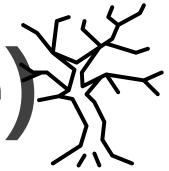
의사결정나무(Decision Tree)



비타민 8기 1조
강호재 서진슬 석민정 우상백

INDEX

01 의사결정나무 개요

01 의사결정나무 개요

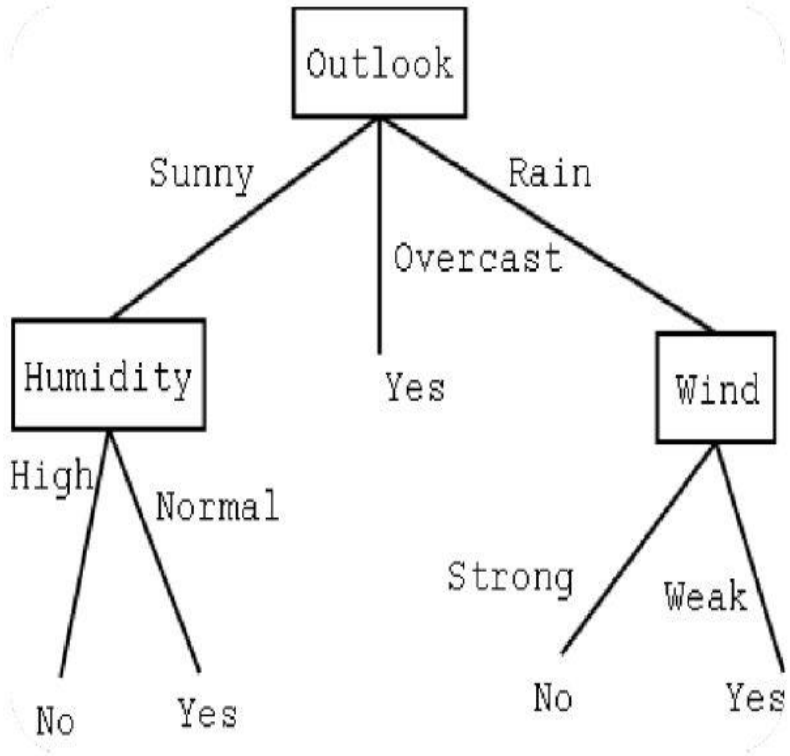
1. 의사결정나무란?

데이터에 내재 되어있는 패턴을 변수의 조합으로 나타내는
예측/분류 모델을 '나무의 형태로 만드는 것

- 질문을 던져서 맞고 틀리는 것에 따라 우리가 생각하고 있는 대상을 점점 좁혀나가는 알고리즘
- ex1) 스무고개 – 제시된 문제를 20번의 질문으로 알아맞히는 오락놀이 (의사결정나무와 비슷한 형태)
- ex2) 아키네이터 (akinator)

01 의사결정나무 개요

1. 의사결정나무란?



- 변수들로 기준을 만들고 이것을 통해 샘플을 분류하고 분류된 집단의 성질을 통하여 추정하는 모형
- 장점: 해석력이 높음, **직관적**, 범용성
- 단점: 높은 변동성, 표본(샘플)에 따라 트리구조가 달라질 수(민감)-> **과적합 가능성**

01 의사결정나무 소개

2. 알고리즘(공통)

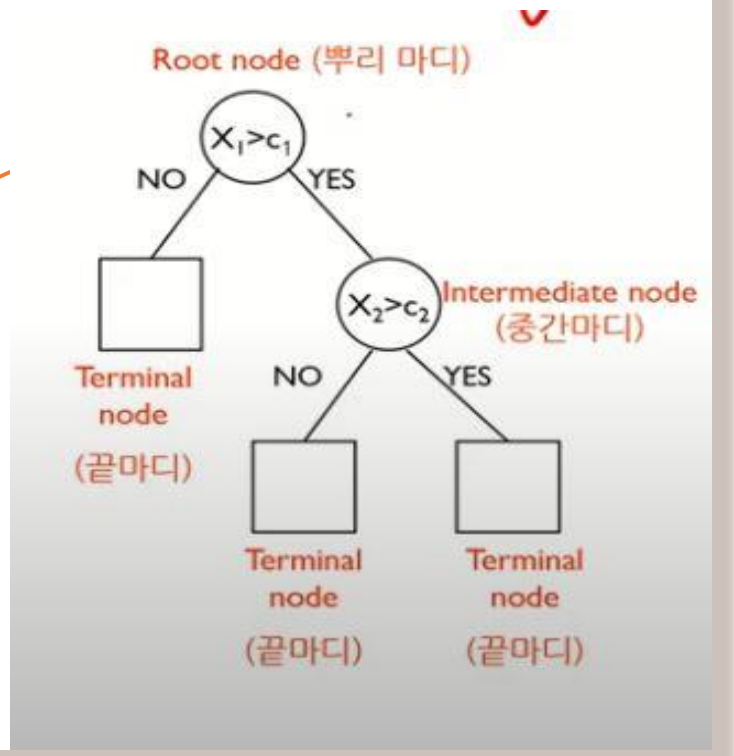
알고리즘 (공통)

Data

Input			Output
$X1$	…	Xp	Y(target)

Algorithm

Model



<알고리즘 설명>

-데이터를 **2개 혹은 그 이상의 부분집합**으로 분할 -> 데이터가 '**균일**'해지도록 분할

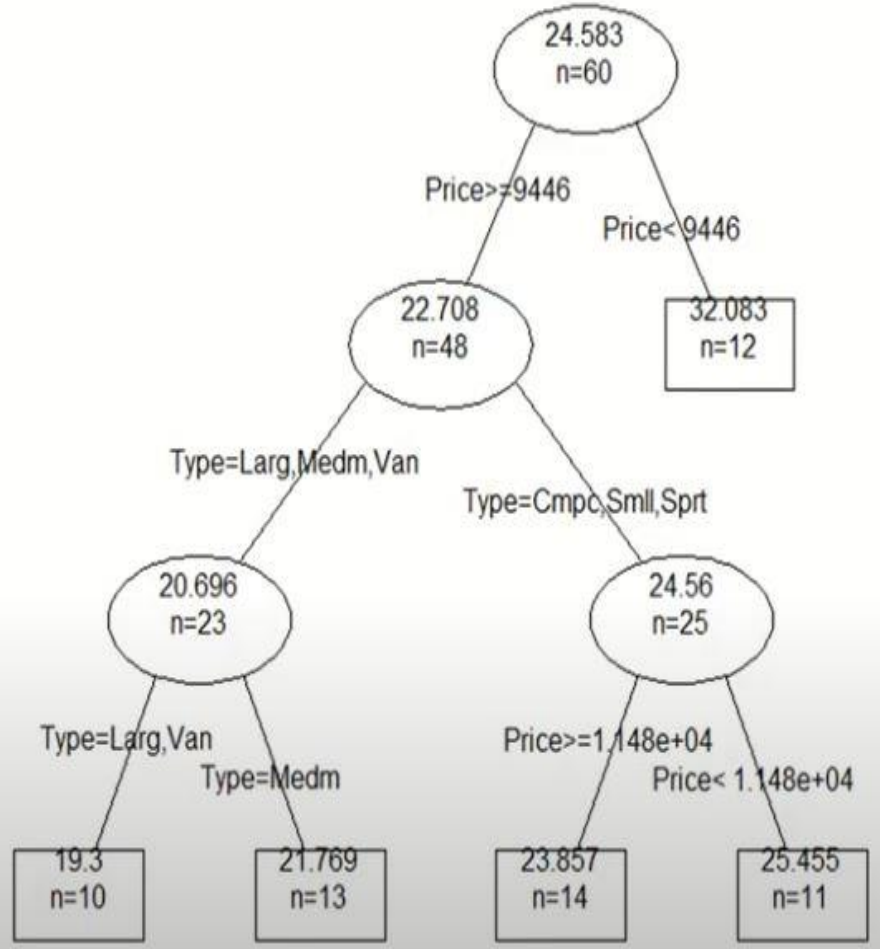
* 분류: **비슷한 범주**를 갖고 있는 관측치끼리

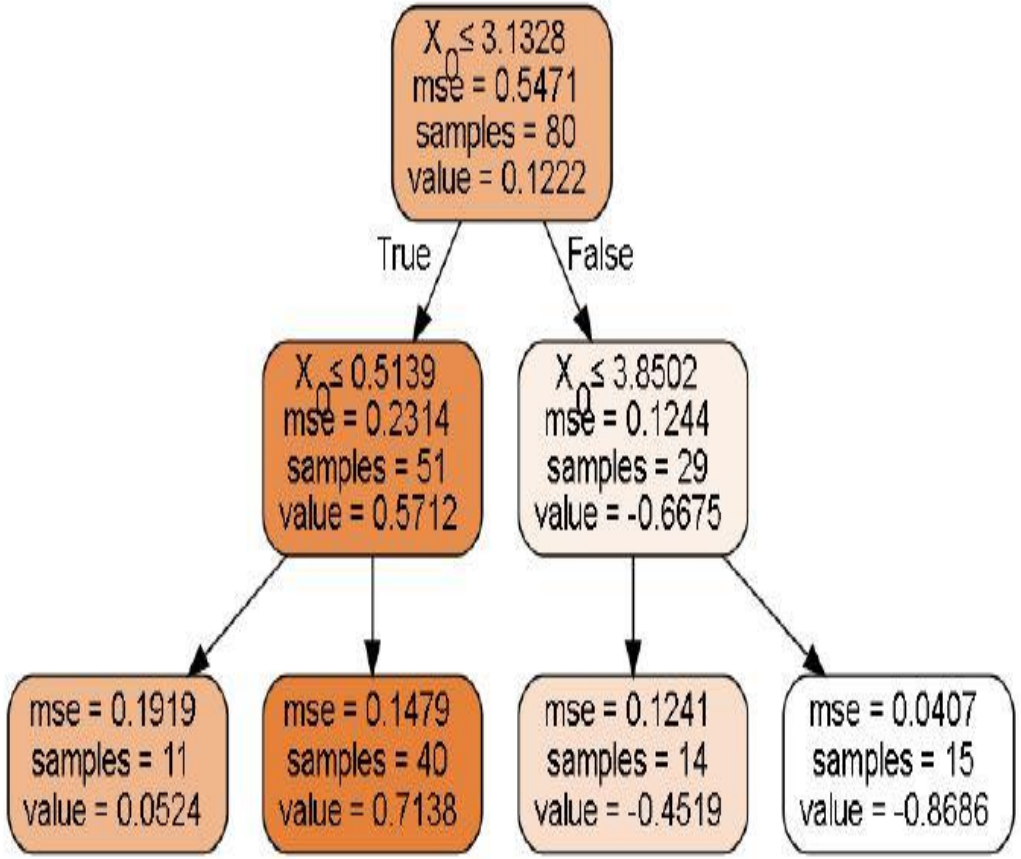
*예측: **비슷한 수치**를 갖고 있는 관측치끼리

02 예측나무모델(Regression Tree)

02 예측나무모델 (Regression Tree)

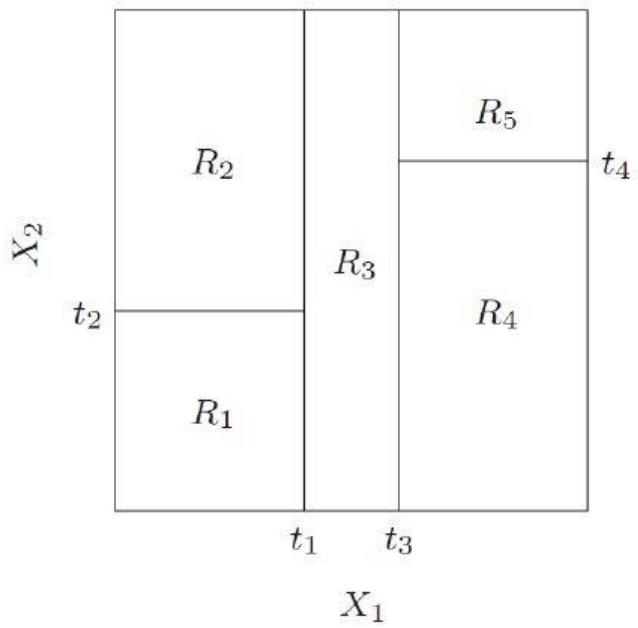
1. 이진분할 (이진트리)

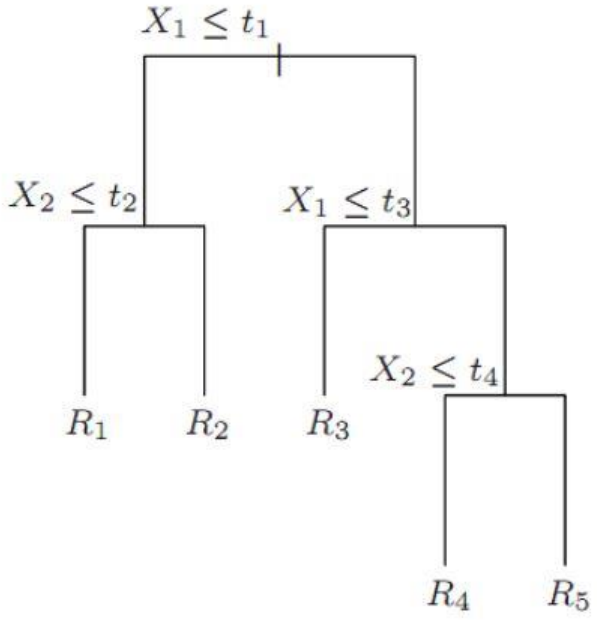


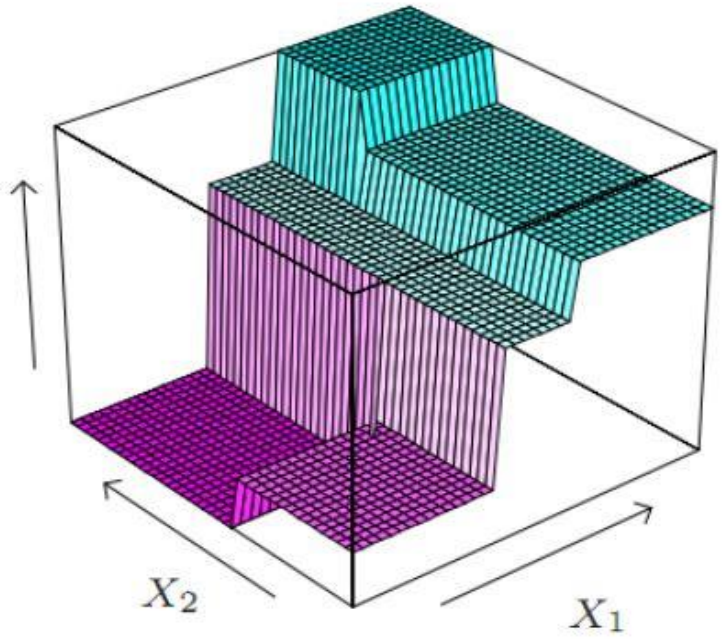


02 예측나무모델 (Regression Tree)

2. 표현 방법







세 개의 그림은 같은 표현! 하지만, 더 차원으로부터 자유로운 것은?

02 예측나무모델 (Regression Tree)

3. 최적 분할 방법

다음 비용함수를 '최소'로 할 때 최적의 분할이 형성된다.

$$\min_{C_m} \sum_{i=1}^{N} (y_i - f(x_i))^2$$

$$= \min_{C_m} \sum_{i=1}^{N} (y_i - \sum_{m=1}^{M} c_m I(x \in R_m))^2$$

$$\hat{c}_m = ave(y_i | x_i \in R_m)$$

즉, 각 분할에 속해 있는 y값들의 평균으로 예측 했을 때, 오류가 최소

02 예측나무모델 (Regression Tree)

3. 최적 분할 방법 – 분할변수(j), 분할점(s)의 결정

다음 식을 '최소'로 할 때 결정된다.

$$R_1(j,s) = \{x | x_j \leq s\}$$
$$R_2(j,s) = \{x | x_j > s\}$$

$$\underset{j,s}{\operatorname{argmin}} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

$$= \underset{j,s}{\operatorname{argmin}} \left[\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2 \right]$$

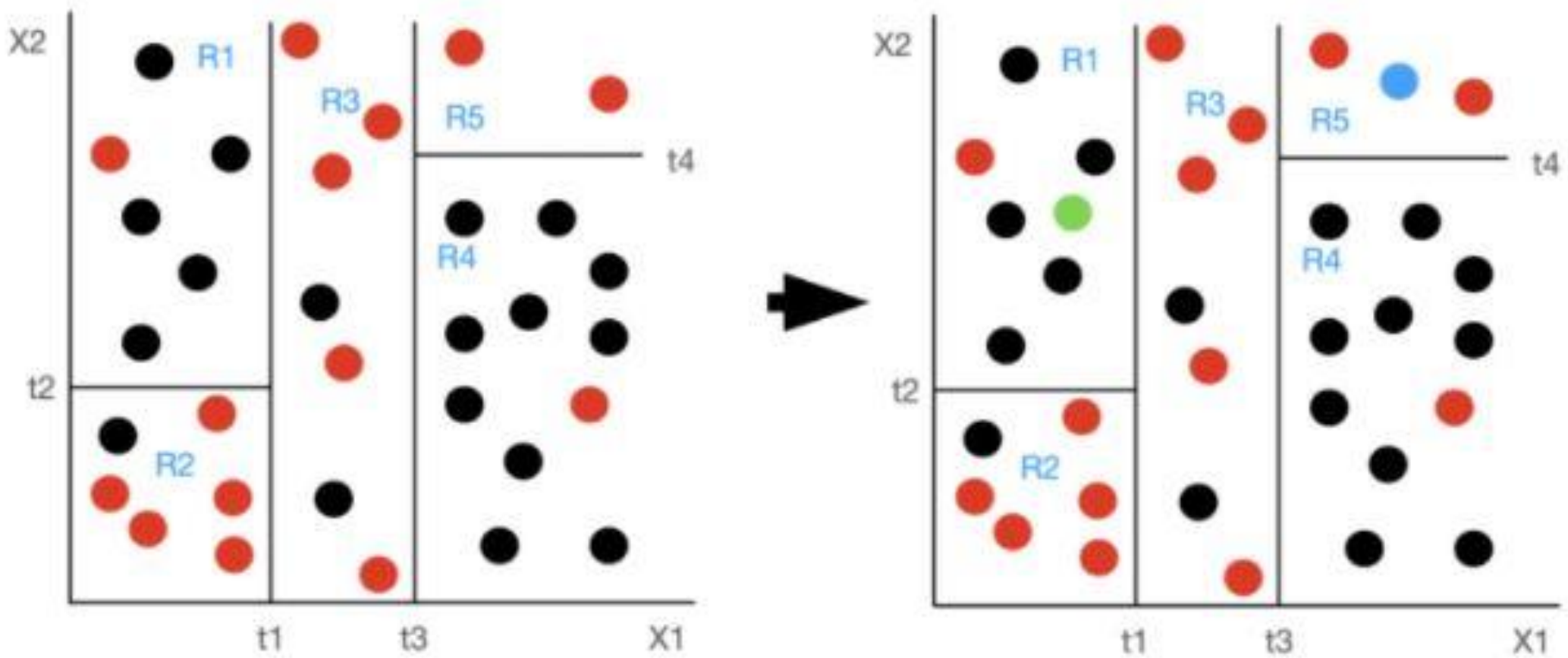
$$\hat{c}_1 = ave(y_i | x_i \in R_1(j,s) \text{ and } \hat{c}_2 = ave(y_i | x_i \in R_2(j,s)$$

이렇게 해서 의사결정나무는 분할변수(특성)과 분할점(임계점)을 결정해서 가지분할을 진행. → '탐욕적 알고리즘'(greedy algorithm)

03 분류나무모델 (*Classification Tree*)

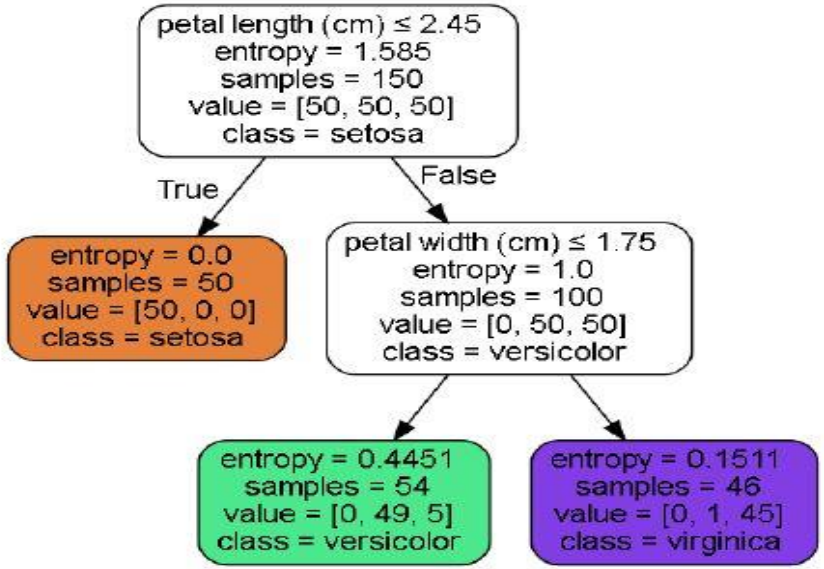
03 분류나무모델 (Classification Tree)

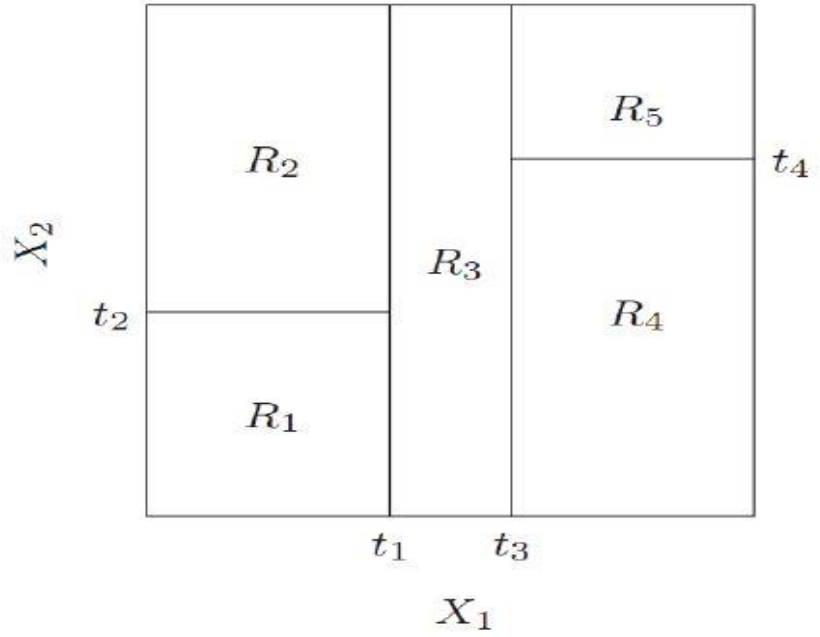
어떤 범주로 분류할까?



03 분류나무모델 (Classification Tree)

1. 표현 방법





두 개의 그림은 같은 표현! 하지만, 더 차원으로부터 자유로운 것은?

03 분류나무모델 (Classification Tree)

2. 최적 분할 방법

다음 비용함수를 '최소'로 할 때 최적의 분할이 형성된다.

1. 지니불순도(지니계수와 반대되는 개념)

$$G_i = 1 - \sum_{}^{n} \left(p_i^{(k)} \right)^2 \qquad s.t. \sum_{}^{n} p_i^{(k)} = 1$$

$$loss = \frac{N_{Left}}{N} \cdot G_{Left} + \frac{N_{Right}}{N} \cdot G_{Right}$$

$$= \frac{N_{Left}}{N} \cdot \left(1 - \sum_{k=1}^{n} \left(p_{Left}^{(k)} \right)^2 \right) + \frac{N_{Right}}{N} \cdot \left(1 - \sum_{k=1}^{n} \left(p_{Right}^{(k)} \right)^2 \right)$$

DecisionTreeClassifier 클래스
criterion 의 기본값

2.엔트로피

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

DecisionTreeClassifier 클래스
criterion = 'entropy'

둘다, '불순도'를 최소화 해야하기 때문에 ,
우리는 '순도'가 가장 높도록 하는 특징을 선별해서 가지를 쳐야한다.

03 분류나무모델 (Classification Tree)

2. 최적 분할 방법

*Information Gain (정보 이득, **불확실성 감소**)*

'비용함수'를 '최소'로 할 때 최적의 분할이 형성

결국, 맨 마지막 노드에서는 '불순도'가 가장 최소가 되도록

부모노드와 자식노드의 '불순도의 차이'는 가장 최대가 되도록

1. 지니불순도(지니계수와 반대되는 개념)

Information gain

불순도 차이 =
부모의 gini
- [(왼쪽 노드 샘플 수 / 부모의 샘플 수) x
왼쪽 노드 gini
+ (오른쪽 노드 샘플 수 / 부모의 샘플 수) x
오른쪽 노드 gini]
(※if '이진분할')

2.엔트로피

Information gain

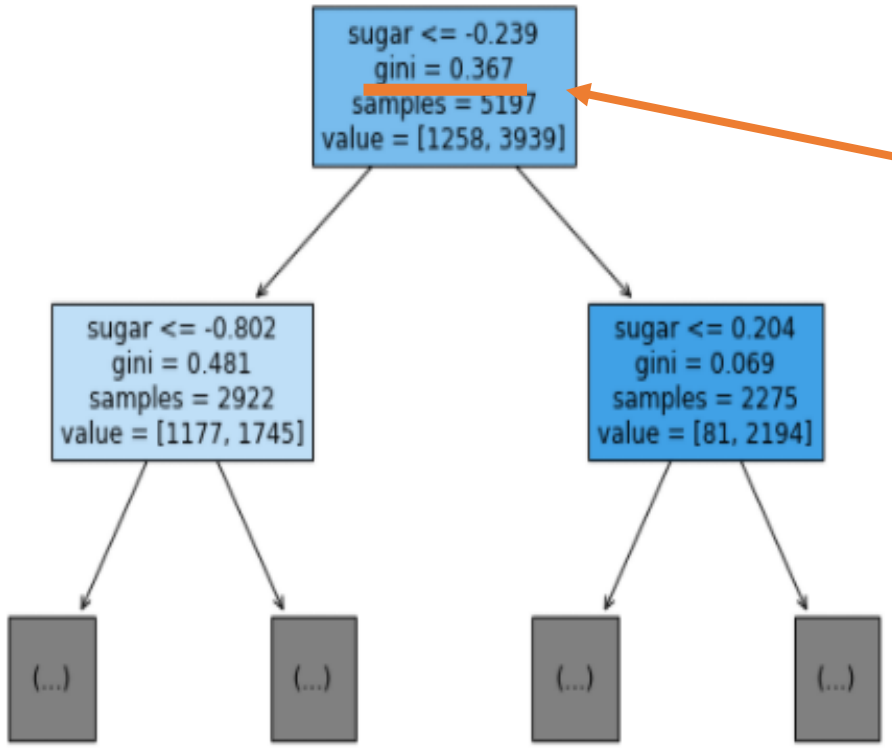
불순도 차이
= Entropy(부모) –
[p(자식$_1$) x Entropy(자식$_1$) + p(자식$_2$) x Entropy(자식$_2$) + …]

03 분류나무모델 (Classification Tree)

3. 지니불순도

지니불순도 *(지니계수와 반대되는 개념)*

불순도 차이



$$Q1.\ gini?$$

지니불순도= 1−(음성 클래스 비율^2 + 양성 클래스 비율^2)

Ex.) 맨 위쪽 부모 노드의 gini =
1- ((1258/5197)^2 + (3939/5197)^2) = 0.367

$$Q2.\ Information\ gain?$$

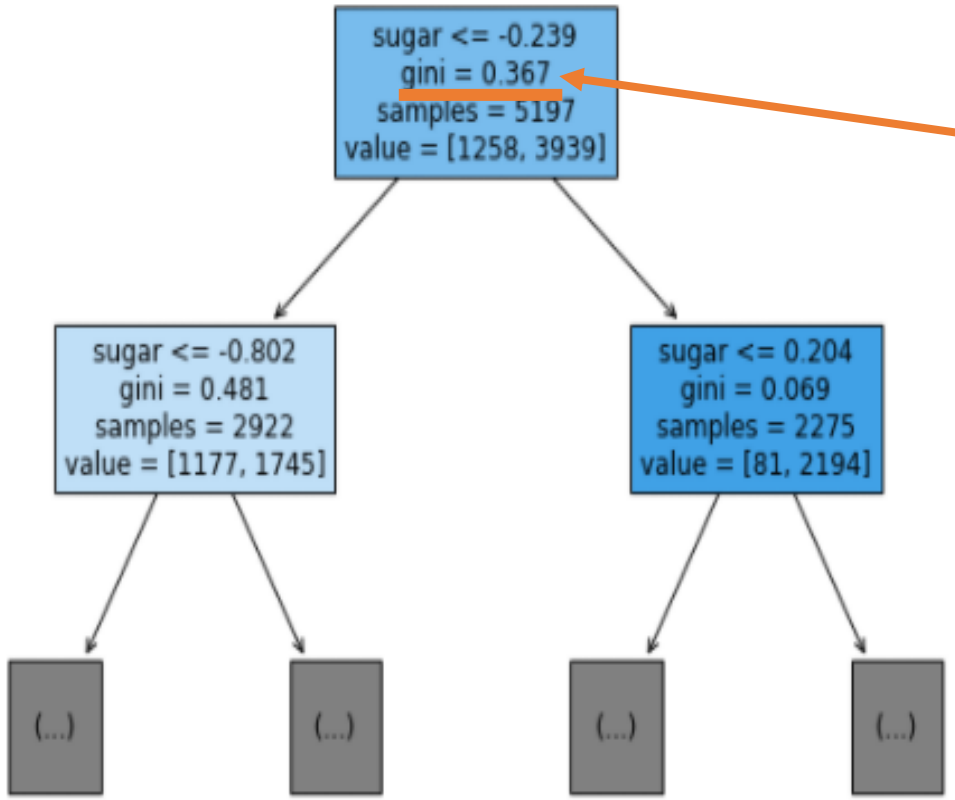
불순도 차이 =

부모의 불순도
- (왼쪽 노드 샘플 수 / 부모의 샘플 수)x 왼쪽 노드 불순도
- (오른쪽 노드 샘플 수 / 부모의 샘플 수) x 오른쪽 노드 불순도

∴0.367 - (2922 / 5197) x 0.481 - (2275 / 5197) x 0.069 = 0.066 >0

03 분류나무모델 (Classification Tree)

4. 엔트로피



Q1. Entropy?

엔트로피 = - 음성 클래스 비율 x log2(음성 클래스 비율)
- 양성 클래스 비율 x log2(양성 클래스 비율)

Ex.) 맨 위쪽 부모 노드의 entropy =

-(1258 / 5197) x log2(1258 / 5197) - (3939 / 5197) x log2(3939 / 5197) = 0.798

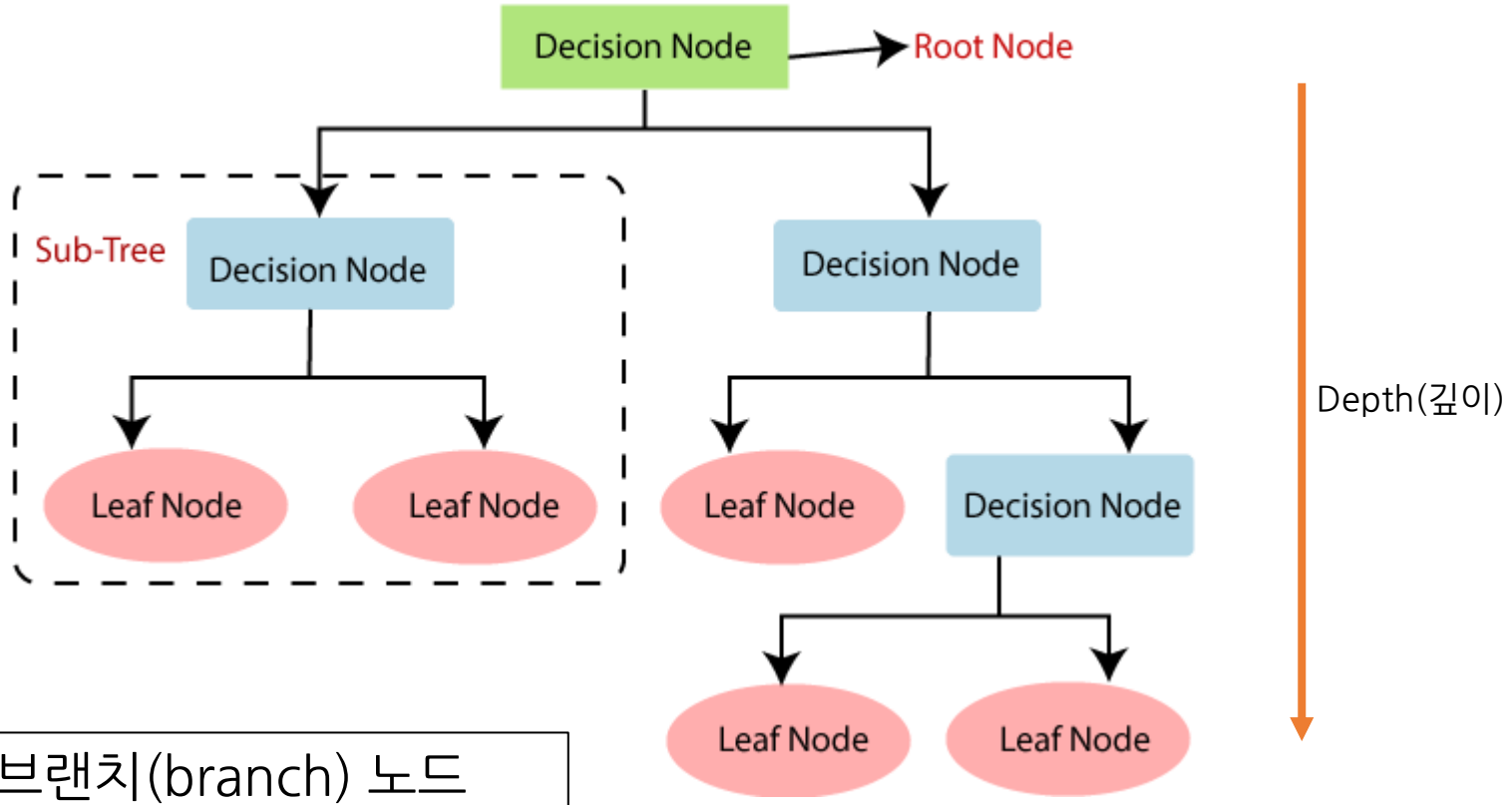
Q2. Information gain?

불순도 차이 = 부모의 불순도
- (왼쪽 노드 샘플 수 / 부모의 샘플 수) **x 왼쪽 노드 불순도**
- (오른쪽 노드 샘플 수 / 부모의 샘플 수) x **오른쪽 노드 불순도**

.:0.798 - (2922 / 5197) x 0.973 - (2275 / 5197) x 0.222= 0.153 >0

03 분류나무모델 (Classification Tree)

5. 의사결정나무 시각화



자식 노드 O → 브랜치(branch) 노드
자식노드 X → 리프(leaf) 노드

03 분류나무모델 (Classification Tree)

5. 의사결정나무 시각화

Graphviz 패키지의 export_graphviz()함수를 사용하여
결정 트리를 시각화

시각화에 필요한 패키지(Graphviz) 설치

```
! pip install graphviz
```

export_graphviz( )

함수인자

- 학습이 완료된 estimator,
- 피처의 이름(리스트),
- 레이블 이름(리스트)
- rounded : 노드의 모양
- filled : 노드 채우기
- out_file : output 파일 명

→ Output : 트리 형태로 시각화

03 분류나무모델 (Classification Tree)

5. 의사결정나무 시각화

```
from sklearn.datasets import load_iris
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
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)

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

```
DecisionTreeClassifier(random_state=156)
```

export_graphviz( )

함수인자

- 학습이 완료된 estimator,
- 피처의 이름(리스트),
- 레이블 이름(리스트)
- rounded : 노드의 모양
- filled : 노드 채우기
- out_file : output 파일 명

→ Output : 트리 형태로 시각화

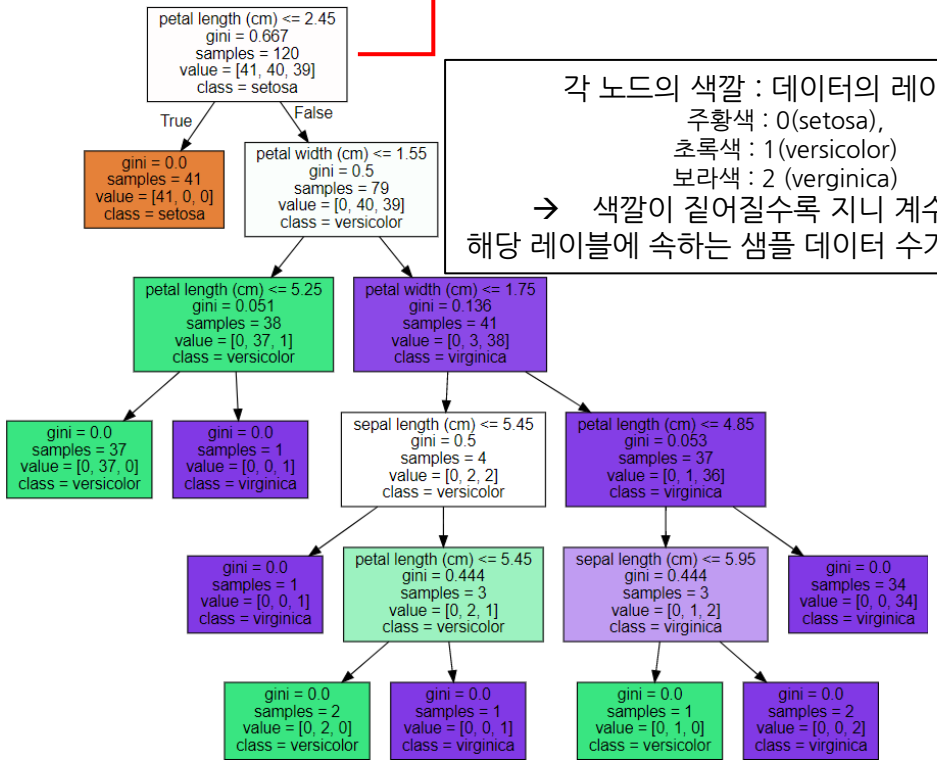
```
# export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함.
export_graphviz(dt_clf, out_file="tree.dot", class_names=iris_data.target_names,
                feature_names=iris_data.feature_names, impurity=True, filled=True)
```

export_graphviz()는 Graphviz가 읽어 들여서 그래프 형태로 시각화할 수 있는 출력 파일을 생성한다.
위의 코드에서는 "tree.dot" 파일을 생성하였다.

03 분류나무모델 (Classification Tree)

5. 의사결정나무 시각화

```
1 import graphviz
2
3 # 위에서 생성된 tree.dot 파일을 Graphviz 읽어서 Jupyter Notebook상에서 시각화
4 with open("tree.dot") as f:
5     dot_graph = f.read()
6 graphviz.Source(dot_graph)
```



각 노드의 색깔 : 데이터의 레이블 값
주황색 : 0(setosa),
초록색 : 1(versicolor)
보라색 : 2 (verginica)
→ 색깔이 짙어질수록 지니 계수가 낮고
해당 레이블에 속하는 샘플 데이터 수가 많다는 의미

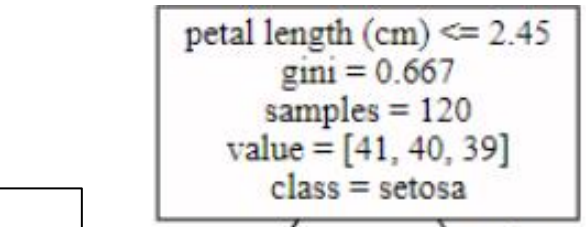
petal length (cm) <= 2.45
gini = 0.667
samples = 120
value = [41, 40, 39]
class = setosa

Petal length(cm) <= 2.45	자식 노드를 만들기 위한 규칙 조건. 조건이 없으면 해당 노드는 리프 노드
gini = 0.667	다음의 value 안에 데이터 분포에서의 지니 계수
Samples = 120	현재 노드의 데이터 수
Value = [41, 40, 39]	Class값 기반의 데이터 수 클래스 값 순서로 Setosa : 41개, Vesicolor : 40개, Virginica : 39개
Class = setosa	하위 노드를 가질 경우 setosa의 개수가 41개로 제일 많다는 뜻

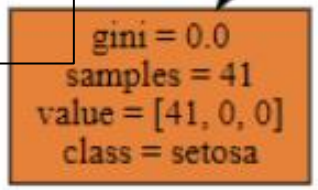
03 분류나무모델 (Classification Tree)

5. 의사결정나무 시각화

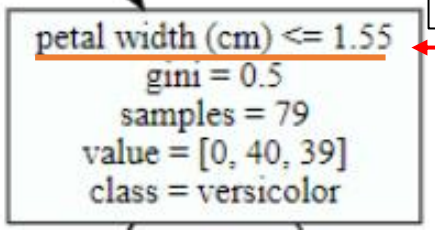
1번 노드



2번 노드
- petal length <= 2.45가 True인 규칙노드
- 41개 샘플 모두 setosa → 예측 class는 setosa
  → 리프 노드(분기X)
- 지니 계수 : 0



2번 노드

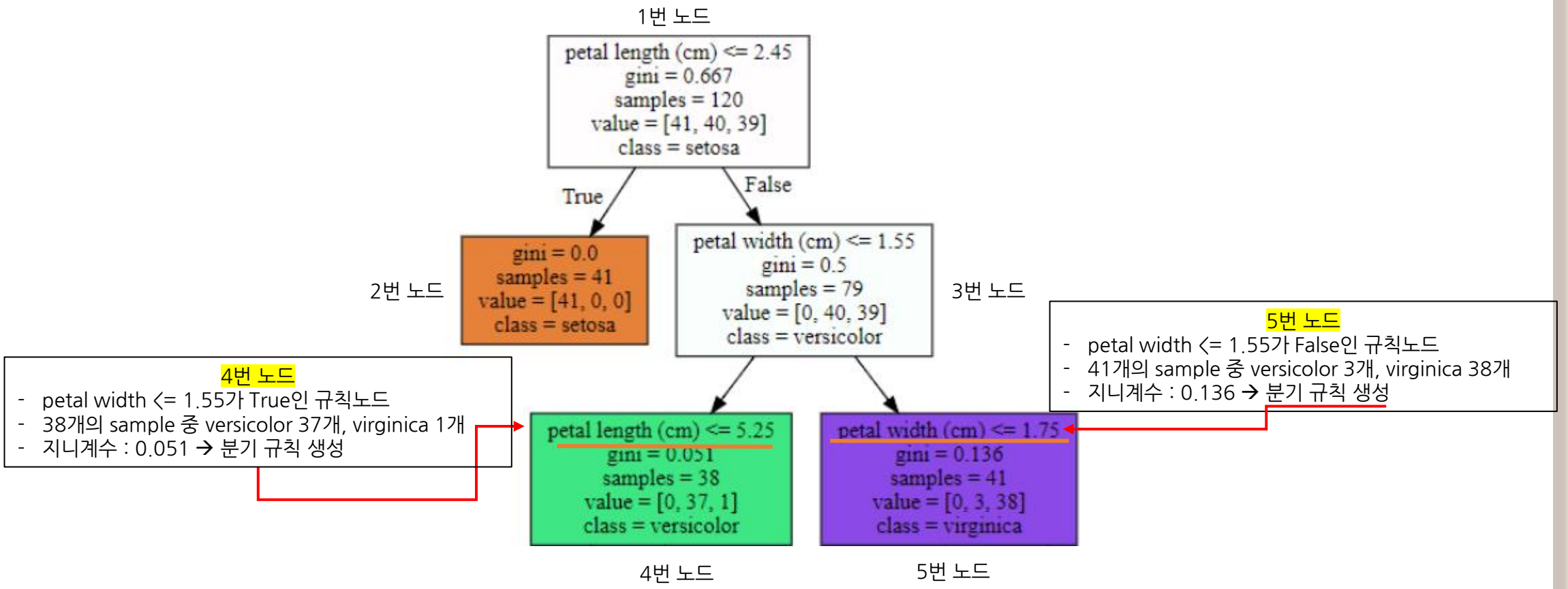


3번 노드

3번 노드
- petal length <= 2.45가 False인 규칙노드
- 79개의 sample 중 versicolor 40개, virginica 39개
- 지니계수 : 0.5 → 분기 규칙 생성

03 분류나무모델 (Classification Tree)

5. 의사결정나무 시각화



03 분류나무모델 (Classification Tree)

6. 파라미터

* 분류나무모델과 예측나무모델 둘다 동일한 파라미터를 사용 *

초모수(Parameter)설정 → 과적합방지

Decision tree는 굉장히 많은 파라미터를 가지고 있다.
대표적으로 다음의 5가지를 살펴보도록 하자.

Parameter	설명
★min_samples_split	노드를 분할하기 위한 최소한의 데이터 sample 수. Default=2이며, 값이 작을수록 하나의 노드의 수가 늘어나 overfitting의 우려가 커짐.
min_samples_leaf	Leaf node가 되기 위한 최소한의 데이터 sample 수 일반적으로 작게 설정할수록 overfitting의 우려가 커짐.
max_features	노드를 분할할 때 고려할 특성의 최대 개수.
★max_depth	트리의 최대 깊이. - Default = None : 완벽한 분류기가 만들어지거나 min_samples_split보다 작아질 때까지 계속 깊이를 증가시킴. - 너무 커지면 overfitting의 우려가 있으므로 적절하게 설정.
max_leaf_nodes	트리가 가질 leaf node의 최대 개수. Default=None으로 제한 없이 가질 수 있게 됨

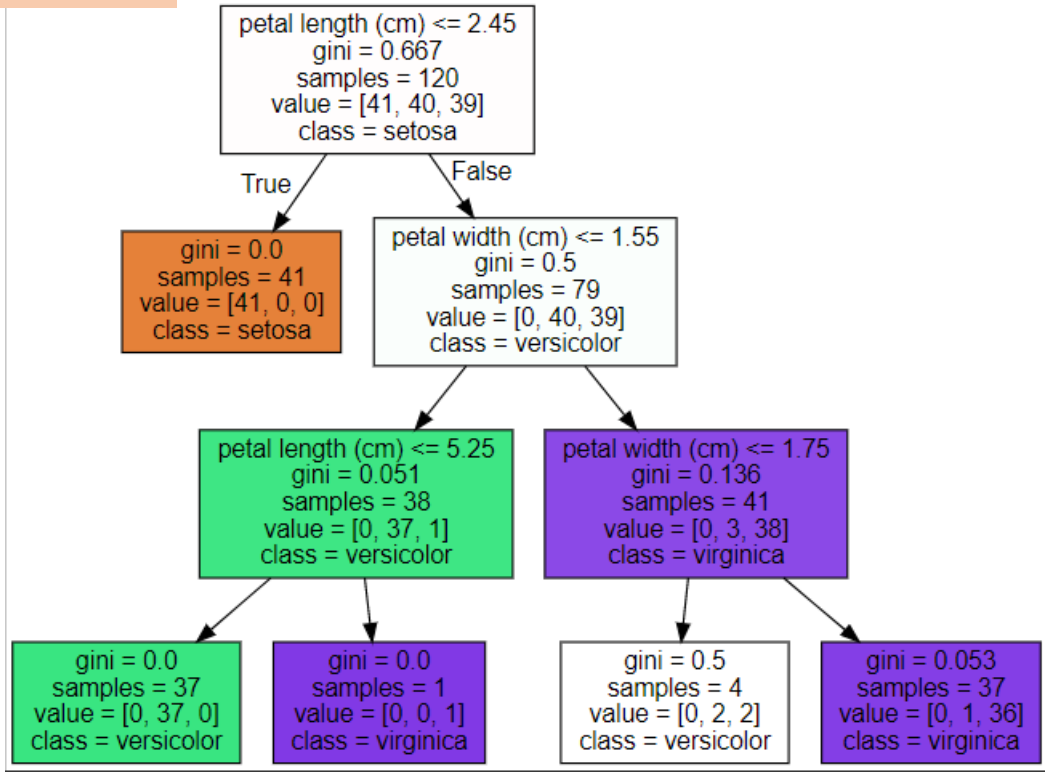
03 분류나무모델 (Classification Tree)

6. 파라미터 - max_depth

트리 최대 깊이 제어

max_depth = 3인경우 Decision Tree

```
##  max_depth = 3인 경우 Decision Tree ##
dt_clf = DecisionTreeClassifier(random_state=156,
                                max_depth=3)
dt_clf.fit(X_train, y_train)

# 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)
import graphviz
# 위에서 생성된 tree.dot 파일을 Graphviz 읽어서 Jupyter Notebook상에서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



기존의 트리보다 더 간결한 트리가 생성된다.

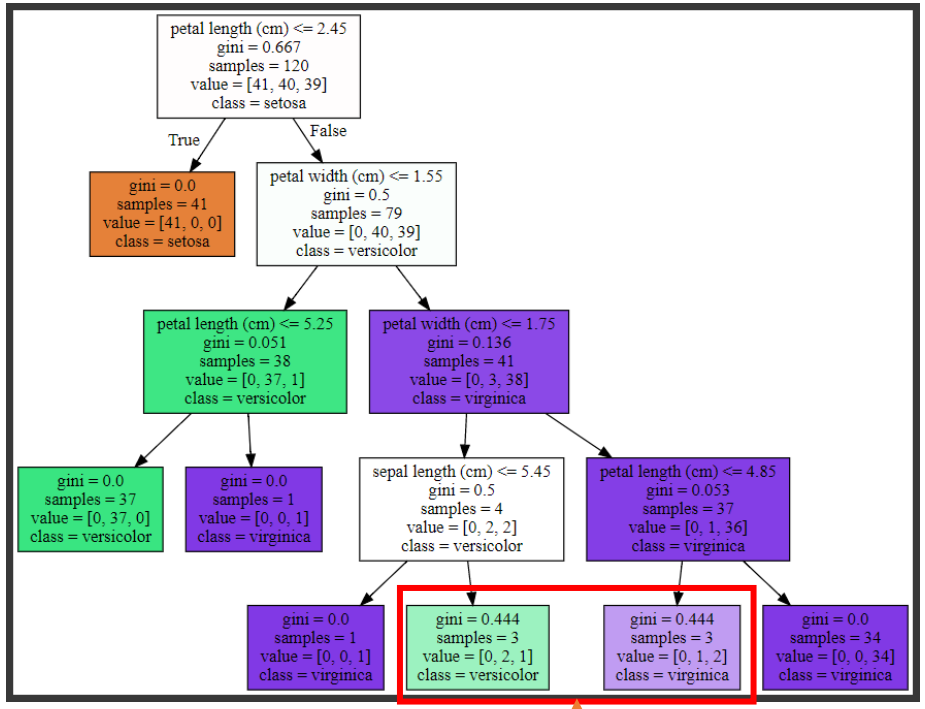
03 분류나무모델 (Classification Tree)

6. 파라미터 - min_samples_split

노드를 분할하기 위한 최소한의 데이터 sample 수

min_samples_split = 4인경우 Decision Tree

```
## min_samples_split = 4인경우 Decision Tree ##
dt_clf = DecisionTreeClassifier(random_state=156,
                                min_samples_split=4)
dt_clf.fit(X_train, y_train)

# 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)
import graphviz
# 위에서 생성된 tree.dot 파일을 Graphviz 읽어서 Jupyter Notebook상에서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



즉 최소 4개의 sample이 있어야 자식 노드를 만들 수 있다.

→ 4개 보다 작은 sample 수를 지닐 경우, sample 내 상이한 값이 있더라도 더 이상 분할 안하고 리프 노드가 된다.

→ 트리 깊이도 줄고 간결한 트리가 생성됨.

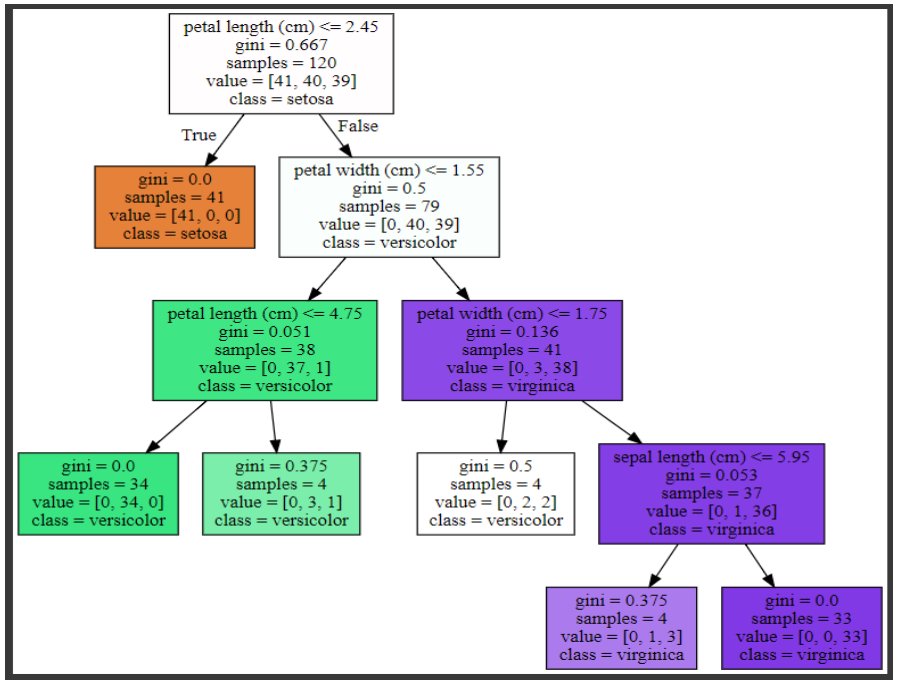
03 분류나무모델 (Classification Tree)

6. 파라미터 - min_samples_leaf

리프 노드가 되기 위한 최소한의 sample 데이터 수

min_samples_leaf = 4인경우 Decision Tree

```
## min_samples_leaf = 4인경우 Decision Tree ##
dt_clf = DecisionTreeClassifier(random_state=156,
                                min_samples_leaf=4)
dt_clf.fit(X_train, y_train)

# 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)
import graphviz
# 위에서 생성된 tree.dot 파일을 Graphviz 읽어서 Jupyter Notebook상에서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



min_samples_leaf 값을 키울 수록 리프 노드가 될 조건이 완화된다.

sample 수 ≤ min_samples_leaf(4) → 리프노드

조건을 만족하면

04 실습

04 실습

1. titanic

1) 패키지 import

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import confusion_matrix
```

2) 데이터 불러오기

```
1 # 데이터 불러오기
2 import pandas as pd
3 titanic_data = pd.read_csv("titanic_data_clean.csv")
```

3) 데이터 살펴보기

```
1 titanic_data
```

	Pclass	Sex	Fare	Survived
0	3	male	7.2500	0
1	1	female	71.2833	1
2	3	female	7.9250	1
3	1	female	53.1000	1
4	3	male	8.0500	0
...	...	...	...	...
886	2	male	13.0000	0
887	1	female	30.0000	1
888	3	female	23.4500	0
889	1	male	30.0000	1
890	3	male	7.7500	0

891 rows × 4 columns

Feature target

```
1 titanic_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    891 non-null    int64
 1   Sex       891 non-null    object
 2   Fare      891 non-null    float64
 3   Survived  891 non-null    int64
dtypes: float64(1), int64(2), object(1)
memory usage: 28.0+ KB
```

Feature
Pclass(선실등급) : 범주형 변수
Sex (성별) : 범주형 변수
Fare (요금) : 연속형 변수

target
Survived (0 : 사망, 1 : 생존)
→ 2개의 class

04 실습

1. titanic

4) feature와 target 나누기

```
1 # Features와 target 나누기
2 t_features = titanic_data[titanic_data.columns[:-1]]
3 t_target = titanic_data[titanic_data.columns[-1]]
```

5) 범주형 변수 처리

- Sex 변수 :
  female → 0, male → 1 으로 변환

- Pclass 변수 :
  One-hot Encoding

```
1 # Sex변수 : female, male
2 # female --> 0, male --> 1 으로 변환
3 t_features["Sex"] = t_features.Sex.map({"female":0, "male":1})
```

```
1 # Pclass 변수 : One-hot Encoding으로 처리해준다.
2 t_features = pd.get_dummies(data = t_features, columns = ['Pclass'], prefix = 'Pclass')
```

* 의사결정나무는 데이터 스케일에 영향을 받지 않음 *
→ 정규화, 표준화 필요 X

6) Train / Test data 분리

```
1 # train: test = 8:2 분리
2 train_features, test_features , train_target, test_target = train_test_split(t_features, t_target, test_size = 0.2
3                                                                                , random_state = 2021, stratify=t_target)
```

04 실습

1. titanic

7) Under Sampling

```
# under sampling 전, survived의 데이터 수
pd.DataFrame(train_target)['Survived'].value_counts()
```

```
0    439
1    273
Name: Survived, dtype: int64
```

불균형 데이터 → Under Sampling 으로 처리

Random Under Sampling 사용

```
# Random Under Sampling

## sampling하기 전에 shuffling 해주기(행 순서 섞기)
import sklearn
x_shuffled = sklearn.utils.shuffle(train_features, random_state=2021)
y_shuffled =sklearn.utils.shuffle(train_target, random_state=2021)

# Random Under Sampling으로 target값(0, 1)을 각각 273으로 만든다.
import imblearn
from imblearn.under_sampling import RandomUnderSampler
train_features_us, train_target_us = RandomUnderSampler(random_state=2021).fit_resample(x_shuffled, y_shuffled)
```

```
# under sampling 후, survived의 데이터 수
pd.DataFrame(train_target_us)['Survived'].value_counts()
```

```
1    273
0    273
Name: Survived, dtype: int64
```

더 작은 1의 개수(273)에 맞게 감소
→ 1 (생존) : 273
→ 0 (사망) : 273

04 실습

1. titanic

8) Decision Tree Modeling

```
1 from sklearn.tree import DecisionTreeClassifier
2
3 # DecisionTree Classifier 생성
4 tree = DecisionTreeClassifier(criterion='gini',random_state = 2021)
5
6 # DecisionTreeClassifier 학습
7 tree_fit = tree.fit(train_features_us, train_target_us)
```

9) Decision Tree 시각화

Graphviz 패키지의 export_graphviz()함수를 사용하여
결정 트리를 시각화

시각화에 필요한 패키지(Graphviz) 설치

```
1 pip install graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10.1)
```

```
1 # 시각화에 필요한 library import
2 from sklearn.tree import export_graphviz
```

export_graphviz( )

함수인자

- 학습이 완료된 estimator,
- 피처의 이름(리스트),
- 레이블 이름(리스트)
- rounded : 노드의 모양
- filled : 노드 채우기

→ Output : 트리 형태로 시각화

```
1 # 피처의 이름 (리스트)
2 feature_names = train_features_us.columns.tolist()
3 # 클래스(target) 이름
4 target_name = np.array(['Dead', 'Survive'])
```

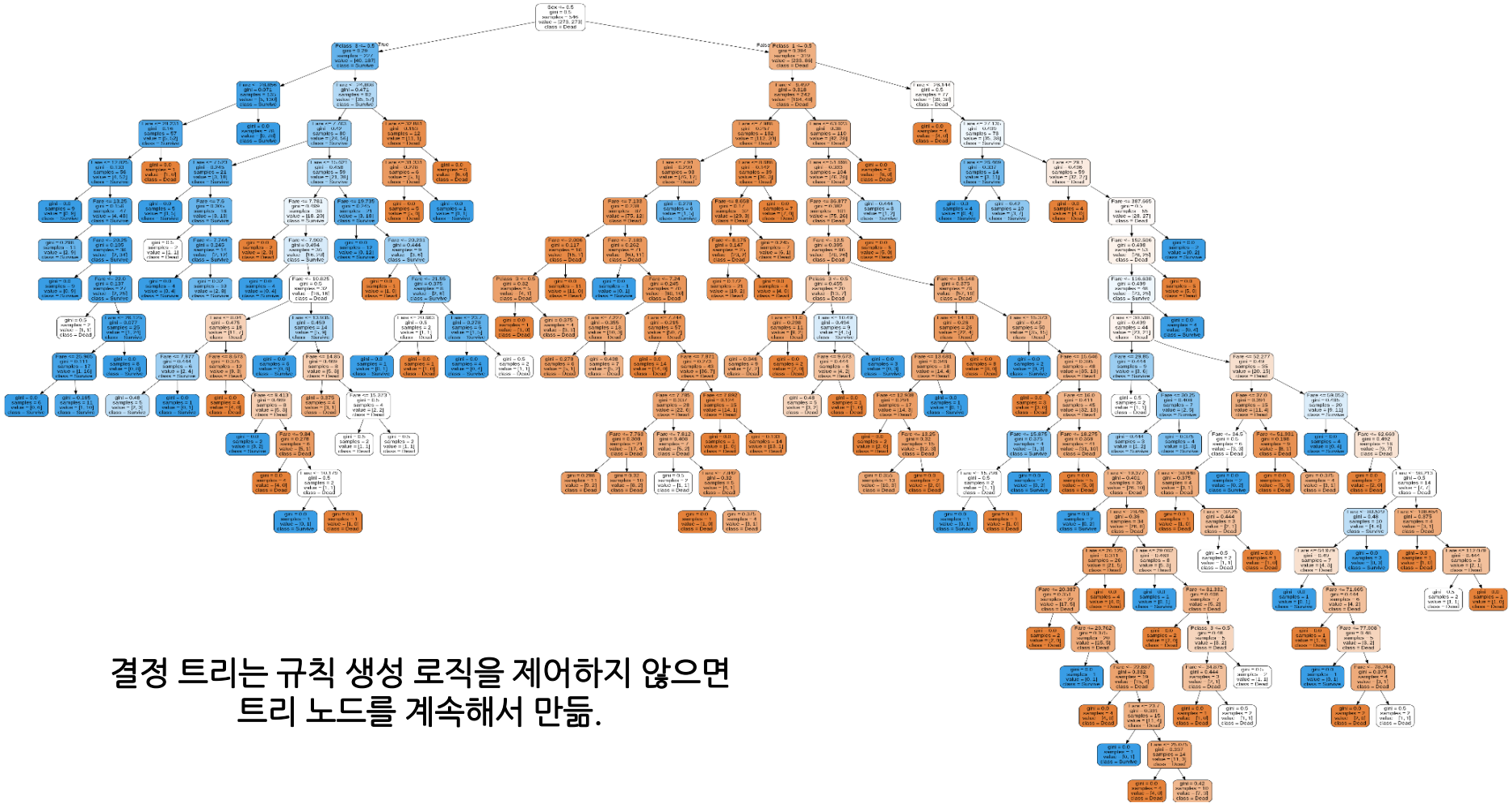
```
1 dot_data = export_graphviz(tree,
2                            filled = True,
3                            rounded = True,
4                            class_names = target_name,
5                            feature_names = feature_names,
6                            )
7
8 import graphviz
9 graphviz.Source(dot_data)
```

→ 결과는 다음 페이지!

04 실습

1. titanic

9) Decision Tree 시각화 결과



결정 트리는 규칙 생성 로직을 제어하지 않으면
트리 노드를 계속해서 만듦.

04 실습

1. titanic

10) 교차검증

```
# 교차 검증을 통한 성능 평가
scores = cross_validate(estimator = tree,
                        X=train_features_us,
                        y=train_target_us,
                        scoring = ['accuracy'],
                        cv=10,
                        n_jobs= -1,
                        return_train_score=False)

print('CV accuracy: %s' % scores['test_accuracy'])
print('CV aacuracy(Mean): %.3f (std: %.3f)' % (np.mean(scores['test_accuracy']),
                                               np.std(scores['test_accuracy'])))
```

```
CV accuracy: [0.83636364 0.8        0.72727273 0.85454545 0.76363636 0.81818182
 0.7037037  0.85185185 0.81481481 0.85185185]
CV aacuracy(Mean): 0.802 (std: 0.051)
```

교차 검증별 정확도

← 평균 검증 정확도

11) test데이터로 예측

```
# test데이터로 예측하기
y_pred = tree.predict(test_features)
y_pred
```

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

12) 오차행렬 구하기

```
# 오차행렬 구하기
confmat = pd.DataFrame(confusion_matrix(test_target, y_pred),
                       index=['True[0]', 'True[1]'],
                       columns=['Predict[0]', 'Predict[1]'])

confmat
```

	Predict[0]	Predict[1]
True[0]	TN 91	FP 19
True[1]	FN 16	TP 53

13) 분류 모델 평가 지표 구하기

```
# 분류 모델 평가 지표(정확도, 정밀도, 재현율, F1-score, AUC) 구하기
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

print('정확도 accuracy: %.3f' % accuracy_score(test_target, y_pred))
print('정밀도 precision: %.3f' % precision_score(y_true= test_target, y_pred=y_pred))
print('재현율 recall: %.3f' % recall_score(y_true=test_target, y_pred=y_pred))
print('F1-score: %.3f' % f1_score(y_true=test_target, y_pred=y_pred))
print('AUC: %.3f' % roc_auc_score(test_target, y_pred))
```

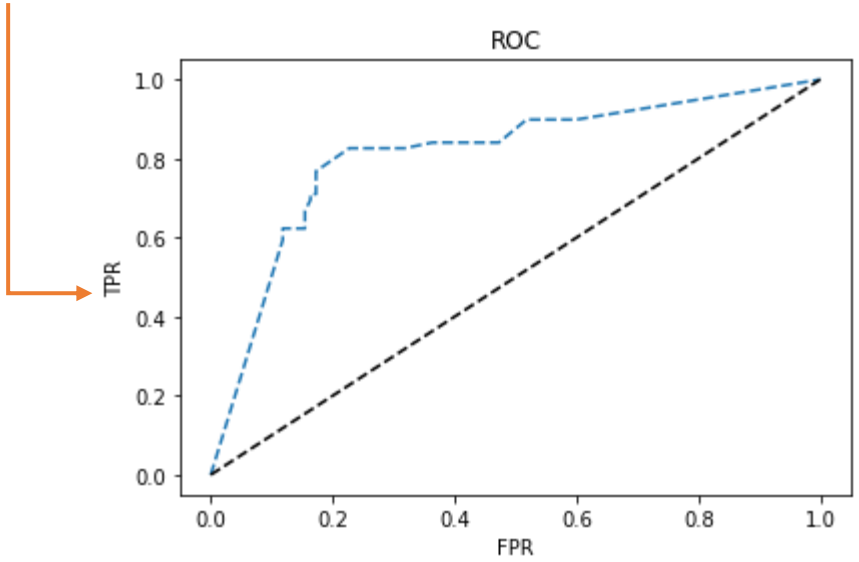
```
정확도 accuracy: 0.804
정밀도 precision: 0.736
재현율 recall: 0.768
F1-score: 0.752
AUC: 0.798
```

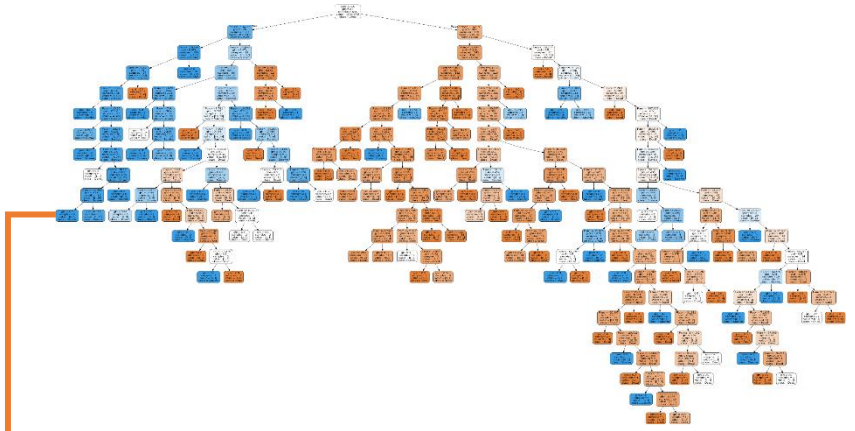
04 실습

1. titanic

14) ROC 커브

```
# ROC 커브를 통한 성능 확인
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve( test_target, tree.predict_proba(test_features)[:, 1] )

plt.plot(fpr, tpr, '--', label = 'Decision Tree')
plt.plot([0,1], [0,1], 'k--', label='random guess')
plt.plot([fpr],[tpr],'r-', ms=10)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC')
plt.show()
```





위 결과에서 볼 수 있듯이,
결정 트리는 규칙 생성 로직을 제어하지 않으면
트리 노드를 계속해서 만듦.

과적합 발생

이를 방지하고자 파라미터를 튜닝을 한다.
(다음장)

04 실습

1. titanic

15) 파라미터 튜닝

```
# DecisionTree Classifier 생성
dt_clf = DecisionTreeClassifier(random_state=2021)

# DecisionTreeClassifier의 하이퍼 파라미터 추출
print('DecisionTreeClassifier 기본 하이퍼 파라미터 : #n', dt_clf.get_params())
```

DecisionTreeClassifier 기본 하이퍼 파라미터 :
{'ccp_alpha': 0.0, 'class_weight': None,
'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'random_state': 2021, 'splitter': 'best'}

튜닝할 파라미터 (max_depth, min_sample_leaf, criterion)

```
## 튜닝할 파라미터 (max_depth, min_sample_leaf, criterion)

# param_range1 : max_depth(트리 최대 깊이)
param_range1 = [1,2,3,4,5,6,7,8,9,10]
# param_range2 : min_sample_leaf ( 말단 노드의 최소 개수 )
param_range2 = [10, 20, 30, 40, 50]
# param_range3 : criterion (불순도 계산 방법)
param_range3 = ['gini', 'entropy']

param_grid = [{'max_depth': param_range1,
               'min_samples_leaf': param_range2,
               'criterion': param_range3}]

# GridSearchCV를 통해 최적의 파라미터 구하기
gs = GridSearchCV(estimator = dt_clf,
                  param_grid = param_grid, # 튜닝할 파라미터
                  scoring = 'accuracy',
                  cv=10,
                  n_jobs= -1)

gs = gs.fit(train_features_us, train_target_us)

print('GridSearchCV 최고 평균 정확도 수치 : {0:.4f}'.format(gs.best_score_))
print('GridSearchCV 최적 하이퍼 파라미터 : ', gs.best_params_)
```

```
GridSearchCV 최고 평균 정확도 수치 : 0.7751
GridSearchCV 최적 하이퍼 파라미터 :  {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 10}
```

```
# 최적의 모델 선택
best_tree = gs.best_estimator_ # 최적의 파라미터로 모델 생성
best_tree.fit(train_features_us, train_target_us)

DecisionTreeClassifier(max_depth=4, min_samples_leaf=10, random_state=2021)
```

최적 파라미터
- Criterion : 'gini'
- max_depth : 4
- min_samples_leaf : 10

04 실습

1. titanic

16) Test 데이터로 예측

```
# test 데이터로 예측하기
y_pred = best_tree.predict(test_features)
y_pred
```

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

17) 오차행렬 구하기

```
# 오차행렬 구하기
confmat = pd.DataFrame(confusion_matrix(test_target, y_pred),
                       index=['True[0]', 'True[1]'],
                       columns=['Predict[0]', 'Predict[1]'])
confmat
```

	Predict[0]	Predict[1]
True[0]	TN 87	FP 23
True[1]	FN 20	TP 49

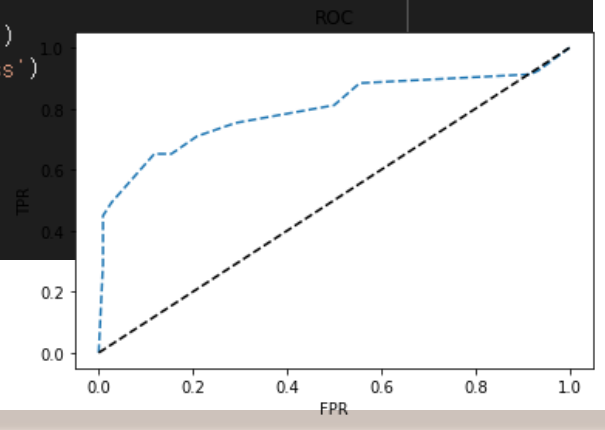
18) 분류 모델 평가 지표 구하기

```
# 분류 모델 평가 지표(정확도, 정밀도, 재현율, F1-score, AUC) 구하기
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

print('정확도 accuracy: %.3f' % accuracy_score(test_target, y_pred))
print('정밀도 precision: %.3f' % precision_score(y_true= test_target, y_pred=y_pred))
print('재현율 recall: %.3f' % recall_score(y_true=test_target, y_pred=y_pred))
print('F1-score: %.3f' % f1_score(y_true=test_target, y_pred=y_pred))
print('AUC: %.3f' % roc_auc_score(test_target, y_pred))
```

```
정확도 accuracy: 0.760
정밀도 precision: 0.681
재현율 recall: 0.710
F1-score: 0.695
AUC: 0.751
```

19) ROC 커브

```
# ROC 커브
fpr, tpr, thresholds = roc_curve( test_target, best_tree.predict_proba(test_features)[:, 1] )

plt.plot(fpr, tpr, '--', label = 'Decision Tree')
plt.plot([0,1], [0,1], 'k--', label='random guess')
plt.plot([fpr],[tpr],'r-', ms=10)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC')
plt.show()
```



04 실습

1. titanic

20) Decision Tree 시각화

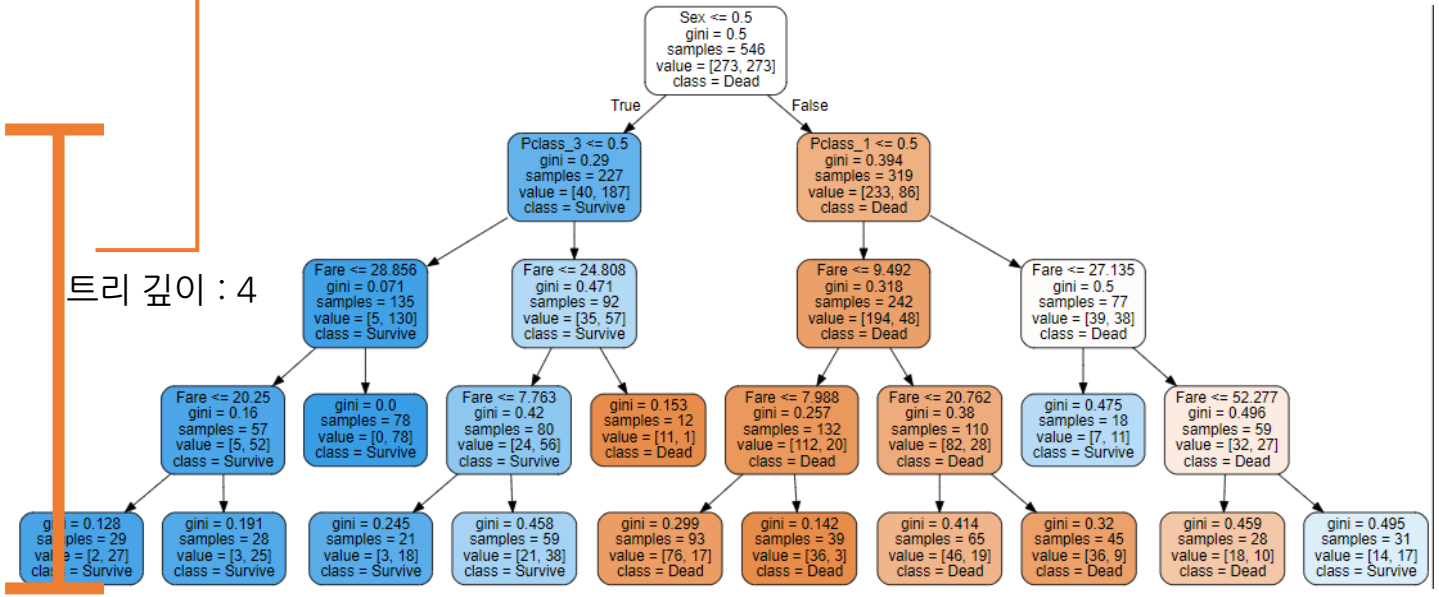
```
# 피처의 이름 (리스트)
feature_names = train_features_us.columns.tolist()
# 클래스(target) 이름
target_name = np.array(['Dead', 'Survive'])
```

```
dot_data_best = export_graphviz(best_tree,
                                filled = True,
                                rounded = True,
                                class_names = target_name,
                                feature_names = feature_names,
                               )

graphviz.Source(dot_data_best)
```

- max_depth : 4
- min_samples_leaf : 10

일때 결정트리

트리 깊이 : 4



하이퍼 파라미터를 튜닝하기 전 보다 훨씬 더 간결해졌다.

04 실습

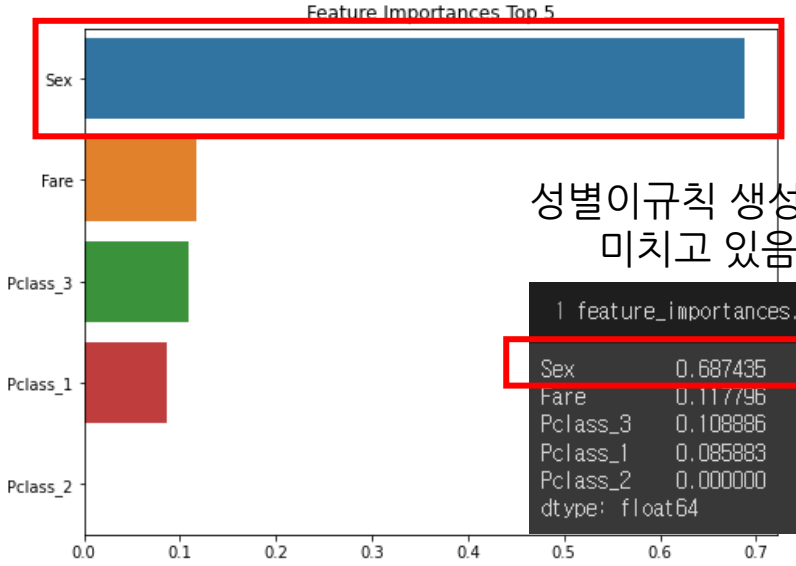
1. titanic

21) Feature Importance

- 결정 트리는 어떤 피처를 규칙 조건으로 선택하느냐가 중요한 요건.
- 이는 선택된 피처가 트리를 만드는데 크게 기여하며, 모델을 좀 더 간결하고 이상치에 강한 모델을 만들어 주기 때문.

→ 사이킷런에서 제공하는 feature_importance_ 속성을 통해 결정 트리에서 각 피처의 중요도를 시각화해보자.

```
# Feature Importance
# feature_importance_ 속성을 통해 결정 트리에서 각 피처의 중요도 구하기
feature_importance_values = best_tree.feature_importances_
feature_importances = pd.Series(feature_importance_values, index=train_features_us.columns)
# 중요도 값이 높은 순으로 5개를 골라 정렬
feature_top5 = feature_importances.sort_values(ascending=False)[:5]

plt.figure(figsize=[8, 6])
plt.title('Feature Importances Top 5')
sns.barplot(x=feature_top5, y=feature_top5.index)
plt.show()
```



성별이규칙 생성에 가장 큰 영향을 미치고 있음을 알 수 있다.

```
feature_importances.sort_values(ascending=False)
```

```
Sex         0.687435
Fare        0.117796
Pclass_3    0.108886
Pclass_1    0.085883
Pclass_2    0.000000
dtype: float64
```

Q & A

감사합니다