Tugas Besar 1B IF3270 - Machine Learning

NIM/Nama : 13517014/Yoel Susanto | 13517065/Andrian Cedric | 13517131/Jan Meyer Saragih | 13517137/Vincent Budianto

Nama file : Tubes1B_13517014.ipynb

Topik : Implementasi modul ID3 dan C4.5

Tanggal : 14 February 2020

In [1]:

```
from DecisionTree import DecisionTree
from FitC45 import *
from sklearn.tree.export import export_text

import Function as f
import id3 as id
import numpy as np
import pandas as pd
import sklearn.datasets as dataset
import sklearn.preprocessing as preprocessing
import sklearn.tree as tree
```

A. Sklearn, ID3Estimator and ID3 + C4.5 Result Comparison

In [2]:

```
iris = dataset.load_iris()
tennis = pd.read_csv('tennis.csv')
tHead = list(tennis.columns)
tennis = np.array(tennis)
decision_tree = tree.DecisionTreeClassifier(criterion='entropy')
estimator = id.Id3Estimator()
encoder = preprocessing.LabelEncoder()
```

A. Dataset iris

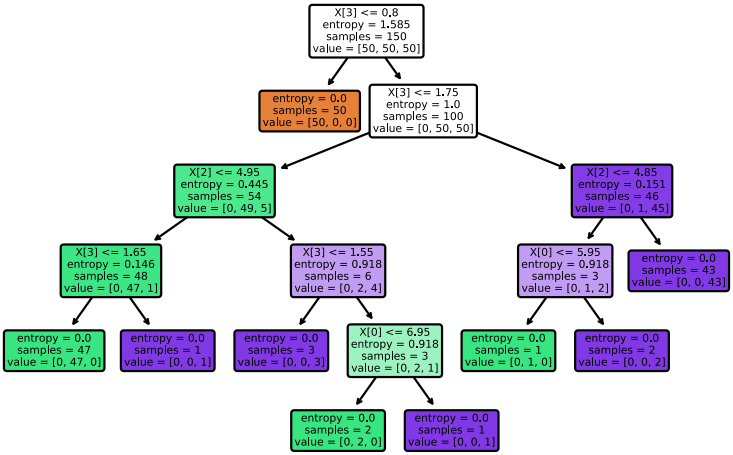
a. DecisionTreeClassifier

In [4]:

```
iris_tree1 = decision_tree.fit(iris.data, iris.target)
tree.plot_tree(iris_tree1, filled=True, rounded=True)
```

Out[4]:

```
[Text(167.4, 199.32, 'X[3] <= 0.8\nentropy = 1.585\nsa
mples = 150\nvalue = [50, 50, 50]'),
 Text(141.64615384615385, 163.07999999999998, 'entropy
= 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(193.15384615384616, 163.07999999999998, 'X[3] <=
1.75\nentropy = 1.0\nsamples = 100\nvalue = [0, 50, 5
0]'),
 Text(103.01538461538462, 126.83999999999999, 'X[2] <=
4.95\nentropy = 0.445\nsamples = 54\nvalue = [0, 49,
5]'),
 Text(51.50769230769231, 90.6, 'X[3] <= 1.65\nentropy
= 0.146\nsamples = 48\nvalue = [0, 47, 1]'),
 Text(25.753846153846155, 54.359999999999985, 'entropy
= 0.0\nsamples = 47\nvalue = [0, 47, 0]'),
 Text(77.26153846153846, 54.359999999999985, 'entropy
= 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(154.52307692307693, 90.6, 'X[3] <= 1.55\nentropy
= 0.918\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(128.76923076923077, 54.359999999999985, 'entropy
= 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(180.27692307692308, 54.359999999999985, 'X[0] <=
6.95\nentropy = 0.918\nsamples = 3\nvalue = [0, 2,
1]'),
 Text(154.52307692307693, 18.119999999999976, 'entropy
= 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(206.03076923076924, 18.119999999999976, 'entropy
= 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(283.2923076923077, 126.83999999999999, 'X[2] <=
4.85\nentropy = 0.151\nsamples = 46\nvalue = [0, 1, 4
5]'),
 Text(257.53846153846155, 90.6, 'X[0] <= 5.95\nentropy
= 0.918\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(231.7846153846154, 54.359999999999985, 'entropy
= 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(283.2923076923077, 54.359999999999985, 'entropy
= 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(309.04615384615386, 90.6, 'entropy = 0.0\nsample
s = 43\nvalue = [0, 0, 43]')]
```


X[3] <= 0.8
entropy = 1.585
samples = 150
value = [50, 50, 50]
entropy = 0.0
samples = 50
value = [50, 0, 0]
X[3] <= 1.75
entropy = 1.0
samples = 100
value = [0, 50, 50]
X[2] <= 4.95
entropy = 0.445
samples = 54
value = [0, 49, 5]
X[2] <= 4.85
entropy = 0.151
samples = 46
value = [0, 1, 45]
X[3] <= 1.65
entropy = 0.146
samples = 48
value = [0, 47, 1]
X[3] <= 1.55
entropy = 0.918
samples = 6
value = [0, 2, 4]
X[0] <= 5.95
entropy = 0.918
samples = 3
value = [0, 1, 2]
entropy = 0.0
samples = 43
value = [0, 0, 43]
entropy = 0.0
samples = 47
value = [0, 47, 0]
entropy = 0.0
samples = 1
value = [0, 0, 1]
entropy = 0.0
samples = 3
value = [0, 0, 3]
X[0] <= 6.95
entropy = 0.918
samples = 3
value = [0, 2, 1]
entropy = 0.0
samples = 1
value = [0, 1, 0]
entropy = 0.0
samples = 2
value = [0, 0, 2]
entropy = 0.0
samples = 2
value = [0, 2, 0]
entropy = 0.0
samples = 1
value = [0, 0, 1]

In [5]:

```
ri1 = export_text(iris_tree1, feature_names=iris['feature_names'])
print(ri1)
```

```
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- petal width (cm) <= 1.65
|   |   |   |   |--- class: 1
|   |   |   |--- petal width (cm) >  1.65
|   |   |   |   |--- class: 2
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- petal width (cm) <= 1.55
|   |   |   |   |--- class: 2
|   |   |   |--- petal width (cm) >  1.55
|   |   |   |   |--- sepal length (cm) <= 6.95
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- sepal length (cm) >  6.95
|   |   |   |   |   |--- class: 2
|   |--- petal width (cm) >  1.75
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- sepal length (cm) <= 5.95
|   |   |   |   |--- class: 1
|   |   |   |--- sepal length (cm) >  5.95
|   |   |   |   |--- class: 2
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

b. Id3Estimator

In [6]:

```
iris_tree2 = estimator.fit(iris.data, iris.target)
```

In [7]:

```
ri2 = id.export_text(iris_tree2.tree_, feature_names=iris['feature_
names'])
print(ri2)
```

```
petal length (cm) <=2.45: 0 (50)
petal length (cm) >2.45
|   petal width (cm) <=1.75
|   |   sepal length (cm) <=7.10
|   |   |   sepal width (cm) <=2.85: 1 (27/4)
|   |   |   sepal width (cm) >2.85: 1 (22)
|   |   sepal length (cm) >7.10: 2 (1)
|   petal width (cm) >1.75
|   |   sepal length (cm) <=5.95
|   |   |   sepal width (cm) <=3.10: 2 (6)
|   |   |   sepal width (cm) >3.10: 1 (1)
|   |   sepal length (cm) >5.95: 2 (39)
```

C. ID3 + C4.5

In [8]:

```
dataHead, data, dataRaw = getCSVData("iris.csv")
prune(dataHead, data, dataRaw)
```

```
Tree Result:
petal.length <= 2.45 : 40
|  Setosa
petal.length > 2.45 : 44/36
|  petal.width <= 1.75 : 43/3
|  |  sepal.length <= 6.95 : 43/2
|  |  |  sepal.width <= 2.25 : 2/1
|  |  |  |  Versicolor
|  |  |  sepal.width > 2.25 : 41/1
|  |  |  |  Versicolor
|  |  sepal.length > 6.95 : 1
|  |  |  Virginica
|  petal.width > 1.75 : 1/33
|  |  sepal.length <= 5.95 : 1/4
|  |  |  sepal.width <= 3.05 : 4
|  |  |  |  Virginica
|  |  |  sepal.width > 3.05 : 1
|  |  |  |  Versicolor
|  |  sepal.length > 5.95 : 29
|  |  |  Virginica
```

C. Dataset tennis

a. DecisionTreeClassifier

In [9]:

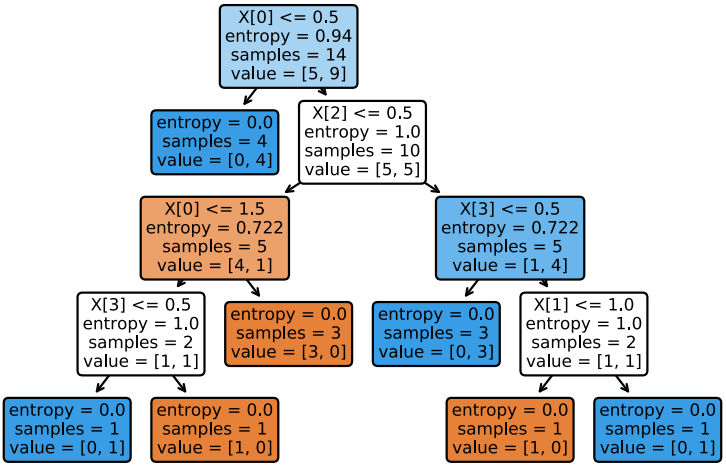
```
tennis[:, 0] = encoder.fit_transform(tennis[:, 0])
tennis[:, 1] = encoder.fit_transform(tennis[:, 1])
tennis[:, 2] = encoder.fit_transform(tennis[:, 2])
tennis[:, 3] = encoder.fit_transform(tennis[:, 3])
tennis[:, 4] = encoder.fit_transform(tennis[:, 4])

tData = tennis[:, :4]
tTarget = list(tennis[:, 4])

tennis_tree1 = decision_tree.fit(tData, tTarget)
tree.plot_tree(tennis_tree1, filled=True, rounded=True)
```

Out[9]:

```
[Text(133.92000000000002, 195.696, 'X[0] <= 0.5\nentro
py = 0.94\nsamples = 14\nvalue = [5, 9]'),
 Text(100.44000000000001, 152.208, 'entropy = 0.0\nsam
ples = 4\nvalue = [0, 4]'),
 Text(167.40000000000003, 152.208, 'X[2] <= 0.5\nentro
py = 1.0\nsamples = 10\nvalue = [5, 5]'),
 Text(100.44000000000001, 108.72, 'X[0] <= 1.5\nentrop
y = 0.722\nsamples = 5\nvalue = [4, 1]'),
 Text(66.96000000000001, 65.232, 'X[3] <= 0.5\nentropy
= 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(33.480000000000004, 21.744, 'entropy = 0.0\nsamp
les = 1\nvalue = [0, 1]'),
 Text(100.44000000000001, 21.744, 'entropy = 0.0\nsamp
les = 1\nvalue = [1, 0]'),
 Text(133.92000000000002, 65.232, 'entropy = 0.0\nsamp
les = 3\nvalue = [3, 0]'),
 Text(234.36, 108.72, 'X[3] <= 0.5\nentropy = 0.722\ns
amples = 5\nvalue = [1, 4]'),
 Text(200.88000000000002, 65.232, 'entropy = 0.0\nsamp
les = 3\nvalue = [0, 3]'),
 Text(267.84000000000003, 65.232, 'X[1] <= 1.0\nentrop
y = 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(234.36, 21.744, 'entropy = 0.0\nsamples = 1\nval
ue = [1, 0]'),
 Text(301.32000000000005, 21.744, 'entropy = 0.0\nsamp
les = 1\nvalue = [0, 1]')]
```



In [10]:

```
rt1 = export_text(tennis_tree1, feature_names=tHead[:4])
print(rt1)
```

```
|--- outlook <= 0.50
|   |--- class: 1
|--- outlook >  0.50
|   |--- humidity <= 0.50
|   |   |--- outlook <= 1.50
|   |   |   |--- windy <= 0.50
|   |   |   |   |--- class: 1
|   |   |   |--- windy >  0.50
|   |   |   |   |--- class: 0
|   |   |--- outlook >  1.50
|   |   |   |--- class: 0
|   |--- humidity >  0.50
|   |   |--- windy <= 0.50
|   |   |   |--- class: 1
|   |   |--- windy >  0.50
|   |   |   |--- temp <= 1.00
|   |   |   |   |--- class: 0
|   |   |   |--- temp >  1.00
|   |   |   |   |--- class: 1
```

b. Id3Estimator

In [11]:

```
tennis_tree2 = estimator.fit(tData, tTarget)
```

In [12]:

```
rt2 = id.export_text(tennis_tree2.tree_, feature_names=tHead[:4])
print(rt2)
```

```
outlook <=0.50: 1 (4)
outlook >0.50
|   humidity <=0.50
|   |   temp <=1.50: 0 (2)
|   |   temp >1.50
|   |   |   windy <=0.50: 0 (1/1)
|   |   |   windy >0.50: 0 (1)
|   humidity >0.50
|   |   windy <=0.50: 1 (3)
|   |   windy >0.50
|   |   |   temp <=1.00: 0 (1)
|   |   |   temp >1.00: 1 (1)
```

C. ID3 + C4.5

In [13]:

```
dataHead, data, dataRaw = getCSVData("tennis.csv")
prune(dataHead, data, dataRaw)
```

```
Tree Result:
humidity == high
|  outlook == sunny
|  |  no
|  outlook == overcast
|  |  yes
|  outlook == rainy
|  |  no
humidity == normal
|  outlook == sunny
|  |  yes
|  outlook == overcast
|  |  yes
|  outlook == rainy
|  |  windy == False
|  |  |  yes
|  |  windy == True
|  |  |  no
```

B. Explanation of Implementation ID3 + C4.5

Pseudo-code

Part 1: Preparing the data

1. Read the CSV data
2. Splits the dataHead (metadata) and the rest of the data
3. Splits the data into x and y
    - X is the data that has all the attributes in it
    - Y is the data that has all the results (the verdict)
4. Encodes all the data in x and y and saves it into the dictionary
    - If the attribute is discrete, the dictionary stores all possible values of that attribute
    - If the attribute is continuous, the dictionary stores the breakpoints of the attribute. Breakpoint is created when there is the change in attribute's value which creates a change in target's value

Part 2: Decision tree building

1. Initiates the all possible attributes to be checked
2. Receives encoded data of X and Y as well as the dictionaries
3. Do until it the data is uniform or until there is no more attributes that can be checked (recursive):
    - Initiates information gain, splitted attributes container, and splitted target container.
    - Loop through all available attributes in the possible attributes:
        - Loop through all possible values that has been stored in the dictionary
            A. Get the entropy of each loop
            B. Store temporary splitted attributes temporary container and splitted target temporary container
        - Calculate the information gain
        - If the information gain > initiated information gain (usually 0)
            A. Then move the all temporary containers (splitted data temporary container and splitted target container) into the actual container
            B. Set the value of maximum information gain (value that must be surpassed if another attribute will be chosen as the root)

- Maximum information gain is chosen. Set the node value into the name of the attribute
- Remove the attribute orm the possible attributes for the next recursive call.
- For all possible roots (the amount of data splitted):
  A. Recursively call function number III again.

C4.5 Improvements

1. Handling continuous attributes
   - C4.5 creates a threshold of values to classify the values. Created as follows:
     - Loop through the data and target.
     - If the target on the current index and the target on the previous index is different as well as if the checked attribute of the current data index is different than the previous index, then create breakpoint
     - Breakpoint = (previous value + current value) / 2
     - All the data from the (previous breakpoint + 1) (or first element if breakpoint hasn't been made yet) is clustered into one group.
   - When looping for the possible best information gain, do as follows:
     - For every breakpoint
       A. Classify the data into 2 groups (one is less or equal than the breakpoint, the other one is bigger than the breakpoint)
       B. Calculate the entropy and information gain
       C. If the breakpoint's information gain, then set the best
     - Get the maximum information gain, which is the maximum information gain from every possible breakpoints in the attribute
     - Get the maximum information gain from all attributes
2. Handling missing attribute value
   - C4.5 allows missing attribute values to be marked as most common target value.
3. Handling attributes with many value

   - Creates a decision tree that has more than two child nodes
   - Used only for discrete values, as the continuous values always be split into two childs
4. Post-pruning
   - Split the data into 80% and 20% (80% becomes training data and 20% becomes testing data)

- Build a decision tree
- Converts data to set of rules
- Calculate initial accuracy with testing data
- Test the pruning set of rules
  - One of the rules was changed by reducing the number of conditions (IF)
  - If the rule (pre-run) does not reduce accuracy, then do the pruning
  - If not, try another rule
  - If all rules have been tried and accuracy is reduced, then stop pruning

C. Pembagian Tugas

NIM	Nama	Tugas
13517014	Yoel Susanto	ID3, Missing attribute value, Attributes with many value, Post-pruning
13517065	Andrian Cedric	ID3, Missing attribute value, Post-pruning, Documentation
13517131	Jan Meyer Saragih	ID3, Continous attributes, Attributes with many value, Post-pruning
13517137	Vincent Budianto	ID3, Continous attributes, Post-pruning, Documentation