**Assignment 9: Decision Tree**

Due Date: 14-March-2021

Student Name: Rohit Byas

Roll No. : 181210043

Agenda for Assignment 9

Understand the working of the Decision Tree

```
import datetime

try:
    from google.colab import drive
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False


print('Name: Rohit Byas \nRoll No: 181210043')
print(datetime.datetime.now())
```

```
Note: using Google CoLab
Name: Rohit Byas
Roll No: 181210043
2021-03-17 03:53:01.412888
```

```
import pandas as pd
import numpy as np
import math
import copy
import gdown
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier


!gdown --id '1W3QWf9UWQzRMn1Th4mWz4tF47mG6pewC'
```

```
Downloading...
From: https://drive.google.com/uc?id=1W3QWf9UWQzRMn1Th4mWz4tF47mG6pewC
To: /content/dataset.csv
100% 421/421 [00:00<00:00, 924kB/s]
```

```
df = pd.read_csv('dataset.csv')
df
```

| | Outlook | Temp | Humidity | Wind | Play |
|---|---|---|---|---|---|
| 0 | Sunny | Hot | High | Weak | no |
| 1 | Sunny | Hot | High | Strong | no |
| 2 | Overcast | Hot | High | Weak | yes |
| 3 | Rain | Mild | High | Weak | yes |
| 4 | Rain | Cool | Normal | Weak | yes |
| 5 | Rain | Cool | Normal | Strong | no |
| 6 | Overcast | Cool | Normal | Strong | yes |
| 7 | Sunny | Mild | High | Weak | no |
| 8 | Sunny | Cool | Normal | Weak | yes |
| 9 | Rain | Mild | Normal | Weak | yes |
| 10 | Sunny | Mild | Normal | Strong | yes |
| 11 | Overcast | Mild | High | Strong | yes |
| 12 | Overcast | Hot | Normal | Weak | yes |
| 13 | Rain | Mild | High | Strong | no |

```
df.columns
```

```
Index(['Outlook', 'Temp', 'Humidity', 'Wind', 'Play'], dtype='object')
```

```
# Input and Output Columns
attributes = ['Outlook', 'Temp', 'Humidity', 'Wind']
label = 'Play'
```

```
# Node and Edge classes for the decision tree
class Node:
   def __init__(self, value):
     self._value = value
     self._edges = []

   def __repr__(self):
     if len(self._edges):
       return f'{self._value} --> {self._edges}'
     else:
       return f'{self._value}'
```

```
    @property
    def value(self):
      return self._value

    @property
    def edges(self):
      return self._edges

    def add_edge(self, edge):
      self._edges.append(edge)

    def find_edge(self, value):
      return next(edge for edge in self._edges if edge.value == value)

# edge is supposed to have a value and it points towards the node
class Edge:
  def __init__(self, value):
    self._value = value
    self._node = None

  def __repr__(self):
    return f'{self._value} --> {self._node}'

  @property
  def value(self):
    return self._value

  @property
  def node(self):
    return self._node

  @node.setter
  def node(self, node):
    self._node = node


# Function to make a decision tree with the given impurity parameter
def build_tree(df, features, cost, min_max, label = 'Play'):
  weighted_sums = {}
  for col in features:
    weighted_sums[col] = cost(df, col, label)
  weighted_sum_vals = list(weighted_sums.values())
  if (float(0) in weighted_sum_vals and len(set(weighted_sum_vals))==1) or not weighted_sum_v
    label = df['Play'].iloc[0]
    return Node(label)

  min_feature = min_max(weighted_sums, key = weighted_sums.get)
  node = Node(min_feature)

  reduced_features = copy.deepcopy(features)
  reduced_features.remove(min_feature)
  min_feature_values = list(df[min_feature].unique())
```

```
    min_feature_values = list(df[min_feature].unique())

    for val in min_feature_values:
      edge = Edge(val)
      node.add_edge(edge)
      reduced_data_points = df[df[min_feature] == val].copy()
      edge.node = build_tree(reduced_data_points, reduced_features, cost, min_max)

    return node
```

Task 1:

Using the Gini as impurity paramter ,construct a Decision Tree

```
def gini(df, column, label):

  play = df[label].unique()
  n = df[label].shape[0]
  total_gini = 0
  u = df[col].unique()

  for element in u:

    tdf = df[df[column]==element]
    number = tdf.shape[0]
    r = 0
    for play in play:
      total = temp_df[temp_df[label] == play].shape[0]
      temp = (total/number) ** 2
      r += temp
    r = 1 - r
    temp = num/n
    total_gini += (temp * r)
  return total_gini


from sklearn.tree import DecisionTreeClassifier



t1= build_tree(df=df, features=attributes, cost=gini, min_max=min, label=label)
t1
```

```
      Outlook --> [Sunny --> Humidity --> [High --> no, Normal --> yes], Overcast --> yes, Ra
```

Task 2:

Construct a Decision Tree using Information Gain as impurity parameter

```
# to define the entropy
def comp_entropy(column):

  count = list(column.value_counts())
  p = [x/len(column) for x in count]

  entropy = 0
  for a in p:
    if a > 0:
      entropy += (a * math.log(a, 2))

  return entropy


#to compute information gain

def comp_ig(data, s_name , t_name):

  og_entropy = comp_entropy(data[t_name])

  values = data[s_name].unique()
  left_split = data[data[s_name] == values[0]]
  right_split = data[data[s_name] == values[1]]

  sub = 0
  for subset in [left_split, right_split]:

    a = (subset.shape[0] / data.shape[0])
    sub += (a * comp_entropy(subset[t_name]))
  return (og_entropy - sub)


t2 = build_tree(df=df, features=attributes, cost=comp_ig, min_max=max)
t2
```

```
Outlook --> [Sunny --> Humidity --> [High --> no, Normal --> yes], Overcast --> yes, Rai
```

## Task 3:

## To Construct a Decision Tree using Misclassification Error as impurity parameter

```
def comp_misclass_error(df, column, label):

  count = list(df[label].value_counts())

  n = df.shape[0]

  p = 1 - max([x/n for x in counts])
```

```
    sub = 0
    value = df[col].unique()


    for v in value:

      tdf = df[df[column]==v]
      a = tdf.shape[0]

      weight = a/n
      cts = list(tdf[label].value_counts())
      vp = 1 - max([x/a for x in cnts])
      sub += (weight*vp)

    return (p - sub)


t3 = build_tree(df=df, features=attrs, cost=calc_misclassification_error, min_max=max)
t3
```

```
Outlook --> [Sunny --> Humidity --> [High --> no, Normal --> yes], Overcast --> yes, Rai
```

Task 4:

Predict and compare the result of above 3 tree the value of PlayTennis for:

Outlook = Sunny, Temp = Cool, Humidity = High, Wind = Strong

```
check = {'Outlook':'Sunny', 'Temp':'Cool', 'Humidity':'High', 'Wind':'Strong'}


# To compute the outcome for the decision tree

def outcome(node, check):

  node_v = node.value
  node_e = node.edges

  for i in node_edges:

    if i.value == check[node_v]:
      return outcome(i.node, check)

  return node_v


outcome(t1, check)
```

```
'no'
```

```
outcome(t2, check)
```

```
'no'
```

```
outcome(t3, check)
```

```
'no'
```

Task 5:

Compare your result with the inbuilt library available

```
df.columns
```

```
Index(['Outlook', 'Temp', 'Humidity', 'Wind', 'Play'], dtype='object')
```

```
attrs = ['Outlook', 'Temp', 'Humidity', 'Wind']
label = 'Play'
```

```
# Encoding all columns of the dataset
labelEncoder = preprocessing.LabelEncoder()
mapping = {}
for col in df.columns:
  df[col] = labelEncoder.fit_transform(df[col])
  mapping[col] = dict(zip(labelEncoder.classes_, labelEncoder.transform(labelEncoder.classes_
```

```
 df
```

| | Outlook | Temp | Humidity | Wind | Play |
|---|---|---|---|---|---|
| **0** | 2 | 1 | 0 | 1 | 0 |
| **1** | 2 | 1 | 0 | 0 | 0 |
| **2** | 0 | 1 | 0 | 1 | 1 |
| **3** | 1 | 2 | 0 | 1 | 1 |

```
mapping
```

```
{'Humidity': {'High': 0, 'Normal': 1},
 'Outlook': {'Overcast': 0, 'Rain': 1, 'Sunny': 2},
 'Play': {'no': 0, 'yes': 1},
 'Temp': {'Cool': 0, 'Hot': 1, 'Mild': 2},
 'Wind': {'Strong': 0, 'Weak': 1}}
```

```
# Converting given conditions to integer as mapped by the encoder
to_check = {'Outlook':'Sunny', 'Temp':'Cool', 'Humidity':'High', 'Wind':'Strong'}
to_check_list = []
for key in to_check.keys():
  to_check_list.append(mapping[key][to_check[key]])
```

| | | | | | |
|---|---|---|---|---|---|
| **13** | 1 | 2 | 0 | 0 | 0 |

```
to_check_list
```

```
[2, 0, 0, 0]
```

## Decision Tree

```
# Making a Decision Tree Classifier for predicting outcome
classifier = tree.DecisionTreeClassifier()
classifier = classifier.fit(df[attrs], df[label])
```

```
pred = classifier.predict([to_check_list])
for key, val in mapping[label].items():
  if val == int(pred):
    print(key)
    break
```

```
no
```

## Random Forest

```
# Making a Random Forest Classifier for predicting outcome
```

```
clf = RandomForestClassifier(n_estimators = 100)
clf = clf.fit(df[attrs], df[label])


pred = clf.predict([to_check_list])
for key, val in mapping[label].items():
  if val == int(pred):
    print(key)
    break
```

```
no
```

**observation**

1. It is revealed that decision wil always be **Yes** if wind weak and outlook were rain
2. Similary decision will be always **NO** if wind were strong and outlook were rain.
3. When Outlook = Sunny, Temp = Cool, Humidity = High, Wind = Strong then decision will be **NO**