Tugas Besar 1A IF3270 - Machine Learning

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

Nama file : Tubes1A_13517014.ipynb

Topik : Eksplorasi scikit-learn pada jupyter notebook dan pembelajaran DTL dengan DecisionTreeClassifier dan ID3Estimator

Tanggal : 03 February 2020

In [1]:
```
from sklearn.tree.export import export_text

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
```

```
C:\Python\Python37-32\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarnin
g: The sklearn.tree.export module is  deprecated in version 0.22 and will be removed in
version 0.24. The corresponding classes / functions should instead be imported from skl
earn.tree. Anything that cannot be imported from sklearn.tree is now part of the privat
e API.
  warnings.warn(message, FutureWarning)
C:\Python\Python37-32\lib\site-packages\sklearn\externals\six.py:31: FutureWarning: The
module is deprecated in version 0.21 and will be removed in version 0.23 since we've dr
opped support for Python 2.7. Please rely on the official version of six (https://pypi.
org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
```

A. Load Datasets

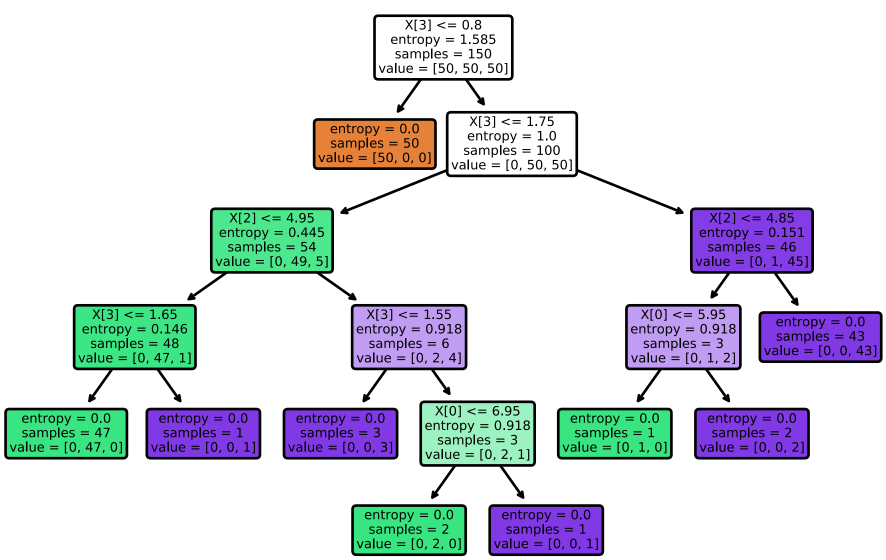
In [2]:
```
iris = dataset.load_iris()
tennis = pd.read_csv('tennis.csv')
tHead = list(tennis.columns)
tennis = np.array(tennis)
decision_tree = tree.DecisionTreeClassifier(criterion='entropy')
estimator = id.Id3Estimator()
encoder = preprocessing.LabelEncoder()
```

B. Dataset iris

a. DecisionTreeClassifier

In [4]:
```python
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\nsamples = 150\nvalue = [50, 50, 5
0]'),
 Text(141.64615384615385, 163.07999999999998, 'entropy = 0.0\nsamples = 50\nvalue = [5
0, 0, 0]'),
 Text(193.15384615384616, 163.07999999999998, 'X[3] <= 1.75\nentropy = 1.0\nsamples = 1
00\nvalue = [0, 50, 50]'),
 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, 45]'),
 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\nsamples = 43\nvalue = [0, 0, 43]')]
```



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. Dataset tennis

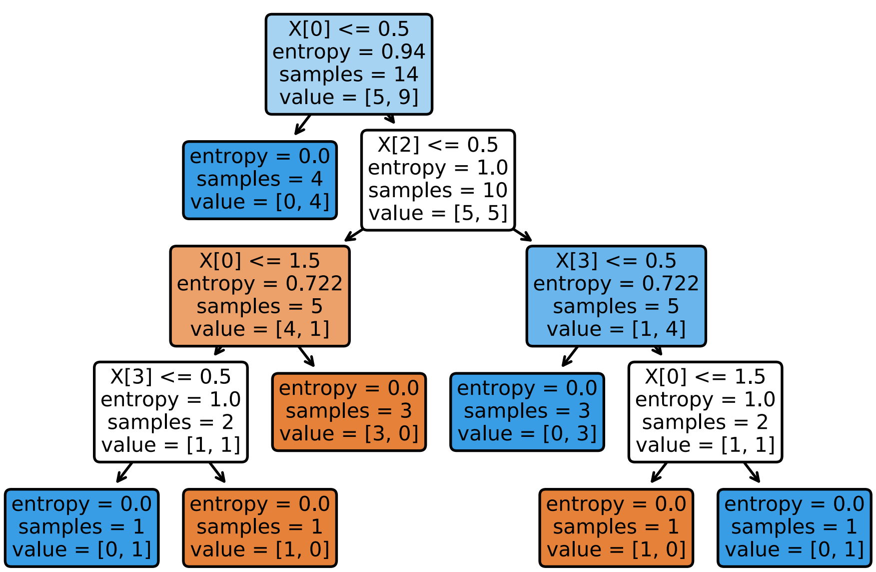
a. DecisionTreeClassifier

In [8]:
```
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[8]:
```
[Text(133.92000000000002, 195.696, 'X[0] <= 0.5\nentropy = 0.94\nsamples = 14\nvalue =
[5, 9]'),
 Text(100.44000000000001, 152.208, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(167.40000000000003, 152.208, 'X[2] <= 0.5\nentropy = 1.0\nsamples = 10\nvalue =
[5, 5]'),
 Text(100.44000000000001, 108.72, 'X[0] <= 1.5\nentropy = 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\nsamples = 1\nvalue = [0, 1]'),
 Text(100.44000000000001, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(133.92000000000002, 65.232, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(234.36, 108.72, 'X[3] <= 0.5\nentropy = 0.722\nsamples = 5\nvalue = [1, 4]'),
 Text(200.88000000000002, 65.232, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(267.84000000000003, 65.232, 'X[0] <= 1.5\nentropy = 1.0\nsamples = 2\nvalue = [1,
1]'),
 Text(234.36, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(301.32000000000005, 21.744, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]')]
```



In [9]:
```python
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
|   |   |   |--- outlook <= 1.50
|   |   |   |   |--- class: 0
|   |   |   |--- outlook >  1.50
|   |   |   |   |--- class: 1
```

b. Id3Estimator

In [10]:
```python
tennis_tree2 = estimator.fit(tData, tTarget)
```

In [11]:
```python
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. Perbedaan dan Persamaan Algoritma DecisionTreeClassifier dengan Id3Estimator

	DecisionTreeClassifier	Id3Estimator	Machine Learning by Tom M. Mitchel
Penentuan atribut terbaik	*DecisionTreeClassifier* menggunakan entropi untuk menentukan atribut terbaik. Beberapa *Attribute Selection Measure* yang popular adalah *Gain Ratio*, *Gini Index*, dan *Information Gain*. Pada *Gini Index*, atribut yang memiliki nilai *Gini Index* terendah akan dipakai sebagai *splitting attribute*. Pada *Gain Ratio*, atribut yang memiliki nilai *Gain Ratio* tertinggi akan dipakai sebagai *splitting attribute*. Pada *Information Gain*, atribut yang memiliki nilai tertinggi akan dipakai sebagai *splitting attribute*.	Id3Estimator hanya menggunakan entropi untuk menentukan *information gain*. Strategi algoritma yang digunakan adalah *Greedy*. Dari atribut-atribut yang ada, dipilih atribut terbaik, yaitu yang memberikan information gain terbanyak. Sama seperti buku, menggunakan rumus $Gain(S, A)$. Setelah salah satu atribut digunakan, maka atribut itu tidak digunakan pada penentuan atribut terbaik dari *node* anaknya (baik cabang *left* maupun *right*). Walaupun begitu, nilai *gain* yang didapat dari Id3Estimator menggunakan atribut *gain ratio*. Yaitu, nilai *gain* dibagi dengan *split information*.	Digunakan untuk menentukan nilai *root*. Menerima sekumpulan atribut bernama *Examples* yang kemudian dipilih salah satu nilai terbaiknya. Buku mengajarkan cara untuk menentukan atribut terbaik dengan melakukan nilai informasi yang didapat dari nilai $Gain(S, A)$ yang paling besar.
Penanganan label dari cabang setiap nilai atribut	Jika semua label positif, mengembalikan nilai +. Jika semua label negatif, mengembalikan nilai -. Jika terdapat nilai tercampur dan atribut tidak kosong, maka dicari atribut yang dijadikan penentu, bisa berupa *sampling* maupun dengan *class* yang ada.	Jika semua label positif, mengembalikan nilai +. Jika semua label negatif, mengembalikan nilai -. Jika terdapat nilai tercampur dan atribut tidak kosong, maka dicari atribut yang dijadikan penentu.	Jika semua label positif, mengembalikan nilai +. Jika semua label negatif, mengembalikan nilai -. Jika terdapat nilai tercampur dan atribut tidak kosong, maka dicari atribut yang dijadikan penentu.
Penentuan label jika examples kosong di cabang tersebut	Jika *Examples* kosong, *node* akan mengembalikan kelas yang ada dan menjadi *leaf node*.	Jika *examples* kosong, maka *node* tersebut akan menghasilkan kembalian “*failure*” *node*. Dengan demikian, setiap *node* harus memiliki *example*.	Jika tidak terdapat *example*, maka *node* tersebut menjadi sebuah *leaf node* dan berikan nilai atribut *target* yang paling umum dari *Example parent*.

	DecisionTreeClassifier	Id3Estimator	Machine Learning by Tom M. Mitchel
Penanganan atribut kontinu	*Algoritma Decision Tree* akan melakukan pengurutan data kontinu mulai dari yang terkecil hingga terbesar. Tahap selanjutnya algoritma akan melakukan kalkulasi untuk mendapatkan titik tengah antara setiap nilai kontinu yang didapat. Misalkan data berupa $[1, 2, 3, 4]$ maka akan dihitung titik tengah antara masing-masing data yaitu $[1.5, 2.5, 3.5]$. Setelah titik tengah tersebut didapat, maka titik tersebut akan digunakan untuk *splitting point*. Lalu algoritma akan menggunakan *splitting point* tersebut untuk menghitung *gain* terbesar pada *decision tree* (*spliting point* yang menghasilkan penurunan entropi terbesar antara entropi sebelum dan sesudah dilakukan *splitting*). Titik tengah yang menghasilkan *gain* terbesar itulah yang akan digunakan sebagai *splitting point pada node tertentu.	*Decision Tree* yang menghasilkan data yang bersifat kontinu akan di-"diskritkan" sebelum dimasukkan ke dalam fungsi *fit*. Dengan kata lain algoritma *fit* tidak menangani data yang bersifat kontinu. Cara yang dilakukan sama dengan buku Tom M. Mitchell, yaitu dengan membuat *breakpoint* dan menghitungkan nilai peningkatan entropi menggunakan $GainRatio(S, A)$. Penggunaan $GainRatio(S, A)$ lakukan sebagai berikut. Pertama dilihat perubahan nilai atribut target sepanjang perubahan. Selama ada perubahan nilai atribut target, disematkan breakpoint di sana. Setelah itu, dihitung nilai GainRatio(S, A) untuk setiap breakpoint yang ada. Breakpoint yang memberikan nilai $GainRatio(S, A)$ terbesar dijadikan *breakpoint* yang dijadikan *node* dalam *decision tree*.	Sebenarnya, algoritma hanya dapat menangani atribut yang bernilai diskrit. Namun, apabila terdapat atribut kontinu, cara berikut dilakukan. Mencari threshold yang tepat untuk menentukan *breakpoint* (sehingga setiap atribut digunakan dalam *range* nilai). Pertama dihitung $SplitInformation(S, A)$ yang merupakan nilai pembagian atribut. Nilai ini akan bernilai semakin besar apabila atribut semakin dibagi. Setelah itu, dicari $GainRatio(S, A)$ yang merupakan nilai $Gain(S, A)$ dibagi $SplitInformation(S, A)$. Dari sini dapat dilihat kalau semakin banyak *breakpoint*, maka semakin rendah nilai $GainRatio(S, A)$ apabila $Gain(S, A)$ bernilai sama. Dengan kata lain, lebih baik mendapatkan informasi apabila breakpoint berkurang. $$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{\lvert S_i \rvert}{\lvert S \rvert} log_2 \frac{\lvert S_i \rvert}{\lvert S \rvert}$$ $$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$ Kedua rumus di atas digunakan untuk menentukan *gain ratio* jika nilai bernilai kontinu. Dengan demikian parameter penentuan atribut baik bukanlah *gain*, tapi *gain ratio*.
Penanganan atribut dengan *missing values*	Atribut dengan *missing values* akan ditangani dengan mencari *value* yang paling umum yang terdapat dalam *node* yang sama. (sama seperti *Id3Estimator*)	Atribut dengan *missing values* akan ditangani dengan mencari *value* yang paling umum yang terdapat dalam *node* yang sama. (sama seperti *DecisionTreeClassifier*)	Algoritma sebenarnya tidak menangani *missing values*. Namun, jika terdapat *missing values*, beberapa cara berikut dapat dilakukan. Pertama, *assign value* tersebut ke nilai atribut yang paling umum dalam node yang sama. Kedua, meng-*assign* probabilitas untuk setiap value yang ada dari node yang sama.

	DecisionTreeClassifier	**Id3Estimator**	**Machine Learning by Tom M. Mitchel**
Pruning dan *parameter confidence*	Ada dua cara dalam melakukan *pruning* pada *DecisionTreeClassifier*. Secara *default*, *DecisionTreeClassifier* menggunakan parameter **min_sample_leaf** dan **max_depth**(kedalaman maksimal dari pohon keputusan) untuk membatasi besaran dari pohon. *Pruning* yang dapat dilakukan yaitu dengan *Cost Complexity Pruning*. Cara kerjanya adalah dengan menganalisis *effective alpha* untuk mencari ikatan yang paling lemah. Kemudian, *node* dengan *effective alpha* terkecil akan di-*pruning*.	Menggunakan 3 parameter untuk *pruning*, yaitu: • **max_depth** Kedalaman maksimum dari sebuah *decision tree* • **min_samples_split** Jumlah minimum nilai sampel yang terbagi dari atribut • **min_entropy_decrease** Nilai *minimum gain* yang didapat dari sebuah atribut untuk dimasukkan sebagai *node* sebuah *tree* *Pruning* yang dilakukan menggunakan *Cost Effective Pruning*, yang memanfaatkan atribut **min_entropy_decrease**. Jika *tree* yang baru dibuat menghasilkan nilai *gain* yang tidak signifikan, maka cabang tersebut akan di-*prune*. $\delta = \frac{err(T_{new})-err(T)}{\|T_{new}\|+\|T\|}$ $\|T\| =$ Number of leaves Selain menggunakan *Cost Effective Pruning*, *Id3Estimator* juga memanfaatkan cara *Random Forest*, yaitu membentuk beberapa *decision trees*. Lalu, mengambil *decision tree* yang paling sering muncul (modus).	Terdapat 2 jenis *pruning*: • *Reduced Error Pruning* Menghasilkan *tree* yang memberikan hasil yang paling akurat dengan *subset* paling kecil. • *Rule-Post Pruning* Digunakan dengan membuat *pruning* setelah membuat *decision tree* hingga akhir. Setelah itu, *convert tree* menjadi *rule*, lalu *generalize rule-rule* yang ada. Setelah itu, *sort rule*-nya sehingga menjadi *sequence* yang diinginkan. Parameter *confidence* merupakan tingkat akurasi *decision tree* terhadap *data training*.