Tubes I Bagian A: Eksplorasi library Decision Tree Learning pada Jupyter Notebook

Oleh:
• Kevin Angelo 13517086
• Juro Sutantra 13517113
• Nurul Utami Amaliah W. 13517134
• Tasya Lailinissa Diandraputri 13517141

In [13]:

```
# Libraries
from sklearn import datasets
import pandas as pd
from sklearn import tree
from sklearn.tree import export_text
from sklearn import preprocessing
from id3 import Id3Estimator
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

In [14]:

```
# Read iris dataset
data_iris = datasets.load_iris()
data_iris
```

Out[14]:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
```

In [15]:

```
# Read play-tennis dataset
data_play_tennis = pd.read_csv('play-tennis.csv')
data_play_tennis = data_play_tennis.values.tolist()
data_play_tennis
```

Out[15]:

```
[['D1', 'Sunny', 'Hot', 'High', 'Weak', 'No'],
 ['D2', 'Sunny', 'Hot', 'High', 'Strong', 'No'],
 ['D3', 'Overcast', 'Hot', 'High', 'Weak', 'Yes'],
 ['D4', 'Rain', 'Mild', 'High', 'Weak', 'Yes'],
 ['D5', 'Rain', 'Cool', 'Normal', 'Weak', 'Yes'],
 ['D6', 'Rain', 'Cool', 'Normal', 'Strong', 'No'],
 ['D7', 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes'],
 ['D8', 'Sunny', 'Mild', 'High', 'Weak', 'No'],
 ['D9', 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes'],
 ['D10', 'Rain', 'Mild', 'Normal', 'Weak', 'Yes'],
 ['D11', 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes'],
 ['D12', 'Overcast', 'Mild', 'High', 'Strong', 'Yes'],
 ['D13', 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes'],
 ['D14', 'Rain', 'Mild', 'High', 'Strong', 'No']]
```

In [16]:

```
# play-tennis target
data_play_tennis_target = []
for x in data_play_tennis:
    data_play_tennis_target.append(x[-1:])
data_play_tennis_target
```

Out[16]:

```
[['No'],
 ['No'],
 ['Yes'],
 ['Yes'],
 ['Yes'],
 ['No'],
 ['Yes'],
 ['No'],
 ['Yes'],
 ['Yes'],
 ['Yes'],
 ['Yes'],
 ['Yes'],
 ['No']]
```

In [17]:

```
# play-tennis data
data_play_tennis_data = []
for x in data_play_tennis:
    data_play_tennis_data.append(x[:-1][1:])
data_play_tennis_data
```

Out[17]:

```
[['Sunny', 'Hot', 'High', 'Weak'],
 ['Sunny', 'Hot', 'High', 'Strong'],
 ['Overcast', 'Hot', 'High', 'Weak'],
 ['Rain', 'Mild', 'High', 'Weak'],
 ['Rain', 'Cool', 'Normal', 'Weak'],
 ['Rain', 'Cool', 'Normal', 'Strong'],
 ['Overcast', 'Cool', 'Normal', 'Strong'],
 ['Sunny', 'Mild', 'High', 'Weak'],
 ['Sunny', 'Cool', 'Normal', 'Weak'],
 ['Rain', 'Mild', 'Normal', 'Weak'],
 ['Sunny', 'Mild', 'Normal', 'Strong'],
 ['Overcast', 'Mild', 'High', 'Strong'],
 ['Overcast', 'Hot', 'Normal', 'Weak'],
 ['Rain', 'Mild', 'High', 'Strong']]
```

In [18]:

```
# DecisionTreeClassifier iris
iris_tree = tree.DecisionTreeClassifier()
iris_tree = iris_tree.fit(data_iris.data, data_iris.target)
iris_tree_text = export_text(iris_tree, feature_names=data_iris['feature_names'])
print(iris_tree_text)
```

```
|--- 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 width (cm) <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

In [26]:

```
# Preprocess play-tennis target
tennis = preprocessing.LabelEncoder()
tennis.fit(data_play_tennis_target)
list(tennis.classes_)
```

```
/Users/Tasya/anaconda3/lib/python3.6/site-packages/sklearn/preprocessi
ng/_label.py:235: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_sampl
es, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[26]:

```
['No', 'Yes']
```

In [27]:

```
# Encode play-tennis target
data_play_tennis_target_labeled = tennis.transform(data_play_tennis_target)
data_play_tennis_target_labeled
```

```
/Users/Tasya/anaconda3/lib/python3.6/site-packages/sklearn/preprocessi
ng/_label.py:268: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_sampl
es, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[27]:

```
array([0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0])
```

In [21]:

```
# Preprocess play-tennis data
data_play_tennis_data_transpose = map(list, zip(*data_play_tennis_data))
data_play_tennis_data_labeled = []
for x in data_play_tennis_data_transpose:
    tennis.fit(x)
    data_play_tennis_data_labeled.append(tennis.transform(x))
data_play_tennis_data_labeled = list(map(list, zip(*data_play_tennis_data_labeled)))
data_play_tennis_data_labeled
```

Out[21]:

```
[[2, 1, 0, 1],
 [2, 1, 0, 0],
 [0, 1, 0, 1],
 [1, 2, 0, 1],
 [1, 0, 1, 1],
 [1, 0, 1, 0],
 [0, 0, 1, 0],
 [2, 2, 0, 1],
 [2, 0, 1, 1],
 [1, 2, 1, 1],
 [2, 2, 1, 0],
 [0, 2, 0, 0],
 [0, 1, 1, 1],
 [1, 2, 0, 0]]
```

In [22]:

```
# DecisionTreeClassifier play-tennis
play_tennis_tree = tree.DecisionTreeClassifier()
play_tennis_tree = play_tennis_tree.fit(data_play_tennis_data_labeled, data_play_ter
play_tennis_tree_text = export_text(play_tennis_tree, feature_names = ['outlook', 't
print(play_tennis_tree_text)
```

```
|--- outlook <= 0.50
|   |--- class: 1
|--- outlook >  0.50
|   |--- humidity <= 0.50
|   |   |--- outlook <= 1.50
|   |   |   |--- wind <= 0.50
|   |   |   |   |--- class: 0
|   |   |   |--- wind >  0.50
|   |   |   |   |--- class: 1
|   |   |--- outlook >  1.50
|   |   |   |--- class: 0
|   |--- humidity >  0.50
|   |   |--- wind <= 0.50
|   |   |   |--- outlook <= 1.50
|   |   |   |   |--- class: 0
|   |   |   |--- outlook >  1.50
|   |   |   |   |--- class: 1
|   |   |--- wind >  0.50
|   |   |   |--- class: 1
```

In [23]:

```
from id3 import export_text
# Id3Estimator iris
estimator_iris = Id3Estimator()
estimator_iris = estimator_iris.fit(data_iris.data, data_iris.target)
estimator_iris_text = export_text(estimator_iris.tree_, feature_names=data_iris['fea
print(estimator_iris_text)
```

```
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)
```

In [25]:

```
# Id3Estimator play-tennis
estimator_play_tennis = Id3Estimator()
estimator_play_tennis = estimator_play_tennis.fit(data_play_tennis_data_labeled, dat
estimator_play_tennis_text = export_text(estimator_play_tennis.tree_, feature_names
print(estimator_play_tennis_text)
```

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

Analisis Algoritma

a. Penentuan atribut terbaik

Pada library DecisionTreeClassifier, atribut terbaik ditentukan oleh salah satu algoritma *decision tree* yaitu CART (Classification and Regression Tree) yang memanfaatkan *Gini Index*. Untuk atribut yang bernilai diskrit, *splitting* atribut akan dipilih berdasarkan subset dengan *Gini Index terkecil* (minimum).

Algoritma ID3 pada buku menentukan atribut terbaik dengan konsep *entropy* untuk menilai seberapa informatif suatu simpul. Kemudian, pemilihan atribut dilakukan berdasarkan nilai *Information Gain* terbesar.

Modul ID3Estimator sama dengan algoritma pada buku karena modul ini dibangun menggunakan algoritma ID3.

Persamaan: Melakukan pemilihan atribut berdasarkan pilihan terbaik sesuai dengan *attribute selection measures* masing-masing

b. Penanganan label dari cabang setiap nilai atribut

Algoritma ID3 pada buku menetukan label berdasarkan nilai data dari *example*. Jika semua *example* bernilai positif akan dibentuk *single-node tree* dari *root* dengan label "+" (positif). Begitu juga jika semua *example* bernilai negatif akan dibentuk *single-node tree* dari *root* dengan label "-" (negatif). Begitu seterusnya hingga ditemukan *example* kosong atau atribut sudah kosong.

Pada library DecisionTreeClassifier yang menggunakan algoritma CART, pemberian label dilakukan mengikuti aturan jumlah terbanyak, karena proses ini akan memilih label dengan peluang terbesar. Begitu juga untuk *example* dengan data yang homogen, proses *splitting* akan dihentikan dan diberi label berdasarkan nilai

terbanyak yaitu nilai satu satunya yang ada pada *example* (karena homogen).

Pada modul ID3Estimator penanganan label sama seperti algoritma ID3 pada buku.

Persamaan: Penanganan label pada data yang homogen, menghentikan proses *splitting* dan membentuk *single-node* dengan nilai yang sesuai dengan nilai pada data homogen tersebut.

c. Penentuan label jika *examples* kosong di cabang tersebut

Pada library DecisionTreeClassifier menentukan label jika *example* kosong sama dengan penanganan label seperti biasanya, seperti yang telah dijabarkan sebelumnya pada penanganan label dari cabang setiap nilai atribut. DecisionTreeClassifier akan melakukan proses pelabelan yang sama hingga simpul terakhir.

Algoritma ID3 pada buku memberi label jika ditemukan example kosong dengan membentuk simpul daun pada cabang tersebut dengan label nilai modus (paling sering muncul) dari target atribut pada *example*.

Pada modul ID3Estimator penanganan label sama seperti algoritma ID3 pada buku yaitu dengan melakukan pelabelan sesuai data terbanyak dari atribut pada *example*.

Persamaan: Penanganan label pada *example* kosong, pada dasarnya ketiganya melakukan proses yang sama yaitu memeberi label sesuai dengan data terbanyak pada atribut.

d. Penanganan atribut kontinu

Library DecisionTreeClassifier mempartisi nilai-nilai pada atribut kontinu tersebut ke interval-interval untuk menentukan kelas yang cocok untuk tiap instans, sehingga tiap simpul pada *decision tree* akan menghitung nilai dari atribut suatu instans dan mencocokkan nilai tersebut ke interval yang sudah ditentukan. Misalnya, untuk dataset jenis kelamin, dan terdapat atribut tinggi badan, maka dapat diaproksimasi interval tinggi < 160 cm akan menjadi kelas 'wanita', dan tinggi >= 160 cm masuk ke kelas 'pria'.

Id3Estimator men'diskrit'kan nilai-nilai kontinu untuk memudahkan klasifikasi. Untuk tiap nilai pada tiap atribut kontinu, akan dihitung suatu poin yang dapat membagi (*split*) rentang nilai atribut ke grup yang paling homogen pada tiap level pembagian, dengan cara menghitung *information gain* dari tiap kemungkinan *splitting* dan memilih *splitting* dengan *information gain* terbesar.

Untuk algoritma pada buku, mirip dengan Id3Estimator, juga ditentukan suatu poin *splitting* yang memiliki *information gain* terbesar, sehingga dapat mengkategorikan nilai-nilai kontinu ke grup yang sehomogen mungkin.

Persamaan: Nilai-nilai atribut dibagi dengan satu atau beberapa interval untuk proses kategorisasi.

e. Penanganan atribut dengan *missing values*

Pada DecisionTreeClassifier, *missing value* diabaikan hanya saat evaluasi (penentuan titik *splitting*), sedangkan saat klasifikasi, instans dengan *missing value* tetap dikategorikan. Instans yang berisi *missing values* biasanya dikategorikan ke dalam target dengan jumlah instans terbanyak yang sudah dikategorikan (modus), ataupun dikategorikan ke dalam kelas lain dengan perhitungan yang sudah disesuaikan.

Id3Estimator secara *default* mengabaikan instans dengan *missing values*, dan dapat berakibat pada tidak optimalnya hasil yang didapatkan seandainya instans yang diabaikan memegang peran penting (memiliki bobot yang besar) pada pembentukan pohon.

Algoritma pada buku melabelkan instans dengan *missing values* dengan jumlah nilai label terbanyak di data latih (modus).

Persamaan: Tidak ada kesamaan ketiga pendekatan dalam penanganan *missing values*. Namun, untuk menghindari pengurangan keakurasian hasil karena *missing values*, biasanya dilakukan imputasi data pada dataset, yaitu menghitung atau memperkirakan nilai yang hilang tersebut. Nilai yang biasanya dipakai adalah median (nilai tengah), mean (rata-rata) ataupun modus (nilai yang paling sering muncul).

f. *Pruning* dan parameter *confidence*

Pruning pada DecisionTreeClassifier dilakukan dengan mengubah nilai parameter `min_samples_leaf` dan `max_depth` yang mengatur jumlah sampel minimum yang diperlukan untuk melakukan *splitting* dan kedalaman dari pohon. Terdapat pula opsi lain untuk mengatur ukuran dari pohon yang dihasilkan, yaitu parameter `cpp_alpha` . Semakin besar nilai `cpp_alpha` , semakin banyak simpul yang akan di-*prune*. Penentuan nilai `cpp_alpha` yang optimal sangat penting, karena simpul yang di-*prune* diharapkan tidak terlalu banyak sehingga nilai parameter *confidence* berkurang, tetapi juga tidak terlalu sedikit agar tidak menyebabkan *overfitting*. Telah disediakan fungsi `DecisionTreeClassifier.cost_complexity_pruning_path` dari `scikit-learn` yang mengembalikan nilai `cpp_alpha` yang efektif, di mana prioritas *pruning* yang diterapkan adalah terhadap simpul dengan *link* terlemah.

Secara *default*, Id3Estimator akan terus membangun pohon sampai didapatkan tingkat parameter *confidence* setinggi mungkin dan *error* pada data latih mencapai 0%, sehingga *pruning* harus dilakukan untuk menghindari *overfitting*. Proses *pruning* yang dilakukan disebut juga sebagai *Reduced-Error Pruning*, dimulai dengan menjadikan tiap simpul bukan daun menjadi daun, lalu simpul tersebut di-*assign* dengan nilai yang paling umum dari anak-anak simpul tersebut. Bila *error* yang dihasilkan pohon baru ini pada data validasi tidak lebih buruk dari *error* yang dihasilkan pohon original, maka pohon baru ini akan menggantikan pohon yang lama. Proses ini dilakukan terus sampai tingkat *error* yang dihasilkan meningkat drastis, yang menyebabkan pengurangan tingkat akurasi pohon.

Algoritma pada buku mirip dengan Id3Estimator, yaitu mengaplikasikan *pruning* dengan membuang simpul yang tidak 'berbahaya', yaitu simpul yang dapat menurunkan tingkat akurasi dan parameter *confidence* pohon secara signifikan.

Persamaan: Ketiga pendekatan mengatur ukuran pohon yang dihasilkan dengan membuang simpul yang dianggap tidak mengganggu dan mengubah tingkat akurasi pohon, sehingga pohon yang dihasilkan tidak terlalu spesifik untuk data latih saja.