H4_5

October 26, 2020

1 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***

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

1.1 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 python

[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

# 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 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0]
pandas 1.1.3
```

1.2 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)

[2]:
```
display(Math(r'H(S) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)'))
```

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

1.3 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

[3]:
```
display(Math(r'IG(S, C) = H(S_{Total}) - \sum_{} p(Z_{Column}) *␣
↪H(S_{Column})'))
```

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

1.4 Load dataset and view dataframe

- **df** - object storage variable
- **pd** - pandas call of function "read_csv" with parameters FILE.csv and separator semicolon

[4]:
```
df = pd.read_csv("playTennis.csv", sep=';')
df
```

[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
```

1.5 Root-Node

1.5.1 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

[5]:
```
# fracture dataframe to export data to compute
tempStorage = df.groupby(['Play']).size()
tempStorage
```

[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

[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)*math.log2(noCount/aggreg)))
```

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

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

1.5.2 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

[7]:
```
# Overview of two selected columns
df.groupby(['Outlook', 'Play']).size()
```

[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

[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

[9]:
```
# fracture dataframe to export compute data
tempStorage = df.groupby(['Outlook', 'Play']).size()['Sunny']

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

```python
aggreg = yesCount + noCount
aggregSunny = aggreg

entropySunny = -(((yesCount/aggreg)*math.log2(yesCount/aggreg)) + ((noCount/
 ↪aggreg)*math.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)*math.log2(noCount/aggreg)))
```

Print out the calculated Entropy values

[10]:
```python
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'

[11]:
```python
# 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

[12]:
```python
print("IG(S, outlook) = {0:.5f}\n".format(infoGain))
```

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

1.5.3 Entropy of column Temperature

Repeat procedure for each column

[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

```
[14]: offsetEntropy = (aggregHot/aggregTotal)*entropyHot + (aggregMild/
       ↪aggregTotal)*entropyMild + (aggregCool/aggregTotal)*entropyCool
      infoGain = entropyTotal - offsetEntropy

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

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

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

1.5.4 Entropy of column Humidity

```
[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)*math.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

[16]:
```
offsetEntropy = (aggregHigh/aggregTotal)*entropyHigh + (aggregNormal/
 ↪aggregTotal)*entropyNormal
infoGain = entropyTotal - offsetEntropy

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

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

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

1.5.5 Entropy of column Windy

---

[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)*math.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

[18]:
```
offsetEntropy = (aggregTrue/aggregTotal)*entropyTrue + (aggregFalse/
 ↪aggregTotal)*entropyFalse
infoGain = entropyTotal - offsetEntropy

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

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

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

2 Overview Information Gain for the Root-Node

2.1 Select Root-Node based on highest Information Gain value

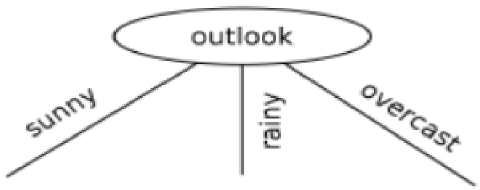
Pretty print out the collected Information Gain vaules with column name.

Reset of the dictionary with selected node as root.

[19]:
```
pprint.pprint(infoGainDict)
print("\nSelected '{0}' because of value {1:.5f}".format("Outlook",␣
 ↪infoGainDict['Root']['Outlook']))

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

```
{'Root': {'Humidity': 0.15183550136234136,
          'Outlook': 0.2467498197744391,
          'Temperature': 0.029222565658954647,
          'Windy': 0.04812703040826927}}

Selected 'Outlook' because of value 0.24675
{'Outlook': {}}
```

Decision tree as image



2.2 Branch Sunny

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

[20]:
```
dfS = df.loc[df['Outlook'] == 'Sunny']
dfS
```

[20]:

	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

2.2.1 Total entropy of branch Sunny dataframe

---

[21]:
```
# 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)*math.log2(noCount/aggreg)))

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

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

2.2.2 Entropy of column Temperature for branch Sunny

---

[22]:
```
# 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

[23]:
```
offsetEntropy = (aggregHot/aggregTotal)*entropyHot + (aggregMild/
 ↪aggregTotal)*entropyMild + (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
```

2.2.3 Entropy of column Humidity for branch Sunny

---

[24]:
```
# 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

[25]:
```python
offsetEntropy = (aggregHigh/aggregTotal)*entropyHigh + (aggregNormal/
 ↪aggregTotal)*entropyNormal
infoGain = entropyTotal - offsetEntropy

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

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

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

2.2.4 Entropy of column Windy for branch Sunny

---

[26]:
```python
# 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)*math.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

[27]:
```
offsetEntropy = (aggregTrue/aggregTotal)*entropyTrue + (aggregFalse/
 ↪aggregTotal)*entropyFalse
infoGain = entropyTotal - offsetEntropy

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

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

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

2.3 Overview Information Gain Branch Sunny

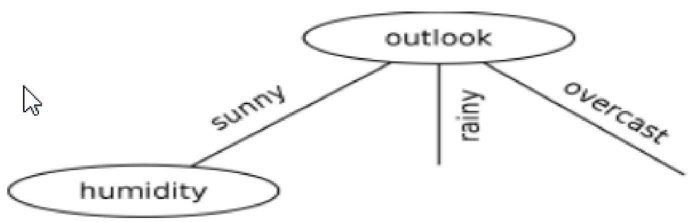
2.3.1 Select Node based on highest Information Gain value

---

[28]:
```
pprint.pprint(infoGainDict)

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

Decision tree



2.4 Subnode Humidity of Branch Sunny

2.4.1 High value

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

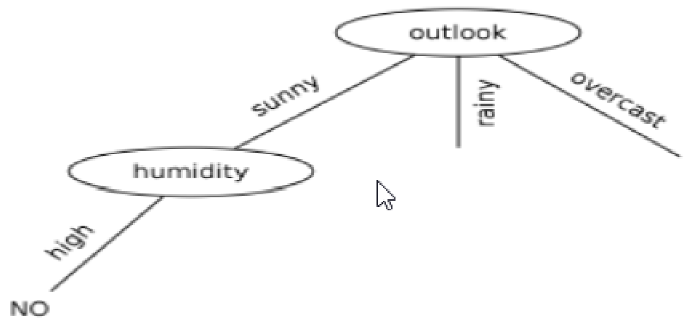
```
[29]:   Outlook Temperature Humidity Windy Play
      0   Sunny         Hot     High     F   No
      1   Sunny         Hot     High     T   No
      7   Sunny        Mild     High     F   No
```

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

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

Leaf end - No

Decision tree



2.4.2 Normal value

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

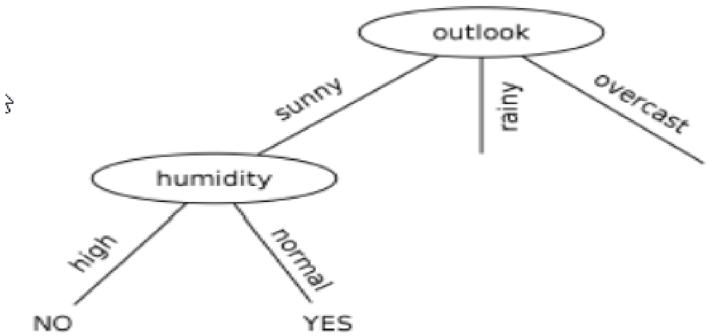
```
[31]:    Outlook Temperature Humidity Windy Play
      8    Sunny        Cool   Normal     F  Yes
      10   Sunny        Mild   Normal     T  Yes
```

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

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

Leaf end - Yes

Decision tree



2.5 Branch Overcast

---

[33]:
```
df0 = df.loc[df['Outlook'] == 'Overcast']
df0
```

[33]:

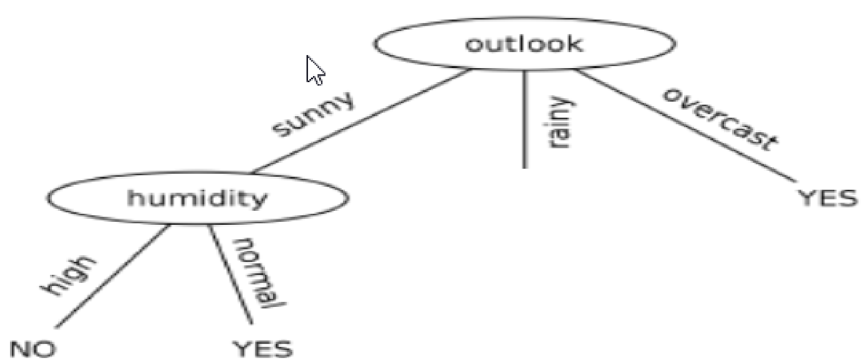
	Outlook	Temperature	Humidity	Windy	Play
2	Overcast	Hot	High	F	Yes
6	Overcast	Cool	Normal	T	Yes
11	Overcast	Mild	High	T	Yes
12	Overcast	Hot	Normal	F	Yes

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

[34]:
```
infoGainDict["Outlook"]['BranchOvercast'] = "Yes"
```

Leaf end - Yes

Decision tree



2.6 Branch Rainy

---

[35]:
```
dfR = df.loc[df['Outlook'] == 'Rainy']
dfR
```

[35]:

	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

2.6.1 Total entropy of branch Rainy dataframe

[36]:
```
# 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)*math.log2(noCount/aggreg)))

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

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

2.6.2 Entropy of column Temperature for branch Rainy

---

[37]:
```
# 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

[38]:
```
offsetEntropy = (aggregMild/aggregTotal)*entropyMild + (aggregCool/
 ↪aggregTotal)*entropyCool
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
```

2.6.3 Entropy of column Humidity for branch Rainy

---

[39]:
```
# 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)*math.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

[40]:
```
offsetEntropy = (aggregHigh/aggregTotal)*entropyHigh + (aggregNormal/
 ↪aggregTotal)*entropyNormal
infoGain = entropyTotal - offsetEntropy

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

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

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

2.6.4 Entropy of column Windy for branch Rainy

---

[41]:
```
# 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

[42]:
```
offsetEntropy = (aggregTrue/aggregTotal)*entropyTrue + (aggregFalse/
↪aggregTotal)*entropyFalse
infoGain = entropyTotal - offsetEntropy

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

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

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

2.7 Overview Information Gain Branch Rainy

2.7.1 Select Node based on highest Information Gain value

---

[43]:
```
pprint.pprint(infoGainDict)
print("\nSelected '{0}' because of value {1:.5f}".format("Windy",␣
↪infoGainDict['Outlook']['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': 'Yes'}}}}

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

2.8 Subnode Windy of Branch Rainy

2.8.1 High value

[44]:
```python
dfRW = dfR.loc[dfR['Windy'] == 'T']
dfRW
```

[44]:

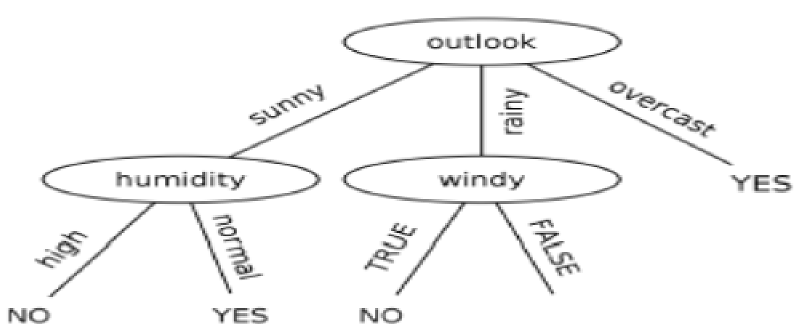
	Outlook	Temperature	Humidity	Windy	Play
5	Rainy	Cool	Normal	T	No
13	Rainy	Mild	High	T	No

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

[45]:
```python
infoGainDict["Outlook"]['BranchRainy']['Windy']['T'] = "No"
```

Leaf end - No

Decision tree



2.8.2 Normal value

[46]:
```
dfRW = dfR.loc[dfR['Windy'] == 'F']
dfRW
```

[46]:

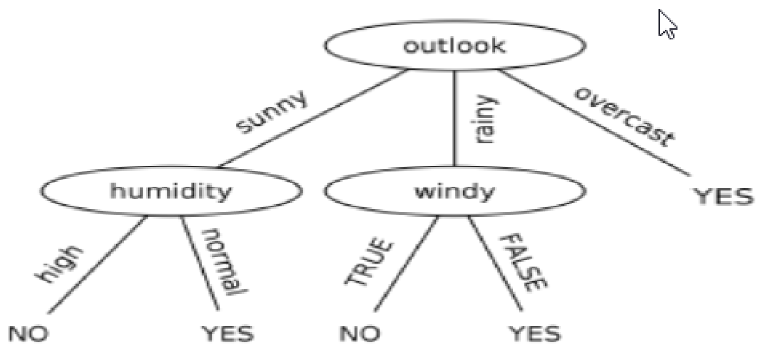
	Outlook	Temperature	Humidity	Windy	Play
3	Rainy	Mild	High	F	Yes
4	Rainy	Cool	Normal	F	Yes
9	Rainy	Mild	Normal	F	Yes

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

[47]:
```
infoGainDict["Outlook"]['BranchRainy']['Windy']['F'] = "Yes"
```

Leaf end - Yes

Decision tree



[48]:
```
#Final pprint of the dicitionary
pprint.pprint(infoGainDict)
```

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

[ ]:

[ ]:

[ ]:

2.9 Automation try

reads dataframe and selects root node

[49]:
```
df = pd.read_csv("playTennis.csv", sep=';')
df
```

[49]:
```
      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
```

[50]:
```python
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/
 ↪sumYesNo)*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)*sunnyEntropy + (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+noCount)

    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,␣
 ↪rootDictInformationGain[selectRootNode]))


blackList[0] = selectRootNode
```

```
Entropies for total and subsets of Root-Node:
{'root': {'Humidity': {'High': {'AggCount': 7, 'Entropy': 0.9852281360342515},
                       'Normal': {'AggCount': 7,
                                  'Entropy': 0.5916727785823275}},
          'Outlook': {'Overcast': {'AggCount': 4, 'Entropy': -0.0},
                      'Rainy': {'AggCount': 5, 'Entropy': 0.9709505944546686},
                      'Sunny': {'AggCount': 5, 'Entropy': 0.9709505944546686}},
          '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': 0.9402859586706309}}}

Information Gains for Root-Node:
{'Humidity': 0.15183550136234136,
 'Outlook': 0.2467498197744391,
 'Temperature': 0.029222565658954647,
 'Windy': 0.04812703040826927}

Selected 'Outlook' because of value 0.24675
```

[ ]: