Prediction of Missing Items in Shopping Cart

Submitted in partial fulfillment

Of

Mini project in Bachelor of Technology

Submitted by

1. **Karthikeya K (411530)**
2. **Ravi Tej M (411544)**
3. **Naveen Reddy G (411525)**

Under the Supervision of
Mr.BKSP Kumar Raju Alluri

NATIONAL INSTITUTE OF TECHNOLOGY,
Andhra Pradesh.

SUMMARY

- Existing research in association mining has focused mainly on how to expedite the search for frequently co-occurring groups of items in “shopping cart” type of transactions.
- This project contributes to the latter task by proposing a technique that uses partial information about the contents of a shopping cart for the prediction of what else the customer is likely to buy i.e. predicting the missing items in shopping cart by calculating the current transaction class and cross checking input items of the transactions with the antecedents of the association rules generated of that particular class.
- It also suggests items based on the offers available at that particular time based on the information of items of the current transaction.
- Our project also recommends some items based on previous transactions of a particular customer i.e. user based prediction.
- The whole project is implemented by a simple user interface where inputs such as customer id, items along with their quantity is taken and predicts items based on generated association rules, offers available and also customers previous history.

1. INTRODUCTION

1.1 Data Mining

Data Mining refers to extracting or mining information from large amounts of data. Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. Data mining, “the extraction of hidden predictive information from large databases”, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions.

1.2 Association Rule Mining

Association Rule Mining is a popular and well researched method for discovering interesting relations between variables in large databases. Association rules are statements of the form {X1, X2, …, Xn} => Y meaning that if all of X1, X2,… Xn is found in the market basket, and then we have good chance of finding Y. the probability of finding Y for us to accept this rule is called the confidence of the rule. Normally rules that have a confidence above a certain threshold only will be searched. In many situations, association rules involves sets of items that appear frequently. For example, a good marketing strategy cannot be run involving items that no one buys. Thus, much data mining starts with the assumption that sets of items with support are only considered. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customer and which items bring them better profits when placed with in close proximity.

1.3 Classification

Classification is a form of data analysis that extracts models describing important data classes. Such models, called classifiers, predict categorical (discrete, unordered) class labels. For example, we can build a classification model to categorize bank loan applications as either safe or risky. Such analysis can help provide us with a better understanding of the data at large. Many classification methods have been proposed by researchers in machine learning, pattern recognition, and statistics. Most algorithms are memory resident, typically assuming a small data size. Recent data mining research has built on such work, developing scalable classification and prediction techniques capable of handling large amounts of disk-resident data. Classification has numerous applications, including fraud detection, target marketing, performance prediction, manufacturing, and medical diagnosis.

1.4 Clustering

Clustering is the process of grouping a set of data objects into multiple groups or *clusters* so that objects within a cluster have high similarity, but are very dissimilar to objects in other clusters. Dissimilarities and similarities are assessed based on the attribute values describing the objects and often involve distance measures. Clustering as a data mining tool has its roots in many application areas such as biology, security, business intelligence, and Web search.

In this project we have used Classification to classify items based on its price into 3 classes (class 1,2,3) by J48 algorithm and also used Association rule mining by apriori algorithm to generate rules on previous transactions so that we can predict missing items from customers shopping cart based on those rules.

2. DATA SET

We got the Dataset from kaggle.
LINK: https://www.kaggle.com/karthickveerakumar/orders-data

It consists of 5000 tuples (transactions) with 138 attributes. Among those attributes there are 134 items along with order id, order hour of day and days since prior order. In a transaction if any item is not bought then 0 (zero) is marked under that attribute and if an item is bought, its quantity is marked under that attribute. A new dataset is created with attributes item name and price and all 134 items along with its price is updated. Another dataset of transactions along with customer id is generated with same items (134) for user based recommendations.

3. OBJECTIVE

The main objective of this project is to predict the missing items in the shopping cart using association rule mining and classification. Those missing items can be predicted from the rules generated from previous data, customer’s previous history and also from offers available.

4. SCHEMATIC VIEW OF MODEL

4.1 Shopping cart prediction Architecture

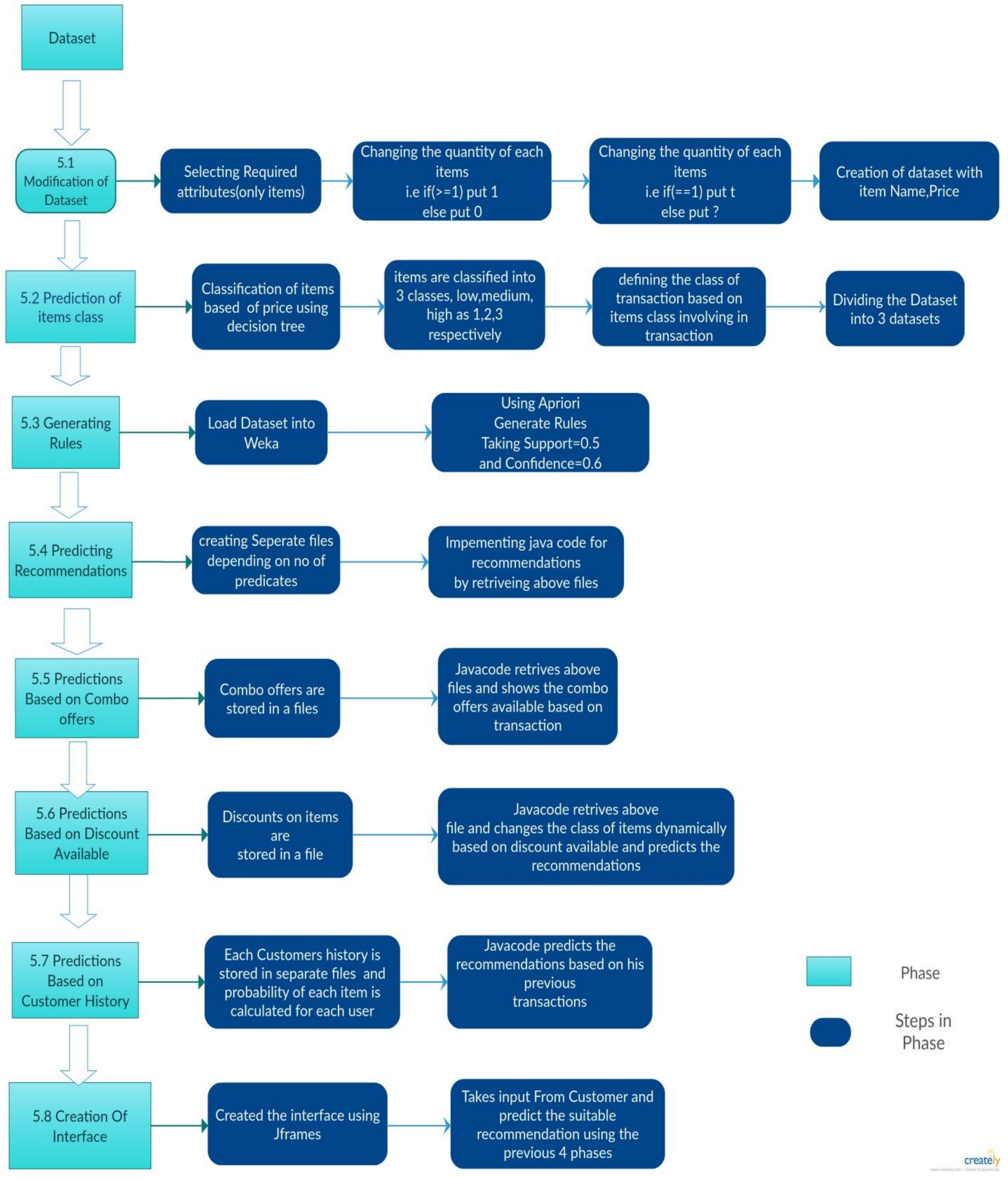

Dataset
5.1 Modification of Dataset
Selecting Required attributes(only items)
Changing the quantity of each items i.e if(>=1) put 1 else put 0
Changing the quantity of each items i.e if(==1) put t else put ?
Creation of dataset with item Name,Price
5.2 Prediction of items class
Classification of items based of price using decision tree
items are classified into 3 classes, low,medium, high as 1,2,3 respectively
defining the class of transaction based on items class involving in transaction
Dividing the Dataset into 3 datasets
5.3 Generating Rules
Load Dataset into Weka
Using Apriori Generate Rules Taking Support=0.5 and Confidence=0.6
5.4 Predicting Recommendations
creating Seperate files depending on no of predicates
Impementing java code for recommendations by retriveing above files
5.5 Predictions Based on Combo offers
Combo offers are stored in a files
Javacode retrives above files and shows the combo offers available based on transaction
5.6 Predictions Based on Discount Available
Discounts on items are stored in a file
Javacode retrives above file and changes the class of items dynamically based on discount available and predicts the recommendations
5.7 Predictions Based on Customer History
Each Customers history is stored in separate files and probability of each item is calculated for each user
Javacode predicts the recommendations based on his previous transactions
5.8 Creation Of Interface
Created the interface using Jframes
Takes input From Customer and predict the suitable recommendation using the previous 4 phases
Phase
Steps in Phase
creately

Figure 4.1.1---Basic architecture of our project

4.2 Use case diagram

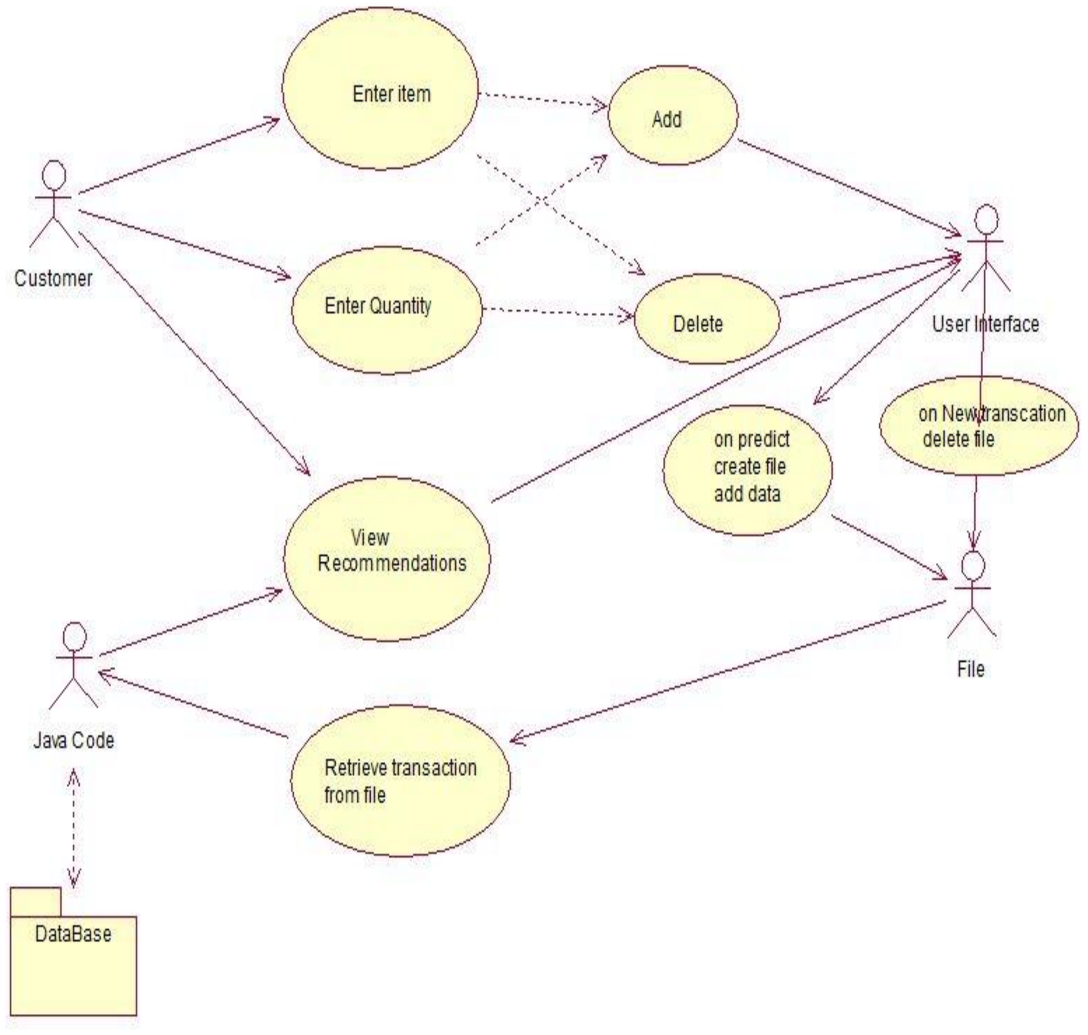
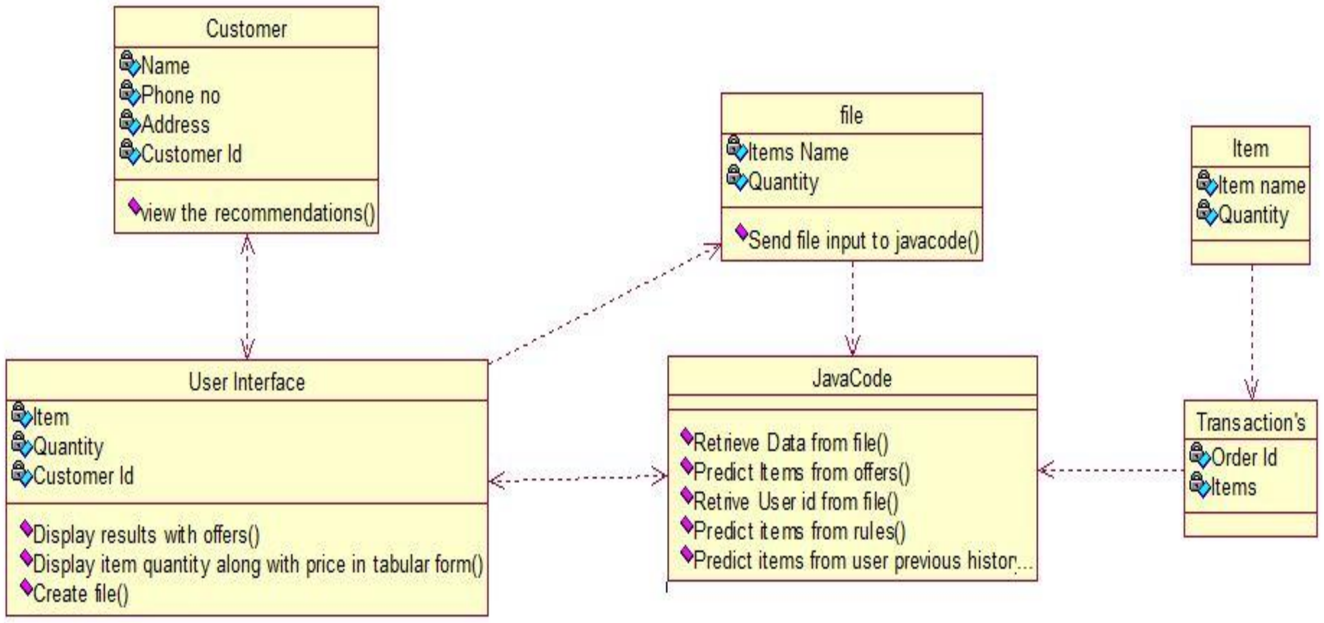
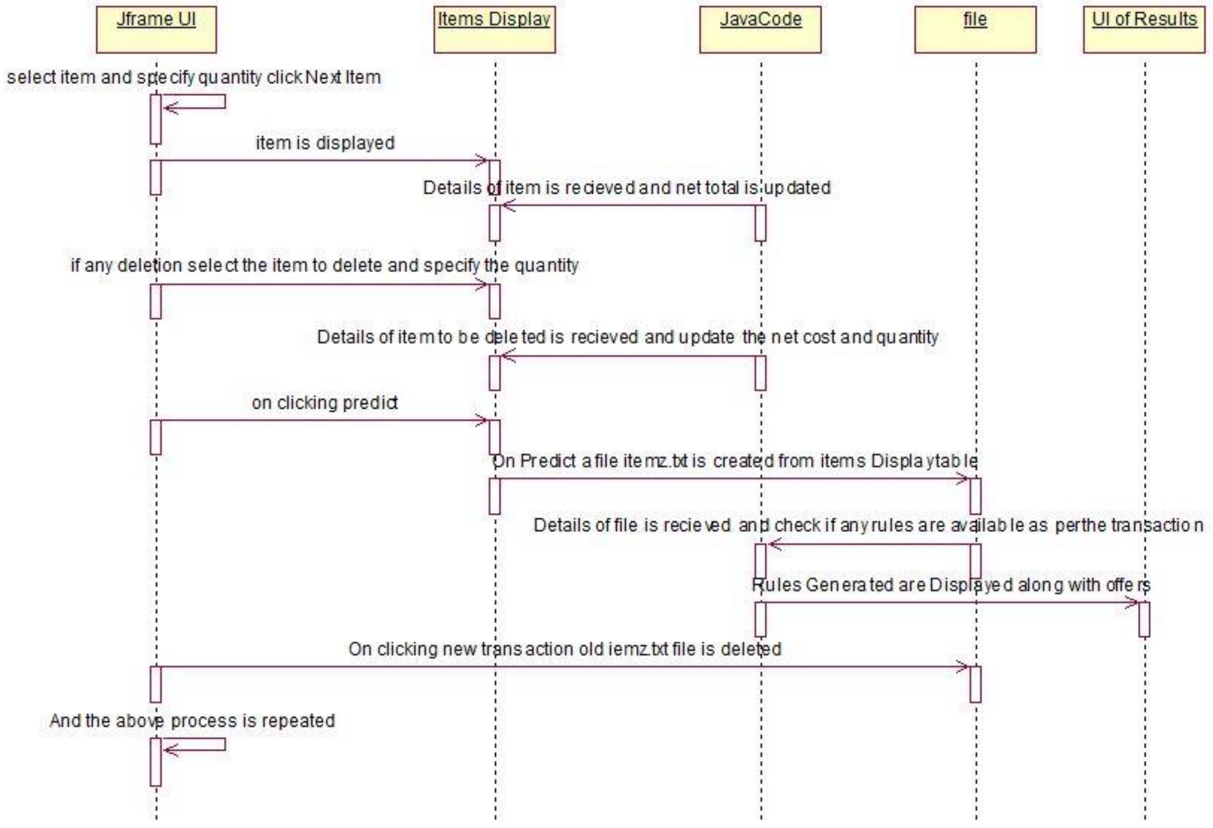


Figure 4.2.1---Use case diagram of our project

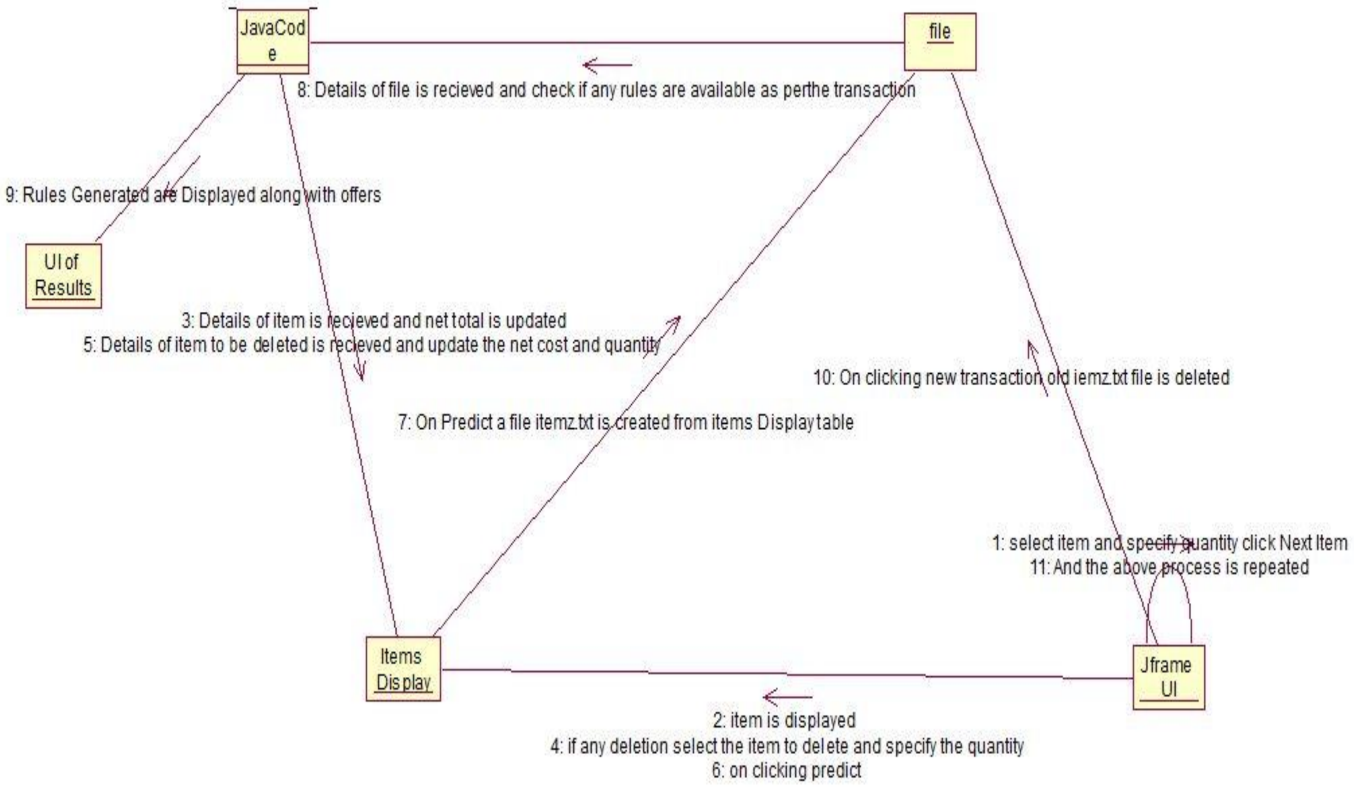
4.3 Class diagram


Customer
Name
Phone no
Address
Customer Id
view the recommendations()
file
Items Name
Quantity
Send file input to javacode()
Item
Item name
Quantity
User Interface
Item
Quantity
Customer Id
Display results with offers()
Display item quantity along with price in tabular form()
Create file()
JavaCode
Retrieve Data from file()
Predict Items from offers()
Retrive User id from file()
Predict items from rules()
Predict items from user previous histor...
Transaction's
Order Id
Items

4.4 Sequence diagram


Jframe UI
Items Display
JavaCode
file
UI of Results
select item and specify quantity click Next Item
item is displayed
Details of item is recieved and net total is updated
if any deletion select the item to delete and specify the quantity
Details of item to be deleted is recieved and update the net cost and quantity
on clicking predict
On Predict a file itemz.txt is created from items Displaytable
Details of file is recieved and check if any rules are available as perthe transaction
Rules Generated are Displayed along with offers
On clicking new transaction old iemz.txt file is deleted
And the above process is repeated

4.5 Collaboration diagram


JavaCode
file
8: Details of file is recieved and check if any rules are available as perthe transaction
9: Rules Generated are Displayed along with offers
UI of Results
3: Details of item is recieved and net total is updated
5: Details of item to be deleted is recieved and update the net cost and quantity
10: On clicking new transaction old iemz.txt file is deleted
7: On Predict a file itemz.txt is created from items Display table
1: select item and specify quantity click Next Item
11: And the above process is repeated
Items Display
Jframe UI
2: item is displayed
4: if any deletion select the item to delete and specify the quantity
6: on clicking predict

5. EXPLAINATION OF MODEL (figure 4.1.1)

5.1 Modification of dataset

Removed attributes from our dataset which are not useful for association rule mining such as order id, hour of day etc. Then we have replaced the quantity of items in transactions which are greater than 1 by 1(since to generate association rules in weka we need the presence of item in the transaction but not the quantity). Next we have replaced all 1's in transaction by 't' and 0's by '?' .

We created a dataset (items with price) with attributes item name and price for all 134 items and wrote its price in the data set.

5.2 Classification of Items and Transactions

Classified dataset of items and price into 3 classes (class 1, 2, 3) i.e. into low, medium and high by using J48 decision tree algorithm. Now classified transactions into 3 classes based on its items and divided the transaction dataset into 3 datasets of class 1, class 2, and class 3.

5.3 Generating Association rules

Loaded dataset of transactions consisting of t's and ?'s into weka and generated association rules using apriori algorithm for each class and then stored the generated rules into database.

5.4 Prediction based on generated association rules

We will calculate the class of the current transaction based on items of the transaction and then predict the missing items by checking the current transaction items with the antecedent of generated rules of same class.

5.5 Prediction based on combo offers

All combo offers are loaded into database and every item of current transaction is crosschecked with the offer and if any offer is available then we'll predict the offered item and tell to the customer.

5.6 Prediction based on discount available

If any item has a discount offer such as 50% off and if current transaction is having that item then we'll reduce the class of that item by 1 (if its original class is greater than 1 and discount is greater than 50%) and then calculate the current transaction class.

5.7 Prediction based on Customer history

We will store all the previous transactions of customers and predict missing items from them. We will calculate probability of a particular customer buying an item and if the probability is greater than 0.5 and if that item is missing in his/her cart then we'll recommend that item to the customer(user based prediction).

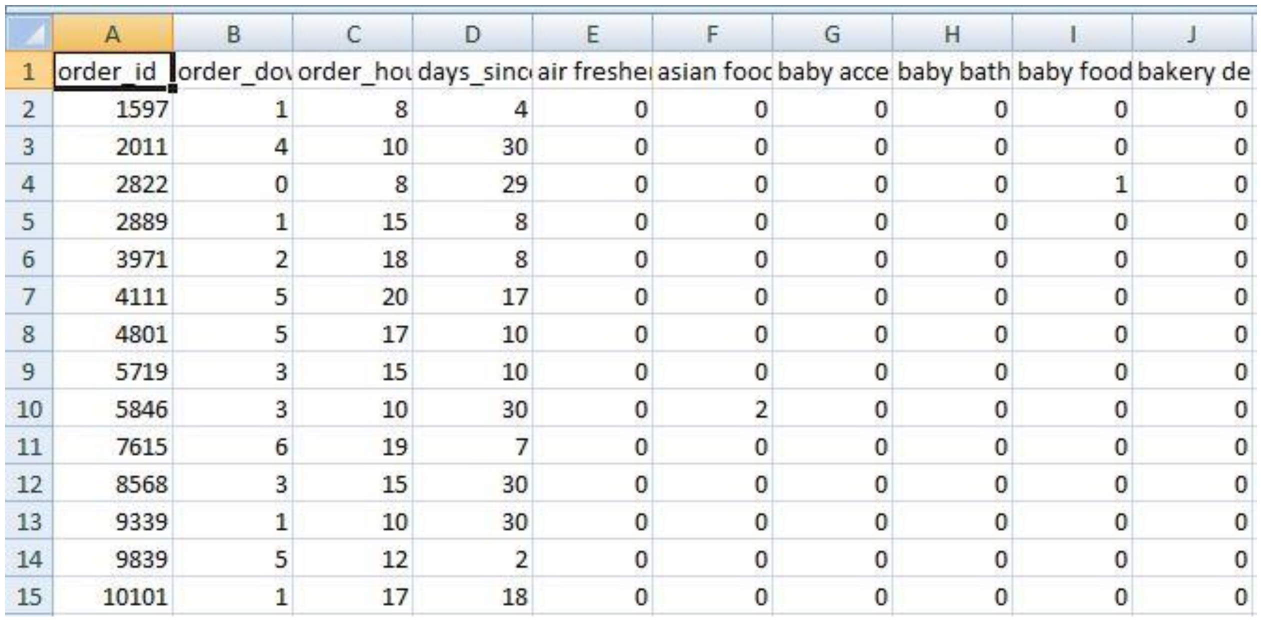
5.8 Creation of interface using JFrames

We have created an interface using Jframes which will take inputs such as customer id, item names along with its quantity and can also delete an item or can reduce its quantity and invoke the java code which would predict missing items.

6. EXPERIMENTS AND EVALUATION

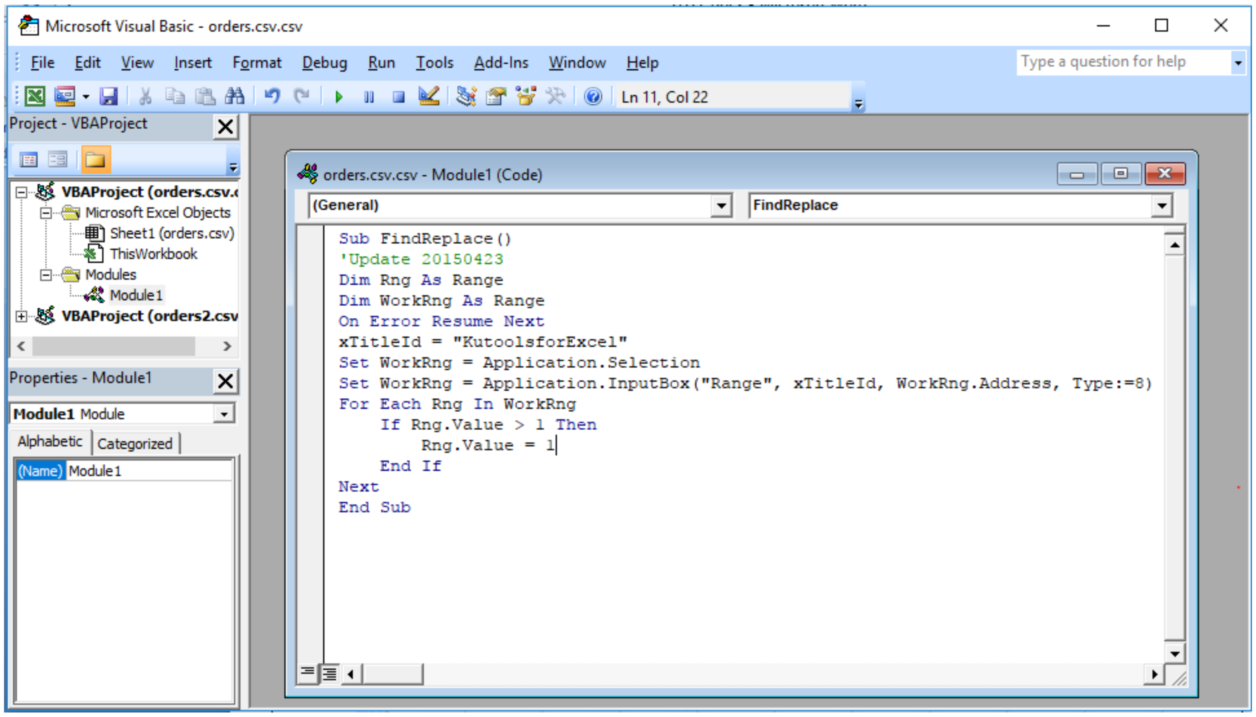
6.1 Implementation of basic module

After downloading the dataset we have removed the attributes which are not used for generating association rules such as order id, hour of day etc. Then we have replaced the quantity of items which are greater than 1 by 1 by inserting a module of code in excel as shown in below figure.



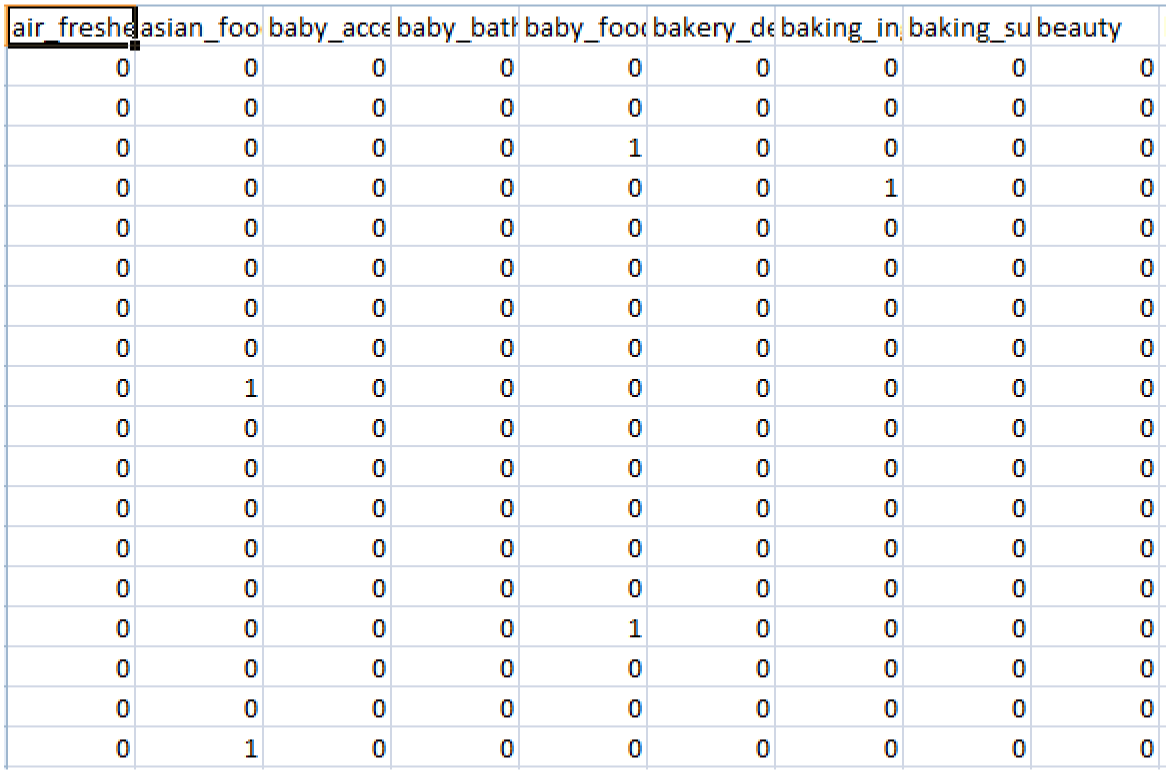
	A	B	C	D	E	F	G	H	I	J
1	order_id	order_dov	order_hou	days_sinc	air freshe	asian fooc	baby acce	baby bath	baby food	bakery de
2	1597	1	8	4	0	0	0	0	0	0
3	2011	4	10	30	0	0	0	0	0	0
4	2822	0	8	29	0	0	0	0	1	0
5	2889	1	15	8	0	0	0	0	0	0
6	3971	2	18	8	0	0	0	0	0	0
7	4111	5	20	17	0	0	0	0	0	0
8	4801	5	17	10	0	0	0	0	0	0
9	5719	3	15	10	0	0	0	0	0	0
10	5846	3	10	30	0	2	0	0	0	0
11	7615	6	19	7	0	0	0	0	0	0
12	8568	3	15	30	0	0	0	0	0	0
13	9339	1	10	30	0	0	0	0	0	0
14	9839	5	12	2	0	0	0	0	0	0
15	10101	1	17	18	0	0	0	0	0	0

Figure 6.1.1---Original Dataset



```
Sub FindReplace()
'Update 20150423
Dim Rng As Range
Dim WorkRng As Range
On Error Resume Next
xTitleId = "KutoolsforExcel"
Set WorkRng = Application.Selection
Set WorkRng = Application.InputBox("Range", xTitleId, WorkRng.Address, Type:=8)
For Each Rng In WorkRng
    If Rng.Value > 1 Then
        Rng.Value = 1
    End If
Next
End Sub
```

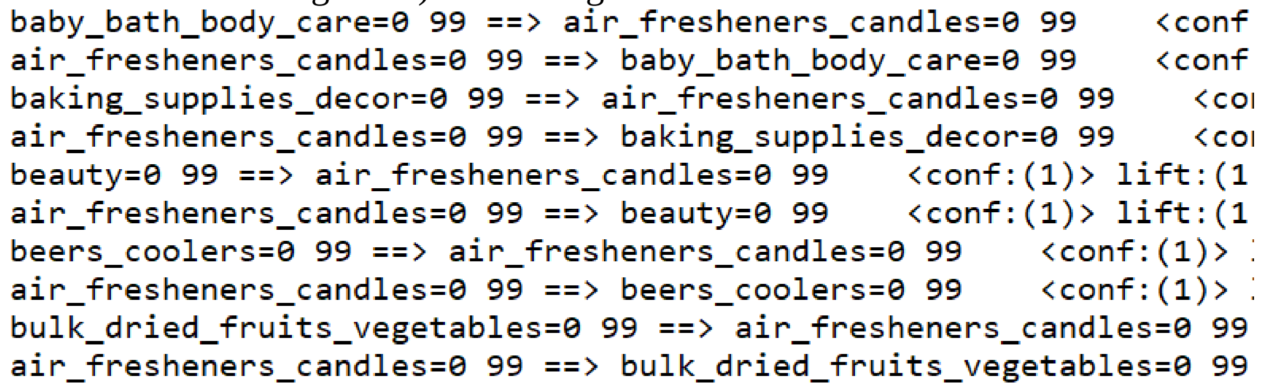
Code 6.1.1---Code for conversion



air_freshe	asian_foo	baby_acce	baby_batl	baby_fooc	bakery_de	baking_in	baking_su	beauty
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0

Figure 6.1.2---Dataset after conversion

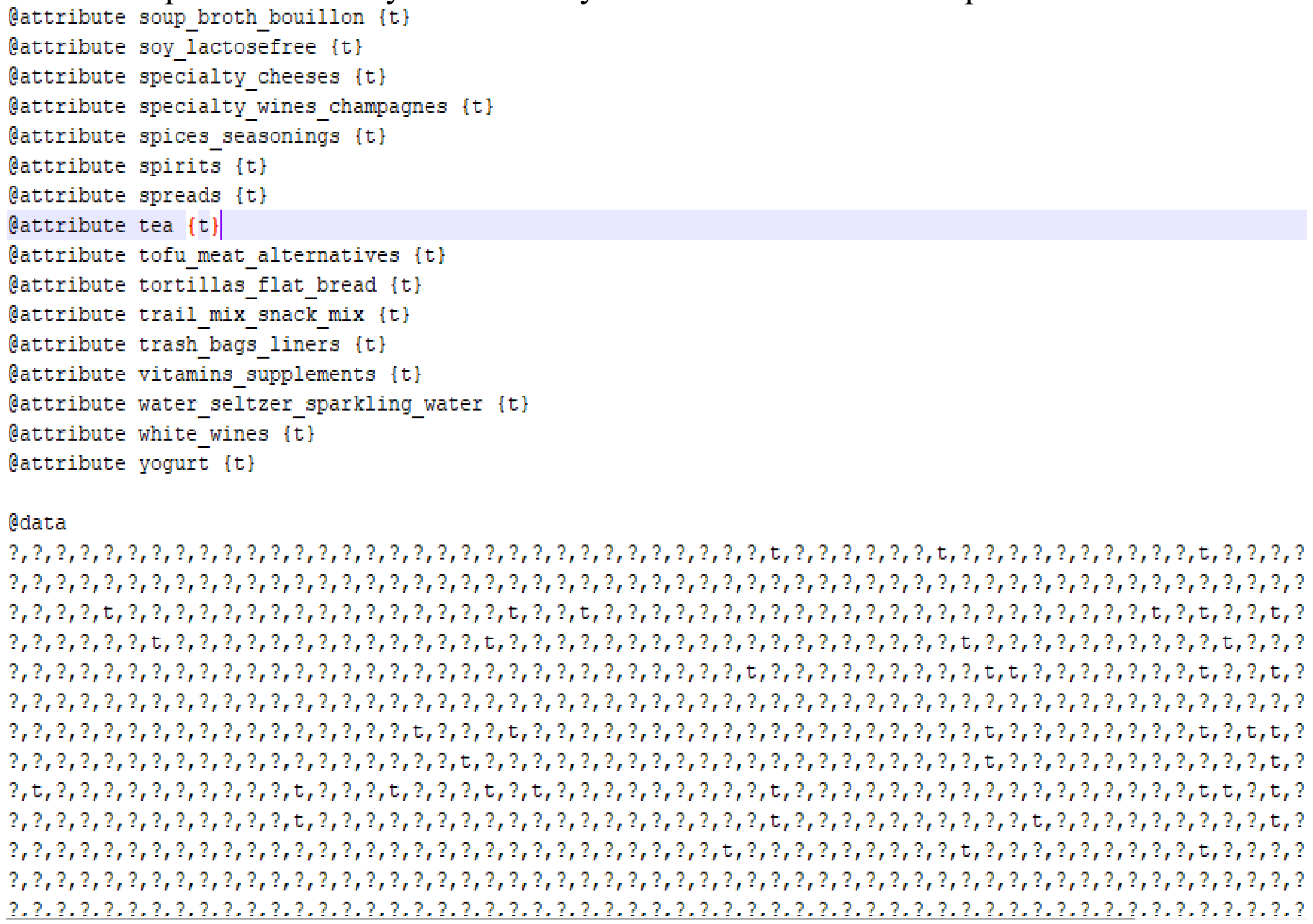
Then we converted the above dataset into arff format in weka and loaded that file into weka and converted all the attributes to nominal form and then generated rules. We observed that rules are generated for the attributes which are not having any quantity also(weka considered 0's in the transaction and generated rules for all such attributes having zeros). Thus we got a lot of rules.



```
baby_bath_body_care=0 99 ==> air_fresheners_candles=0 99    <conf
air_fresheners_candles=0 99 ==> baby_bath_body_care=0 99    <conf
baking_supplies_decor=0 99 ==> air_fresheners_candles=0 99    <co
air_fresheners_candles=0 99 ==> baking_supplies_decor=0 99    <co
beauty=0 99 ==> air_fresheners_candles=0 99    <conf:(1)> lift:(1
air_fresheners_candles=0 99 ==> beauty=0 99    <conf:(1)> lift:(1
beers_coolers=0 99 ==> air_fresheners_candles=0 99    <conf:(1)>
air_fresheners_candles=0 99 ==> beers_coolers=0 99    <conf:(1)>
bulk_dried_fruits_vegetables=0 99 ==> air_fresheners_candles=0 99
air_fresheners_candles=0 99 ==> bulk_dried_fruits_vegetables=0 99
```

Figure 6.1.3---Rules generated by considering zeros

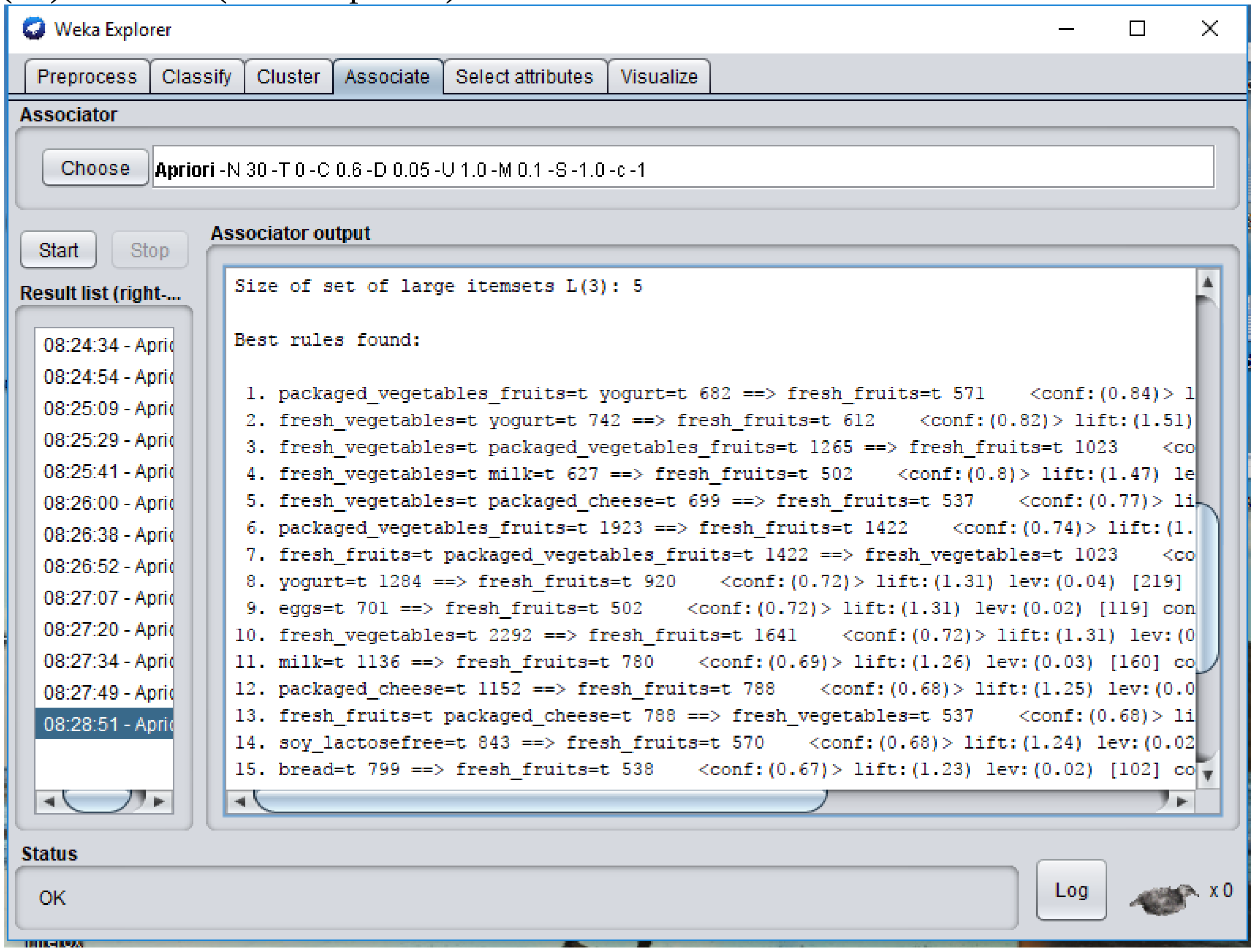
Then we replaced all 1's by't' and 0's by '?' in the arff file in notepad++.



```
@attribute soup_broth_bouillon {t}
@attribute soy_lactosefree {t}
@attribute specialty_cheeses {t}
@attribute specialty_wines_champagnes {t}
@attribute spices_seasonings {t}
@attribute spirits {t}
@attribute spreads {t}
@attribute tea {t}
@attribute tofu_meat_alternatives {t}
@attribute tortillas_flat_bread {t}
@attribute trail_mix_snack_mix {t}
@attribute trash_bags_liners {t}
@attribute vitamins_supplements {t}
@attribute water_seltzer_sparkling_water {t}
@attribute white_wines {t}
@attribute yogurt {t}

@data
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,t,?,?,t,?
?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,t,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,t,t,?,?,?,?,?,?,?,t,?,?,t,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,t,?,t,t,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,t,?
?,t,?,?,?,?,?,?,?,?,?,?,t,?,?,?,t,?,?,?,t,?,t,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,t,?,t,?
?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,t,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,t,?,?,?,?,?,?,?,?,?,t,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?
```

Figure 6.1.4---Dataset in arff format with 't' as values

Now we loaded this arff file into weka and then generated association rules using apriori algorithm with support=0.3 and confidence=0.6. Stored all generated rules (23) into a file(mainoutput.txt).


Associator: Apriori -N 30 -T 0 -C 0.6 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Associator output

```
Size of set of large itemsets L(3): 5

Best rules found:

 1. packaged_vegetables_fruits=t yogurt=t 682 ==> fresh_fruits=t 571    <conf:(0.84)> l
 2. fresh_vegetables=t yogurt=t 742 ==> fresh_fruits=t 612    <conf:(0.82)> lift:(1.51)
 3. fresh_vegetables=t packaged_vegetables_fruits=t 1265 ==> fresh_fruits=t 1023    <co
 4. fresh_vegetables=t milk=t 627 ==> fresh_fruits=t 502    <conf:(0.8)> lift:(1.47) le
 5. fresh_vegetables=t packaged_cheese=t 699 ==> fresh_fruits=t 537    <conf:(0.77)> li
 6. packaged_vegetables_fruits=t 1923 ==> fresh_fruits=t 1422    <conf:(0.74)> lift:(1.
 7. fresh_fruits=t packaged_vegetables_fruits=t 1422 ==> fresh_vegetables=t 1023    <co
 8. yogurt=t 1284 ==> fresh_fruits=t 920    <conf:(0.72)> lift:(1.31) lev:(0.04) [219]
 9. eggs=t 701 ==> fresh_fruits=t 502    <conf:(0.72)> lift:(1.31) lev:(0.02) [119] con
10. fresh_vegetables=t 2292 ==> fresh_fruits=t 1641    <conf:(0.72)> lift:(1.31) lev:(0
11. milk=t 1136 ==> fresh_fruits=t 780    <conf:(0.69)> lift:(1.26) lev:(0.03) [160] co
12. packaged_cheese=t 1152 ==> fresh_fruits=t 788    <conf:(0.68)> lift:(1.25) lev:(0.0
13. fresh_fruits=t packaged_cheese=t 788 ==> fresh_vegetables=t 537    <conf:(0.68)> li
14. soy_lactosefree=t 843 ==> fresh_fruits=t 570    <conf:(0.68)> lift:(1.24) lev:(0.02
15. bread=t 799 ==> fresh_fruits=t 538    <conf:(0.67)> lift:(1.23) lev:(0.02) [102] co
```

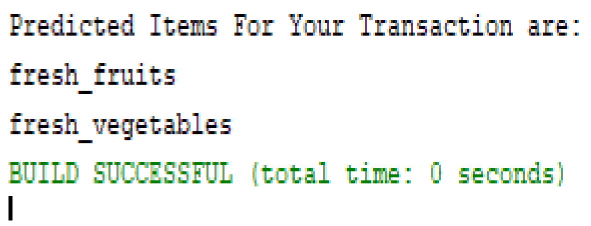
Result list (right-...

```
08:24:34 - Aprio
08:24:54 - Aprio
08:25:09 - Aprio
08:25:29 - Aprio
08:25:41 - Aprio
08:26:00 - Aprio
08:26:38 - Aprio
08:26:52 - Aprio
08:27:07 - Aprio
08:27:20 - Aprio
08:27:34 - Aprio
08:27:49 - Aprio
08:28:51 - Aprio
```

Status: OK

Figure 6.1.5---Generated rules in weka

Then we have written a java code in which we will take input of items for current transaction and then check the each input item with antecedent of generated rules that are stored in file and predict the consequent(if the consequent is not present in input list). Output of the code is



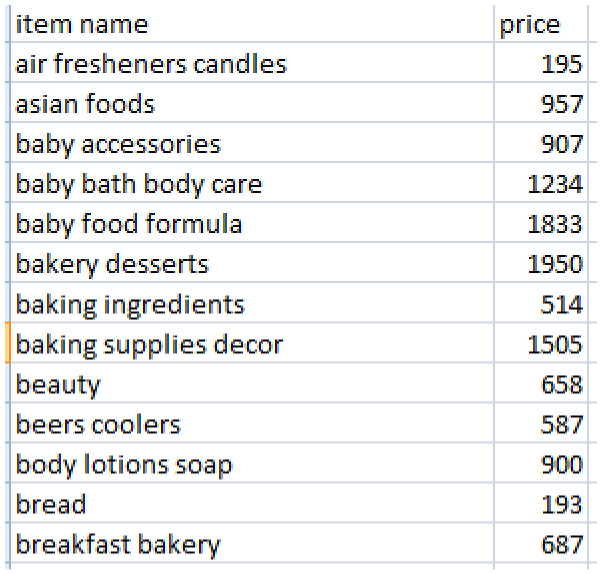
```
Predicted Items For Your Transaction are:
fresh_fruits
fresh_vegetables
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 6.1.6---Output of this module

Thus implementation of basic model has been completed.

6.2 Prediction based on transaction class

We have created a dataset with attributes item name, price and filled the price of all 134 items with appropriate value as shown in below figure 6.2.1.



item name	price
air fresheners candles	195
asian foods	957
baby accessories	907
baby bath body care	1234
baby food formula	1833
bakery desserts	1950
baking ingredients	514
baking supplies decor	1505
beauty	658
beers coolers	587
body lotions soap	900
bread	193
breakfast bakery	687

Figure 6.2.1---Items with price dataset

Then we have classified the above dataset into 3 classes class 1,2,3 i.e. low, medium, high respectively in weka by J48 decision tree classification algorithm. Below is the figure of the decision tree.

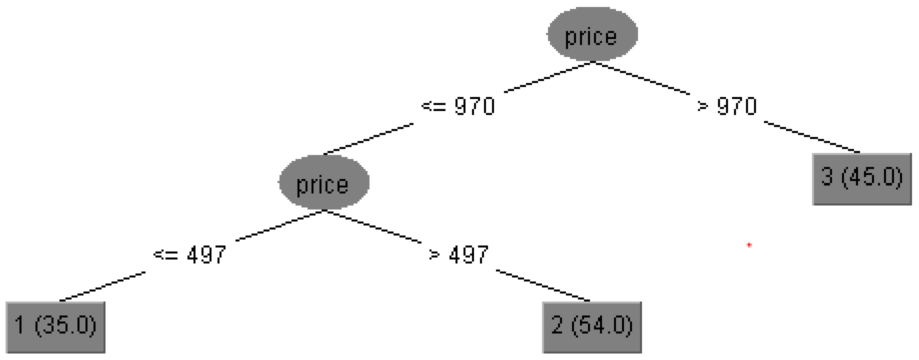
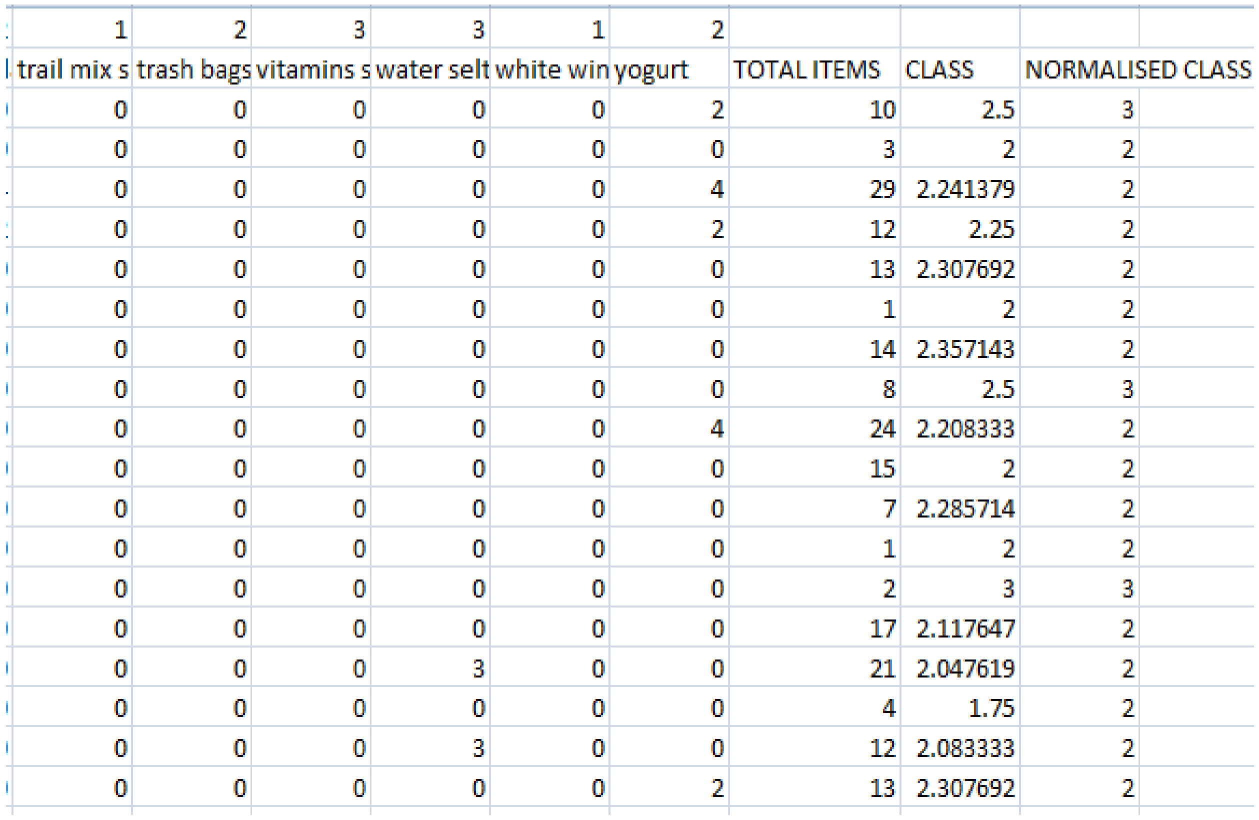


Figure 6.2.2---Decision tree classification

As we can see 134 items are divided into 3 classes i.e. items with price less than 497 into class 1(35 items) and price less than 970 and greater than 497 into class 2(54 items) and price greater than 970 into class 3(45 items). Now we got classes of each item. Based on classes of items we have to calculate class of the

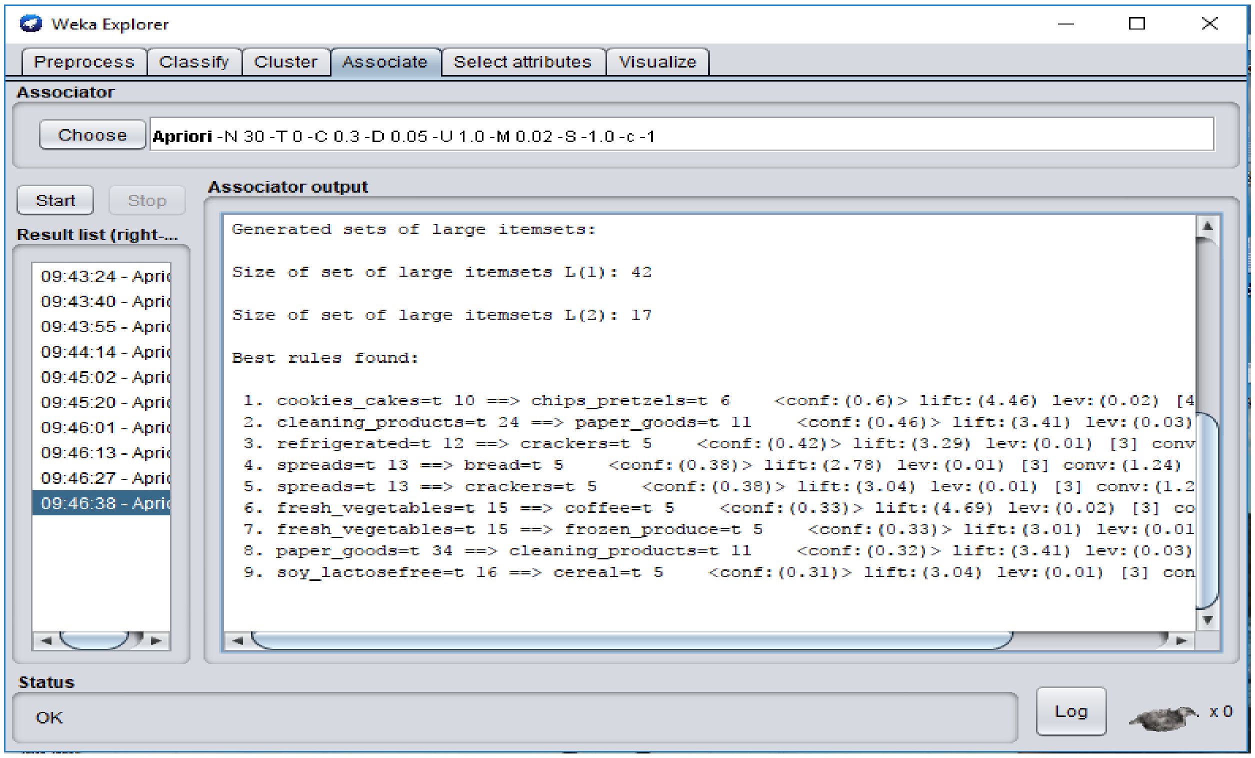
transaction. For that we have counted the sum of quantities of items bought in that transaction and stored as separate column named count. Then we multiplied the quantity of an item with its class and replaced quantity of that item with this value.
Now we calculated the total sum of quantity in the transaction i.e. sum(Quantity of item * class of that item) and divided it with corresponding count attribute and stored as separate attribute named class. Then we have normalised class attribute i.e. if class is less than 1.5 we normalised it to class 1 and if class is less than 2.5 then class 2 and greater than 2.5 to class 3 as shown in below figure.



1	2	3	3	1	2			
trail mix s	trash bags	vitamins s	water selt	white win	yogurt	TOTAL ITEMS	CLASS	NORMALISED CLASS
0	0	0	0	0	2	10	2.5	3
0	0	0	0	0	0	3	2	2
0	0	0	0	0	4	29	2.241379	2
0	0	0	0	0	2	12	2.25	2
0	0	0	0	0	0	13	2.307692	2
0	0	0	0	0	0	1	2	2
0	0	0	0	0	0	14	2.357143	2
0	0	0	0	0	0	8	2.5	3
0	0	0	0	0	4	24	2.208333	2
0	0	0	0	0	0	15	2	2
0	0	0	0	0	0	7	2.285714	2
0	0	0	0	0	0	1	2	2
0	0	0	0	0	0	2	3	3
0	0	0	0	0	0	17	2.117647	2
0	0	0	3	0	0	21	2.047619	2
0	0	0	0	0	0	4	1.75	2
0	0	0	3	0	0	12	2.083333	2
0	0	0	0	0	2	13	2.307692	2

Figure 6.2.3---Division of transactions into classes

We divided the transactions dataset into 3 sub datasets i.e. all the transactions of class 1 into a file and all the transactions of class 2 into another file and class 3 into another file. Then converted all these files into arff format in weka and then replaced its values by ‘t’ and ‘?’ following the procedure mentioned above and generated rules separately. We stored generated rules of each class into separate files and also 2 predicate rules in one file and 3 predicate rules into another. Below pictures shows the generated rules for each class.



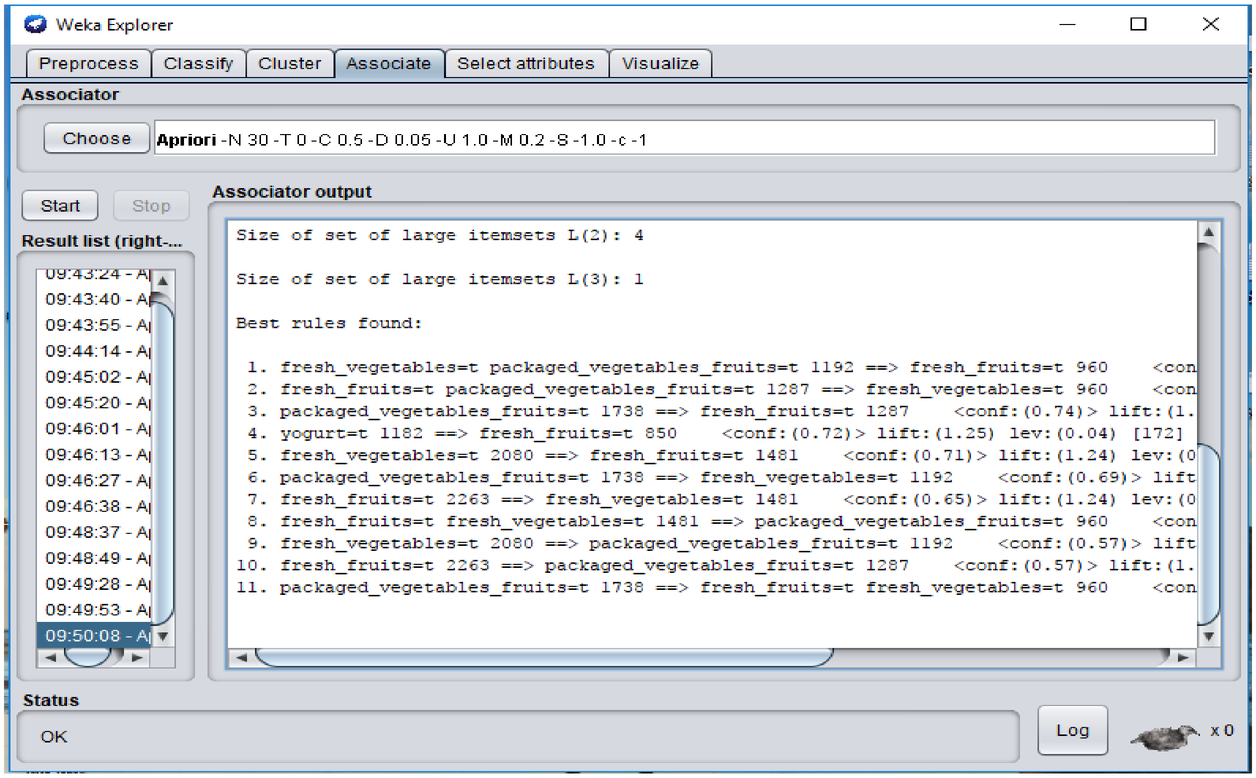
Apriori -N 30 -T 0 -C 0.3 -D 0.05 -U 1.0 -M 0.02 -S -1.0 -c -1

```
09:43:24 - Aprio
09:43:40 - Aprio
09:43:55 - Aprio
09:44:14 - Aprio
09:45:02 - Aprio
09:45:20 - Aprio
09:46:01 - Aprio
09:46:13 - Aprio
09:46:27 - Aprio
09:46:38 - Aprio
```

```
Generated sets of large itemsets:

Size of set of large itemsets L(1): 42

Size of set of large itemsets L(2): 17

Best rules found:

 1. cookies_cakes=t 10 ==> chips_pretzels=t 6    <conf:(0.6)> lift:(4.46) lev:(0.02) [4
 2. cleaning_products=t 24 ==> paper_goods=t 11    <conf:(0.46)> lift:(3.41) lev:(0.03)
 3. refrigerated=t 12 ==> crackers=t 5    <conf:(0.42)> lift:(3.29) lev:(0.01) [3] conv
 4. spreads=t 13 ==> bread=t 5    <conf:(0.38)> lift:(2.78) lev:(0.01) [3] conv:(1.24)
 5. spreads=t 13 ==> crackers=t 5    <conf:(0.38)> lift:(3.04) lev:(0.01) [3] conv:(1.2
 6. fresh_vegetables=t 15 ==> coffee=t 5    <conf:(0.33)> lift:(4.69) lev:(0.02) [3] co
 7. fresh_vegetables=t 15 ==> frozen_produce=t 5    <conf:(0.33)> lift:(3.01) lev:(0.01
 8. paper_goods=t 34 ==> cleaning_products=t 11    <conf:(0.32)> lift:(3.41) lev:(0.03)
 9. soy_lactosefree=t 16 ==> cereal=t 5    <conf:(0.31)> lift:(3.04) lev:(0.01) [3] con
```

OK

Figure 6.2.4---Generated rules for class 1 transactions



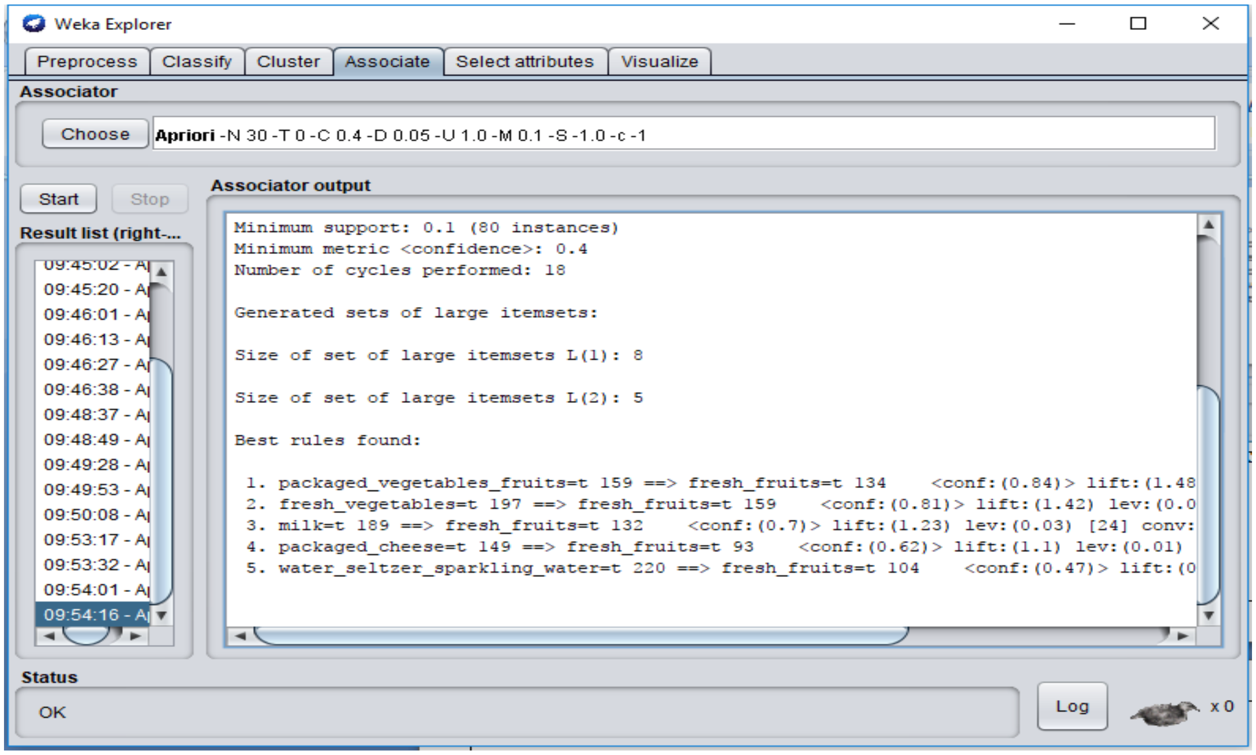
Apriori -N 30 -T 0 -C 0.5 -D 0.05 -U 1.0 -M 0.2 -S -1.0 -c -1

```
09:43:24 - A
09:43:40 - A
09:43:55 - A
09:44:14 - A
09:45:02 - A
09:45:20 - A
09:46:01 - A
09:46:13 - A
09:46:27 - A
09:46:38 - A
09:48:37 - A
09:48:49 - A
09:49:28 - A
09:49:53 - A
09:50:08 - A
```

```
Size of set of large itemsets L(2): 4

Size of set of large itemsets L(3): 1

Best rules found:

 1. fresh_vegetables=t packaged_vegetables_fruits=t 1192 ==> fresh_fruits=t 960    <con
 2. fresh_fruits=t packaged_vegetables_fruits=t 1287 ==> fresh_vegetables=t 960    <con
 3. packaged_vegetables_fruits=t 1738 ==> fresh_fruits=t 1287    <conf:(0.74)> lift:(1.
 4. yogurt=t 1182 ==> fresh_fruits=t 850    <conf:(0.72)> lift:(1.25) lev:(0.04) [172]
 5. fresh_vegetables=t 2080 ==> fresh_fruits=t 1481    <conf:(0.71)> lift:(1.24) lev:(0
 6. packaged_vegetables_fruits=t 1738 ==> fresh_vegetables=t 1192    <conf:(0.69)> lift
 7. fresh_fruits=t 2263 ==> fresh_vegetables=t 1481    <conf:(0.65)> lift:(1.24) lev:(0
 8. fresh_fruits=t fresh_vegetables=t 1481 ==> packaged_vegetables_fruits=t 960    <con
 9. fresh_vegetables=t 2080 ==> packaged_vegetables_fruits=t 1192    <conf:(0.57)> lift
10. fresh_fruits=t 2263 ==> packaged_vegetables_fruits=t 1287    <conf:(0.57)> lift:(1.
11. packaged_vegetables_fruits=t 1738 ==> fresh_fruits=t fresh_vegetables=t 960    <con
```

OK

Figure 6.2.5---Generated rules for class 2 transactions



Apriori -N 30 -T 0 -C 0.4 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
Minimum support: 0.1 (80 instances)
Minimum metric <confidence>: 0.4
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 8

Size of set of large itemsets L(2): 5

Best rules found:

 1. packaged_vegetables_fruits=t 159 ==> fresh_fruits=t 134    <conf:(0.84)> lift:(1.48
 2. fresh_vegetables=t 197 ==> fresh_fruits=t 159    <conf:(0.81)> lift:(1.42) lev:(0.0
 3. milk=t 189 ==> fresh_fruits=t 132    <conf:(0.7)> lift:(1.23) lev:(0.03) [24] conv:
 4. packaged_cheese=t 149 ==> fresh_fruits=t 93    <conf:(0.62)> lift:(1.1) lev:(0.01)
 5. water_seltzer_sparkling_water=t 220 ==> fresh_fruits=t 104    <conf:(0.47)> lift:(0
```

```
09:45:02 - A
09:45:20 - A
09:46:01 - A
09:46:13 - A
09:46:27 - A
09:46:38 - A
09:48:37 - A
09:48:49 - A
09:49:28 - A
09:49:53 - A
09:50:08 - A
09:53:17 - A
09:53:32 - A
09:54:01 - A
09:54:16 - A
```

OK

Figure 6.2.6---Generated rules for class 3 transactions

Now we have written a java code. First we have created a hashmap which stores item names along with its class.

```
        HashMap<String, Integer> map = new HashMap<>();
        File file=new File("F:\\items.txt");//Reading Items From File
        File file1=new File("F:\\itemss.txt");//Reading Item Class From File
        BufferedReader br=new BufferedReader(new FileReader(file));
        BufferedReader br1=new BufferedReader(new FileReader(file1));
        String st,st1;
        int ff;
        while((st=br.readLine())!=null && (st1=br1.readLine())!=null){
            int i=Integer.parseInt(st);
            map.put(st1,i);//Creating hashmap for items along with their
classes
             }
```

Code 6.2.1----Hashmap creation

Next we have taken input i.e. item names along with its quantity through an interface which will be explained later and stored in an array as shown in code 6.2.2.

```
Scanner scanner = new Scanner(new FileInputStream("F:\\input.txt"));
//Reading Current Transaction Items From File
String offer[]=new String[100];
int oin=0;
ArrayList<String> list=new ArrayList<String>();
int a[]=new int[100];
int ind=0;
while(scanner.hasNext())
{
list.add(scanner.next());//Adding Current Transaction Item Names
a[ind++]=Integer.parseInt(scanner.next());//Adding Quantity Of above item
}
```

Code 6.2.2---Storing current items along with its quantity

Now we calculated the class of current transaction using below formula.

$$class\ of\ current\ transaction = \frac{\sum(Quantity\ of\ item \times class\ of\ item)}{\sum Quantity\ of\ items}$$

After calculating the class we normalised it. If calculated class is less than 1.5 we have replaced it with 1 and if it is calculated class is less than 2.5, we replaced it with 2 and greater than 2.5 is replaced with 3 as shown in below code 6.2.3.

After calculating the class of current transaction we have cross checked the input items with the antecedent of the rules generated of particular class and predicted the items as shown in below figure 6.2.7.

```
 int sum=0;ind=0;int count=0;
for(String q:list){
    if(map.containsKey(q)){
        Integer z=map.get(q);//Retrieving Class of Current item}
      count=count+a[ind];
        sum=sum+(z*a[ind++]);
      } }
   sum=sum/count;
  int cls;
  if(sum<=1.5)
    cls=1;
else if(sum<=2.5)
    cls=2;
else cls=3;
```

Code 6.2.3---Calculation of current transaction class

```
Class of Current Customer is 2

Recommended Items For Your Present Transaction :
packaged_vegetables_fruits
```

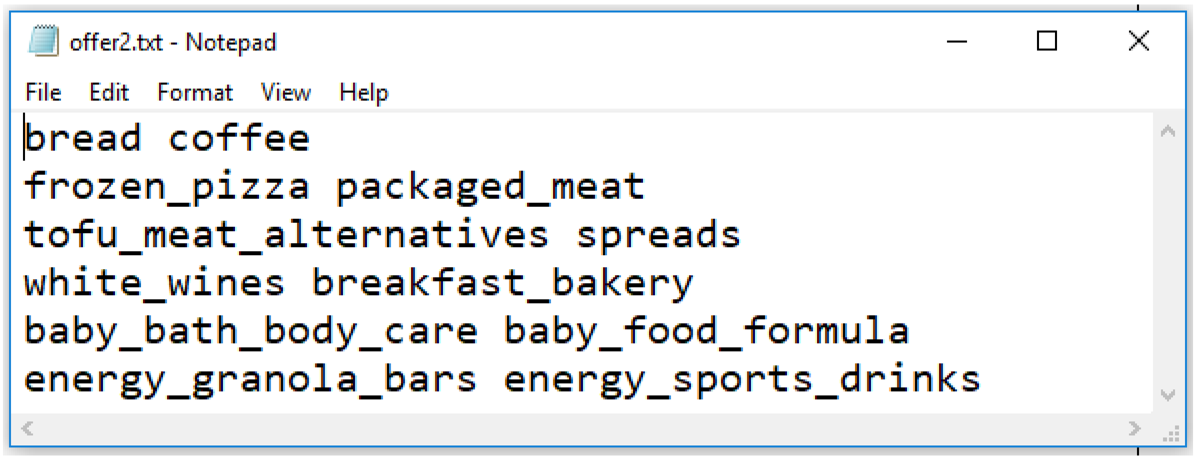
Figure 6.2.7---Output of the code

6.3 Prediction based on offers available

Offers are of two types. Namely Combo offers and Discount offers. In our model we have predicted missing items based on offers available also. But we have dealt with these offers differently as stated below.

6.3.1 Prediction based on Combo Offers

Combo offers are nothing but if we buy an item say item1 then we have some discount on item2 or item2 is free of cost. So if a customer buys item1 then we'll suggest item2 to customer. To implement this we have stored combo offers in a file (offer2.txt) with all item1's as antecedents and respective item2 as consequent as shown in below figure 6.3.1.1.



```
bread coffee
frozen_pizza packaged_meat
tofu_meat_alternatives spreads
white_wines breakfast_bakery
baby_bath_body_care baby_food_formula
energy_granola_bars energy_sports_drinks
```

Figure 6.3.1.1---combo offers text file

Then we checked all current items of the transaction with the antecedents of combo offers and if any input item is matched and corresponding consequent is not there in customer's current transaction then we'll suggest that particular consequent to the customer. Implementation of this whole combo offers process in shown below in code 6.3.1.1. Output of this prediction is given in figure 6.3.1.2.

```
for(String q:list){
            if(map.containsKey(q)){
                Scanner sc1 = new Scanner(new
FileInputStream("F:\\offer2.txt"));//Reading combo offers
                while(sc1.hasNext()){
            String d1=sc1.next();
            String d2=sc1.next();
            if(d1.equals(q)){
                offer[oin++]=q;
                offer[oin++]=d2;
                break;
            }}
            }}
```

Code 6.3.1.1---Predicting items based on combo offers

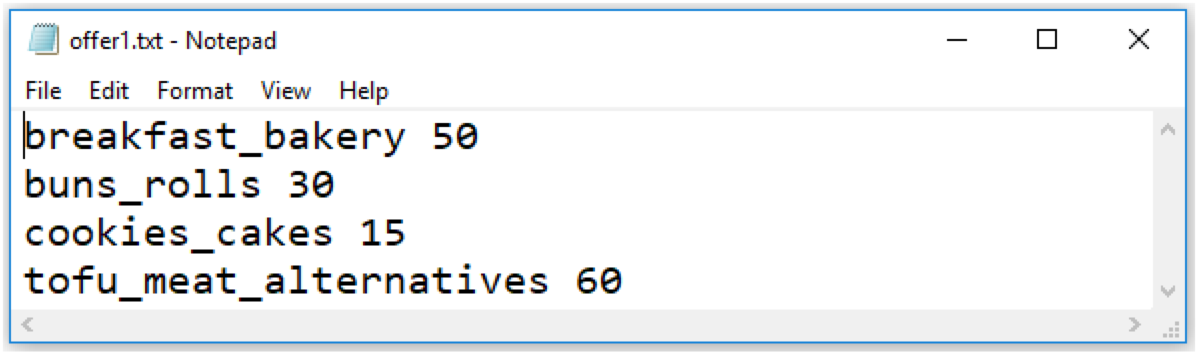
```
Offer Available:
Offer for baby_food_formula: bread
Offer for coffee: frozen_pizza
Offer for packaged_meat: tofu_meat_alternatives
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 6.3.1.2---Output of prediction of items of combo offers

6.3.2 Discount Offers

Discount offers means reducing the cost of an item by some percentage. By these offers our model can be disturbed since in calculating the class of an item we used its price as discussed in section 6.2. But when such offers are given then we have to reduce the item class correspondingly so that it does not affect the current transaction class. And also these offers will be valid for few days only. To overcome these challanges we have came up with an idea as follows.

We have stored discount offers in a file (offer1.txt) with item name as antecedent and its corresponding discount given as consequent as shown in figure 6.3.2.1.



```
breakfast_bakery 50
buns_rolls 30
cookies_cakes 15
tofu_meat_alternatives 60
```

Figure 6.3.2.1—Discount offers text file

Then we have checked all input items of current transaction with the antecedents of the discount file and if an item is matched and if the discount offered is greater than or equal to 50% and if class of that item is greater than 1 then we have decreased its class by 1 and then calculated the class of current transaction which is the optimal solution to the challange. For example as shown in the figure 6.3.2.1 tofu_meat_alternatives has discount of 60% and its class is 3, so we have reduced its class to 2 and then calculated the transaction class. Implementation of this solution is given below in code 6.3.2.1.

```
        int sum=0;ind=0;int count=0;
        for(String q:list){
            if(map.containsKey(q)){
                Integer z=map.get(q);//Retrieving Class of Current item
                Scanner sc = new Scanner(new
FileInputStream("F:\\offer1.txt"));//Reading Offers On an item
                while(sc.hasNext()){
            String b=sc.next();
            int i1=sc.nextInt();
            if(b.equals(q) && i1>=50 && z>1){
                z--;
                break;
            }}
                count=count+a[ind];
                sum=sum+(z*a[ind++]);
            }}
        sum=sum/count;
        int cls;
        if(sum<=1.5)
            cls=1;
        else if(sum<=2.5)
            cls=2;
        else cls=3;
```

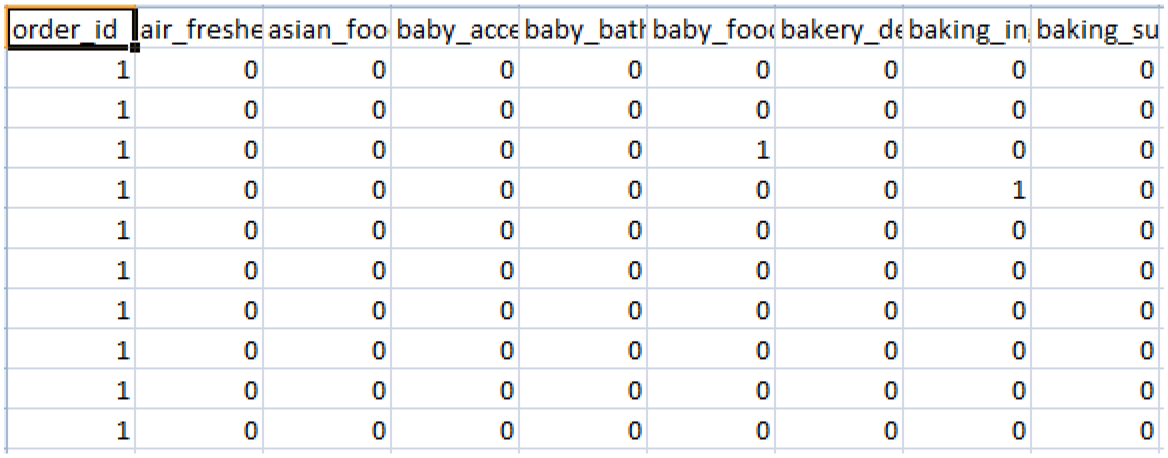
Code 6.3.2.1---Code for implementation of discount offers

6.4 User based Recommendation System

In this module we have predicted the missing items in the shopping cart of a customer based on customer previous history i.e. customer’s previous transaction. For this we have introduced a new attribute in our project named customer id for customer identification. Till now since the data to be stored is less we used files to store them but now to store customer’s previous transaction we used sqlyog database application as the data will be huge

6.4.1 Generation of Dataset

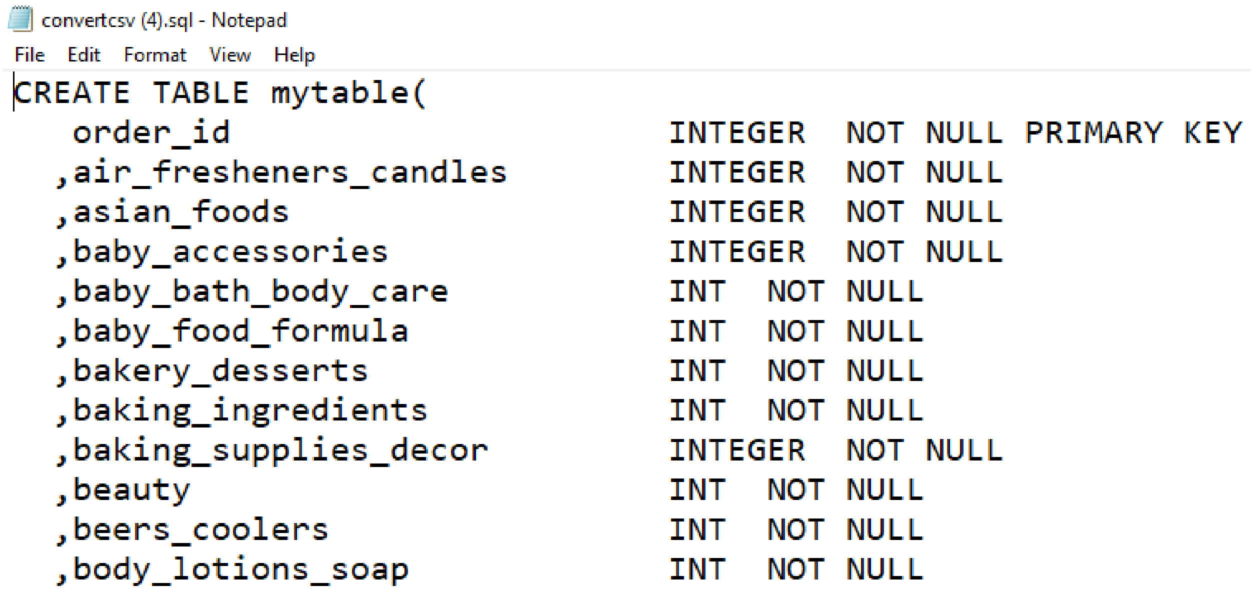
We copied last 600 transactions data of our main dataset which is discussed in section 2 into an excel sheet and created an attribute named order_id and generated numbers from 1 to 20 for all 600 transactions such that every order_id has 30 transactions using excel functions. Dataset is shown in below figure 6.4.1.1.



order_id	air_freshe	asian_foo	baby_acce	baby_bath	baby_food	bakery_de	baking_in	baking_su
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0

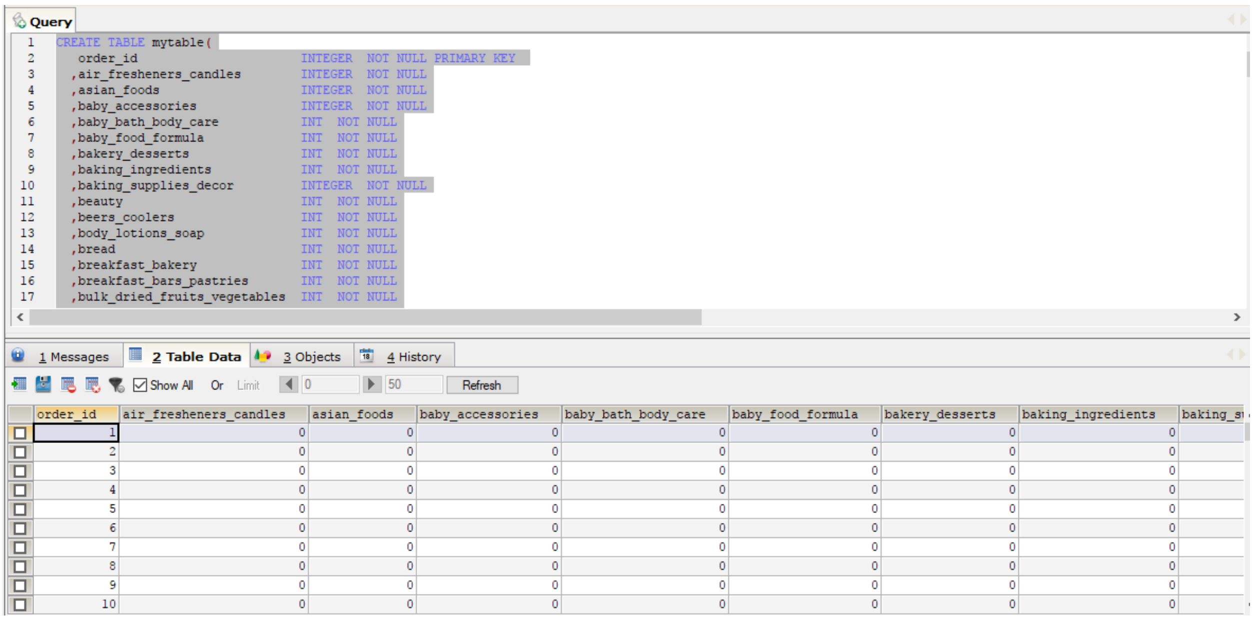
Figure 6.4.1.1---Generated dataset for user based recommendation system

Next we have converted this dataset into sql file through an online converter and executed that sql file in sqlyog. Sql file and its execution in sqlyog is shown below in figures 6.4.1.2 and 6.4.1.3 respectively.



```
CREATE TABLE mytable(
   order_id                       INTEGER  NOT NULL PRIMARY KEY
  ,air_fresheners_candles         INTEGER  NOT NULL
  ,asian_foods                    INTEGER  NOT NULL
  ,baby_accessories               INTEGER  NOT NULL
  ,baby_bath_body_care            INT  NOT NULL
  ,baby_food_formula              INT  NOT NULL
  ,bakery_desserts                INT  NOT NULL
  ,baking_ingredients             INT  NOT NULL
  ,baking_supplies_decor          INTEGER  NOT NULL
  ,beauty                         INT  NOT NULL
  ,beers_coolers                  INT  NOT NULL
  ,body_lotions_soap              INT  NOT NULL
```

Figure 6.4.1.2---sql file for the dataset generated



```sql
CREATE TABLE mytable(
   order_id                       INTEGER  NOT NULL PRIMARY KEY
  ,air_fresheners_candles         INTEGER  NOT NULL
  ,asian_foods                    INTEGER  NOT NULL
  ,baby_accessories               INTEGER  NOT NULL
  ,baby_bath_body_care            INT  NOT NULL
  ,baby_food_formula              INT  NOT NULL
  ,bakery_desserts                INT  NOT NULL
  ,baking_ingredients             INT  NOT NULL
  ,baking_supplies_decor          INTEGER  NOT NULL
  ,beauty                         INT  NOT NULL
  ,beers_coolers                  INT  NOT NULL
  ,body_lotions_soap              INT  NOT NULL
  ,bread                          INT  NOT NULL
  ,breakfast_bakery               INT  NOT NULL
  ,breakfast_bars_pastries        INT  NOT NULL
  ,bulk_dried_fruits_vegetables   INT  NOT NULL
```

order_id	air_fresheners_candles	asian_foods	baby_accessories	baby_bath_body_care	baby_food_formula	bakery_desserts	baking_ingredients	baking_s
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	

Figure 6.4.1.3---Execution of sql file in sqlyog

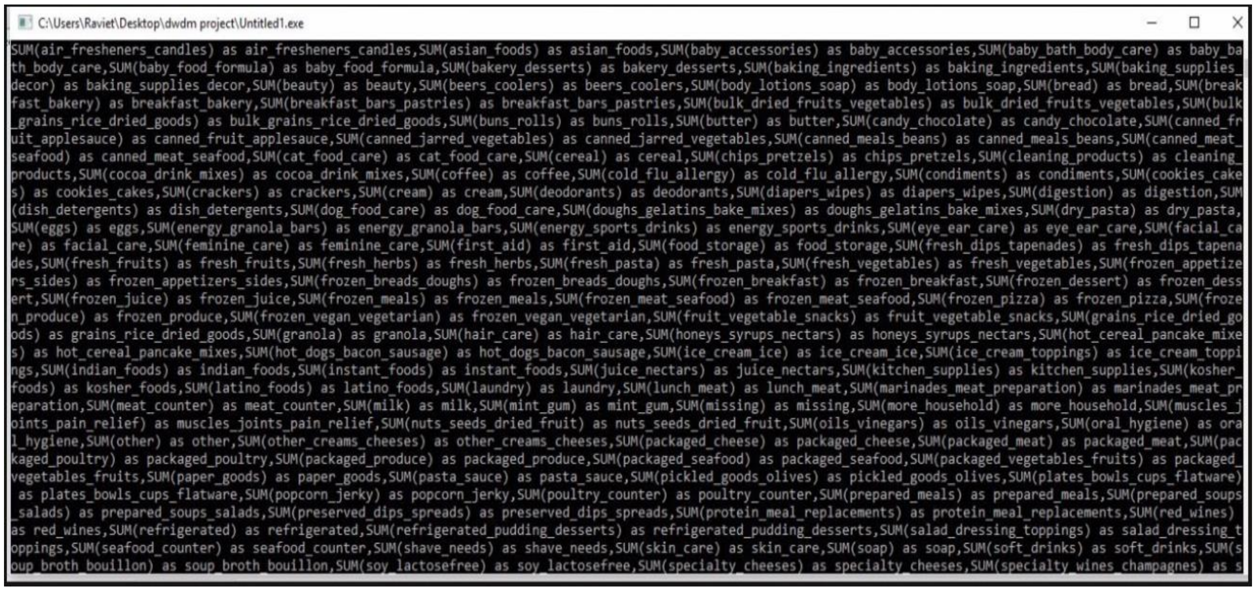
Next we have calculated the probability of a customer to buy a particular item and stored in a new table. Formula used to calculate probability is given below.

$$\mathit{Probability\ of\ a\ customer\ to\ buy\ an\ item} = \frac{\mathit{Number\ of\ transactions\ in\ which\ customer\ has\ bought\ that\ item}}{\mathit{Total\ number\ of\ transactions\ customer\ has\ done\ previously}}$$

For this we have to write an sql query but the challenge we faced was that we have 134 items and we should write all 134 item names in our query. To overcome this challenge we wrote a c++ program (as shown in code 6.4.1.1) which prints all the item names onto the console as shown in figure 6.4.1.4.

```cpp
#include<iostream>
#include<fstream>
using namespace std;
 int main()
 {ifstream f("C:\\Users\\Raviet\\Desktop\\itemss.txt");
 string s;
 while(!f.eof())
 {getline(f,s);
 cout<<"SUM("<<s<<") as "<<s<<",";
 }
 return 0;
 }
```

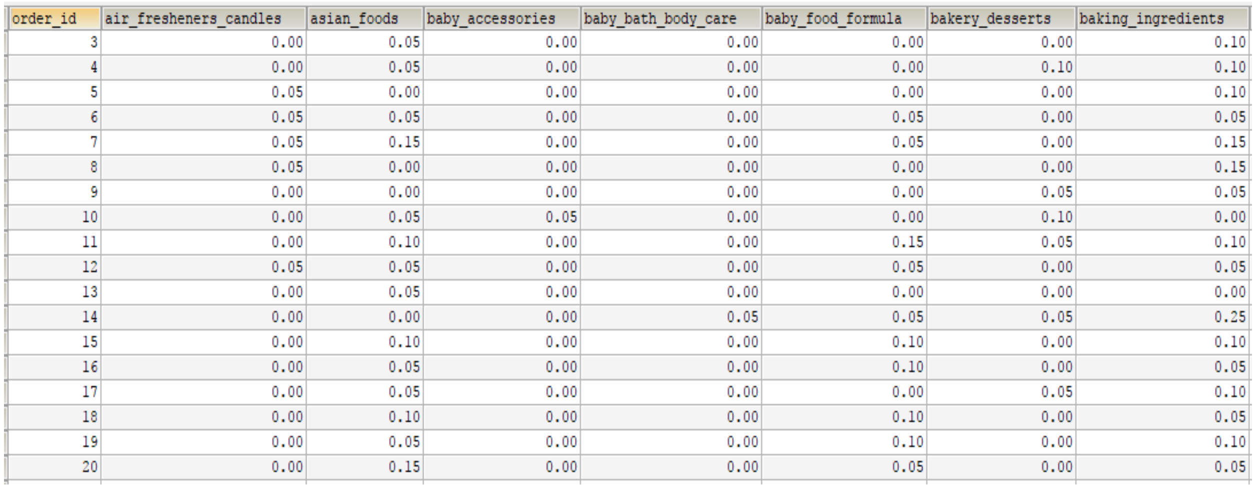
Code 6.4.1.1---For generation of item names



```
SUM(air_fresheners_candles) as air_fresheners_candles,SUM(asian_foods) as asian_foods,SUM(baby_accessories) as baby_accessories,SUM(baby_bath_body_care) as baby_ba
th_body_care,SUM(baby_food_formula) as baby_food_formula,SUM(bakery_desserts) as bakery_desserts,SUM(baking_ingredients) as baking_ingredients,SUM(baking_supplies_
decor) as baking_supplies_decor,SUM(beauty) as beauty,SUM(beers_coolers) as beers_coolers,SUM(body_lotions_soap) as body_lotions_soap,SUM(bread) as bread,SUM(break
fast_bakery) as breakfast_bakery,SUM(breakfast_bars_pastries) as breakfast_bars_pastries,SUM(bulk_dried_fruits_vegetables) as bulk_dried_fruits_vegetables,SUM(bulk
_grains_rice_dried_goods) as bulk_grains_rice_dried_goods,SUM(buns_rolls) as buns_rolls,SUM(butter) as butter,SUM(candy_chocolate) as candy_chocolate,SUM(canned_fr
uit_applesauce) as canned_fruit_applesauce,SUM(canned_jarred_vegetables) as canned_jarred_vegetables,SUM(canned_meals_beans) as canned_meals_beans,SUM(canned_meat_
seafood) as canned_meat_seafood,SUM(cat_food_care) as cat_food_care,SUM(cereal) as cereal,SUM(chips_pretzels) as chips_pretzels,SUM(cleaning_products) as cleaning_
products,SUM(cocoa_drink_mixes) as cocoa_drink_mixes,SUM(coffee) as coffee,SUM(cold_flu_allergy) as cold_flu_allergy,SUM(condiments) as condiments,SUM(cookies_cake
s) as cookies_cakes,SUM(crackers) as crackers,SUM(cream) as cream,SUM(deodorants) as deodorants,SUM(diapers_wipes) as diapers_wipes,SUM(digestion) as digestion,SUM
(dish_detergents) as dish_detergents,SUM(dog_food_care) as dog_food_care,SUM(doughs_gelatins_bake_mixes) as doughs_gelatins_bake_mixes,SUM(dry_pasta) as dry_pasta,
SUM(eggs) as eggs,SUM(energy_granola_bars) as energy_granola_bars,SUM(energy_sports_drinks) as energy_sports_drinks,SUM(eye_ear_care) as eye_ear_care,SUM(facial_ca
re) as facial_care,SUM(feminine_care) as feminine_care,SUM(first_aid) as first_aid,SUM(food_storage) as food_storage,SUM(fresh_dips_tapenades) as fresh_dips_tapena
des,SUM(fresh_fruits) as fresh_fruits,SUM(fresh_herbs) as fresh_herbs,SUM(fresh_pasta) as fresh_pasta,SUM(fresh_vegetables) as fresh_vegetables,SUM(frozen_appetize
rs_sides) as frozen_appetizers_sides,SUM(frozen_breads_doughs) as frozen_breads_doughs,SUM(frozen_breakfast) as frozen_breakfast,SUM(frozen_dessert) as frozen_dess
ert,SUM(frozen_juice) as frozen_juice,SUM(frozen_meals) as frozen_meals,SUM(frozen_meat_seafood) as frozen_meat_seafood,SUM(frozen_pizza) as frozen_pizza,SUM(froze
n_produce) as frozen_produce,SUM(frozen_vegan_vegetarian) as frozen_vegan_vegetarian,SUM(fruit_vegetable_snacks) as fruit_vegetable_snacks,SUM(grains_rice_dried_go
ods) as grains_rice_dried_goods,SUM(granola) as granola,SUM(hair_care) as hair_care,SUM(honeys_syrups_nectars) as honeys_syrups_nectars,SUM(hot_cereal_pancake_mixe
s) as hot_cereal_pancake_mixes,SUM(hot_dogs_bacon_sausage) as hot_dogs_bacon_sausage,SUM(ice_cream_ice) as ice_cream_ice,SUM(ice_cream_toppings) as ice_cream_toppi
ngs,SUM(indian_foods) as indian_foods,SUM(instant_foods) as instant_foods,SUM(juice_nectars) as juice_nectars,SUM(kitchen_supplies) as kitchen_supplies,SUM(kosher_
foods) as kosher_foods,SUM(latino_foods) as latino_foods,SUM(laundry) as laundry,SUM(lunch_meat) as lunch_meat,SUM(marinades_meat_preparation) as marinades_meat_pr
eparation,SUM(meat_counter) as meat_counter,SUM(milk) as milk,SUM(mint_gum) as mint_gum,SUM(missing) as missing,SUM(more_household) as more_household,SUM(muscles_j
oints_pain_relief) as muscles_joints_pain_relief,SUM(nuts_seeds_dried_fruit) as nuts_seeds_dried_fruit,SUM(oils_vinegars) as oils_vinegars,SUM(oral_hygiene) as ora
l_hygiene,SUM(other) as other,SUM(other_creams_cheeses) as other_creams_cheeses,SUM(packaged_cheese) as packaged_cheese,SUM(packaged_meat) as packaged_meat,SUM(pac
kaged_poultry) as packaged_poultry,SUM(packaged_produce) as packaged_produce,SUM(packaged_seafood) as packaged_seafood,SUM(packaged_vegetables_fruits) as packaged_
vegetables_fruits,SUM(paper_goods) as paper_goods,SUM(pasta_sauce) as pasta_sauce,SUM(pickled_goods_olives) as pickled_goods_olives,SUM(plates_bowls_cups_flatware)
 as plates_bowls_cups_flatware,SUM(popcorn_jerky) as popcorn_jerky,SUM(poultry_counter) as poultry_counter,SUM(prepared_meals) as prepared_meals,SUM(prepared_soups
_salads) as prepared_soups_salads,SUM(preserved_dips_spreads) as preserved_dips_spreads,SUM(protein_meal_replacements) as protein_meal_replacements,SUM(red_wines)
as red_wines,SUM(refrigerated) as refrigerated,SUM(refrigerated_pudding_desserts) as refrigerated_pudding_desserts,SUM(salad_dressing_toppings) as salad_dressing_t
oppings,SUM(seafood_counter) as seafood_counter,SUM(shave_needs) as shave_needs,SUM(skin_care) as skin_care,SUM(soap) as soap,SUM(soft_drinks) as soft_drinks,SUM(s
oup_broth_bouillon) as soup_broth_bouillon,SUM(soy_lactosefree) as soy_lactosefree,SUM(specialty_cheeses) as specialty_cheeses,SUM(specialty_wines_champagnes) as s
```

Figure 6.4.1.4---Output of the above code

Then we copied the above output and put in our sql query and executed it in sqlyog. Thus we created a table with all probabilities of items for all the customers. Dataset is shown in below figure 6.4.1.5.



order_id	air_fresheners_candles	asian_foods	baby_accessories	baby_bath_body_care	baby_food_formula	bakery_desserts	baking_ingredients
3	0.00	0.05	0.00	0.00	0.00	0.00	0.10
4	0.00	0.05	0.00	0.00	0.00	0.10	0.10
5	0.05	0.00	0.00	0.00	0.00	0.00	0.10
6	0.05	0.05	0.00	0.00	0.05	0.00	0.05
7	0.05	0.15	0.00	0.00	0.05	0.00	0.15
8	0.05	0.00	0.00	0.00	0.00	0.00	0.15
9	0.00	0.00	0.00	0.00	0.00	0.05	0.05
10	0.00	0.05	0.05	0.00	0.00	0.10	0.00
11	0.00	0.10	0.00	0.00	0.15	0.05	0.10
12	0.05	0.05	0.00	0.00	0.05	0.00	0.05
13	0.00	0.05	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.05	0.05	0.05	0.25
15	0.00	0.10	0.00	0.00	0.10	0.00	0.10
16	0.00	0.05	0.00	0.00	0.10	0.00	0.05
17	0.00	0.05	0.00	0.00	0.00	0.05	0.10
18	0.00	0.10	0.00	0.00	0.10	0.00	0.05
19	0.00	0.05	0.00	0.00	0.10	0.00	0.10
20	0.00	0.15	0.00	0.00	0.05	0.00	0.05

Figure 6.4.1.5---Probability table

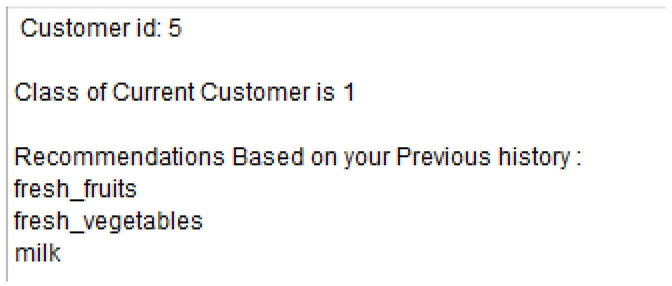
6.4.2 Implementation of User based recommendations

As our datasets are ready for user based recommendation system we implemented this system by writing a java code in which we connect to sqlyog database through a connector and a jar files named **mysql-connector-java-5.1.18-bin.jar** and **javax.servlet-3.1.jar.** The code for connection of database is shown below in code 6.4.2.1.

```
              Connection  con=null;
              PreparedStatement pstmt=null;
              ResultSet rs;
              try {
                  Class.forName("com.mysql.jdbc.Driver");
                  con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dwdm","root","pass")
;
              } catch (Exception e) {
                  System.out.println(e);
                  System.exit(0);
              }
```

Code 6.4.2.1---Connection to database

After connecting to the database we take input of the customer id through an interface which will be explained in the next section and retrieve all items probability of that particular customer and stored in a result set. Next we print all the item names that are having probability greater than 0.5 and are not in the current transaction. The output of this module is given below in figure 6.4.2.1.

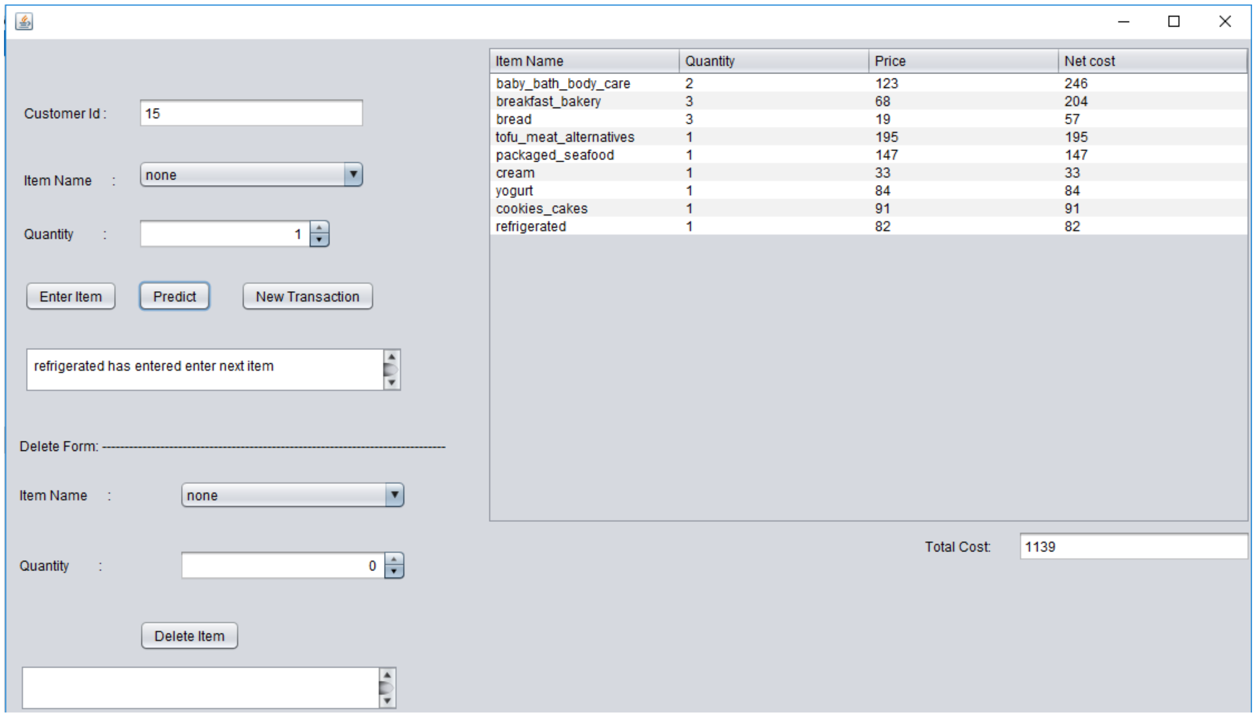


```
Customer id: 5

Class of Current Customer is 1

Recommendations Based on your Previous history :
fresh_fruits
fresh_vegetables
milk
```

Figure 6.4.2.1---Output of user based recommendation system.

6.5 Creation of User Interface

To implement the whole project we have created a user interface using Jframes which is implemented at the billing process. In this interface the billing person would enter the customer id, item name along with its quantity and also can delete the entered item or reduce the quantity of an item. It also calculates the total cost of the transaction by retrieving the item costs from a file as shown in the figure 6.2.1. After entering all the details of the current transaction there is a button named predict which when pressed would give all the predicted items based on association rules, offers available and also customer’s previous transactions. The UI is shown in the below figure 6.5.1.



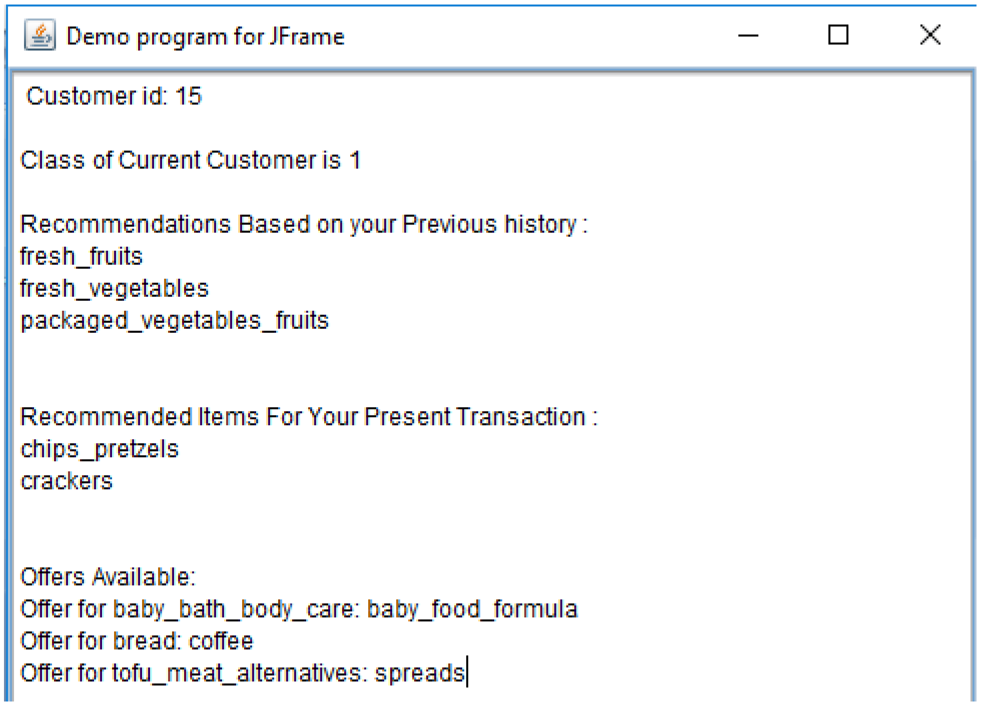
refrigerated has entered enter next item

Item Name	Quantity	Price	Net cost
baby_bath_body_care	2	123	246
breakfast_bakery	3	68	204
bread	3	19	57
tofu_meat_alternatives	1	195	195
packaged_seafood	1	147	147
cream	1	33	33
yogurt	1	84	84
cookies_cakes	1	91	91
refrigerated	1	82	82

Total Cost: 1139

Figure 6.5.1---User Interface

The output of the above transaction in figure 6.5.1 is given in figure 6.5.2.



```
Customer id: 15

Class of Current Customer is 1

Recommendations Based on your Previous history :
fresh_fruits
fresh_vegetables
packaged_vegetables_fruits


Recommended Items For Your Present Transaction :
chips_pretzels
crackers


Offers Available:
Offer for baby_bath_body_care: baby_food_formula
Offer for bread: coffee
Offer for tofu_meat_alternatives: spreads
```

Figure 6.5.2---Output of above transaction

When we press the predict button all the item names along with their quantity is retrieved and stored in a file. This file is taken as an input to the java code where transaction of the current class along with predictions based on previous history, association rules generated for that particular class and offers available is calculated and given as output in a new Jframe.

6.6 Novelty

6.6.1 Division of items into classes by using classification algorithm

Generally the available system for predicting the missing items in shopping cart will generate association rules irrespective of item prices. But in our project we have divided items into 3 classes namely low, medium, high based on its price as stated in section 6.2.

For this classification we have implemented J48 decision tree classification algorithm in weka and generated classes for the items. Decision tree is shown in the figure 6.2.2.

6.6.2 Prediction based on current transaction class

After dividing items into classes we have divided our training data that is data containing transactions into 3 classes with the formula mentioned in 6.2. Then we divided the dataset into 3 divisions based on classes and then generated association rules and stored those rules in different files separately.

Next we’ll calculate the class of current transaction using same formula and then predict missing items by cross checking input items with the association rules of the same class as shown in figure 6.2.1.

6.6.3 Prediction based on combo offers

Shopping malls often gives offers on some items during festive seasons. Every customer may not be aware of those combo offers and he/she may buy only one item of that offer leaving the other item in the offer. To overcome this problem our model checks all the items of current transaction with the items which are present in the combo offer and if that customer has failed to buy the other item of combo offer then we’ll suggest that other item to customer as shown in figure 6.3.1.2.

6.6.4 Prediction based on discount offers

Some malls may reduce the item cost by some percentage in discount sales. In our model we are considering item price to decide its class which will decide transaction class also. In discount sales most of the customers would buy that product (product which is having discount) and if that item is of class 3 then the weight of current transaction increases.

To deal with this challenge we cross checked each input item with the items which are under discount sale and if the discount is greater than 50% and class of that item is greater than 2, then we've reduced its class by 1 and then calculated transaction class which balances the weight of the transaction.

6.6.5 Prediction based on customer's previous transactions

Generally modules of prediction of missing items in shopping cart present in the market are item based predictions as customers previous transactions are not stored normally. But our module has implemented user based prediction where probabilities of items present in particular customer is calculated and if customer has failed to buy an item which is having probability greater than 0.5 then we'll suggest those items as shown in figure 6.4.2.1.

6.6.6 Creation of User interface

To implement this whole module we have created user interface by using java frame where total billing process of a transaction takes place which accepts items of current transaction along with its quantity and also calculates dynamic sum of the transaction along with functionality to delete or to decrease the quantity of entered item as shown in the figure 6.5.1.

There is a button named predict which when pressed predicts the missing items based on above mentioned methods and prints those items in another java frame as shown in 6.5.2.

7. CONCLUSION

Prediction of missing items in shopping cart means to recommend few items which are not bought by a customer but are bought by many other customers with few conditions. To implement this, our approach included classification of items based on its price using J48 decision tree classification algorithm and generating association rules for each class of transactions using apriori based association rule mining in weka.

8. Future Work

- Based on socio-economic factors such as tweets, facebook posts, Quora etc predict its price for that particular amount of time (month, week) and notify customer.(customer friendly)
- Considering an item predict its sales at every location and if a place has more sales than expected then we can suggest the producer to have a production unit over there .(producer friendly)
- Based on attributes of customer such as shirt brand , colours ,size, age etc store the data of customer attributes and the attributes of products he bought for at least one year and train the data with a module and then test the module with the attributes of current customer to predict the products customer is going to buy.
- Observe the movements of the customer through camera and predict the places where customer has failed to visit and give the data to the manager so that he can attract customers to those places in mall by arranging some attractions such as lights, toys, sound system etc.

9. References

- **Base paper 1 link :-**
  http://www.ijcaonline.org/volume21/number5/pxc3873385.pdf
- **Base paper 2 link :-**
  http://ijcsmc.com/docs/papers/November2013/V2I11201311.pdf
- **Data set link :-**
  https://www.kaggle.com/karthickveerakumar/orders-data
- Website used to learn excel modules
  https://www.extendoffice.com/documents/excel/2749-excel-find-and-replace-values-greater-than-less-than.html

10. Time Line

Date	Description
05/09/2017	Dataset collection from kaggle.
10/09/2017	Data Pre-Processing.
14/09/2017	Implementation of basic module.
02/10/2017	Items with price Dataset creation.
04/10/2017	Classification of items based on price.
09/10/2017	Classification of transactions dataset.
11/10/2017	Generation of association rules in weka.
12/10/2017	Implementing second module i.e. prediction based on transaction class.
21/10/2017	Implementing third module i.e. prediction based on offers available.
27/10/2017	Designing and implementation of User interface.
30/10/2017	Adding more functionalities to user interface such as deletion of item, calculating cost of transaction etc.
02/11/2017	Dataset creation for user based prediction.
05/11/2017	Implementation of final module i.e. user based prediction.
09/11/2017	Determining Future work for our project.

11. Code

11.1 User Interface code

```java
package swingdemoexample;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.Writer;
import java.util.HashMap;
import java.util.Scanner;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.table.DefaultTableModel;


public class interfac extends javax.swing.JFrame {

    public interfac() throws FileNotFoundException, IOException {
         HashMap<String, Integer> map = new HashMap<>();
         String st1,st;
         File file=new File("F:\\itmpri1.txt");//Reading Items From File
        File file1=new File("F:\\itemss.txt");//Reading Item Class From File
        BufferedReader br=new BufferedReader(new FileReader(file));
        BufferedReader br1=new BufferedReader(new FileReader(file1));
         while((st=br.readLine())!=null && (st1=br1.readLine())!=null){
            int i=Integer.parseInt(st);
            map.put(st1,i);//Creating hashmap for items along with their
classes
        }
         System.out.println(map);
              File realName = new File("F:\\itemz1.txt");
realName.delete();
        initComponents();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        int sum=Integer.parseInt(jTextField2.getText());
      int  i=0;
      System.out.println(itemname.getSelectedItem().toString());
      Object sp=quantity.getValue();
      if(itemname.getSelectedItem() =="none")
      { jTextField1.setText(itemname.getSelectedItem() =="none" ? "Select an
item" : itemname.getSelectedItem().toString() );}
      else{ PrintWriter writer = null;
        try {
            writer = new PrintWriter(new BufferedWriter(
                    new FileWriter("F:\\itemz.txt", true)));
        } catch (IOException ex) {
            Logger.getLogger(interfac.class.getName()).log(Level.SEVERE,
null, ex);
        }
      BufferedReader br = null;
FileReader reader = null;
try {
    reader = new FileReader("F:\\itemz.txt");
    br = new BufferedReader(reader);
    HashMap<String, Integer> map = new HashMap<>();
         //Scanner scanner = new Scanner(new
FileInputStream("F:\\ordwitpri.xlsx"));
         String st1,st;
         File file=new File("F:\\itmpri1.txt");//Reading Items From File
        File file1=new File("F:\\itemss.txt");//Reading Item Class From File
        BufferedReader br2=new BufferedReader(new FileReader(file));
        BufferedReader br1=new BufferedReader(new FileReader(file1));
         while((st=br2.readLine())!=null && (st1=br1.readLine())!=null){
           //System.out.print(st1+" ");
            //System.out.println(st);
            int q=Integer.parseInt(st);
            map.put(st1,q);//Creating hashmap for items along with their
classes


        }
         System.out.println(map);
            writer.print(itemname.getSelectedItem());
             writer.print(" ");
              writer.println(sp.toString());
              int e=(Integer)sp;
            Integer z=map.get(itemname.getSelectedItem());
            z=z/10;
            writer.close();
            i=1;
            int r=z*e;
            sum=sum+r;
            String str=Integer.toString(sum);
            jTextField2.setText(str);
            DefaultTableModel model=(DefaultTableModel)mdl.getModel();
                     model.addRow(new
Object[]{itemname.getSelectedItem(),quantity.getValue(),z,r});



} catch (FileNotFoundException e) {
    e.printStackTrace();
}         catch (IOException ex) {
```

```
              Logger.getLogger(interfac.class.getName()).log(Level.SEVERE, 
null, ex);
          }finally{
          try {
              reader.close();
          } catch (IOException ex) {
              Logger.getLogger(interfac.class.getName()).log(Level.SEVERE, 
null, ex);
          }


}
      jTextField1.setText(itemname.getSelectedItem()+" has entered enter next 
item");

      itemname.setSelectedItem("none");}

    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        File realName = new File("F:\\itemz.txt");
realName.delete();
jTextField1.setText("enter new transaction");
jTextField2.setText("0");
DefaultTableModel dm = (DefaultTableModel)mdl.getModel();
int rowCount = dm.getRowCount();
//Remove rows one by one from the end of the table
for (int i = rowCount - 1; i >= 0; i--) {
    dm.removeRow(i);
}
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
      try {
         PrintWriter   writer = new PrintWriter("F:\\itemz1.txt");
         writer.print("");
writer.close();
        } catch (FileNotFoundException ex) {
            Logger.getLogger(interfac.class.getName()).log(Level.SEVERE, 
null, ex);
        }

       String cid=(jTextField3.getText());
try {
            FileWriter outFile = new FileWriter("F:\\itemz1.txt", true);
    PrintWriter out1 = new PrintWriter(outFile);
    try {
         DefaultTableModel dm = (DefaultTableModel)mdl.getModel();
int rowCount = dm.getRowCount();
System.out.println(rowCount);
out1.println(cid);
for(int count=0;count<=rowCount-1;count++){
    out1.append(dm.getValueAt(count, 0).toString());
           out1.print(" ");
   out1.println(dm.getValueAt(count, 1).toString());

}
```

```
        } finally {
           out1.close();
           outFile.close();
        }
    }       catch (IOException ex) {
                Logger.getLogger(interfac.class.getName()).log(Level.SEVERE,
    null, ex);
            } finally {
       //outFile.close();
    }



            NewClass fet = null;
            try {
                fet = new NewClass();
            } catch (Exception ex) {
                Logger.getLogger(interfac.class.getName()).log(Level.SEVERE,
    null, ex);

           }    }

        private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {
            // TODO add your handling code here:
        }

        private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel table= (DefaultTableModel)mdl.getModel();
    int t1,t2;
            for(int j = 0; j < mdl.getRowCount(); j++){//For each column in that
    row

    if(mdl.getModel().getValueAt(j,0).equals(itemname1.getSelectedItem())){//Sear
    ch the model
                    {
                    if( (Integer)quantity1.getValue()<
    (Integer)mdl.getModel().getValueAt(j,1))
                    {t1= (Integer)mdl.getModel().getValueAt(j,1)-
    (Integer)quantity1.getValue();
                    t2=((Integer)mdl.getModel().getValueAt(j,1)-
    (Integer)quantity1.getValue())*(Integer)mdl.getModel().getValueAt(j,2);
                        table.setValueAt(t1,j,1);
                        table.setValueAt(t2,j,3);
                         int sum=Integer.parseInt(jTextField2.getText())-
    (((Integer)quantity1.getValue())*(Integer)mdl.getModel().getValueAt(j,2));
                        String str=Integer.toString(sum);
                jTextField2.setText(str);
                        jTextField4.setText("Item Quantity Succefully updated");
                         jTextField1.setText("");
                    }
                    else{
                        int sum=Integer.parseInt(jTextField2.getText())-
    ((Integer)mdl.getModel().getValueAt(j,3));
                        String str=Integer.toString(sum);
                        jTextField2.setText(str);
                        table.removeRow(j);
                        jTextField4.setText("Item Successfully Deleted");
                        jTextField1.setText("");
                    }
                        }

                }
//Print if found string
                else{
                    jTextField4.setText("Enter Valid Item");
                }
                }        // TODO add your handling code here:
        }

        private void jTextField3ActionPerformed(java.awt.event.ActionEvent evt) {
            // TODO add your handling code here:
        }



        /**
         * @param args the command line arguments
         */
        public static void main(String args[]) throws FileNotFoundException,
IOException {
            try {
                for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                    if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                        break;
                    }
                }
            } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(interfac.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
            } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(interfac.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
            } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(interfac.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
            } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(interfac.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
            }
            //</editor-fold>


            /* Create and display the form */
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
                    try {
                        new interfac().setVisible(true);
                    } catch (FileNotFoundException ex) {

    Logger.getLogger(interfac.class.getName()).log(Level.SEVERE, null, ex);
                    } catch (IOException ex) {

    Logger.getLogger(interfac.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            });
        }

        // Variables declaration - do not modify
        private javax.swing.JComboBox itemname;
        private javax.swing.JComboBox itemname1;
        private javax.swing.JButton jButton1;
        private javax.swing.JButton jButton2;
        private javax.swing.JButton jButton3;
        private javax.swing.JButton jButton4;
        private javax.swing.JLabel jLabel1;
        private javax.swing.JLabel jLabel2;
        private javax.swing.JLabel jLabel3;
        private javax.swing.JLabel jLabel4;
        private javax.swing.JLabel jLabel5;
        private javax.swing.JLabel jLabel6;
        private javax.swing.JLabel jLabel7;
        private javax.swing.JPanel jPanel1;
        private javax.swing.JPanel jPanel2;
        private javax.swing.JPanel jPanel3;
        private javax.swing.JScrollPane jScrollPane1;
        private javax.swing.JScrollPane jScrollPane2;
        private javax.swing.JScrollPane jScrollPane3;
        private javax.swing.JTextArea jTextField1;
        private javax.swing.JTextField jTextField2;
        private javax.swing.JTextField jTextField3;
        private javax.swing.JTextArea jTextField4;
        private javax.swing.JTable mdl;
        private javax.swing.JSpinner quantity;
        private javax.swing.JSpinner quantity1;
        // End of variables declaration
    }
```

11.2 Main Code

```
package swingdemoexample;
import com.mysql.jdbc.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;

public class NewClass extends javax.swing.JFrame {


    NewClass() throws FileNotFoundException, IOException{
         try {
             Connection  con=null;
             PreparedStatement pstmt=null;
             ResultSet rs;
             try {
                 Class.forName("com.mysql.jdbc.Driver");
                 con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dwdm","root","pass")
;
             } catch (Exception e) {
                 System.out.println(e);
                 System.exit(0);
             }
             String selsql="select * from probability where order_id=?";

             HashMap<String, Integer> map = new HashMap<>();
             final JFrame frame = new JFrame("Demo program for JFrame");
             JTextArea textFieldUserName = new JTextArea();
             JButton b1=new JButton("Back to Transaction");
             frame.add(textFieldUserName);
             //frame.add(b1);
             frame.setSize(400,400);
             File file=new File("F:\\items.txt");//Reading Items From File
             File file1=new File("F:\\itemss.txt");//Reading Item Class From
File
             BufferedReader br=new BufferedReader(new FileReader(file));
             BufferedReader br1=new BufferedReader(new FileReader(file1));
             String st,st1,st2,o="";
             int ff;
             while((st=br.readLine())!=null && (st1=br1.readLine())!=null){
                 int i=Integer.parseInt(st);
                 map.put(st1,i);//Creating hashmap for items along with their
classes


             }
             // System.out.println("Customer id:" + cid);
             print(map);
             System.out.println();
             Scanner scanner = new Scanner(new
FileInputStream("F:\\itemz1.txt"));//Reading Current Transaction Items From
File
             String offer[]=new String[100];
             int oin=0;
             String cid=scanner.next();
             ArrayList<String> list1=new ArrayList<String>();
             System.out.println("Customer id:"+cid);
              pstmt=con.prepareStatement(selsql);
             pstmt.setString(1,cid);
             rs=pstmt.executeQuery();

             textFieldUserName.append(" Customer id: "+cid+"\n\n");
             ArrayList<String> list=new ArrayList<String>();
             int a[]=new int[100];
             int ind=0;
             while(scanner.hasNext()){
                 list.add(scanner.next());//Adding Current Transaction Item
Names
                 a[ind++]=Integer.parseInt(scanner.next());//Adding Quantity
Of above item
             }
             BufferedReader br2=new BufferedReader(new FileReader(file1));
              while(rs.next()){
                  while((st2=br2.readLine())!=null){


                  int f1=0;
                 //st2=br2.readLine();
                float prob= Float.parseFloat(rs.getString(st2));

                if(prob>=0.5){
                    Iterator itrr=list.iterator();
  while(itrr.hasNext()){
      String f2=itrr.next().toString();
      if(f2.equals(st2)){
          f1=1;
          break;
      }

  // System.out.println(f2);
  }
if(f1==0){
    list1.add(st2);
}
                }
                  }
```

```
                    //System.out.print(st2+" ");
                }
                int sum=0;
                ind=0;
                int count=0;
                for(String q:list){
                    if(map.containsKey(q)){
                        Integer z=map.get(q);//Retrieving Class of Current item
                        Scanner sc = new Scanner(new
FileInputStream("F:\\offer1.txt"));//Reading Offers On an item
                        Scanner sc1 = new Scanner(new
FileInputStream("F:\\offer2.txt"));//Reading combo offers
                        while(sc.hasNext()){
                            String b=sc.next();
                            int i1=sc.nextInt();
                            if(b.equals(q) && i1>=50 && z>1){
                                z--;
                                break;
                            }

                        }
                        count=count+a[ind];
                        sum=sum+(z*a[ind++]);
                        while(sc1.hasNext()){
                            String d1=sc1.next();
                            String d2=sc1.next();
                            if(d1.equals(q)){
                                offer[oin++]=q;
                                offer[oin++]=d2;
                                break;
                            }
                        }
                    }
                }
                sum=sum/count;
                int cls;
                if(sum<=1.5)
                    cls=1;
                else if(sum<=2.5)
                    cls=2;
                else cls=3;
                System.out.println("Class of Current Customer is "+cls);
                textFieldUserName.append("Class of Current Customer is
"+cls+"\n\n");
                System.out.println();
                String result[]=new String[100];
                textFieldUserName.append("Recommendations Based on your Previous
history :\n");
                Iterator itr=list1.iterator();
  while(itr.hasNext()){
      //System.out.println("hi");
       textFieldUserName.append(itr.next().toString()+"\n");
  }
                int r=0;
                if(cls==1){
                    Scanner sc = new Scanner(new
FileInputStream("F:\\o1op.txt"));//Reading Class 1 Assosiation Rules
Generated from Weka.
                    String i1[]=new String[100];
                    String o1[]=new String[100];
                    int ind1=0;
                    while(sc.hasNext()){
                        i1[ind1]=sc.next();//Reading Antecedent Of generated
Rules
                        o1[ind1++]=sc.next();//Reading Consequent Of generated
Rules
                    }
                    for(String q:list){
                        for(int g=0;g<ind1;g++){
                            if(i1[g].equals(q)){
                                int flag=0;
                                for(int p=0;p<r;p++){
                                    if(result[p].equals(o1[g])){//Checking For
Non-Repetition of Predicted Items
                                        flag=1;
                                        break;
                                    }
                                }
                                if(flag==0){
                                    result[r++]=o1[g];
                                }
                            }
                        }
                    }
                }
                else if(cls==2){
                    Scanner sc1 = new Scanner(new
FileInputStream("F:\\o2op1.txt"));//Reading Class 2 Assosiation Rules with
one Antecedent Generated from Weka.
                    Scanner sc2 = new Scanner(new
FileInputStream("F:\\o2op2.txt"));//Reading Class 2 Assosiation Rules with
two Antecedent Generated from Weka.
                    String i1[]=new String[100];
                    String o1[]=new String[100];
                    String i2[][]=new String[100][2];
                    String o2[]=new String[100];
                    int ind1=0;
                    while(sc1.hasNext()){
                        i1[ind1]=sc1.next();//Reading Antecedent Of generated
Rules
                        o1[ind1++]=sc1.next();//Reading Consequent Of generated
Rules
                    }
                    int ind2=0;
                    while(sc2.hasNext()){
                        i2[ind2][0]=sc2.next();//Reading 1st Antecedent Of
generated Rules
                        i2[ind2][1]=sc2.next();//Reading 2nd Antecedent Of
generated Rules
                        o2[ind2++]=sc2.next();//Reading Consequent Of generated
Rules
                    }
```

```
                 for(String q:list){
                     for(int g=0;g<ind1;g++){
                         if(i1[g].equals(q)){
                             int flag=0;
                             for(int p=0;p<r;p++){
                                 if(result[p].equals(o1[g])){
                                     flag=1;
                                     break;
                                 }
                             }
                             if(flag==0){
                                 result[r++]=o1[g];
                             }
                         }
                     }
                 }
                 for(String q:list){
                     for(String q1:list){
                         for(int g=0;g<ind2;g++){
                             if((i2[g][0].equals(q) && i2[g][1].equals(q1))
|| (i2[g][1].equals(q) && i2[g][0].equals(q1))){
                                 int flag=0;
                                 for(int p=0;p<r;p++){
                                     if(result[p].equals(o2[g])){
                                         flag=1;
                                         break;
                                     }
                                 }
                                 if(flag==0){
                                     result[r++]=o1[g];
                                 }
                             }
                         }
                     }
                 }
             }
             else {
                 Scanner sc = new Scanner(new
FileInputStream("F:\\o3op.txt"));//Reading Class 3 Assosiation Rules
Generated from Weka.
                 String i1[]=new String[100];
                 String o1[]=new String[100];
                 int ind1=0;
                 while(sc.hasNext()){
                     i1[ind1]=sc.next();//Reading Antecedent Of generated
Rules
                     o1[ind1++]=sc.next();//Reading Consequent Of generated
Rules
                 }
                 for(String q:list){
                     for(int g=0;g<ind1;g++){
                         if(i1[g].equals(q)){
                             int flag=0;
                             for(int p=0;p<r;p++){
                                 if(result[p].equals(o1[g])){
                                     flag=1;
                                     break;
```

```
                                }
                            }
                            if(flag==0){
                                result[r++]=o1[g];
                            }
                        }
                    }
                }
            }
            System.out.println("Recommended Items For Your Present
Transaction :");
            //o=o+"Recommended Items For Your Present Transaction :/n";
            textFieldUserName.append("\n\nRecommended Items For Your Present
Transaction :");
            for(int h=0;h<r;h++){
                ff=0;
                for(String q:list){
                    if(q.equals(result[h])){
                        ff=1;
                        break;
                    }
                }
                if(ff==0){
                    textFieldUserName.append("\n"+result[h]);
                    System.out.println(result[h]);}
            }
            System.out.println();
            textFieldUserName.append("\n\n\nOffers Available:");
            System.out.println("Offer Available:");
            for(int h=0;h<oin;h++){
                ff=0;
                for(String q:list){
                    if(q.equals(result[h])){
                        ff=1;
                        break;
                    }
                }
                if(ff==0){
                    if(offer[h+1]!=null){
                        textFieldUserName.append("\nOffer for "+offer[h]+":
"+offer[h+1]);
                        System.out.println("Offer for "+offer[h++]+"->
"+offer[h]);}}
            }
            //textFieldUserName.setText(o);
            frame.setSize(500, 500);
            frame.setVisible(true);
            File realName2 = new File("F:\\itemz.txt");
            realName2.delete();
            frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        } catch (SQLException ex) {
Logger.getLogger(NewClass.class.getName()).log(Level.SEVERE, null,ex);
                            }


    }
    public static void print(Map<String, Integer> map)
    {
        if (map.isEmpty())
        {
            System.out.println("map is empty");
        }

        else
        {
            System.out.println(map);
        }
    }
}
```