ASSIGNMENT 2

Practice on Various Machine Learning Algorithms

Student Name: Udaya Bhaskar Reddy Malkannagari

Student ID: 13368171

1. [Data Pre-Processing and Exploration] **[8 MARKS]**

For each of the cases below, which names an attribute of a university student and then gives the record (example), state if the data are nominal, ordinal, interval or ratio scale:
a) Gender – Male
b) Class – Third year
c) Heart rate – 75 beats per minute
d) Blood pressure – 80/120 mm of Hg
e) Blood type – A positive
f) Blood sugar – 4
g) Known allergies – Nil
h) Body temperature – 36_0C

Answer:

a) Gender - Nominal (Categorical)
b) Class - Ordinal (Categorical)
c) Heart rate - Ratio (Numeric)
d) Blood pressure – Ratio (Numeric)
e) Blood type - Nominal (Categorical)
f) Blood sugar - Ratio (Numeric)
g) Known allergies – Missing Values
h) Body temperature – Interval(Numeric)

2. [Data Pre-Processing, Clustering] [20 MARKS]

Why is attribute scaling (e.g. normalization) of the data important? The following table contains sample records having the number of house sales and the total revenue generated by branches of an estate agency chain. Use the table as an example to discuss the necessity of normalization in any proximity measurement for clustering purposes.

BranchNo	Total Sales Quantity	Total Sales Value

B1	29	$5,500,000
B2	10	$5,000,000
B3	29	$5,000,000
B4	12	$890,000
B5	20	$2,500,000
B6	20	$3,200,000
B7	15	$678,000
B8	29	$5,200,000
B9	30	$5,300,000
B10	29	$5,500,000
B11	10	$5,000,000

Answer:

Data preprocessing is an important step in the knowledge discovery process. It is considered as a fundamental building block of data mining. There are various steps to data preprocessing which include extraction, discretization, transformation and loading of data. Normalization can be considered as a data transformation technique that serves multiple purposes in making the data more interpretable. It can be used as means to remove the effect of the outlier values and also make the values more comparable.

We can understand this preprocessing method better by gaining insight into the concept of measuring distance between two instances using the Euclidean distance formula.

Euclidean Distance:

Consider two points in a two-dimensional space (x1, y1) and (x2, y2). The distance between these two points can be measured by the formula

$$\sqrt{(x1-x2)^2+(y1-y2)^2}$$

This is called the Euclidean distance between the points.

We can extend the same concept to multidimensional space. For example, if the points are (a1, b1, c1, d1…) and (a2, b2, c2, d2…), the distance between the points is,

$$\sqrt{(a1-a2)^2+(b1-b2)^2+(c1-c2)^2+(d1-d2)^2+....}$$

Now, we look at our sample dataset.

BranchNo	Total Sales Quantity	Total Sales Value
B1	29	$5,500,000
B2	10	$5,000,000
B3	29	$5,000,000
B4	12	$890,000

B5	20	$2,500,000
B6	20	$3,200,000
B7	15	$678,000
B8	29	$5,200,000
B9	30	$5,300,000
B10	29	$5,500,000
B11	10	$5,000,000

In this dataset, the first column gives the branch number which is an identifier and the rest of the columns contain information. This information is related to the number of house sales and the total revenue generated by the separate branches. This is evaluated based on the two attributes, "Total Sales Quantity" and "Total Sales Value". This dataset can be used for clustering purposes as explained below.

Clustering:

Cluster analysis, is basically finding the similarities between data according to the characteristics found in the data and grouping similar data objects into clusters.

By analyzing the data, we can see that the total sales values for different branches can be clustered into a high range (around $5,000,000), a medium range (around $2,500,000) and low range ($678,000) clusters in correlation to the number of house sales.

We can see that the data relies on these two attributes to form its clusters.

Ignoring the branch ID column, if we consider each column as a dimension we can assume that each branch is represented by a point in two-dimensional space. We can use the Euclidean distance to calculate the distance between each of these points. **The closest points can be grouped together to form separate clusters based on the Euclidean distance between them.**

For example, Euclidean distance between B1 and B2 is

√ (5500000 – 5000000) ^2 + (29 – 10) ^2

= **500,000.000361**

However, we can observe that because of the extremely high numerical value of the first attribute "Total sales values", the second attribute value "Total Sales Quantity" seems irrelevant in calculating the Euclidean distance between these two points.

We can observe this by measuring the difference between the "Total sales value" of B1 and B2.

5500000 -5000000 = **500,000**

As we can see, both the results are almost the same. This means the clustering mechanism for this dataset is entirely depending on the attribute "Total Sales Values". The other attribute "Total Sales Quantity" is getting ignored because of the large numerical difference.

We now repeat this process and calculate the Euclidean distance between B1 and B4

√ (5500000 – 890000) ^ 2 + (29 – 12) ^ 2

= 4,610,000.0000313

Difference between the total sales values of B1 and B4 is

5500000 – 890000 = **4,610,000**

No matter what branches we choose the Euclidean distances between them are almost the same as the differences in "Total Sales value". This implies the Euclidean distances are dominated by this single attribute and the other attribute is getting ignored.

Solving the problem through Normalization:

As observed from the above results, we do not want one attribute to dominate the Euclidean distance calculation. We can achieve this by applying the Data preprocessing technique, Normalization to the given dataset.

What is Normalization?

The attribute is scaled to fit in a specific range. There are many types of normalization available, we will use one technique called the Min Max Normalization to scale the data.

Min Max Normalization transforms a value A to B which fits in the range [C, D] as given in the formula below.

$$B = \left(\frac{(A - \text{minimum value of } A)}{(\text{maximum value of } A - \text{minimum value of } A)}\right) * (D - C) + C$$

For example, we take the "Total Sales value" of B5, 2500000. We shall transform this into the range [0.0, 1.0].

The maximum value for this attribute is 5,500,000

The minimum value for this attribute is 678,000

The new scaled value for "Total Sales value" of B5 is,

{(2500000 – 678000) / (5500000 – 678000)} * (1 – 0) + 0

= 0.3778515

We can now apply this normalization technique to the "Total Sales Quantity" of B5, which is 20 and normalize it to the [0.0, 1.0] scale.

The maximum value for this attribute is 30

The minimum value for this attribute is 10

The new scaled value for "Total Sales Quantity' of B5 is,

{(20 – 10) / (30 – 10)} * (1 – 0) + 0

= 0.5

As we can see, using normalization we have scaled down the two attribute values to the same range **[0.0, 1.0].**

Normalized Total Sales Value for B5 = 0.3778515 = 0.38 (approx.)

Normalized Total Sales Quantity for B5 = 0.5

In the same way, we normalize the values for B3.

The new scaled value for "Total Sales value" of B3 is,

{(5000000 – 678000) / (5500000 – 678000)} * (1 – 0) + 0

= 0.896308

The new scaled value for "Total Sales Quantity' of B3 is,

{(29 – 10) / (30 – 10)} * (1 – 0) + 0

= 0.95

Normalized Total Sales Value for B5 = 0.896308 = 0.896 (approx.)

Normalized Total Sales Quantity for B5 = 0.95

Now, if we calculate the Euclidean distance for clustering purposes both the attribute values become relevant and we can get an accurate result based on the total dataset. The distances are no more dominated by the single attribute, "Total Sales Value".

Euclidean distance between B3 and B5 =

√ (0.896 – 0.38) ^ 2 + (0.95 – 0.5) ^ 2 = 0.684657

All values can now be normalized into the same scale and used for clustering purposes.

Other Normalization techniques like z-score normalization can be used for calculation using the statistical mean and standard deviation of the attribute values. Other measures like Pearson coefficient, Tanimoto Coefficient can be used as an alternative for Euclidean distance to calculate the similarity between two records. The goal of all these techniques is to create a scalable range for all the attributes in the dataset.

The k-means clustering algorithm is widely used to determine the clusters of individual groups in the dataset. This algorithm minimizes the squared Euclidean distance of the instances from their corresponding cluster centers. Normalization of attribute values helps greatly in calculating the accurate distance measures for such clustering purposes.

We can now easily scale the branches into three different clusters, High, Medium and Low, based on their total sales values and the total sales quantity by the use of normalization techniques.

3. [Classification – Decision Tree algorithm] [20 MARKS]

Use the soybean dataset (soybean.arff) to perform decision tree induction in Weka using three different decision tree induction algorithms; J48, REPTree, and RandomTree. Investigate different options, particularly looking at differences between pruned trees and unpruned trees. In discussing your results, consider the following questions.

a) What are the effects of pruning on the results for the soybean datasets?
b) Are there differences in the performances of the three decision tree algorithms?
c) What impacts do other parameters of the algorithms have on the results?

Answer:

Description of the dataset:

We are using the soybean dataset in our experiments to compare the three different classifiers. This dataset contains 36 attributes over 683 instances having 19 possible class labels.

Classifiers Used:

J48 Classifier:

J48 classifier is a straightforward C4.5 decision tree for classification, which creates a binary tree. It is the most useful decision tree approach for classification problems. This technique constructs a tree to model the classification process. After the tree is built, the algorithm is applied to each tuple in the database and results in classification for that tuple.

REPTree Classifier:

Reduces Error Pruning (REP) Tree Classifier is a fast decision tree learning algorithm and is based on the principle of computing the information gain with entropy and minimizing the error arising from variance.

RandomTree Classifier:

RandomTree classifier is used to build a decision tree based on a random set of columns. It constructs a tree that considers K randomly chosen attributes at each node. It performs no pruning.

Performance analysis:

We now run the algorithms on the dataset provided and compare the results to gain an understanding of their functionalities in predicting the models. We use 10-fold cross validation as a test setting to evaluate the performance of these classifiers.

We can run the J48 algorithm and the REPTree in both the pruned and the unpruned modes using Weka. The RandomTree algorithm performs no pruning. Hence the results for this classifier are based on the unpruned tree structure.

The confidence factor for the J48 is set to the default. A change in this value affects the pruning of the decision tree generated. The variations in the confidence will be analyzed in detail in the later sections. For now, we leave this setting to its default value of 0.25. The minimum number of

instances per leaf is also set to the default 2. Weka runs this algorithm on the pruned mode by default. That setting remains unchanged.

For the REPTree algorithm, we use the pruned mode is set as the default option.

Pruned Results Comparison Table:

Classifier	Accuracy	Correctly classified instances	Time taken to build the model	Size of the Tree	Relative Absolute Error	Mean Absolute Error	Root Mean Squared Error
J48	91.5081%	625	0.02 secs	93	14.0484%	0.0135	0.0842
REPTree	84.7731%	579	0.02 secs	89	26.5051%	0.0255	0.1127
RandomTree	84.041%	574	0.01 secs	668	19.8793%	0.0191	0.1255

Unpruned Results Comparison Table:

Classifier	Accuracy	Correctly classified instances	Time taken to build the model	Size of the Tree	Relative Absolute Error	Mean Absolute Error	Root Mean Squared Error
J48	91.3616%	624	0.01 secs	175	12.5318%	0.012	0.0864
REPTree	89.6047%	612	0.04 secs	137	18.3987%	0.0177	0.0976
RandomTree	84.041%	574	0.01 secs	668	19.8793%	0.0191	0.1255

** RandomTree classifier performs no pruning. The values are same in both the tables for comparison purposes.*

Table analysis:

<u>Effects of pruning on the results for the soybean dataset:</u>

As we can see from the pruned results table, the J48 classifier outperforms the other two classifiers with the chosen dataset. It shows a high accuracy of 91.5081% with a total of 625 correctly classified instances. The time taken to build the model is at 0.02 seconds which indicates a high performance in predictive analysis. The relative absolute error and the mean absolute error are also the lowest for the J48 classifier. The size of the tree generated which indicates the number of nodes is also comparatively smaller in relation to the other classifiers.

The unpruned results also show a similar pattern with the J48 performing the best. We can clearly observe the unpruned results for J48 are almost the same as the pruned results with a high accuracy of 91.3616%. The only difference is the size of the tree (175) which is almost double that of the pruned tree. This indicates that pruning had little effect on the accuracy of the model generated. Even after parts of the decision tree are cut through pruning, it had little

impact on the accuracy of the model. This implies that the data is consistently spread out to apply pruning for testing purposes.

The REPTree classifier performs well with a tree size of 89 in the pruned mode and 137 in the unpruned mode. This is significantly smaller in comparison to the tree size of the RandomTree classifier, which is 668.

We can also observe that the unpruned mode gives a slightly better accuracy for the REPTree classifier. This means the classifier works better with a larger tree even though the time taken to build the model is slightly higher. REPTree uses the regression tree logic and creates multiple trees in different iterations. After that it selects best one from all generated trees. That will be considered as the representative.

Differences in the performance of the three classifiers:

Comparing the size of the tree:

The size of the tree indicates the number of nodes in the tree and can be a visual tool in analyzing the data. The larger the size of the tree, the more difficult the model becomes to comprehend. Computing probabilities of different possible branches, determining the best split of each node and selecting optimal combining weights to prune algorithms contained in the decision tree are complicated tasks that require expertise and time. As mentioned above, the Random Tree has the largest tree size among the three different models generated (668). The J48 and REPTree has trees which are much smaller than the Random Tree. Random Tree is a supervised Classifier; it is an ensemble learning algorithm that generates many individual learners. It employs a bagging idea to produce a random set of data for constructing a decision tree. However, the visual interpretation of such trees is complex and confusing.

Furthermore, there are several disadvantages of having a larger decision tree structure. There may be a possibility of duplication with the same sub-tree on a different path, inadequacy of regression and predicting continuous values, and presentation problems that may arise with large tree structures.

By our analysis, we can conclude that the J48 and the REPTree models are better suited with the current dataset if we consider the tree size as a decisive parameter as they have relatively smaller size in comparison to the Random Tree.

Comparing the Root Mean Squared Error (RSME):

In pruning the tree, the measure used is the mean square error on the predictions made by the tree, which is 0.1127. For the unpruned mode, the root mean square error is calculated as 0.0976 which is slightly lower. This represents the sample standard deviation of the differences between predicted values and observed values. It is a good measure of accuracy, but only to compare forecasting errors of different models for a particular variable and not between variables, as it is scale-dependent. It serves to address the magnitude of errors in predictions into a single measure of predictive power. The J48 has the lowest value for this measure at 0.0842 in the pruned mode. The RandomTree value for this measure stands at 0.1255 which is the highest among the three classifiers.

Root Mean Squared Error (**RMSE**) can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of **RSME** indicate a better fit. A well-fitting regression model results in

predicted values close to the observed data values. **RMSE** is a good measure of how accurately the model predicts the response, and is the most important criterion for fit if the main purpose of the model is prediction. From the above results, we can safely conclude that the J48 has the best fit in predicting the model.

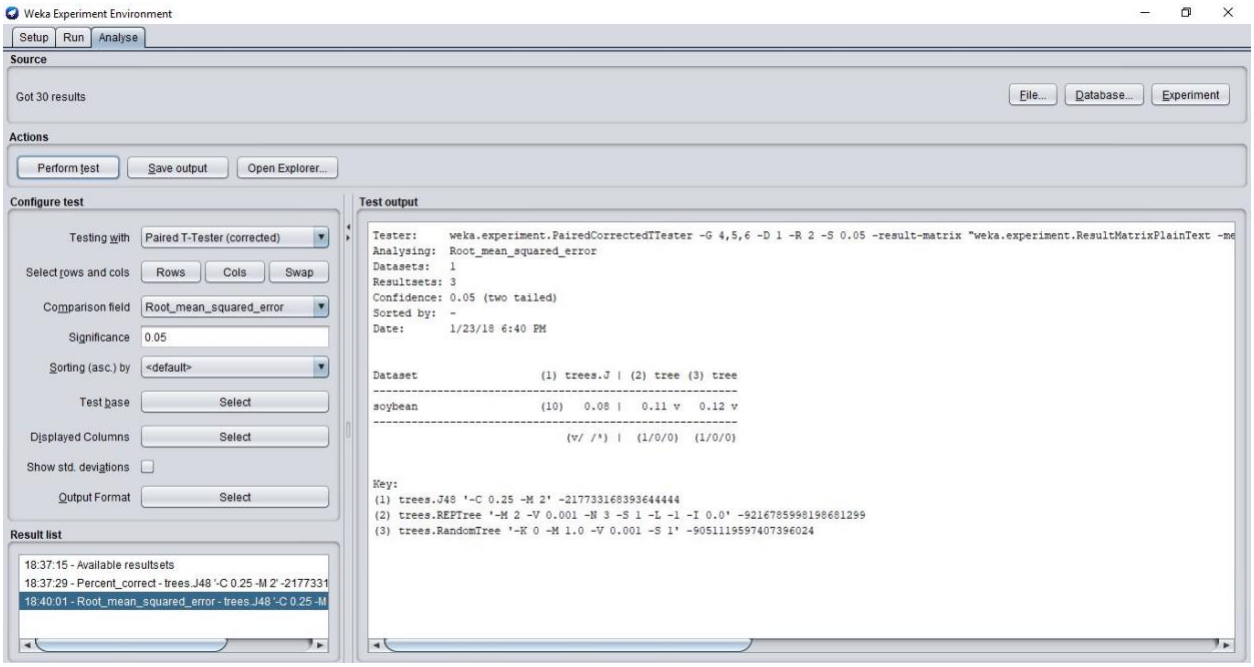


Figure 1: Showing the comparison of RSME for the three classifiers

Comparing the accuracies of the correctly classified instances:

Classification accuracy is perhaps the biggest measure in checking the performance of a model. The higher the accuracy of the correctly classified instances, the more predictive the model is. As we have discussed earlier, the J48 shows the best accuracy in predicting the models in comparison to the other two classifiers. It shows a total of 625 classified instances with an accuracy of 91.5081% which is very high. The REPTree Classification accuracy stands at 84.7731% with 579 correctly classified instances in the pruned mode and at 89.6047% with 612 correctly classified instances in the unpruned mode. For the Random Tree, the classification accuracy is at 84.041% with 574 correctly classified instances.

We can conclude the J48 provides a better measure of accuracy in comparison to the other classifiers.

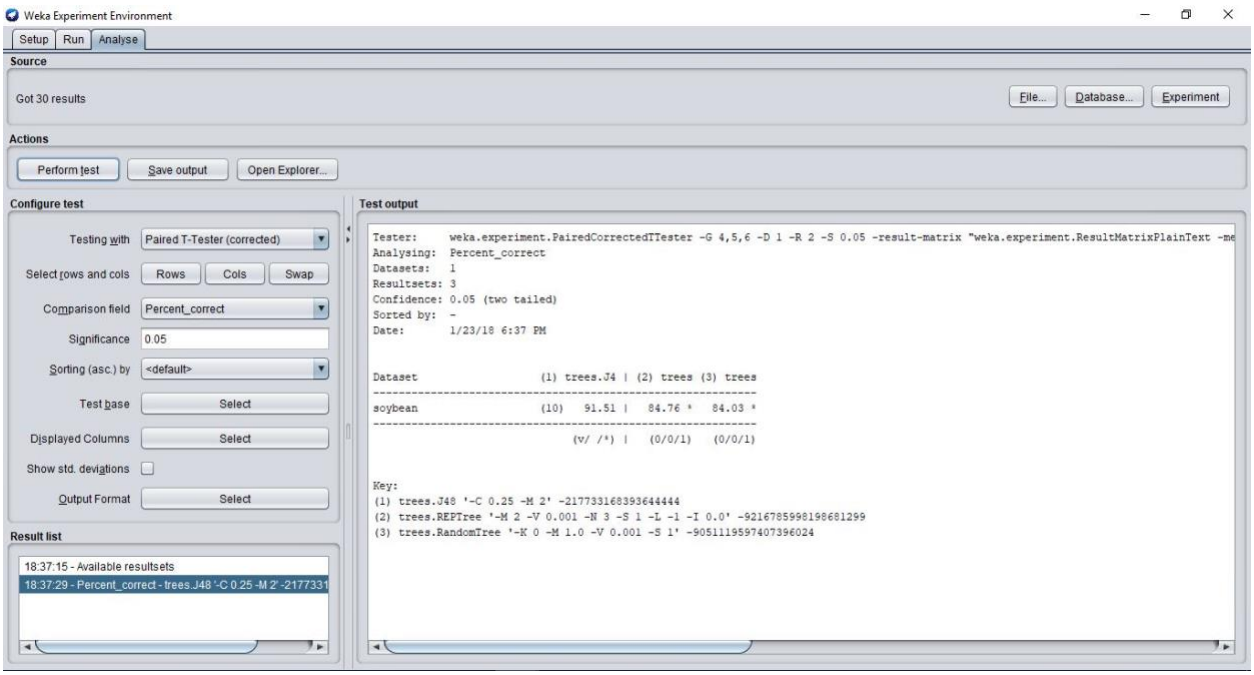


Figure 2: Showing the comparison of accuracies of the three classifiers

Comparing the time taken to build the model:

The time taken to build the model is also a good measure in analyzing the performance of a classifier. We can equate it to the computation costs required to generate a model based on its speed. The faster the time taken the lesser the costs involved.

When we compare this measure, we can see that all the classifiers perform equally well with this parameter with the REPTree in the unpruned mode being the highest at 0.04 seconds. This is not a significant difference in comparison to the other classifiers speeds. The J48 (unpruned) and the Random Tree stand at 0.01 seconds in building the models.

We can conclude that all the classifiers perform well on this measure for faster generation of their models.

Summary of the differences of the classifiers:

From the above comparisons, we can summarize that the J48 algorithm is the best suited classifier for this dataset with a smaller tree structure and a better accuracy and fit. It also generates the model faster with a better time to build it.

C)-Impact of other parameters on the algorithms

We now try to analyze the impact of different parameters on the classifiers.

Confidence Factor

The parameter altered to test the effectiveness of post-pruning was labeled by Weka as the confidence factor. In the Weka J48 classifier, lowering the confidence factor decreases the amount of post-pruning. To understand how this works, some discussion of the confidence factor is necessary.

Post-pruning in the C4.5 algorithm is the process of evaluating the decision error (estimated percent misclassifications) at each decision junction and propagating this error up the tree. At each junction, the algorithm compares

(1) the weighted error of each child node versus

(2) the misclassification error if the child nodes were deleted and the decision node were assigned the class label of the majority class.

The training data misclassifications at each node would not provide a sufficient error estimate - the tree was created from this data so it would not lead to any pruning. Instead, the misclassification error must be understood as an approximation of the actual error based on incomplete data. This is where the statistical notion of confidence comes into play. We take a pessimistic guess of the actual classification error based on the training data. If a given node is labeled "positive" and contains 1 positive and 1 negative (incorrectly classified) instance, its error is 50%. The actual error, however, for classifying future instances at this node could be anywhere between 28% and 72% (with a confidence factor of 0.25). Since we are pessimistic about our classifier, we assign the high error of 72%. If we have less confidence in our training data (which corresponds to a lower "confidence factor"), the error estimate for each node goes up, increasing the likelihood that it will be pruned away in favor of a more stable node upstream. Nodes reached by very few instances from the training data are penalized, since we cannot make confident assumptions about their actual classification error.

We test the J48 with confidence factor ranging from 0.1 to 1.0 by an increment of 0.1. The number of instances per node (minNumObj) was held at 2 and cross validation folds was held at 10.

```
Datasets:   1
Resultsets: 10
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       1/23/18 7:14 PM

Dataset                 (1) trees.J4 | (2) trees (3) trees (4) trees (5) trees (6) trees (7) trees (8) trees (9) trees (10) tree
-------------------------------------------------------------------------------------------------------------------------------
soybean                 (10)   91.36 |   91.36     91.95     92.09     92.09     91.50     91.50     91.50     91.50     91.50
-------------------------------------------------------------------------------------------------------------------------------
                             (v/ /*) |  (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)

Key:
(1) trees.J48 '-C 0.1 -M 2' -217733168393644444
(2) trees.J48 '-C 0.2 -M 2' -217733168393644444
(3) trees.J48 '-C 0.3 -M 2' -217733168393644444
(4) trees.J48 '-C 0.4 -M 2' -217733168393644444
(5) trees.J48 '-C 0.5 -M 2' -217733168393644444
(6) trees.J48 '-C 0.6 -M 2' -217733168393644444
(7) trees.J48 '-C 0.7 -M 2' -217733168393644444
(8) trees.J48 '-C 0.8 -M 2' -217733168393644444
(9) trees.J48 '-C 0.9 -M 2' -217733168393644444
(10) trees.J48 '-C 1.0 -M 2' -217733168393644444
```

Figure 3: Showing the accuracies of correctly classified instances at various confidence levels

From the figure above, we can clearly see that the accuracy rate of the correctly classified instances is more or less unaffected by a change in the confidence factor. They are all in the same range of around 91%. It reaches a peak value of 92.09% at 0.5 confidence and later

stabilizes at the 91.50% for all the remaining higher confidence values. This implies pruning of the decision tree is not affecting the classification accuracy of the J48 algorithm. We can deduce that the given dataset has no overfitting or underfitting associated with its data based on the above results.

We have already summarized the pruned and unpruned results of the J48 classifier in the previous section. The results obtained now builds upon the assumption made based on the pruned and unpruned accuracy rates.

minNum parameter

This parameter gives us the minimum number of instances per leaf in the generated tree. It is defaulted to 2 for the REPTree parameter setting in Weka. This parameter can also be used to avoid overfitting of data. Generally, higher values of this algorithm provide simpler tree structures with smaller size.

We test the REPTree classifier with minNum ranging from 0.2 to 0.8 with an increment of 2 and analyze the results. The noPruning option is set to False so that we can get pruned trees with separate values of minNum. The numFolds which indicates the amount opf dat used for pruning is set to the default value 3.

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -m
Analysing:  Percent_correct
Datasets:   1
Resultsets: 4
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       1/23/18 7:42 PM


Dataset                   (1) trees.RE | (2) trees (3) trees (4) trees
----------------------------------------------------------------------
soybean                   (10)   84.76 |   80.23 *   78.04 *   76.72 *
----------------------------------------------------------------------
                               (v/ /*) |   (0/0/1)   (0/0/1)   (0/0/1)


Key:
(1) trees.REPTree '-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(2) trees.REPTree '-M 4 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(3) trees.REPTree '-M 6 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(4) trees.REPTree '-M 8 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
```

Figure 4: Showing the accuracy of the REPTree at multiple values of minNum

We can observe that changing the value of minNum affects the accuracy of the classifier. As the value of minNum increases, the accuracy of the algorithm seems to decrease. (It can also be noticed that the size of the tree also decreases).

We now compare the Root Mean Squared Error values.

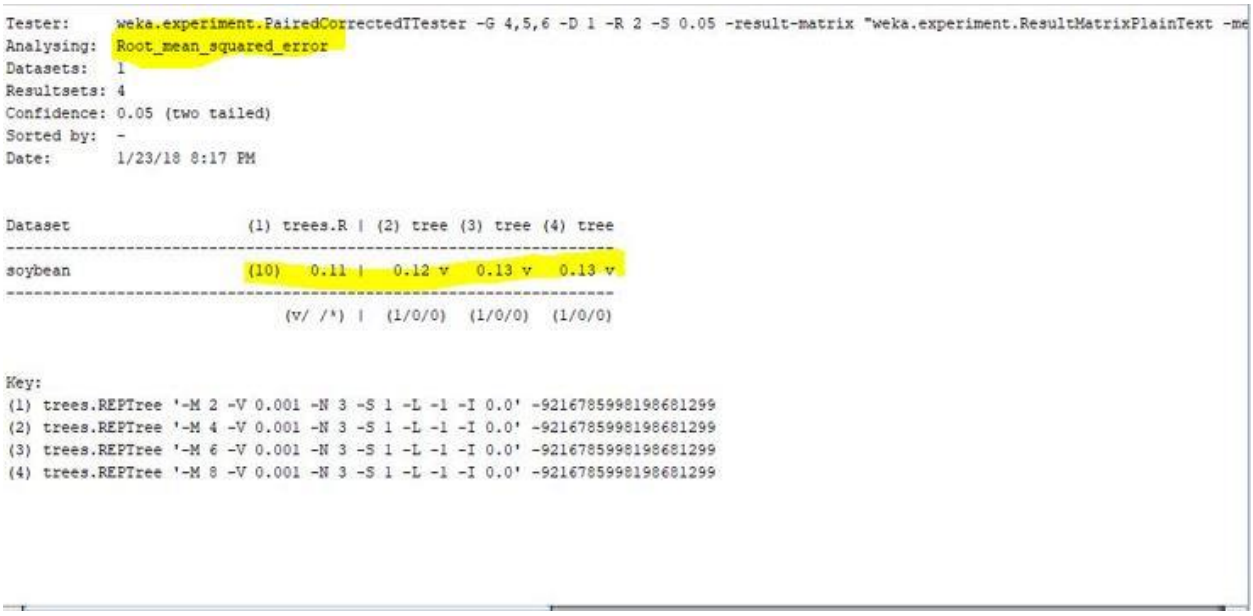


Figure 5: Showing the RSME for different values of minNum

We can clearly see that the RSME is the lowest when minNum is 2 and decreases gradually as the value of minNum is increased.

As explained in the previous section, a low value of RSME is a good measure to avoid overfitting of data. We now have two cases to support our argument that minNum at 0.2 has the highest accuracy of 84.76% and the lowest RSME value of 0.2

We can conclude that this parameter value of minNum = 0.2 can be used as an optimal setting in generating a better predictive model for the soybean dataset.

KValue parameter for RandomTree:

This parameter is used to set the number of randomly chosen attributes when running the Random Tree Classifier. As explained above, Random Tree is a supervised Classifier; it is an ensemble learning algorithm that generates many individual learners. It employs a bagging idea to produce a random set of data for constructing a decision tree.

We test this classifier by using different number of randomly chosen attributes ranging from 0 to 25 with an increment of 5. All the other values are set to default. This algorithm performs no pruning.

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -me
Analysing:  Percent_correct
Datasets:   1
Resultsets: 6
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       1/23/18 8:45 PM


Dataset                   (1) trees.Ra | (2) trees (3) trees (4) trees (5) trees (6) trees
----------------------------------------------------------------------------------------------
soybean                  (100)   84.71 |   83.42     86.86     88.21     87.92     89.18 v
----------------------------------------------------------------------------------------------
                              (v/ /*) |   (0/1/0)   (0/1/0)   (0/1/0)   (0/1/0)   (1/0/0)


Key:
(1) trees.RandomTree '-K 0 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(2) trees.RandomTree '-K 5 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(3) trees.RandomTree '-K 10 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(4) trees.RandomTree '-K 15 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(5) trees.RandomTree '-K 20 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(6) trees.RandomTree '-K 25 -M 1.0 -V 0.001 -S 1' -9051119597407396024
```

Figure 6: Showing the accuracy rates of Random Tree for different k values

We can observe that the accuracy rates show an increase with higher number of attributes chosen. We get the highest accuracy of 89.18% for the largest value of k. The size of the tree is also does not vary much.

We now check for the RSME value.

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mea
Analysing:  Root_mean_squared_error
Datasets:   1
Resultsets: 6
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       1/23/18 8:45 PM


Dataset                   (1) trees.R | (2) tree (3) tree (4) tree (5) tree (6) tree
----------------------------------------------------------------------------------------
soybean                  (100)   0.12 |   0.13     0.11     0.11     0.11     0.10 *
----------------------------------------------------------------------------------------
                              (v/ /*) |  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/0/1)


Key:
(1) trees.RandomTree '-K 0 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(2) trees.RandomTree '-K 5 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(3) trees.RandomTree '-K 10 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(4) trees.RandomTree '-K 15 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(5) trees.RandomTree '-K 20 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(6) trees.RandomTree '-K 25 -M 1.0 -V 0.001 -S 1' -9051119597407396024
```

Figure 7: Showing the RSME for different k values

It can be observed that the RSME is lowest at k = 25 and shows a gradual decrease as the value of k increases. It implies that an increase in the number of attributes lowers the RSME value thereby predicting a better model.

We can also check for the size of the predicted regions to see if the number of attribute values have any impact on the size of the final model.

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -me
Analysing:  Size_of_Predicted_Regions
Datasets:   1
Resultsets: 6
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       1/23/18 8:47 PM


Dataset                   (1) trees.R |  (2) tree  (3) tree  (4) tree  (5) tree  (6) tree
------------------------------------------------------------------------------------------
soybean                  (100)    6.71 |    6.83     6.50     6.22 *    6.07 *    5.94 *
------------------------------------------------------------------------------------------
                              (v/ /*) |  (0/1/0)  (0/1/0)  (0/0/1)  (0/0/1)  (0/0/1)


Key:
(1) trees.RandomTree '-K 0 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(2) trees.RandomTree '-K 5 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(3) trees.RandomTree '-K 10 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(4) trees.RandomTree '-K 15 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(5) trees.RandomTree '-K 20 -M 1.0 -V 0.001 -S 1' -9051119597407396024
(6) trees.RandomTree '-K 25 -M 1.0 -V 0.001 -S 1' -9051119597407396024
```

Figure 8: showing the size of the predicted regions for different k-values

As we can see, the size of the predicted regions gradually decreases with an increase in the number of randomly chosen attributes. This implies that the value of k can be used to generate a decision tree that can be viewed in a presentable visual format.

We can summarize the above results and predict that the classifier works best when it has the required number of randomly chosen attributes. Our results indicate a high accuracy rate and a low RSME at KValue = 25. This can be used as a metric in predicting the optimal models based on this parameter.

4.[Classification – Naïve Bayes algorithm] [10 MARKS]

Suppose we have data on a few individuals randomly examined for basic health check. The following table gives the data on these individuals' health-related attributes.

Body Weight	Body Height	Blood Pressure	Blood Sugar Level	Habit	Class
Heavy	Short	High	3	Smoker	P

Heavy	Short	High	1	Nonsmoker	P
Normal	Tall	Normal	3	Nonsmoker	N
Heavy	Tall	Normal	2	Smoker	N
Low	Medium	Normal	2	Nonsmoker	N
Low	Tall	Normal	1	Nonsmoker	P
Normal	Medium	High	3	Smoker	P
Low	Short	High	2	Smoker	P
Heavy	Tall	High	2	Nonsmoker	P
Low	Medium	Normal	3	Smoker	P
Heavy	Medium	Normal	3	Nonsmoker	N

Required Task:
Use the data together with the Naïve Bayes classifier to perform a new classification for the following new instance:

Body Weight	Body Height	Blood Pressure	Blood Sugar Level	Habit	Class
Heavy	Medium	High	1	Smoker	?

Solution:

Naive-Bayes Classification:

Given n-attributes, we can write Bayes' rule using a product of per attribute probabilities (because of the assumption that the attributes are conditionally independent).

$$P(H \mid E) = P(E_1 \mid H)P(E_3 \mid H)\square\ P(E_n \mid H)P(H) / P(E)$$

Evidence E – instance's non-class attribute values

Event H – class value of instance

P(E) will cancel off during normalization. So, we are not required to evaluate it.

Body Weight

	P	N
Heavy	3	2
Normal	1	1
Low	3	1

Ratio	P	N
Heavy	3/7	2/4
Normal	1/7	1/4
Low	3/7	1/4

Body Height

	P	N
Tall	2	2
Medium	2	2
Short	3	0

Ratio	P	N
Tall	2/7	2/4
Medium	2/7	2/4
Short	3/7	0/4

Blood Pressure

	P	N
High	5	0
Normal	2	4

Ratio	P	N
High	5/7	0/7
Normal	2/7	4/7

Blood Sugar Level

	P	N
1	2	0
2	2	2
3	3	2

Ratio	P	N
1	2	0
2	2	2
3	3	2

Habit

	P	N
Smoker	4	1
Nonsmoker	3	3

Ratio	P	N
Smoker	4/7	1/4
Nonsmoker	3/7	3/4

Class

P	N
7	4

P	N
7/11	4/11

Zero Frequency Problem:

In the above table, we can see that the attribute 'Blood Pressure = High' does not occur with the class value 'N'. There are zero cases of 'N' for 'Blood sugar' in the given sample.

We can also observe the same for the attribute 'Body Height = Short' for the class value 'N', and for another attribute, 'Blood sugar Level = 1' with zero 'N' values.

This indicates a zero-frequency problem, which means

- Probability will be zero:
  - P (Blood Pressure = High | N) = 0
  - P (Body Height = Short | N) = 0
  - P (Blood Sugar level = 1 | N) = 0
- A posteriori probability will also be zero regardless of how likely the other values are.

Remedy:

To counter this problem, we add 1 to the count of every attribute value-class combination (Laplace Estimator)

This will make sure that the probabilities will never be zero. It can be used to stabilize the probability estimates computed from small samples of data as the given one.

We now apply Laplace smoothing and add 1 to all instances except the classifier, Class.

Body Weight		
	P	N
Heavy	4	3
Normal	2	2
Low	4	2

Body Height		
	P	N
Tall	3	3
Medium	3	3
Short	4	1

Blood Pressure		
	P	N
High	6	1
Normal	3	5

Blood Sugar Level		
	P	N
1	3	1
2	3	3
3	4	3

Habit		
	P	N
Smoker	5	2
Nonsmoker	4	4

Class	
P	N
7	4

Ratio	P	N
Heavy	4/10	3/7
Normal	2/10	2/7
Low	4/10	2/7

Ratio	P	N
Tall	3/10	3/7
Medium	3/10	3/7
Short	4/10	1/7

Ratio	P	N
High	6/9	1/6
Normal	3/9	5/6

Ratio	P	N
1	3/10	1/7
2	3/10	3/7
3	4/10	3/7

Ratio	P	N
Smoker	5/9	2/6
Nonsmoker	4/9	4/6

P	N
7/11	4/11

P (Class = P) = 7/11

P (Class = N) = 4/11

Likelihood of the two classes, P and N:

P (Class = P/A) = {P (A/Class = P)} {P (Class = P)}/ P(A)

P (Class = P | Given attribute) = P (Body Weight = Heavy | Class = P) * P (Body Height = Medium | Class = P) * P (Blood Pressure = High | Class = P) * P (Blood Sugar Level = 1 | Class = P) * P (Habit = Smoker | Class = P) * P (Class = P)

= {(4/10) * (3/10) * (6/9) * (3/10) *(5/9)} * (7/11) = **0.00848**

P (Class = N | Given Attribute) =

P (Body Weight = Heavy | Class = N) * P (Body Height = Medium | Class = N) * P (Blood Pressure = High | Class = N) * P (Blood Sugar Level = 1 | Class = N) * P (Habit = Smoker | Class = N) * P (Class = N)

= {(3/7) * (3/7) * (1/6) * (1/7) * (2/6)} * (4/11) = **0.00053**

Conversion into a probability by normalization:

Probability ('P') = 0.00848 / 0.00848 + 0.00053 = 0.00848/0.00901 = 0.9412 = **94.12%**

Probability ('N') = 0.00053 / 0.00848 + 0.00053 = 0.00053 / 0.00901 = 0.0588 = **5.88%**

Normalization value indicates there is a higher probability of Class being 'P' which is around 94.12%

Naïve Bayes predicts that the 'Class' of the patient based on his health checkup is 'P'

Body Weight	Body Height	Blood Pressure	Blood Sugar Level	Habit	Class
Heavy	Medium	High	1	Smoker	P

5. [Association Rules Mining] **[20 MARKS]**

The following table contains transactions for item purchases.

TransID	Items
100	Apple, Bread, Butter, Egg, Orange, Sugar
200	Bread, Butter, Coke, Egg, Sugar
300	Apple, Butter, Ketchup, Potato, Sugar
400	Apple, Coke
500	Bread, Butter, Coke, Egg, Sugar
600	Coke, Ketchup, Orange, Potato, Sugar
700	Egg, Sugar
800	Coke, Egg, Orange
900	Bread, Butter, Egg, Ketchup, Potato, Sugar

a) Follow the steps outlined in Practical 07 and conduct a mining task for Boolean association rules using the Apriori algorithm in Weka.
b) Set different parameters and observe the association rules discovered.
c) Weka provides association evaluation parameters other than support and confidence. Observe the evaluation results by those evaluation parameters of example rules.

Answer:

a) Boolean Association Rules

Although WEKA provides number of association rule solutions, it directly supports only quantitative association rule mining, not Boolean Association Rule mining. This is because WEKA does not accept the transactional database directly but only accepts relational tables as input data format.

We convert the transactional database into relational database format as shown below. The transformed table takes transactions as rows and items as columns. The occurrence of an item in a transaction is represented by a Boolean value 1. The absence of an item is depicted by a missing value, the character '?' in the ARFF file format.

transID	Apple	Bread	Butter	Coke	Egg	Ketchup	Orange	Potato	Sugar
100	1	1	1	0	1	0	1	0	1
200	0	1	1	1	1	0	0	0	1
300	1	0	1	0	0	1	0	1	1
400	1	0	0	1	0	0	0	0	0
500	0	1	1	1	1	0	0	0	1
600	0	0	0	1	0	1	1	1	1
700	0	0	0	0	1	0	0	0	1
800	0	0	0	1	1	0	1	0	0
900	0	1	1	0	1	1	0	1	1

Using the relational data table, we create the ARFF file with the declaration as shown below.

```
@relation trans

@attribute transID numeric
@attribute Apple {0,1}
@attribute Bread {0,1}
@attribute Butter {0,1}
@attribute Coke {0,1}
@attribute Egg {0,1}
@attribute Ketchup {0,1}
@attribute Orange {0,1}
@attribute Potato {0,1}
@attribute Sugar {0,1}

@data
100,1,1,1,?,1,?,1,?,1
200,?,1,1,1,1,?,?,?,1
300,1,?,1,?,?,1,?,1,1
400,1,?,?,1,?,?,?,?,?
500,?,1,1,1,1,?,?,?,1
600,?,?,?,1,?,1,1,1,1
700,?,?,?,?,1,?,?,?,1
800,?,?,?,1,1,?,1,?,?
900,?,1,1,?,1,1,?,1,1
```

Another approach:

We can also create a csv file using the transactional database and replace the o's with blank spaces. We can then change the data which is in numeric format to binary by applying the numericToBinary filter in Weka.

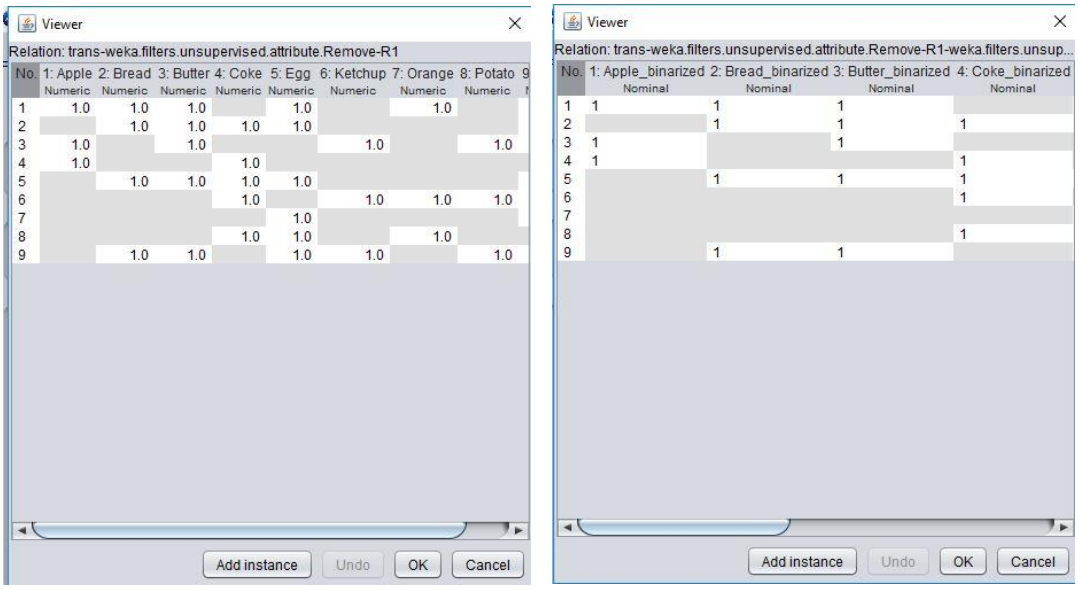


Figure 9: Converting the numeric values to binary using the Weka filter

We can choose either of the two approaches mentioned above to create a readable dataset for Weka. Weka can run both the CSV as well as the ARFF file formats.

We now delete the transID attribute and set the class to "No Class" on the class selection tab as shown in the figure below. We can also observe the 6 missing values in the screenshot for the "Apple" attribute. This means of the total 9 transaction IDs in the dataset, only 3 have apple indicating a support of 3/9 = 33.33%

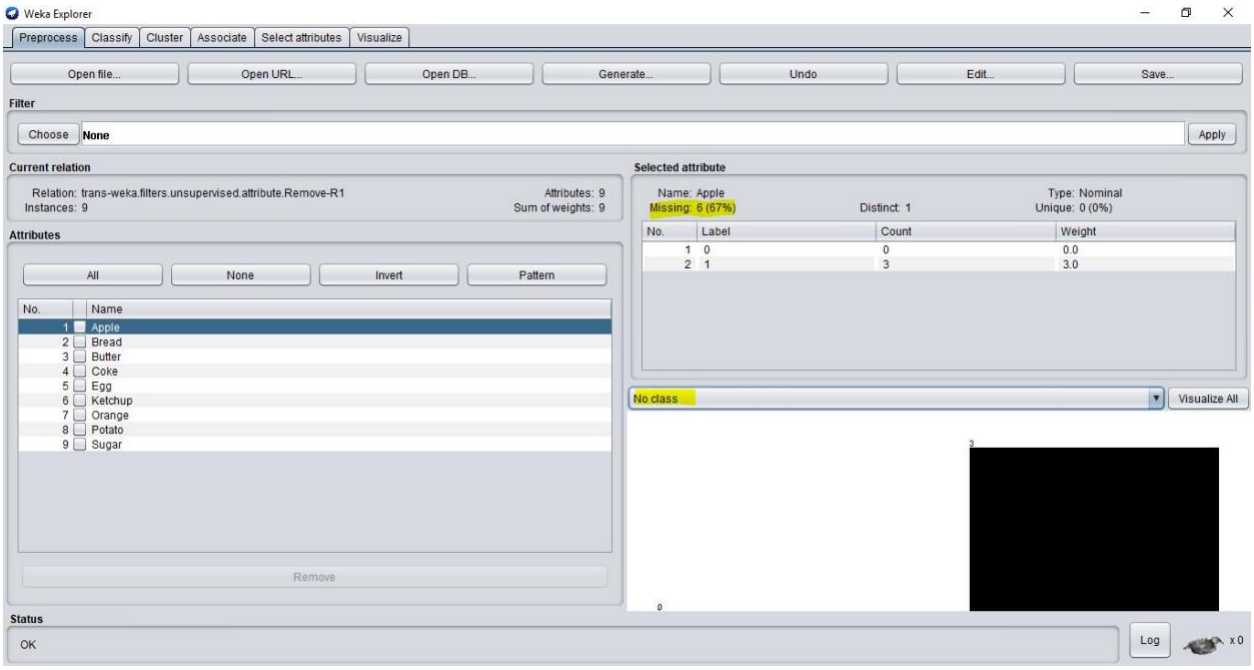


Figure 10: Showing the "no class" selection and the missing values for the Apple attribute

Apriori Algorithm:

Objective of using Apriori algorithm is to find frequent item-sets and association between them i.e. association rule. These rules can be very helpful in analyzing customer behavior in retail sales, banking system etc.,

We now run the Apriori algorithm on the current dataset with the default parameters in Weka.

- The *lowerBoundMinSupport* refers to the minimum support threshold which is at 0.1.
- The parameter *delta* refers to the fraction of support to be iteratively reduced from the maximum level, specified against the parameter *upperBoundMinSupport* towards the minimum *lowerBoundMinSupport*. This is set to its default value at 0.05
- The minimum confidence is set as a fraction against the parameter *minMetric* after selecting *Confidence* for the parameter *metricType*. This is set to its default value of 0.9
- The maximum number of rules is set to 10

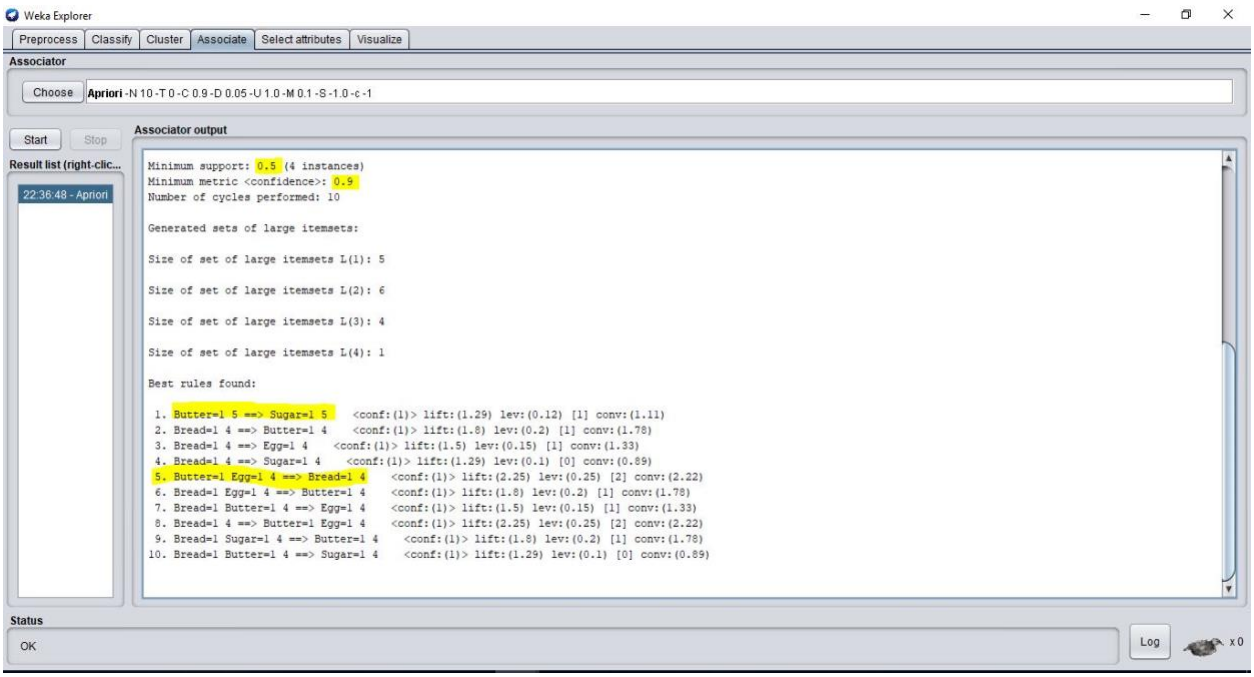


Figure 11: Showing the Apriori results for the default parameters

If we look at the first rule generated, we can see that the customer who buys Butter has tendency to buy Sugar as well. The confidence of this rule is 100% which is very believable. Using this logic, we can interpret all the other rules that the algorithm has revealed.

Item=1 in a rule indicates the existence of the item. The integer after the antecedent of a rule refers to the actual level of support for the itemset in terms of absolute number of transactions. The integer after the decedent of a rule refers to the support of the rule.

For instance, in rule 5, **integer 4 after Butter = 1 and Egg = 1** indicates that the support for {Butter, Egg} is 4 transactions of the total 9 (4/9 = 44.44%). The integer 4 after Bread=1 shows that the support for {Bread, Butter, Egg} is 4 transactions (44.44%). These rules are found with a minimum confidence of 90%. The minimum support is counted as 4 instances. All the best association rules found will satisfy these criteria. For example, the best rules generated are shown in the expression below with the confidence and the support factors.

Butter -> Sugar Confidence = 100% and Support = 55.55%

Bread -> Butter Confidence = 100% and Support = 44.44%

We can quickly summarize the results as below:

- Minimum support was 0.5 (4 instances)
- Number of rules produced was 10
- All the rules produced had a confidence level of 1.0 or 100%
- The largest itemset was 4
- There was one itemset {Butter, Bread, Egg, Sugar} with a size of 4.

We can also observe other parameters like lift and leverage which can help us understand whether there is a positive or negative correlation of occurrence of items in the dataset. This will be discussed in detail in the later sections.

b) Setting different parameters and comparing the association rules.

We now change a few parameter settings and analyze the results.

Confidence metric and numRules:

If we try to find a number of high confidence rules, we may end up finding a huge number of very low support and high confidence rules in a large dataset. We can have a large number of 100% confidence rules but they may not very relevant since they have minimal support values.

We try to change the confidence metric and give it different values ranging from 0.1 to 1.0 with an increment of 0.1. However, we observe that leaving the numRules parameter to 10 is not going to affect the results as all the top 10 rules actually produced had a confidence level of 100%.

We try to tweak the numRules setting and give it different values from 10 to 50 and analyze the results. The confidence metric is set to the default 0.9.

numRules = 10:

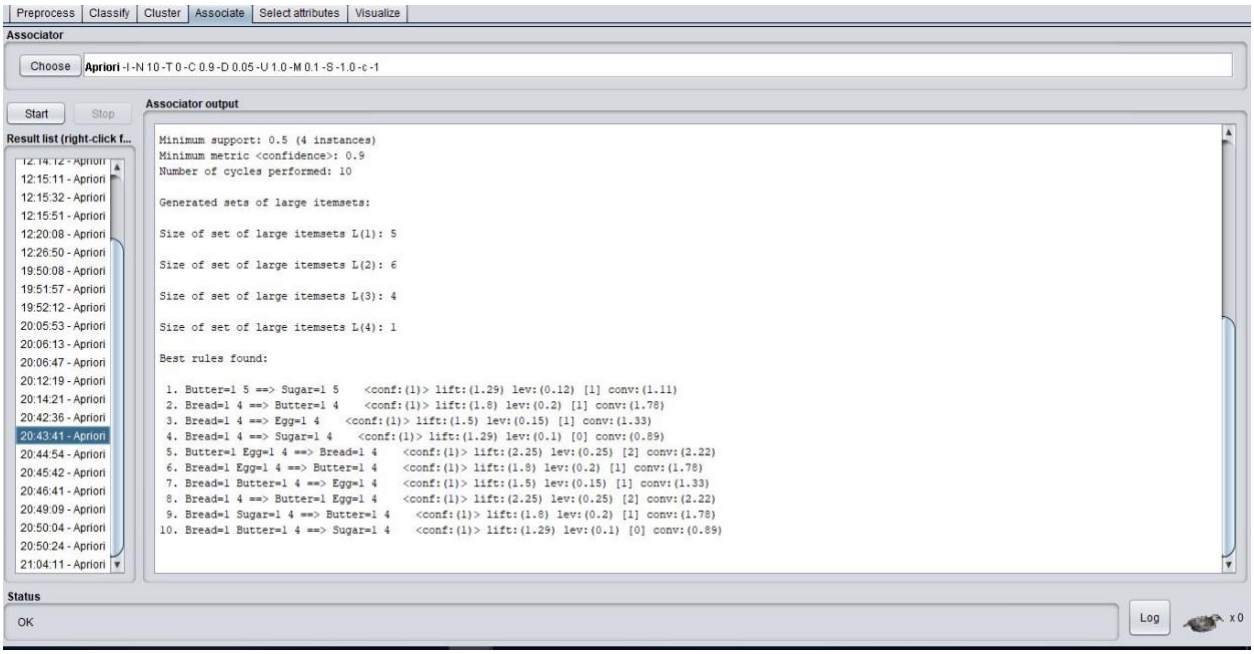


Figure 12: Showing the minimum support at 4 instances with 10 rules generated

numRules = 20

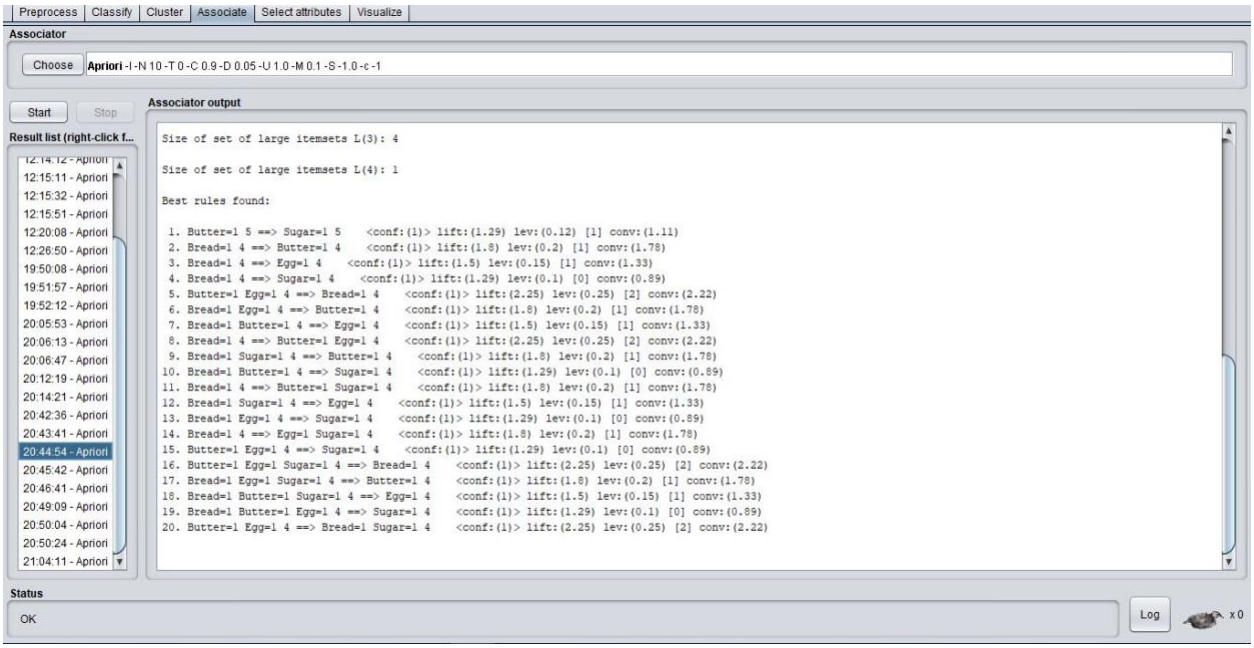


Figure 13: Showing the minimum support at 4 instances with 20 rules generated

numRules = 30

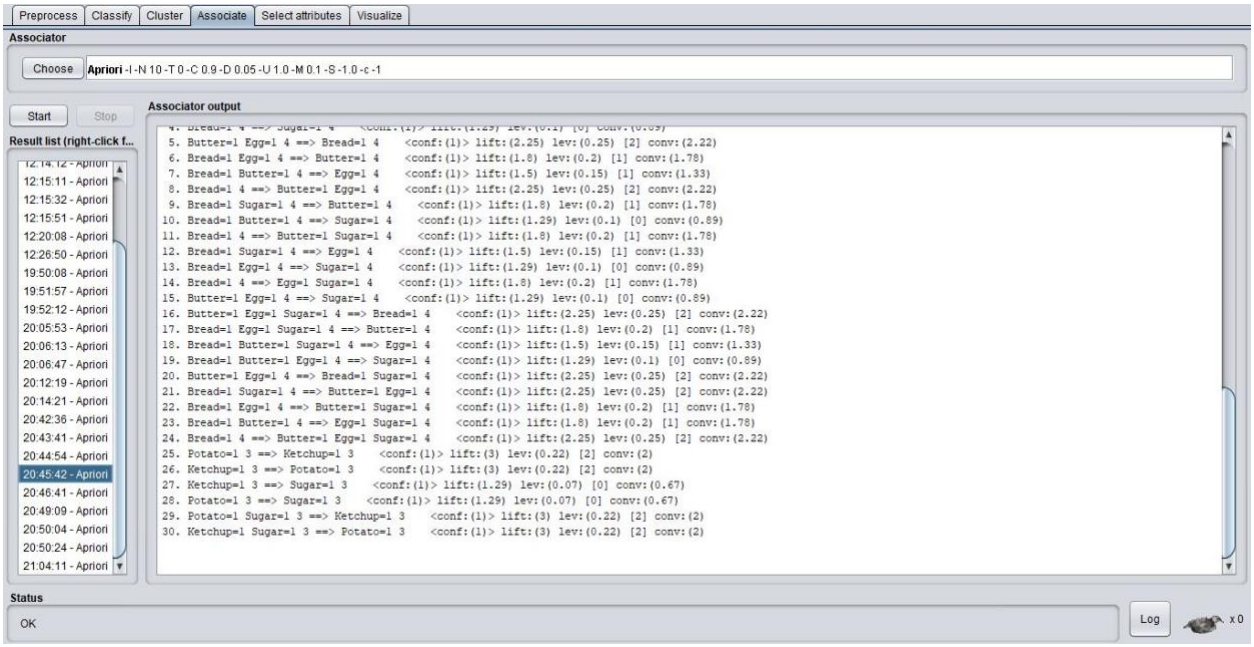


Figure 14: Showing the minimum support at 3 instances with 30 rules generated

numRules = 40

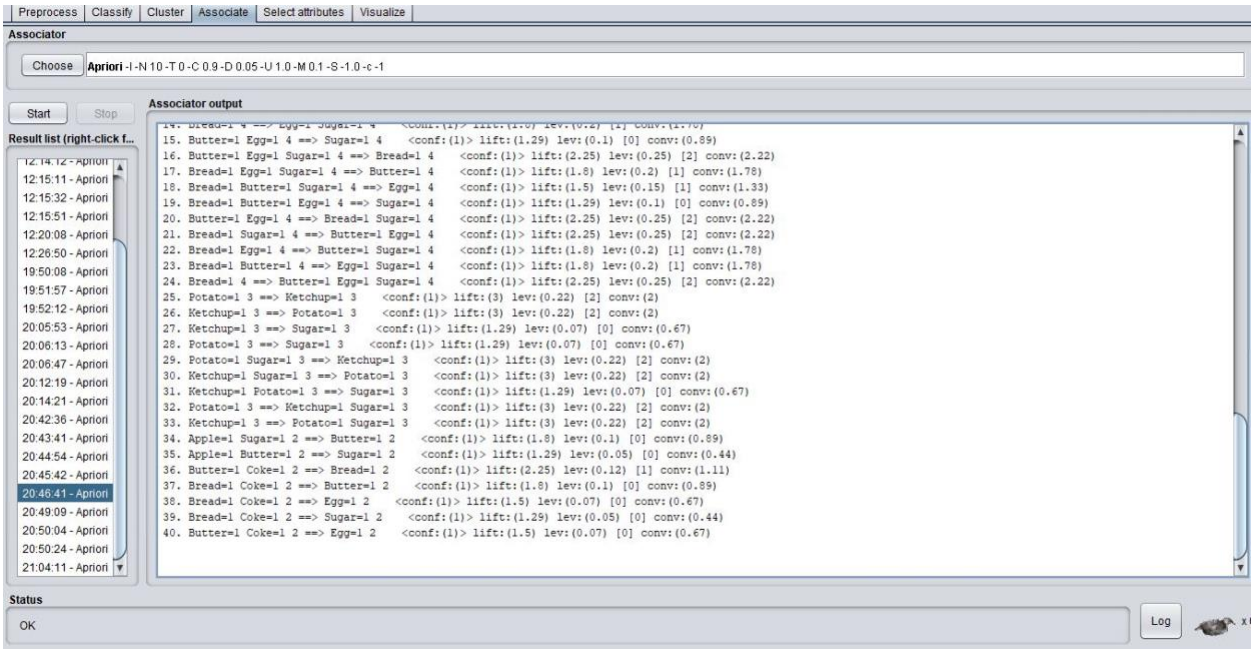


Figure 15: Showing the minimum support at 2 instances with 40 rules generated

numRules = 50

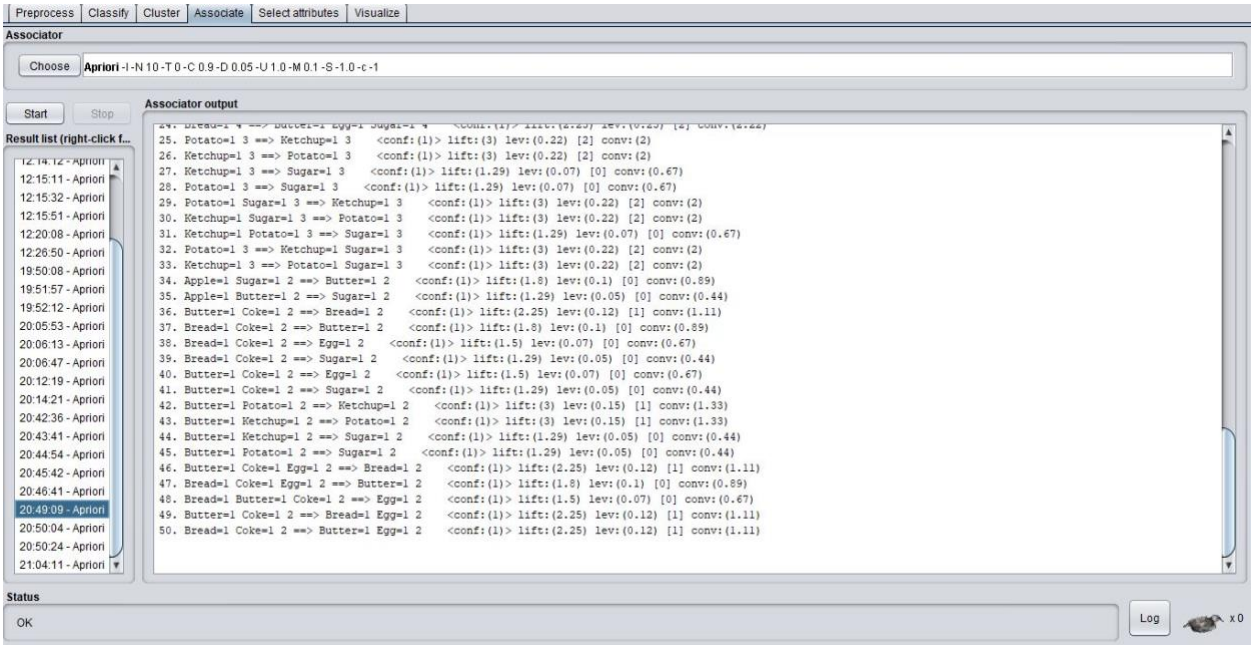


Figure 16: Showing the minimum support at 2 instances with 50 rules generated

Summary Table:

Confidence metric	Minimum Support (instances)	Number of Rules	No of 5 item-sets produced	No of 4 item-sets produced	No of 3 item-sets produced	No of 2 item-sets produced	No of 1 item-set produced
0.9	0.5 (4)	10	0	1	4	6	5
0.9	0.5 (4)	20	0	1	4	6	5
0.9	0.35 (3)	30	0	1	5	11	9
0.9	0.25 (2)	40	1	6	15	20	9
0.9	0.25 (2)	50	1	6	15	20	9
0.9	0.25 (2)	60	1	6	15	20	9

As we can observe, by generating more rules using the Apriori we can see an increase in the number of frequent item-sets. But the minimum support for this rule gradually decreases. For example, when the number of rules is 40, we can find a 5 itemset which satisfies the 90% confidence metric but the support is only 0.25% or just 2 instances out of the total 9. This is not very useful as the support is very low.

Our goal is find a frequent itemset with a high support and high confidence. From the table we can find a single itemset which satisfies a minimum support of 50% and confidence of 90%.

We can use the outItemSets parameter and set it to “True” in order to see the itemset which satisfy the criteria.

{Bread, Butter, Egg, Sugar} occurs in 4 instances and satisfies the criteria for 50% minimum support and 90% confidence as shown in the figure below.

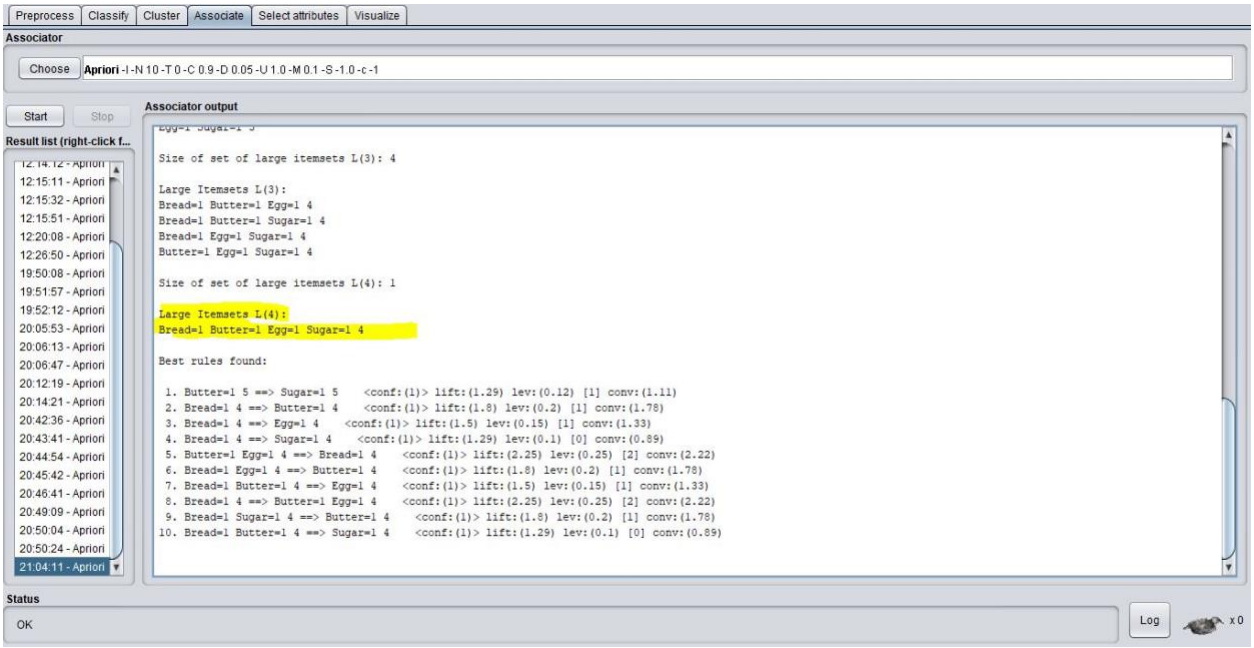


Figure 17: Showing the largest itemset and the number of instances it occurs

If we further increase the number of rules to 100, the minimum support falls to a low value of 1 instance or 10 % which is irrelevant in finding associations in the dataset.

We now try to increase the support to 75% and see there are any rules generated. We find no rules with this setting.

We set the lowerBoundMinSupport to 0.6 and check if it generates any rules. We find a rule of 2 itemset {Butter, Sugar} with 5 instances generated. This was already observed when we ran the Apriori with the default settings.

All in all, the largest itemset generated was {Butter, Bread, Egg, Sugar} with 4 instances.

The basic purpose of Apriori is to find the association between the items and how frequently they occur. We can set different values for confidence and support and base our analysis on the association rules generated as explained above.

c)Other evaluation parameters – lift, conversion, leverage

We now evaluate the other parameters lift, conversion and leverage to determine the association of items based on these factors.

Lift:

Lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model.

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \times \text{supp}(Y)}$$

We now run the Apriori in the default setting with a confidence of 0.9 and numRules at 40.

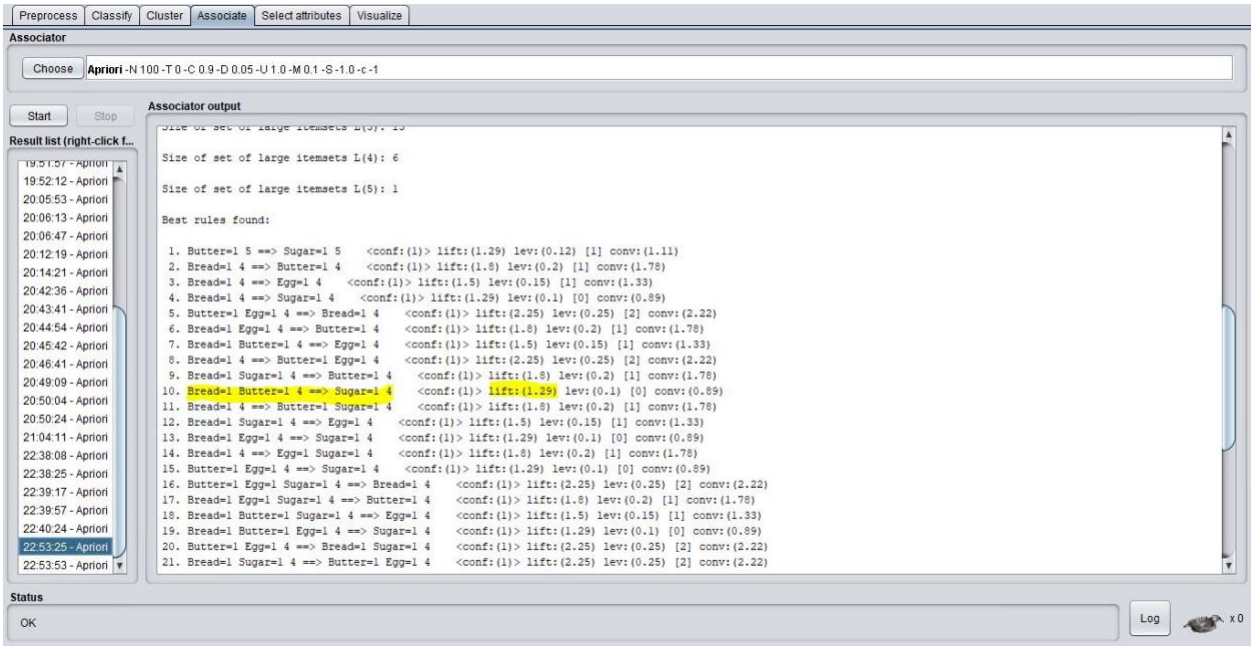


Figure 18: Showing the lift value for rule 10

As we can observe, all the lift values in the rules generated are greater than 1. This implies, that the occurrence of one item has a positive impact on the occurrence of the other item.

For example, in rule 10, the occurrence of {Bread, Butter} has a positive impact on the occurrence of {Sugar}. This rule has a lift value of 1.29 which is greater than 1. We can presume that if {Bread, Butter} occur together, there is a high chance of the customer buying Sugar as well. Of course, the support for this rule is only 4 instances which is competent.

If we get a lift value of less than 1 we can assume that the two items are negatively correlated. That means, the two items appear less often than expected.

A lift value near 1 indicates that both items appear almost as often together as expected; this means that the occurrence of one item has almost no effect on the occurrence of the other or that both the items have zero correlation.

Lift can be an easy tool to identify the correlation between the items of a given dataset. We can set the minimum value of life using the minMetric and easily gauge the correlation between two items.

Leverage:

Leverage and lift measure similar things, except that leverage measures the difference between the probability of co-occurrence of L and R (in a given rule L -> R) as the independent probabilities of each of L and R, i.e.,

$$\textit{leverage} = \mathbf{Pr}(L, R) - \mathbf{Pr}(L).\mathbf{Pr}(R).$$

In comparison,

$$\textit{lift} = \mathbf{Pr}(L, R) / \mathbf{Pr}(L).\mathbf{Pr}(R)$$

In other words, leverage measures the proportion of additional cases covered by both L and R above those expected if L and R were independent of each other. Thus, for leverage, values above 0 are desirable, whereas for lift, we want to see values greater than 1.

We change the metricType to Leverage and minmetric to 0.1 in Weka and generate the rules.

We now look at rule 1 in the figure below. We can see the leverage value is at 0.25 which implies a positive correlation between {Bread} and {Butter, Egg}. This association indicates the highest leverage generated rule with 4 instances. We can also observe another association between {Butter, Egg} and {Bread, Sugar} with the same leverage and confidence.

Leverage is another simple way to analyze the correlation of items in the dataset.

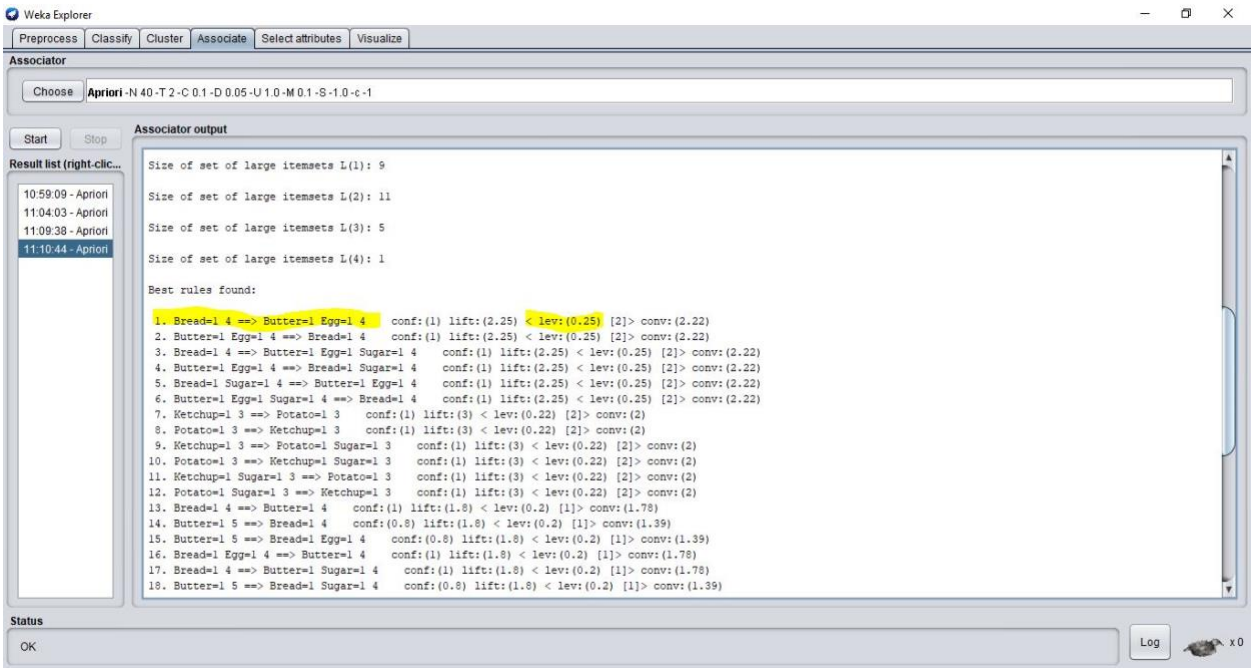


Figure 19: Showing the leverage value for rule 1

Conviction:

Conviction is similar to lift, but it measures the effect of the right-hand-side not being true. It also inverts the ratio. So, conviction is measured as:

$$\textit{conviction} = \mathbf{Pr}(L).\mathbf{Pr}(\text{not } R) \,/\, \mathbf{Pr}(L,R).$$

Thus, conviction, in contrast to lift is not symmetric (and also has no upper bound).

We now change the metricType to Conviction and generate the rules in Weka. We can observe the conviction value of 2.25 between the items {Bread} and {Butter, Egg}

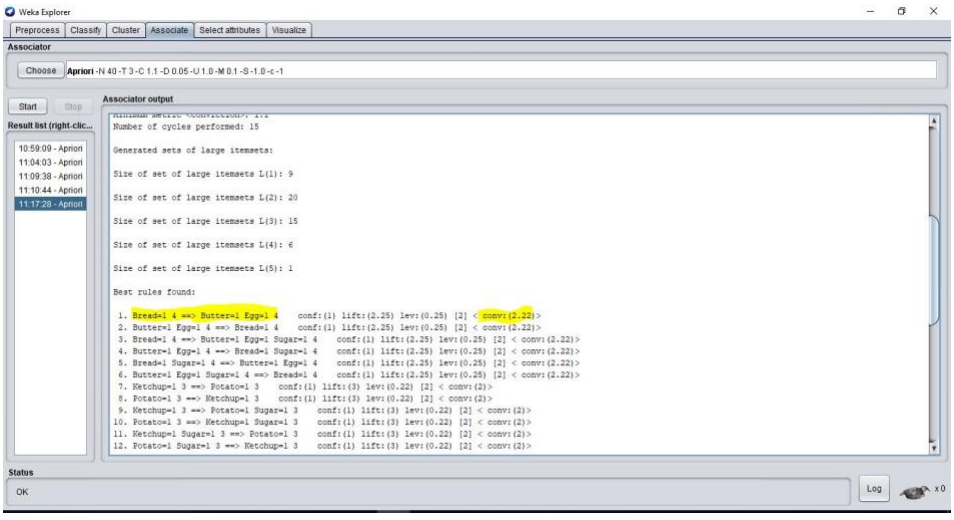


Figure 20: Showing the leverage value for rule 1

In most cases, it is sufficient to focus on a combination of support, confidence, and either lift or leverage to quantitatively measure the "quality" of the rule. However, the real value of a rule, in terms of usefulness and actionability is subjective and depends heavily of the particular domain and business objectives.

6. [Clustering] [10 MARKS]

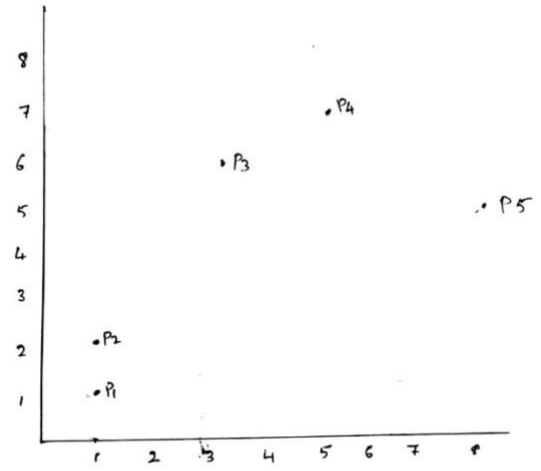
Consider the following 2-dimensional point data set presented in (x, y) coordinates: $P_1(1,1)$, $P_1(1,2)$, $P_1(3,6)$, $P_1(5,7)$, $P_1(8,5)$.
Apply the hierarchically clustering method (using Agglomerative algorithm) to get final two clusters.
(Use the Manhattan distance function to measure the distance between points and use the single-linkage scheme to do clustering)

Answer:

Hierarchical Clustering – Agglomerative Clustering

We consider the following 2-dimensional point data set presented in (x, y) coordinates: $P_1(1,1)$, $P_2(1,2)$, $P_3(3,6)$, $P_4(5,7)$, $P_5(8,5)$ for agglomerative clustering.

Points can be shown in the figure below



We use the Manhattan distance function and the single-linkage method to do agglomerative hierarchical clustering of these data.

Manhattan distance (City-block distance):

This distance is simply the addition of all the distances across dimensions, calculated by the following function: Manhattan distance (A, B) $=\sum_i |(x_i - y_i)|$

The process is described as a sequence of the following steps:

Step 1:

At the beginning, each element P_i is a cluster on its own, and therefore, the set of clusters are:

$C = \{\{P_1\}, \{P_2\}, \{P_3\}, \{P_4\}, \{P_5\}\}$

Step 2 (First iteration):

	P_1	P_2	P_3	P_4	P_5
P_1	×	1	7	10	11
P_2	1	×	6	9	10
P_3	7	6	×	3	6
P_4	10	9	3	×	5
P_5	11	10	6	5	×

We seek to find the minimum Manhattan distance between the points. We can see that this value is 1 and occurs between the points P_1, and P_2. So, we combine P_1 and P_2 to form the cluster P_1P_2, and remove those as individual clusters from the set to get the new cluster set:

$C_1 = \{\{P_1, P_2\}, \{P_3\}, \{P_4\}, \{P_5\}\}$

Step 3 (Second iteration)

	P_1, P_2	P_3	P_4	P_5
P_1, P_2	×	6	9	10
P_3	6	×	3	6
P_4	9	3	×	5
P_5	10	6	5	×

In the cluster set C_1, the minimum Manhattan distance is 3 between P_3, and P_4. By combining P_3, and P_4 we get the cluster set:

$C_2 = \{\{P_1, P_2\}, \{P_3, P_4\}, \{P_5\}\}$

By applying the single-linkage strategy, the closest point is used as the representative of the cluster and is used for the distance calculation between the cluster and other one.

Step 4 (Third iteration)

	P_1, P_2	P_3, P_4	P_5
P_1, P_2	×	6	10
P_3, P_4	6	×	5
P_5	10	5	×

In C_2, the minimum distance is 5 (using single-linkage strategy). This is between P_5 and P_3P_4. By combining these, we get the cluster set:

$C_3 = \{\{P_1, P_2\}, \{P_3, P_4, P_5\}\}$

Step 5 (Fourth iteration)

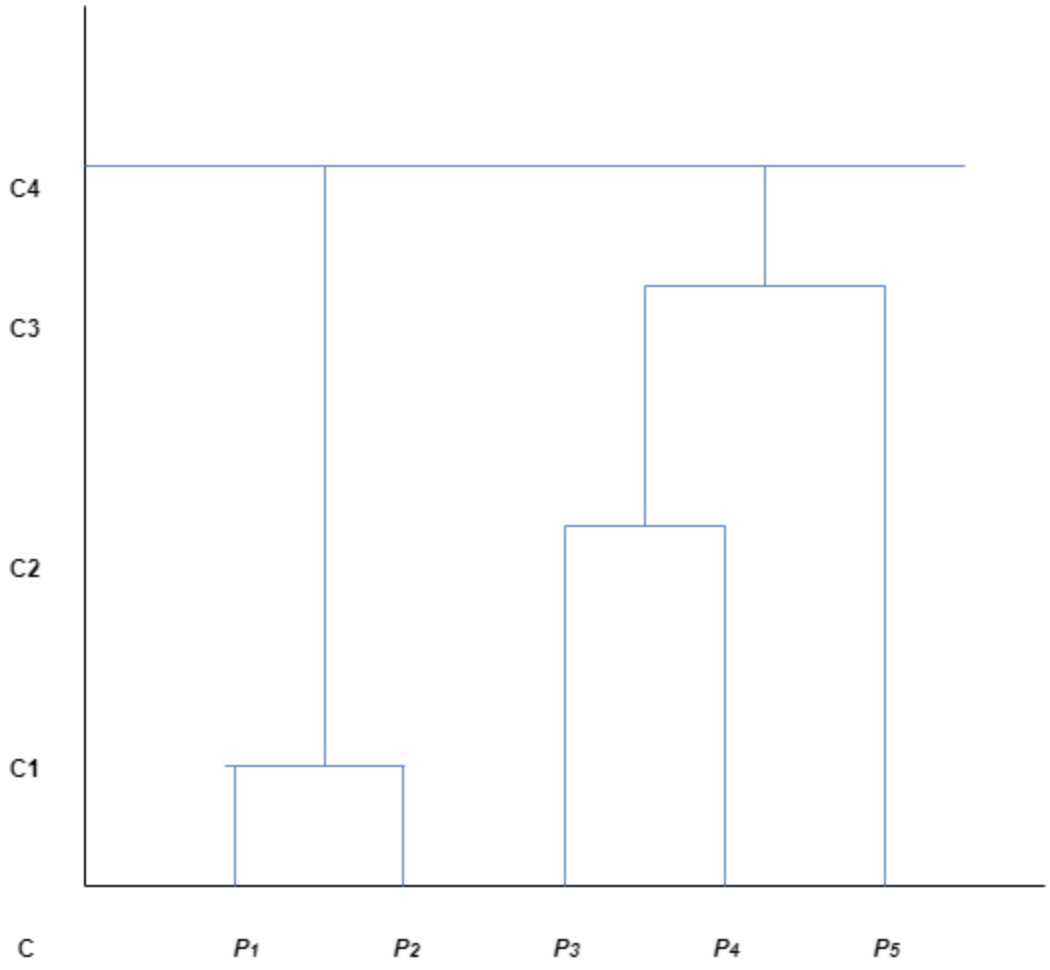
	P_1, P_2	P_3, P_4, P_5
P_1, P_2	×	6
P_3, P_4, P_5	6	×

All points are in one cluster C_4. The clustering process is complete. The minimum distance between the points is at 6.

The final two clusters are $\{P_1, P_2\}$ and $\{P_3, P_4, P_5\}$

Dendrogram:

The corresponding dendrogram formed from the hierarchy is presented here:



This demonstrates that in hierarchical clustering a nested set of clusters is created and each level in the hierarch has a separate set of clusters. At the root level, all items belong to one cluster, and at the leaf level each item is in its own unique cluster.

The two final clusters and formed between {P1, P2} and {P3, P4, P5} at minimum manhattan distance of 6 between them.