Machine Learning

Decision Tree,
Label Encoding, One-hot Encoding, Cross validation

학습목표

- **Decision Tree 알고리즘을 이해 할 수 있다.**
- **결측치가 있는지 확인할 수 있다.**
- **Label 인코딩과 One-hot 인코딩을 이해 할 수 있다.**
- **정답(레이블)과 연관 관계가 높은 특성을 선택할 수 있다**

  .

- **교차 검증 기법을 이해 할 수 있다.**

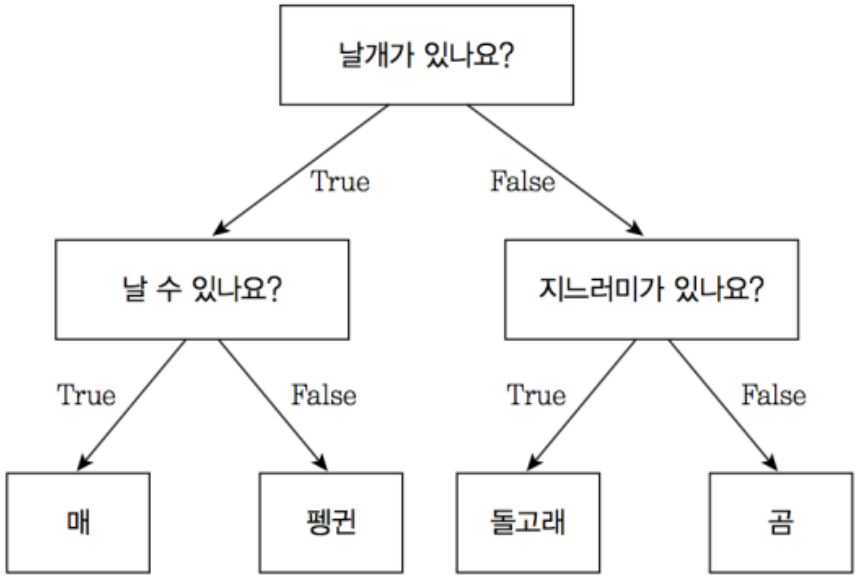
Decision Tree

Machine Learning

Decision Tree

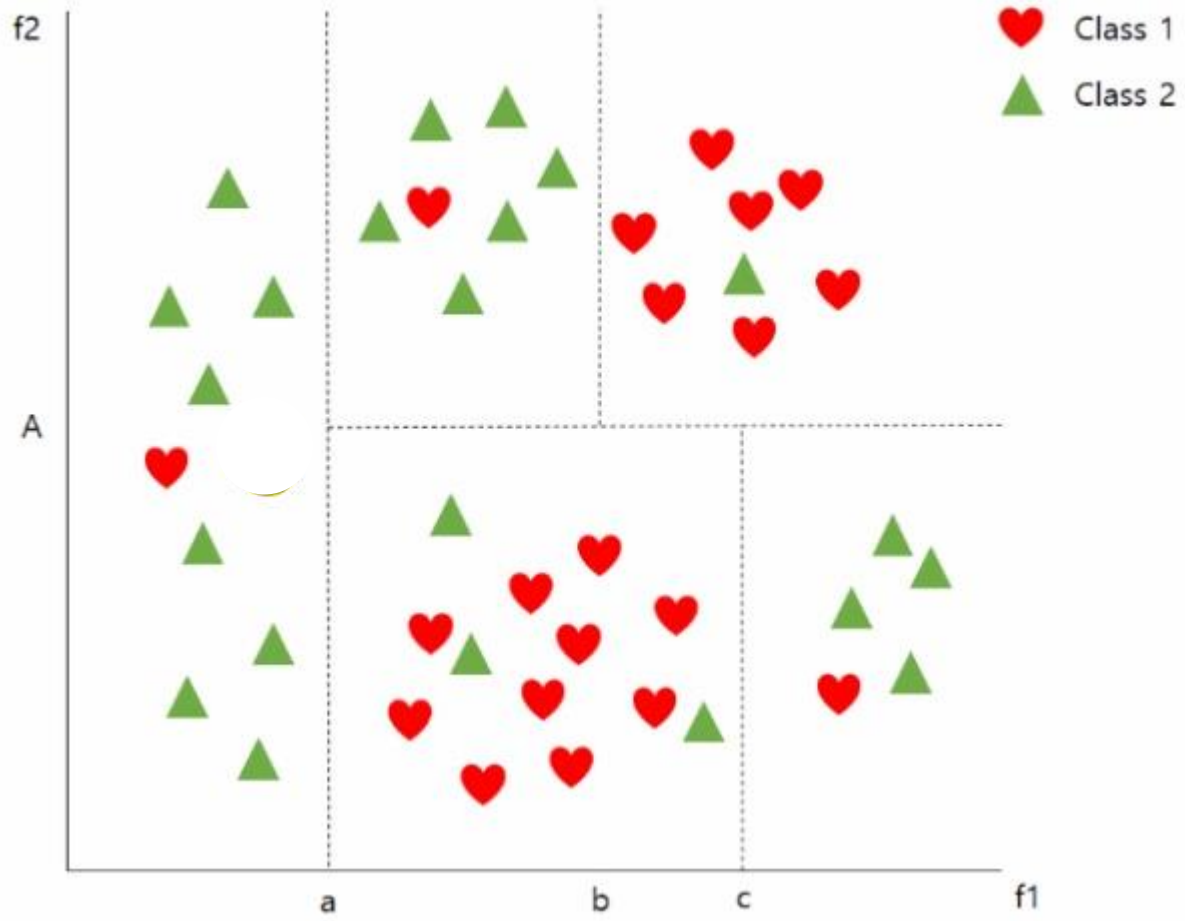
- 스무고개 하듯이 예/아니오 질문을 반복하며 학습
- 특정 기준(질문)에 따라 데이터를 구분하는 모델
- 분류와 회귀에 모두 사용 가능

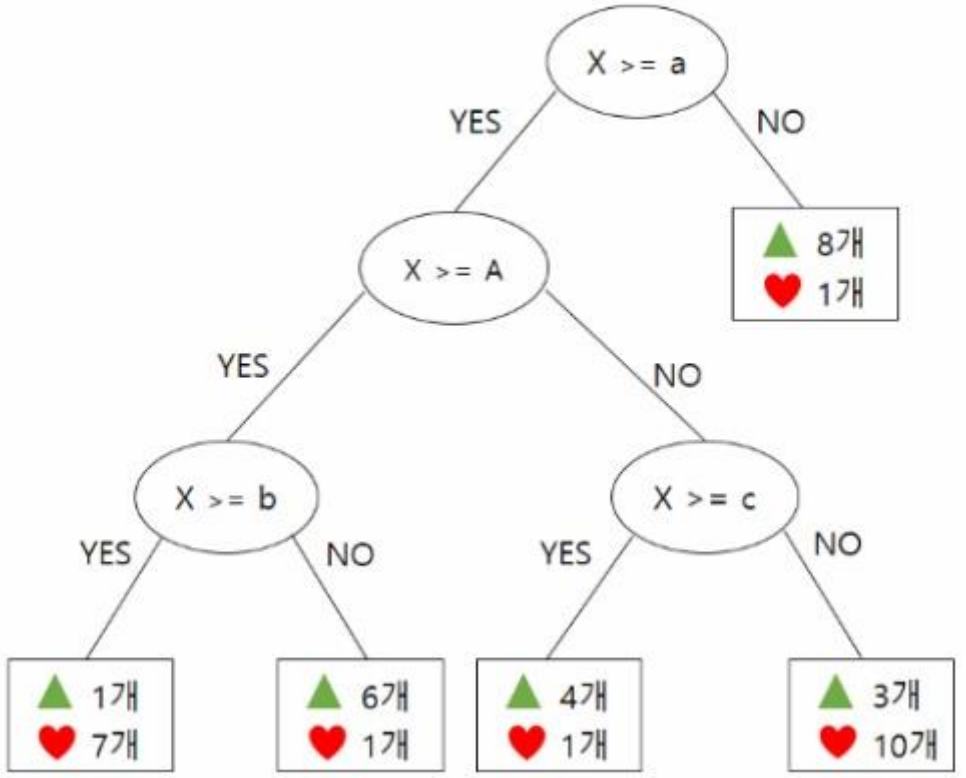
예) 매, 펭귄, 돌고래, 곰을 구분



Machine Learning

Decision Tree 의사 결정 방향 → 불순도가 낮아지는 방향

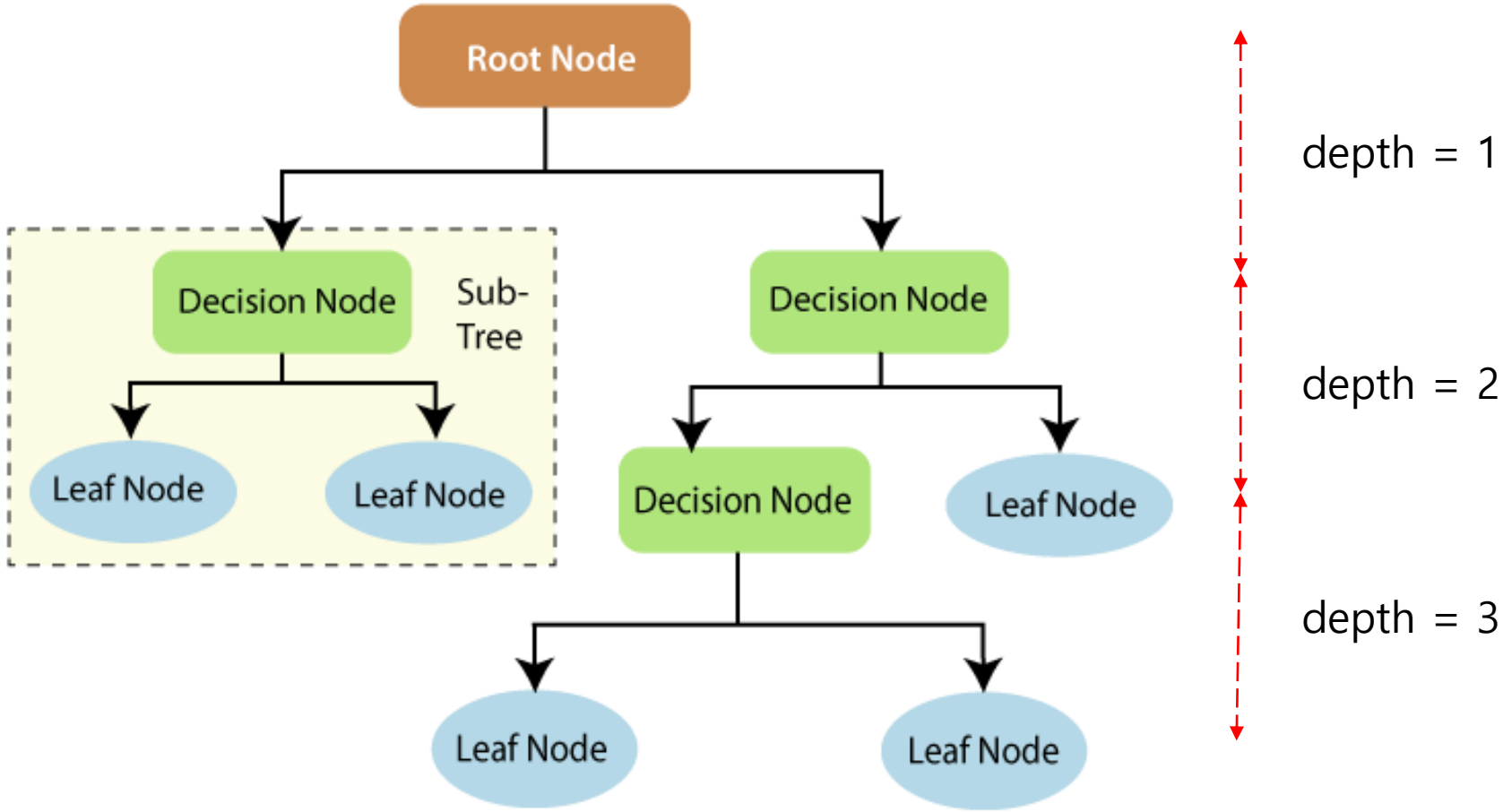




Machine Learning

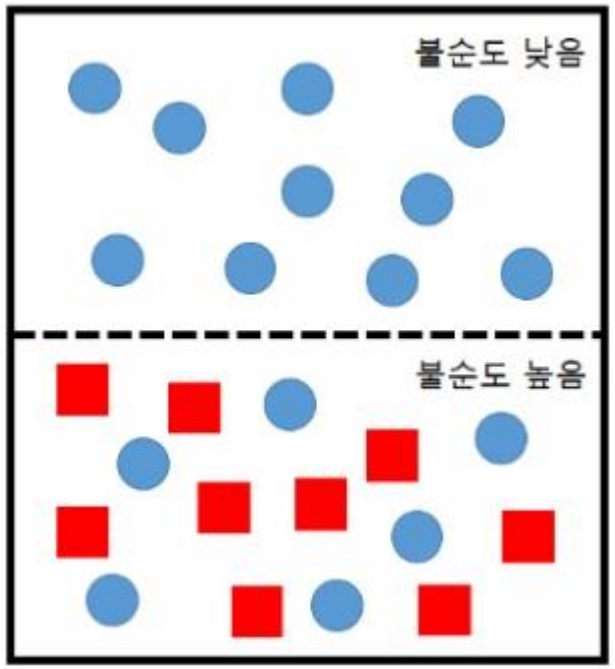
Decision Tree

Root Node, Decision Node, Leaf Node, depth, Sub Tree

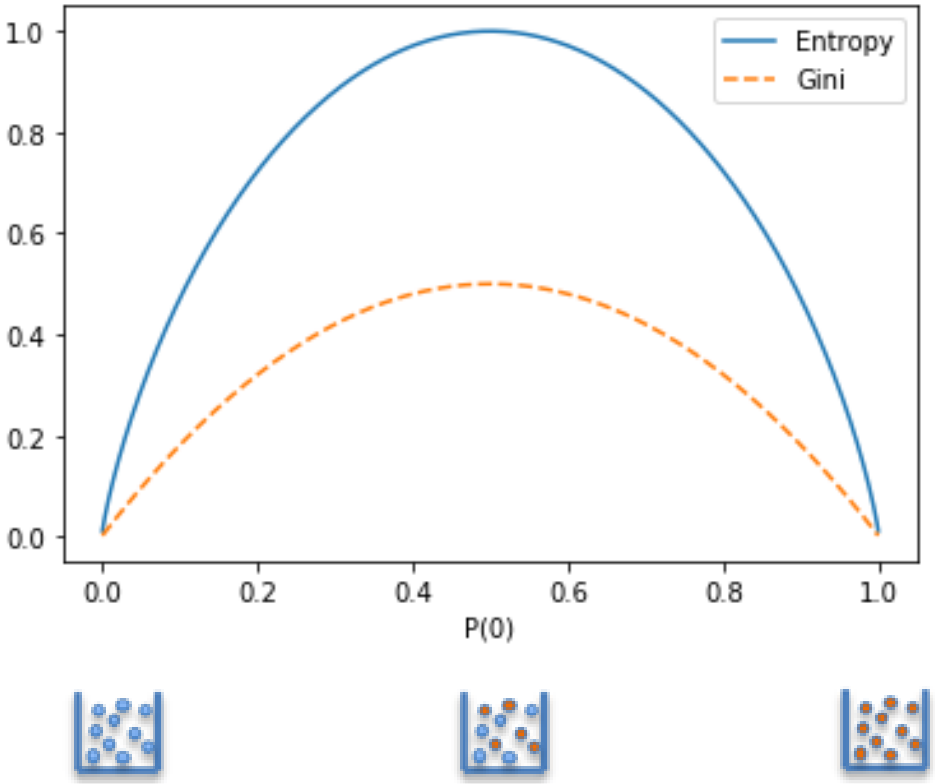


Machine Learning

Decision Tree 불순도 측정



Gini (지니 불순도) Entropy(엔트로피)



Machine Learning

Decision Tree 사용법

```
# 필요한 라이브러리 임폴트
from sklearn.tree import DecisionTreeClassifier


# 결정 트리 분류 모델 생성 함수 호출, 결정 트리 분류 모델 객체 생성
clf = DecisionTreeClassifier(하이퍼 파라미터, random_state)
```

Machine Learning

Decision Tree 사용법 : 주요 매개변수(hyperparameter)

DecisionTreeClassifier(criterion, max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes)

- **criterion : 불순도 측정 방법 (gini, entropy)**
- **max_depth : 트리의 최대 깊이**
- **min_samples_split : 노드를 분할하기 위한 최소 샘플 수**
- **min_samples_leaf : 리프 노드가 가져야할 최소 샘플 수**
- **max_leaf_nodes : 리프 노드의 최대 개수**

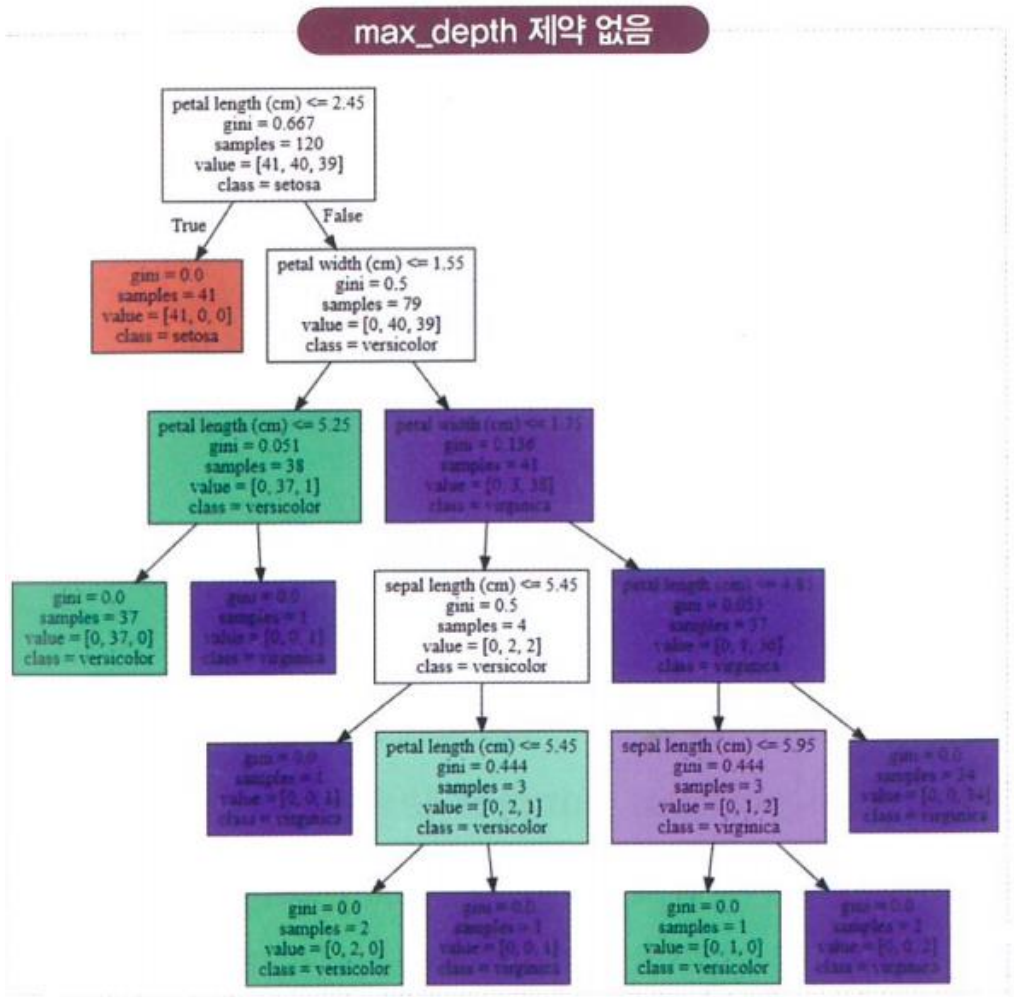
Machine Learning

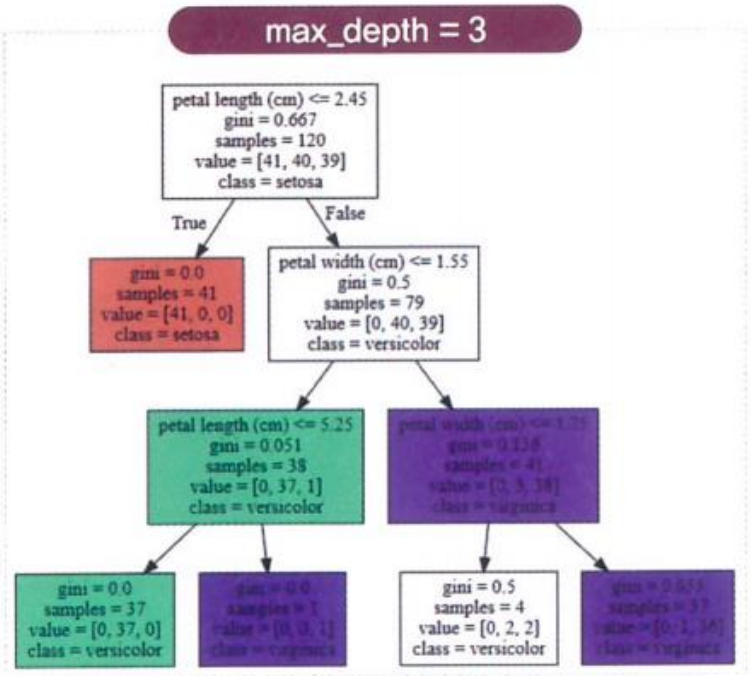
Decision Tree 과대적합 제어

- **max_depth : 트리의 최대 깊이, 값이 클수록 모델의 복잡도가 올라간다.**
- **max_leaf_nodes : 리프 노드의 최대 개수, 크게 설정될수록 분할되는 노드가 많아져서 과대적합 가능성 증가**
- **min_samples_split : 노드를 분할하기 위한 최소 샘플 수, 작게 설정될수록 분할되는 노드가 많아져서 과대적합 가능성 증가**
- **min_samples_leaf : 리프 노드가 가져야할 최소 샘플 수, 작게 설정될수록 분할되는 노드가 많아져서 과대적합 가능성 증가**

Machine Learning

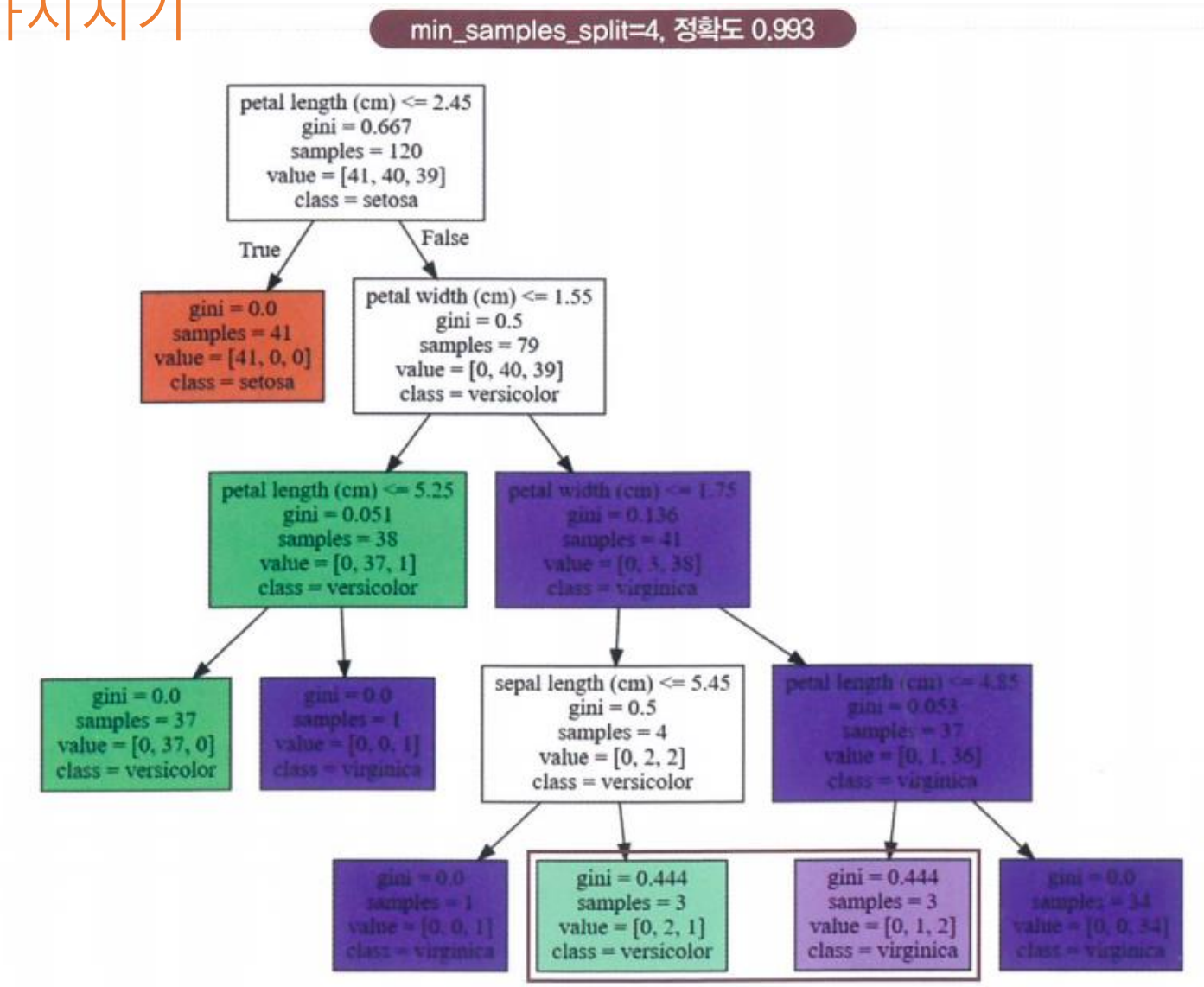
Decision Tree 사전 가지치기





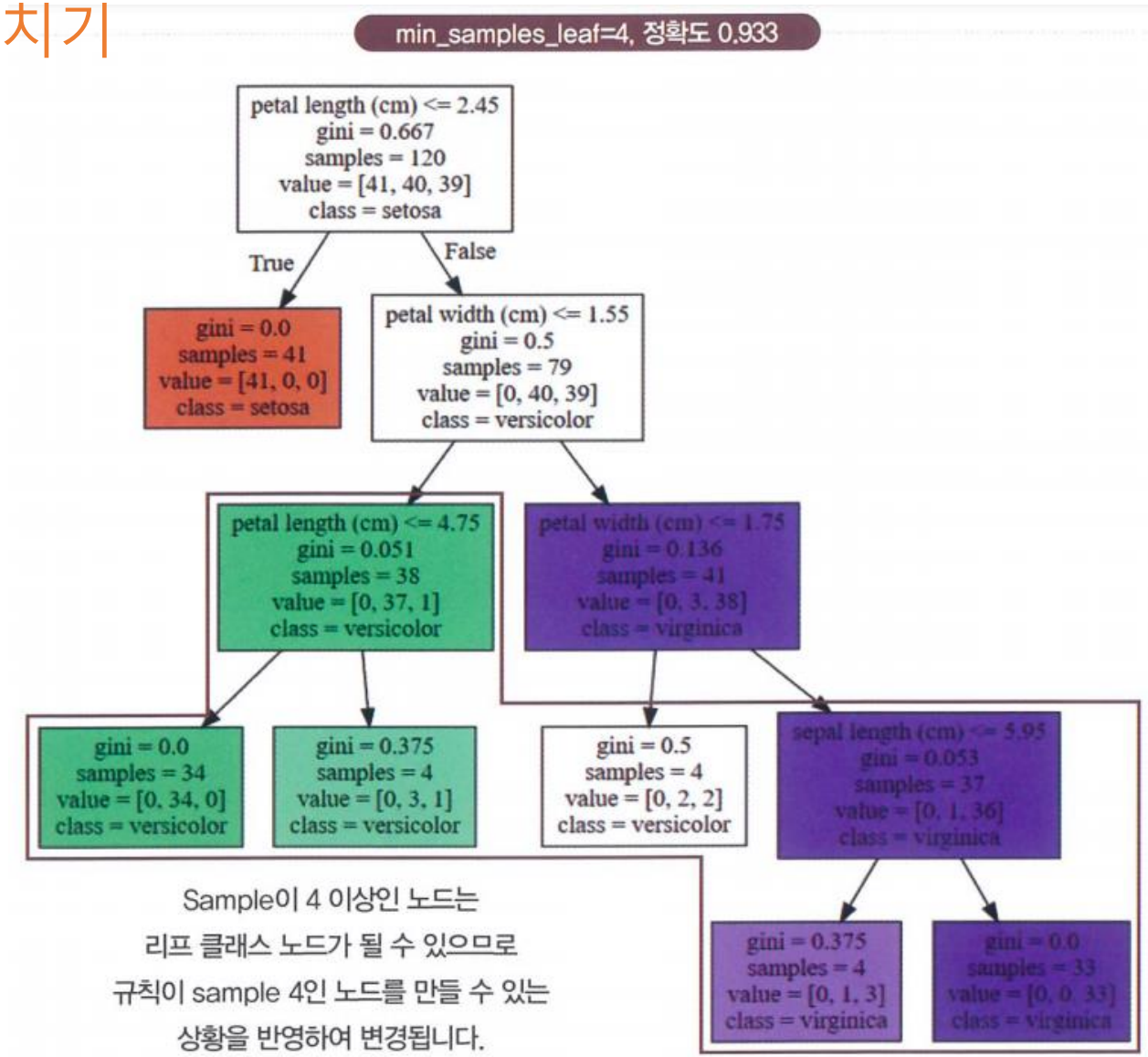
Machine Learning

Decision Tree 사전 가지치기



Machine Learning

Decision Tree 사전 가지치기



Machine Learning

Decision Tree 장단점

장점	단점
• 쉽다 • 직관적이다	• 과대적합이 발생하기 쉬움 • 이를 극복하기 위해서 트리의 키기를 사전에 제한하는 튜닝이 필요 (사전 가지치기)

Machine Learning

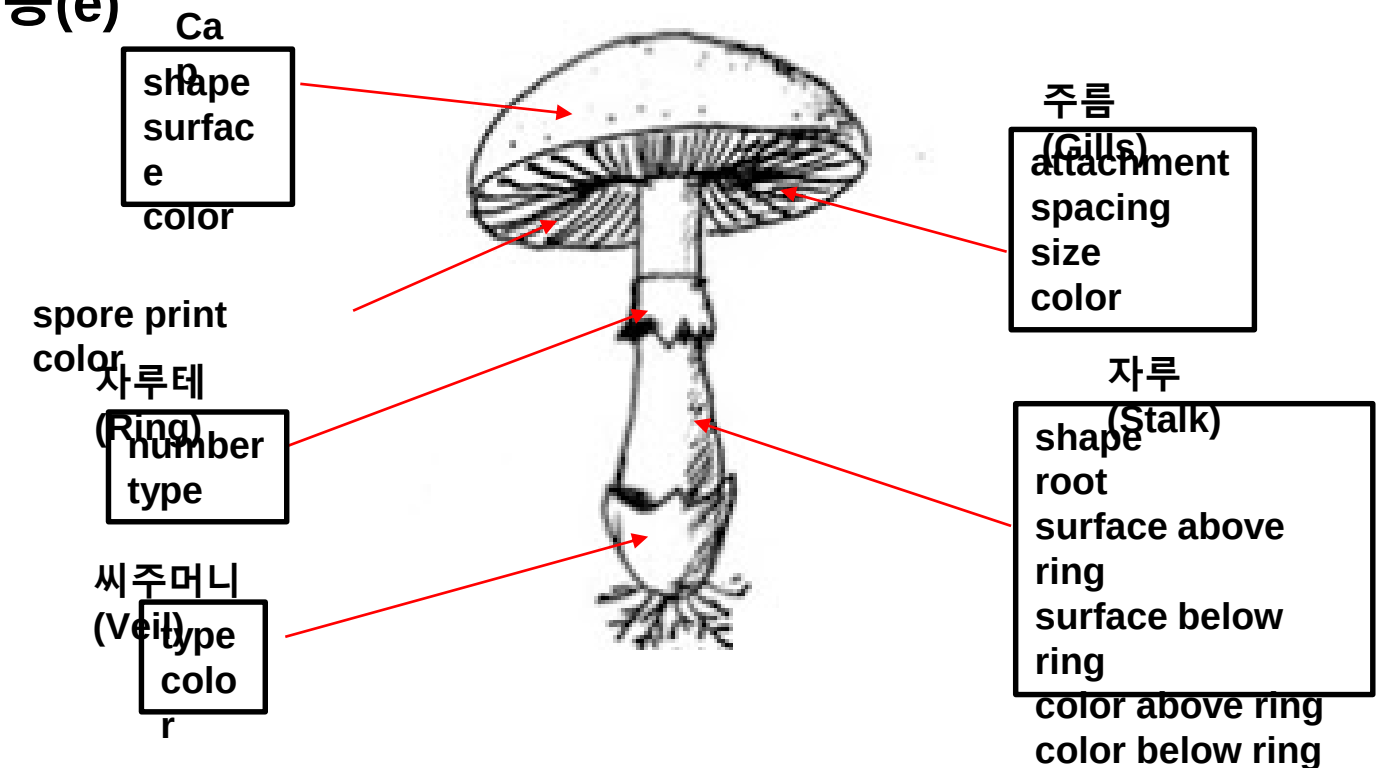
Decision Tree 실습

Mushroom 데이터 활용
Decision Tree 분류 실습

Machine Learning

Mushroom 데이터 셋

- 8124개의 버섯 종류 데이터
- 22개의 특징 (18개의 버섯 특징, 4개의 다른 특징 (Habitat (서식지), Population(분포 형태), Bruises(타박상), Odor(냄새)))
- 라벨 : 독성 (p), 식용(e)



함수2. value_counts() 함수 – 데이터의 종류와 개수 확인

X['cap-shape'].value_counts()

```
x    3656
f    3152
k     828
b     452
s      32
c       4
Name: cap-shape, dtype: int64
```

함수3. unique() 함수 – 데이터의 종류 확인

```
X['cap-shape'].unique()
```

```
['x' 'b' 's' 'f' 'k' 'c']
```

Machine Learning

Mushroom 데이터셋 : 범주형(이산형) 데이터, 인코딩 필요

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w

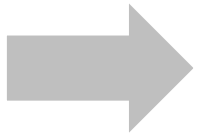
Label 인코딩 or One-hot 인코딩 방식을 이용해 수치화

Machine Learning

데이터 전처리 Label Encoding : 레이블을 숫자로 mapping

X1["capshape"] = X1["capshape"].map({"x":0, "f":1, "k":2, "b":3, "s":4, "c":5})

cap-shape
x
b
x
k
f
s
b
s
c



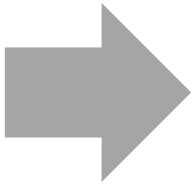
cap-shape
0
3
0
2
1
4
3
4
5

Machine Learning

데이터 전처리 One-hot Encoding : 분류하고자 하는 범주(종류) 만큼의 자릿수를 만들고 단 한 개의 1과 나머지 0으로 채워서 숫자화 하는 방식

X_one_hot = pd.get_dummies(X2)

cap-shape
x
b
x
k
f
s
b
s
c



cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x
0	0	0	0	0	1
1	0	0	0	0	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	0	0
0	0	0	0	1	0
1	0	0	0	0	0
0	0	0	0	1	0
0	1	0	0	0	0

Machine Learning

결정 트리 분류 모델 생성

```
# trian_test_split() 함수를 이용, 데이터를 학습용과 테스트용으로 분할


# 결정트리 분류 모델 생성 함수 호출, 하이퍼 파라미터는 전부 기본값 설정,
분류 모델 객체 생성
tree_model = DecisionTreeClassifier()


# 분류 모델 학습 진행
tree_model.fit(X_train, y_train)
```

Machine Learning

결정 트리 분류 모델 생성

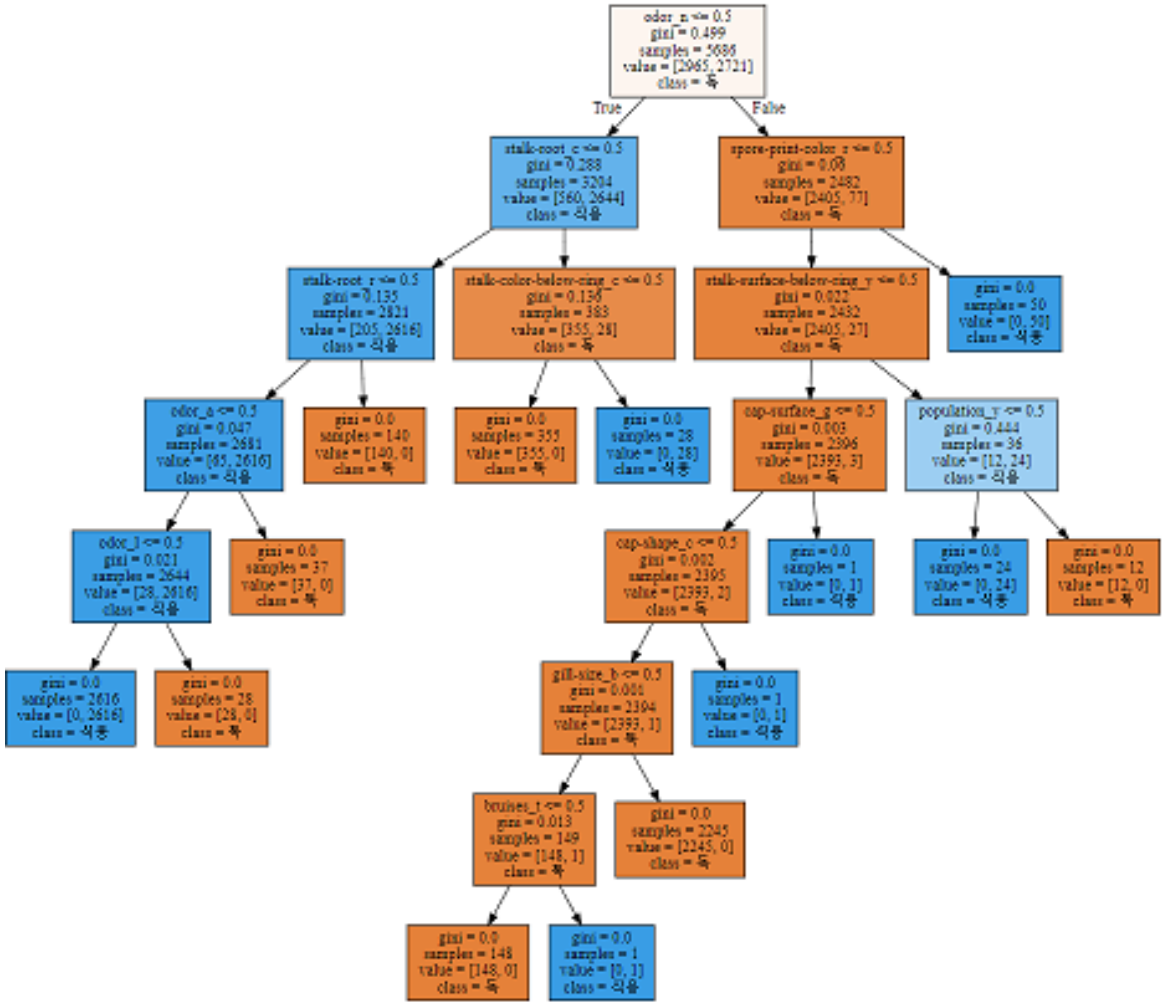
```
# 테스트 데이터를 이용해서 예측
y_pred = tree_model.predict(X_test)


# 테스트 데이터에 대한 분류 모델의 성능(평균 정확도) 확인
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

Machine Learning

사전 가지치기(pre-pruning)를 이용한 모델의 성능 향상

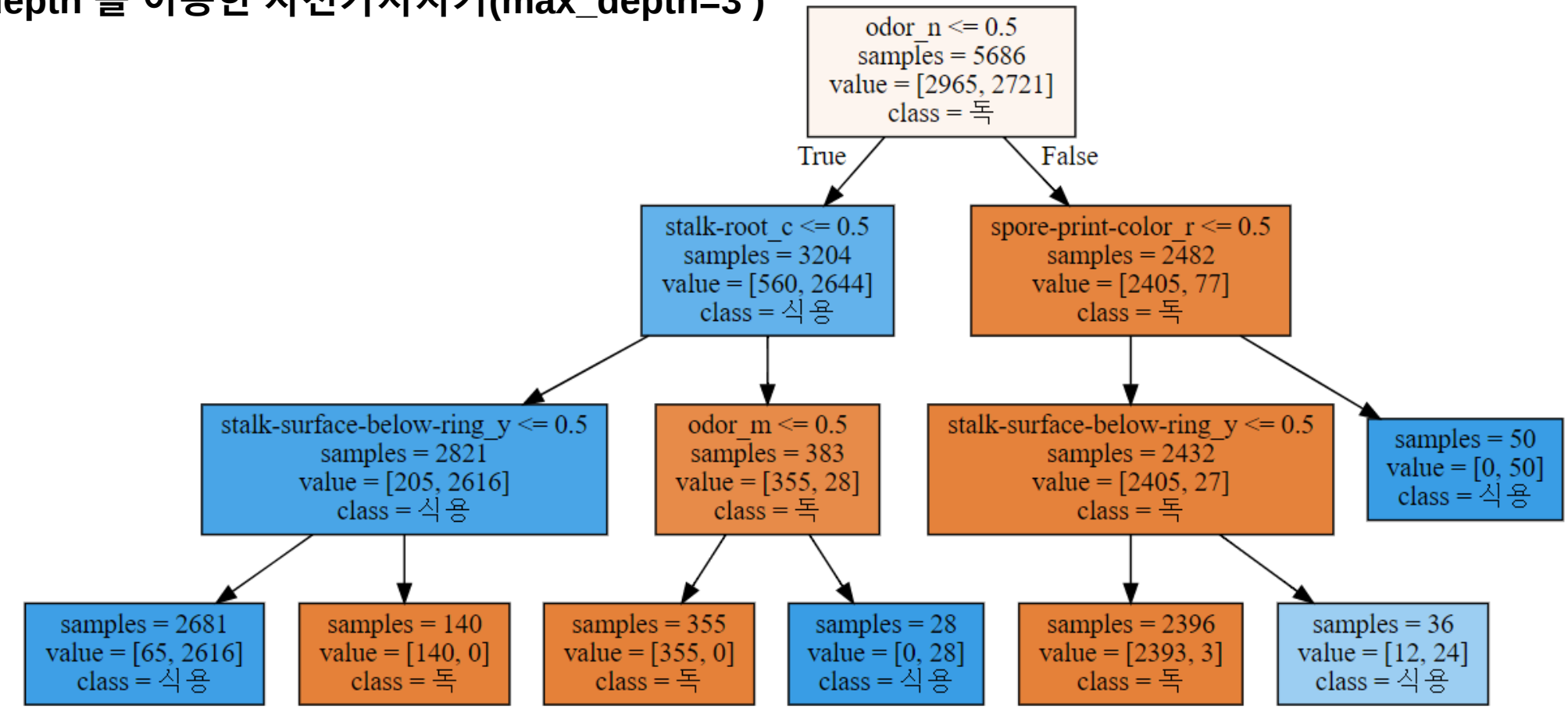
- 사전 가지치기를 하지 않은 기본 모델 (tree_model)



Machine Learning

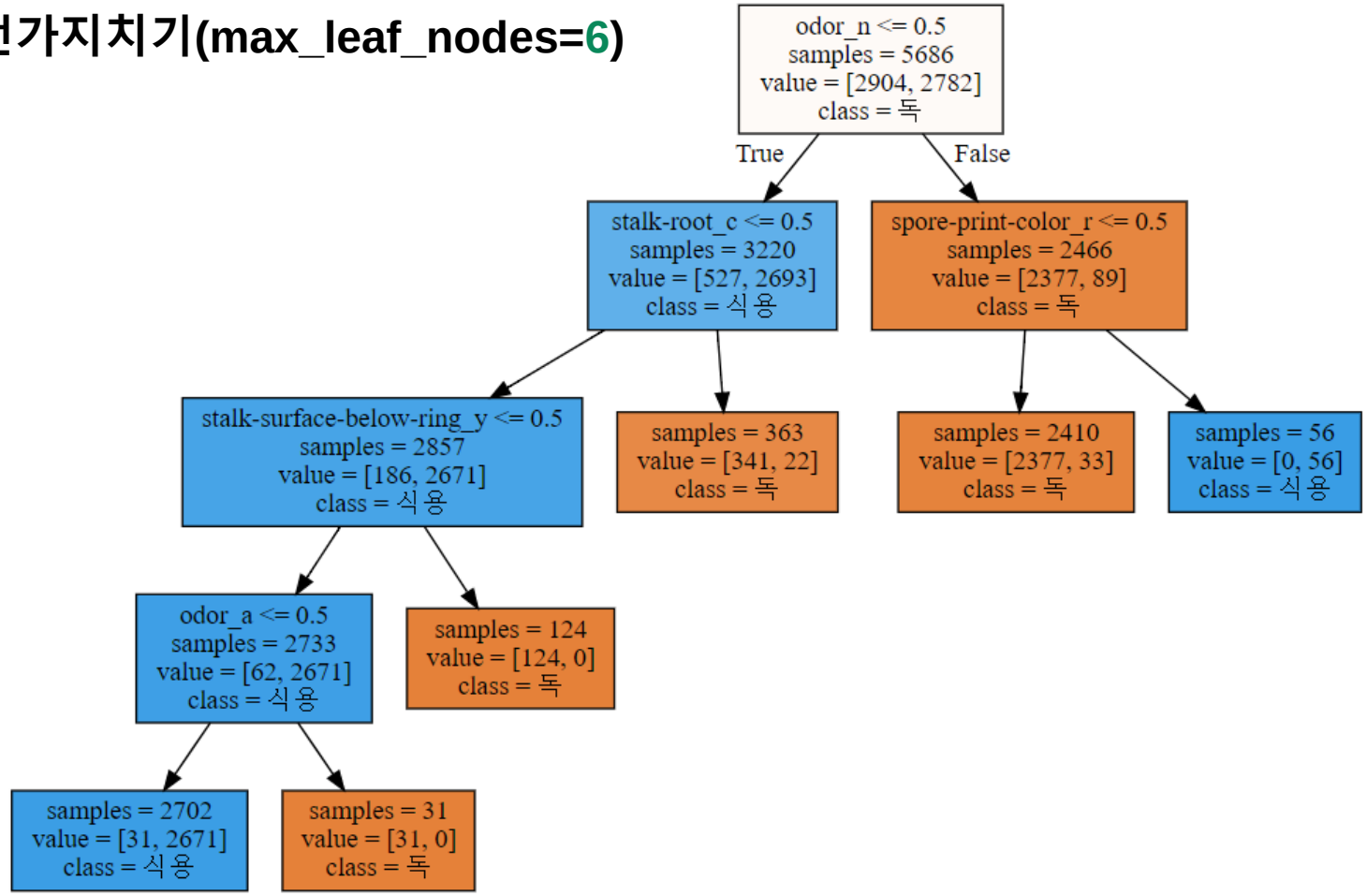
사전 가지치기(pre-pruning)를 이용한 모델의 성능 향상

- max_depth 를 이용한 사전가지치기(max_depth=3)



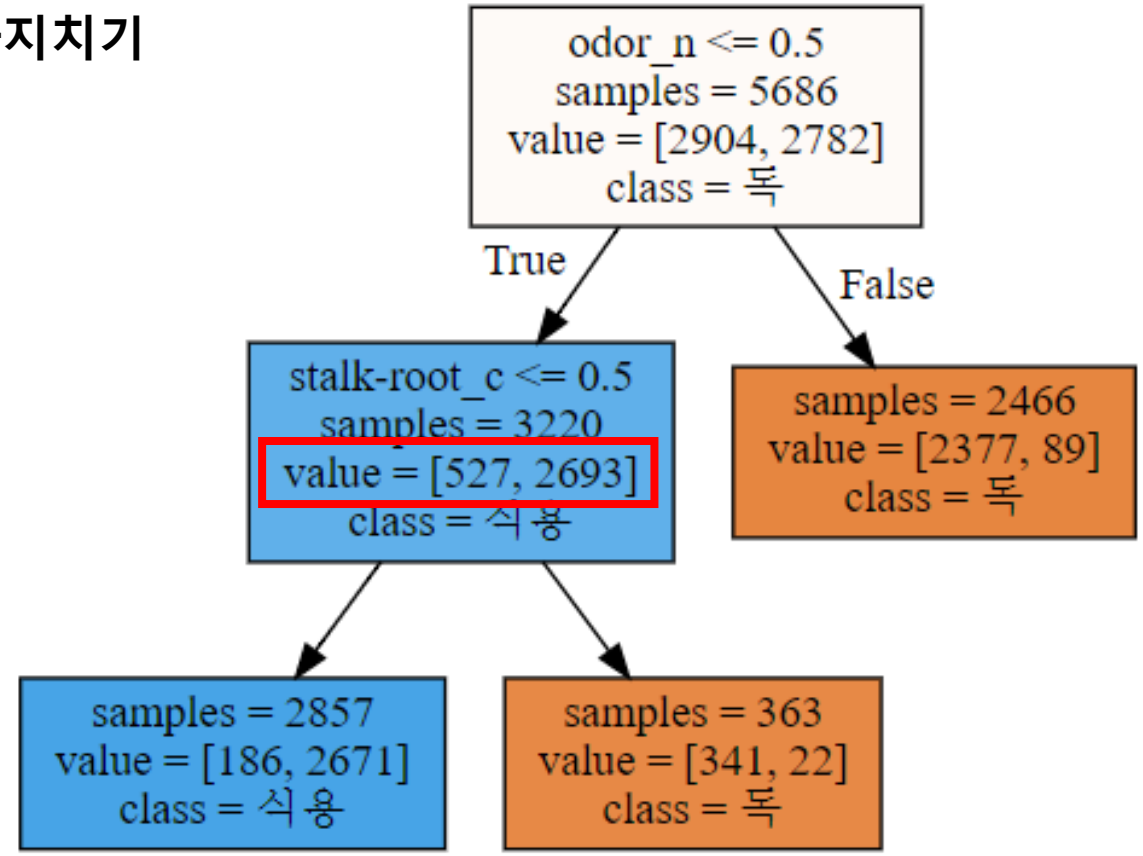
사전 가지치기(pre-pruning)를 이용한 모델의 성능 향상

- max_leaf_nodes를 이용한 사전가지치기(max_leaf_nodes=6)



사전 가지치기(pre-pruning)를 이용한 모델의 성능 향상

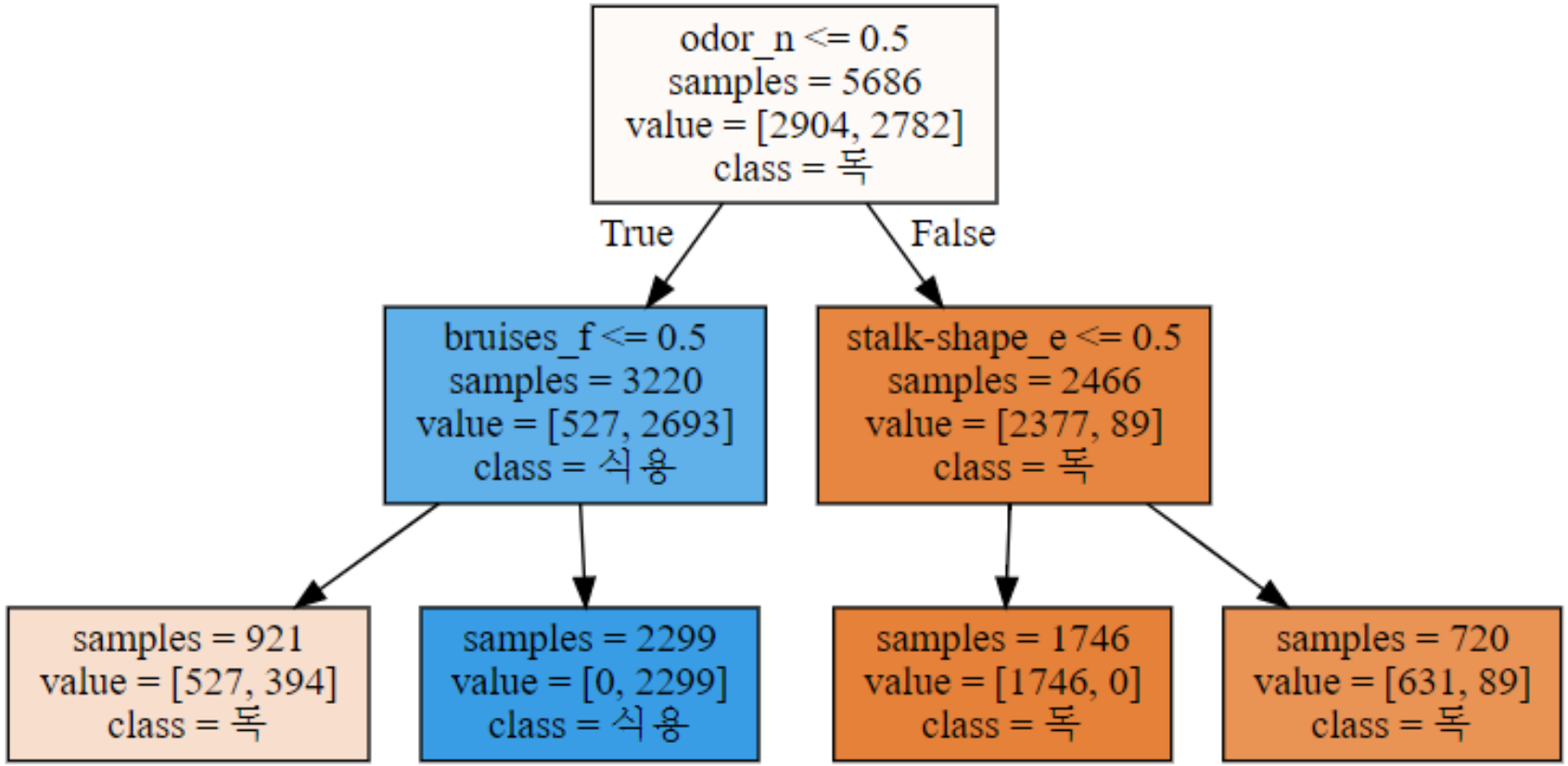
- **min_samples_split를 이용한 사전가지치기 (min_samples_split=3000)**



Machine Learning

사전 가지치기(pre-pruning)를 이용한 모델의 성능 향상

- min_samples_leaf를 이용한 사전가지치기(min_samples_leaf=700)



Machine Learning

특성 선택 : 정답(레이블)과 연관 관계가 높은 특성을 선택하는 작업

```
importance = tree_model.feature_importances_
df = pd.DataFrame(importance, index=X_one_hot.columns)
df.sort_values(by='name', ascending=False)
```

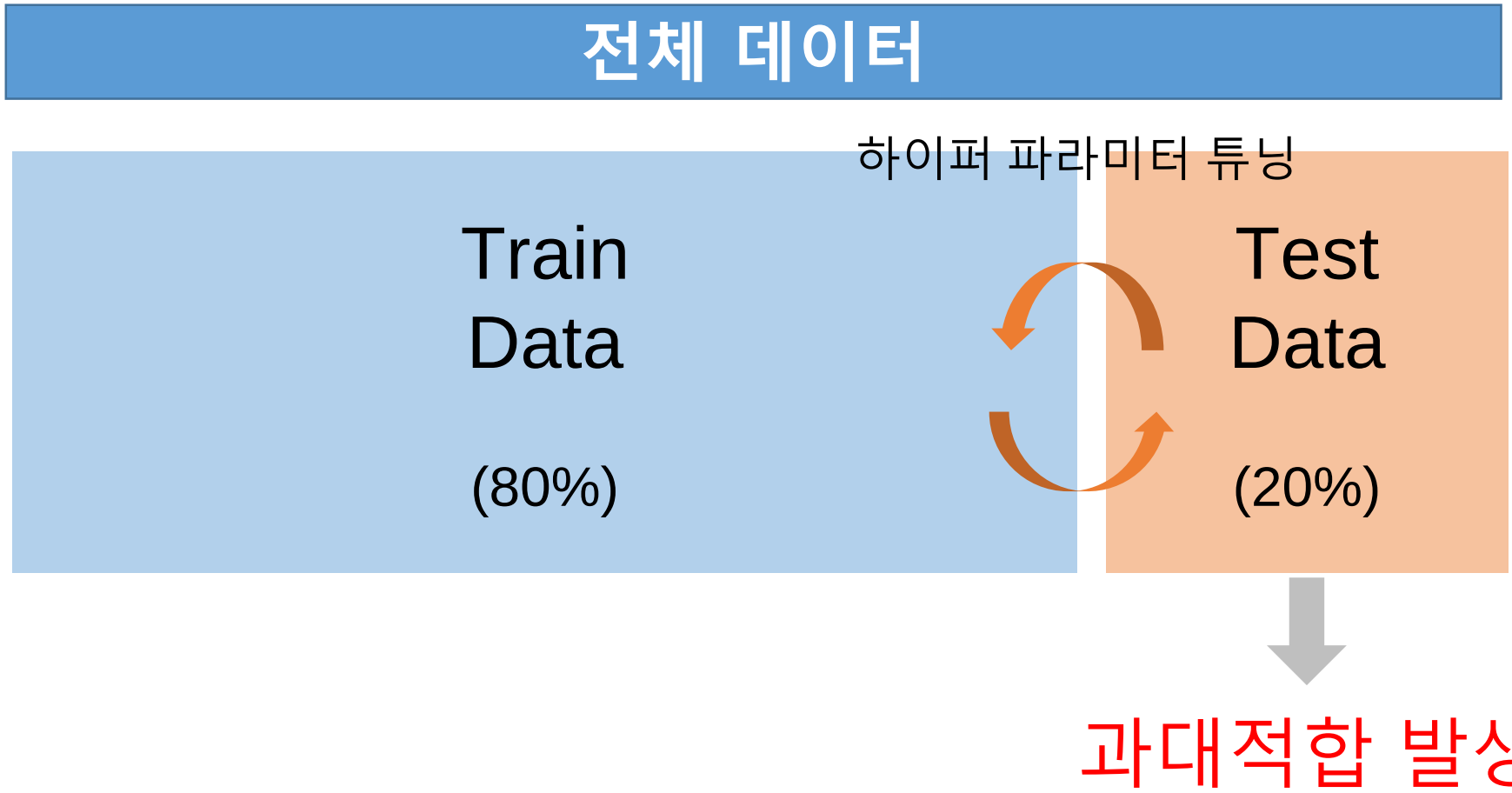
odor_n	0.611853
stalk-root_c	0.177321
stalk-root_r	0.089846
spore-print-color_r	0.033995
odor_a	0.023077
odor_l	0.022260
stalk-surface-below-ring_y	0.017375
stalk-surface-above-ring_k	0.013956
ring-number_o	0.005403
cap-surface_g	0.002102

모델 일반화 성능 평가

Machine Learning

모델 성능 평가

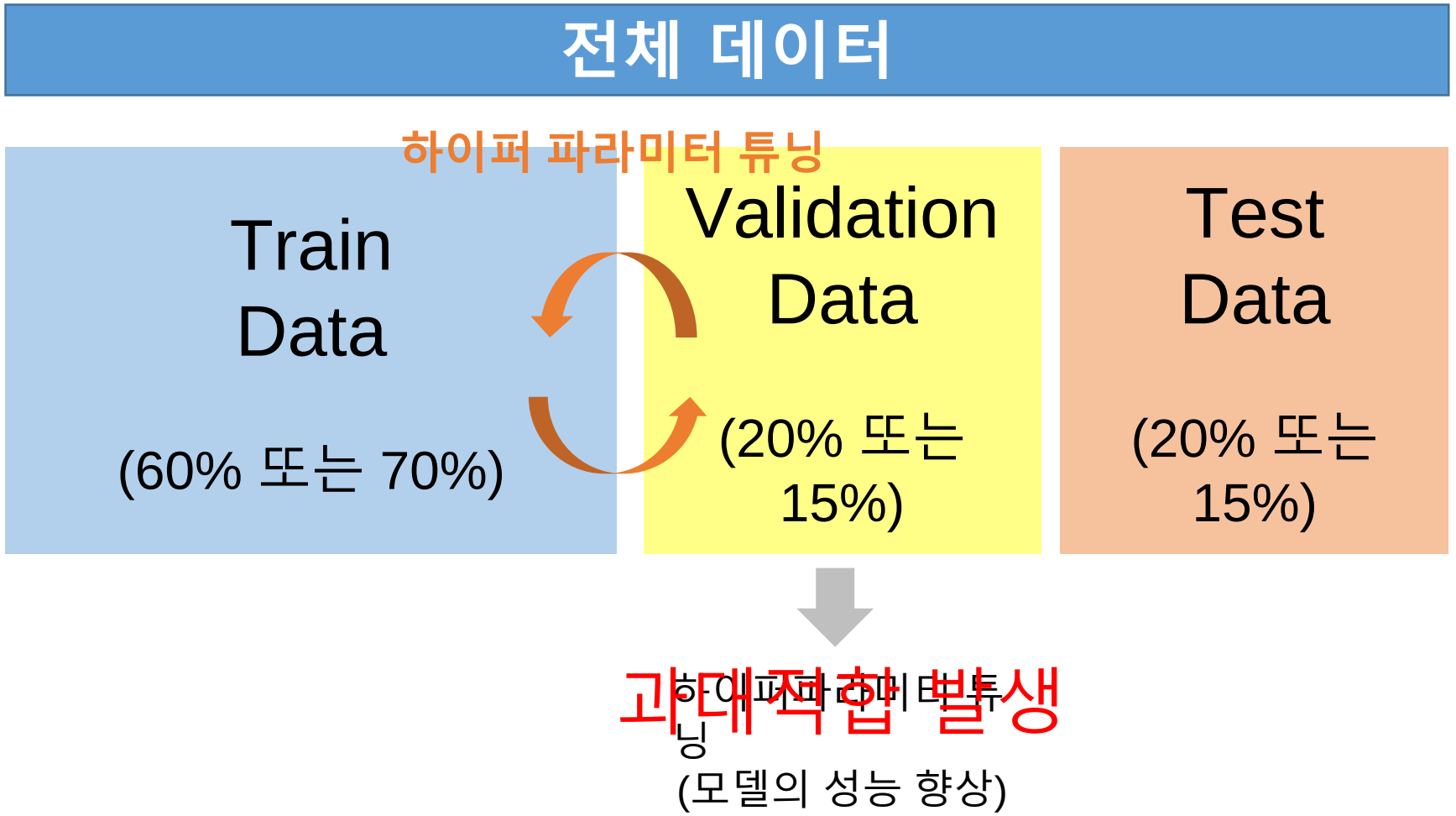
▶ 일반적인 경우



Machine Learning

모델 성능 평가

▶ 데이터의 분할과 용도



Machine Learning

모델 성능 평가

▶ **해결책 : 검증 Data를 하나로 고정하지 않고, Test 데이터의 모든 부분을 사용**

Machine Learning

모델 성능 평가

▶ K-fold cross-validation

1. Test 데이터를 k개의 그룹으로 나누기
2. K-1개의 그룹을 학습에 사용
3. 나머지 1개의 그룹을 이용해서 평가 수행
4. 2번, 3번 과정을 k번 반복
5. 모든 결과의 평균을 구하기

Machine Learning

모델 성능 평가

▶ K-fold cross-validation 동작 방법(K=5)

Machine Learning

모델 성능 평가K-fold cross-validation 장/단점

장점

1. 모든 Test 데이터 셋을 평가에 활용할 수 있다.
   - 평가에 사용되는 데이터 편중을 막을 수 있다.
   - 특정 평가 데이터 셋에 overfit 되는 것을 방지할 수 있다
2. 모든 Test 데이터 셋을 훈련에 활용할 수 있다.
   - 정확도를 향상시킬 수 있다.
   - 데이터 부족으로 인한 underfitting을 방지할 수 있다
3. 데이터 세트 크기가 충분하지 않은 경우에도 유용하게 사용 가능하다.

단점

1. 여러 번 학습하고 평가하는 과정을 거치기 때문에 모델 훈련/평가 시간이 오래 걸린다.

모델 성능 평가

▶ **K-fold cross-validation 사용법 :** cross_val_score() 함수

```
from sklearn.model_selection import cross_val_score

score = cross_val_score(모델, X, y, cv=나눌 개수)

score
```

```
[0.86680328 0.87704918 0.875      0.90554415 0.8788501 ]
```

THE END