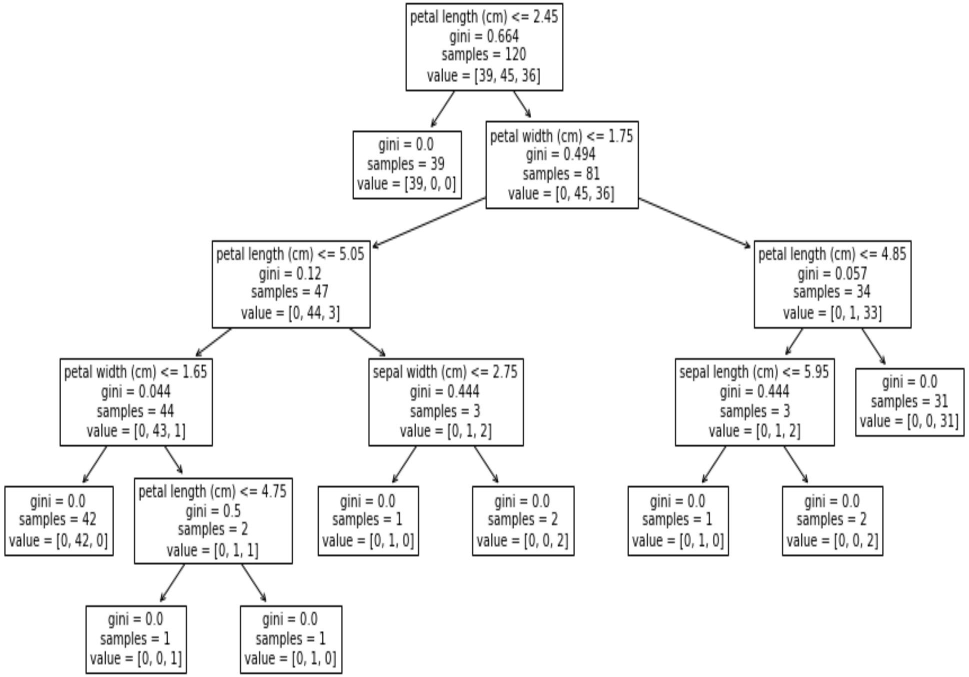
[MXML-2] Machine Learning / Decision Tree

MX-AI



2. Decision Tree

ID3/C4.5 and CART algorithm

Classification / Regression

www.youtube.com/@meanxai

[MXML-2-11] Machine Learning / 2. Decision Tree - Contents

MX-AI

1. Introduction to Decision Tree

[MXML-2-01]
- 1-1. A brief history of Decision Trees
- 1-2. Overview of ID3/C4.5 and CART
- 1-3. Comparison of ID3/C4.5 tree and CART tree
- 1-4. Making a tree, split point, majority vote

2. ID3/C4.5 Classification

[MXML-2-02]
- 2-1. Impurity, Gini index, Entropy
- 2-2. Information gain (IG)
- 2-3. Tree split: categorical feature
- 2-4. Tree split: continuous feature

[MXML-2-03]
- 2-5. Information Gain Ratio (IGR)
- 2-6. Pruning based on confidence intervals
- 2-7. Coding practice: ID3/C4.5 classification

3. ID3/C4.5 Regression

[MXML-2-04]
- 3-1. Regression tree
- 3-2. Node split using MSE, Var, SDR, CV

4. CART Classification

[MXML-2-05]
- 4-1. CART overview
- 4-2. Categorical feature
  - Ordinal and Nominal data
  - Label encoding and one-hot encoding

[MXML-2-06]
- 4-3. Information Gain (IG) and the best split point
- 4-4. Creating a tree using the best split point
- 4-5. Continuous numerical feature and the best split point

[MXML-2-07] → 4-6. Coding practice: CART classification from scratch

[MXML-2-08]
- 4-7. Feature importance
- 4-8. Optimal tree depth through cross-validataion
- 4-9. DecisionTreeClassifier of scikit learn

[MXML-2-09] → 4-10. Cost Complexity Pruning (CCP): post pruning

[MXML-2-10]
- 4-11. Coding practice: CCP implementation
- 4-12. Multi-class Classification

5. CART Regression

[MXML-2-11]
- 5-1. Regression tree algorithm
- 5-2. Coding practice: implementation of regression tree

[MXML-2-01] Machine Learning / 2. Decision Tree - History

A brief history of Decision Trees

- Before the 1980s, algorithms such as AID (1963, Morgan & Sonquist, Automatic), THAID (1972, Messenger & Mandell), and CHAID (1980, Kass) were proposed. (1st generation) AID is the first regression tree algorithm (piecewise-constant model). THAID is the first classification tree algorithm. And the CHAID algorithm is a descendant of THAID.
- In the 1980s, CART (Breiman et al. 1984, Classification and Regression Tree), ID3 (Quinlan, 1979, 1986, Induction of Decision Trees), and C4.5 appeared. (2nd generation) C4.5 is software created by ID3 proponent Quinlan with additional details. (Later extended to C5.0 and See5). CART and ID3/C4.5 were proposed independently of each other around the same time..
- Before 2000, QUEST (Loh & Shih, 1997), CRUISE (Kim & Loh, 2001, 2003), and Bayesian CART (Chipman et al. 1998, Denison et al. 1998) were proposed. (3rd generation)
- After 2000, GUIDE (Loh, 2002, 2009; Loh and Zheng, 2013; Loh et al., 2015), CTREE (Hothorn et al., 2006), MOB (Zeileis et al., 2008); Random forest (Breiman, 2001), TARGET (Fan and Gray, 2005; Gray and Fan, 2008), BART (Chipman et al., 2010), etc. were proposed. (4th generation)
- In this lecture, we will focus on ID3, C4.5, and CART, which were proposed in the 1980s.

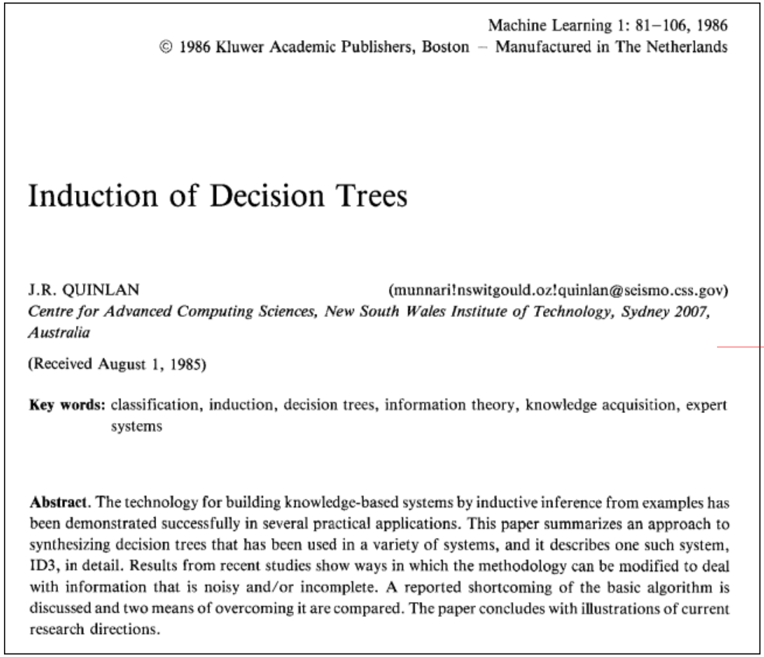
* Reference : A Brief History of Classification and Regression Trees
Wei-Yin Loh, Department of Statistics University of Wisconsin–Madiso www.stat.wisc.edu/~loh/

[MXML-2-01] Machine Learning / 2. Decision Tree - History

MX-AI

Brief history of ID3, C4.5 and CART

ID3 (Iterative Dichotomiser 3)



Machine Learning 1: 81–106, 1986
© 1986 Kluwer Academic Publishers, Boston – Manufactured in The Netherlands

Induction of Decision Trees

J.R. QUINLAN (munnari!nswitgould.oz!quinlan@seismo.css.gov)
Centre for Advanced Computing Sciences, New South Wales Institute of Technology, Sydney 2007, Australia

(Received August 1, 1985)

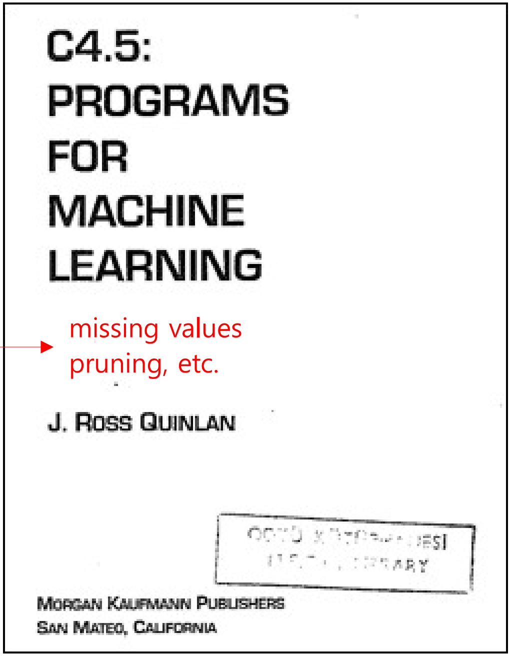
Key words: classification, induction, decision trees, information theory, knowledge acquisition, expert systems

Abstract. The technology for building knowledge-based systems by inductive inference from examples has been demonstrated successfully in several practical applications. This paper summarizes an approach to synthesizing decision trees that has been used in a variety of systems, and it describes one such system, ID3, in detail. Results from recent studies show ways in which the methodology can be modified to deal with information that is noisy and/or incomplete. A reported shortcoming of the basic algorithm is discussed and two means of overcoming it are compared. The paper concludes with illustrations of current research directions.

J.R. QUINLAN
Centre for Advanced Computing Sciences, New South Wales
Institute of Technology, Sydney 2007, Australia

extend

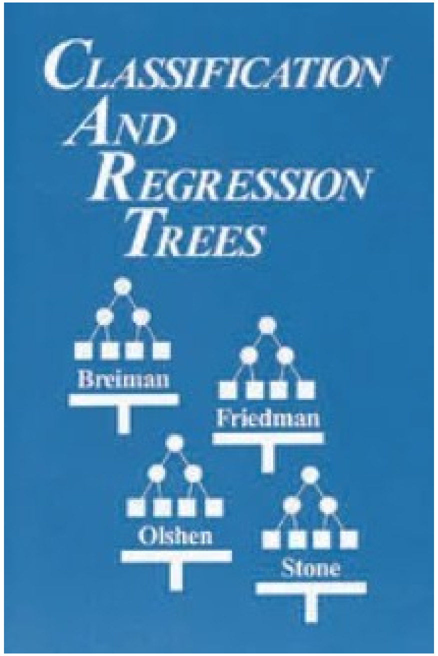
C4.5 is an extension of Quinlan's earlier ID3 algorithm.



missing values
pruning, etc.

Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

CART



By Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen
Copyright Year 1984, 1st Edition

* CART Decision Tree is more widely used than ID3, and is used as the base model for ensembles such as Random Forest, GBM, xGBoost, LGBM, etc.

[MXML-2-01] Machine Learning / 2. Decision Tree – ID3/C4.5 vs. CART

MX-AI

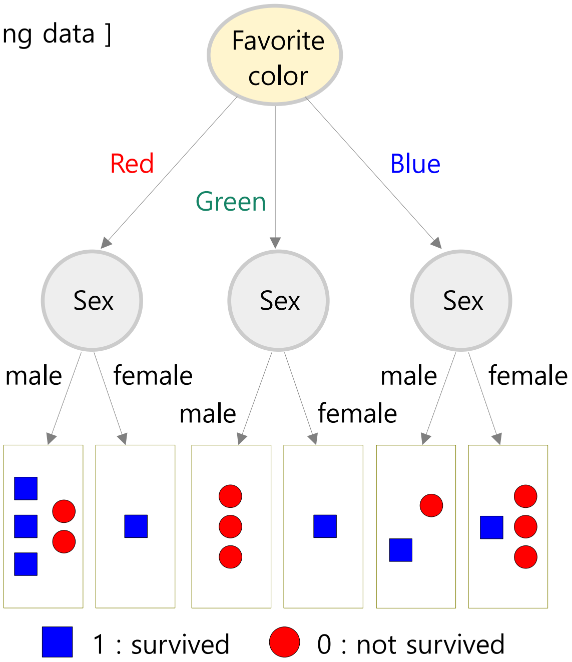
■ ID3/C4.5 tree vs. CART tree

[training data]

feature			target
Favorite color	Sex	Age	Survived
Red	male	24	0
Blue	female	38	1
Red	male	32	1
Blue	male	37	0
Green	male	24	0
Green	male	21	0
Red	female	58	1
Green	female	36	1
Blue	female	14	0
Red	male	36	1
Red	male	4	1
Blue	male	26	1
Blue	female	10	0
Blue	female	41	0
Red	male	50	0
Green	male	34	0

Part of the Titanic dataset.
“Favorite color” is added for test purpose.

1. ID3/C4.5 Tree



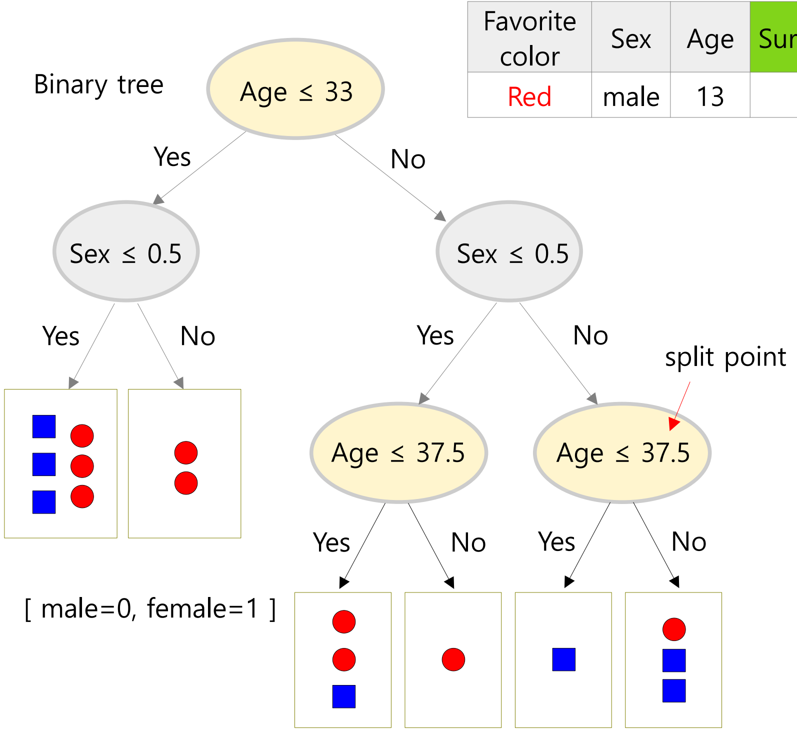
How do I handle the "age" feature?
“Age” is a continuous feature.
Which feature should I use first for tree split?

*** We'll learn more about how to deal with these issues later.**

2. CART Tree

[test data]

Favorite color	Sex	Age	Survived
Red	male	13	?



How do I handle the “Favorite color” feature?
“Favorite color” is a nominal categoricy feature.
How to find the split points: 33, 0.5, 37.5, etc?

[MXML-2-01] Machine Learning / 2. Decision Tree - Introduction

MX-AI

Making a tree, Split point, majority vote

- When arranging the following training data in a two-dimensional space, let's consider four split points: a, b, c on the x1 axis, and a' on the x2 axis.
- Using these four split points, we can build a tree like the one on the right.
- Our goal is to estimate the class y of the test data using the tree.
- When test data A is displayed in space, it is located in the upper right square area. The class of the test data is then estimated to be 1 by majority vote.
- Similarly, when the test data A is entered into the tree, it is located in the first leaf node, which is also estimated to be 1 by majority vote.
- There are many candidates for the splitting point. Therefore, the main topic of Decision Tree is finding the optimal split point.
- Simply put, the Gini index or Entropy is used to find the optimal splitting point. We'll cover this in more detail later.

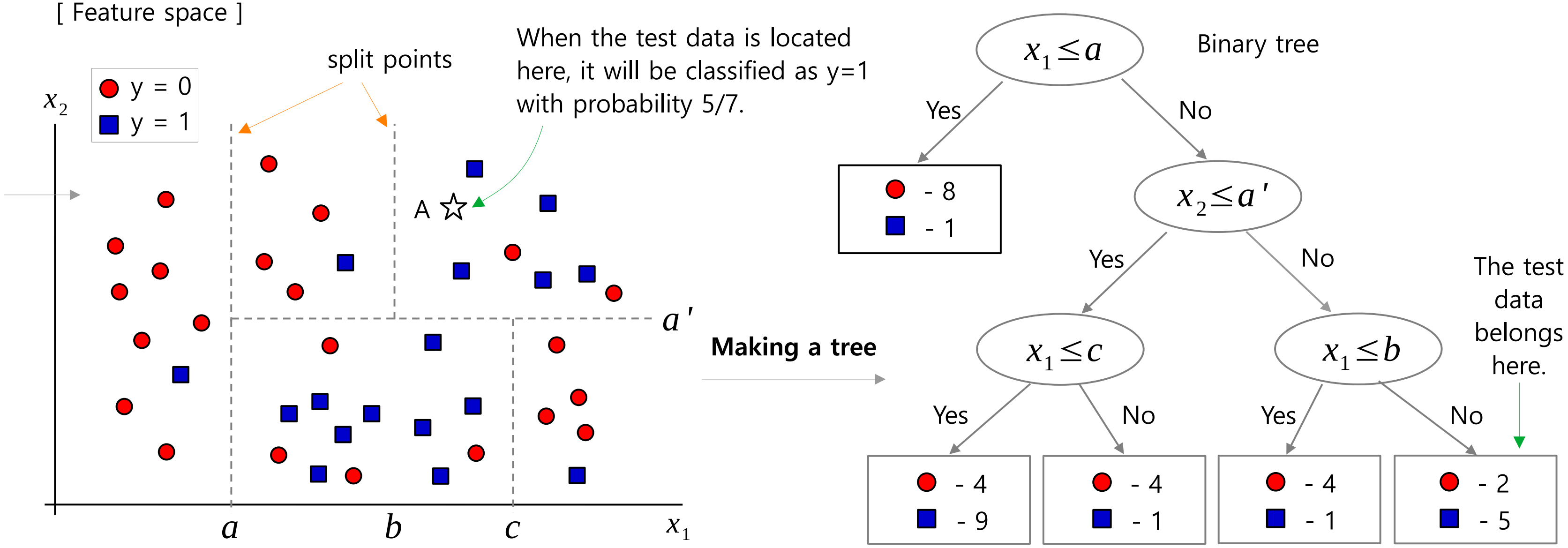
[training data]

feature		class
x_1	x_2	y
0.16	0.01	0
-0.09	-0.03	1
0.06	-0.08	0
0.58	-0.1	1
0.46	-0.07	0
0.23	0.54	1
0.55	0.07	1
:	:	:

[test data]

	feature		target
	x_1	x_2	y
A	0.5	0.4	?
B	0.19	-0.21	?

[Feature space]



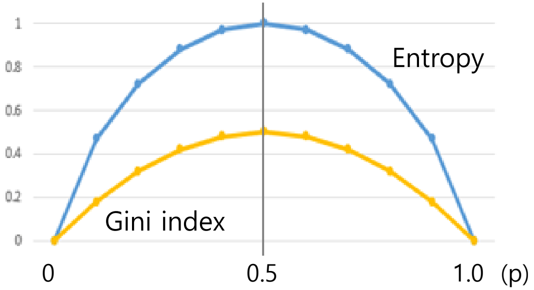
When the test data is located here, it will be classified as y=1 with probability 5/7.

The test data belongs here.

[MXML-2-02] Machine Learning / 2. Decision Tree - Impurity

Impurity, Entropy and Gini index

- When there are 4 nodes (A, B, C, and D) as follows, the impurity of each node can be measured by the entropy or Gini index. Impurity is a measure of how much things with different properties are mixed within a node.
- Node A has the highest purity (lowest impurity) because it contains all of the same colors. Node C has the lowest purity (highest impurities) because it is a half-and-half mixture.
- Impurity can be used to express the amount of information held by each node.



Maximum impurity at p=0.5.
(for c = 2)

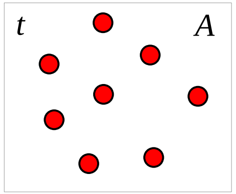
- Entropy

$$H(t)=-\sum_{i=1}^{c} p(i|t)\cdot\log_2 p(i|t)$$

- Gini index

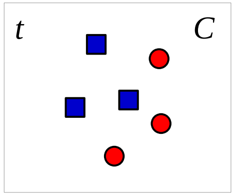
(c : the number of class)

$$G(t)=\sum_{i=1}^{c} p(i|t)\cdot(1-p(i|t)) = 1-\sum_{i=1}^{c} p(i|t)^2$$



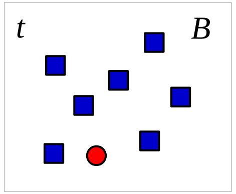
$$H(t)=-(\frac{8}{8}\times\log_2\frac{8}{8} + \frac{0}{8}\times\log_2\frac{0}{8})=0$$

$$G(t)=1-((\frac{8}{8})^2 + (\frac{0}{8})^2)=0$$



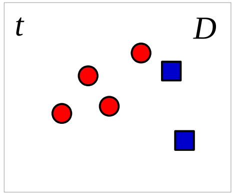
$$H(t)=-(\frac{3}{6}\times\log_2\frac{3}{6} + \frac{3}{6}\times\log_2\frac{3}{6})=1.0$$

$$G(t)=1-((\frac{3}{6})^2 + (\frac{3}{6})^2)=0.5$$



$$H(t)=-(\frac{1}{8}\times\log_2\frac{1}{8} + \frac{7}{8}\times\log_2\frac{7}{8})=0.544$$

$$G(t)=1-((\frac{1}{8})^2 + (\frac{7}{8})^2)=0.219$$



$$H(t)=-(\frac{4}{6}\times\log_2\frac{4}{6} + \frac{2}{6}\times\log_2\frac{2}{6})=0.918$$

$$G(t)=1-((\frac{4}{6})^2 + (\frac{2}{6})^2)=0.444$$

[MXML-2-02] Machine Learning / 2. Decision Tree – Information Gain

MX-AI

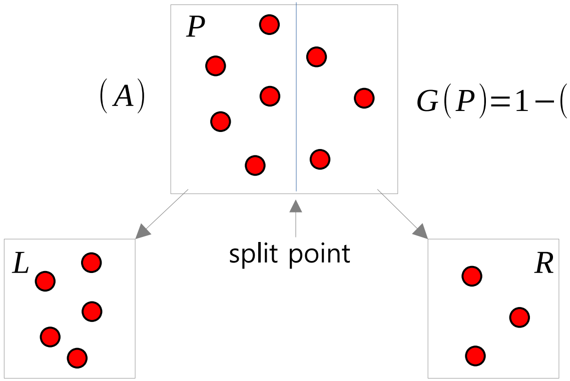
Information Gain (IG)

- The information gain (IG) can be measured through the change in impurities due to tree node splitting. Lowering impurity when splitting nodes is a good thing and provides informational benefits. Information gain is measured as the difference in weighted average impurities before and after splitting.
- For node A, the impurity before splitting is already 0, so there is no benefit from splitting (no need to split). For node B, splitting lowers impurities and produces an information gain of 0.278 (need to split).
- Decision tree algorithms build a tree by finding the split points that provide the greatest information gain.

- Gini index: $G(t)=1-\sum_{i=1}^{c} p(i|t)^2$

- Information gain: $IG=G(P)-\frac{N_L}{N}\times G(L)-\frac{N_R}{N}\times G(R)$

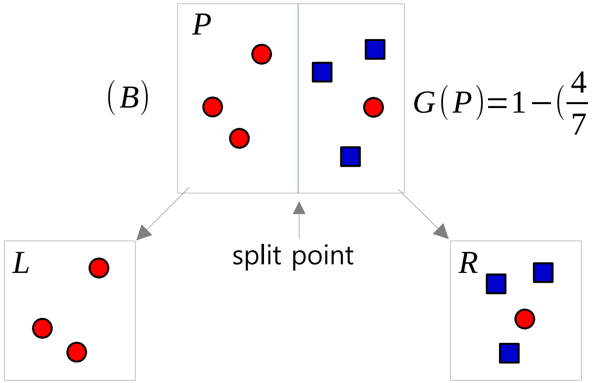
$$IG=0-\frac{5}{8}\times 0-\frac{3}{8}\times 0=0$$



$$G(P)=1-(\frac{8}{8})^2-(\frac{0}{8})^2=0$$

$$G(L)=1-(\frac{5}{5})^2-(\frac{0}{5})^2=0 \qquad G(R)=1-(\frac{3}{3})^2-(\frac{0}{3})^2=0$$

$$IG=0.49-\frac{3}{7}\times 0-\frac{4}{7}\times 0.375=0.278\ >\ 0$$



$$G(P)=1-(\frac{4}{7})^2-(\frac{3}{7})^2=0.490$$

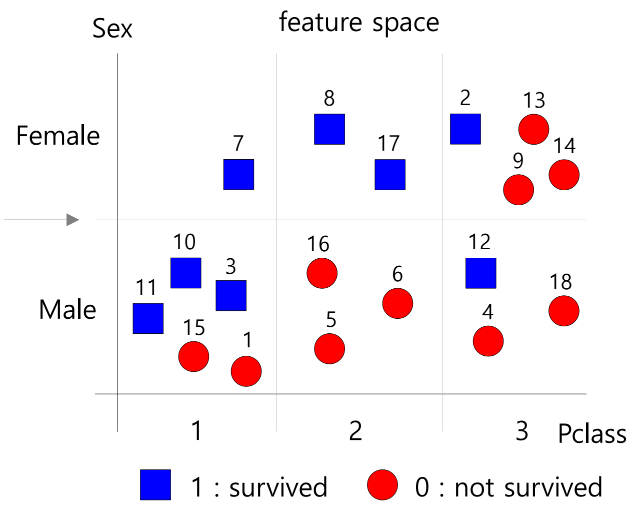
$$G(L)=1-(\frac{3}{3})^2-(\frac{0}{3})^2=0 \qquad G(R)=1-(\frac{3}{4})^2-(\frac{1}{4})^2=0.375$$

Information Gain (IG) for ID3 (and C4.5)

- Calculate IG using entropy or Gini index and find the best feature to use for node splitting.

	feature		target
No	Pclass	Sex	Survived
1	1	male	0
2	3	female	1
3	1	male	1
4	3	male	0
5	2	male	0
6	2	male	0
7	1	female	1
8	2	female	1
9	3	female	0
10	1	male	1
11	1	male	1
12	3	male	1
13	3	female	0
14	3	female	0
15	1	male	0
16	2	male	0
17	2	female	1
18	3	male	0

[Part of the Titanic dataset]



1. Entropy of root node

$$H(root) = -\sum_{i=1}^{c} p(i|t)\log_2 p(i|t)$$

$$= -\frac{8}{18}\times\log_2\frac{8}{18} - \frac{10}{18}\times\log_2\frac{10}{18} = 0.991$$

2. Entropy of Pclass node

$$H(Pclass=1) = -\frac{4}{6}\times\log_2\frac{4}{6} - \frac{2}{6}\times\log_2\frac{2}{6} = 0.918$$

$$H(Pclass=2) = -\frac{2}{5}\times\log_2\frac{2}{5} - \frac{3}{5}\times\log_2\frac{3}{5} = 0.971$$

$$H(Pclass=3) = -\frac{2}{7}\times\log_2\frac{2}{7} - \frac{5}{7}\times\log_2\frac{5}{7} = 0.863$$

$$H(Pclass) = \frac{6}{18}\times 0.918 + \frac{5}{18}\times 0.971 + \frac{7}{18}\times 0.863 = 0.911$$

3. Entropy of Sex node

$$H(Sex=male) = -\frac{4}{11}\times\log_2\frac{4}{11} - \frac{7}{11}\times\log_2\frac{7}{11} = 0.946$$

$$H(Sex=female) = -\frac{4}{7}\times\log_2\frac{4}{7} - \frac{3}{7}\times\log_2\frac{3}{7} = 0.985$$

$$H(Sex) = \frac{11}{18}\times 0.946 + \frac{7}{18}\times 0.985 = 0.961$$

4. Information Gain

Select Pclass as the initial split.

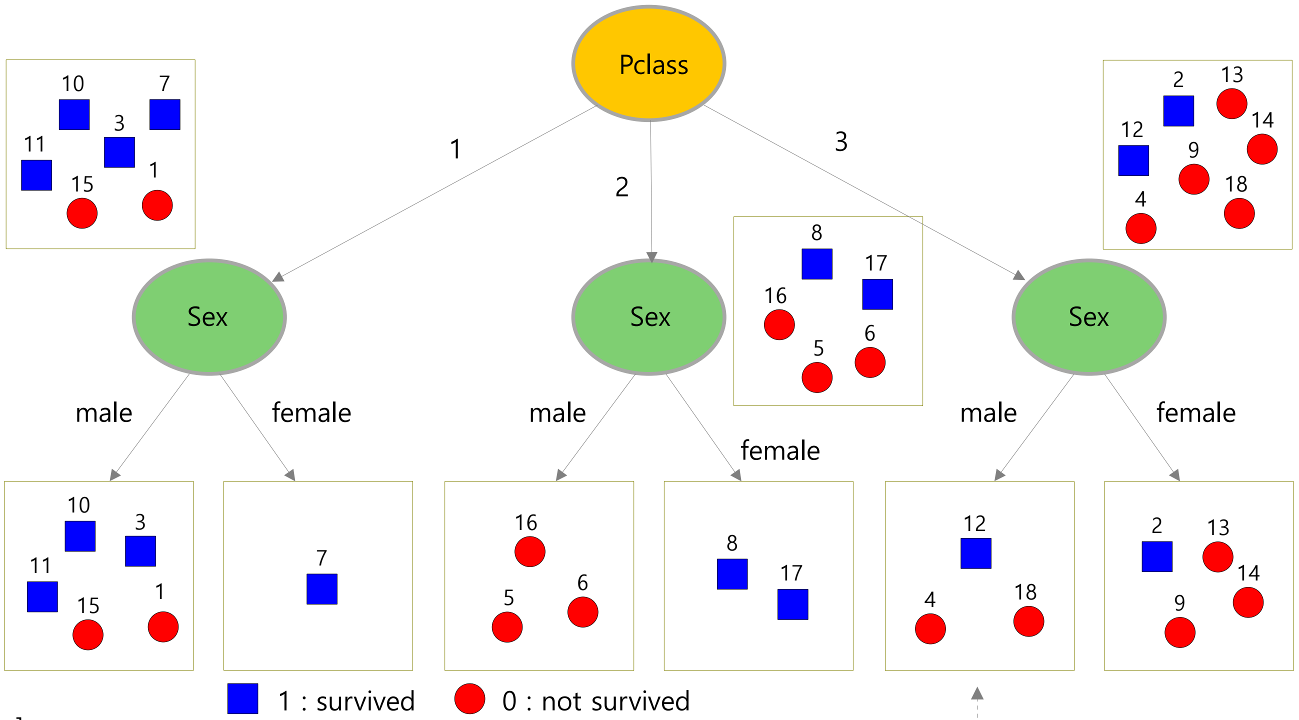
$$IG(Pclass) = H(root) - H(Pclass) = 0.991 - 0.911 = 0.08$$

$$IG(Sex) = H(root) - H(Sex) = 0.991 - 0.961 = 0.03$$

■ Split tree: Categorical feature

- Pclass is selected as the initial splitting condition. ID3 branches to all categories of Pclass simultaneously. (CART is a binary tree, but ID3 is not.)

	feature		target
No	Pclass	Sex	Survived
1	1	male	0
2	3	female	1
3	1	male	1
4	3	male	0
5	2	male	0
6	2	male	0
7	1	female	1
8	2	female	1
9	3	female	0
10	1	male	1
11	1	male	1
12	3	male	1
13	3	female	0
14	3	female	0
15	1	male	0
16	2	male	0
17	2	female	1
18	3	male	0



[test data]

No	Pclass	Sex	Survived
1	3	male	?

← When this data is fed into this tree, it is estimated to have a "2/3 chance of non-survival."

Split tree: Continuous feature

	feature			target	
No	Pclass	Sex	Age	Survived	Age-C
1	1	male	24	0	2
2	3	female	38	1	4
3	1	male	32	1	3
4	3	male	37	0	4
5	2	male	24	0	2
6	2	male	21	0	1
7	1	female	58	1	4
8	2	female	36	1	3
9	3	female	14	0	1
10	1	male	36	1	3
11	1	male	4	1	1
12	3	male	26	1	2
13	3	female	10	0	1
14	3	female	41	0	4
15	1	male	50	0	4
16	2	male	34	0	3
17	2	female	29	1	2
18	3	male	25	0	2

```
import numpy as np
p = [25, 50, 75]
q1, m, q3 = np.percentile(나이, p)
# → [24.0, 30.5, 36.75]

for i in x:
    if i < q1:
        print('1')
    elif i >= q1 and i < m:
        print('2')
    elif i >= m and i < q3:
        print('3')
    else:
        print('4')
```

With Age, you get this much information. It is better to use Age since Pclass's IG=0.08.

[Part of the Titanic dataset]

- Entropy and IG of Age

$$H(Age<24.0)=-\frac{3}{4}\times\log_2\frac{3}{4}-\frac{1}{4}\times\log_2\frac{1}{4}=0.811$$

$$H(24.0\leq Age<30.5)=-\frac{3}{5}\times\log_2\frac{3}{5}-\frac{2}{5}\times\log_2\frac{2}{5}=0.971$$

$$H(30.5\leq Age<36.75)=-\frac{3}{4}\times\log_2\frac{3}{4}-\frac{1}{4}\times\log_2\frac{1}{4}=0.811$$

$$H(Age\geq 36.75)=-\frac{3}{5}\times\log_2\frac{3}{5}-\frac{2}{5}\times\log_2\frac{2}{5}=0.971$$

$$H(Age)=\frac{4}{18}\times 0.811+\frac{5}{18}\times 0.971+$$

$$\frac{4}{18}\times 0.811+\frac{5}{18}\times 0.971=0.900$$

$$IG(Age)=H(root)-H(Age)=0.991-0.900=0.091$$

Information Gain Ratio (IGR)

- IG tends to increase as the number of categories increases. Pclass has 3 categories, "Sex" has 2 and "Age" has 4. In this case, there is a problem that IG of 'Age' is highly evaluated. To solve this problem, it is necessary to impose some kind of penalty on features with a large number of categories. Information Gain Ratio (IGR) can solve this problem. When IGR is applied, Pclass is selected as the initial splitting condition for the tree.

No	Pclass	Sex	Age	Survived
1	1	male	2	0
2	3	female	4	1
3	1	male	3	1
4	3	male	4	0
5	2	male	2	0
6	2	male	1	0
7	1	female	4	1
8	2	female	3	1
9	3	female	1	0
10	1	male	3	1
11	1	male	1	1
12	3	male	2	1
13	3	female	1	0
14	3	female	4	0
15	1	male	4	0
…	…	…	…	…

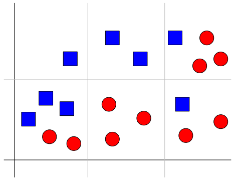
[Part of the Titanic dataset]

- Entropy

$$H(root) = 0.991 \qquad H(Pclass) = 0.911$$
$$H(Sex) = 0.961 \qquad H(Age) = 0.900$$

- Information Gain (IG)

$$IG(Pclass) = H(root) - H(Pclass) = 0.08$$
$$IG(Sex) = H(root) - H(Sex) = 0.03$$
$$IG(Age) = H(root) - H(Age) = 0.091$$



- Split Information (SI)

$$SI(Pclass) = -\frac{6}{18}\times\log_2\frac{6}{18} - \frac{5}{18}\times\log_2\frac{5}{18} - \frac{7}{18}\times\log_2\frac{7}{18} = 1.57$$
$$SI(Sex) = -\frac{11}{18}\times\log_2\frac{11}{18} - \frac{7}{18}\times\log_2\frac{7}{18} = 0.96$$
$$SI(Age) = -\frac{4}{18}\times\log_2\frac{4}{18} - \frac{5}{18}\times\log_2\frac{5}{18} - \frac{4}{18}\times\log_2\frac{4}{18} - \frac{5}{18}\times\log_2\frac{5}{18} = 1.99$$

Nodes with many splits have large SI values.

- Information Gain Ratio

This is the biggest.

$$IGR(Pclass) = \frac{IG(Pclass)}{SI(Pclass)} = \frac{0.08}{1.57} = 0.051$$
$$IGR(Sex) = \frac{IG(Sex)}{SI(Sex)} = \frac{0.03}{0.96} = 0.031$$
$$IGR(Age) = \frac{IG(Age)}{SI(Age)} = \frac{0.091}{1.99} = 0.046$$

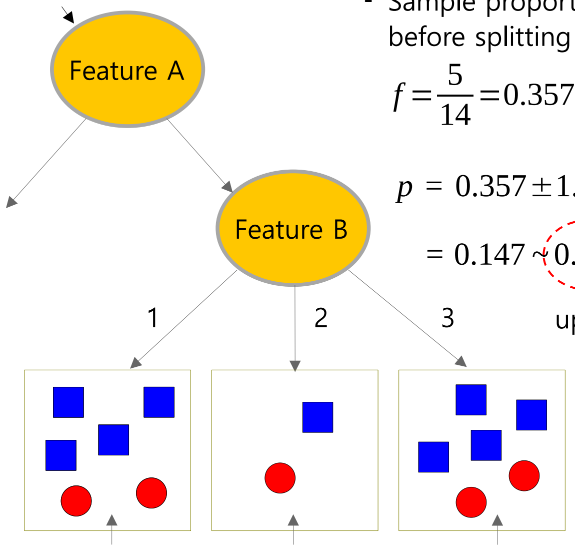
IG tends to increase with more splits. SI is intended to correct this.

$$SI = -\sum_{i=1}^{n} \frac{N(t_i)}{N(t)} \log_2 \frac{N(t_i)}{N(t)}$$

(n: the number of category)

Pruning based on confidence intervals

- ID3 splits the tree until all classes of all features in the data are used. As the tree gets deeper, overfitting is likely to occur, so pruning is necessary.



- Sample proportion (f) and population proportion (p) before splitting feature B

$$f = \frac{5}{14} = 0.357$$ ← misclassification rate

$$p = 0.357 \pm 1.64\sqrt{\frac{0.357(1-0.357)}{14}}$$

$$= 0.147 \sim 0.567$$

upper bound

$$f = \frac{2}{6} = 0.333 \qquad f = \frac{1}{2} = 0.5 \qquad f = \frac{2}{6} = 0.333$$ ← misclassification rate for each node

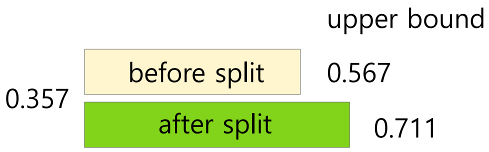
$$p_{up} = 0.649 \qquad p_{up} = 1.080 \qquad p_{up} = 0.649$$ ← upper bound

$$p_m = 0.649 \times \frac{6}{14} + 1.080 \times \frac{2}{14} + 0.649 \times \frac{6}{14} = 0.711$$ ← weighted average

$$f_m = \frac{2}{6} \cdot \frac{6}{14} + \frac{1}{2} \cdot \frac{2}{14} + \frac{2}{6} \cdot \frac{6}{14} = \frac{5}{14} = 0.357$$ ← weighted average

- population proportion (p)

$$p = f \pm z\sqrt{\frac{f(1-f)}{n}} \qquad z = 1.64\,(90\%\ CI)$$

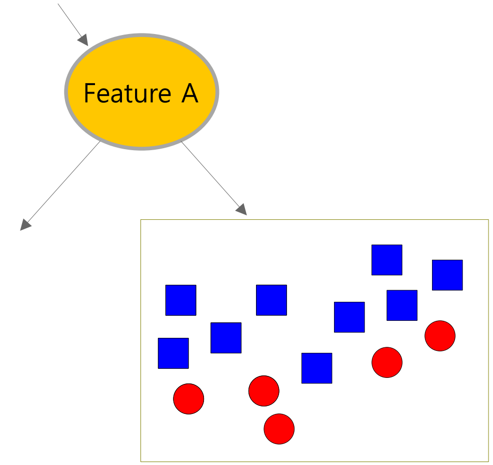


- The smaller the upper bound, the better the performance.

- Pruned
- **It is better not to split feature B.**

pruning

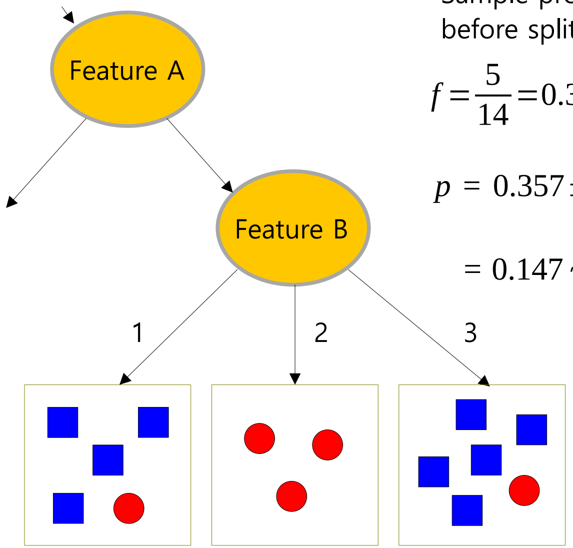
$$0.567 < 0.711$$



reference : http://www.cs.bc.edu/~alvarez/ML/statPruning.html

Pruning based on confidence intervals

- If feature B is split as shown below, it is better to split rather than prune.



$f=\frac{1}{5}=0.2$ $\quad$ $f=\frac{0}{3}=0$ $\quad$ $f=\frac{1}{6}=0.167$ ← misclassification rate for each node. This is lower than the misclassification rate on the previous page.

$p_{up}=0.493$ $\quad$ $p_{up}=0$ $\quad$ $p_{up}=0.416$ ← upper bound

$p_m=0.493\times\frac{5}{14}+0\times\frac{3}{14}+0.416\times\frac{6}{14}=0.354$ ← weighted average

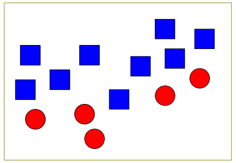
$f_m=0.2\times\frac{5}{14}+0\times\frac{3}{14}+0.167\times\frac{6}{14}=0.143$ ← weighted average

- Sample proportion (f) and population proportion (p) before splitting feature B

$f=\frac{5}{14}=0.357$ ← misclassification rate

$$p = 0.357 \pm 1.64\sqrt{\frac{0.357(1-0.357)}{14}}$$

$$= 0.147 \sim 0.567$$



before split

- population proportion (p)

$$p = f \pm z\sqrt{\frac{f(1-f)}{n}} \qquad z=1.64(90\%\ CI)$$

		upper bound
0.357	before split	0.567
0.143	after split	0.354

- The smaller the upper bound, the better the performance.

No prune

$0.354 < 0.567$

- No prune
- **It is better to split feature B.**

reference : http://www.cs.bc.edu/~alvarez/ML/statPruning.html

■ Coding practice: ID3/C4.5 classification - Using only some features of the Titanic dataset

```
# 1.ID3(titanic_part).py
# ID3/C4.5 decision tree test code
# CART is widely used than ID3/C4.5. Sklearn  supports CART.
#
# I used the package below to test ID3/C4.5.
# https://github.com/svaante/decision-tree-id3
# pip install decision-tree-id3
# pip install pydot
# pip install graphviz
# sudo apt install graphviz
# -----------------------------------------------------------

# "from sklearn.externals import six" is used for id3, but "six"
# is missing in the sklearn.externals, resulting in the following
# error: cannot import name "six" from 'sklearn.externals'
# Add following to prevent errors.
import six
import sys; sys.modules['sklearn.externals.six'] = six
import pandas as pd
from id3 import Id3Estimator
from id3 import export_graphviz
import pydot
from sklearn.model_selection import train_test_split

# Use just 3 features in the Titanic dataset:
feat_names = ['Pclass', 'Sex', 'Age']
df = pd.read_csv('data/titanic.csv')[feat_names + ['Survived']]
df = df.dropna().reset_index()
df.info()
```

```
# Separate the data into feature and target.
x_data = df[feat_names].copy()
y_data = df['Survived']

# Convert string (Sex) to number. female = 0, male = 1
x_data['Sex'] = x_data['Sex'].map({'female':0, 'male':1})

# Convert real numbers (Age) to 4 categories.
x_data['Age'] = pd.qcut(x_data['Age'], 4, labels=False)

# Split the data into training and test data.
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data)

# Build ID3/C4.5 decision tree.
estimator = Id3Estimator(gain_ratio=True, prune=True)
estimator = estimator.fit(x_train, y_train, check_input=False)

# Evaluate performance with test data.
y_pred = estimator.predict(x_test)
acc = (y_pred == y_test).mean()
print('\nAccuracy of test data = {:.4f}'.format(acc))

# Evaluate performance with training data.
y_pred = estimator.predict(x_train)
acc = (y_pred == y_train).mean()
print('Accuracy of train data = {:.4f}\n'.format(acc))
```

[MXML-2-03] Machine Learning / 2. Decision Tree – ID3/C4.5 (Classification)

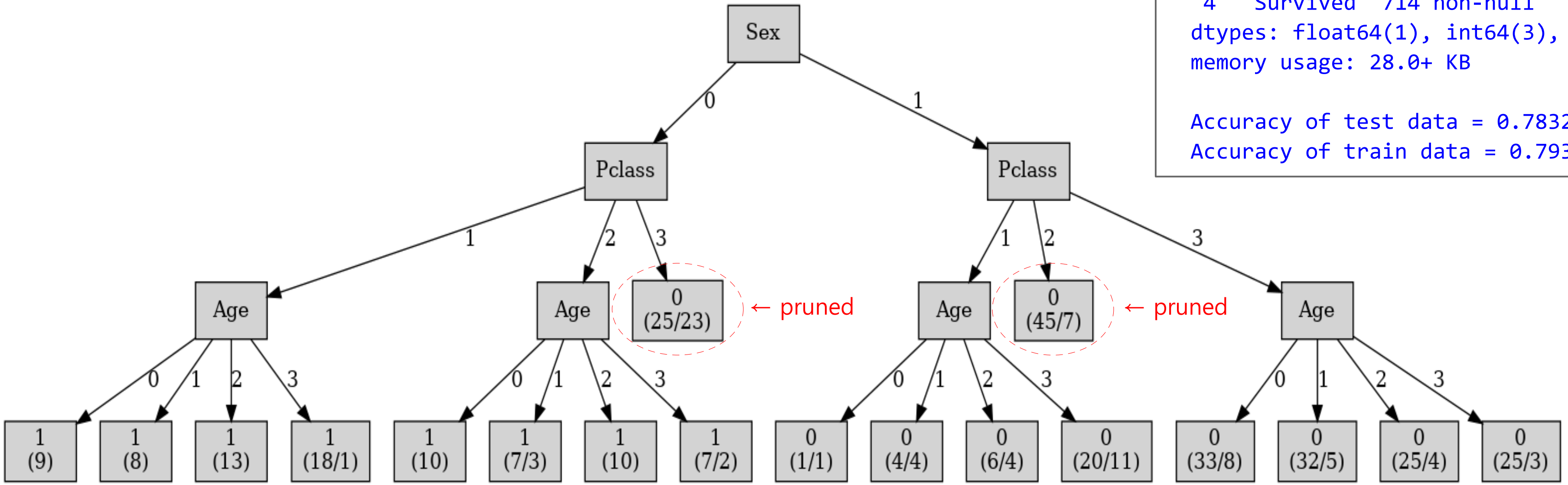
■ Coding practice: ID3/C4.5 classification - Using only some features of the Titanic dataset

```
# Visualize the tree result
tree = export_graphviz(estimator.tree_, 'id3_tree.dot', feat_names)
(graph,) = pydot.graph_from_dot_file('id3_tree.dot')
graph.write_png('id3_tree.png')
!nomacs 'id3_tree.png'   # Check the tree image with the image viewer.
```

■ Code execution result

```
RangeIndex: 714 entries, 0 to 713
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   index     714 non-null    int64
 1   Pclass    714 non-null    int64
 2   Sex       714 non-null    object
 3   Age       714 non-null    float64
 4   Survived  714 non-null    int64
dtypes: float64(1), int64(3), object(1)
memory usage: 28.0+ KB

Accuracy of test data = 0.7832
Accuracy of train data = 0.7933
```

■ Tree (id3_tree.png)



[MXML-2-04] Machine Learning / 2. Decision Tree – ID3/C4.5 (Regression)

MX-AI

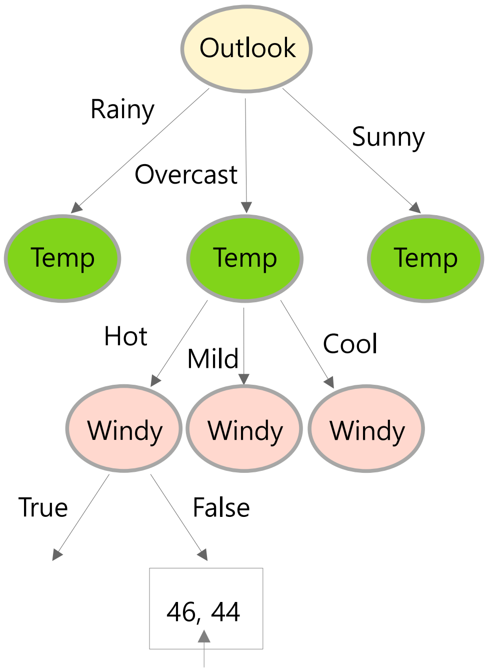
ID3/C4.5 Regression Tree

- In a regression tree, target value of test data is estimated as the mean value of the leaf node to which the test data belongs.

		feature		target value
No	Outlook	Temp	Windy	Hours Played
1	Rainy	Hot	False	25
2	Rainy	Hot	True	30
3	Overcast	Hot	False	46
4	Sunny	Mild	False	45
5	Sunny	Cool	False	52
6	Sunny	Cool	True	23
7	Overcast	Cool	True	43
8	Rainy	Mild	False	35
9	Rainy	Cool	False	38
10	Sunny	Mild	False	46
11	Rainy	Mild	True	48
12	Overcast	Mild	True	52
13	Overcast	Hot	False	44
14	Sunny	Mild	True	30

⋮

Regression Tree



If the test data belongs to this leaf node, the target value of the test data is estimated to be 45, which is the mean value of the leaf node.

	Classification	Regression
Performance	Misclassification rate Accuracy	Mean squared error (MSE) R2 score
Estimation	Majority vote	Mean
Node split	Impurity, Entropy, Gini index, IG, IGR	MSE, VAR, Standard Deviation Reduction (SDR)
Pruning	post-pruning by confidence intervals (CI)	pre-pruning early stopping by coefficient of variation (CV)

Regression Tree: MSE, Var, CV, SDR

- Regression tree uses mean squared error (MSE) instead of entropy or Gini index. The error is calculated using the difference between the actual value and the average value of the leaf nodes. Then the MSE is equal to variance (Var).
- A regression tree splits nodes in a direction that reduces variance (or standard deviation). Variance Reduction or Standard Deviation Reduction : SDR.

	feature			target value
No	Outlook	Temp	Windy	Hours Played
1	Rainy	Hot	False	25
2	Rainy	Hot	True	30
3	Overcast	Hot	False	46
4	Sunny	Mild	False	45
5	Sunny	Cool	False	52
6	Sunny	Cool	True	23
7	Overcast	Cool	True	43
8	Rainy	Mild	False	35
9	Rainy	Cool	False	38
10	Sunny	Mild	False	46
11	Rainy	Mild	True	48
12	Overcast	Mild	True	52
13	Overcast	Hot	False	44
14	Sunny	Mild	True	30

$$MSE = Var = \frac{1}{N}\sum_{i=1}^{N}\left(\overset{\text{actual value}}{x_i} - \overset{\text{estimated value}}{\bar{x}}\right)^2 \qquad S = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}$$

- Average, standard deviation, coefficient of variation (CV) of Root node

Average : $m(root) = \dfrac{25 + 30 + \dots + 30}{14} = 39.79$

Standard deviation : $S(root) = \sqrt{\frac{1}{14}\left((26-39.79)^2 + (30-39.79)^2 + \dots + (30-39.79)^2\right)} = 9.32$

Coefficient of variation : $CV(root) = \dfrac{S}{m} = \dfrac{9.32}{39.79} = 0.234$

- CV is used for early stopping.

- SDR = Standard deviation of root node – standard deviation of leaf node

■ Regression Tree (SDR calculation)

	feature			target value
No	Outlook	Temp	Windy	Hours Played
1	Rainy	Hot	False	25
2	Rainy	Hot	True	30
3	Overcast	Hot	False	46
4	Sunny	Mild	False	45
5	Sunny	Cool	False	52
6	Sunny	Cool	True	23
7	Overcast	Cool	True	43
8	Rainy	Mild	False	35
9	Rainy	Cool	False	38
10	Sunny	Mild	False	46
11	Rainy	Mild	True	48
12	Overcast	Mild	True	52
13	Overcast	Hot	False	44
14	Sunny	Mild	True	30

- Calculating the SDR of each feature, the Outlook feature has the highest SDR at 1.66. Therefore, the root node is split into Outlook.

▪ SDR of Outlook

Outlook = Rainy → Hours Played = [25, 30, 35, 38, 48] → S = 7.78
Outlook = Overcast → Hours Played = [46, 43, 52, 44] → S = 3.49
Outlook = Sunny → Hours Played = [45, 52, 23, 46, 30] → S = 10.87

$$S(Outlook)=\frac{5}{14}\times 7.78+\frac{4}{14}\times 3.49+\frac{5}{14}\times 10.87=7.66$$ - weighted average

$$SDR(Outlook)=S(root)-S(Outlook)=9.32-7.66=\boxed{1.66}$$ ← highest

▪ SDR of Temp

Temp = Hot → Hours Played = [25, 30, 46, 44] → S = 8.95
Temp = Mild → Hours Played = [45, 35, 46, 48, 52, 30] → S = 7.65
Temp = Cool → Hours Played = [52, 23, 43, 38] → S = 10.51

$$S(Temp)=\frac{4}{14}\times 8.95+\frac{6}{14}\times 7.65+\frac{4}{14}\times 10.51=8.84$$ - weighted average

$$SDR(Temp)=S(root)-S(Temp)=9.32-8.84=\boxed{0.48}$$

▪ SDR of Windy

Windy = False → Hours Played = [25, 46, 45, 52, 35, 38, 46, 44] → S = 7.87
Windy = True → Hours Played = [30, 23, 43, 48, 52, 30] → S = 10.59

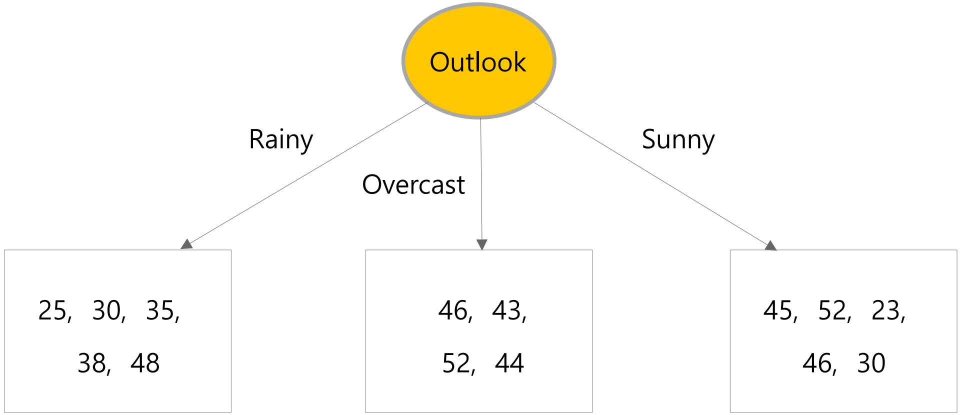
$$S(Windy)=\frac{8}{14}\times 7.87+\frac{6}{14}\times 10.59=9.04$$ - weighted average

$$SDR(Windy)=S(root)-S(Windy)=9.32-7.66=\boxed{0.28}$$

■ Regression Tree : Early stopping by CV

No	Outlook	Temp	Windy	Hours Played
1	Rainy	Hot	False	25
2	Rainy	Hot	True	30
3	Overcast	Hot	False	46
4	Sunny	Mild	False	45
5	Sunny	Cool	False	52
6	Sunny	Cool	True	23
7	Overcast	Cool	True	43
8	Rainy	Mild	False	35
9	Rainy	Cool	False	38
10	Sunny	Mild	False	46
11	Rainy	Mild	True	48
12	Overcast	Mild	True	52
13	Overcast	Hot	False	44
14	Sunny	Mild	True	30

⋮


Outlook

Rainy

Overcast

Sunny

25, 30, 35, 38, 48

46, 43, 52, 44

45, 52, 23, 46, 30

standard deviation → $s = 7.78$ | $s = 3.49$ | $s = 10.87$

mean → $m = 35.2$ | $m = 46.25$ | $m = 39.2$

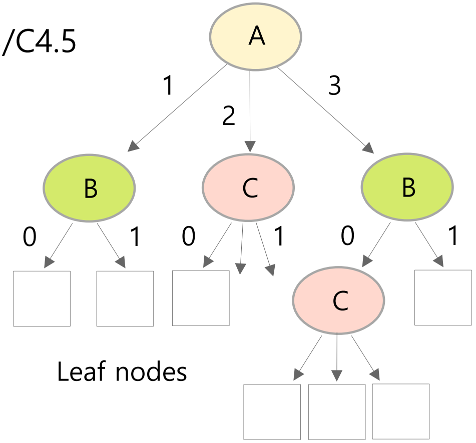
coefficient of variation → $CV = 0.22$ | $CV = 0.08$ | $CV = 0.28$

- Set threshold for CV. (hyper-parameter). Ex : 0.1 (10%). This is an early stopping condition to prevent overfitting. Cross-validation determines the optimal threshold.
- The CV for Overcast is 0.08, which is below the threshold. Overcast node is no longer split.
- For "Rainy" and "Sunny", the CV is greater than the threshold, so we recalculate SDR of each node and repeat the above process to split the nodes.

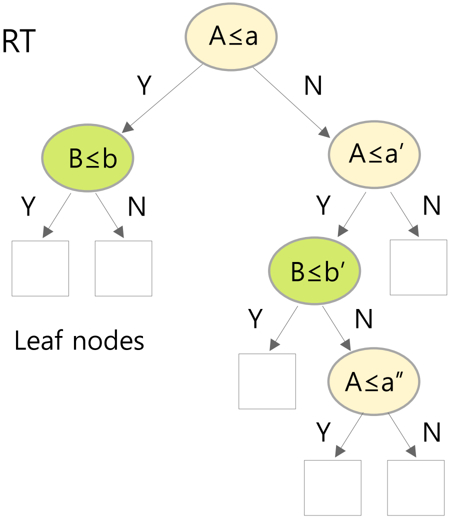
CART : Classification And Regression Tree

- CART is a Decision Tree algorithm proposed by Leo Breiman et al. in 1984.
- **ID3** is a relatively simple. ID3 splits all categories simultaneously, as shown on the left below. Since the tree is first divided into important features at the top of the tree and then into less important features as it goes down, performance does not decrease significantly even if the depth of the tree is reduced. ID3 is suitable for categorical features. This way, there is no big problem in dealing with ordinal and nominal categorical data. For continuous numeric features, we cannot split all cases simultaneously, so we convert them to categories for coarse splitting. Therefore, ID3 may not be suitable for numerical features.
- **CART** uses binary trees. It uses not only the important feature at the top of the tree, but also the best split point for the feature, allowing the depth to be made smaller. It is suitable for ensemble techniques that use many small trees. However, the CART algorithm may not be suitable for categorical features. In particular, it is not very good for nominal categorical data, such as ['red', 'green', 'blue']. On the other hand, CART can precisely process continuous numerical data. (You can find the exact split point). In order to use CART, categorical data must be converted to numeric type in advance. Overall, it has more advantages than ID3.

- ID3/C4.5



- CART



[MXML-2-05] Machine Learning / 2. Decision Tree – CART (Classification)

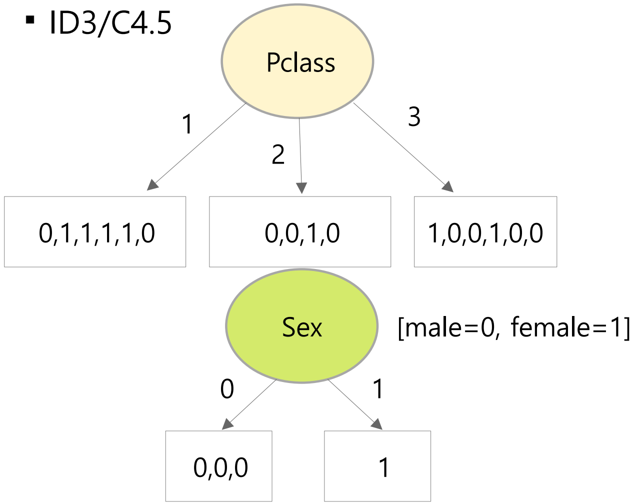
MX-AI

■ Categorical feature : Ordinal feature

- For ordinal features (Pclass) with size order, we can create trees as following.
- Pclass feature is in the following order: [Level 1, Level 2, Level 3]. We can assume that the level 1 is 'high', level 2 is 'medium', and level 3 is 'low'.
- In CART, there is no problem because Pclass is an ordinal feature, but there may be a problem if it is a nominal feature, such as ['red', 'green', 'blue']. Nominal features need to be converted to numbers using one-hot encoding or something like that.

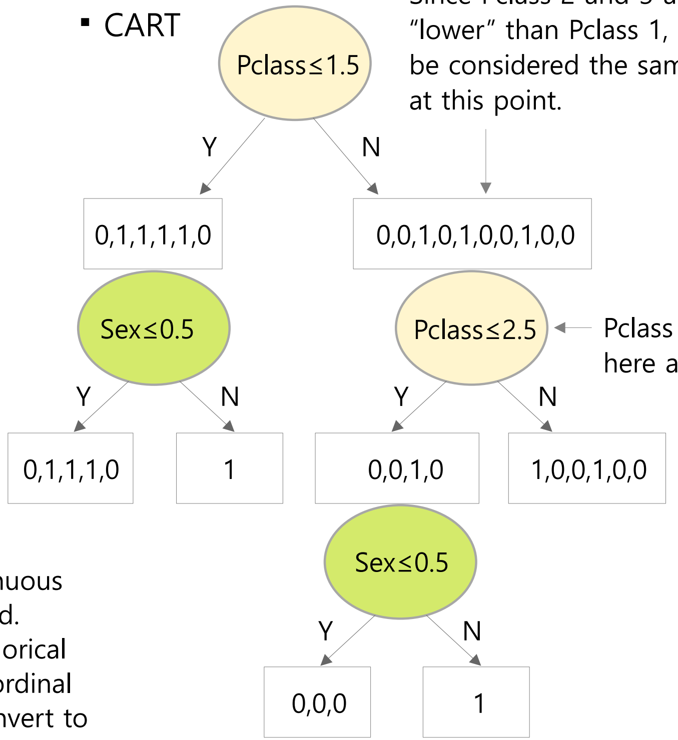
No	Pclass	Sex	Survived
1	1	male	0
2	3	female	1
3	1	male	1
4	3	male	0
5	2	male	0
6	2	male	0
7	1	female	1
8	2	female	1
9	3	female	0
10	1	male	1
11	1	male	1
12	3	male	1
13	3	female	0
14	3	female	0
15	1	male	0
16	2	male	0

- ID3/C4.5



[male=0, female=1]

- CART



Since Pclass 2 and 3 are both "lower" than Pclass 1, they can be considered the same group at this point.

Pclass uses here again.

- ID3/C4.5 is suitable for dealing with categorical data. Continuous numerical data is converted to categorical and approximated.
- CART is good at handling continuous numerical data. Categorical data must be converted to numeric type. For example, for ordinal types, convert to label encoding, and for nominal types, convert to one-hot encoding.

[MXML-2-05] Machine Learning / 2. Decision Tree – CART (Classification)

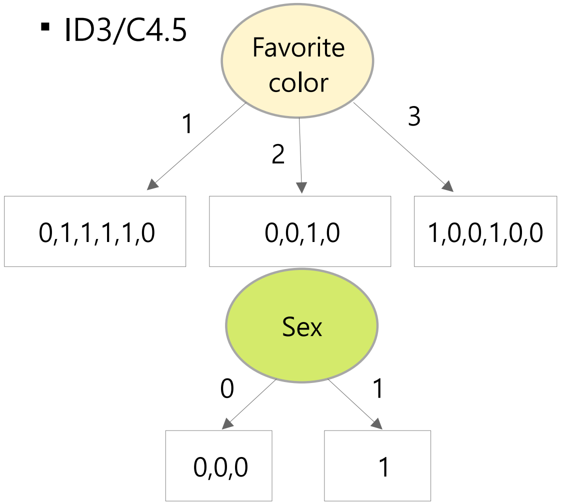
MX-AI

■ Categorical feature : Nominal feature

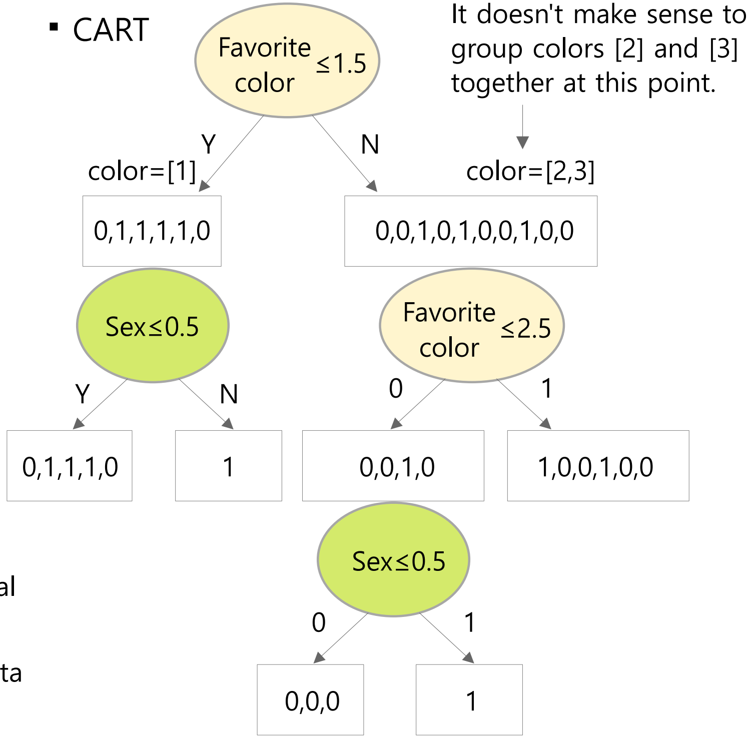
- For nominal feature with no size or order, you might consider the following trees. ID3 tree is fine, but CART tree may have problem.
- The "Favorite color" feature is the passenger's favorite color, marked using label encoding as [1=red, 2=blue, 3=yellow]. Colors have no concept of large or small size. If the colors are split into [1] and [2, 3], It doesn't make sense to group colors [2] and [3] together. In the case of Sex, even though it is nominal feature, there is no problem because there are only two types [0, 1].

No	Favorite color	Sex	Survived
1	1	male	0
2	3	female	1
3	1	male	1
4	3	male	0
5	2	male	0
6	2	male	0
7	1	female	1
8	2	female	1
9	3	female	0
10	1	male	1
11	1	male	1
12	3	male	1
13	3	female	0
14	3	female	0
15	1	male	0
16	2	male	0

- ID3/C4.5



- CART


It doesn't make sense to group colors [2] and [3] together at this point.

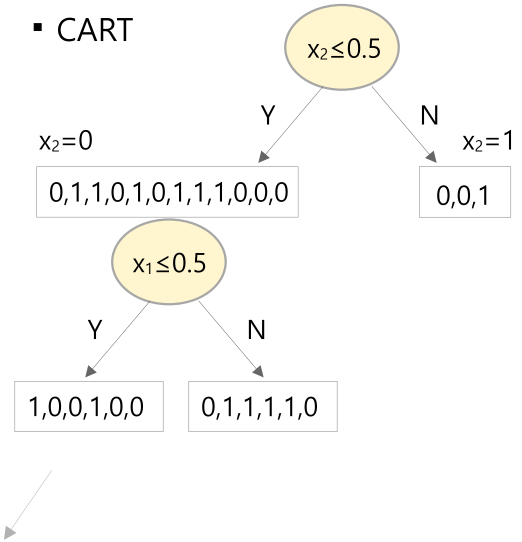
- In ID3/C4.5, there is no problem in processing nominal data because all cases are divided simultaneously.
- CART is a binary tree, so it cannot handle nominal data directly.

■ Categorical feature : One-hot encoding

- For nominal data, you can consider converting to one-hot encoding as follows. Instead of the "Favorite color" feature, we can create 3 new features (x_1, x_2, x_3). Since x_1, x_2, and x_3 are independent and all equidistant, there is no concept of size or order. Also, since there are only 0 and 1, the problem shown on the previous page does not occur.
- However, as the number of classes of "Favorite color" increases, the number of new features also increases, and the number of 0 also increases (sparsity). This will cause the tree to lean in one direction and increase its depth.
- Pruning deep tree in this way may reduce performance.
- In particular, ensemble models using shallow trees may further degrade performance.
- For reference, ensemble models such as LightGBM is able to process large amounts of data quickly by merging sparse features. It is called Exclusive Feature Bundling. Then x1, x2, and x3 features might be merged back into one feature, the original feature of "Favorite color".
- One-hot encoding can be considered for nominal data, but analysts must be aware of these problems and be prepared in advance.
- When using label encoding, one-hot encoding, or binary encoding methods, data scientists must make careful decisions considering their learning objectives and the characteristics of the dataset.

	Favorite color				
No	x_1	x_2	x_3	Sex	Survived
1	1	0	0	male	0
2	0	0	1	female	1
3	1	0	0	male	1
4	0	0	1	male	0
5	0	1	0	male	0
6	0	1	0	male	0
7	1	0	0	female	1
8	0	1	0	female	1
9	0	0	1	female	0
10	1	0	0	male	1
11	1	0	0	male	1
12	0	0	1	male	1
13	0	0	1	female	0
14	0	0	1	female	0
15	1	0	0	male	0

- CART



As data increases and the number of one-hot features increases, the tree becomes skewed to the left. The tree grows deeper.

[MXML-2-06] Machine Learning / 2. Decision Tree – CART (Classification)

Information Gain (IG) and the best split point

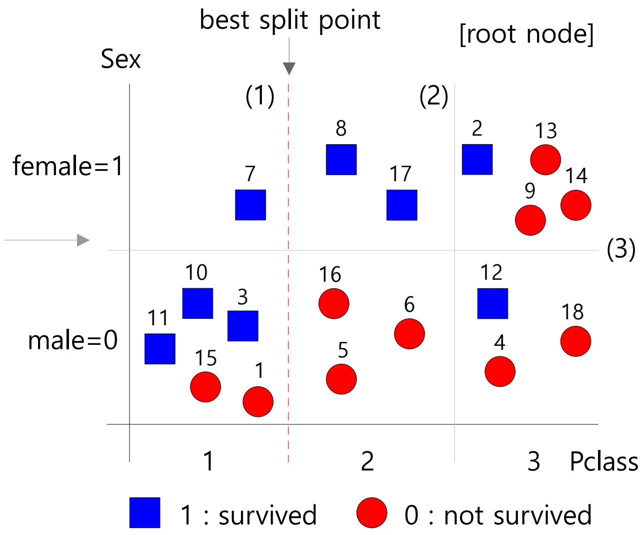
- Calculate IG using entropy or Gini index and find the optimal split point. IG calculation method is the same as ID3.
- Our goal is to find the best split point among the three possible candidates: (1), (2), (3).

	feature		target
No	Pclass	Sex	Survived
1	1	male	0
2	3	female	1
3	1	male	1
4	3	male	0
5	2	male	0
6	2	male	0
7	1	female	1
8	2	female	1
9	3	female	0
10	1	male	1
11	1	male	1
12	3	male	1
13	3	female	0
14	3	female	0
15	1	male	0
16	2	male	0
17	2	female	1
18	3	male	0

1. Gini index of Root node

$$G(root) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

$$= 1 - \left(\frac{8}{18}\right)^2 - \left(\frac{10}{18}\right)^2 = 0.494$$



[Part of the Titanic dataset]

2. Gini index and information gain (IG)

(1)

$$G(Pclass \le 1.5, Yes) = 1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2 = 0.444$$

$$G(Pclass \le 1.5, No) = 1 - \left(\frac{4}{12}\right)^2 - \left(\frac{8}{12}\right)^2 = 0.444$$

$$IG(Pclass \le 1.5) = 0.494 - \frac{6}{18} \times 0.444 - \frac{12}{18} \times 0.444 = 0.050$$

This is the biggest.

(2)

$$G(Pclass \le 2.5, Yes) = 1 - \left(\frac{6}{11}\right)^2 - \left(\frac{5}{11}\right)^2 = 0.496$$

$$G(Pclass \le 2.5, No) = 1 - \left(\frac{2}{7}\right)^2 - \left(\frac{5}{7}\right)^2 = 0.408$$

$$IG(Pclass \le 2.5) = 0.494 - \frac{11}{18} \times 0.496 - \frac{7}{18} \times 0.408 = 0.032$$

(3)

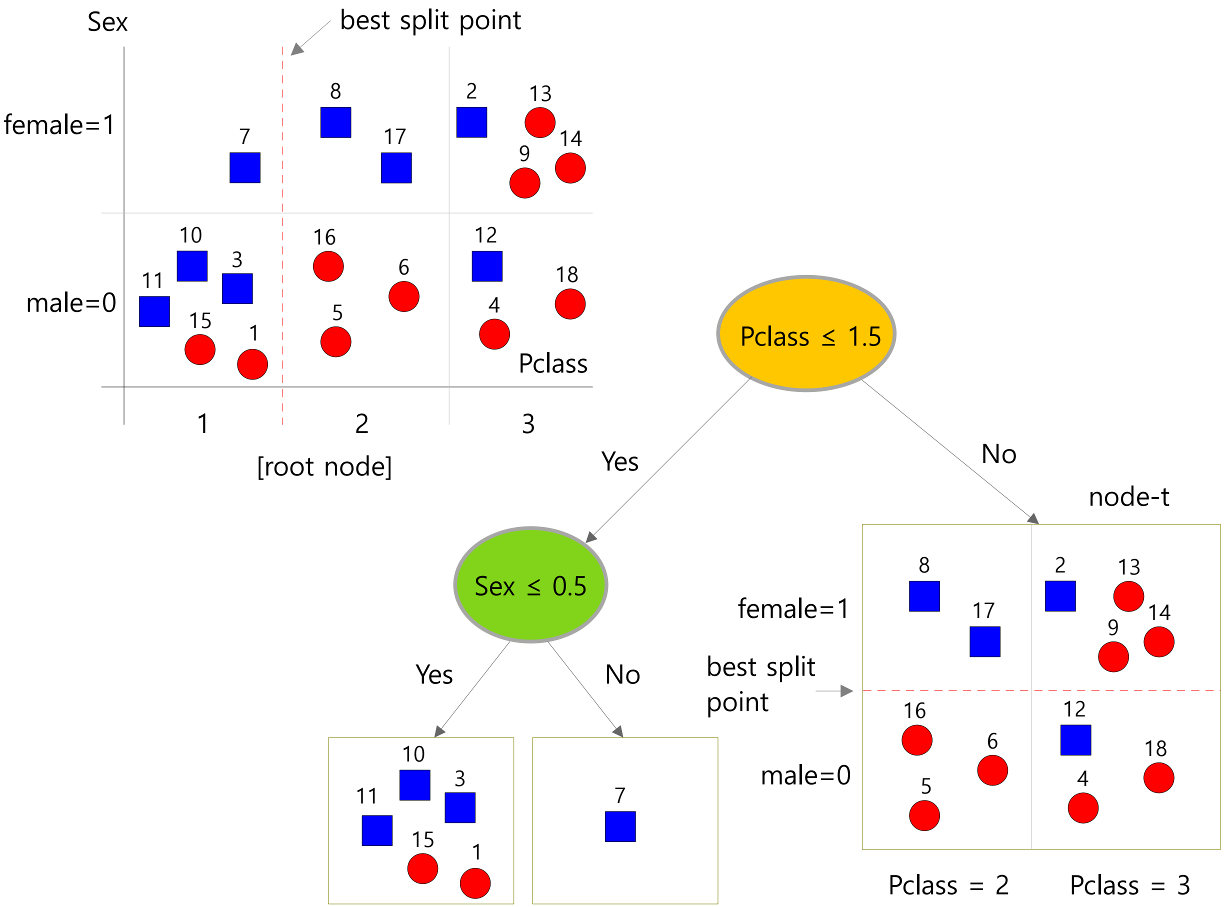
$$G(Sex \le 0.5, Yes) = 1 - \left(\frac{4}{11}\right)^2 - \left(\frac{7}{11}\right)^2 = 0.463$$

$$G(Sex \le 0.5, No) = 1 - \left(\frac{4}{7}\right)^2 - \left(\frac{3}{7}\right)^2 = 0.490$$

$$IG(Sex \le 0.5) = 0.494 - \frac{11}{18} \times 0.463 - \frac{7}{18} \times 0.490 = 0.021$$

[MXML-2-06] Machine Learning / 2. Decision Tree – CART (Classification)

■ Creating a tree using the best split points



Gini index of the node-t

$$G(t)=1-\sum_{i=1}^{c} p(i|t)^2$$

$$=1-\left(\frac{4}{12}\right)^2-\left(\frac{8}{12}\right)^2=0.444$$: Gini index of node-t

$$G(Pclass\le2.5\,,Yes)=1-\left(\frac{2}{5}\right)^2-\left(\frac{3}{5}\right)^2=0.480$$

$$G(Pclass\le2.5\,,No)=1-\left(\frac{2}{7}\right)^2-\left(\frac{5}{7}\right)^2=0.408$$

$$IG(Pclass\le2.5)=0.444-\frac{5}{12}\times0.480-\frac{7}{12}\times0.408=0.056$$

$$G(Sex\le0.5\,,Yes)=1-\left(\frac{1}{6}\right)^2-\left(\frac{5}{6}\right)^2=0.278$$

$$G(Sex\le0.5\,,No)=1-\left(\frac{3}{6}\right)^2-\left(\frac{3}{6}\right)^2=0.500$$

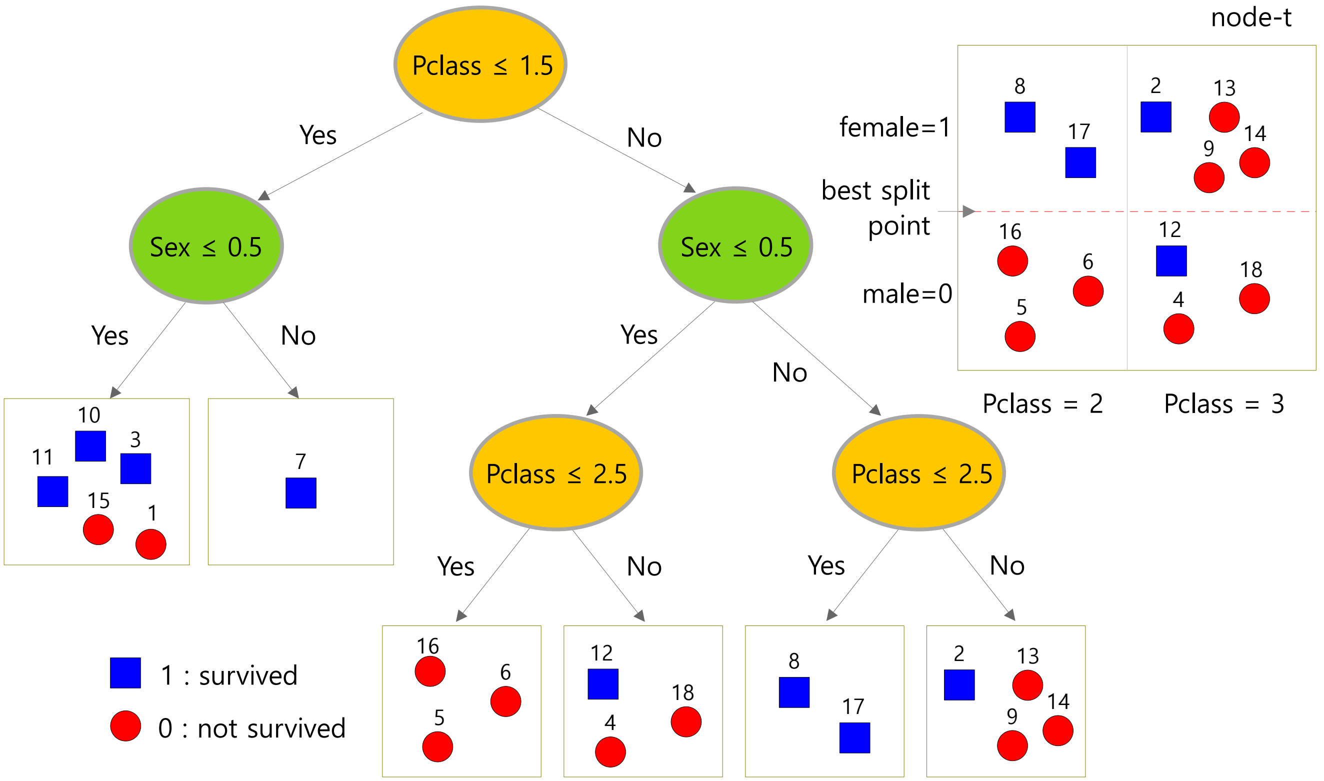
bigger

$$IG(Sex\le0.5)=0.444-\frac{6}{12}\times0.278-\frac{6}{12}\times0.500=0.105$$

[MXML-2-06] Machine Learning / 2. Decision Tree – CART (Classification)

Creating a tree using the best split points

- After creating a tree using the training data, the tree is used to estimate the target class of the test data.



- Target class estimation for test data

No	Pclass	Sex	Survived
1	3	female	?

- This test data is estimated to be survival = 0 (not-survived).
- The estimation accuracy can be said to be 75%. Three out of four are non-survival.

* In this example, A part of the Titanic dataset is used, and "pclass" is split first. If you use the full dataset, "sex" will be split first.

[MXML-2-06] Machine Learning / 2. Decision Tree – CART (Classification)

■ Continuous numerical features and the best split point

	feature			target
No	Pclass	Sex	Age	Survived
1	1	male	24	0
2	3	female	38	1
3	1	male	32	1
4	3	male	37	0
5	2	male	24	0
6	2	male	21	0
7	1	female	58	1
8	2	female	36	1
9	3	female	14	0
10	1	male	36	1
11	1	male	4	1
12	3	male	26	1
13	3	female	10	0
14	3	female	41	0
15	1	male	50	0
16	2	male	34	0
17	2	female	29	1
18	3	male	25	0

[Part of the Titanic dataset]

1) Remove duplicate "Age" feature and sort in ascending order.

4	10	14	21	24	25	26	29	32	34	36	37	38	41	50	58

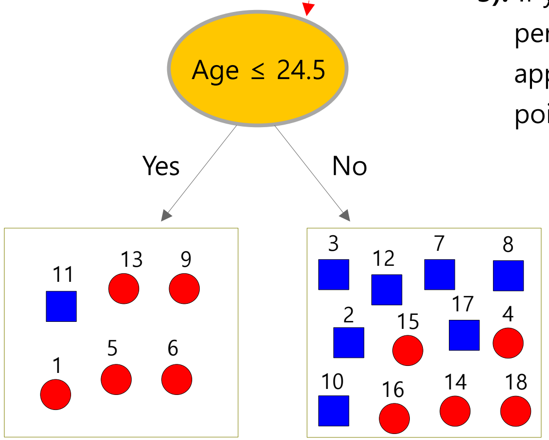
This is the most time-consuming part of CART. Problems arise when using large amounts of data.

2) Calculate the average of the two numbers and set it as the candidate split point.

...

7.0	12.0	17.5	22.5	24.5	25.5	27.5	30.5	33.0	35.0	36.5	37.5	39.5	45.5	54.0

example



🟦 1 : survived

🔴 0 : not survived

3). If you have a lot of data, you can reduce the number of candidates using percentile, quantile, or histogram-based methods. However, these methods find approximate split points. Additionally, parallel processing can speed up split point search. This will be covered in a later XGBoost and LightGBM session.

- Gini index before split

$$G(root)=1-\sum_{i=1}^{c} p(i|t)^2 = 1-\left(\frac{8}{18}\right)^2-\left(\frac{10}{18}\right)^2=0.494$$

- Gini index and information gain after split

$$G(Age\le 24.5, Yes)=1-\left(\frac{1}{6}\right)^2-\left(\frac{5}{6}\right)^2=0.28$$

$$G(Age\le 24.5, No)=1-\left(\frac{7}{12}\right)^2-\left(\frac{5}{12}\right)^2=0.486$$

$$IG(Age\le 24.5)=0.494-\frac{6}{18}\times 0-\frac{12}{18}\times 0.486=0.077$$

Bigger than the other candidates on the left.

$$IG(Pclass\le 1.5)=0.050$$

$$IG(Pclass\le 2.5)=0.032$$

$$IG(Sex\le 0.5)=0.021$$

■ Coding practice: Create MyDTreeClassifier class from scratch

```python
# MyDTreeClassifier.py
import numpy as np
from collections import Counter
import copy

# Implement the Decision Tree Classifier using binary tree.
class MyDTreeClassifier:
    def __init__(self, max_depth):
        self.max_depth = max_depth
        self.u_class = None      # unique class (target y value)
        self.estimator1 = dict() # tree result-1
        self.estimator2 = dict() # tree result-2
        self.feature = None # will be x_train when fit() is called
        self.target = None  # will be y_train when fit() is called

    # Calculate Gini index of a leaf node
    def gini_index(self, leaf):
        n = leaf.shape[0]
        gini = 1.0
        for c in self.u_class:
            cnt = (self.target[leaf] == c).sum()
            gini -= (cnt / n) ** 2
        return gini

    # split a node into left and right.
    # Find the best split point with highest information gain,
    # and split node with it.
    # did: data index on the leaf node.
    def node_split(self, did):
        n = did.shape[0]

        # Gini index of parent node before splitting.
        p_gini = self.gini_index(did)
```

```python
        # Split the node into all candidates for all features and
        # find the best feature and the best split point with the
        # highest information gain.
        # fid: feature_id
        max_ig = -999999
        for fid in range(self.feature.shape[1]):
            # feature data to be split
            x_feat = self.feature[did, fid].copy()

            # split x_feat using the best feature and the best
            # split point.
            # Note:
            # The code below is inefficient because it sorts x_feat
            # every time it is split. Future improvements are needed.

            # remove duplicates of x_feat and sort in ascending
            # order
            x_uniq = np.unique(x_feat)

            # list up all the candidates, which are the midpoints
            # of adjacent data.
            s_point = [np.mean([x_uniq[i-1], x_uniq[i]]) \
                       for i in range(1, len(x_uniq))]

            # len(s_point) > 1:
            #    Calculate the information gain for all candidates,
            #    and find the candidate with the largest IG.
            # len(s_point) < 1:
            #    skip the for-loop. x_feat either has only one data
            #    or all have the same value. No need to split.
            for p in s_point:
                # split x_feat into the left and the right node.
                left = did[np.where(x_feat <= p)[0]]
                right = did[np.where(x_feat > p)[0]]
```

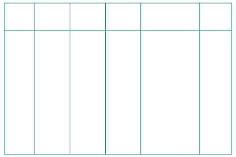
[MXML-2-07] Machine Learning / 2. Decision Tree – CART (Classification)

MX-AI

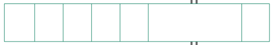
- Coding practice: Create MyDTreeClassifier class from scratch

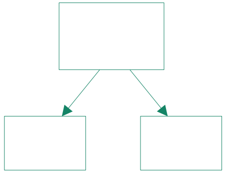
```python
# MyDTreeClassifier.py
import numpy as np
from collections
import copy

# Decision Tree Classifier
class MyDTreeClassifier:
    def __init__(self, max_depth):
        self.max_depth = max_depth
        self.u_class = None
        self.estimator1 = dict()
        self.estimator2 = dict()
        self.feature = None
        self.target = None

    # Calculate Gini index of a leaf node
    def gini_index(self, leaf):
        n = leaf.shape[0]
        gini = 1.0
        for c in self.u_class:
            cnt = (self.target[leaf] == c).sum()
            gini -= (cnt / n) ** 2
        return gini

    # split a node into left and right.
    # Find the best split point with highest IG.
    # and split node with it.
    # did: data index on the leaf node.
    def node_split(self, did):
        n = did.shape[0]

        # Gini index of parent node before splitting.
        p_gini = self.gini_index(did)
```

{'fid':
 'split_point':
 'left': value
 {'fid':, 'split_point':, 'left':, 'right':}
 'right':value
 {'fid':, 'split_point':, 'left':, 'right':}



[x_train]

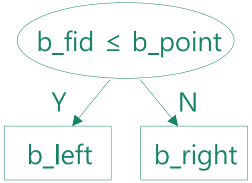




```python
        # Split the node into all candidates for all features and
        # find the best feature and the best split point with the
        # highest information gain.
        # fid: feature_id
        max_ig = -999999
        for fid in range(self.feature.shape[1]):
            # feature data to be split
            x_feat = self.feature[did, fid].copy()

            # split x_feat using the best feature and the best
            # split point.
            # Note:
            # The code below is inefficient because it sorts x_feat
            # every time it is split. Future improvements are needed.

            # remove duplicates of x_feat and sort in ascending
            # order
            x_uniq = np.unique(x_feat)

            # list up all the candidates, which are the midpoints
            # of adjacent data.
            s_point = [np.mean([x_uniq[i-1], x_uniq[i]]) \
                       for i in range(1, len(x_uniq))]

            # len(s_point) > 1:
            #    Calculate the information gain for all candidates,
            #    and find the candidate with the largest IG.
            # len(s_point) < 1:
            #    skip the for-loop. x_feat either has only one data
            #    or all have the same value. No need to split.
            for p in s_point:
                # split x_feat into the left and the right node.
                left = did[np.where(x_feat <= p)[0]]
                right = did[np.where(x_feat > p)[0]]
```

Coding practice: Create MyDTreeClassifier class from scratch

```
                # calculate Gini index after splitting.
for fid         l_gini = self.gini_index(left)
                r_gini = self.gini_index(right)

                # calculate information gain (IG)
                ig = p_gini - (l_gini * left.shape[0] / n) -\
for p in                      (r_gini * right.shape[0] / n)
s_point:
                # find where the information gain is greatest.
                if ig > max_ig:
                    max_ig = ig
                    b_fid = fid      # best feature id
                    b_point = p      # best split point
                    b_left = left    # data index on the left
                    b_right = right  # data index on the right

        if max_ig > 0.:     # split
            return {'fid':b_fid, 'split_point':b_point,\
                    'left':b_left, 'right':b_right}
        else:
            return  None    # No split

    # Create a binary tree using recursion
    def recursive_split(self, node, curr_depth):
        left = node['left']
        right = node['right']

        # exit recursion
        if curr_depth >= self.max_depth:
            return
```



```
        # recursion
        s = self.node_split(left)
        if isinstance(s, dict):   # split to the left, done.
            node['left'] = s
            self.recursive_split(node['left'], curr_depth+1)

        s = self.node_split(right)
        if isinstance(s, dict):   # split to the right, done.
            node['right'] = s
            self.recursive_split(node['right'], curr_depth+1)

    # majority vote
    def majority_vote(self, did):
        c = Counter(self.target[did])
        return c.most_common(1)[0][0]

    # Change the data in the leaf node to majority class.
    def update_leaf(self, d):
        if isinstance(d, dict):
            for key, value in d.items():
                if key == 'left' or key == 'right':
                    rtn = self.update_leaf(value)
                    if rtn[0] == 1:      # leaf node
                        d[key] = rtn[1]
            return 0, 0  # the first 0 means this is not a leaf node.
        else:            # leaf node
            # the first 1 means this is a leaf node.
            return 1, self.majority_vote(d)
```

[MXML-2-07] Machine Learning / 2. Decision Tree – CART (Classification)

- Coding practice: Create MyDTreeClassifier class from scratch

```
# create a tree using training data, and return the result
# of the tree.
# x : feature data, y: target data
def fit(self, x, y):
    self.feature = x
    self.target = y
    self.u_class = np.unique(y)

    # Initially, the root node holds all data indices.
    root = self.node_split(np.arange(x.shape[0]))
    if isinstance(root, dict):
        self.recursive_split(root, curr_depth=1)

    # tree result-1. Every leaf node has data indices.
    # It is used for predict_proba(), etc.
    self.estimator1 = root

    # tree result-2. Every leaf node has the majority class.
    # It is used for predict().
    self.estimator2 = copy.deepcopy(self.estimator1)
    self.update_leaf(self.estimator2)      # tree result-2
    return self.estimator2
```

```
# Estimate the target class of a test data.
def x_predict(self, p, x):
    if x[p['fid']] <= p['split_point']:
        if isinstance(p['left'], dict):# recursion if not leaf
            return self.x_predict(p['left'], x)   # recursion
        else:         # return the value in the leaf, if leaf.
            return p['left']
    else:
        if isinstance(p['right'], dict):# recursion if not leaf
            return self.x_predict(p['right'], x)  # recursion
        else:            # return the value in the leaf, if leaf.
            return p['right']

# Estimate the target class of a x_test.
def predict(self, x_test):
    p = self.estimator2     # predictor
    y_pred = [self.x_predict(p, x) for x in x_test]
    return np.array(y_pred)
```

- Coding practice: Compare MyDTreeClassifier and DecisionTreeClassifier in sklearn.

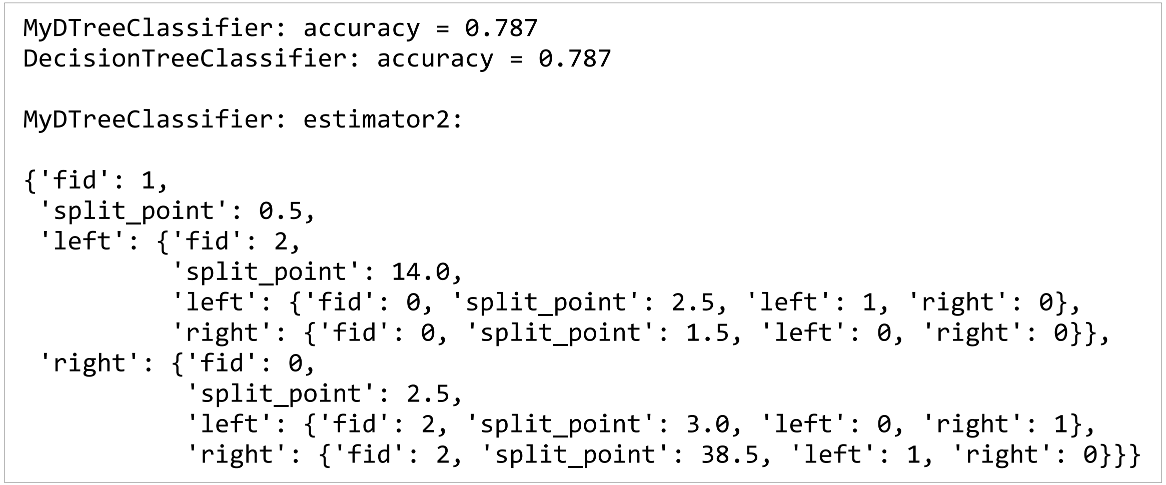
```
# 2.CART(classification).py
import numpy as np
import pandas as pd
from MyDTreeClassifier import MyDTreeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pprint

# Read the Titanic dataset and perform simple preprocessing.
df = pd.read_csv('data/titanic.csv')
df['Age'].fillna(df['Age'].mean(), inplace=True) # Replace with mean
df['Embarked'].fillna('N', inplace = True)       # Replace with 'N'
df['Sex'] = df['Sex'].factorize()[0]             # label encoding
df['Embarked'] = df['Embarked'].factorize()[0]   # label encoding
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1,\
          inplace=True)

#  Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# 0        0       3    0  22.0      1      0   7.2500         0
# 1        1       1    1  38.0      1      0  71.2833         1
# 2        1       3    1  26.0      0      0   7.9250         0
# 3        1       1    1  35.0      1      0  53.1000         0
# 4        0       3    0  35.0      0      0   8.0500         0
```

```
# split the data into train, validation and test data.
y = np.array(df['Survived'])
x = np.array(df.drop('Survived', axis=1))
x_train, x_test, y_train, y_test = train_test_split(x, y)

depth = 3
my_model = MyDTreeClassifier(max_depth = depth)
my_model.fit(x_train, y_train)
my_pred = my_model.predict(x_test)
acc = (y_test == my_pred).mean()
print('MyTreeClassifier: accuracy = {:.3f}'.format(acc))

# Compare the results with sklearn's DecisionTreeClassifier.
# -------------------------------------------------------------
sk_model = DecisionTreeClassifier(max_depth=depth,
                                  random_state=1)
sk_model.fit(x_train, y_train)
sk_pred = sk_model.predict(x_test)
acc = (y_test == sk_pred).mean()
print('DecisionTreeClassifier: accuracy = {:.3f}'.format(acc))

print('\nMyTreeClassifier: estimator2:')
pprint.pprint(my_model.estimator2, sort_dicts=False)

plt.figure(figsize=(12, 6))
tree.plot_tree(sk_model)
plt.show()
```

[MXML-2-07] Machine Learning / 2. Decision Tree – CART (Classification)

MX-AI

■ Coding practice: Compare MyDTreeClassifier and DecisionTreeClassifier in sklearn.

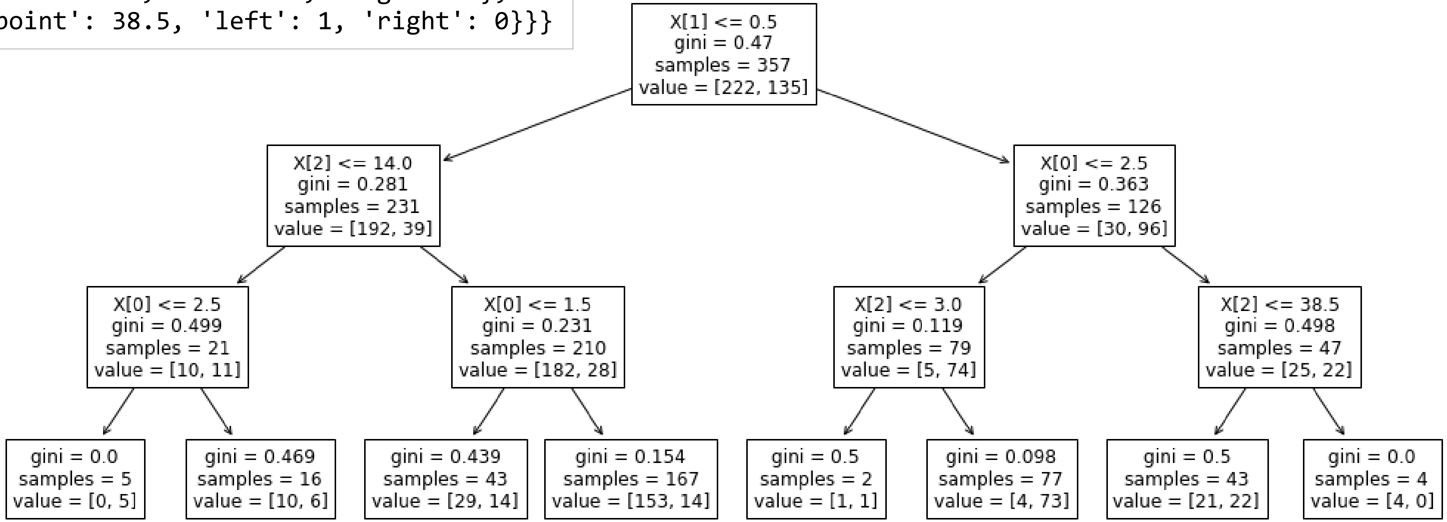


```
MyDTreeClassifier: accuracy = 0.787
DecisionTreeClassifier: accuracy = 0.787

MyDTreeClassifier: estimator2:

{'fid': 1,
 'split_point': 0.5,
 'left': {'fid': 2,
          'split_point': 14.0,
          'left': {'fid': 0, 'split_point': 2.5, 'left': 1, 'right': 0},
          'right': {'fid': 0, 'split_point': 1.5, 'left': 0, 'right': 0}},
 'right': {'fid': 0,
           'split_point': 2.5,
           'left': {'fid': 2, 'split_point': 3.0, 'left': 0, 'right': 1},
           'right': {'fid': 2, 'split_point': 38.5, 'left': 1, 'right': 0}}}
```

- The result of MyDTreeClassifier

- The result of sklearn.tree.DecisionTreeClassifier

* The two results agree well.

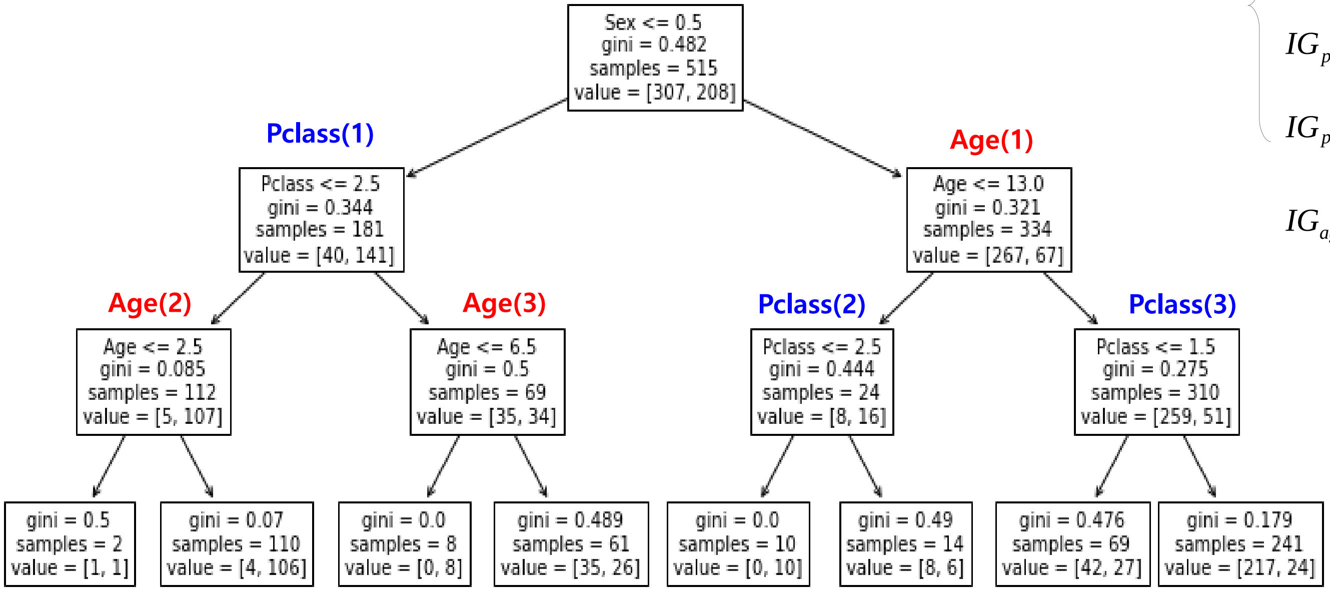


* If you handle multi-class or change sklearn conditions, the two trees may look different. But the accuracy is similar.

Feature importance

- Each feature in the dataset has different importance. Some features may be very useful in describing the target class, and some may not be so helpful. Information gain (IG) can be used to determine the importance of features in the dataset. The greater the information gain, the more important the feature. In the tree below, the 'Sex' feature is the most important, followed by the 'Pclass' feature.

$$IG_{sex} = 0.482 - \left(0.344 \times \frac{181}{515} + 0.321 \times \frac{334}{515}\right) = 0.153$$



$$IG_{pclass(1)} = 0.344 - \left(0.085 \times \frac{112}{181} + 0.5 \times \frac{69}{181}\right) = 0.101$$

$$IG_{pclass(2)} = 0.444 - \left(0 \times \frac{10}{24} + 0.49 \times \frac{14}{24}\right) = 0.158$$

$$IG_{pclass(3)} = 0.275 - \left(0.476 \times \frac{69}{310} + 0.179 \times \frac{241}{310}\right) = 0.030$$

$$IG_{pclass} = 0.101 \times \frac{181}{515} + 0.158 \times \frac{24}{515} + 0.030 \times \frac{310}{515} = 0.061$$

$IG_{age} = 0.032$ - Calculated in the same way as Pclass.

Normalization $$IG_{sex} = \frac{IG_{sex}}{GI_{sex} + IG_{pclass} + IG_{age}}$$

$$IG_{sex} = 0.622$$

$$IG_{pclass} = 0.248$$

$$IG_{age} = 0.130$$

Importance of each feature.

[MXML-2-08] Machine Learning / 2. Decision Tree – CART (Classification)

- Hands-on with sklearn's DecisionTreeClassifier: optimal depth of tree, feature importance

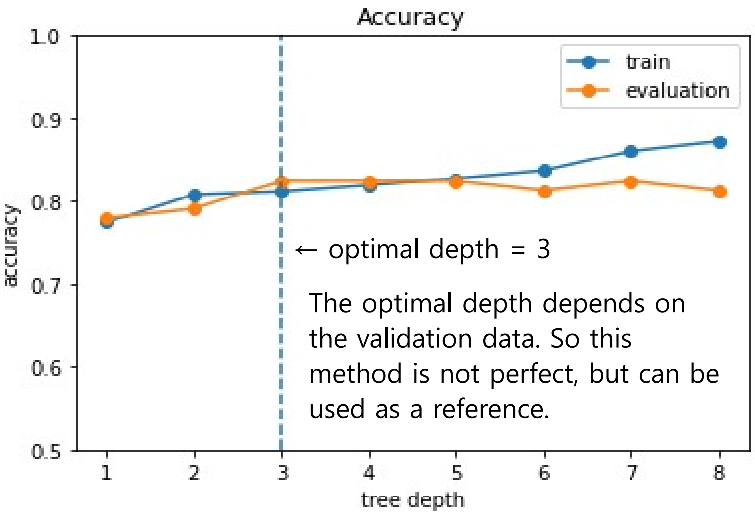
```python
# [MXML-2-08] 3.CART(sklearn).py
# DecisionTreeClassifier in sklearn
#
# The characteristics of DecisionTreeClassifier:
# 1. Use the CART algorithm (binary tree).
#    ID3/C4.5 (general tree) is not supported.
# 2. Categorical feature is not directly supported.
#    All categorical features (e.g. 'female', 'male') must be
#    converted to numeric data (e.g. 0, 1).
#    All numeric features are treated as continuous features.
#    Split using inequality. (e.g. sex ≤ 0.5)
# ------------------------------------------------------------
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree
import matplotlib.pyplot as plt

# Of the Titanic dataset, only three features are used.
feat_names = ['Pclass', 'Sex', 'Age']
df = pd.read_csv('data/titanic.csv')[feat_names + ['Survived']]
df['Sex'] = df['Sex'].factorize()[0] # convert string to number
df = df.dropna()           # Delete all rows with missing values.
col_names = list(df.columns)

# Separate the Titanic data into features and target class.
x_data = np.array(df[feat_names])  # features
y_data = np.array(df['Survived'])  # target class

# Split the data into training, validation and test data.
x_train, x_test, y_train, y_test = \
    train_test_split(x_data, y_data, test_size = 0.3)

x_test, x_eval, y_test, y_eval = \
    train_test_split(x_test, y_test, test_size = 0.5)
```

```python
# Create decision tree models of various depths,
# and measure the accuracy of validation data for each model.
train_acc = []
eval_acc = []
max_depth = 8
for d in range(1, max_depth+1):
    model = DecisionTreeClassifier(max_depth=d)
    model.fit(x_train, y_train)

    # Measure the accuracy of this model using the training data.
    y_pred = model.predict(x_train)
    train_acc.append((y_pred == y_train).mean())

    # Measure the accuracy of this model using the validation data.
    y_pred = model.predict(x_eval)
    eval_acc.append((y_pred == y_eval).mean())
    print('Depth = {}, train_acc = {:.4f}, eval_acc = {:.4f}'\
          .format(d, train_acc[-1], eval_acc[-1]))

# Find the optimal depth with the highest accuracy of validation data.
opt_depth = np.argmax(eval_acc) + 1

# Visualize accuracy changes as depth changes.
plt.plot(train_acc, marker='o', label='train')
plt.plot(eval_acc, marker='o', label='evaluation')
plt.legend()
plt.title('Accuracy')
plt.xlabel('tree depth')
plt.ylabel('accuracy')
plt.xticks(np.arange(max_depth), np.arange(1, max_depth+1))
plt.axvline(x=opt_depth-1, ls='--')
plt.ylim(0.5, 1.0)
plt.show()
```

- Hands-on with sklearn's DecisionTreeClassifier: optimal depth of tree, feature importance

```
# Regenerate the tree with optimal depth.
# model = DecisionTreeClassifier(max_depth=opt_depth)

# I set max_step=3 as a constant value for tree visualization.
model = DecisionTreeClassifier(max_depth=3)
model.fit(x_train, y_train)

# Use test data to evaluate final performance.
y_pred = model.predict(x_test)
test_acc = (y_pred == y_test).mean()
print('Optimal depth = {}, test_acc = {:.4f}'.\
       format(opt_depth, test_acc))

# Visualize the tree
# plt.figure(figsize=(20,10))
plt.figure(figsize=(14,6))
tree.plot_tree(model, feature_names = feat_names, fontsize=10)
plt.show()

# Analyze the importance of features.
feature_importance = model.feature_importances_
n_feature = x_train.shape[1]
idx = np.arange(n_feature)

plt.barh(idx, feature_importance, align='center')
plt.yticks(idx, col_names[:-1], size=12)
plt.xlabel('importance', size=15)
plt.ylabel('feature', size=15)
plt.show()

print('feature importance = {}'.format(feature_importance.round(3)))
```

```
Result:

Depth = 1, train_acc = 0.7748, eval_acc = 0.7802
Depth = 2, train_acc = 0.8078, eval_acc = 0.7912
Depth = 3, train_acc = 0.8117, eval_acc = 0.8242
Depth = 4, train_acc = 0.8194, eval_acc = 0.8242
Depth = 5, train_acc = 0.8272, eval_acc = 0.8242
Depth = 6, train_acc = 0.8369, eval_acc = 0.8132
Depth = 7, train_acc = 0.8602, eval_acc = 0.8242
Depth = 8, train_acc = 0.8718, eval_acc = 0.8132
```

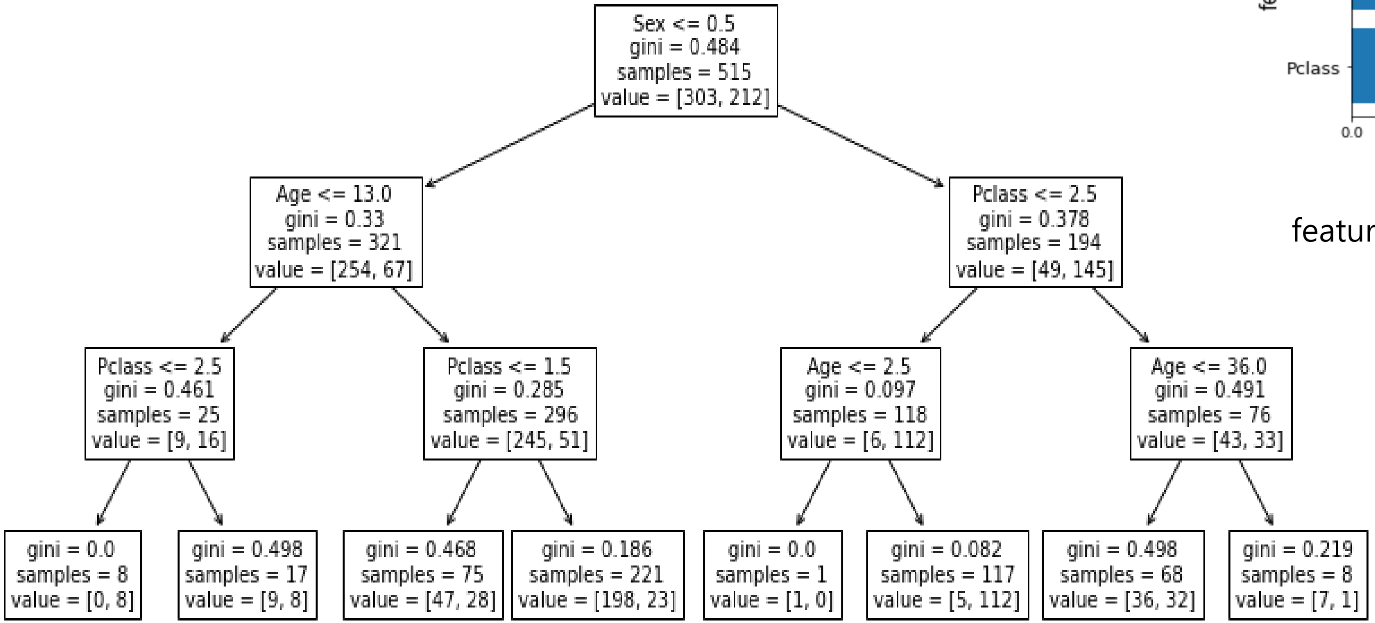


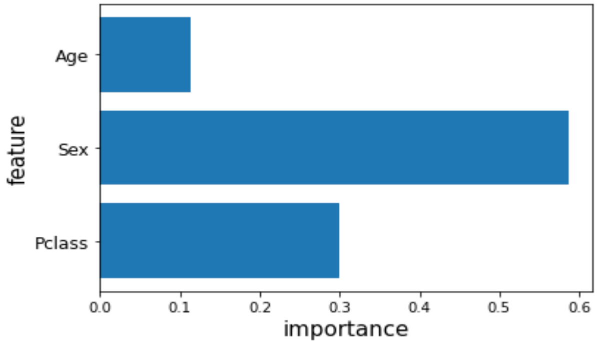
← optimal depth = 3

The optimal depth depends on the validation data. So this method is not perfect, but can be used as a reference.

[MXML-2-08] Machine Learning / 2. Decision Tree – CART (Classification)

- Hands-on with sklearn's DecisionTreeClassifier: optimal depth of tree, feature importance





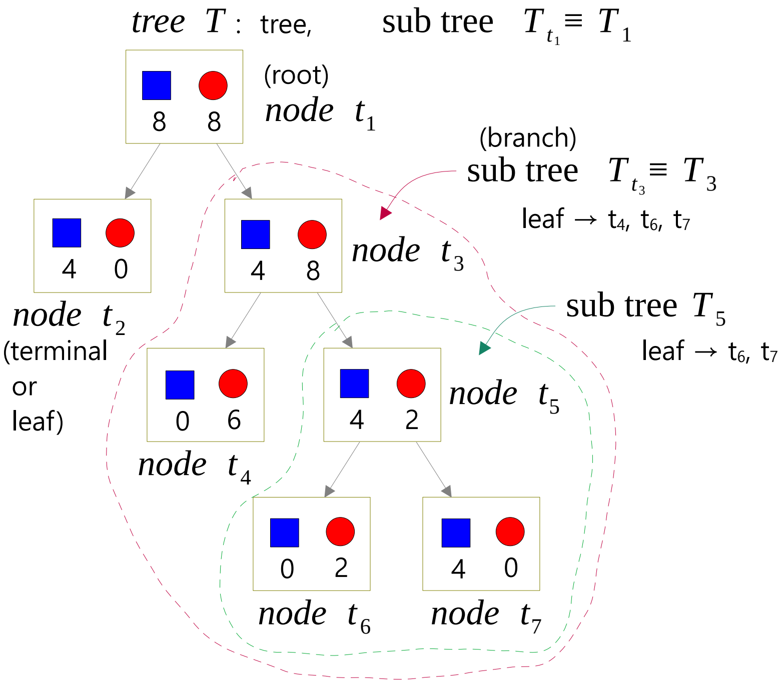
feature importance = [0.299 0.587 0.114]

[MXML-2-09] Machine Learning / 2. Decision Tree – CART (Classification)

Cost Complexity Pruning (CCP) : post-pruning - Notation

- As the tree gets deeper, the error in the training data decreases, but the error in the test data may increase. ← overfitting
- To prevent overfitting of the decision tree, branches of the complex tree are pruned to create a less complex tree. ← pruning
- Initially, the tree is constructed complex enough to have no misclassification cases at the leaf nodes, and then unnecessary branches are pruned. ← post pruning
- CCP is a pruning algorithm proposed by Breiman, Stone, and Olsen when they introduced CART in 1984.

Notation



Mis-classification error rate

$R(T)$: error rate of (sub) tree T $r(t)$: error rate of node t   m: "#" of minority

$$R(T) = \sum_{t \in leaf} r(t) \cdot p(t) = \sum_{t \in leaf} R(t) \qquad r(t) = \frac{m}{n(t)} \qquad p(t) = \frac{n(t)}{n}$$

$n(t)$ - the number of data in node (t) n - the number of training data

$$r(t_1) = \frac{8}{16} \qquad r(t_2) = \frac{0}{4} \qquad r(t_3) = \frac{4}{12} \qquad r(t_4) = \frac{0}{6} \qquad \cdots$$

$$p(t_1) = \frac{16}{16} \qquad p(t_2) = \frac{4}{16} \qquad p(t_3) = \frac{12}{16} \qquad p(t_4) = \frac{6}{16} \qquad \cdots$$

$$R(t) = r(t) \cdot p(t) = \frac{m}{n} \qquad R(t_1) = r(t_1) \cdot p(t_1) = \frac{8}{16} \cdot \frac{16}{16}$$

$$R(T_1) = r(t_2)p(t_2) + r(t_4)p(t_4) + r(t_6)p(t_6) + r(t_7)p(t_7) = 0$$ ← leaf node only

$$= \frac{0}{4} \times \frac{4}{16} + \frac{0}{6} \times \frac{6}{16} + \ldots = 0$$ ← no errors in tree T₁

$$R(T_3) = R(t_4) + R(t_6) + R(t_7) = 0$$ ← leaf node only

* Reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning, https://online.stat.psu.edu/stat508/lesson/11/11.8/11.8.1

[MXML-2-09] Machine Learning / 2. Decision Tree – CART (Classification)

MX-AI

■ Cost Complexity Pruning (CCP) : post-pruning – definition of Cost Complexity

- Since trying to lower the misclassification rate R(T) results in a deeper tree, we define a new measure C(T) and build a tree that lowers C(T).
- C(T) is called cost complexity. Objective: Min C(T) instead of Min R(T)
- C(T) is R(T) plus penalty. The deeper the tree, the greater the penalty.

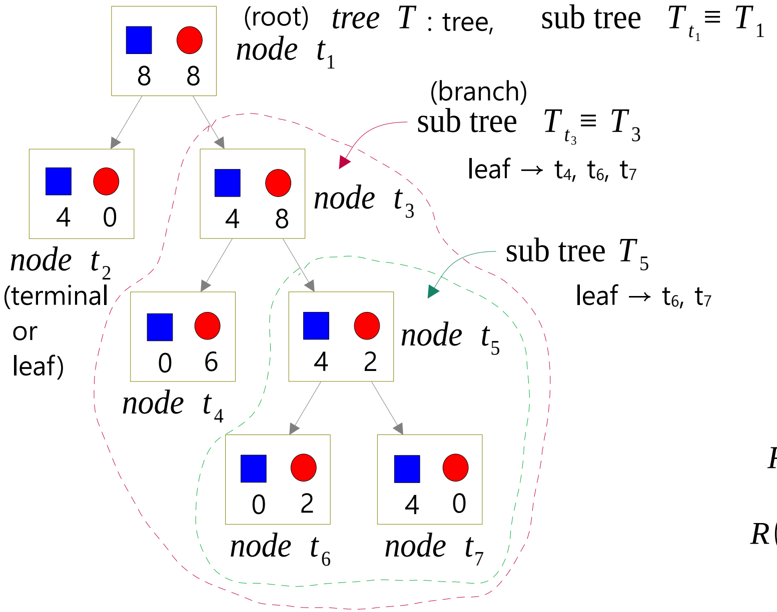
regularization constant

misclassification rate of a tree T

the number of leaf node in the tree

- Cost Complexity: $C(T) = R(T) + \alpha|T|$

Penalty term



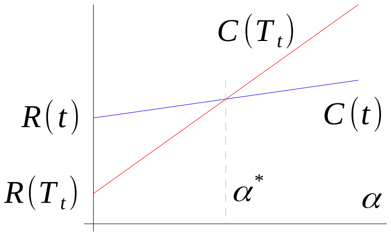
$C(T_t) = R(T_t) + \alpha|T_t|$ ← Cost complexity of the subtree into which node-t is split.

$$= \sum_{t\in leaf} r(t)\cdot p(t) + \alpha|T_t|$$

$C(t) = R(t) + \alpha = r(t)\cdot p(t) + \alpha$ ← Cost complexity of node-t. |T|=1

$C(t_3) = R(t_3) + \alpha = r(t_3)\cdot p(t_3) + 0.1 = \frac{4}{12}\cdot\frac{12}{16} + 0.1 = 0.35$ (α = 0.1)

$C(T_3) = R(t_4) + R(t_6) + R(t_7) + 3\alpha = 0 + 0.3 = 0.30$



- The smaller α is, the larger C(t) is. R(T) is smaller because the error decreases as the tree is split.
- As α increases, the number of leaf nodes increases, which increases the penalty and increases the cost complexity.
- Because the slope of C(Tt) is large, the larger α is, the larger C(Tt) is.

$$\alpha^* = \frac{R(t) - R(T_t)}{|T_t| - 1}$$ ←

In regions where α is smaller than this value, the misclassification rate decreases when splitting the tree. $R(T_t) < R(t)$.

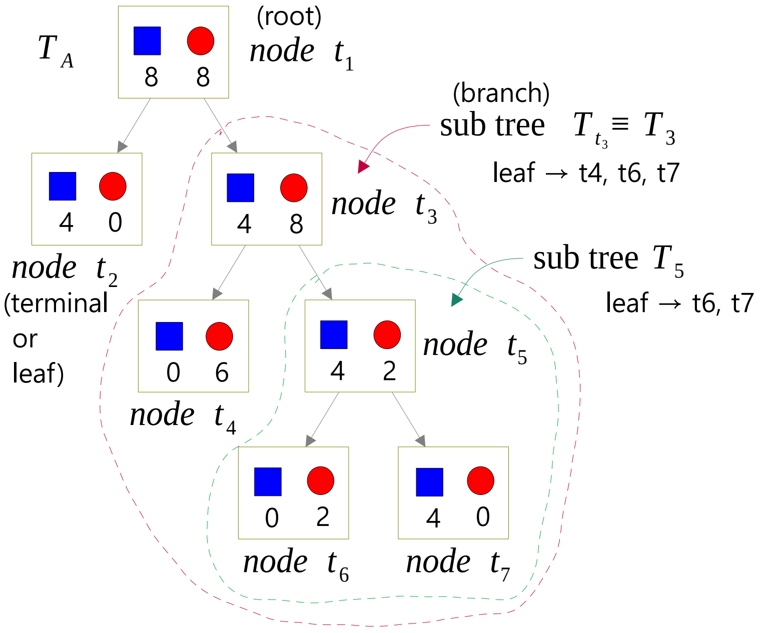
* reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning
: https://online.stat.psu.edu/stat508/lesson/11/11.8/11.8.1

MX-AI

■ Cost Complexity Pruning (CCP) : post-pruning – Example

- **Step-1**: Build a tree as deep as possible and find the subtree with the smallest α.

tree T : tree, sub tree $T_{t_1} \equiv T_1$



$$C(t) = R(t) + \alpha \qquad C(T_t) = R(T_t) + \alpha|T_t| \qquad \alpha^* = \frac{R(t) - R(T_t)}{|T_t| - 1}$$

$$C(t_1)=\frac{8}{16}\times\frac{16}{16} + \alpha \qquad C(T_1)=\frac{0}{4}\times\frac{4}{16} + \frac{0}{6}\times\frac{6}{16} + \ldots + \alpha\times 4 \longrightarrow \alpha^* = \frac{1}{6}$$

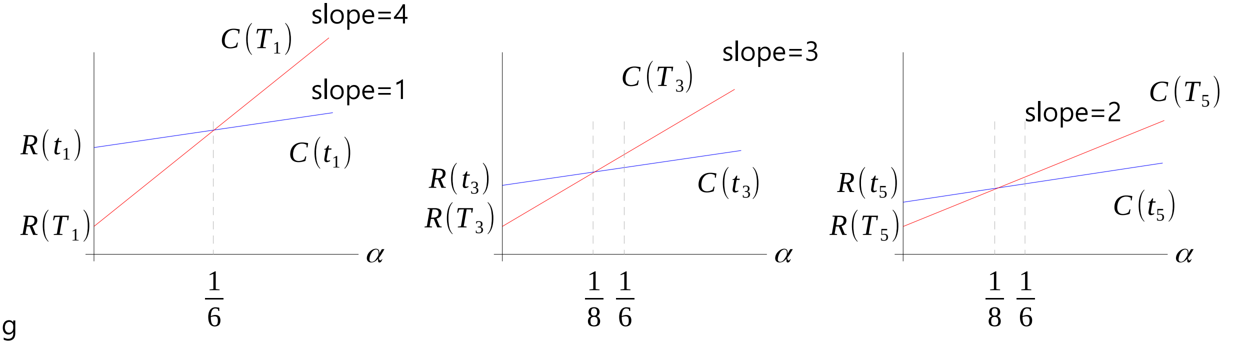
$$C(t_3)=\frac{4}{12}\times\frac{12}{16} + \alpha \qquad C(T_3)=\frac{0}{6}\times\frac{6}{16} + \frac{0}{2}\times\frac{2}{16} + \ldots + \alpha\times 3 \longrightarrow \alpha^* = \frac{1}{8}$$

$$C(t_5)=\frac{2}{6}\times\frac{6}{16} + \alpha \qquad C(T_5)=\frac{0}{2}\times\frac{2}{16} + \frac{0}{4}\times\frac{4}{16} + \alpha\times 2 \longrightarrow \alpha^* = \frac{1}{8}$$

* in case of a tie, we choose the one that prunes fewer nodes.

weakest link

- Choose the one with the smallest α*. For the same case, select the lower subtree. |T| is smaller.
- Small |T| is not always a weak link. |T| = 3 or 4 may be weak.



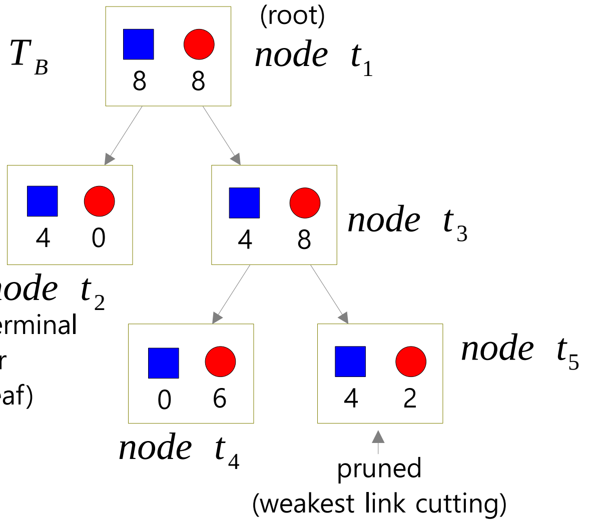
* Reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning

MX-AI

■ Cost Complexity Pruning (CCP) : post-pruning – Example

* Reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning

- **Step-2** : Prune the sub tree found in step-1, and find the sub tree with the smallest α again.

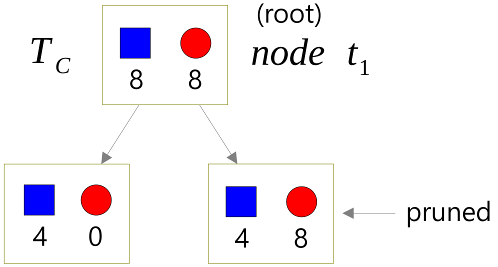


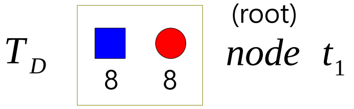
$$C(t_1)=\frac{8}{16}\times\frac{16}{16}+\alpha \qquad C(T_1)=\frac{0}{4}\times\frac{4}{16}+\frac{0}{6}\times\frac{6}{16}+\frac{2}{6}\times\frac{6}{16}+3\alpha \rightarrow \alpha^*=\frac{3}{16}$$

weakest link

$$C(t_3)=\frac{4}{12}\times\frac{12}{16}+\alpha \qquad C(T_3)=\frac{0}{6}\times\frac{6}{16}+\frac{2}{6}\times\frac{6}{16}+2\alpha \rightarrow \alpha^*=\frac{1}{8}$$

- **Step-3** : Prune the sub tree found in step-2, and find the sub tree with the smallest α. Repeat this process.





$$C(t_1)=\frac{8}{16}\times\frac{16}{16}+\alpha \qquad C(T_1)=\frac{0}{4}\times\frac{4}{16}+\frac{4}{12}\times\frac{12}{16}+2\alpha \rightarrow \alpha^*=\frac{1}{4}$$

- **final step** : Create a list of α candidates and determine the final α through cross-validation on the candidate trees.

$\alpha=[0,\frac{1}{8},\frac{1}{8},\frac{1}{4}]$ ← α candidates: α is getting bigger.
The larger α, the smaller the tree.

(by definition: α ≥ 0)

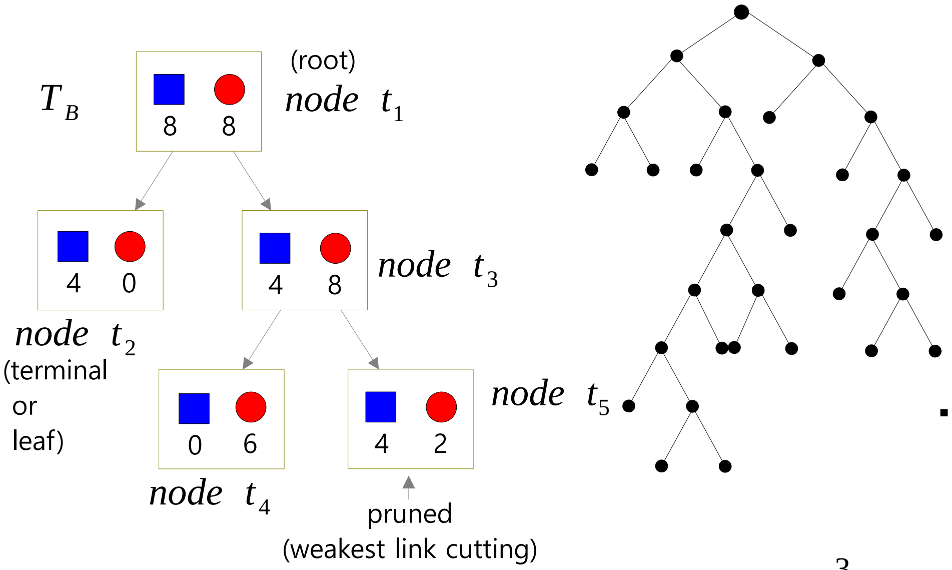
- We added 0 to the list of α candidates. This is the case of not regularization.
- If 0≤α<1/8, T_A is the best, if α=1/8, T_B is best, if 1/8<α<1/4, $T_C$is best, and if α≥1/4, T_D is the best.
- Perform cross-validation on TA, TB, TC, and TD to select α with the smallest error. The tree corresponding to the selected α is the optimal tree.
- In this example, all subtrees have been pruned, but in a real tree, not all subtrees are pruned. Therefore, not all subtrees are candidates.

[MXML-2-09] Machine Learning / 2. Decision Tree – CART (Classification)

■ Cost Complexity Pruning (CCP) : post-pruning – Example

* Reference : http://mlwiki.org/index.php/Cost-Complexity_Pruning

- **Step-2** : Prune the sub tree found in step-1, and find the sub tree with the smallest α again.

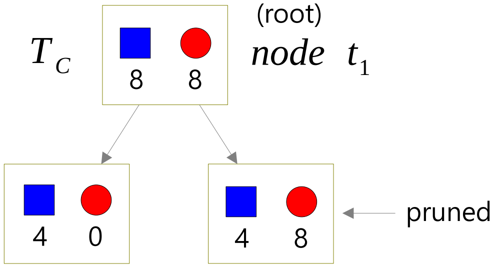


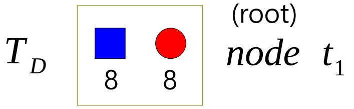
$$C(t_1)=\frac{8}{16}\times\frac{16}{16}+\alpha \qquad C(T_1)=\frac{0}{4}\times\frac{4}{16}+\frac{0}{6}\times\frac{6}{16}+\frac{2}{6}\times\frac{6}{16}+3\alpha \rightarrow \alpha^*=\frac{3}{16}$$

weakest link

$$C(t_3)=\frac{4}{12}\times\frac{12}{16}+\alpha \qquad C(T_3)=\frac{0}{6}\times\frac{6}{16}+\frac{2}{6}\times\frac{6}{16}+2\alpha \rightarrow \alpha^*=\frac{1}{8}$$

- **Step-3** : Prune the sub tree found in step-2, and find the sub tree with the smallest α. Repeat this process.





$$C(t_1)=\frac{8}{16}\times\frac{16}{16}+\alpha \qquad C(T_1)=\frac{0}{4}\times\frac{4}{16}+\frac{4}{12}\times\frac{12}{16}+2\alpha \rightarrow \alpha^*=\frac{1}{4}$$

- **final step** : Create a list of α candidates and determine the final α through cross-validation on the candidate trees.

$$\alpha=\left[0,\ \frac{1}{8},\ \frac{1}{8},\ \frac{1}{4}\right] \leftarrow$$ α candidates: α is getting bigger. The larger α, the smaller the tree.

(by definition: α ≥ 0)

- We added 0 to the list of α candidates. This is the case of not regularization.
- If 0≤α<1/8, T_A is the best, if α=1/8, T_B is best, if 1/8<α<1/4, $T_C$is best, and if α≥1/4, T_D is the best.
- Perform cross-validation on TA, TB, TC, and TD to select α with the smallest error. The tree corresponding to the selected α is the optimal tree.
- In this example, all subtrees have been pruned, but in a real tree, not all subtrees are pruned. Therefore, not all subtrees are candidates.

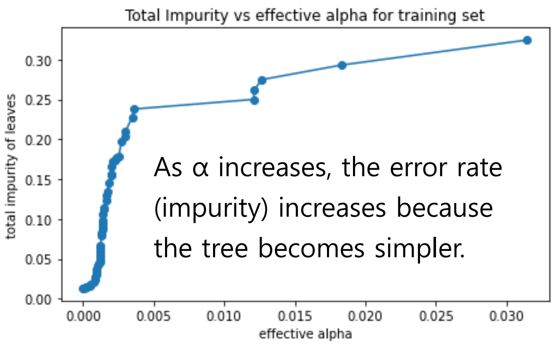
Coding practice: Cost Complexity Pruning (CCP) implementation

```
# [MXML-2-10]: 4.CART_CCP(titanic).py
# Reference: https://scikit-learn.org/stable/auto_examples/tree
#                  /plot_cost_complexity_pruning.html
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Read the Titanic data and simply preprocess it.
df = pd.read_csv('data/titanic.csv')
df['Age'].fillna(df['Age'].mean(), inplace=True)# Replace with average
df['Embarked'].fillna('N', inplace = True)      # Replace with 'N'
df['Sex'] = df['Sex'].factorize()[0]            # encoding
df['Embarked'] = df['Embarked'].factorize()[0]  # encoding
df.drop(['PassengerId','Name','Ticket','Cabin'], axis=1, inplace=True)

#  Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# 0       0       3    0  22.0      1      0   7.2500         0
# 1       1       1    1  38.0      1      0  71.2833         1
# 2       1       3    1  26.0      0      0   7.9250         0
# 3       1       1    1  35.0      1      0  53.1000         0
# 4       0       3    0  35.0      0      0   8.0500         0

# Generate training and test data
y = df['Survived']
x = df.drop('Survived', axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Apply Cost Complexity Pruning (CCP) and get the alpha-star list.
model = DecisionTreeClassifier()
path = model.cost_complexity_pruning_path(x_train, y_train)
```

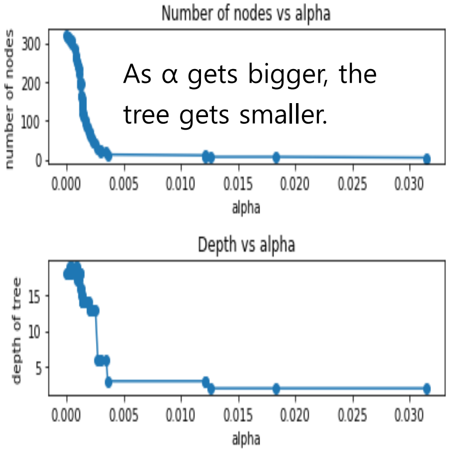
```
ccp_alpha = path.ccp_alphas[:-1]  # exclude the last one.
impurity = path.impurities[:-1]

# Observe impurity changes for alpha changes. As alpha increases,
# the penalty for |T| increases, resulting in simple trees and
# increased impurity (misclassification error).
plt.figure(figsize=(7,4))
plt.plot(ccp_alpha, impurity, marker='o')
plt.xlabel("effective alpha")
plt.ylabel("total impurity of leaves")
plt.title("Total Impurity vs effective alpha for training set")
plt.show()

# C(T) = R(T) + α|T|
# Create trees for each alpha in the alpha-list.
models = []
for i, alpha in enumerate(ccp_alpha):
    model = DecisionTreeClassifier(ccp_alpha=alpha)
    model.fit(x_train, y_train)
    models.append(model)
    print('%d) alpha = %.4f done.' % (i, alpha))

# You can see that as alpha increases,
# the number and depth of nodes decrease.
node_counts = [model.tree_.node_count for model in models]
depth = [model.tree_.max_depth for model in models]

fig, ax = plt.subplots(2, 1)
ax[0].plot(ccp_alpha, node_counts, marker="o")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
```

■ Coding practice: Cost Complexity Pruning (CCP) implementation

```
ax[1].plot(ccp_alpha, depth, marker="o")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()

# Among the candidate trees, find the one with the lowest
# misclassification rate on the test data. It's the same to find
# the tree with the highest score.
# Calculate the score of the tree (model) with alpha applied.
train_score = [model.score(x_train, y_train) for model in models]
test_score = [model.score(x_test, y_test) for model in models]

# Find the alpha that creates the tree with the highest score on
# the test data. This is the optimal alpha, and the tree is
# optimal.
i_max = np.argmax(test_score)
opt_alpha = ccp_alpha[i_max]
opt_model = models[i_max]

# Observe the change in score for the change in alpha.
plt.figure(figsize=(8,5))
plt.plot(ccp_alpha, train_score, marker='o', label='train')
plt.plot(ccp_alpha, test_score, marker='o', label='test')
plt.axvline(x=opt_alpha, ls='--', lw=1.0)
plt.legend()
plt.xlabel('alpha')
plt.ylabel('tree score')
plt.show()

# Evaluate the performance of the final tree.
print('Accuracy of test data = %.4f' % opt_model.score(x_test,\
        y_test))
print('Optimal alpha = %.8f' % opt_alpha)

Accuracy of test data = 0.8430,   Optimal alpha = 0.00273707
```



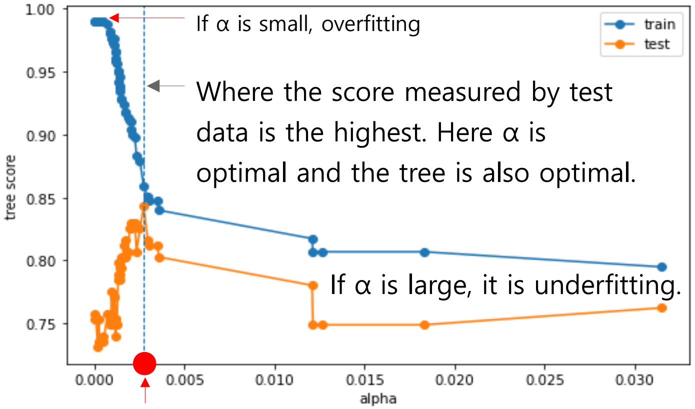
As α increases, the error rate (impurity) increases because the tree becomes simpler.



As α gets bigger, the tree gets smaller.

```
0) alpha = 0.0000 done.
1) alpha = 0.0000 done.
2) alpha = 0.0001 done.
3) alpha = 0.0002 done.
4) alpha = 0.0002 done.
5) alpha = 0.0002 done.
...
36) alpha = 0.0018 done.
37) alpha = 0.0020 done.
38) alpha = 0.0020 done.
39) alpha = 0.0020 done.
40) alpha = 0.0021 done.
41) alpha = 0.0022 done.
42) alpha = 0.0023 done.
43) alpha = 0.0025 done.
44) alpha = 0.0027 done.
45) alpha = 0.0030 done.
46) alpha = 0.0030 done.
47) alpha = 0.0035 done.
48) alpha = 0.0036 done.
49) alpha = 0.0121 done.
50) alpha = 0.0121 done.
51) alpha = 0.0126 done.
52) alpha = 0.0183 done.
53) alpha = 0.0315 done.
54) alpha = 0.1529 done.
```

Score (accuracy) of training data and test data for α change



If α is small, overfitting

Where the score measured by test data is the highest. Here α is optimal and the tree is also optimal.

If α is large, it is underfitting.

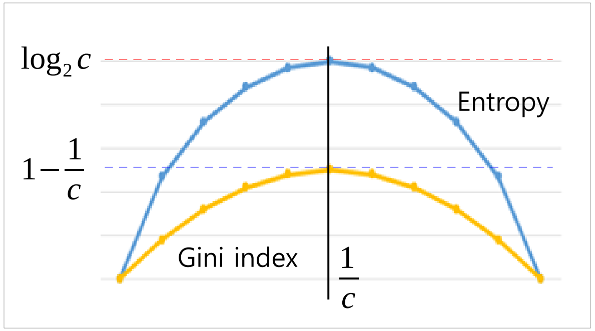
α value when C(T) is minimum.

[MXML-2-10] Machine Learning / 2. Decision Tree – CART (Classification)

MX-AI

Multi-class classification

- For multiple classes, the entropy and Gini index can be calculated as follows, so the tree can be created using information gain in the same way as the previous process. (c : the number of class).



- c = the number of class
- If c > 2, the entropy can be greater than 1.0, and the Gini index can also be greater than 0.5.

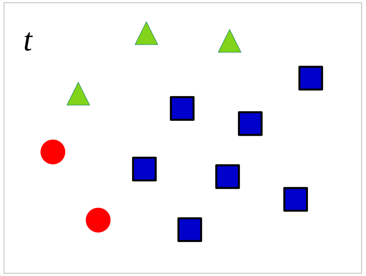
- Entropy

$$H(t)=-\sum_{i=1}^{c} p(i|t)\log_2 p(i|t)$$

- Gini index

$$G(t)=\sum_{i=1}^{c} p(i|t)(1-p(i|t))=1-\sum_{i=1}^{c} p(i|t)^2$$

- Example



$$H(t) = -(\frac{7}{12}\times\log_2\frac{7}{12}+\frac{3}{12}\times\log_2\frac{3}{12}+\frac{2}{12}\times\log_2\frac{2}{12})$$

$$=1.384$$

$$G(t) = 1-(\frac{7}{12})^2-(\frac{3}{12})^2-(\frac{2}{12})^2=0.569$$

- Maximum if there are (n/c) pieces of data per class inside a node.

$$p_1 = p_2 = \dots p_c = \frac{n/c}{n} = \frac{1}{c}$$

$$H(t)_{max} = -\sum_{i=1}^{c}\frac{1}{c}\cdot\log_2\frac{1}{c} = \log_2 c$$

$$G(t)_{max} = 1-\sum_{i=1}^{c}\frac{1}{c^2} = 1-\frac{1}{c}$$

■ Coding practice: Multiclass classification

```
# [MXML-2-10]: 5.CART(multiclass).py
# Multiclass classification test code
import numpy as np
from sklearn.datasets import load_iris
from MyDTreeClassifier import MyDTreeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Load iris dataset
# x: data, the number of samples=150, the number of features=4
# y: target data with class (0,1,2)
x, y = load_iris(return_X_y=True)

# Generate training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Model-1: using our model - refer to [MXML-2-07] video
model1 = MyDTreeClassifier(max_depth=3)
model1.fit(x_train, y_train)

# Estimate the class of validation date.
y_pred1 = model1.predict(x_test)

# Measure the accuracy for validation data
accuracy1 = (y_test == y_pred1).mean()
print('\nAccuracy of Model-1 = {:.3f}'.format(accuracy1))

# Model-2: using sklearn
model2 = DecisionTreeClassifier(max_depth=3)
model2.fit(x_train, y_train)
```

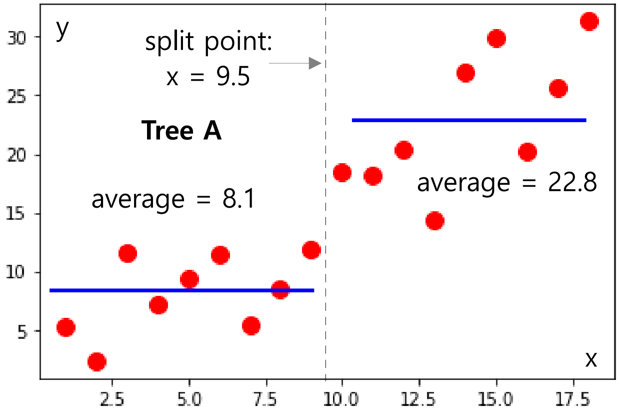
```
# Estimate the class of validation date.
y_pred2 = model2.predict(x_test)

# Measure the accuracy for validation data
accuracy2 = (y_test == y_pred2).mean()
print('Accuracy of Model-2 = {:.3f}'.format(accuracy2))

print("\nModel-1: y_pred1")
print(y_pred1)
print("\nModel-2: y_pred2")
print(y_pred2)
```

```
Result:

Accuracy of Model-1 = 0.947
Accuracy of Model-2 = 0.947

Model-1: y_pred1
[0 1 1 0 1 2 0 2 0 1 0 1 2 2 0 1 1 1 0 0 1 0 2 2 0 2 0 1 0 2 2
 0 1 1 0 2 1 2]

Model-2: y_pred2
[0 1 1 0 1 2 0 2 0 1 0 1 2 2 0 1 1 1 0 0 1 0 2 2 0 2 0 1 0 2 2
 0 1 1 0 2 1 2]
```

Regression tree

- Regression trees use mean square error (MSE) instead of entropy or Gini index. Select the split point with the smallest MSE.

x	y
12.0	20.34
3.0	11.6
10.0	18.49
16.0	20.27
5.0	9.43
18.0	31.34
1.0	5.28
15.0	29.84
2.0	2.33
8.0	8.52
11.0	18.13
7.0	5.40
13.0	14.42
4.0	7.20
9.0	11.80
17.0	25.60
6.0	11.38
14.0	26.96

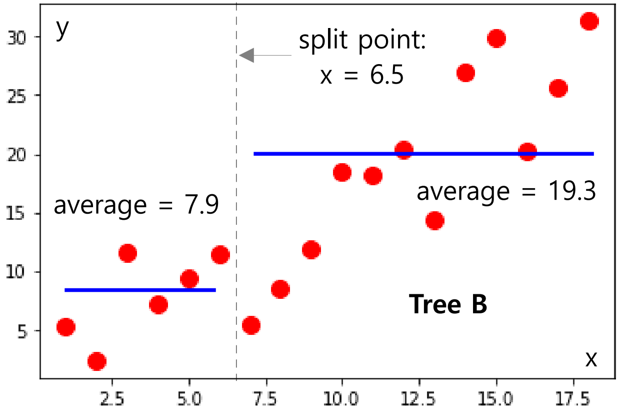


split point:
x = 9.5

Tree A

average = 8.1

average = 22.8



split point:
x = 6.5

average = 7.9

average = 19.3

Tree B

$$MSE=\frac{1}{N}\sum_{i=1}^{N}(y_i-\bar{y})^2$$ ← Mean-squared error for each node

It is the same as Variance.

$$MSE_{\text{left}}=\frac{1}{9}\times((5.28-8.1)^2+(2.33-8.1)^2+\ldots+(11.80-8.1)^2)=9.777$$

$$MSE_{\text{right}}=\frac{1}{9}\times((18.49-22.8)^2+(18.13-22.8)^2+\ldots+(31.34-22.8)^2)=30.077$$

$$MSE_A=9.777\times\frac{9}{18}+30.077\times\frac{9}{18}=19.93$$ ← weighted average

$$MSE_{\text{left}}=\frac{1}{6}\times((5.28-7.9)^2+(2.33-7.9)^2+\ldots+(11.38-7.9)^2)=11.086$$

$$MSE_{\text{right}}=\frac{1}{12}\times((5.40-19.3)^2+(8.52-19.3)^2+\ldots+(31.34-19.3)^2)=62.327$$

$$MSE_B=11.086\times\frac{6}{18}+62.327\times\frac{12}{18}=45.24$$ ← weighted average

$MSE_A < MSE_B$: Tree A is better.

[MXML-2-11] Machine Learning / 2. Decision Tree – CART (Regression)

■ Coding practice: Regression tree - MyDTreeRegressor from scratch (Please refer to [MXML-2-07] for classification)

```python
# MyDTreeRegressor.py
import numpy as np
from collections import Counter
import copy

# Implement the Decision Tree Regressor using binary tree.
class MyDTreeRegressor:
    def __init__(self, max_depth):
        self.max_depth = max_depth
        self.estimator1 = dict() # tree result-1
        self.estimator2 = dict() # tree result-2
        self.feature = None      # feature value.
        self.target = None       # target value.

    # Split a node into left and right node.
    # Find the point with the smallest MSE and split the node
    # at that point. did: data index on the leaf node.
    def node_split(self, did):
        n = did.shape[0]

        # Split the node into all candidates for all features
        # and find the best feature and the best split point
        # with the smallest MSE. fid: feature_id
        min_mse = 999999
        for fid in range(self.feature.shape[1]):
            # feature data to be split
            x_feat = self.feature[did, fid].copy()

            # split x_feat using the best feature and the best
            # split point. Note: The code below is inefficient
            # because it sorts x_feat every time it is split.
            # Future improvements are needed.

            # remove duplicates of x_feat and sort
            # in ascending order
            x_uniq = np.unique(x_feat)
```

```python
            # list up all the candidates, which are the midpoints of
            # adjacent data.
            s_point = [np.mean([x_uniq[i-1], x_uniq[i]]) \
                       for i in range(1, len(x_uniq))]

            # len(s_point) > 1: Calculate MSE for all candidates,
            # and find the candidate with the smallest MSE.
            # len(s_point) < 1: skip the for-loop. x_feat either has
            # only one data or all have the same value. No need to split.

            for p in s_point:
                # split x_feat into the left and the right node.
                left = did[np.where(x_feat <= p)[0]]
                right = did[np.where(x_feat > p)[0]]

                # calculate MSE after splitting. MSE is the same as
                # variance in this case.
                l_mse = self.target[left].var()
                r_mse = self.target[right].var()
                mse = l_mse * (left.shape[0] / n) + \
                      r_mse * (right.shape[0] / n)

                # find where the MSE is smallest.
                if mse < min_mse:
                    min_mse = mse
                    b_fid = fid        # best feature id
                    b_point = p        # best split point
                    b_left = left      # data index on the left node.
                    b_right = right    # data index on the right node.

        if min_mse < 999999.:   # split
            return {'fid':b_fid, 'split_point':b_point, \
                    'left':b_left, 'right':b_right}
        else:
            return  None        # No split
```

- Coding practice: Regression tree - MyDTreeRegressor from scratch (Please refer to [MXML-2-07] for classification)

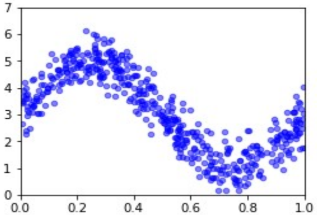
```
# Create a binary tree using recursion
def recursive_split(self, node, curr_depth):
    left = node['left']
    right = node['right']

    # exit recursion
    if curr_depth >= self.max_depth:
        return

    # process recursion
    s = self.node_split(left)
    if isinstance(s, dict):   # split to the left done.
        node['left'] = s
        self.recursive_split(node['left'], curr_depth+1)

    s = self.node_split(right)
    if isinstance(s, dict):   # split to the right done.
        node['right'] = s
        self.recursive_split(node['right'], curr_depth+1)

# Change the data in the leaf node to average value.
def update_leaf(self, d):
    if isinstance(d, dict):
        for key, value in d.items():
            if key == 'left' or key == 'right':
                rtn = self.update_leaf(value)
                if rtn[0] == 1:      # leaf node
                    d[key] = rtn[1]
        return 0, 0  # the first 0 indicates this is not
                     # a leaf node.
    else:            # leaf node
        # the first 1 indicates this is a leaf node.
        return 1, self.target[d].mean()
```

```
# create a tree using training data, and return the result of
# the tree. x : feature data, y: target data
def fit(self, x, y):
    self.feature = x
    self.target = y

    # Initially, the root node holds all data indices.
    root = self.node_split(np.arange(x.shape[0]))
    if isinstance(root, dict):
        self.recursive_split(root, curr_depth=1)

    # tree result-1. Every leaf node has data indices.
    self.estimator1 = root

    # tree result-2. Every leaf node has average value.
    self.estimator2 = copy.deepcopy(self.estimator1)
    self.update_leaf(self.estimator2)   # tree result-2
    return self.estimator2

# Estimate the target value of a test data.
def x_predict(self, p, x):
    if x[p['fid']] <= p['split_point']:
        if isinstance(p['left'], dict):  # recursion if not leaf.
            return self.x_predict(p['left'], x)  # recursion
        else:            # return the value in the leaf, if leaf.
            return p['left']
    else:
        if isinstance(p['right'], dict):# recursion if not leaf.
            return self.x_predict(p['right'], x) # recursion
        else:            # return the value in the leaf, if leaf.
            return p['right']

# Estimate the target class of x_test.
def predict(self, x_test):
    p = self.estimator2    # predictor
    y_pred = [self.x_predict(p, x) for x in x_test]
    return np.array(y_pred)
```

MX-AI

- Coding practice: Regression tree: Compare with sklearn's DecisionTreeClassifier

```python
# [MXML-02-11] 6.CART(regression).py
import numpy as np
from MyDTreeRegressor import MyDTreeRegressor
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn import tree
import pprint

# Plot the training data and draw the estimated curve.
def plot_prediction(x, y, x_test, y_pred, title):
    plt.figure(figsize=(6,4))
    plt.scatter(x, y, c='blue', s=20, alpha=0.5, \
                label='train data')
    plt.plot(x_test, y_pred, c='red', lw=2.0.\
             label='prediction')
    plt.xlim(0, 1)
    plt.ylim(0, 7)
    plt.legend()
    plt.title(title)
    plt.show()

# Generate nonlinear data for regression testing.
def noisy_sine_data(n, s):
   rtn_x, rtn_y = [], []
   for i in range(n):
       x= np.random.random()
       y= 2.0*np.sin(2.0*np.pi*x)+np.random.normal(0.0, s) + 3.0
       rtn_x.append(x)
       rtn_y.append(y)
   return np.array(rtn_x).reshape(-1,1), np.array(rtn_y)
```

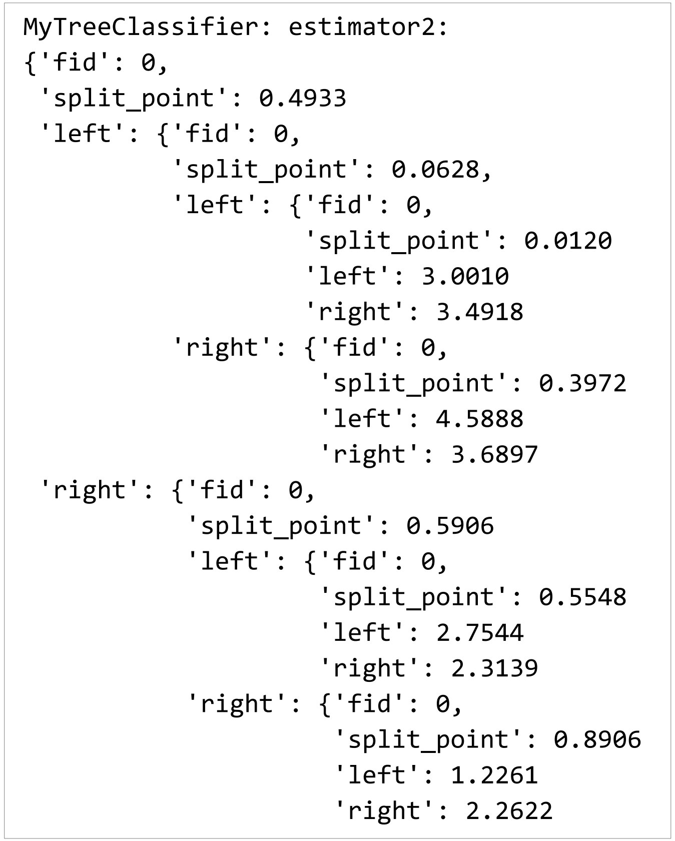


```python
# Create training and test data
x_train, y_train = noisy_sine_data(n=500, s=0.5)
x_test = np.linspace(0, 1, 50).reshape(-1, 1)

depth = 3
my_model = MyDTreeRegressor(max_depth = depth)
my_model.fit(x_train, y_train)
my_pred = my_model.predict(x_test)

# Plot the training data and draw the estimated curve.
plot_prediction(x_train, y_train, x_test, my_pred,
               'MyDTreeRegressor')

# Compare with sklearn's DecisionTreeRegressor() results.
sk_model = DecisionTreeRegressor(max_depth = depth)
sk_model.fit(x_train, y_train)
sk_pred = sk_model.predict(x_test)

# Plot the training data and draw the estimated curve.
plot_prediction(x_train, y_train, x_test, sk_pred,
                'DecisionTreeRegressor')

# Compare trees created by the two models.
print('\nMyDTreeRegressor: estimator2:')
pprint.pprint(my_model.estimator2, sort_dicts=False)

plt.figure(figsize=(12,7))
tree.plot_tree(sk_model)
plt.show()
```

[MXML-2-11] Machine Learning / 2. Decision Tree – CART (Regression)

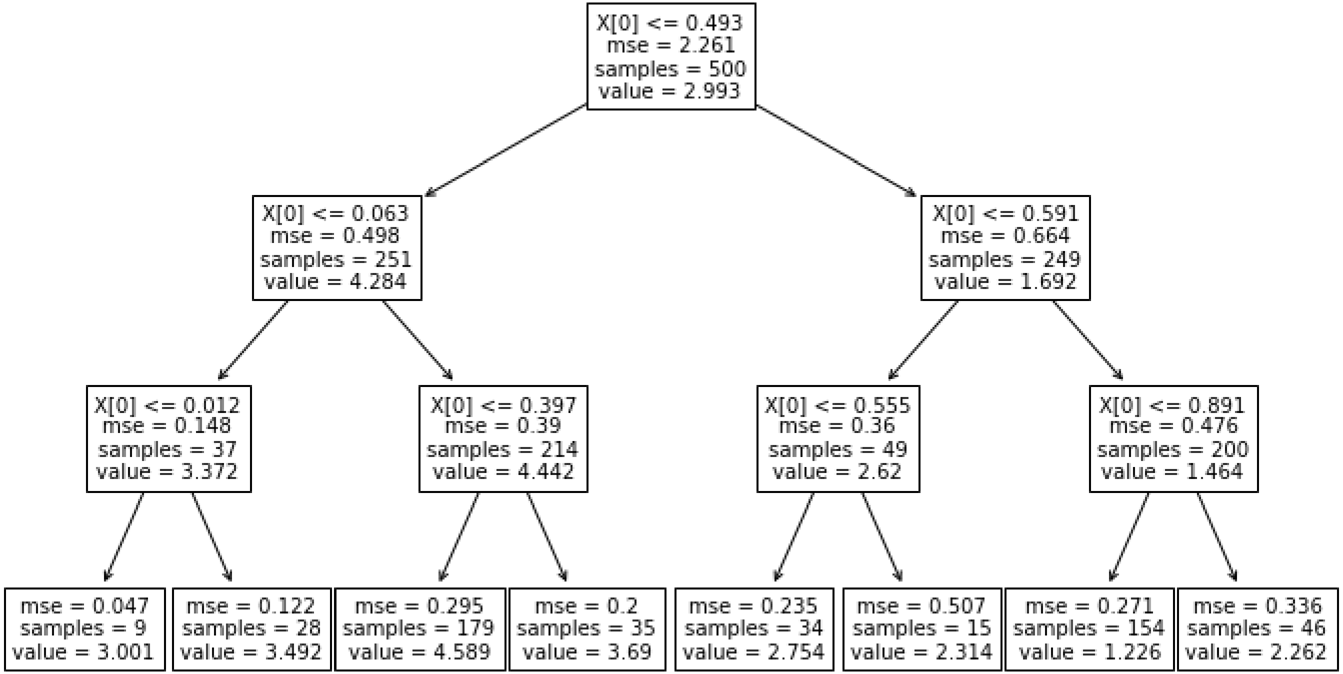
Coding practice: Regression tree: Compare with sklearn’s DecisionTreeClassifier

- max_depth = 3. The two results agree well.
  - The result of MyDTreeRegerssor



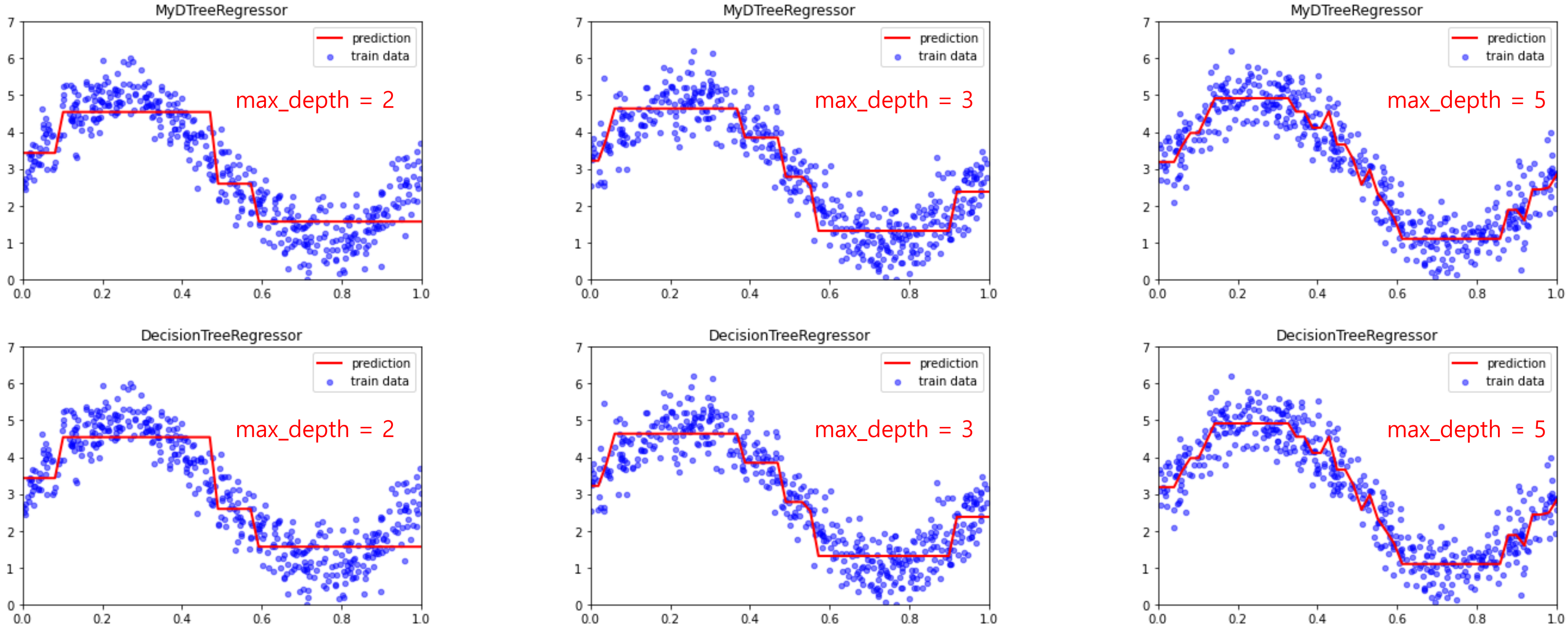
```
MyTreeClassifier: estimator2:
{'fid': 0,
 'split_point': 0.4933
 'left': {'fid': 0,
          'split_point': 0.0628,
          'left': {'fid': 0,
                   'split_point': 0.0120
                   'left': 3.0010
                   'right': 3.4918
          'right': {'fid': 0,
                    'split_point': 0.3972
                    'left': 4.5888
                    'right': 3.6897
 'right': {'fid': 0,
           'split_point': 0.5906
           'left': {'fid': 0,
                    'split_point': 0.5548
                    'left': 2.7544
                    'right': 2.3139
           'right': {'fid': 0,
                     'split_point': 0.8906
                     'left': 1.2261
                     'right': 2.2622
```

- The result of sklearn’s DecisionTreeRegressor



Coding practice: Regression tree: Compare with sklearn's DecisionTreeClassifier

- The two results match well.
- It works as a non-linear regression rather than a linear regression.
- The shallower the tree, the more likely it is to be underfitting, and the deeper the tree, the more likely it is to be overfitting.
- As with classification, pruning is necessary to prevent overfitting. Cost Complexity Pruning (CCP) can also be applied for regression.
- It can also be applied when there are multiple features (multiple regression).


MyDTreeRegressor
MyDTreeRegressor
MyDTreeRegressor
prediction
train data
max_depth = 2
max_depth = 3
max_depth = 5
DecisionTreeRegressor
DecisionTreeRegressor
DecisionTreeRegressor
max_depth = 2
max_depth = 3
max_depth = 5