# COSC 3337 : Data Science I


N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

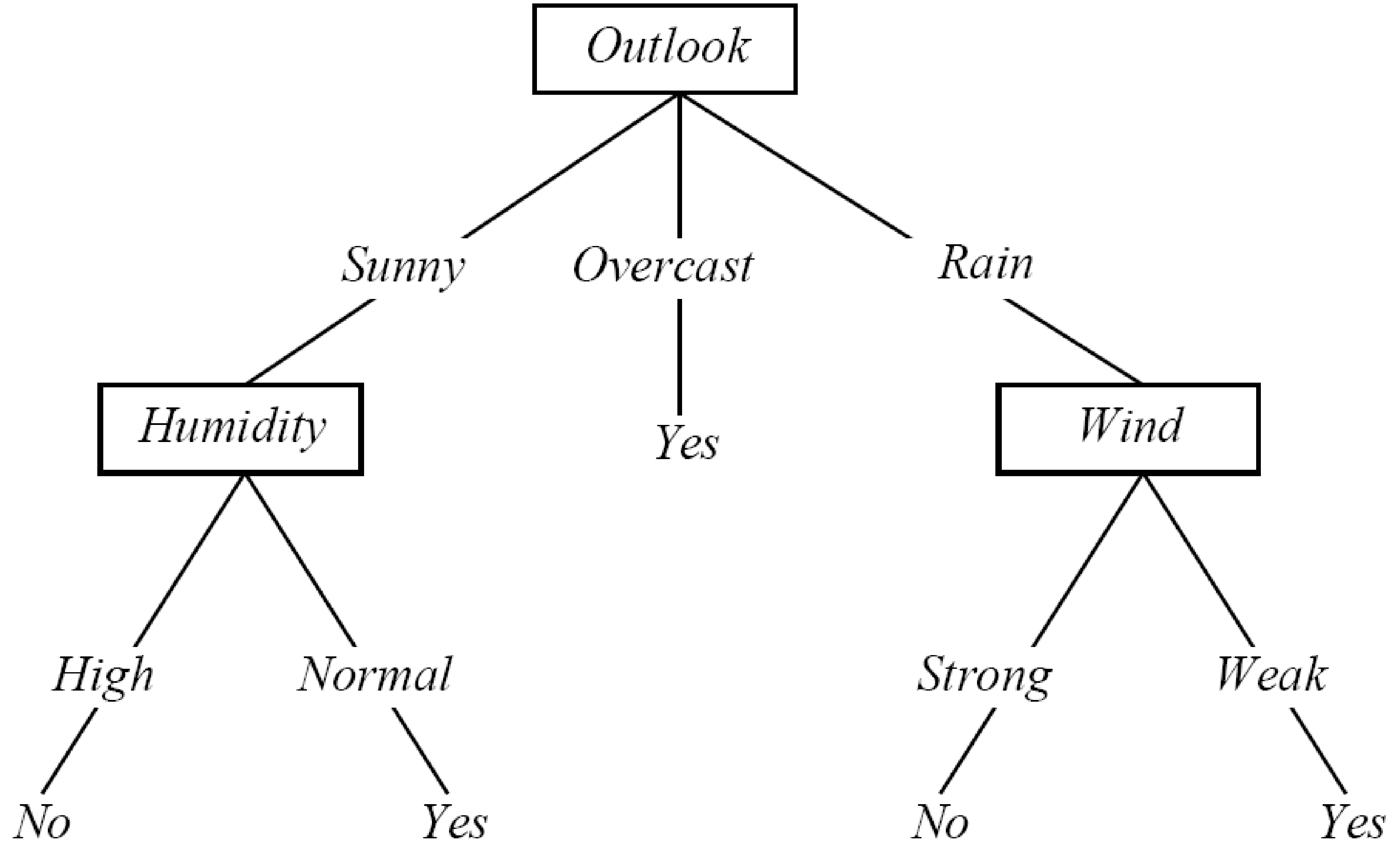
University of Houston

# Decision Trees

# When do I play tennis?

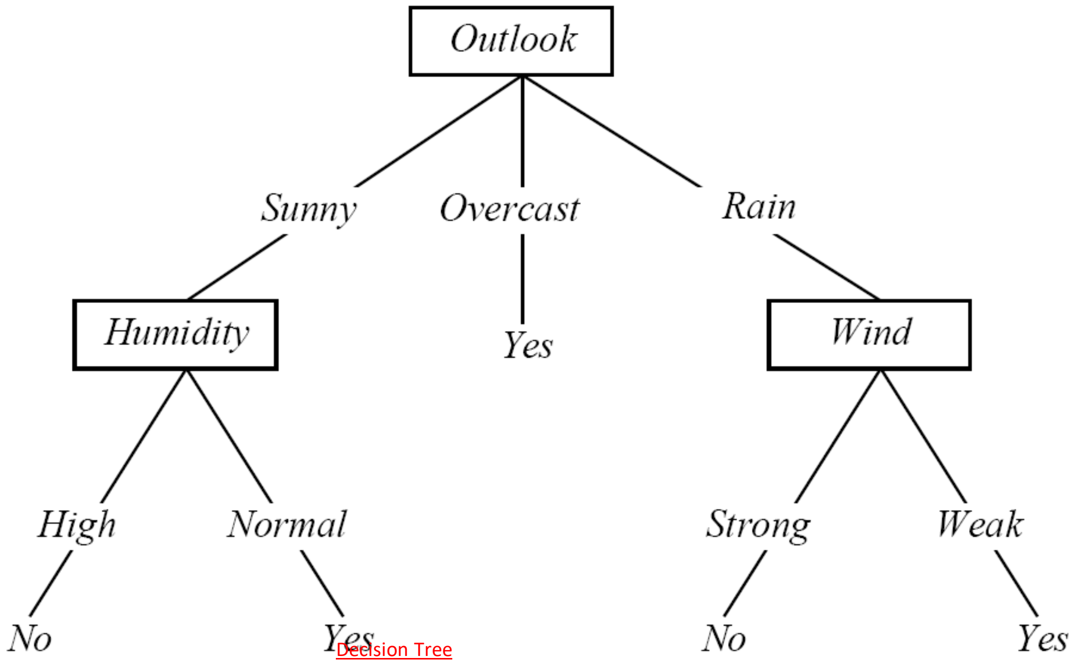
| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

# Decision Tree



# Is the decision tree correct?

- Let's check whether the split on Wind attribute is correct.
- We need to show that Wind attribute has the highest information gain.



# When do I play tennis?

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

# Wind attribute – 5 records match

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | [illegible] | [illegible] | [illegible] | [illegible] |
| D4 | Rain | [illegible] | [illegible] | [illegible] | [illegible] |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

Note: calculate the entropy only on examples that got "routed" in our branch of the tree (Outlook=Rain)

# Calculation

- Let

  $S = \{D4, D5, D6, D10, D14\}$
- Entropy: ( =-p/(p+n)log(p/(p+n) -n/(p+n)log(n/(p+n)

  $H(S) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971$
- Information Gain

  $IG(S,Temp) = H(S) - H(S|Temp) = 0.01997$

  $IG(S, Humidity) = H(S) - H(S|Humidity) = 0.01997$

  $IG(S,Wind) = H(S) - H(S|Wind) = 0.971$

# Entropy of outlook

14=9(yes)+5(No)

| Outlook | P | n | H(sunny or ..) |
|---|---|---|---|
| Sunny | 2 | 3 | 0.971 |
| Overcast | 4 | 0 | 0 |
| Rain | 3 | 2 | 0.971 |

H(Playtennis,outlook)=Σpro*H(sunny or ..)

H(Playtennis,Outlook)=(2+3)/14*0.971+(4+0)/14*0+ (3+2)/14*0.971=0.692

Entropy: ( =-p/(p+n)log(p/(p+n) -n/(p+n)log(n/(p+n)

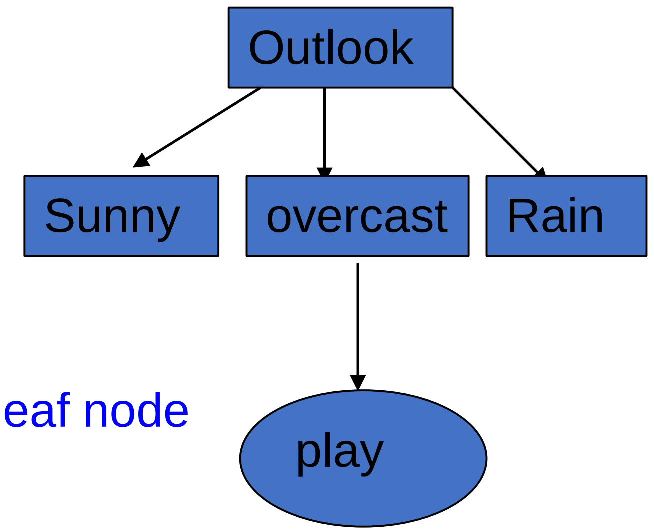
H(PlayingTennis)=-9/14*log(9/14) -5/14*log(5/14)= 0.940

Information G(outlook)= 0.940-0.692=0.248

…G(temp)=0.029 ; G(Humidity)=0.151 ;G(Wind)=0.048 Highest

# More about Decision Trees

- The root is Outlook(high IG)




Since H(overcast)=0 ➔leaf node

H(sunny)!=0 ➔Split again

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

# Split Sunny

Entropy: ( =-p/(p+n)log(p/(p+n) -n/(p+n)log(n/(p+n)

$$H(S) = -1/5\log(1/5) - 2/5\log(2/5) = 1/5\ 2.3219 + 2/5*1.3219$$
$$= 0.464 + 0.528$$

**Entropy of Windy**

| | | | |
|---|---|---|---|
| Weak | 1 | 2 | H(Weak) =0.918 |
| Strong | 1 | 1 | H(Strong) =1 |

H(Windy)=(1+2)/5 * 0.918 + (1+1)/5 * 1=0.950

Gain= 0.971-0.950=0.02

Sunny should not split on Windy

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

# Split Sunny?

Entropy: ( =-p/(p+n)log(p/(p+n) -n/(p+n)log(n/(p+n)

$$H(S) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971$$

**Entropy of Humidity**

| | | | |
|---|---|---|---|
| High | 0 | 3 | H(high)=0 |
| Normal | 2 | 0 | H(Normal) =0 |

Humidity=0

Gain= 0.971-0.0=0.971

**Sunny should be split on humidity …high and Normal**

# Python implementation of Decision Tree

## Balance Scale Data Set

This data set was generated to model psychological experimental results.

Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced.

The attributes are the left weight, the left distance, the right weight, and the right distance.

The correct way to find the class is the

greater of (left-distance * left-weight) and (right-distance * right-weight).

If they are equal, it is balanced.


Decision Tree

Inside importdata
Function add commands
to understand
the data
….info()…describe…find
missing value

```
In [2]: # Importing the required packages
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 ("Dataset Lenght: ", len(balance_data))
    print ("Dataset Shape: ", balance_data.shape)

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

```
Dataset Lenght:  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
```

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

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

    # Spliting 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
```

```python
# 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
```

clf_gini = train_using_gini(X_train, X_test, y_train)

```python
# 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
```

```
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']
```

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

```python
# 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))
```

cal_accuracy(y_test, y_pred_gini)

```
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
```

```python
# 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
```

```python
# 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
```

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

```
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']
```

```python
# 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))
```

```python
cal_accuracy(y_test, y_pred_entropy)
```

```
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
```

```python
# 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()
```

# Import libraries

- import numpy as np
- import pandas as pd
- from sklearn.cross_validation import train_test_split
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.metrics import accuracy_score
- from sklearn import tree

# Read from file

- data = pd.read_csv(
- ... 'http://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data',
- ... sep= ',', header= None)

# Print length of dataset

- `print('Dataset length:', len(name))`

- `Dataset length: 625`

# Data Slicing

- Dataset consists of 5 attributes
- 4 feature attributes and 1 target attribute
- The index of the target attribute is 1st

- `X = data.values[:,1:5]`
- `Y = data.values[:,0]`

# Split dataset between train and test

- `X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3)`

- X_train and y_train = training data
- X_test and y_test = test data

- Test_size = test set will be 30% of whole dataset and training will be 70%

# Decision Tree Training

- `clf_entropy = DecisionTreeClassifier(max_depth=3)`
- `clf_entropy.fit(X_train, y_train)`

- **Result**

```
DecisionTreeClassifier(compute_importances=None, criterion='gini',
            max_depth=3, max_features=None, max_leaf_nodes=None,
            min_density=None, min_samples_leaf=1, min_samples_split=2,
            random_state=None, splitter='best')
```

# Prediction

- `y_pred_en = clf_entropy.predict(X_test)`
- `y_pred_en`