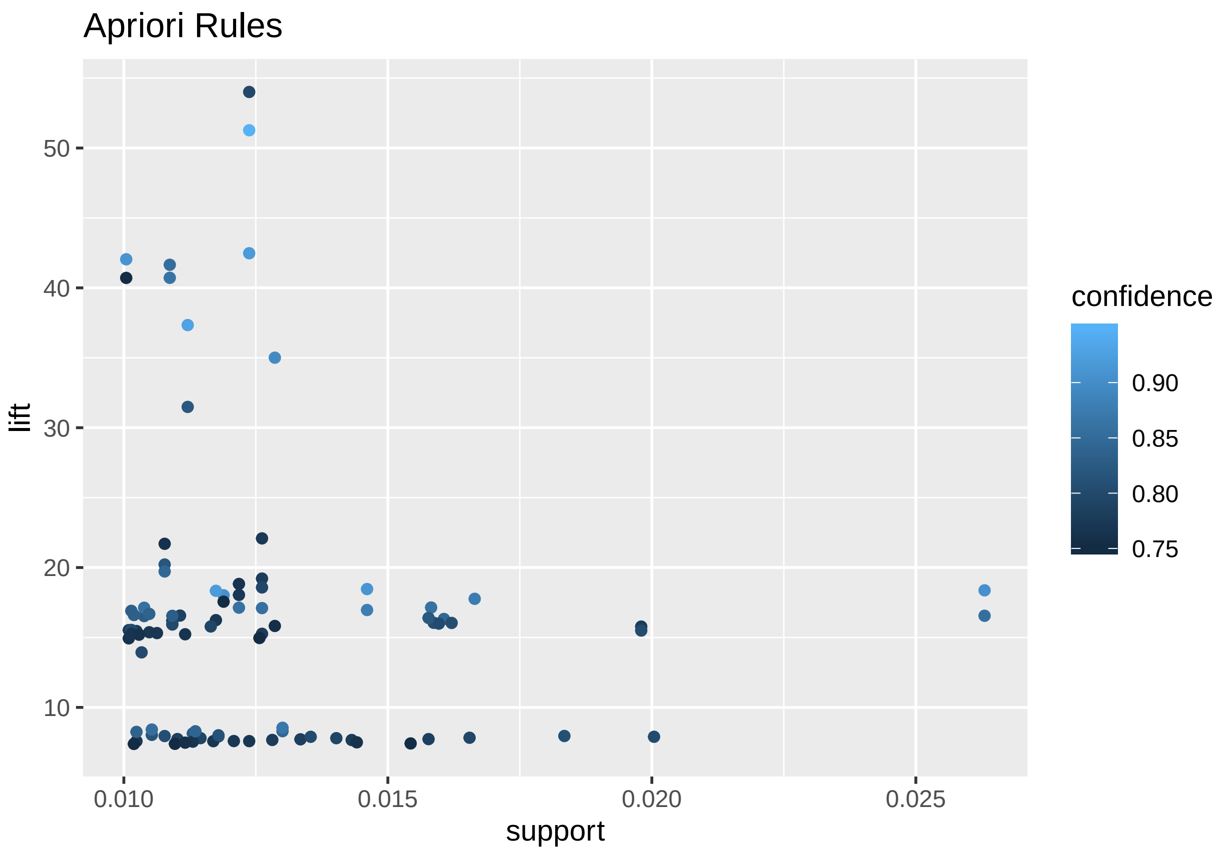
```r
head(apriori.rules_df)
```

```
##                                                           lhs
## 1   {ALARM CLOCK BAKELIKE GREEN,ALARM CLOCK BAKELIKE ORANGE}
## 2          {REGENCY TEA PLATE GREEN ,REGENCY TEA PLATE PINK}
## 3          {REGENCY TEA PLATE PINK,REGENCY TEA PLATE ROSES }
## 4        {REGENCY TEA PLATE GREEN ,REGENCY TEA PLATE ROSES }
## 5  {POPPY'S PLAYHOUSE KITCHEN,POPPY'S PLAYHOUSE LIVINGROOM }
## 6 {POPPY'S PLAYHOUSE BEDROOM ,POPPY'S PLAYHOUSE LIVINGROOM }
##                             rhs    support confidence     lift count
## 1  {ALARM CLOCK BAKELIKE RED } 0.01009366  0.7619048 14.93870   208
## 2   {REGENCY TEA PLATE ROSES } 0.01237444  0.9172662 42.47664   255
## 3   {REGENCY TEA PLATE GREEN } 0.01237444  0.9479554 51.27170   255
## 4     {REGENCY TEA PLATE PINK} 0.01237444  0.7993730 54.00879   255
## 5 {POPPY'S PLAYHOUSE BEDROOM } 0.01087009  0.8549618 41.65059   224
## 6  {POPPY'S PLAYHOUSE KITCHEN} 0.01087009  0.8615385 40.71955   224
```

Apriori Scatter Plot

A scatter plot is an easy way to interpret the association rules graphically. Along the x-axis is `Support`, plotted against `Lift` in the y-axis, with `Confidence` being the 3rd dimension depicted through the color spectrum on the right of the graph

```r
#plot(apriori.rules, method = "scatterplot", main = "Apriori Rules")
qplot(
  x = support,
  y = lift,
  color = confidence,
  data = apriori.rules_df,
  geom = "point",
  main = 'Apriori Rules'
)
```



FP Growth Algorithm

Description

The Frequent Pattern Growth algorithm, or FP Growth in short, is an alternate frequent pattern mining algorithm. The algorithm first iterates through each transaction to determine the number of occurrences of itemsets in a database, not unsimilar to the Apriori algorithm where it too has to do the same step in the beginning. The root node is null, but thereafter each node represents a different itemset, while the branch represents the association formed between each itemset. The FP Growth algorithm the scans the itemsets again, and iterates through each transaction. For a given transaction, the FP Growth algorithm looks for the itemset with the maximum number of occurrences is moved to the top of the tree, before moving onto the next transaction. The itemsets are ordered in descending order based on the number of occurrences. As the algorithm iterates through each transaction, the count of the itemset is incremented. Once the tree-like structure is generated, the tree is then minded for conditional pattern bases. Starting from the lowest node, the tree is then traversed upwards to the root node in order to generate the conditional pattern bases. Once the conditional pattern bases are generated, the Frequent Pattern Tree is then generated by aggregating the count of itemsets through each conditional pattern base.

The FP Growth algorithm has several advantages: (a) it is faster than the Apriori algorithm, as the runtime for FP Growth is linear, whereas the runtime for Apriori algorithm is exponential. The FP Growth algorithm only scans the database of itemsets twice in order to build the FP tree, while the Apriori algorithm must scan the database of itemsets multiple times to generate candidate sets. However, the tree structure generated by the FP Growth algorithm is large, and if the database is too large, the algorithm may run into issues fitting the tree into memory. This issue was immediately encountered when attempting to run the FP Growth algorithm using the `rCBA` library, as it needed to allocate memory of size 96GB, which was far more than what was available on our local machines.

Therefore, we had to use the FP Growth Algorithm from `Sparklyr` library, which is a R implementation of Apache Spark. Spark is an open-source distributed cluster-computing framework maintained by the Apache Software Foundation. The `Sparklyr` library offers its implementation of the FP Growth Algorithm, which will be used for this dataset. The following tutorial at https://longhowlam.wordpress.com/2017/11/23/association-rules-using-fpgrowth-in-spark-mllib-through-sparklyr/, which was written by Longhow Lam, provided a very useful explanation and usage of the `Sparklyr` implementation of the FP Growth Algorithm, and a good portion of the code written below in this paper is based off the tutorial with some modifications.

Installation of Spark

In order to use `Sparklyr`, `Sparklyr` and its corresponding Spark Hadoop binaries must be installed. At the time of this paper, the version of the Spark Hadoop binaries used was `2.4.3`. To install `Sparklyr`, first run `install.packages('sparklyr')`. Confirm that the library is loaded via `library(sparkly)`. Afterwards, the Spark Hadoop binaries can be installed via `install_spark()`. If the aforementioned command does not work, then the binaries must be downloaded manually and then installed locally via `spark_install_tar('folder/spark-2.4.3-bin-hadoop2.7.tgz)`.

Data Preparation for `Sparklyr` Implementation

Spark must first be instantiated locally, although it can be instantiated on a cluster if one is available. Because the `Sparklyr` implementation of FP Growth does not accept a `transactions` type, the dataset had to be converted into a dataframe manually. All columns except for `InvoiceNo` and `Description` were removed. For any duplicates itemsets found in a transaction, the duplicates were removed from the dataset in order for this the `Sparklyr` implementation of the FP Growth algorithm to run. Afterwards, the cleaned dataframe is then uploaded to the Spark local cluster. Lastly, the Spark dataframe is grouped by `InvoiceNo`, which represents the UID (unique ID) of each transaction, after which the itemsets are aggregated into a list so that the grouped and aggregated Spark dataframe is now ready to be used by the FP Growth Algorithm.

```
#to run the FP Growth algorithm, Spark needs to be installed
#to install Spark, try install_spark() first

#if not, install spark from tar
#spark_install_tar('C:\\Users\\alexf\\Downloads\\spark-2.4.3-bin-hadoop2.7.tgz')

#spark connect and instantiate locally
sc <- spark_connect(master = "local")

#columns 2  (InvoiceNo) and 4 (Description) are needed
#apply each column as factor afterwards, then convert into dataframe
data_cols_removed <- mydata[ -c(1, 3, 5:9) ]
train <- sapply(data_cols_removed, as.factor)
train <- data.frame(train, check.names=FALSE)

#remove duplicates from train
train_no_duplicates = train[!duplicated(train), ]

#upload to spark
trx_tbl  = copy_to(sc, train_no_duplicates, overwrite = TRUE)

#we first group by InvoiceNo, which represents an unique id for each transaction
#the items in Description are then aggregated into a list
#this format is required by the Sparklyr implementation of the FP Growth algorith
trx_agg = trx_tbl %>%
  group_by(InvoiceNo) %>%
  summarise(
    items = collect_list(Description)
  )
```

Run the FP Growth Algorithm

After the dataset is prepared and uploaded to Spark, the `Sparklyr` implementation of the FP Growth Algorithm is then invoked on the Spark cluster. Because Spark jobs are lazily evaulated, the job to create the FP Growth model is not executed until the association rules are extracted.

```
uid = sparklyr:::random_string("fpgrowth_")
jobj = invoke_new(sc, "org.apache.spark.ml.fpm.FPGrowth", uid)

FPGmodel = jobj %>%
  invoke("setItemsCol", "items") %>%
  invoke("setMinConfidence", 0.75) %>%
  invoke("setMinSupport", 0.01)  %>%
  invoke("fit", spark_dataframe(trx_agg))
```

Extract the Association Rules from FP Growth Model

A function to extract the association rules from the FP Growth model is declared and defined below. The association rules are generated upon the invocation of `assocationRules`, where the FP Growth algorithm is finally executed. As stated before, much of the code to extract is derived from the work of Longhow Lam's tutorial. The code is slightly modified to include the calculation of Support, which is unfortunately not provided by the `Sparklyr` implementation of the FP Growth Algorithm.

```r
#ml_fpgrowth_extract_rules extracts the rules generated by Sparkylr FPGrowth function
#FPGmodel is the FP Growth model as created by Sparklyr
#nLHS is the desired number of antecedent items
#nRHS is the desired number of consequent items

#much of this function can be found at Longhow Lam's tutorial @ https://longhowlam.wordpress.com/2017/1
ml_fpgrowth_extract_rules = function(FPGmodel, nLHS, nRHS)
{
  #invoke associationRules on FPGmodel
  #then register the resulting rules into Spark
  rules = FPGmodel %>% invoke("associationRules")
  sdf_register(rules, "rules")

  #separate antecedent into LHSItem[0:nLHS], where nLHS is number of antecedent items
  exprs1 <- lapply(
    0:(nLHS - 1),
    function(i) paste("CAST(antecedent[", i, "] AS string) AS LHSitem", i, sep="")
  )
  #similarly, separate consequent into RHSItem[0:nRHS], where nHRS is the number of consequent items
  exprs2 <- lapply(
    0:(nRHS - 1),
    function(i) paste("CAST(consequent[", i, "] AS string) AS RHSitem", i, sep="")
  )

  #invoke the expressions
  #then register the results into Spark
  splittedLHS = rules %>% invoke("selectExpr", exprs1)
  splittedRHS = rules %>% invoke("selectExpr", exprs2)
  p1 = sdf_register(splittedLHS, "tmp1")
  p2 = sdf_register(splittedRHS, "tmp2")

  ##finally, combine the columns together
  #calculate support based on confidence/lift
  bind_cols(
    sdf_bind_cols(p1, p2) %>% collect(),
    rules %>% collect() %>% select(confidence, lift) %>% mutate(support = confidence/lift)
  )
}

#extract grocery rules by calling the function ml_fpgrowth_extract_rules
#we set a minimum of 5 precedent items
#and result of 1 consequent item
GroceryRules = FPGmodel %>% ml_fpgrowth_extract_rules(
  nLHS = 5,
  nRHS = 1
)
```

FP Growth Association Rules Result

Below the first few association rules generated by the FP Growth model are also printed below.

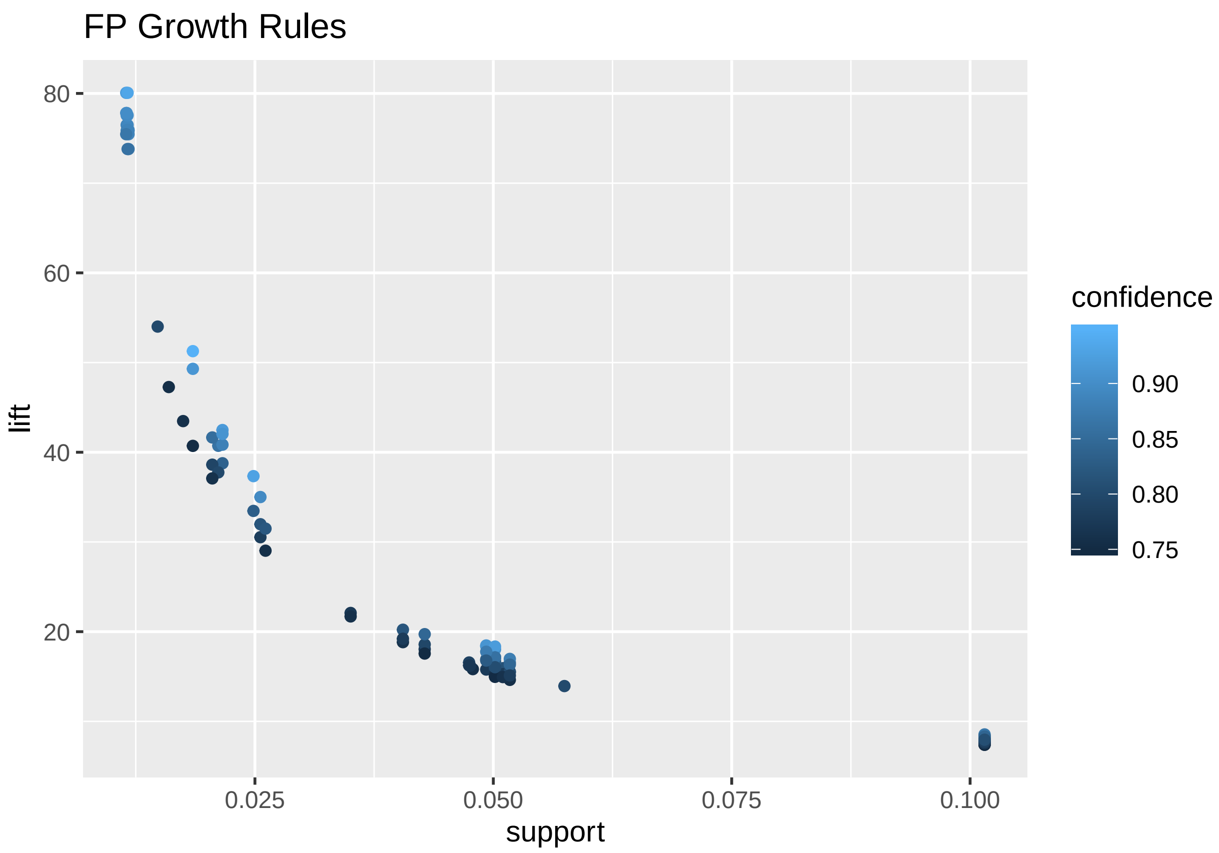
```
#print the first few rules
head(GroceryRules)
```

```
## # A tibble: 6 x 9
##   LHSitem0 LHSitem1 LHSitem2 LHSitem3 LHSitem4 RHSitem0 confidence  lift support
##   <chr>    <chr>    <chr>    <chr>    <chr>    <chr>         <dbl> <dbl>   <dbl>
## 1 JUMBO B~ JUMBO S~ <NA>     <NA>     <NA>     JUMBO B~      0.762  7.50  0.102
## 2 STRAWBE~ CHARLOT~ RED RET~ <NA>     <NA>     CHARLOT~      0.856 16.6   0.0517
## 3 STRAWBE~ CHARLOT~ RED RET~ <NA>     <NA>     WOODLAN~      0.774 22.1   0.0350
## 4 HERB MA~ <NA>     <NA>     <NA>     <NA>     HERB MA~      0.837 38.8   0.0216
## 5 STRAWBE~ CHARLOT~ WOODLAN~ <NA>     <NA>     RED RET~      0.751  7.40  0.102
## 6 JUMBO B~ JUMBO S~ <NA>     <NA>     <NA>     JUMBO B~      0.862 40.7   0.0212
```

FP Growth Scatter Plot

A scatter plot is an easy way to interpret the association rules graphically. Along the x-axis is `Support`, plotted against `Lift` in the y-axis, with `Confidence` being the 3rd dimension depicted through the color spectrum on the right of the graph

```
qplot(
  x = support,
  y = lift,
  color = confidence,
  data = GroceryRules,
  geom = "point",
  main = 'FP Growth Rules'
)
```



Network Diagram

A function to create the network diagram of the association rules from the FP Growth model is declared and defined below. The network diagram can be seen As stated before, much of the code to extract is derived from the work of Longhow Lam's tutorial.

Because the resulting interactive Network Diagram of the FP Growth Rules is a HTML Object with Javascript and CSS, it cannot be output into this PDF report. The Network Diagram is attached in the repository, and can be found here: https://github.com/patrick-osborne/CSML1000-Group_10-Assignment_2/tree/master/source/markdown/fpgrowth_sparklyr_network.html.

```r
####plot_rules plots the generated rules of FP Growth in a network graph
#rules is the rules generated by FP Growth model function in Sparklyr
#LHS is the name of the first antecedent column
#RHS is the name of the first consequent column
#cf is the desired filtering confidence score
#lft is the desired filtering lift score
#spt is the desired filtering support score

#much of this function can be found at Longhow Lam's tutorial
#@ https://longhowlam.wordpress.com/2017/11/23/association-rules-using-fpgrowth-in-spark-mllib-through-

plot_rules = function(rules, LHS = "LHSitem0", RHS = "RHSitem0", cf, lft, spt)
{
  #filter out rules based on confidence, lift, and support threshold parameterse
  rules = rules %>% filter(confidence > cf, lift > lft, support > spt)
  #grab unique rules
  nds = unique(
    c(
      rules[,LHS][[1]],
      rules[,RHS][[1]]
    )
  )

  #extract nodes from the dataframe unique rules
  nodes = data.frame(id = nds, label = nds, title = nds) %>% arrange(id)

  #calculate the edges based off the nodes
  edges = data.frame(
    from =  rules[,LHS][[1]],
    to = rules[,RHS][[1]]
  )

  #output network graph using the nodes and edges
  visNetwork(nodes, edges, main = "Online Purchases Network", width = "100%") %>%
    visOptions(highlightNearest = TRUE, nodesIdSelection = TRUE) %>%
    visEdges(smooth = FALSE) %>%
    visLayout(randomSeed = 1) #to keep the network look consistently the same
}


#plot a network graph for grocery rules
#we set filtering for the following parameters:
#confidence = 0.5
#lift = 1
#support = 0.01
#commented out, because we cannot output a HTML/JS object in PDF
# plot_rules(
#    GroceryRules,
#    cf = 0.5,
#    lft = 1,
#    spt = 0.01
#  )
```

ECLAT algorithm

Description

ECLAT stands for Equivalence Class Clustering and bottom-up Lattice Traversal. The ECLAT algorithm also works on datasets that are transactions. As already seen, the dataset has been converted into "transactions" and hence the model can be readily run. The ECLAT algorithm differs from the apriori algorithm by the fact that it is a depth first algorithm, where it mines for frequent subsets by traversing the length of each branch of transaction. It starts by creating a "frequent transaction set" for each item, with a minimum support threshold. Then the function is recursively called and in each recursion, more transactions are paired and continued till there are no more transactions to pair. The main difference between Apriori & ECLAT is, in Apriori, each item is listed and paired with every other item, with Support being the stop criterion, while in ECLAT, items are listed and paired only if a corresponding transaction of that pair exists.

Running ECLAT Algorithm

```r
#ECLAT algorithm
eclat.itemset <- eclat(trans, parameter = list(supp=0.01,maxlen=5))
eclat.rules <- ruleInduction(eclat.itemset, trans, confidence = 0.75)
eclat.rules_df<- data.frame(lhs = labels(lhs(eclat.rules)), rhs = labels(rhs(eclat.rules)),
                            eclat.rules@quality)
```

ECLAT Association Rules Result

The results of the ECLAT algorithm are association rules which are directly seen in R Console. For ease of understanding, the rules have been converted into a data frame and the head of the data frame is shown below.

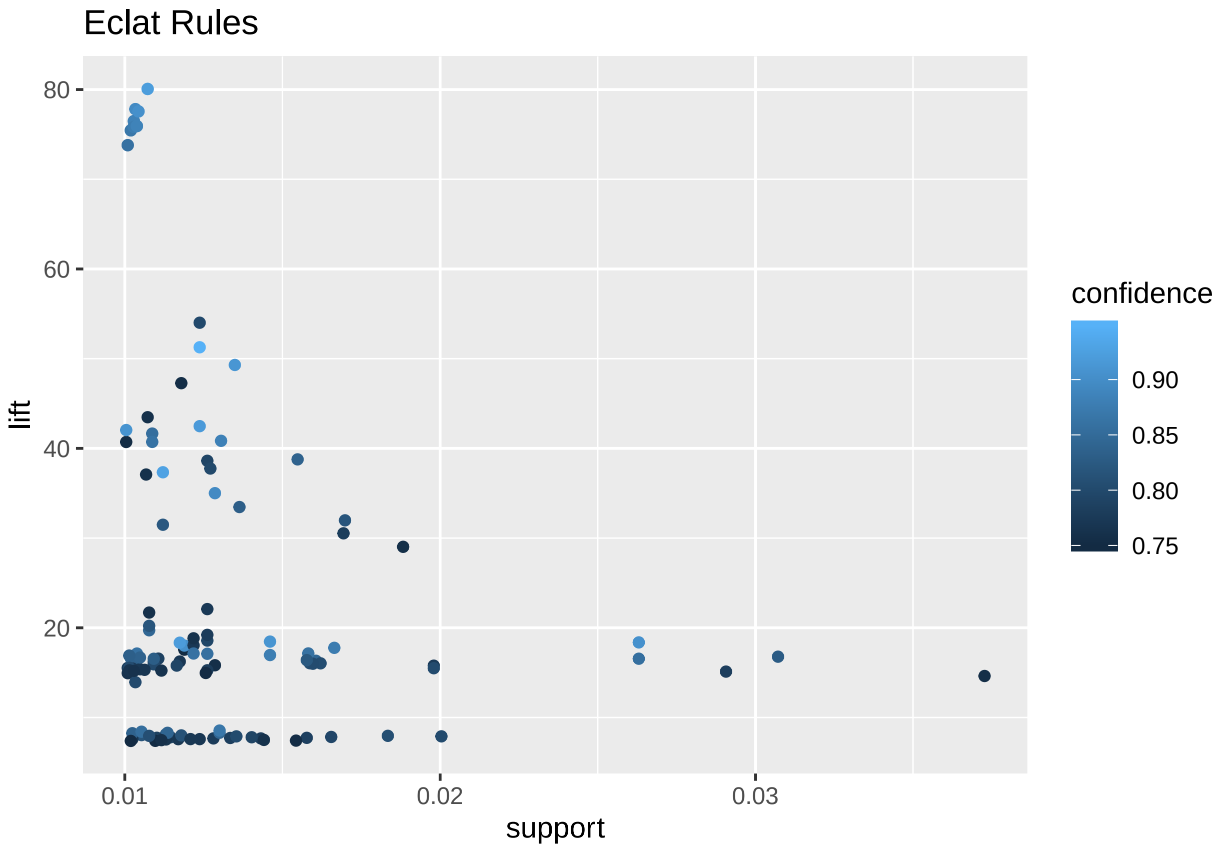
```r
head(eclat.rules_df)
```

```
##                                   lhs                               rhs
## 2    {CHILDRENS CUTLERY DOLLY GIRL }     {CHILDRENS CUTLERY SPACEBOY }
## 4  {CHILDRENS CUTLERY POLKADOT BLUE} {CHILDRENS CUTLERY POLKADOT PINK}
## 17            {HERB MARKER ROSEMARY}                {HERB MARKER MINT}
## 18                {HERB MARKER MINT}            {HERB MARKER ROSEMARY}
## 19                {HERB MARKER MINT}               {HERB MARKER BASIL}
## 20               {HERB MARKER BASIL}                {HERB MARKER MINT}
##       support confidence     lift itemset
## 2  0.01072451  0.7594502 43.47219       1
## 4  0.01067598  0.7612457 37.08508       2
## 17 0.01028777  0.8833333 75.84521       9
## 18 0.01028777  0.8833333 75.84521       9
## 19 0.01009366  0.8666667 73.79917      10
## 20 0.01009366  0.8595041 73.79917      10
```

Eclat Scatter Plot

A scatter plot is an easy way to interpret the association rules graphically. Along the x-axis is `Support`, plotted against `Lift` in the y-axis, with `Confidence` being the 3rd dimension depicted through the color spectrum on the right of the graph

```
#plot(eclat.rules, method = "scatterplot", main = "ECLAT Rules")
qplot(
  x = support,
  y = lift,
  color = confidence,
  data = eclat.rules_df,
  geom = "point",
  main = 'Eclat Rules'
)
```



Model Selection & Conclusion

The primary differentiation between Apriori & ECLAT is that the former is a “breadth-first” alogorithm and the latter is a “depth-first” algorithm. Since ECLAT is depth first, it requires less memory than Apriori algorithm. That also implies that ECLAT is a faster algorithm. To test this, the two algorithms’ execution time was measured for varying support levels. It was observed that with the given dataset, there isn’t a discernible difference between the two algorithms. In this case, therefore, both the algorithms are deployed in the final solution and the discretion to choose the algorithms is left to the user.

Apriori Runtime

```r
ap_start <- Sys.time()
apriori.rules <- apriori(trans, parameter = list(supp=0.04, conf=0.8, maxlen=5))
apriori.rules_df<- data.frame(lhs = labels(lhs(apriori.rules)), rhs = labels(rhs(apriori.rules))
                              , apriori.rules@quality)
ap_end <- Sys.time()
ap_time <- as.numeric(ap_end - ap_start)
```

ECLAT Runtime

```r
  eclat_start <- Sys.time()
  eclat.itemset <- eclat(trans, parameter = list(supp=0.04,maxlen=5))
  eclat.rules <- ruleInduction(eclat.itemset, trans, confidence = 0.8)
  eclat_end <- Sys.time()
  eclat_time <- as.numeric(eclat_end - eclat_start)
```

Table 2: Runtime comparison

Support	ap_time	eclat_time
0.01	0.403	1.826
0.02	0.623	0.697
0.03	0.271	0.562
0.04	0.320	0.504

While a runtime graph was not generated for FP Growth, it must be noted that a more powerful machine with a better CPU (Intel i7-8700 CPU @ 3.20 GHZ), and more RAM (16GB) had to be used in order to run the FP Growth Algorithm, as the tree-like structure it uses was sufficiently large enough to cause issues running the code in regular machines. In addition, there was not enough time to manipulate the output dataframe into a Rules class, which the Shiny app uses. Therefore, because of the need to use a machine with better specifications, as well as the lack of time needed to modify the output of the Sparklyr implementation of the FP Growth Algorithm into a suitable input for the Shiny app, the FP Growth algorithm was not deployed into the final solution.

Ethical Considerations

There are many cases of Machine Learning models lacking ethical considerations. From being insensitive to data changes to being downright inappropriate, it is a wide spectrum of issues to deal with. As Cathy O'Neil, a renowned mathematician explains, poorly built ML models can not only be insensitive to inequality & discrimination but sometimes reinforce them.

The team has done due diligence in ensuring that no part of it's work is unethical or reinforces and/or is insensitive to unethical practices. In the context of this project, the models built for generating product recommendations do not do the following as they are unnecessary to the objective and could potentially lead to discriminatory practices

- Not identify individual customers
- Not generate recommendations by Country of the transaction