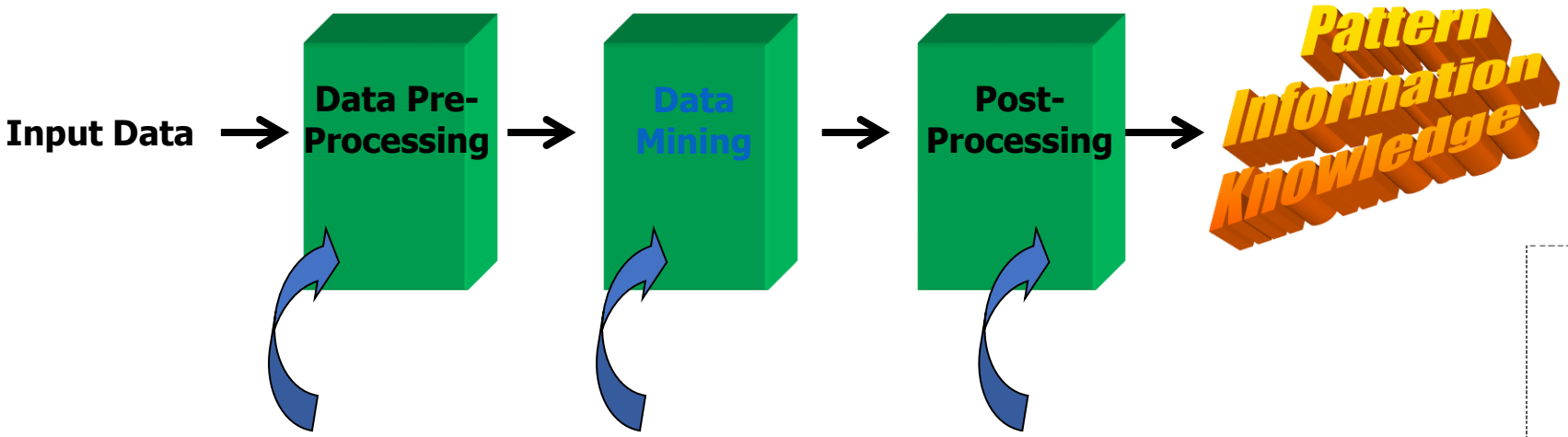
클래스 분류

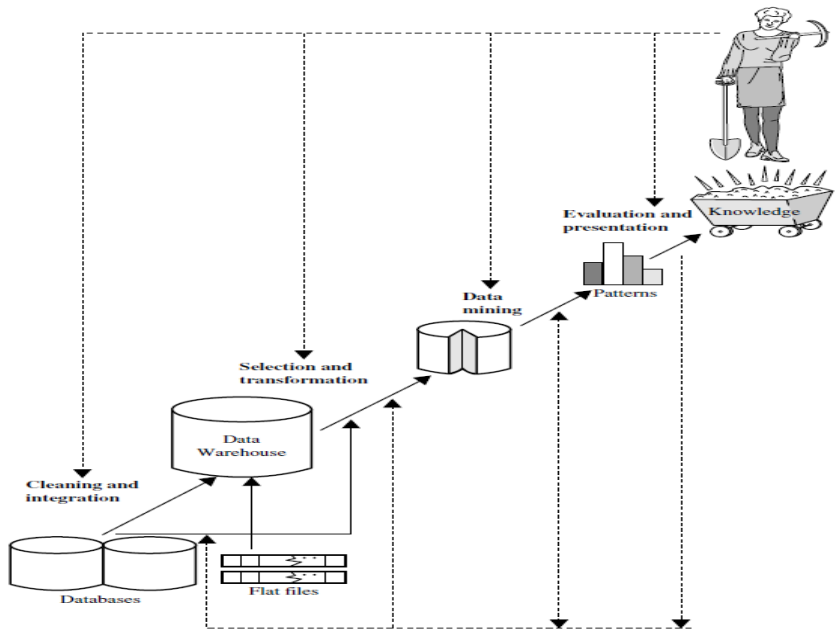
과목명: 데이터사이언스

AI융합학부 박건우

Recap: 지식 발견 절차

넓은 의미의 데이터 마이닝은 지식 발견 절차 전체를 의미하며, 좁은 의미의 데이터 마이닝은 정제된 데이터로부터 자동화된 절차 및 알고리즘을 통해 특정 결과를 뽑는 세부 단계를 말한다.





Contents

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

Classification: Basic Concepts

- **Classification (분류)**: a model or classifier is constructed to predict class (categorical) labels
- **Supervised learning** (지도 학습)
  - Supervision (지도): The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations
  - New data is classified based on the training set
- **Unsupervised learning** (비지도 학습)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data (clustering)

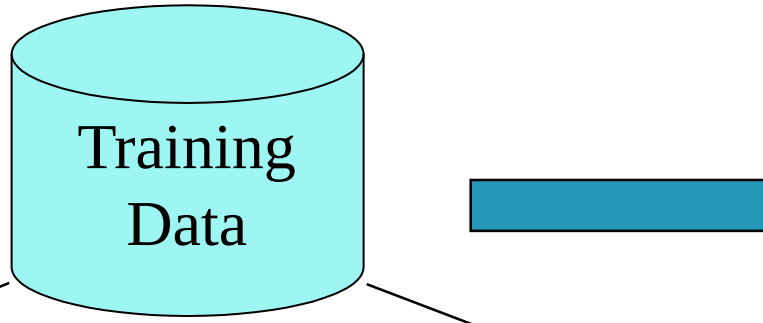
Prediction Problems:
Classification vs. Numeric Prediction

- **Classification**
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
  - **Numeric Prediction (or Regression)** 과 구분됨
    - models continuous-valued functions -- the trained model is called a predictor
    - i.e., predicts unknown or missing values
- Typical applications: Credit/loan approval, Medical diagnosis, Fraud detection

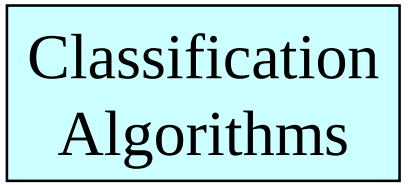
Classification – A Two-Step Process

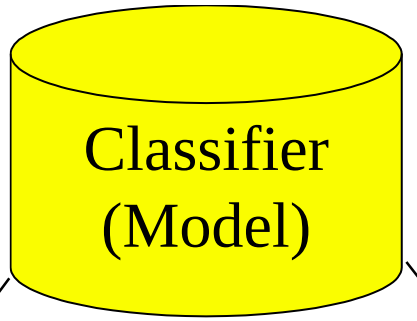
- **Model construction**: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the **class label attribute**
  - The set of tuples used for model construction is **training set**
  - The model is represented as classification rules, decision trees, or mathematical formulae
- **Model usage**: for classifying future or unknown objects
  - **Estimate accuracy** of the model
    - The known label of test sample is compared with the classified result from the model
    - **Accuracy** rate is the percentage of test set samples that are correctly classified by the model
    - **Test set** is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to **classify new data**
- Note: If the test set is used to select models, it is called validation (test) set

Process (1): Model Construction



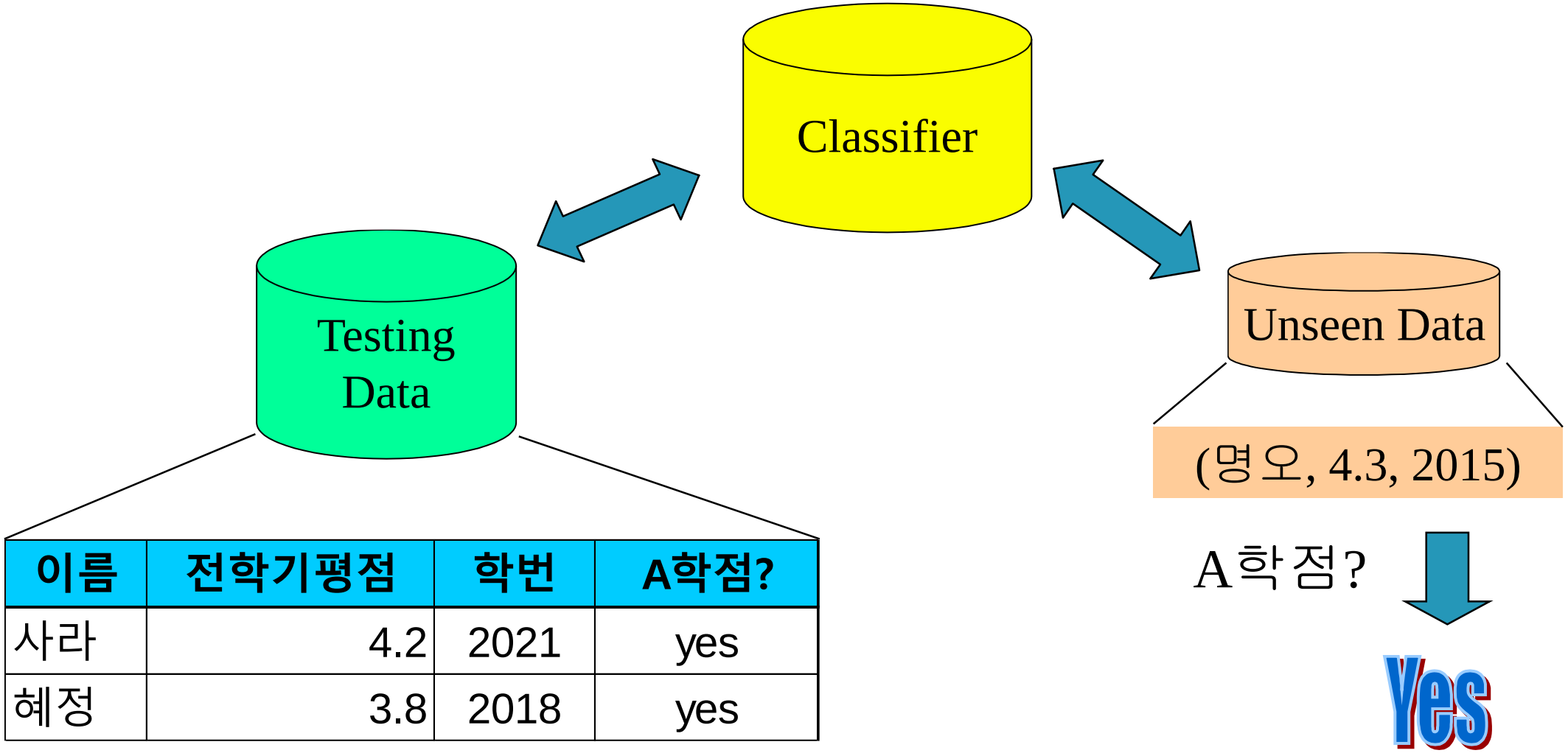
이름	전학기평점	학번	A학점?
동은	4	2021	no
연진	4.4	2021	yes
재준	2.8	2018	yes
여정	4.5	2019	yes
도영	3.5	2017	no
현남	3.3	2021	no





IF 전학기평점 > 4
THEN A학점? = 'yes'

Process (2): Using the Model in Prediction



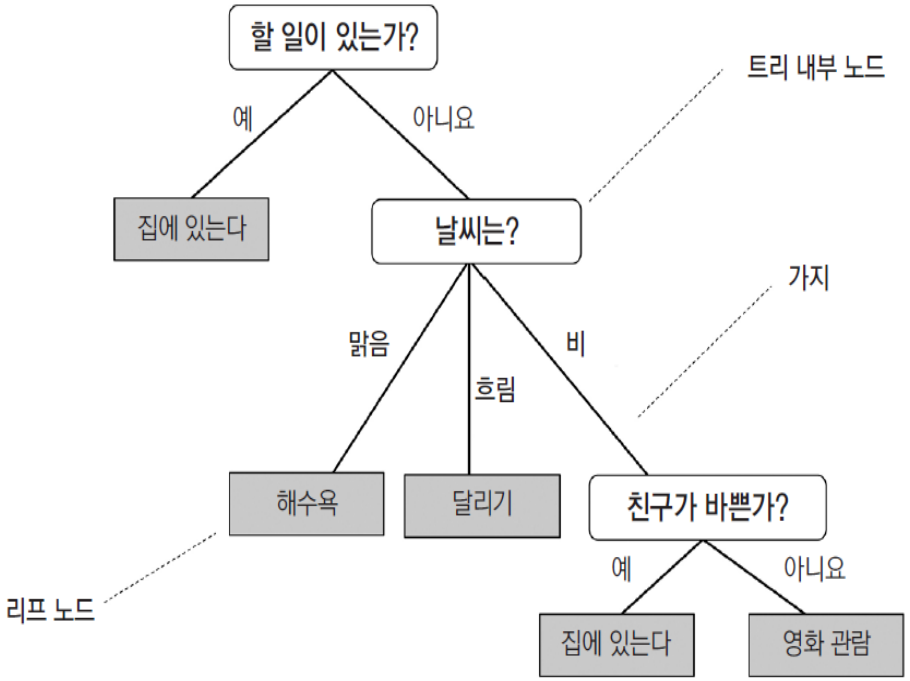
이름	전학기평점	학번	A학점?
사라	4.2	2021	yes
혜정	3.8	2018	yes

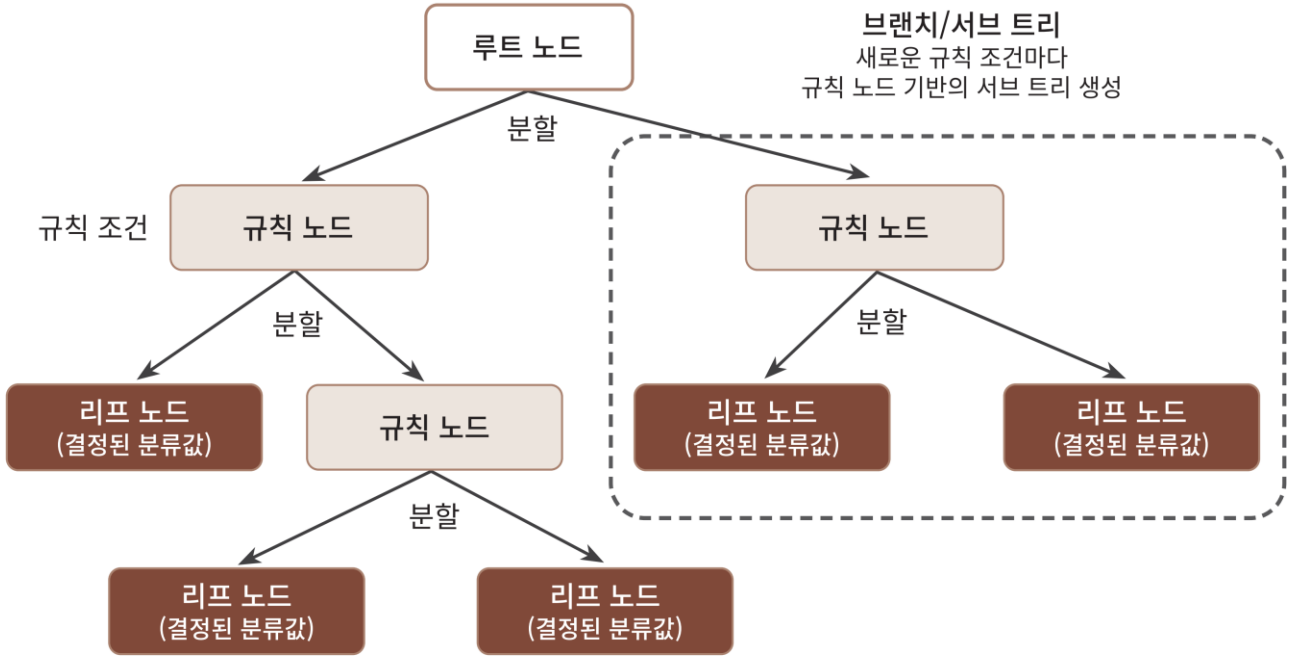
Contents

- Classification: Basic Concepts
- **Decision Tree Induction**
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

결정 트리 (Decision Tree)

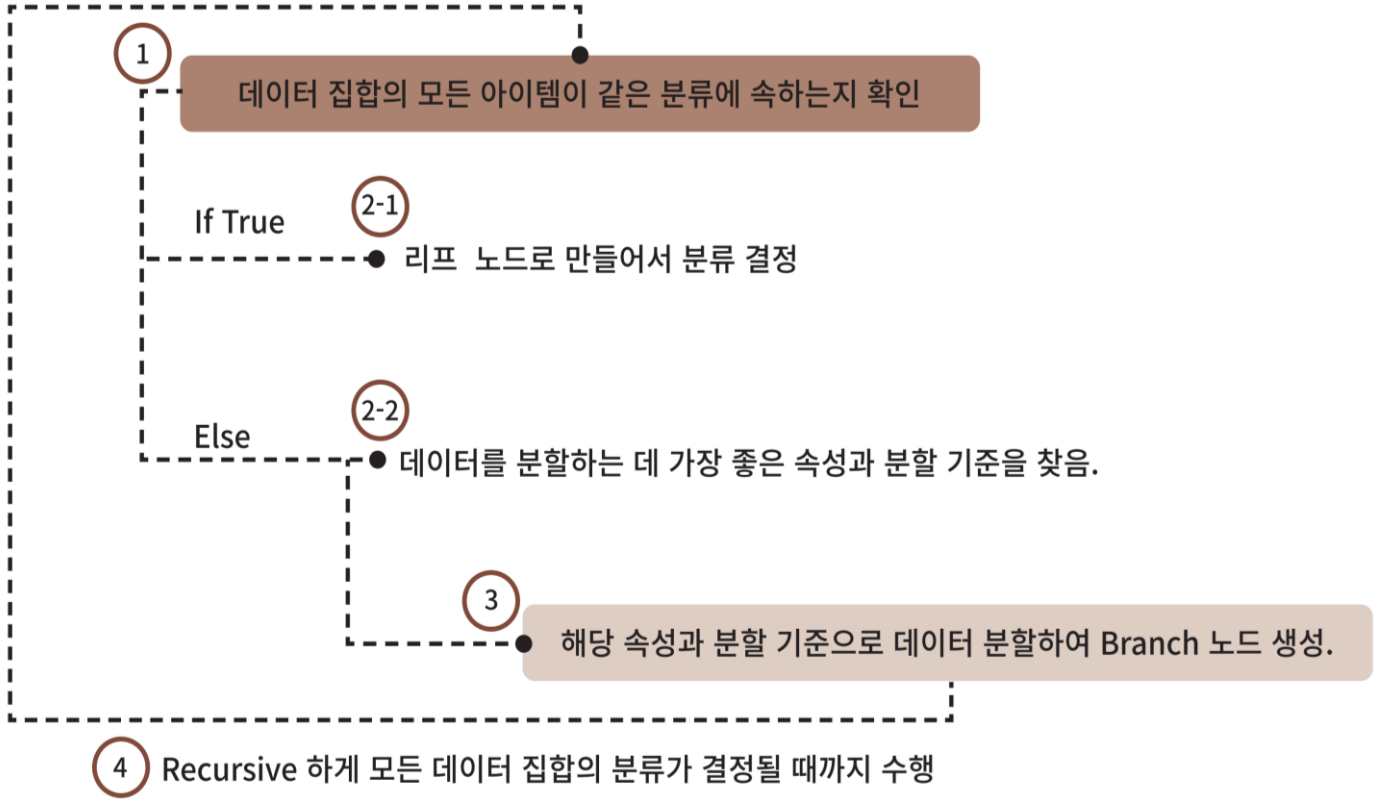
- 결정 트리는 가장 기본적인 분류 모델 중 하나로, 데이터에서 각 특징에 대한 if/else 기반의 연속된 규칙을 학습(Learning)을 통해 찾아내며, 트리 형태로 표현한다.
- 일련의 질문에 대한 결정을 통해 데이터를 분해하는 모델로 생각할 수 있다.





학습 알고리즘

- 가장 좋은 분할을 만드는 특징을 선택해 데이터를 분할하는 규칙 노드로 만든다.
- 리프 노드를 만들 수 있을 때 까지 그 과정을 반복한다.



학습 알고리즘

- 그럼 최적의 분할을 만드는 특징을 어떻게 선택할까?

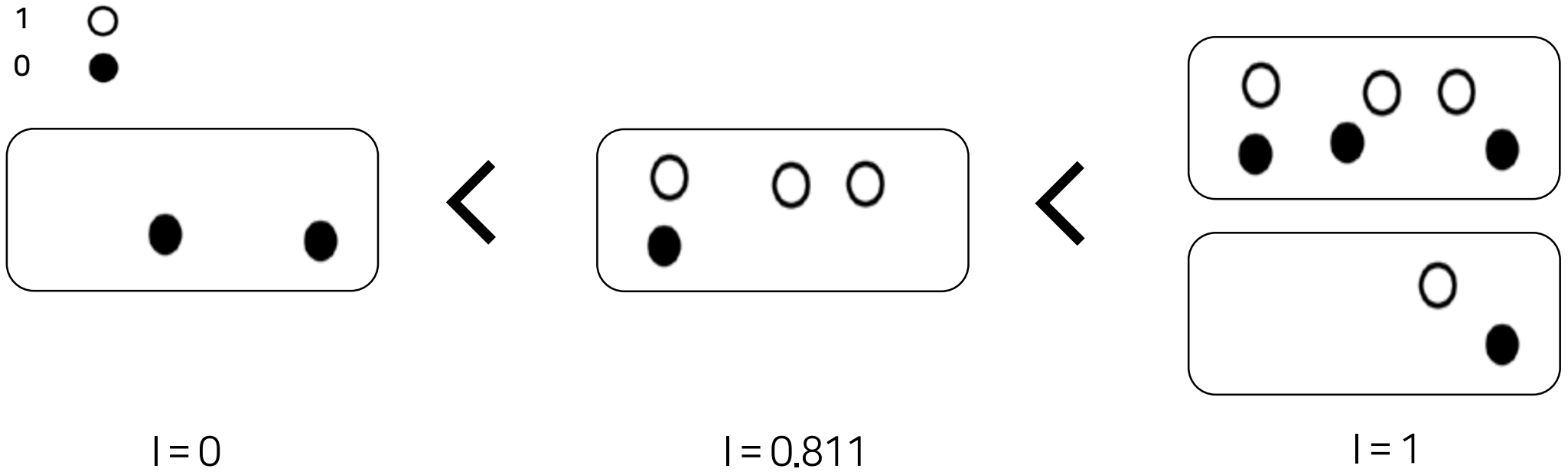
$$IG\left(D_p, f\right) = I\left(D_p\right) - \frac{N_{left}}{N_p} I\left(D_{left}\right) - \frac{N_{right}}{N_p} I\left(D_{right}\right)$$

정보 이득 (Information Gain)을 최대화하는 특징을 선택한다.:
부모 노드의 불순도와 자식 노드의 불순도 평균의 차이

- f : 분할에 사용할 특징
- D_p, D_{left}, D_{right}: 각각 부모 노드, 왼쪽 자식 노드, 오른쪽 자식 노드를 방문하는 데이터셋
- N_p, N_{left}, N_{right}: 각각 부모 노드, 왼쪽 자식 노드, 오른쪽 자식 노드를 방문하는 샘플 수
- I : 불순도(impurity) 지표

학습 알고리즘

- 불순도(impurity)는 주어진 노드에 속하는 샘플들의 클래스 분포가 어느 한 쪽으로 쏠릴 경우 낮은 값을 갖고, 균등한 분포를 지니면 높은 값을 갖는다.

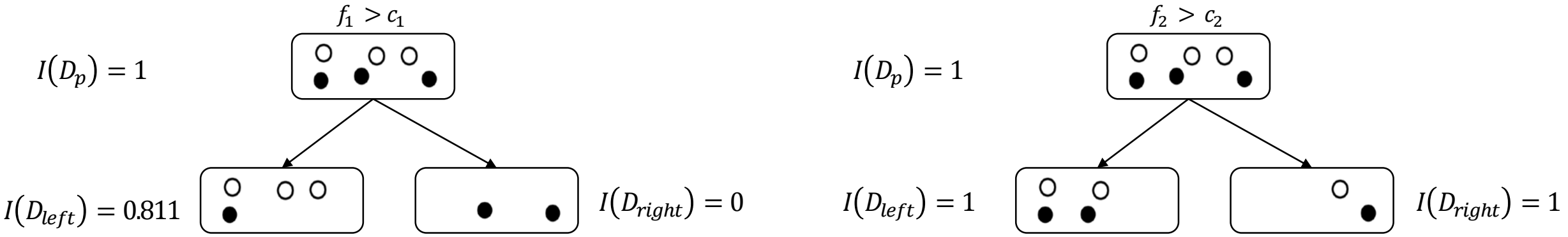


학습 알고리즘

- 부모 노드와 자식 노드의 불순도 차이를 크게 만드는 특징을 선택했을 때 정보 이득을 높일 수 있기 때문에, 결정 트리는 클래스 분포가 한 쪽으로 쏠리도록 하는 특징을 부모 노드의 규칙으로 선택한다.

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

1 ○
0 ●



불순도 지표

- 이진 결정 트리에 널리 사용되는 불순도 지표 (또는 분할 조건)는 세 가지:
  (1) 엔트로피(entropy)
  (2) 지니 불순도(Gini impurity)
  (3) 분류 오차(classification error)

불순도 지표 (1) 엔트로피

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

- $p(i|t)$: 특정 노드 t 에서 클래스 i 에 속한 샘플 비율
- Examples
  - 균등 분포 (0.5:0.5); $p(i=1|t) = 0.5$ $p(i=0|t) = 0.5$; 따라서, $-\log_2 0.5 = 1$
  - 불균등 분포 (1:0); $p(i=1|t) = 1$ $p(i=0|t) = 0$; 따라서, 0
  - 불균등 분포 (0.9:0.1); $-0.9(\log_2 0.9 = -0.15) - 0.1(\log_2 0.1 = -3.32) = 0.467$
- 엔트로피 조건을 트리의 상호 의존 정보를 최대화하는 것으로 이해할 수 있음

불순도 지표 (2) 지니 계수

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

- 잘못 분류될 확률을 최소화하기 위한 기준으로 이해할 수 있음
- 엔트로피와 유사하게, 클래스 분포가 완전한 균형을 이룰 때, 즉 완벽하게 클래스가 섞여 있을 때, 최대값을 가진다.
- Examples
  - 균등 분포 (0.5:0.5); $1 - \sum_{i=1}^{c} 0.5^2 = 1 - (0.25 + 0.25) = 0.5$
  - 불균등 분포 (1:0); $1 - 1^2 - 0^2 = 0$
  - 불균등 분포 (0.9:0.1); $1 - 0.9^2 - 0.1^2 = 1 - 0.81 - 0.01 = 0.18$

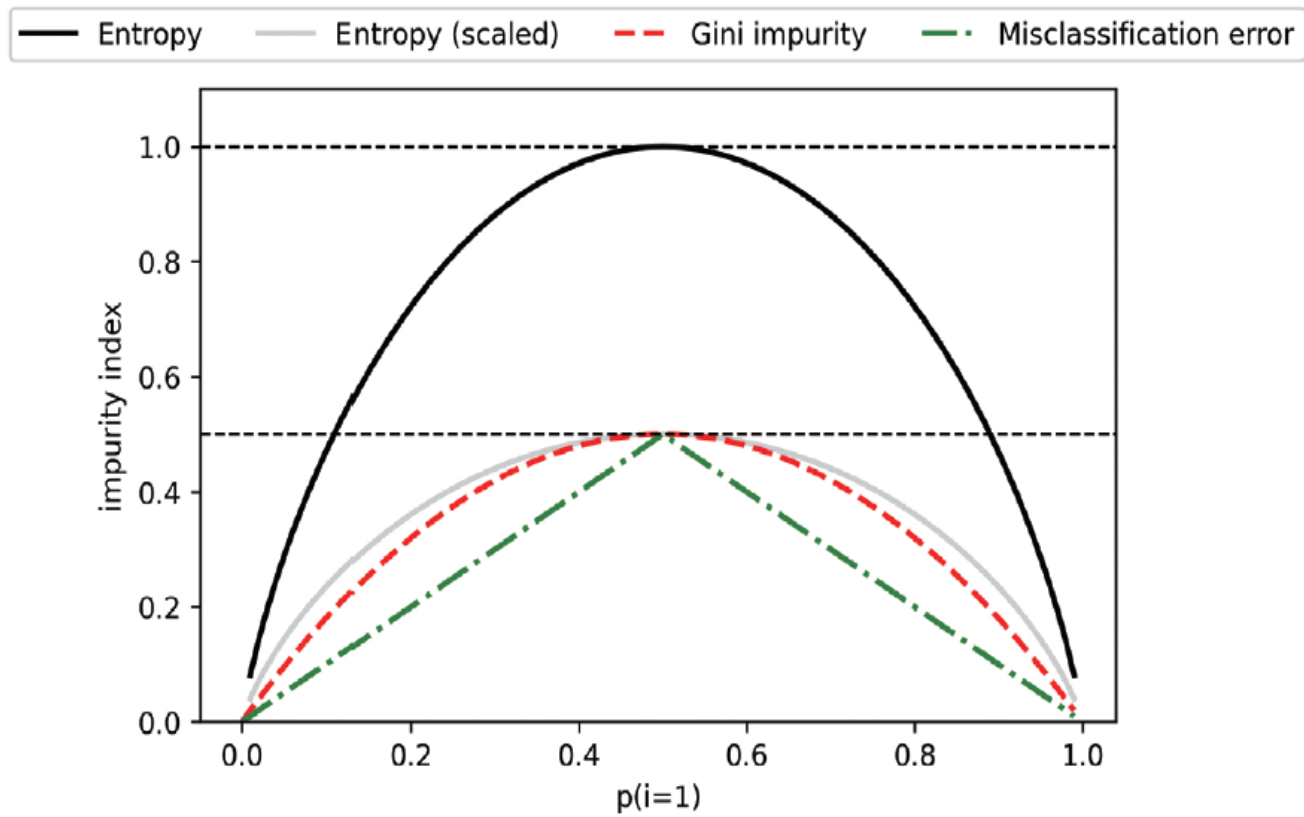
불순도 지표 (3) 분류 오차

- 분류 오차를 불순도 지표로 사용할 수도 있다. $I_E = 1 - \max\left\{ p\left(i \mid t\right)\right\}$
- 가지치기에는 좋은 기준이나, 결정 트리에서 분할 기준을 결정할 땐 권장하지 않는다.

결정 트리 불순도 지표 비교

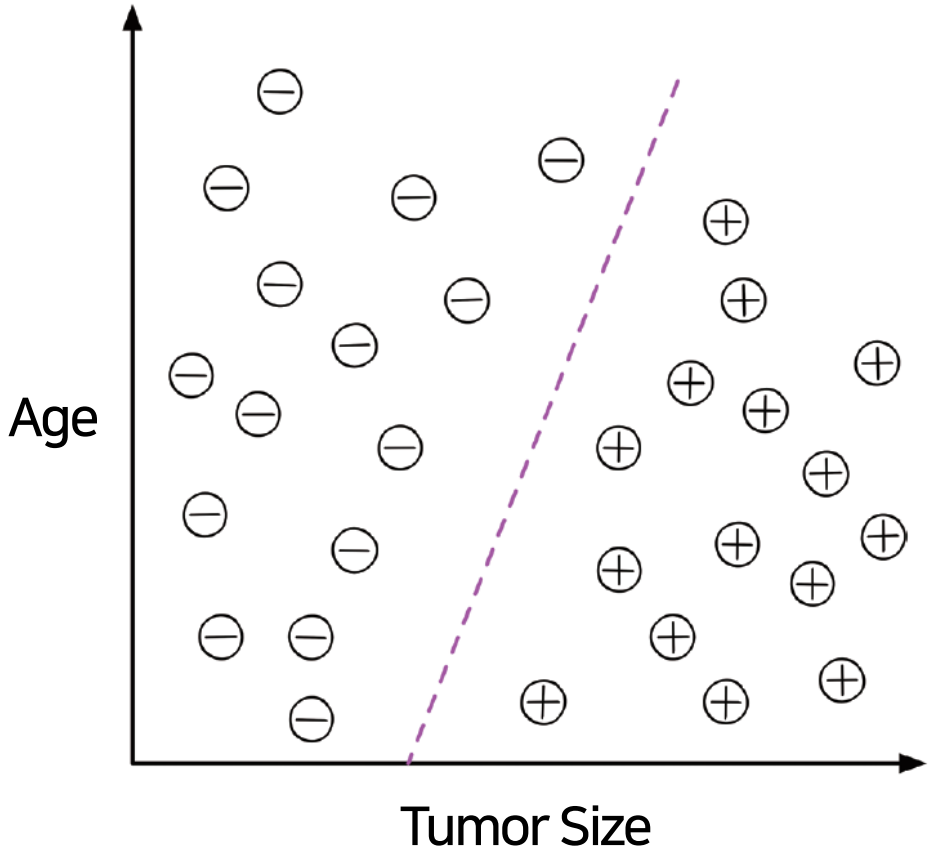
```
import matplotlib.pyplot as plt
import numpy as np

def gini(p):
    return p * (1 - p) + (1 - p) * (1 - (1 - p))

def entropy(p):
    return - p * np.log2(p) - (1 - p) * np.log2((1 - p))

def error(p):
    return 1 - np.max([p, 1 - p])
x = np.arange(0.0, 1.0, 0.01)

ent = [entropy(p) if p != 0 else None for p in x]
sc_ent = [e * 0.5 if e else None for e in ent]
err = [error(i) for i in x]

fig = plt.figure()
ax = plt.subplot(111)
for i, lab, ls, c, in zip([ent, sc_ent, gini(x), err],
                          ['Entropy', 'Entropy (scaled)',
                           'Gini impurity',
                           'Misclassification error'],
                          ['-', '-', '--', '-.'],
                          ['black', 'lightgray',
                           'red', 'green', 'cyan']):
    line = ax.plot(x, i, label=lab,
                   linestyle=ls, lw=2, color=c)
```

```
ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15),
          ncol=5, fancybox=True, shadow=False)

ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')
ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')
plt.ylim([0, 1.1])
plt.xlabel('p(i=1)')
plt.ylabel('impurity index')
plt.show()
```

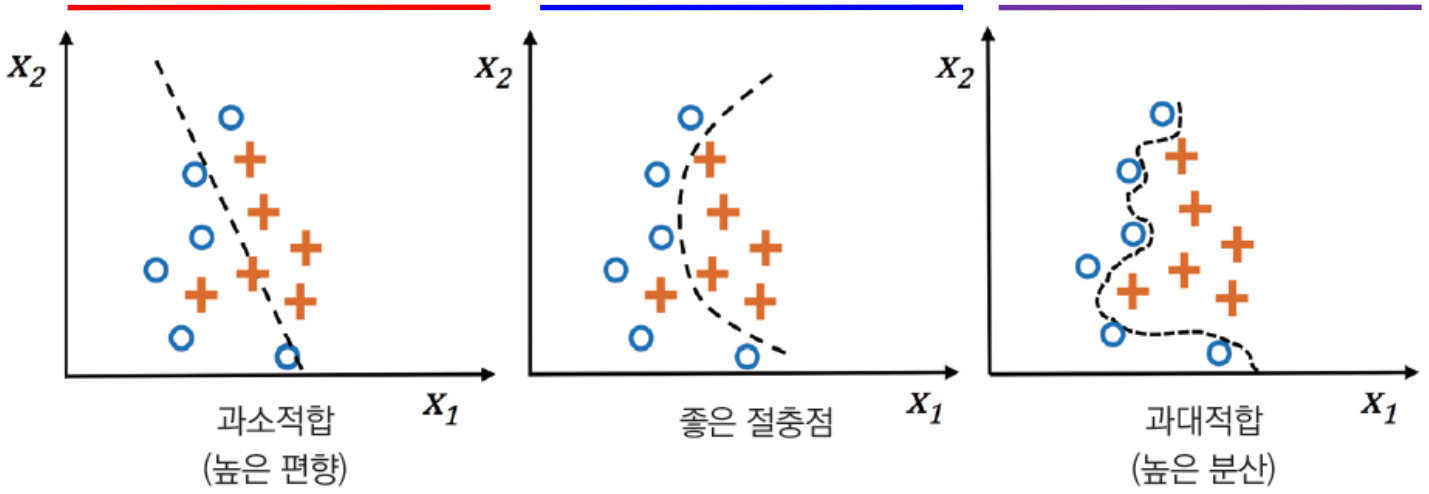


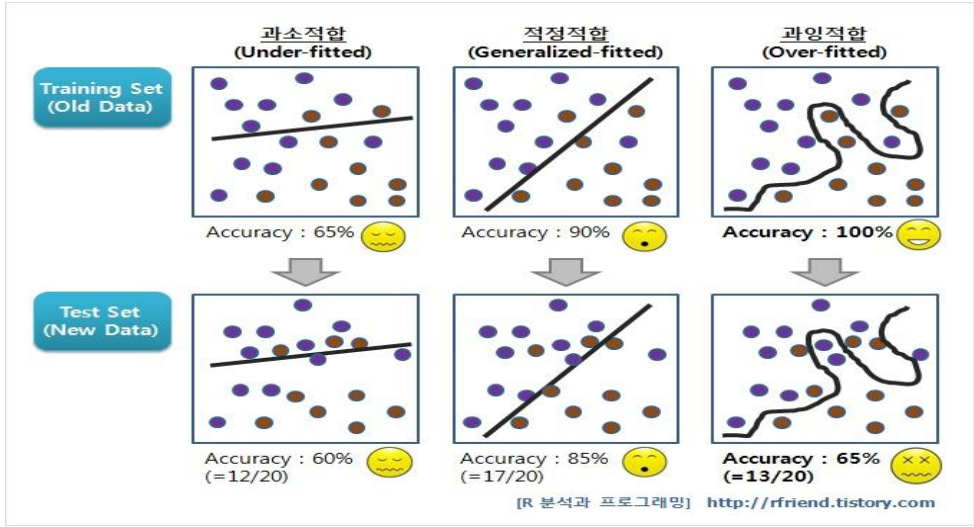
분류 모델의 수학적인 해석

- 분류 모델은 데이터를 구분하는 결정 경계(Decision boundary)선을 찾는 것으로 생각할 수 있다.



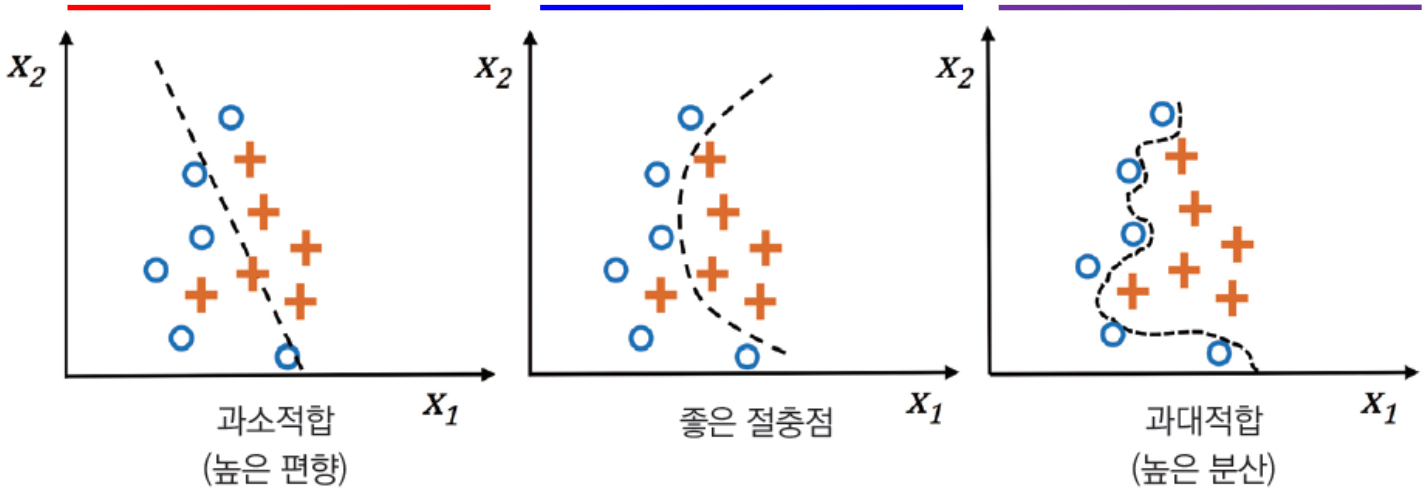
Overfitting and Underfitting

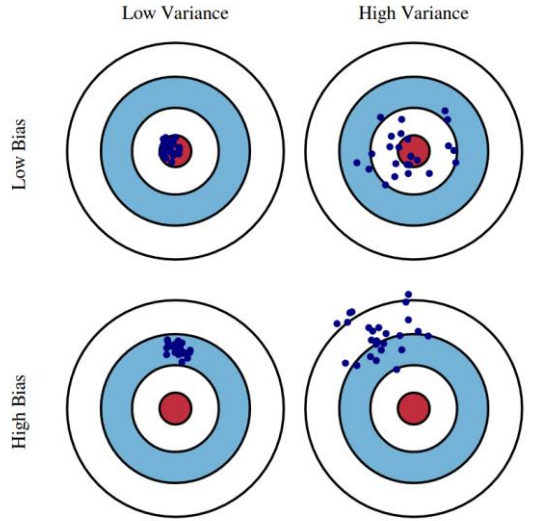




- 과소적합(underfitting)
  - 분류기가 학습 데이터를 잘 분류하지 못하는 상태
  - 학습이 덜 되었거나, (데이터 지니는 복잡성 보다) 단순한 결정 경계를 가지는 모델
- 과대적합(overfitting)
  - 분류기가 학습 데이터를 잘 분류. 하지만 테스트 데이터를 잘 분류하지 못함
  - (데이터 지니는 복잡성 보다) 복잡한 결정 경계를 가지는 모델: 불필요한 패턴까지 학습

Bias-Variance Tradeoff





- (데이터 지니는 복잡성 보다) 단순한 결정 경계를 가지는 (과소적합된) 모델은 높은 편향(bias)을 지녔다고 하며 낮은 분산(variance)을 가졌다고 말한다. 편향은 분류기가 주어진 데이터를 잘못 이해하고 있는 것으로 생각할 수 있다.
- (데이터 지니는 복잡성 보다) 복잡한 결정 경계를 가지는 (과대적합된) 모델은 높은 분산(variance), 낮은 편향(bias)을 지녔다고 말한다. 높은 분산은 학습 데이터에 따라 분류기가 보이는 예측 패턴이 많이 달라질 수 있음을 나타낸다.
- 편향과 분산은 일반적으로 하나가 증가하면 하나가 감소하는 경향이 있는데, 이 관계를 **편향-분산 트레이드오프**라 한다.
- 편향-분산 트레이드오프를 고려하여 학습 데이터에 있는 잡음은 잡지 않고, 유의미한 패턴만을 학습하는 적당히 복잡한 결정 경계를 갖는 모델을 선택해야 한다. 복잡도는 알고리즘 종류 또는 하이퍼파라미터로 조정 가능하다.

결정 트리 방법의 특징

- 결정 트리는 각 규칙 노드가 하나의 특징을 이용해 분기를 만든다: 예. $f > 0.2$
- 여러 특징을 동시에 사용하지 않기 때문에, 특징의 스케일을 조정할 필요가 없다.
- 하나의 특징에 해당하는 결정 경계는 직선 형태로 표현되며, 트리가 깊어질 수록 결정 경계가 복잡해진다 → 과대적합되기 쉽다.

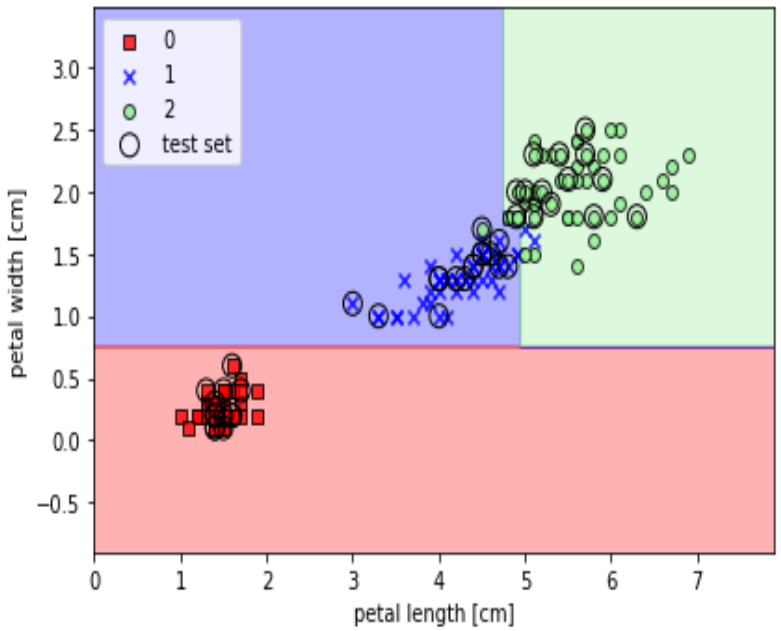
결정 트리 장점	결정 트리 단점
• 쉽다. 직관적이다 • 피처의 스케일링이나 정규화 등의 사전 가공 영향도가 크지 않음.	• 과적합으로 알고리즘 성능이 떨어진다. 이를 극복하기 위해 트리의 크기를 사전에 제한하는 튜닝 필요.

결정 트리 모델 학습 예제

- 붓꽃 데이터를 이용해 분류 모델을 구축한다.

```
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)

X_combined = np.vstack((X_train, X_test))
y_combined = np.hstack((y_train, y_test))
```

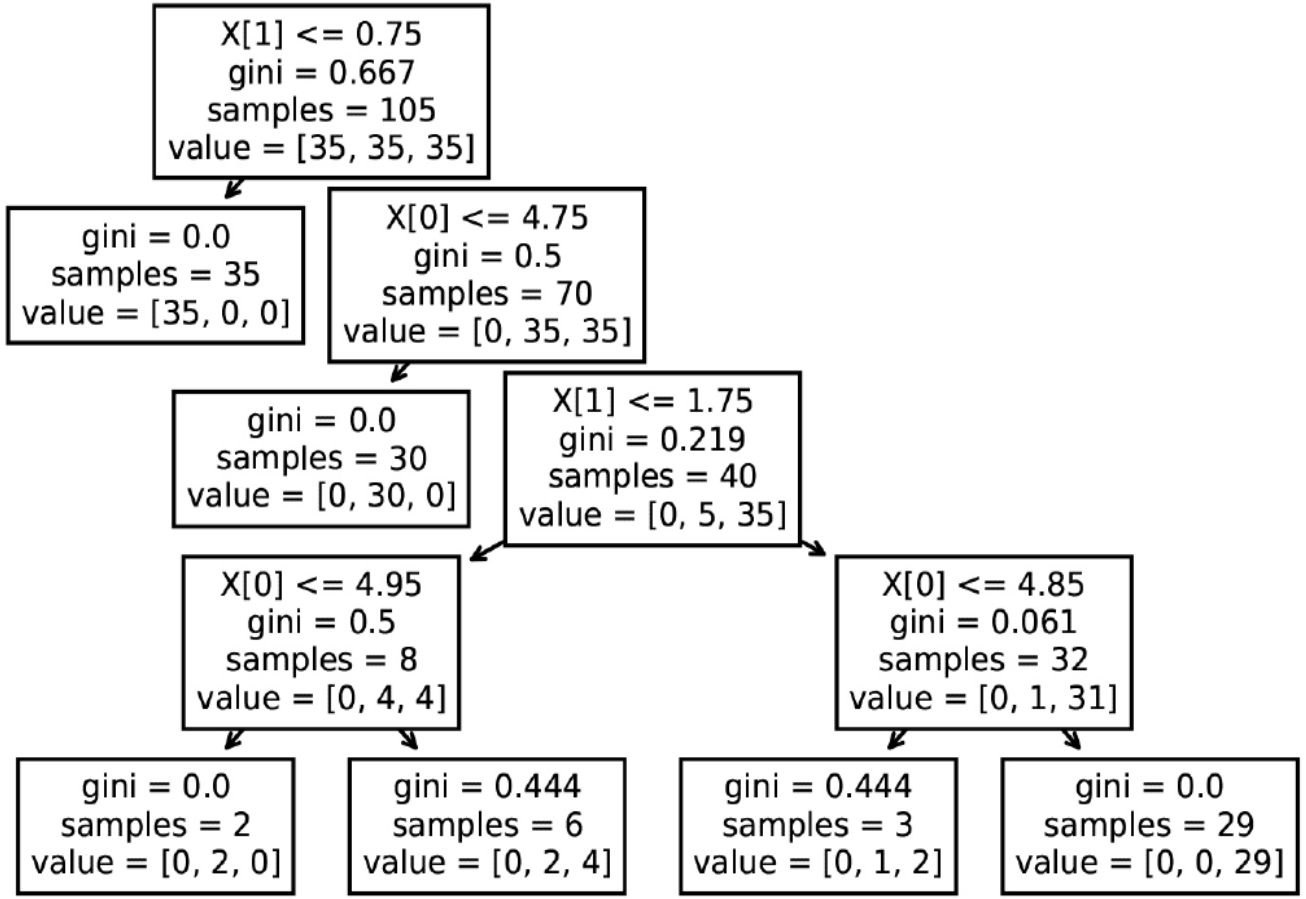
```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(criterion='gini',
                                    max_depth=4,
                                    random_state=1)

tree_model.fit(X_train, y_train)

plot_decision_regions(X_combined,
                      y_combined,
                      classifier=tree_model,
                      test_idx=range(105,150))

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



결정 트리 모델 시각화

- 사이킷런 내장 함수를 이용해 학습이 끝난 결정 트리 모델을 시각화할 수 있다.

```
from sklearn import tree
tree.plot_tree(tree_model)
plt.show()
```

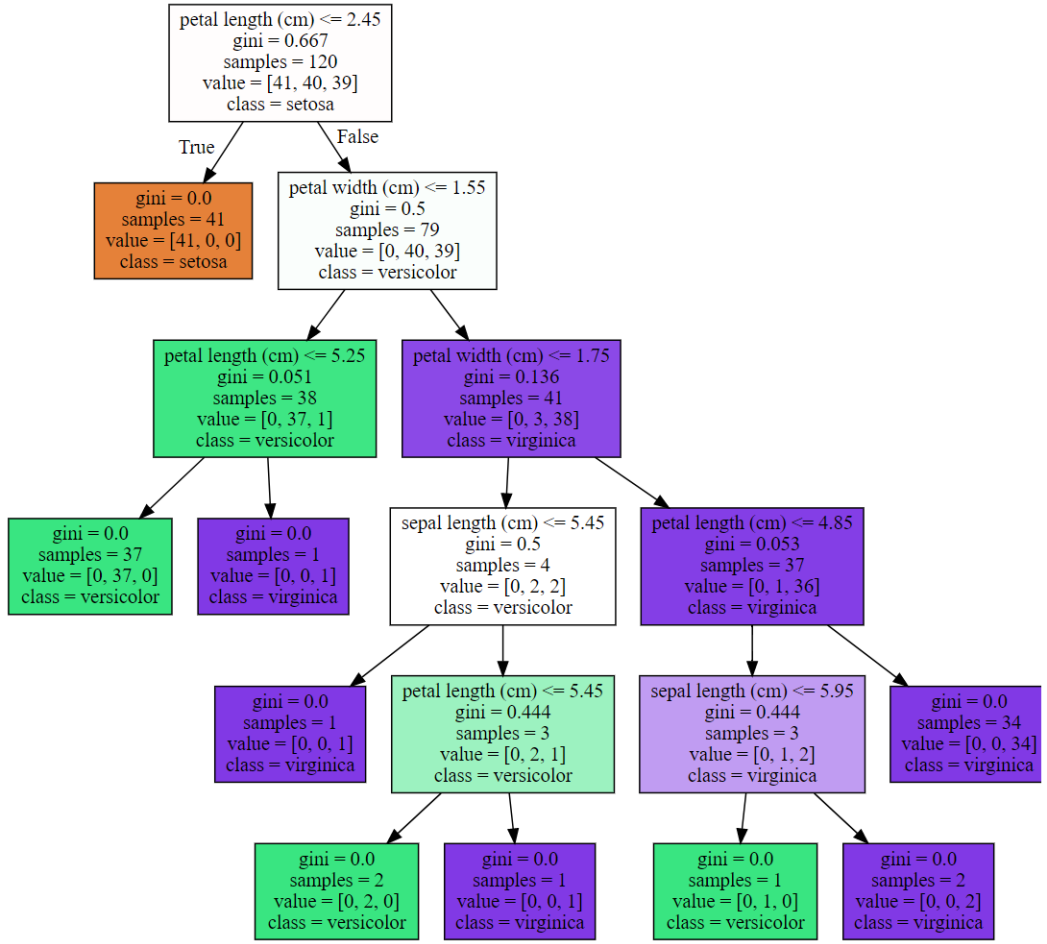


결정 트리 모델 시각화

- 외부 라이브러리 graphviz를 활용해 보다 멋진 시각화가 가능하다.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

# DecisionTree Classifier 생성
dt_clf = DecisionTreeClassifier(random_state=156)

# 붓꽃 데이터를 로딩하고, 학습과 테스트 데이터 셋으로 분리
iris_data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target,
                                                    test_size=0.2, random_state=11)

# DecisionTreeClassifier 학습.
dt_clf.fit(X_train, y_train)
```

```
from sklearn.tree import export_graphviz

# export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함.
export_graphviz(dt_clf, out_file='tree.dot', class_names=iris_data.target_names,
                feature_names = iris_data.feature_names, impurity=True, filled=True)
```



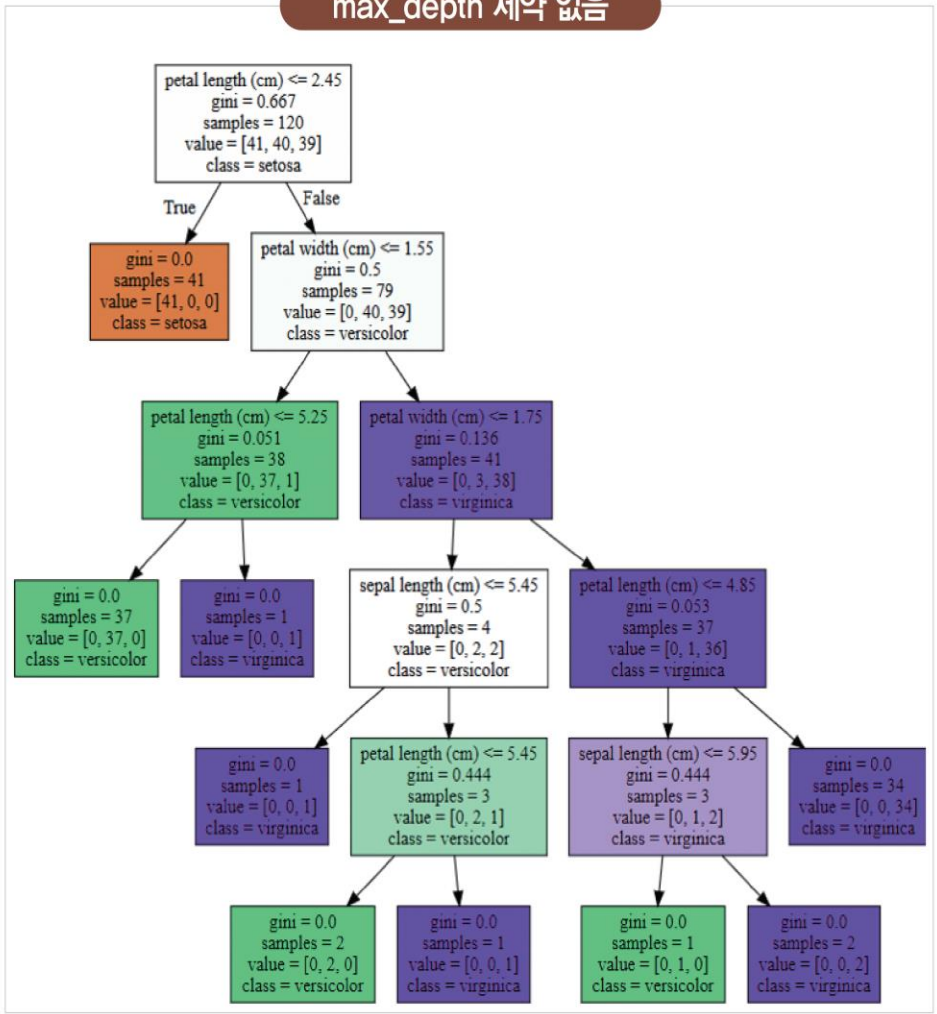
- 첫번째 줄: 결정 트리의 조건. 참/거짓 여부에 따라 분기 결정
- Samples: 학습 데이터 중 해당 노드를 거친 샘플 수

하이퍼파라미터와 트리 모델 가지치기

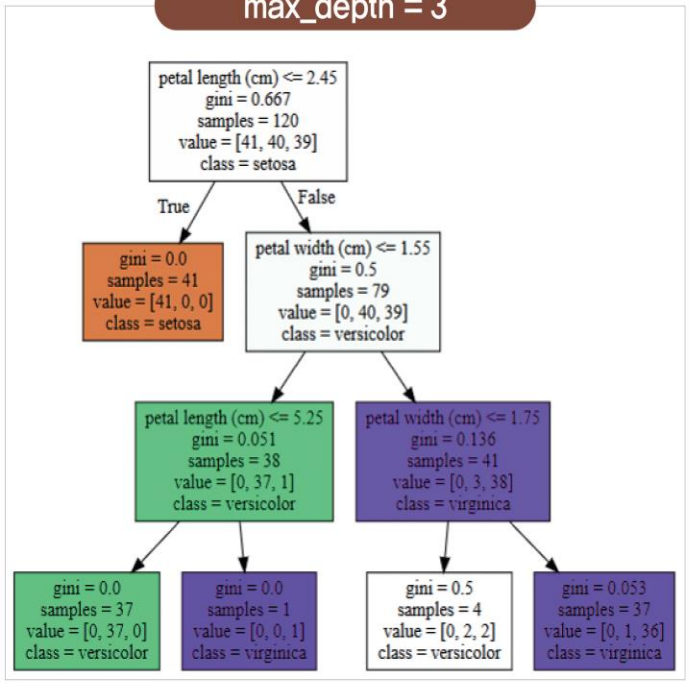
- 하이퍼파라미터: 학습 과정을 제어하는데 사용되는 파라미터
- 가지치기(pruning): 결정 트리의 하이퍼파라미터를 조정하여 학습이 완전히 이루어지지 않은 상태에서 (가지를 완전히 피지 않은 상태에서) 학습을 종료할 수 있다.
- 최대 깊이, 분할 시 최소로 필요한 샘플 수 하이퍼파라미터를 변경해 가지치기 할 수 있다: 제약 없는 트리에 비해 덜 복잡하므로 과대적합 위험이 감소한다 (분산 감소)

하이퍼파라미터와 트리 모델 가지치기

max_depth 제약 없음

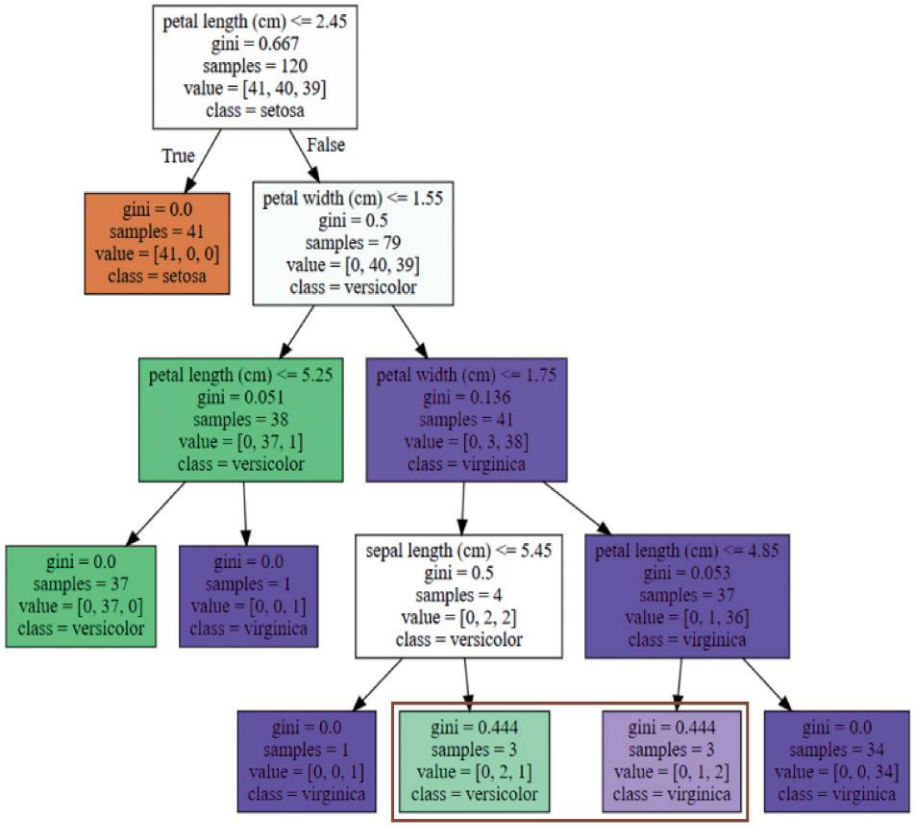


max_depth = 3



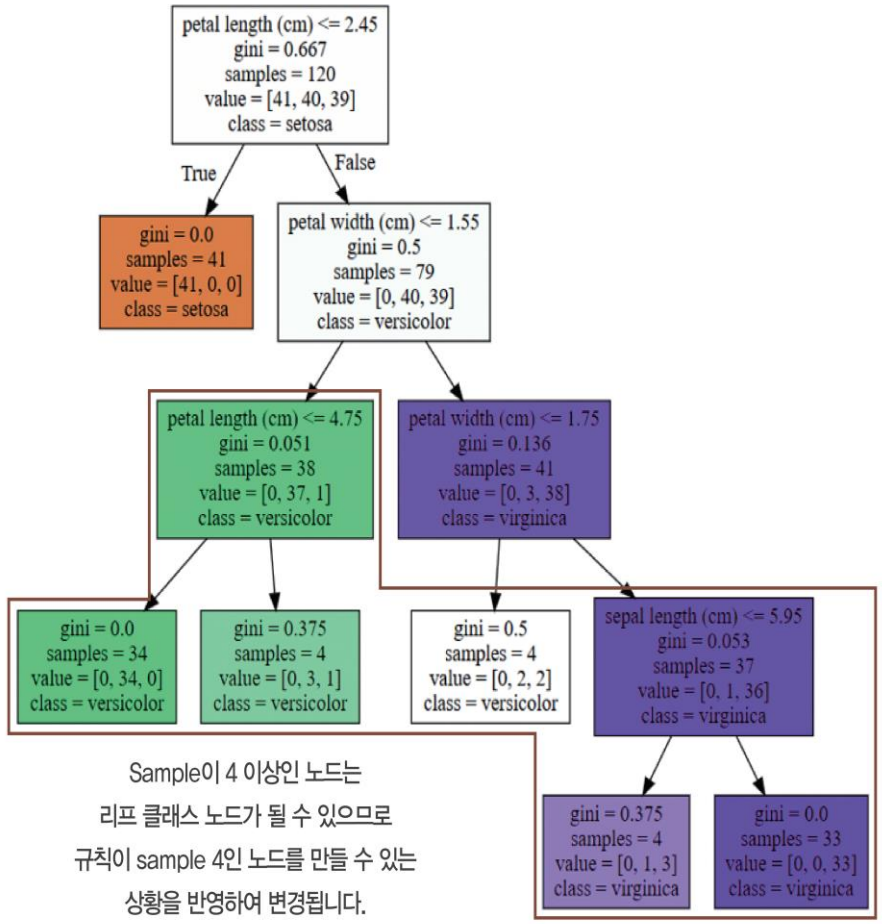
하이퍼파라미터와 트리 모델 가지치기

min_samples_split=4, 정확도 0.993



min_samples_split=4인데, Samples가 3개이므로
서로 Class 값이 있어도 Split하지 않습니다.

min_samples_leaf=4, 정확도 0.933

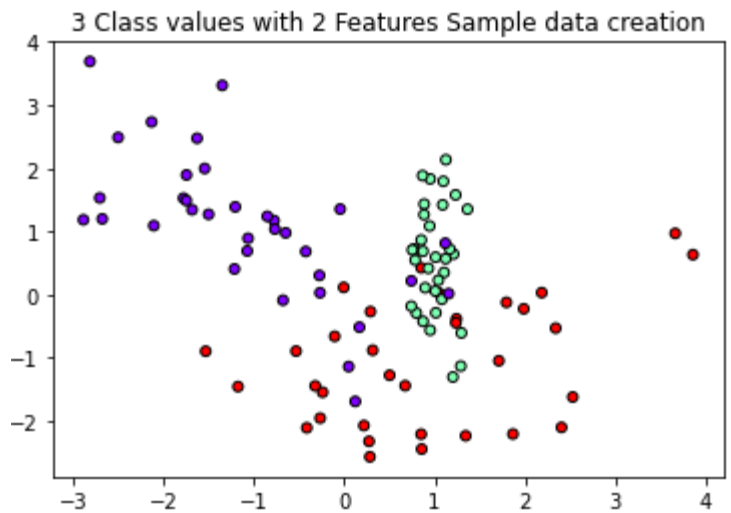


Sample이 4 이상인 노드는
리프 클래스 노드가 될 수 있으므로
규칙이 sample 4인 노드를 만들 수 있는
상황을 반영하여 변경됩니다.

결정 경계와 과대적합

- 결정 경계 시각화를 통해 하이퍼파라미터가 모델 학습에 미치는 영향을 살펴보자

```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
%matplotlib inline

plt.title('3 Class values with 2 Features Sample data creation')

# 2차원 시각화를 위해서 feature는 2개, 결정값 클래스는 3가지 유형의 classification 샘플 데이터 생성.
X_features, y_labels = make_classification(n_features=2, n_redundant=0, n_informative=2,
                                           n_classes=3, n_clusters_per_class=1, random_state=0)

# plot 형태로 2개의 feature로 2차원 좌표 시각화, 각 클래스값은 다른 색깔로 표시됨.
plt.scatter(X_features[:, 0], X_features[:, 1], marker='o', c=y_labels, s=25, cmap='rainbow', edgecolor='k')
```

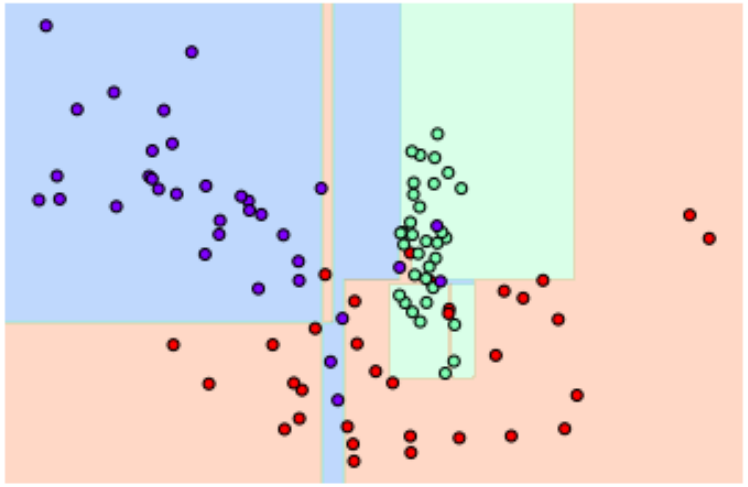



```
import numpy as np

# Classifier의 Decision Boundary를 시각화 하는 함수
def visualize_boundary(model, X, y):
  fig, ax = plt.subplots()

  # 학습 데이터 scatter plot으로 나타내기
  ax.scatter(X[:, 0], X[: 1], c=y, s=25, cmap='rainbow', edgecolor='k',
             clim=(y.min(), y.max()), zorder=3)
  ax.axis('tight')
  ax.axis('off')
  xlim_start, xlim_end = ax.get_xlim()
  ylim_start, ylim_end = ax.get_ylim()

  # 호출 파라미터로 들어온 training 데이터로 model 학습.
  model.fit(X, y)
  # meshgrid 형태인 모든 좌표값으로 예측 수행
  xx, yy = np.meshgrid(np.linspace(xlim_start, xlim_end, num=200),
                       np.linspace(ylim_start, ylim_end, num=200))
  Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

  # contourf()를 이용하여 class boundary를 visualization 수행.
  n_classes = len(np.unique(y))
  contours = ax.contourf(xx, yy, Z, alpha=0.3,
                         levels=np.arange(n_classes + 1) - 0.5,
                         cmap='rainbow', clim=(y.min(), y.max()),
                         zorder=1)
```

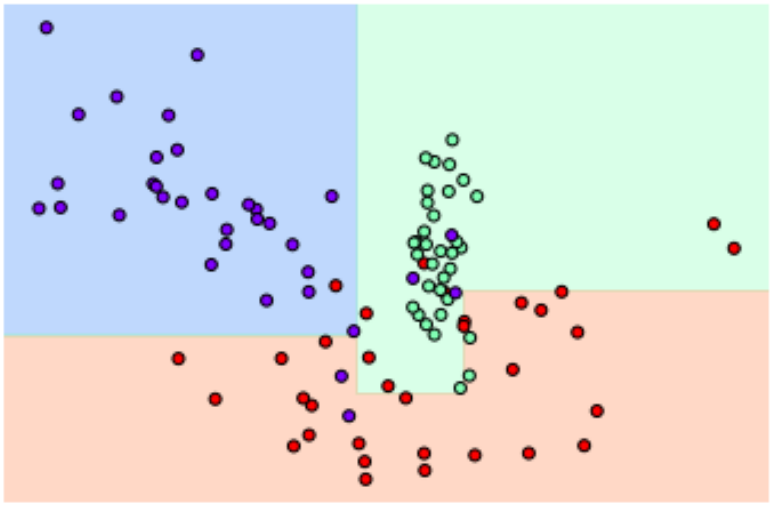
결정 경계와 과대적합

- 왼쪽 모델은 트레이닝 데이터에 너무 예민하게 학습되었다.
- 오른쪽 모델처럼 가지치기를 통해 적정한 수준의 편향을 가지도록 강제할 수 있으나, 최적 성능을 얻는 하이퍼파라미터를 찾기 위해 튜닝 과정이 필요하다. → 결정 트리 모델의 단점

```
from sklearn.tree import DecisionTreeClassifier

# 특정한 트리 생성 제약없는 결정 트리의 Decision Boundary 시각화.
dt_clf = DecisionTreeClassifier().fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```



```
# min_samples_leaf=6 으로 트리 생성 조건을 제약한 Decision Boundary 시각화
dt_clf = DecisionTreeClassifier(min_samples_leaf=6).fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```

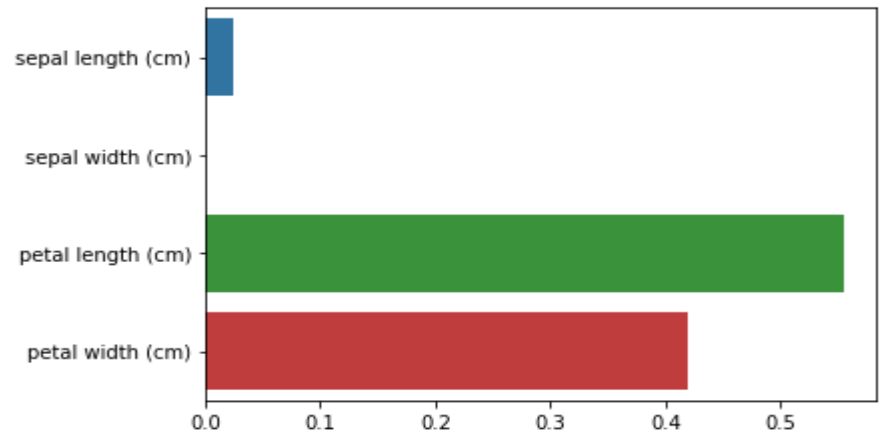


결정 트리 모델의 특징 중요도

- 트리의 각 특징을 이용해 분류를 수행했을 때 얻게 되는 정보 이득에 기반해 각 특징의 중요도를 측정할 수 있다: 한 트리에 한 특징이 여러 번 규칙 노드에 사용될 수 있기 때문에, 평균적인 정보 이득을 측정한다.
- 중요한 특징만 모델링에 사용하는 방법으로 응용도 가능하다 (특징 선택)

```
import seaborn as sns
import numpy as np
%matplotlib inline

# feature importance 추출
print('Feature importances:₩n{0}'.format(np.round(dt_clf.feature_importances_, 3)))

# feature importance 매핑
for name, value in zip(iris_data.feature_names, dt_clf.feature_importances_):
  print('{0} : {1:.3f}'.format(name, value))

# feature importance를 column 별로 시각화 하기
sns.barplot(x=dt_clf.feature_importances_, y=iris_data.feature_names)
```

```
Feature importances:
[0.025 0.    0.555 0.42 ]
sepal length (cm) : 0.025
sepal width (cm) : 0.000
petal length (cm) : 0.555
petal width (cm) : 0.420
<matplotlib.axes._subplots.AxesSubplot at 0x7f8940c1d350>
```



결정 트리 응용: 랜덤 포레스트(Random Forest)

- 랜덤 포레스트는 결정 트리의 앙상블(ensemble)로 생각할 수 있다:
  앙상블 방법이 무엇인지는 이후 강의에서 더 자세히 다룬다.
- 가지치기를 하지 않은 결정 트리 모델을 여러 개를 학습하여 합친다.
- 각 트리는 분산이 높은 문제가 있지만 여러 개의 트리를 합친 견고한 모델을 만들어 과대적합의 위험을 줄인다.

랜덤 포레스트 학습 알고리즘

1. n개의 랜덤한 부트스트랩(bootstrap) 샘플을 뽑는다.
(훈련 데이터셋에서 중복을 허용하면서 랜덤하게 n개의 샘플을 선택)

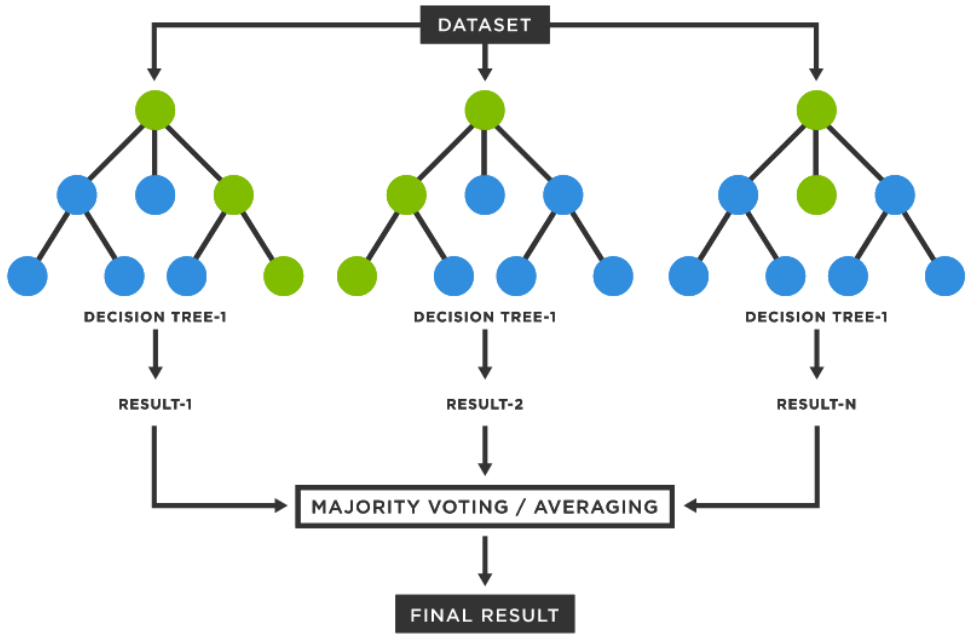
2. 각 부트스트랩 샘플에 대해 결정 트리를 학습한다.

   a. 중복을 허용하지 않고 랜덤하게 d 개의 특징을 선택

   b. 정보 이득 기준으로 최선의 분할을 만드는 특징으로 노드 분할

3. 단계 1~2를 k 번 반복

4. 각 트리의 예측을 모아 다수결 투표로 레이블 할당



* 랜덤 포레스트는 결정 트리와 비교해 하이퍼파라미터 튜닝에 큰 노력을 기울이지 않아도 되며, 트리 개수 k가 성능에 영향을 주는 중요하게 고려해야 할 하이퍼파라미터이다.