Decision Tree for the Use Case “Playing Tennis” using ID3 method

Homework H4.5 from Exercises to Lesson ML4Homework of the lecture "Machine Learning - Concepts & Algorithms". DHBW Stuttgart (WS2020)

By Brian Brandner and Daniel Rück; 26. October 2020

Reviewed and extended by Hermann Völlinger; 29. October 2020

The ID3 (Iterative Dichotomiser 3) method is used to generate a decision tree from a dataset. To achieve this the algorithm needs the **Entropy** formula to determine impurity of data and the **Information Gain**, which indicates the most relevant dataset attribut

Prerequisites

- **input data** - check the input file 'playTennis.csv'
- **images** - all images are located in the directory 'Images/'

Import of libraries

- **pandas** - loads the dataset and provids necessary frame details
- **math** - calculates in the alogarithm to the base 2
- **pprint** - prints the dictionary storage
- **IPython** - uses display, Math and Latex to for printing the formula
- **sys** - version information to pythonImport of libraries

In [1]:

```
# Libraries to import
import pandas as pd
import math
import pprint
from IPython.display import display, Math, Latex
# python version check library
import sys

# to check the time of execution, import function time
import time

# print python version, for some imports this version number is viewed as theirs.
print("python {}".format(sys.version))

# version of pandas
print("pandas {}".format(pd.__version__))
```

```
python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 b
it (AMD64)]
pandas 1.0.3
```

Entropy formula

H - greek Eta, Entropy

S - Dataset

p(x_i) - Proportion of classification to results (Quantity of Yes or No / sum of Yes and No)

In [2]:

```
display(Math(r'H(S) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)'))
```

$$H(S) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

Information Gain formula

IG - Information Gain

S - Dataset

C - Column

H(S_Total) - Total entropy of the dataframe

p(Z_Column) - Value count of active column divided by max column length

H(S_Column) - Entropy of active column value

In [3]:

```
display(Math(r'IG(S, C) = H(S_{Total}) - \sum_{} p(Z_{Column}) * H(S_{Column})'))
```

$$IG(S, C) = H(S_{Total}) - \sum_{} p(Z_{Column}) * H(S_{Column})$$

In [4]:

```
df = pd.read_csv("playTennis.csv", sep=';')
df
```

Out[4]:

	Outlook	Temperature	Humidity	Windy	Play
0	Sunny	Hot	High	F	No
1	Sunny	Hot	High	T	No
2	Overcast	Hot	High	F	Yes
3	Rainy	Mild	High	F	Yes
4	Rainy	Cool	Normal	F	Yes
5	Rainy	Cool	Normal	T	No
6	Overcast	Cool	Normal	T	Yes
7	Sunny	Mild	High	F	No
8	Sunny	Cool	Normal	F	Yes
9	Rainy	Mild	Normal	F	Yes
10	Sunny	Mild	Normal	T	Yes
11	Overcast	Mild	High	T	Yes
12	Overcast	Hot	Normal	F	Yes
13	Rainy	Mild	High	T	No

Root-Node

Total Entropy of the whole dataframe

- **tempStorage** - variable to store selected informations
- **df** - loaded dataframe object, function groupby(['Column']) selects the passed column, function size() of it counts column variables

In [5]:

```
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Play']).size()
tempStorage
```

Out[5]:

```
Play
No      5
Yes     9
dtype: int64
```

- **yesCount** - variable holds column value "Yes" count
- **noCount** - variable holds column value "No" count
- **aggreg** - Elements in Column
- **aggregTotal** - Elements in Column for Information Gain calculation
- **entropyTotal** - calculated entropy

In [6]:

```
# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregTotal = aggreg

entropyTotal = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))


print("H(S) = {0:.5f}\n".format(entropyTotal))
```

```
H(S) = 0.94029
```

Entropy of column Outlook

This time a selection of two columns is necessary, first column combined with result column (in this example: 'Play') size() provides again the amount of value frequency

In [7]:

```
# Overview of two selected columns
df.groupby(['Outlook', 'Play']).size()
```

Out[7]:

```
Outlook   Play
Overcast  Yes     4
Rainy     No      2
          Yes     3
Sunny     No      3
          Yes     2
dtype: int64
```

To access the data, a call for the column value is needed.

For 'Overcast' only "Yes" is an option, hence noCount is set to 0 and the formular misses the addition of its calculation

In [8]:

```
# fracture dataframe to export compute data
tempStorage = df.groupby(['Outlook', 'Play']).size()['Overcast']

# write values to variables
yesCount = tempStorage['Yes']
noCount = 0
aggreg = yesCount + noCount
aggregOvercast = aggreg

entropyOvercast = -((yesCount/aggreg)*math.log2(yesCount/aggreg))
```

For "Sunny" and "Rainy" both result classifications are available

In [9]:

```
# fracture dataframe to export compute data
tempStorage = df.groupby(['Outlook', 'Play']).size()['Sunny']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregSunny = aggreg

entropySunny = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))

# fracture dataframe to export compute data
tempStorage = df.groupby(['Outlook', 'Play']).size()['Rainy']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregRainy = aggreg

entropyRainy = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))
```

Print out the calculated Entropy values

In [10]:

```
print("H(Outlook == Overcast) = {0:.5f}".format(entropyOvercast))
print("H(Outlook == Sunny) = {0:.5f}".format(entropySunny))
print("H(Outlook == Rainy) = {0:.5f}".format(entropyRainy))
```

```
H(Outlook == Overcast) = -0.00000
H(Outlook == Sunny) = 0.97095
H(Outlook == Rainy) = 0.97095
```

Information Gain for column Outlook

Finally the Information Gain analyse for the selected column 'Outlook'

In [11]:

```
# Entropy offset
offsetEntropy = (aggregOvercast/aggregTotal)*entropyOvercast + (aggregSunny/aggregTotal
)*entropySunny + (aggregRainy/aggregTotal)*entropyRainy
# Information Gain
infoGain = entropyTotal - offsetEntropy

# Save IG to dictionary
infoGainDict = dict()
infoGainDict['Root'] = dict()
infoGainDict['Root']['Outlook'] = infoGain
```

Print out the calculated Information Gain value

In [12]:

```
print("IG(S, outlook) = {0:.5f}\n".format(infoGain))
```

```
IG(S, outlook) = 0.24675
```

Entropy of column Temperature

Repeat procedure for each column

In [13]:

```
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Temperature', 'Play']).size()['Hot']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregHot = aggreg

entropyHot = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math.
log2(noCount/aggreg)))

##############################################
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Temperature', 'Play']).size()['Mild']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregMild = aggreg

entropyMild = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Temperature', 'Play']).size()['Cool']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregCool = aggreg

entropyCool = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

print("H(Temperature == Hot) = {0:.5f}".format(entropyHot))
print("H(Temperature == Mild) = {0:.5f}".format(entropyMild))
print("H(Temperature == Cool) = {0:.5f}".format(entropyCool))
```

```
H(Temperature == Hot) = 1.00000
H(Temperature == Mild) = 0.91830
H(Temperature == Cool) = 0.81128
```

Information Gain for column Temperature

In [14]:

```
offsetEntropy = (aggregHot/aggregTotal)*entropyHot + (aggregMild/aggregTotal)*entropyMi
ld + (aggregCool/aggregTotal)*entropyCool
infoGain = entropyTotal - offsetEntropy

infoGainDict['Root']['Temperature'] = infoGain

print("IG(S, Temperature) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Temperature) = 0.02922
```

Entropy of column Humidity

In [15]:

```
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Humidity', 'Play']).size()['High']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregHigh = aggreg

entropyHigh = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Humidity', 'Play']).size()['Normal']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregNormal = aggreg

entropyNormal = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*ma
th.log2(noCount/aggreg)))

print("H(Humidity == High) = {0:.5f}".format(entropyHigh))
print("H(Humidity == Normal) = {0:.5f}".format(entropyNormal))
```

```
H(Humidity == High) = 0.98523
H(Humidity == Normal) = 0.59167
```

Information Gain for column Humidity

In [16]:

```
offsetEntropy = (aggregHigh/aggregTotal)*entropyHigh + (aggregNormal/aggregTotal)*entro
pyNormal
infoGain = entropyTotal - offsetEntropy

infoGainDict['Root']['Humidity'] = infoGain

print("IG(S, Humidity) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Humidity) = 0.15184
```

Entropy of column Windy

In [17]:

```
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Windy', 'Play']).size()['T']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregTrue = aggreg

entropyTrue = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Windy', 'Play']).size()['F']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregFalse = aggreg

entropyFalse = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))

print("H(Windy == True) = {0:.5f}".format(entropyTrue))
print("H(Windy == False) = {0:.5f}".format(entropyFalse))
```

```
H(Windy == True) = 1.00000
H(Windy == False) = 0.81128
```

Information Gain for column Windy

In [18]:

```
offsetEntropy = (aggregTrue/aggregTotal)*entropyTrue + (aggregFalse/aggregTotal)*entrop
yFalse
infoGain = entropyTotal - offsetEntropy

infoGainDict['Root']['Windy'] = infoGain

print("IG(S, Windy) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Windy) = 0.04813
```

Overview Information Gain for the Root-Node

Select Root-Node based on highest Information Gain value

Pretty print out the collected Information Gain vaules with column name.

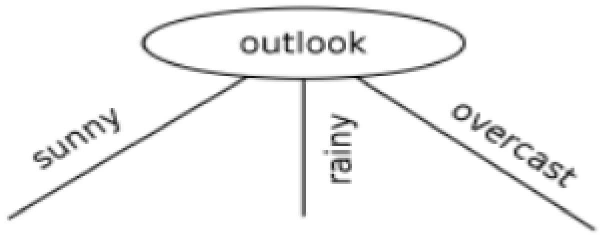
Reset of the dictionary with selected node as root.

In [19]:

```
pprint.pprint(infoGainDict)
print("\nSelected '{0}' because of value {1:.5f}".format("Outlook", infoGainDict['Root'
]['Outlook']))

infoGainDict.clear()
infoGainDict["Outlook"] = dict()
print(infoGainDict)

print("Current status of decision tree after root node is defined")

from IPython.display import Image
Image('Images/H4_5_01-root.png')
```

```
{'Root': {'Humidity': 0.15183550136234159,
          'Outlook': 0.24674981977443933,
          'Temperature': 0.02922256565895487,
          'Windy': 0.04812703040826949}}

Selected 'Outlook' because of value 0.24675
{'Outlook': {}}
Current status of decision tree after root node is defined
```

Out[19]:



Current status of decision tree after root node is defined

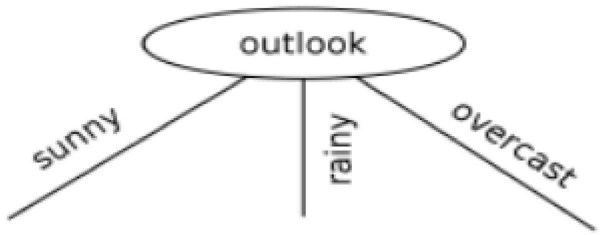
Decision tree as image

In [20]:

```
print("********** Decision tree as image ************")
from IPython.display import Image
Image('Images/H4_5_01-root.png')
```

```
********** Decision tree as image ************
```

Out[20]:



Branch Sunny

- **dfS** - modified dataframe object to filter for specific string

In [21]:

```
dfS = df.loc[df['Outlook'] == 'Sunny']
dfS
```

Out[21]:

	Outlook	Temperature	Humidity	Windy	Play
0	Sunny	Hot	High	F	No
1	Sunny	Hot	High	T	No
7	Sunny	Mild	High	F	No
8	Sunny	Cool	Normal	F	Yes
10	Sunny	Mild	Normal	T	Yes

Total entropy of branch Sunny dataframe

In [22]:

```
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Play']).size()

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregTotal = aggreg

entropyTotal = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))

print("H(S) = {0:.5f}\n".format(entropyTotal))
```

```
H(S) = 0.97095
```

Entropy of column Temperature for branch Sunny

In [23]:

```
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Temperature', 'Play']).size()['Hot']

# write values to variables
yesCount = 0
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregHot = aggreg

entropyHot = -((noCount/aggreg)*math.log2(noCount/aggreg))

##############################################
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Temperature', 'Play']).size()['Mild']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregMild = aggreg

entropyMild = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Temperature', 'Play']).size()['Cool']

# write values to variables
yesCount = tempStorage['Yes']
noCount = 0
aggreg = yesCount + noCount
aggregCool = aggreg

entropyCool = -((yesCount/aggreg)*math.log2(yesCount/aggreg))

print("H(Temperature == Hot) = {0:.5f}".format(entropyHot))
print("H(Temperature == Mild) = {0:.5f}".format(entropyMild))
print("H(Temperature == Cool) = {0:.5f}".format(entropyCool))
```

```
H(Temperature == Hot) = -0.00000
H(Temperature == Mild) = 1.00000
H(Temperature == Cool) = -0.00000
```

Information Gain for column Temperature

In [24]:

```
offsetEntropy = (aggregHot/aggregTotal)*entropyHot + (aggregMild/aggregTotal)*entropyMi
ld + (aggregCool/aggregTotal)*entropyCool
infoGain = entropyTotal - offsetEntropy

infoGainDict["Outlook"]['BranchSunny'] = dict()
infoGainDict['Outlook']['BranchSunny']['Temperature'] = infoGain

print("IG(S, Temperature) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Temperature) = 0.57095
```

Entropy of column Humidity for branch Sunny

In [25]:

```
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Humidity', 'Play']).size()['High']

# write values to variables
yesCount = 0
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregHigh = aggreg

entropyHigh = -((noCount/aggreg)*math.log2(noCount/aggreg))

##############################################
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Humidity', 'Play']).size()['Normal']

# write values to variables
yesCount = tempStorage['Yes']
noCount = 0
aggreg = yesCount + noCount
aggregNormal = aggreg

entropyNormal = -((yesCount/aggreg)*math.log2(yesCount/aggreg))

print("H(Humidity == High) = {0:.5f}".format(entropyHigh))
print("H(Humidity == Normal) = {0:.5f}".format(entropyNormal))
```

```
H(Humidity == High) = -0.00000
H(Humidity == Normal) = -0.00000
```

Information Gain for column Humidity

In [26]:

```
offsetEntropy = (aggregHigh/aggregTotal)*entropyHigh + (aggregNormal/aggregTotal)*entro
pyNormal
infoGain = entropyTotal - offsetEntropy

infoGainDict['Outlook']['BranchSunny']['Humidity'] = infoGain

print("IG(S, Humidity) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Humidity) = 0.97095
```

Entropy of column Windy for branch Sunny

In [27]:

```
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Windy', 'Play']).size()['T']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregTrue = aggreg

entropyTrue = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export compute data
tempStorage = dfS.groupby(['Windy', 'Play']).size()['F']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregFalse = aggreg

entropyFalse = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))

print("H(Windy == True) = {0:.5f}".format(entropyTrue))
print("H(Windy == False) = {0:.5f}".format(entropyFalse))
```

```
H(Windy == True) = 1.00000
H(Windy == False) = 0.91830
```

Information Gain for column Windy

In [28]:

```
offsetEntropy = (aggregTrue/aggregTotal)*entropyTrue + (aggregFalse/aggregTotal)*entrop
yFalse
infoGain = entropyTotal - offsetEntropy

infoGainDict['Outlook']['BranchSunny']['Windy'] = infoGain

print("IG(S, Windy) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Windy) = 0.01997
```

Overview Information Gain Branch Sunny

Select Node based on highest Information Gain value

In [29]:

```
pprint.pprint(infoGainDict)

print("\nSelected '{0}' because of value {1:.5f}".format("Humidity", infoGainDict['Outl
ook']['BranchSunny']['Humidity']))

infoGainDict['Outlook']['BranchSunny'].clear()
infoGainDict['Outlook']['BranchSunny']['Humidity'] = dict()
pprint.pprint(infoGainDict)
```

```
{'Outlook': {'BranchSunny': {'Humidity': 0.9709505944546686,
                             'Temperature': 0.5709505944546686,
                             'Windy': 0.01997309402197489}}}

Selected 'Humidity' because of value 0.97095
{'Outlook': {'BranchSunny': {'Humidity': {}}}}
```

Current status of decision tree after first inner node is defined

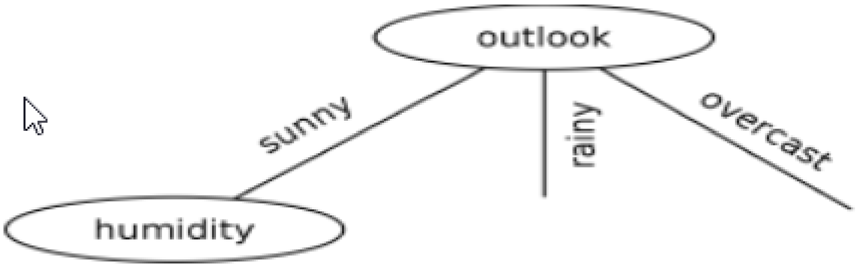
Decision tree as image

In [30]:

```
print("*************** Decision tree as image *****************")
from IPython.display import Image
Image('Images/H4_5_02-firstInnerNode.png')
```

```
*************** Decision tree as image *****************
```

Out[30]:



Subnode Humidity of Branch Sunny

High value

In [31]:

```
dfSH = dfS.loc[dfS['Humidity'] == 'High']
dfSH
```

Out[31]:

	Outlook	Temperature	Humidity	Windy	Play
0	Sunny	Hot	High	F	No
1	Sunny	Hot	High	T	No
7	Sunny	Mild	High	F	No

All results of the dataframe are "No", hence no calculation of entropy necessary

In [32]:

```
infoGainDict["Outlook"]['BranchSunny']['Humidity']['High'] = "No"
```

Current status of decision tree with value of first leaf

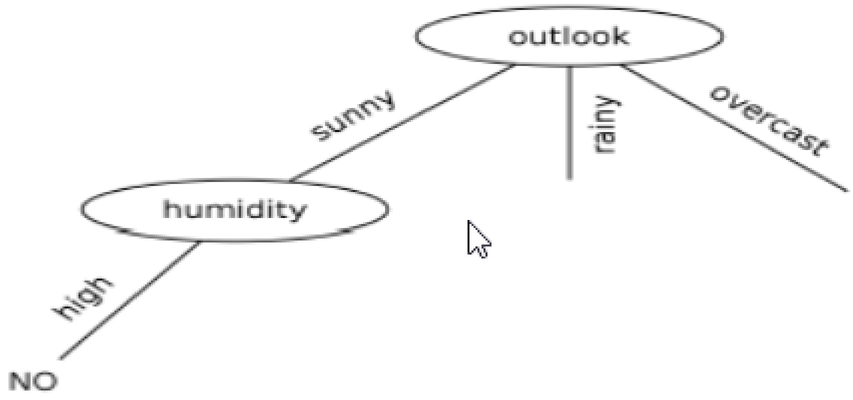
Decision tree as image with Leaf = {No}

In [33]:

```
print("*************** Decision tree as image *****************")
from IPython.display import Image
Image('Images/H4_5_03-fIN1.png')
```

```
*************** Decision tree as image *****************
```

Out[33]:



Normal value

In [34]:

```
dfSH = dfS.loc[dfS['Humidity'] == 'Normal']
dfSH
```

Out[34]:

	Outlook	Temperature	Humidity	Windy	Play
8	Sunny	Cool	Normal	F	Yes
10	Sunny	Mild	Normal	T	Yes

All results of the dataframe are "Yes", hence no calculation of entropy necessary

In [35]:

```
infoGainDict["Outlook"]['BranchSunny']['Humidity']['Normal'] = "Yes"
```

Current status of decision tree with value of second leaf

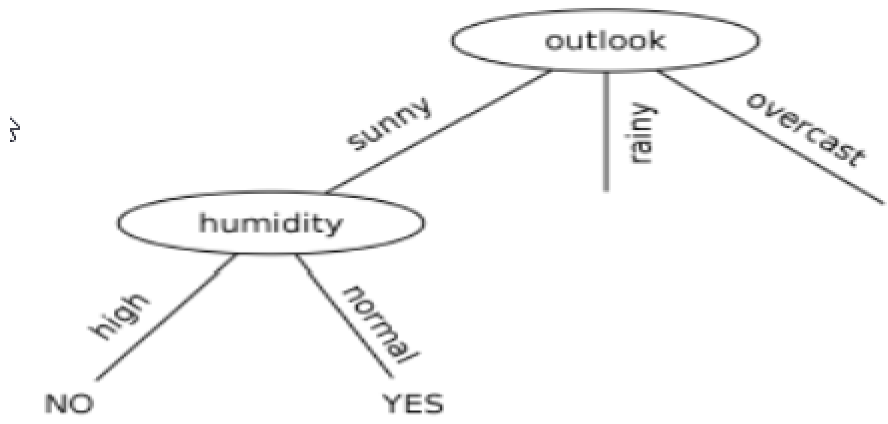
Decision tree as image with Leaf = {Yes}

In [36]:

```
print("*************** Decision tree as image *****************")
from IPython.display import Image
Image('Images/H4_5_04-fIN2.png')
```

```
*************** Decision tree as image *****************
```

Out[36]:



Branch Overcast

In [37]:

```
dfO = df.loc[df['Outlook'] == 'Overcast']
dfO
```

Out[37]:

	Outlook	Temperature	Humidity	Windy	Play
2	Overcast	Hot	High	F	Yes
6	Overcast	Cool	Normal	T	Yes
11	Overcast	Mild	High	T	Yes
12	Overcast	Hot	Normal	F	Yes

All results of the dataframe are "Yes", hence no calculation of entropy necessary

In [38]:

```
infoGainDict["Outlook"]['BranchOvercast'] = "Yes"
```

Current status of decision tree with value branch 'overcast'

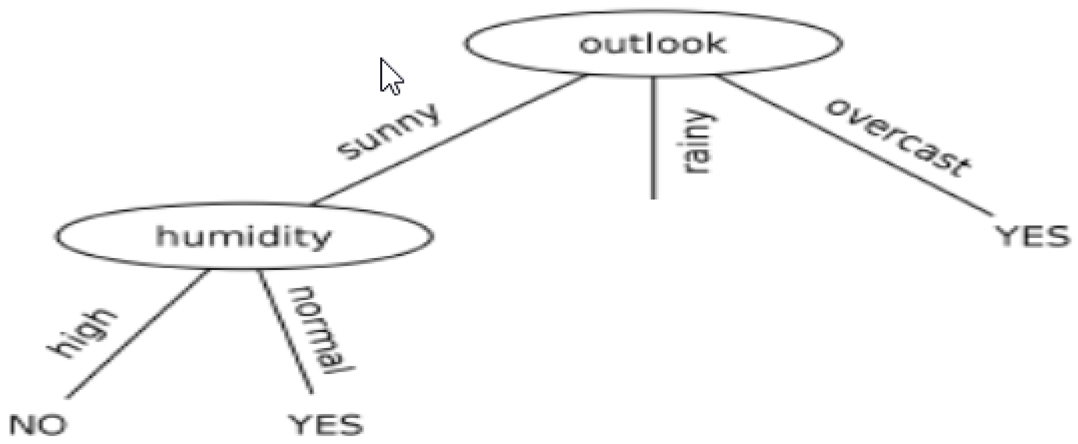
Decision tree as image with Leaf = {Yes} for branch 'overcast'

In [39]:

```
print("*************** Decision tree as image *****************")
from IPython.display import Image
Image('Images/H4_5_05_secondInnerNode.png')
```

```
*************** Decision tree as image *****************
```

Out[39]:



Branch Rainy

In [40]:

```
dfR = df.loc[df['Outlook'] == 'Rainy']
dfR
```

Out[40]:

	Outlook	Temperature	Humidity	Windy	Play
3	Rainy	Mild	High	F	Yes
4	Rainy	Cool	Normal	F	Yes
5	Rainy	Cool	Normal	T	No
9	Rainy	Mild	Normal	F	Yes
13	Rainy	Mild	High	T	No

Total entropy of branch Rainy dataframe

In [41]:

```
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Play']).size()

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregTotal = aggreg

entropyTotal = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*mat
h.log2(noCount/aggreg)))

print("H(S) = {0:.5f}\n".format(entropyTotal))
```

```
H(S) = 0.97095
```

Entropy of column Temperature for branch Rainy

In [42]:

```
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Temperature', 'Play']).size()['Mild']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregMild = aggreg

entropyMild = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Temperature', 'Play']).size()['Cool']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregCool = aggreg

entropyCool = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

print("H(Temperature == Mild) = {0:.5f}".format(entropyMild))
print("H(Temperature == Cool) = {0:.5f}".format(entropyCool))
```

```
H(Temperature == Mild) = 0.91830
H(Temperature == Cool) = 1.00000
```

Information Gain for column Temperature

In [43]:

```
offsetEntropy = (aggregMild/aggregTotal)*entropyMild + (aggregCool/aggregTotal)*entropy
Cool
infoGain = entropyTotal - offsetEntropy

infoGainDict['Outlook']['BranchRainy'] = dict()
infoGainDict['Outlook']['BranchRainy']['Temperature'] = infoGain

print("IG(S, Temperature) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Temperature) = 0.01997
```

Entropy of column Humidity for branch Rainy

In [44]:

```
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Humidity', 'Play']).size()['High']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregHigh = aggreg

entropyHigh = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*math
.log2(noCount/aggreg)))

##############################################
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Humidity', 'Play']).size()['Normal']

# write values to variables
yesCount = tempStorage['Yes']
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregNormal = aggreg

entropyNormal = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/aggreg)*ma
th.log2(noCount/aggreg)))

print("H(Humidity == High) = {0:.5f}".format(entropyHigh))
print("H(Humidity == Normal) = {0:.5f}".format(entropyNormal))
```

```
H(Humidity == High) = 1.00000
H(Humidity == Normal) = 0.91830
```

Information Gain for column Humidity

In [45]:

```
offsetEntropy = (aggregHigh/aggregTotal)*entropyHigh + (aggregNormal/aggregTotal)*entro
pyNormal
infoGain = entropyTotal - offsetEntropy

infoGainDict['Outlook']['BranchRainy']['Humidity'] = infoGain

print("IG(S, Humidity) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Humidity) = 0.01997
```

Entropy of column Windy for branch Rainy

In [46]:

```
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Windy', 'Play']).size()['T']

# write values to variables
yesCount = 0
noCount = tempStorage['No']
aggreg = yesCount + noCount
aggregTrue = aggreg

entropyTrue = -((noCount/aggreg)*math.log2(noCount/aggreg))

##############################################
# fracture dataframe to export compute data
tempStorage = dfR.groupby(['Windy', 'Play']).size()['F']

# write values to variables
yesCount = tempStorage['Yes']
noCount = 0
aggreg = yesCount + noCount
aggregFalse = aggreg

entropyFalse = -((yesCount/aggreg)*math.log2(yesCount/aggreg))

print("H(Windy == True) = {0:.5f}".format(entropyTrue))
print("H(Windy == False) = {0:.5f}".format(entropyFalse))
```

```
H(Windy == True) = -0.00000
H(Windy == False) = -0.00000
```

Information Gain for column Windy

In [47]:

```
offsetEntropy = (aggregTrue/aggregTotal)*entropyTrue + (aggregFalse/aggregTotal)*entrop
yFalse
infoGain = entropyTotal - offsetEntropy

infoGainDict['Outlook']['BranchRainy']['Windy'] = infoGain

print("IG(S, Windy) = {0:.5f}\n".format(infoGain))
```

```
IG(S, Windy) = 0.97095
```

Overview Information Gain Branch Rainy

Select Node based on highest Information Gain value

In [48]:

```
pprint.pprint(infoGainDict)
print("\nSelected '{0}' because of value {1:.5f}".format("Windy", infoGainDict['Outloo
k']['BranchRainy']['Windy']))

infoGainDict['Outlook']['BranchRainy'].clear()
infoGainDict['Outlook']['BranchRainy']['Windy'] = dict()
pprint.pprint(infoGainDict)
```

```
{'Outlook': {'BranchOvercast': 'Yes',
             'BranchRainy': {'Humidity': 0.01997309402197489,
                             'Temperature': 0.01997309402197489,
                             'Windy': 0.9709505944546686},
             'BranchSunny': {'Humidity': {'High': 'No', 'Normal': 'Ye
s'}}}}

Selected 'Windy' because of value 0.97095
{'Outlook': {'BranchOvercast': 'Yes',
             'BranchRainy': {'Windy': {}},
             'BranchSunny': {'Humidity': {'High': 'No', 'Normal': 'Ye
s'}}}}
```

Subnode Windy of Branch Rainy

High value

In [49]:

```
dfRW = dfR.loc[dfR['Windy'] == 'T']
dfRW
```

Out[49]:

	Outlook	Temperature	Humidity	Windy	Play
5	Rainy	Cool	Normal	T	No
13	Rainy	Mild	High	T	No

All results of the dataframe are "No", hence no calculation of entropy necessary

In [50]:

```
infoGainDict["Outlook"]['BranchRainy']['Windy']['T'] = "No"
```

Current status of decision tree with node 'windy" and first leaf

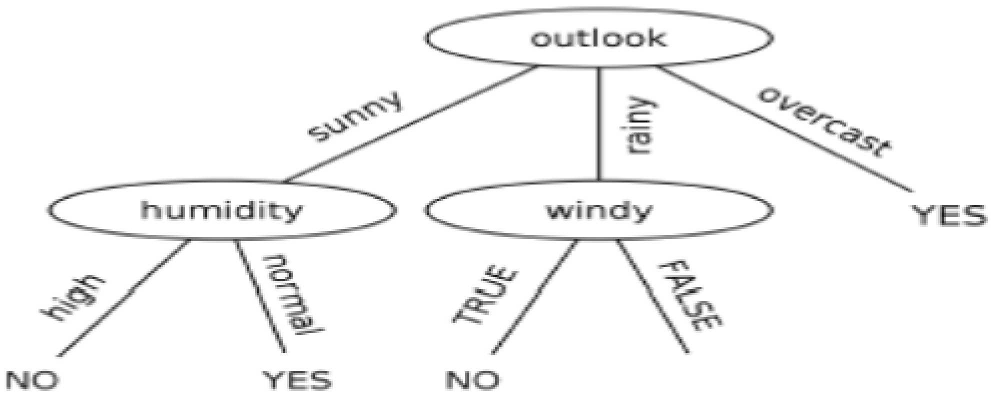
Decision tree as image with innner node 'windy'and first Leaf = {No}

In [51]:

```
print("*************** Decision tree as image ******************")
from IPython.display import Image
Image('Images/H4_5_06-tIN1.png')
```

```
*************** Decision tree as image ******************
```

Out[51]:



Normal value

In [52]:

```
dfRW = dfR.loc[dfR['Windy'] == 'F']
dfRW
```

Out[52]:

	Outlook	Temperature	Humidity	Windy	Play
3	Rainy	Mild	High	F	Yes
4	Rainy	Cool	Normal	F	Yes
9	Rainy	Mild	Normal	F	Yes

All results of the dataframe are "Yes", hence no calculation of entropy necessary

In [53]:

```
infoGainDict["Outlook"]['BranchRainy']['Windy']['F'] = "Yes"
```

Leaf end - Yes

Decision tree

In [54]:

```
#Final pprint of the dicitionary
pprint.pprint(infoGainDict)
```

```
{'Outlook': {'BranchOvercast': 'Yes',
             'BranchRainy': {'Windy': {'F': 'Yes', 'T': 'No'}},
             'BranchSunny': {'Humidity': {'High': 'No', 'Normal': 'Ye
s'}}}}
```

Final decision tree with all nodes and leafs

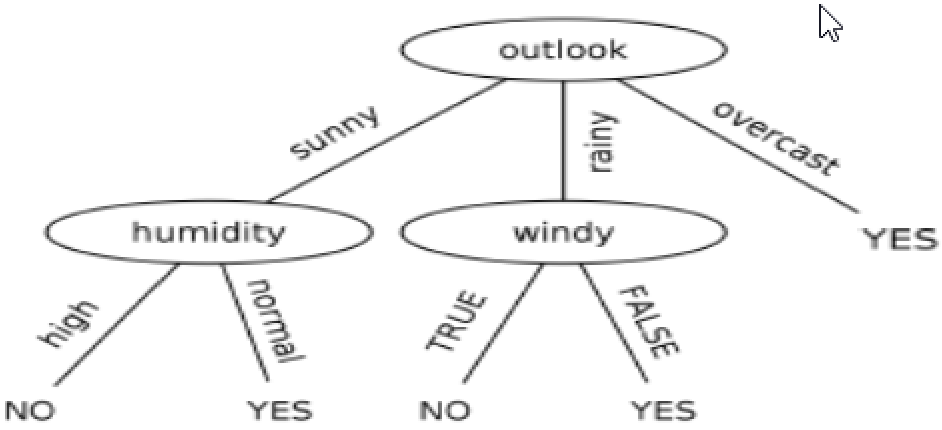
Decision tree as image with all nodes and leafs

In [55]:

```
print("*************** Final Decision tree as image ******************")
from IPython.display import Image
Image('Images/H4_5_07-tIN2.png')
```

```
*************** Final Decision tree as image ******************
```

Out[55]:



Automation try

reads dataframe and selects root node

In [56]:

```
df = pd.read_csv("playTennis.csv", sep=';')
df
```

Out[56]:

	Outlook	Temperature	Humidity	Windy	Play
0	Sunny	Hot	High	F	No
1	Sunny	Hot	High	T	No
2	Overcast	Hot	High	F	Yes
3	Rainy	Mild	High	F	Yes
4	Rainy	Cool	Normal	F	Yes
5	Rainy	Cool	Normal	T	No
6	Overcast	Cool	Normal	T	Yes
7	Sunny	Mild	High	F	No
8	Sunny	Cool	Normal	F	Yes
9	Rainy	Mild	Normal	F	Yes
10	Sunny	Mild	Normal	T	Yes
11	Overcast	Mild	High	T	Yes
12	Overcast	Hot	Normal	F	Yes
13	Rainy	Mild	High	T	No

In [57]:

```
def getEntropy(yesCount, noCount, sumYesNo):
    if (yesCount == 0):
        return -((noCount/sumYesNo)*math.log2(noCount/sumYesNo))
    elif (noCount == 0):
        return -((yesCount/sumYesNo)*math.log2(yesCount/sumYesNo))
    else:
        return -(((yesCount/sumYesNo)*math.log2(yesCount/sumYesNo)) + ((noCount/sumYesN
o)*math.log2(noCount/sumYesNo)))

def getInformationGain(totalDict, selectedDict):
    totalEntropy = totalDict['Entropy']
    totalAggCount = totalDict['AggCount']

    sumSelected = 0
    for f in selectedDict:
        selectedAggCount = selectedDict[f]['AggCount']
        selectedEntropy = selectedDict[f]['Entropy']
        sumSelected += ((selectedAggCount/totalAggCount)*selectedEntropy)

    return totalEntropy - sumSelected
    #return totalEntropy - ((mO/totalAggCount)*outcastEntropy + (mS/totalAggCount)*sunn
yEntropy + (mR/mD)*rainyEntropy)

def runThrough(dataFrame, node, blackList):
    colList = dataFrame.columns
    resCol = 'Play'
    resItems = pd.Series(dataFrame[resCol]).unique()

    autdic = {}
    autdic[node] = {}
    #################### Total Entropy
    aggCount = dataFrame.groupby([resCol]).size()[1] + dataFrame.groupby([resCol]).size
()[0]
    totalEntropy = getEntropy(dataFrame.groupby([resCol]).size()[1], dataFrame.groupby
([resCol]).size()[0], aggCount)

    autdic[node]['total'] = {}
    autdic[node]['total']['AggCount'] = aggCount
    autdic[node]['total']['Entropy'] = totalEntropy
    #################### Entropy each Col
    for i in colList:
        if (i != resCol and i not in blackList):
            autdic[node][i] = {}
            uniqueItems = pd.Series(dataFrame[i]).unique()
            for n in uniqueItems:
                autdic[node][i][n] = {}
                try:
                  yesCount = dataFrame.groupby([i, resCol]).size()[n]['Yes']
                except:
                  yesCount = 0
                try:
                  noCount = dataFrame.groupby([i, resCol]).size()[n]['No']
                except:
                  noCount = 0

                autdic[node][i][n]['AggCount'] = yesCount+noCount
                autdic[node][i][n]['Entropy'] = getEntropy(yesCount,noCount,yesCount+no
Count)
```

```
    return autdic


### "Main"

blackList = {}
rootDictEntropy = runThrough(df, "root", "")
rootDictInformationGain = {}


for i in rootDictEntropy['root']:
    if i != "total":
        rootDictInformationGain[i] = getInformationGain(rootDictEntropy['root']['total'
], rootDictEntropy['root'][i])


print("Entropies for total and subsets of Root-Node:")
pprint.pprint(rootDictEntropy)
print("\nInformation Gains for Root-Node:")
pprint.pprint(rootDictInformationGain)

import operator
selectRootNode = max(rootDictInformationGain.items(), key=operator.itemgetter(1))[0]

print("\nSelected '{0}' because of value {1:.5f}".format(selectRootNode, rootDictInform
ationGain[selectRootNode]))


blackList[0] = selectRootNode
```

```
Entropies for total and subsets of Root-Node:
{'root': {'Humidity': {'High': {'AggCount': 7, 'Entropy': 0.98522813603425
16},
                       'Normal': {'AggCount': 7,
                                  'Entropy': 0.5916727785823275}},
          'Outlook': {'Overcast': {'AggCount': 4, 'Entropy': -0.0},
                      'Rainy': {'AggCount': 5, 'Entropy': 0.97095059445466
86},
                      'Sunny': {'AggCount': 5, 'Entropy': 0.97095059445466
86}},
          'Temperature': {'Cool': {'AggCount': 4,
                                   'Entropy': 0.8112781244591328},
                          'Hot': {'AggCount': 4, 'Entropy': 1.0},
                          'Mild': {'AggCount': 6,
                                   'Entropy': 0.9182958340544896}},
          'Windy': {'F': {'AggCount': 8, 'Entropy': 0.8112781244591328},
                    'T': {'AggCount': 6, 'Entropy': 1.0}},
          'total': {'AggCount': 14, 'Entropy': 0.9402859586706311}}}

Information Gains for Root-Node:
{'Humidity': 0.15183550136234159,
 'Outlook': 0.24674981977443933,
 'Temperature': 0.02922256565895487,
 'Windy': 0.04812703040826949}

Selected 'Outlook' because of value 0.24675
```

In [58]:

```
# print current date and time
print("date",time.strftime("%d.%m.%Y %H:%M:%S"))
print ("*** End of Homework-H4.5_DecTree_ID3 ***")
```

```
date 30.10.2020 22:01:35
*** End of Homework-H4.5_DecTree_ID3 ***
```