An Approach to Optimize Manual Testing Using Apriori

White Paper

Deepak Singh Topwal

Topwal.Deepak.Singh@gmail.com

Linkedin.Com/in/DeepakSinghTopwal

September 2019

Contents

Abstract **3**

Introduction to Algorithm: Apriori **4**

- Introduction 4
- Measure of Rules 5

Data & Tools **6**

- Data Overview 6
- Statistical Tool: R 6

Solution Implementation **8**

- Installation 8
- Data Import 8
- Data Exploration & Interpretation 8
- Model Training 10
- Evaluation of Trained Model 11
- Interpretation of Rules 12
- Improving Model Performance 13

Conclusions **14**

- Finding Association among Test cases 14
- Preparing Optimized Test suites/Bucket using rules 15

Export Rules **17**

Appendix **18**

Acknowledgements & References **19**

Section 1

Abstract

Preparing test suites in manual testing is manual resource intensive task and often needs expertise that comes with experience. Testing all the test cases is also a time consuming activity and is not the best strategy as it needs intensive manual work which increases the overall cost.

An illustration to aid our understanding: Tester ‘A' has been working with a project for 10 years and knows by his experience which test cases are correlated such that changes in some might lead to deviations in others as well. Application functionality checks need these correlated tests to execute and test the application. However, what if this activity is assigned to a new tester who has no prior experience.
Such a situation makes testing challenging and may increase the number of defects as the new testers have no prior knowledge of test case dependencies or experience in preparing test suites.

This paper outlines a proven approach to achieve the following:

1. Using historical data to find the associations or dependencies of test cases without any prior knowledge or experience.
2. Preparing monthly Testing Suite/Buckets with optimum number of test cases to reduce the overall testing cost

Section **2**

Introduction to Algorithm: Apriori

Introduction

Apriori, Also known as Association Rules or Market Basket Analysis is an algorithm used for association rule learning .It was first introduced by Rakesh Aggarwal and Ramakrishnan Srikant in 1994.Apriori works on a simple a priori belief to reduce the association rule search space that all subsets of a frequent itemset must also be frequent. This heuristic is known as the Apriori property. Using this astute observation, it is possible to dramatically limit the number of rules to be searched. For example, the set {X, Y} can only be frequent if both {X} and {Y} occur frequently as well. Consequently, if either X or Y is infrequent, any set containing these items can be excluded from the search.

```
Apriori$(T, \epsilon)$
    $L_1 \leftarrow \{\text{large } 1-\text{itemsets}\}$
    $k \leftarrow 2$
    **while** $L_{k-1} \neq \emptyset$
        $C_k \leftarrow \{c = a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a, \{s \subseteq c \mid |s| = k-1\} \subseteq L_{k-1}\}$
        **for** transactions $t \in T$
            $D_t \leftarrow \{c \in C_k \mid c \subseteq t\}$
            **for** candidates $c \in D_t$
                $count[c] \leftarrow count[c] + 1$
        $L_k \leftarrow \{c \in C_k \mid count[c] \geq \epsilon\}$
        $k \leftarrow k + 1$
    **return** $\bigcup_k L_k$
```

Measure of Rules

1. **Support**:
   The support of an item set or rule measures how frequently it occurs in the data.
   For instance if item X appears in 6 out of total 10 data points so support for X will be 60% (6/10 = .6) as it appears in 60% of the dataset.

$$support(X) = \frac{count(x)}{N}$$

2. **Confidence**:
   Essentially, the confidence tells us the proportion of transactions where the presence of an item results in the presence of other item sets.
   For example if Confidence of item X and item Y is $confidence(X \rightarrow Y) = 0.70$, so this represents that item X is accompanied by item Y 70% of times.
   Or it can be understood this way that $confidence(Milk \rightarrow Bread) = 0.90$, represents that Item Bread is also bought 90% of times whenever item Milk is bought.

$$confidence(X \rightarrow Y) = \frac{support(X,Y)}{support(X)}$$

3. **Lift** :
   The lift of a rule measures how much more likely one item or item set is appeared relative to its typical rate of occurrence, given that you know another item or item set has been occurred .
   For example, suppose at a grocery store, most people purchase milk and bread.
   By chance alone, we would expect to find many transactions with both milk and bread.
   However, if lift (milk → bread) is greater than one, it implies that the two items are found together more often than one would expect by chance. A large lift value is therefore a strong indicator that a rule is important, and reflects a true connection between the items.
   This is defined by the following equation:

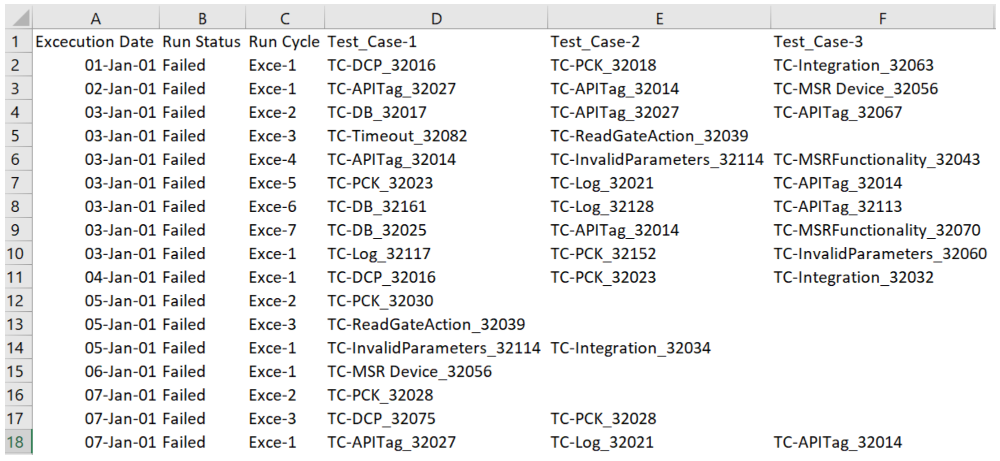
$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}$$

Section 3

Data & Tools

Data Overview

We have a testing data file consisting testing results for 11419 executions run during 2001-2016. Column A in data shows the date when the execution was run, Column B shows the Execution status, For this implementation we have only considered failed test cases as we are more interested in finding association for test cases which failed. Column C shows different run cycles (1, 2, 3 etc.) in which the testing was done and in other columns we have all the test cases which failed in testing



	A	B	C	D	E	F
1	Excecution Date	Run Status	Run Cycle	Test_Case-1	Test_Case-2	Test_Case-3
2	01-Jan-01	Failed	Exce-1	TC-DCP_32016	TC-PCK_32018	TC-Integration_32063
3	02-Jan-01	Failed	Exce-1	TC-APITag_32027	TC-APITag_32014	TC-MSR Device_32056
4	03-Jan-01	Failed	Exce-2	TC-DB_32017	TC-APITag_32027	TC-APITag_32067
5	03-Jan-01	Failed	Exce-3	TC-Timeout_32082	TC-ReadGateAction_32039	
6	03-Jan-01	Failed	Exce-4	TC-APITag_32014	TC-InvalidParameters_32114	TC-MSRFunctionality_32043
7	03-Jan-01	Failed	Exce-5	TC-PCK_32023	TC-Log_32021	TC-APITag_32014
8	03-Jan-01	Failed	Exce-6	TC-DB_32161	TC-Log_32128	TC-APITag_32113
9	03-Jan-01	Failed	Exce-7	TC-DB_32025	TC-APITag_32014	TC-MSRFunctionality_32070
10	03-Jan-01	Failed	Exce-1	TC-Log_32117	TC-PCK_32152	TC-InvalidParameters_32060
11	04-Jan-01	Failed	Exce-1	TC-DCP_32016	TC-PCK_32023	TC-Integration_32032
12	05-Jan-01	Failed	Exce-2	TC-PCK_32030		
13	05-Jan-01	Failed	Exce-3	TC-ReadGateAction_32039		
14	05-Jan-01	Failed	Exce-1	TC-InvalidParameters_32114	TC-Integration_32034	
15	06-Jan-01	Failed	Exce-1	TC-MSR Device_32056		
16	07-Jan-01	Failed	Exce-2	TC-PCK_32028		
17	07-Jan-01	Failed	Exce-3	TC-DCP_32075	TC-PCK_32028	
18	07-Jan-01	Failed	Exce-1	TC-APITag_32027	TC-Log_32021	TC-APITag_32014

Statistical Tool: R

R is a language and environment for statistical computing and graphics
R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering etc.) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

R Studio:

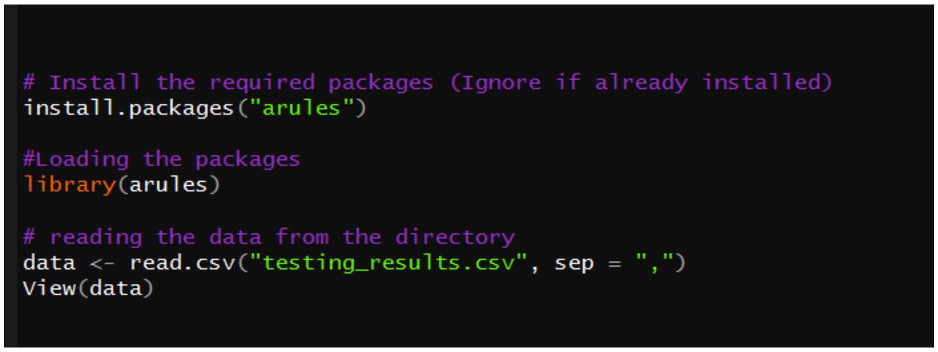
RStudio is a free, open source IDE (integrated development environment) for R. (You must install R before you can install RStudio.) Its interface is organized so that the user can clearly view graphs, data tables, R code, and output all at the same time. It also offers an Import-Wizard-like feature that allows users to import CSV, Excel, SAS (*.sas7bdat), SPSS (*.sav), and Stata (*.dta) files into R without having to write the code to do so.
A Free Desktop version of R Studio can be downloaded under the terms of the Free Software Foundation's GNU General Public License.

Section 4

Solution Implementation

Installation

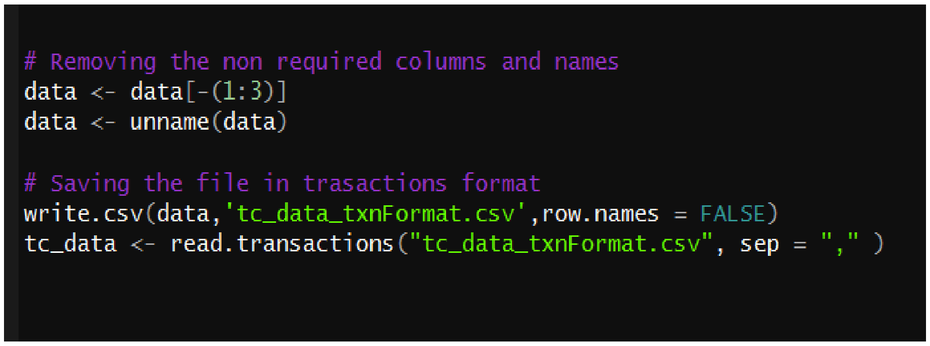
The following command installs and import the required package (arules) which will be used for association rules mining in R and then the dataset can be imported using read.csv command.



```
# Install the required packages (Ignore if already installed)
install.packages("arules")

#Loading the packages
library(arules)

# reading the data from the directory
data <- read.csv("testing_results.csv", sep = ",")
View(data)
```

Data Import

Then we remove the non-required features (Execution Data, Run Status & Run cycle) from data as these won't be used to apply Apriori.
Once we have data in transactions format, we can save and import it.

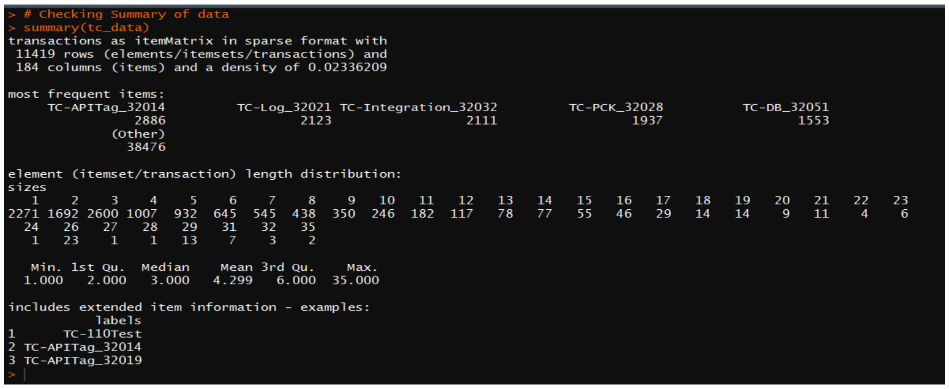


```
# Removing the non required columns and names
data <- data[-(1:3)]
data <- unname(data)

# Saving the file in trasactions format
write.csv(data,'tc_data_txnFormat.csv',row.names = FALSE)
tc_data <- read.transactions("tc_data_txnFormat.csv", sep = "," )
```

Data Exploration & Interpretation

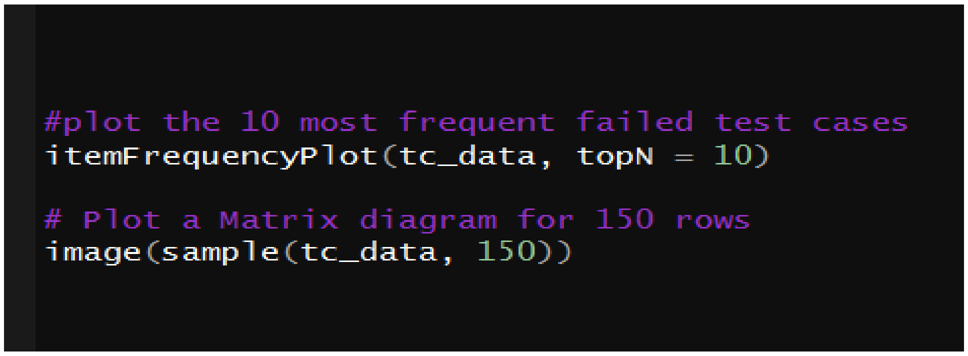
The following command is used to explore the data and gives a small summary about the sparse matrix we created.
The first block shows the number of rows and columns in dataset.
In our data we have 11419 testing rows and 184 different test cases.

The density value of 0.02336209 (Approx. 2.3%) refers to the proportion of nonzero matrix cells. The next tab shows the size of testing buckets, i.e. Number 15 shows that the numbers of executions where the data had 15 test cases were 55. The avg. cases per execution were approx. 4 (4.299) and the TC-APITag_32014 and TC-Log_32021 were the most frequent test cases in the data as shown in the 'Most frequent items' tab.

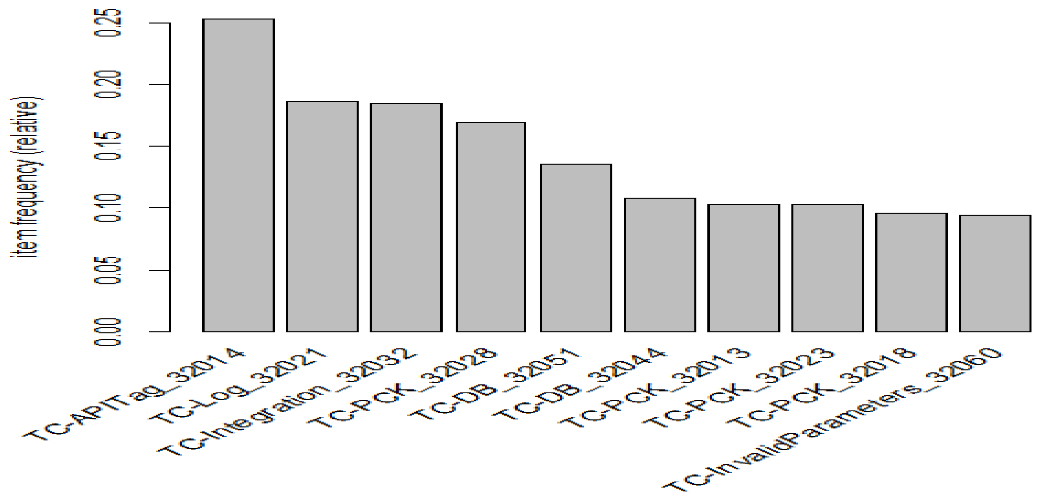


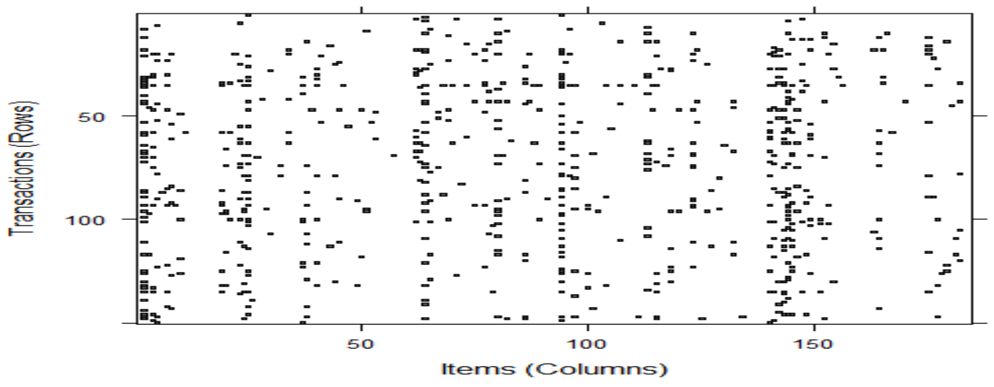
```
> # Checking Summary of data
> summary(tc_data)
transactions as itemMatrix in sparse format with
 11419 rows (elements/itemsets/transactions) and
 184 columns (items) and a density of 0.02336209

most frequent items:
     TC-APITag_32014         TC-Log_32021 TC-Integration_32032          TC-PCK_32028           TC-DB_32051
                2886                 2123                 2111                  1937                  1553
             (Other)
               38476

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23
2271 1692 2600 1007  932  645  545  438  350  246  182  117   78   77   55   46   29   14   14    9   11    4    6
  24   26   27   28   29   31   32   35
   1   23    1    1   13    7    3    2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   4.299   6.000  35.000

includes extended item information - examples:
           labels
1       TC-110Test
2 TC-APITag_32014
3 TC-APITag_32019
> 
```

Next we visualize our dataset to see which are the test cases which frequently fail with the help of a Bar chart. Next we create a Matrix Diagram for top 150 rows to visualize the distribution of test cases per execution. And from the charts we can analyze that Test cases 32014, 32021 and 32032 are more prone to failure and failed most of the times.



```
#plot the 10 most frequent failed test cases
itemFrequencyPlot(tc_data, topN = 10)

# Plot a Matrix diagram for 150 rows
image(sample(tc_data, 150))
```

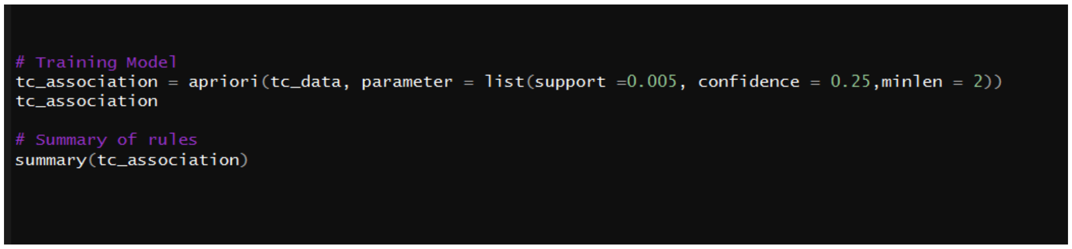




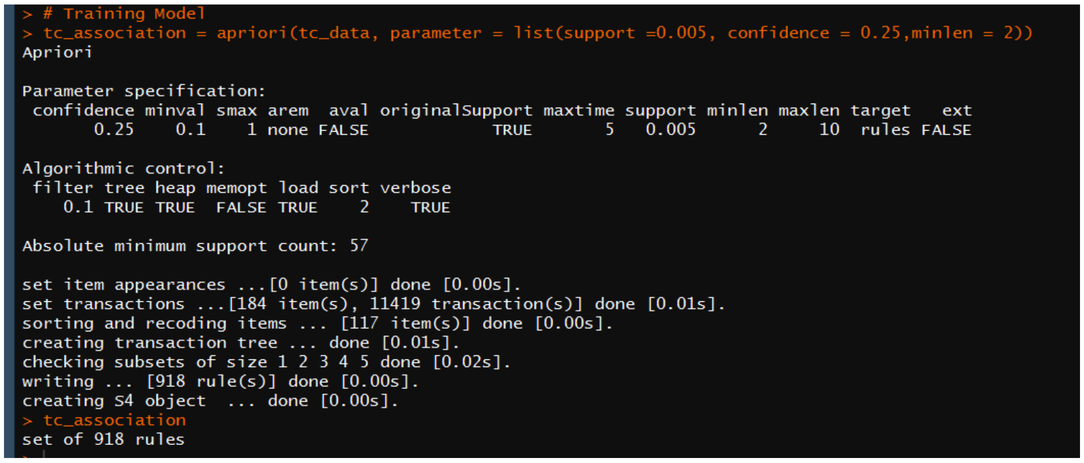
Model Training

Once data preparation is complete we can now work on training our Apriori model and finding the association rules for same.
As parameters we are taking threshold to get association rules with minlen = 2, which will consider only execution where at least 2 test cases were run ,with support and confidence thresholds of 5% and 25%.

And once the model is trained we can see the number of rules were extracted by it. In our case our model has 918 association rules
Note: The threshold values can be changed as per requirement



```
# Training Model
tc_association = apriori(tc_data, parameter = list(support =0.005, confidence = 0.25,minlen = 2))
tc_association

# Summary of rules
summary(tc_association)
```

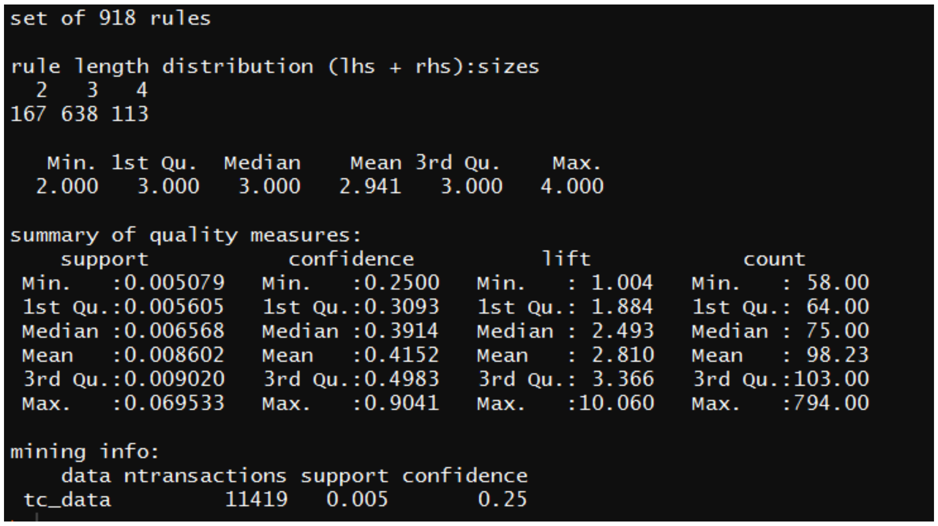


```
> # Training Model
> tc_association = apriori(tc_data, parameter = list(support =0.005, confidence = 0.25,minlen = 2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
       0.25    0.1    1 none FALSE            TRUE       5   0.005      2     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 57

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[184 item(s), 11419 transaction(s)] done [0.01s].
sorting and recoding items ... [117 item(s)] done [0.00s].
creating transaction tree ... done [0.01s].
checking subsets of size 1 2 3 4 5 done [0.02s].
writing ... [918 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> tc_association
set of 918 rules
```

Evaluation of Trained Model

To obtain a high-level overview of the association rules, we can use summary() command. The rule length distribution tells us about the items /rule. In our rules set, 167 rules have only two items, while 638 have three, and 113 have four. The summary statistics associated with this distribution are also given:

Next, we see the summary statistics of the rule quality measures: support, confidence, and lift. We might be alarmed if most or all of the rules had support and confidence very near the minimum thresholds, as this would mean that we may have set the bar too high. This is not the case here, as there are many rules with much higher values of each.
And next we see the mining info which tells us the criteria on which the rules were extracted. In our case it will be same as the threshold we set for Support and confidence



```
set of 918 rules

rule length distribution (lhs + rhs):sizes
  2   3   4
167 638 113

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   3.000   3.000   2.941   3.000   4.000

summary of quality measures:
    support           confidence          lift            count
 Min.   :0.005079   Min.   :0.2500   Min.   : 1.004   Min.   : 58.00
 1st Qu.:0.005605   1st Qu.:0.3093   1st Qu.: 1.884   1st Qu.: 64.00
 Median :0.006568   Median :0.3914   Median : 2.493   Median : 75.00
 Mean   :0.008602   Mean   :0.4152   Mean   : 2.810   Mean   : 98.23
 3rd Qu.:0.009020   3rd Qu.:0.4983   3rd Qu.: 3.366   3rd Qu.:103.00
 Max.   :0.069533   Max.   :0.9041   Max.   :10.060   Max.   :794.00

mining info:
    data ntransactions support confidence
 tc_data         11419   0.005       0.25
```

Interpretation of Rules

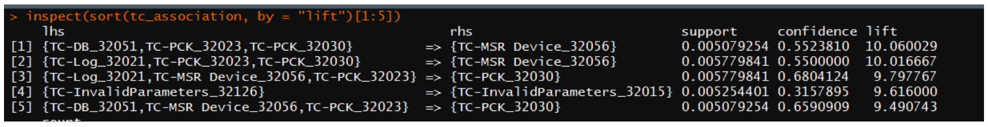
The result of a market basket analysis is a collection of association rules that specify patterns found in the relationships among item sets.
Association rules are always composed from subsets of itemset and are denoted by relating one itemset on the left-hand side (LHS) of the rule to another itemset on the right-hand side (RHS) of the rule. The LHS is the condition that needs to be met in order to trigger the rule, and the RHS is the expected result of meeting that condition. A rule identified from the example transaction might be expressed in the form:

$$\{Test\ case\ X, Test\ case\ Y\} \rightarrow \{Test\ Case\ M\}$$

Which can be interpreted as whenever the Test Case X and Test Case Y Failed, Test Case M also resulted in a failure.
This will be briefed in details in next section “Finding Association among Test cases”

Improving Model Performance

It’s always not easy to inspect thousands of rules and to select them .Therefor it’s a good strategy to sort the rules as per our interest on different criteria’s and extract the strong ones. In this way, we can improve the performance of our model by making the results more actionable
Depending on our interest we can extract the rules specifying the specific parameter (i.e. Support, Lift & Confidence) and can sort them as per their associations.
We can use Sort command to do this.
In the following command we are going to view top 5 rules which have highest lift so that we can first list the rules which have higher association.
And as we can see the first rule has 10 times higher lift value which makes it very strong association rule for TC 32051, 32023 and 32030.
Similarly we can view other rules as per Support and confidence too.



```
> inspect(sort(tc_association, by = "lift")[1:5])
    lhs                                                 rhs                             support     confidence lift
[1] {TC-DB_32051,TC-PCK_32023,TC-PCK_32030}          => {TC-MSR Device_32056}           0.005079254 0.5523810  10.060029
[2] {TC-Log_32021,TC-PCK_32023,TC-PCK_32030}         => {TC-MSR Device_32056}           0.005779841 0.5500000  10.016667
[3] {TC-Log_32021,TC-MSR Device_32056,TC-PCK_32023} => {TC-PCK_32030}                  0.005779841 0.6804124   9.797767
[4] {TC-InvalidParameters_32126}                     => {TC-InvalidParameters_32015} 0.005254401 0.3157895   9.616000
[5] {TC-DB_32051,TC-MSR Device_32056,TC-PCK_32023}  => {TC-PCK_32030}                  0.005079254 0.6590909   9.490743
```

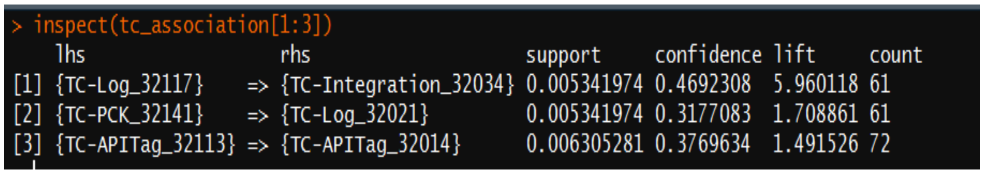
Section 5

Conclusions

Finding Association among Test cases

On inspecting the rules we can see the first rule which states association of Test case 32117 with Test case 32014 with support of 0.005341074, which tells that it occurred in approx. 5% of all executions,
Confidence of 0.4692308 shows whenever test case 32117 failed it also resulted in failure of Test case 32034 approx. 47% of times.
And high lift value of 5.960118 reflects that how strongly these 2 Test cases are associated or dependent on each other and chances of Test case 32117 and Test case 32014 failing together is 6 Times higher than test case 32117 failing alone.
Which shows a very strong association of these 2 test cases.

In the similar way we can extract other rules sorting them by Support, lift or confidence with set thresholds.
For example all the 918 rules may not be very interesting or may not have such strong association so we can set a threshold to consider rules only where we have lift >=2 or confidence >=0.60.



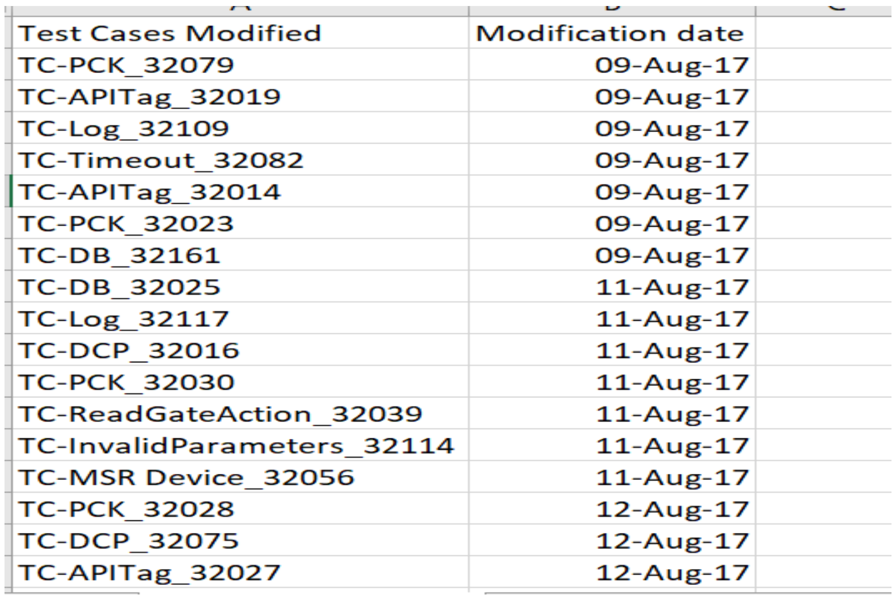
```
> inspect(tc_association[1:3])
    lhs                   rhs                    support     confidence lift     count
[1] {TC-Log_32117}     => {TC-Integration_32034} 0.005341974 0.4692308  5.960118 61
[2] {TC-PCK_32141}     => {TC-Log_32021}         0.005341974 0.3177083  1.708861 61
[3] {TC-APITag_32113} => {TC-APITag_32014}      0.006305281 0.3769634  1.491526 72
```

Preparing Optimized Test Suites using rules

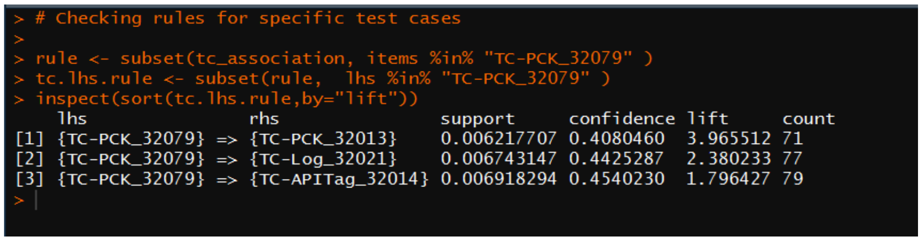
Once the association of test cases has been analyzed, we can use this information for creating and optimizing our Testing Suite every month.
For example we have a file 'Tc modification log.csv' which contains the name of Test cases which were updated/modified in last cycle for development and now need to be checked if they are functioning properly.
But there would also be some other test cases which will have dependency on these and since these were modified so it also becomes important to test other dependent test cases in order to make the application function properly.

Generally finding such Test cases needs one's experience and expertise to know which Test case is related to which one, but using association rules this can also be done without having any prior experience of knowledge.
The next part explains how this can be done using the association rules we extracted from the dataset.

In our case we have 22 test cases which were modified recently.
So we can pick first test case 'TC-PCK_32079' as we are interested in seeing which other test cases are associated or dependent with this specific one. So we run the following command which will extract the only rules for us where TC 32079 was executed and failed



Test Cases Modified	Modification date
TC-PCK_32079	09-Aug-17
TC-APITag_32019	09-Aug-17
TC-Log_32109	09-Aug-17
TC-Timeout_32082	09-Aug-17
TC-APITag_32014	09-Aug-17
TC-PCK_32023	09-Aug-17
TC-DB_32161	09-Aug-17
TC-DB_32025	11-Aug-17
TC-Log_32117	11-Aug-17
TC-DCP_32016	11-Aug-17
TC-PCK_32030	11-Aug-17
TC-ReadGateAction_32039	11-Aug-17
TC-InvalidParameters_32114	11-Aug-17
TC-MSR Device_32056	11-Aug-17
TC-PCK_32028	12-Aug-17
TC-DCP_32075	12-Aug-17
TC-APITag_32027	12-Aug-17



```
> # Checking rules for specific test cases
>
> rule <- subset(tc_association, items %in% "TC-PCK_32079" )
> tc.lhs.rule <- subset(rule,  lhs %in% "TC-PCK_32079" )
> inspect(sort(tc.lhs.rule,by="lift"))
    lhs                 rhs                 support     confidence lift     count
[1] {TC-PCK_32079} => {TC-PCK_32013}    0.006217707 0.4080460  3.965512 71
[2] {TC-PCK_32079} => {TC-Log_32021}    0.006743147 0.4425287  2.380233 77
[3] {TC-PCK_32079} => {TC-APITag_32014} 0.006918294 0.4540230  1.796427 79
>
```

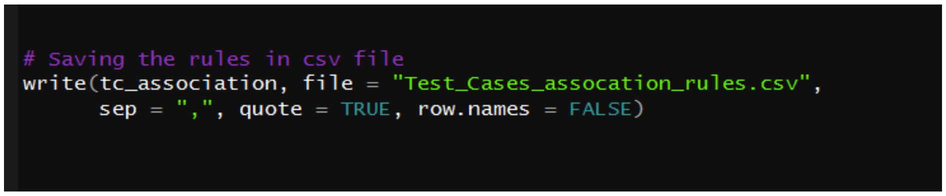
We can see this shows 3 rules with Test cases 32013, 32021 and 32014 which also failed 41%-45% of times whenever test case 32079 failed, with the lift value > 1.5.
So we can conclude these test cases also have a strong association and dependency on TC 32079 and also prone to failure if we have made any changes to TC 32079. So these should also be tested and included in our testing suite to avoid any defect or application failure.
So we can pick these 3 Test cases as part of our testing bucket.
And we can do same for other remaining 21 test case and create a testing suite with an optimized number of test cases.

Sometimes it will also not be feasible to include all the test cases shown in rules so in such cases we can pick the strong rules only with setting thresholds (I.E rules where lift value > 2 only) and reduce the number of test cases in testing suite.

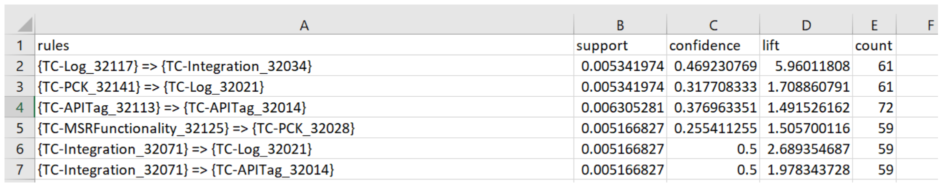
Section 6

Export Rules

To share the results of our Association Rules analysis, we can save the rules to a CSV file with the write() function. This will produce a CSV file that can be used in most spreadsheet programs including Microsoft Excel and can be shared with anyone later on for use.
We can use R's write function for this.



```
# Saving the rules in csv file
write(tc_association, file = "Test_Cases_assocation_rules.csv",
      sep = ",", quote = TRUE, row.names = FALSE)
```



	A	B	C	D	E	F
1	rules	support	confidence	lift	count	
2	{TC-Log_32117} => {TC-Integration_32034}	0.005341974	0.469230769	5.96011808	61	
3	{TC-PCK_32141} => {TC-Log_32021}	0.005341974	0.317708333	1.708860791	61	
4	{TC-APITag_32113} => {TC-APITag_32014}	0.006305281	0.376963351	1.491526162	72	
5	{TC-MSRFunctionality_32125} => {TC-PCK_32028}	0.005166827	0.255411255	1.505700116	59	
6	{TC-Integration_32071} => {TC-Log_32021}	0.005166827	0.5	2.689354687	59	
7	{TC-Integration_32071} => {TC-APITag_32014}	0.005166827	0.5	1.978343728	59	

Section 7

Appendix

1) Installations :
   - R :
     https://cran.r-project.org/bin/windows/base/

   - R Studio :
     https://www.rstudio.com/products/rstudio/download/

2) R Project Files :
   - Dataset & Code
     https://github.com/deepaksinghtopwal/Test_Suit_Optimization_using_Apriori

Section **8**

Acknowledgements & References

1) Study of Apriori & Association rules mining
    - Apriori :
      Rakesh Agrawal and Ramakrishnan Srikant
      Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

    - Apriori Algorithm:
      https://en.wikipedia.org/wiki/Apriori_algorithm

2) Tools :
    - Overview – R
      https://www.r-project.org/about.html

    - R Studio
      https://libguides.library.kent.edu/statconsulting/r

    - Apriori R documentation :
      https://www.rdocumentation.org/packages/arules/versions/1.6-3/topics/apriori