Data Warehousing & Data Mining LAB - G2
EXPERIMENT 6

- ASHISH KUMAR
- 2K18/SE/041

Aim:- Perform Association Rule Mining on dataset using Apriori Algorithm.

Theory:-

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently an itemset occurs in a transaction.

We would be using Apriori algorithms for this purpose

Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties.

The key concept of Apriori algorithm is its anti-monotonicity of support measure. Apriori assumes that all subsets of a frequent itemset must be frequent (Apriori property). If an itemset is infrequent, all its supersets will be infrequent. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

The Apriori algorithm is used for mining frequent itemsets and devising association rules from a transactional database. The parameters "support" and "confidence" are used. Support refers to items' frequency of occurrence; confidence is a conditional probability. Items in a transaction form an item set. The algorithm begins by identifying frequent, individual items (items with a frequency greater than or equal to the given support) in the database and continues to extend them to larger, frequent itemsets.

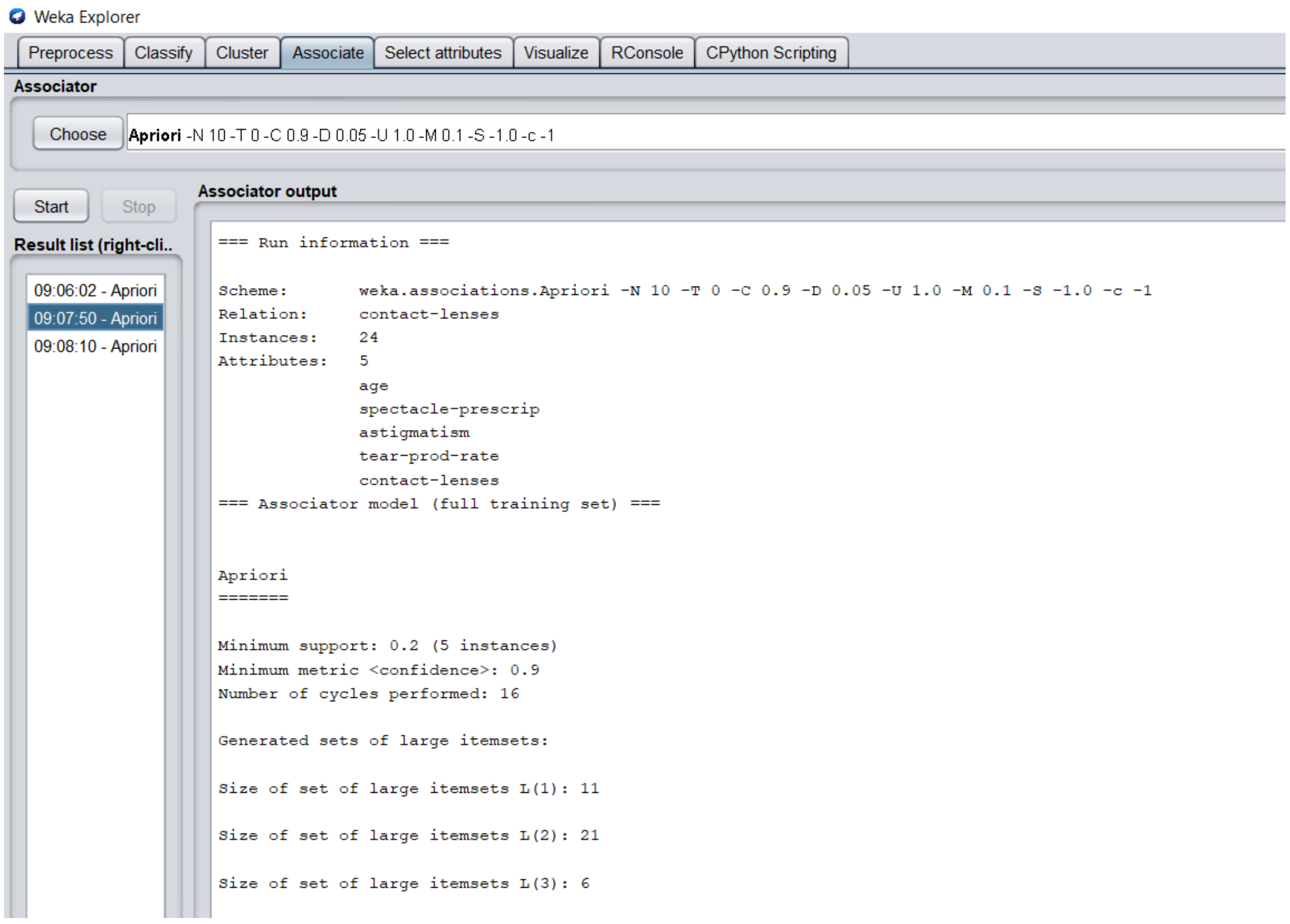
Procedure:

1. Go to WEKA Explorer.
2. Choose dataset in WEKA / data (contact-lenses.arff).
3. Go to Associate tab.
4. Choose an algorithm.
5. Click start.
6. Navigate to Associate tab and under associator, choose Apriori and then hit start.

Note: Ma'am I have tried to use segment-test.arff dataset, but for that start button is not active. So I have to use contact-lenses.arff dataset for associate rule mining.

OUTPUT-

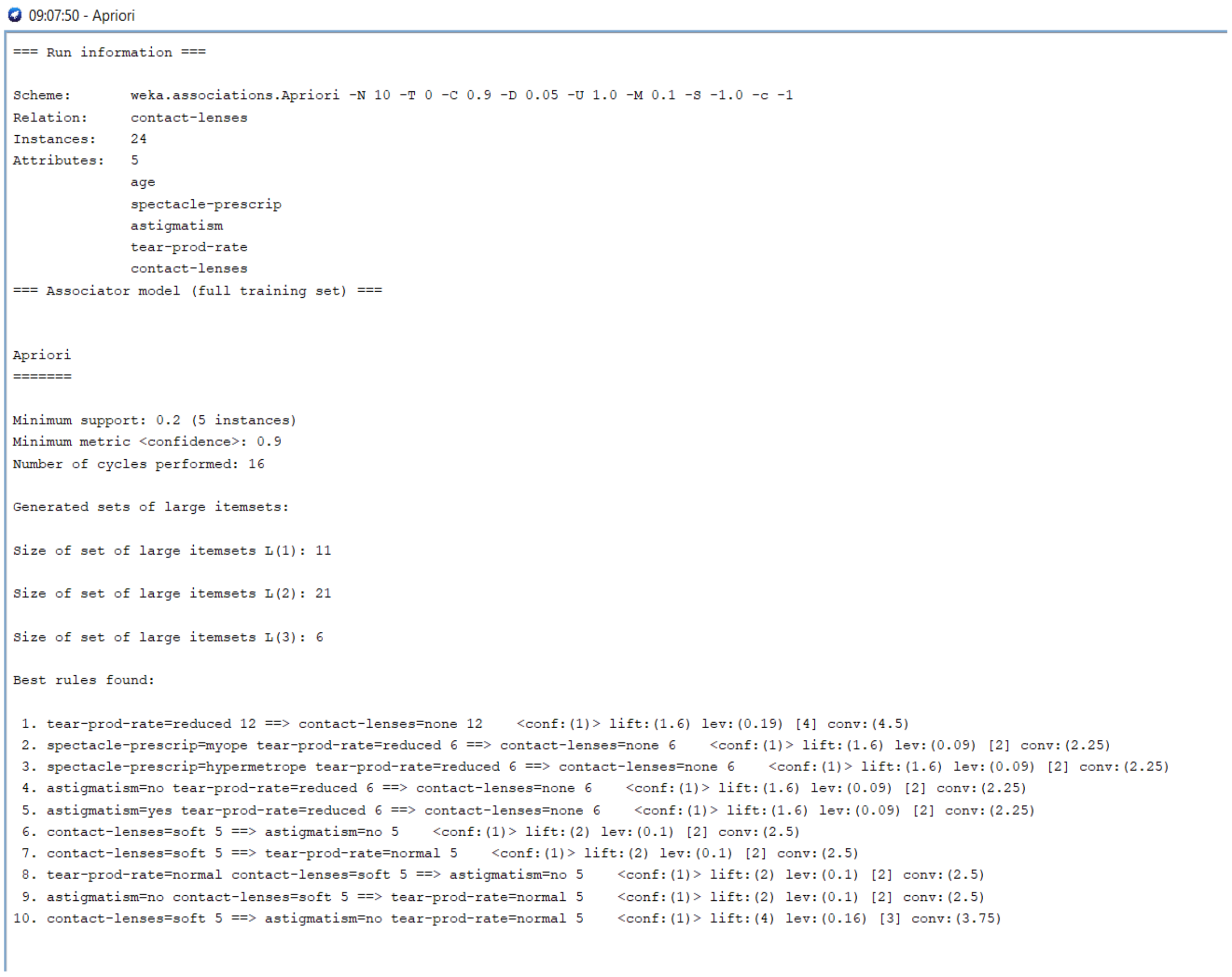
Using Apriori Algorithm



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     contact-lenses
Instances:    24
Attributes:   5
              age
              spectacle-prescrip
              astigmatism
              tear-prod-rate
              contact-lenses
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 21

Size of set of large itemsets L(3): 6
```

Zoomed Output View


09:07:50 - Apriori

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     contact-lenses
Instances:    24
Attributes:   5
              age
              spectacle-prescrip
              astigmatism
              tear-prod-rate
              contact-lenses
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 21

Size of set of large itemsets L(3): 6

Best rules found:

 1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12    <conf:(1)> lift:(1.6) lev:(0.19) [4] conv:(4.5)
 2. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 4. astigmatism=no tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 5. astigmatism=yes tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 6. contact-lenses=soft 5 ==> astigmatism=no 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
 7. contact-lenses=soft 5 ==> tear-prod-rate=normal 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
 8. tear-prod-rate=normal contact-lenses=soft 5 ==> astigmatism=no 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
 9. astigmatism=no contact-lenses=soft 5 ==> tear-prod-rate=normal 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
10. contact-lenses=soft 5 ==> astigmatism=no tear-prod-rate=normal 5    <conf:(1)> lift:(4) lev:(0.16) [3] conv:(3.75)
```

Findings and Learning:

1. We learnt to use Association rule mining in WEKA.
2. We learnt the use of Apriori Algorithm in WEKA.

Aim:- Implement Decision Tree Classification Algorithm in any Language.

Theory:-

Decision Tree is a **supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

Advantages of the Decision Tree:

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Pseudocode :

1. Find the best attribute and place it on the root node of the tree.
2. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.
3. Find leaf nodes in all branches by repeating 1 and 2 on each subset.

While implementing the decision tree we will go through the following two phases:

1. Building Phase
- Preprocess the dataset.
- Split the dataset from train and test using Python sklearn package.
- Train the classifier.
2. Operational Phase
- Make predictions.
- Calculate the accuracy.

In Python, sklearn is the package which contains all the required packages to implement Machine learning algorithm. You can install the sklearn package by following the commands given below :

pip install -U scikit-learn

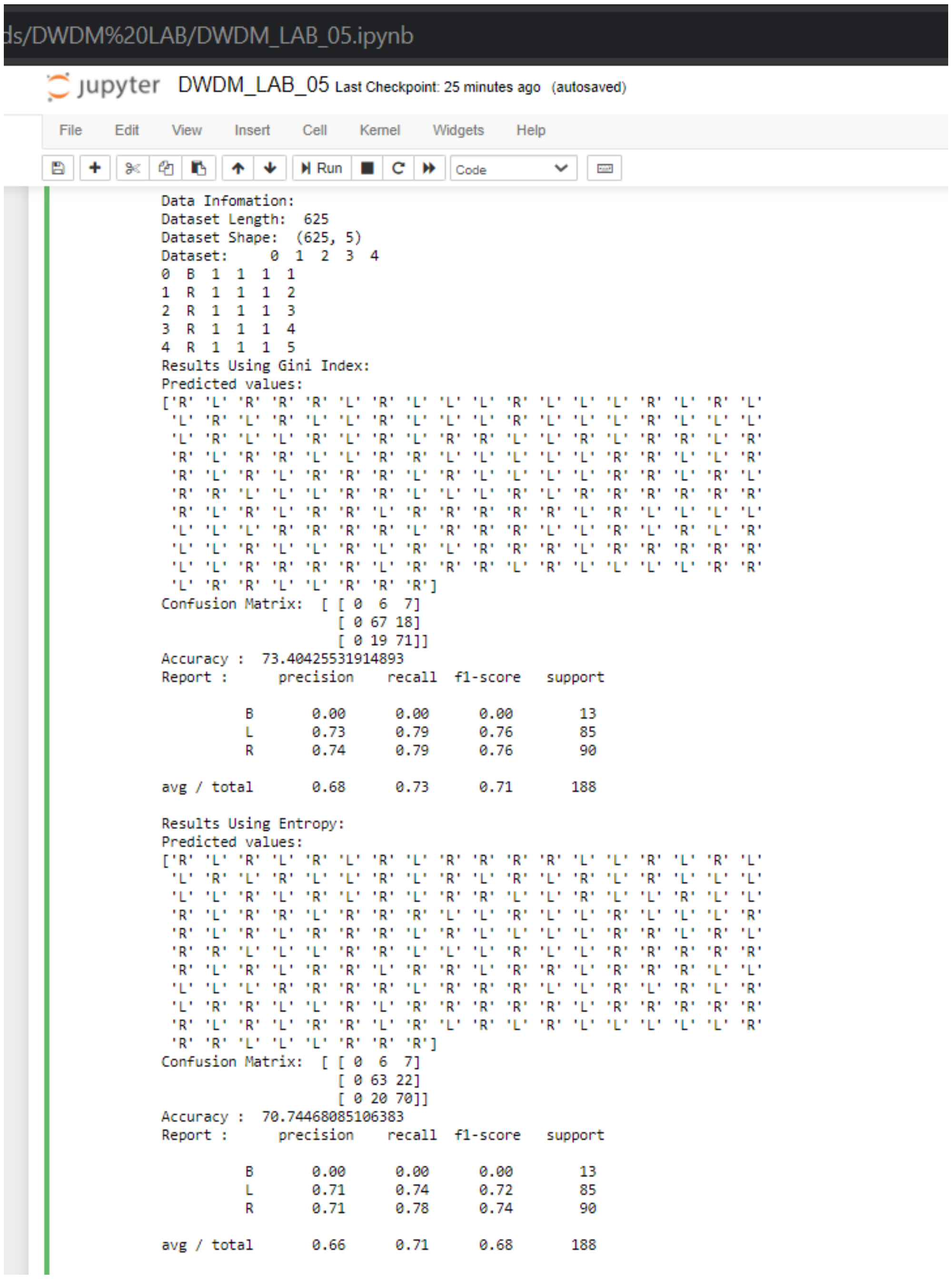
Source Code (in python):

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
    balance_data = pd.read_csv( 'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
    sep= ',', header = None)

    # Printing the dataswet shape
    print("Data Infomation:")
    print ("Dataset Length: ", len(balance_data))
    print ("Dataset Shape: ", balance_data.shape)

    # Printing the dataset obseravtions
    print ("Dataset: ",balance_data.head())
    return balance_data

# Function to split the dataset
def splitdataset(balance_data):

    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.3, random_state = 100)

    return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)
```

```
 # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))
```

```
# Driver code
def main():

    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
    main()
```

OUTPUT-


ls/DWDM%20LAB/DWDM_LAB_05.ipynb
jupyter DWDM_LAB_05 Last Checkpoint: 25 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run Code

```
Data Infomation:
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [ [ 0  6  7]
                     [ 0 67 18]
                     [ 0 19 71]]
Accuracy :  73.40425531914893
Report :       precision    recall  f1-score   support

          B       0.00      0.00      0.00        13
          L       0.73      0.79      0.76        85
          R       0.74      0.79      0.76        90

avg / total       0.68      0.73      0.71       188

Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [ [ 0  6  7]
                     [ 0 63 22]
                     [ 0 20 70]]
Accuracy :  70.74468085106383
Report :       precision    recall  f1-score   support

          B       0.00      0.00      0.00        13
          L       0.71      0.74      0.72        85
          R       0.71      0.78      0.74        90

avg / total       0.66      0.71      0.68       188
```

Findings and Learning:

- We have successfully implemented Decision tree in python 3.
- We have learned the nuances of the Decision tree learning.
- We have learnt about the applications, strengths and weaknesses of Decision tree Learning.