

# 4강: 의사결정트리의 이해

인공지능 일반강좌 : 기계학습의 이해(L2-1)

# Contents

의사결정나무 소개

결정트리 분류(예제)

결정트리 회귀(예제)

결정트리 분류(타이타닉 예제)

실습

숙제

# 결정트리 소개 (1)

## 결정 트리?\*\*

- 결정 트리는 분류 문제를 해결하기 위하여 가장 많이 사용된 지도(supervised) 기계학습 알고리즘의 한 종류
- 물론 회귀(Regression)에도 사용 가능함
- 결정 트리의 결과는 나무 같은 간단한 그래프으로 설명이 되어짐
- 그래서 블랙박스가 아닌 실제로 일어난 일들을 볼수 있음.

## 결정 트리 용어

루트 노드 - 부모 노드로 알려짐. 데이터셋의 길이와 모든 가지가 여기서 출발 함.

브랜치 - 가지는 루트 노드의 서브 노드로 나눈다. 물론 데이터도 나눔

결정 노드 - 결정노드들은 서브노드를 더 깊게 나눔. 더 이상 나눌 것이 없으면 리프노드임.

리프노드: 더이상 나눌 수 없는 노드.

알고리즘 - '지니'(Gini)는 경제학에서 불평등 지수를 나타냄. 0이 가장 평등하며, 1이 불평등하다.

- 즉, 데이터가 다양한 값을 가질 경우 평등(0)하며
- 특정 값으로 쓸릴경우 불평등(1에 가까움)
- 엔트로피는 무질서도를 나타내며, 무질서도(혼잡도)는 서로 다른 값이 섞여 있으면 높다. 혼잡도가 높으면 1, 적으면 0

$$\text{지니 지수: } G = 1 - \sum_i^c p_i^2 , \quad 0 \leq G \leq 1/2$$

$$\text{엔트로피 지수: } E = - \sum_i^c p_i \log_2 p_i , \quad 0 \leq E \leq 1$$

# 지니 불순도 또는 엔트로피

- 기본적으로 gini를 사용 Entropy (엔트로피) 불순도를 사용해도됨
  - ✓ 엔트로피는 문자의 무질서도를 측정하는 것 (열역학의 개념)
    - 문자가 안정되고 질서 정연하면 엔트로피는 0에 가까움
    - 정보이론에서 모든 메시기가 동일할때 엔트로피는 0에 가까움

# 머신러닝에서 불순도의 측정방법

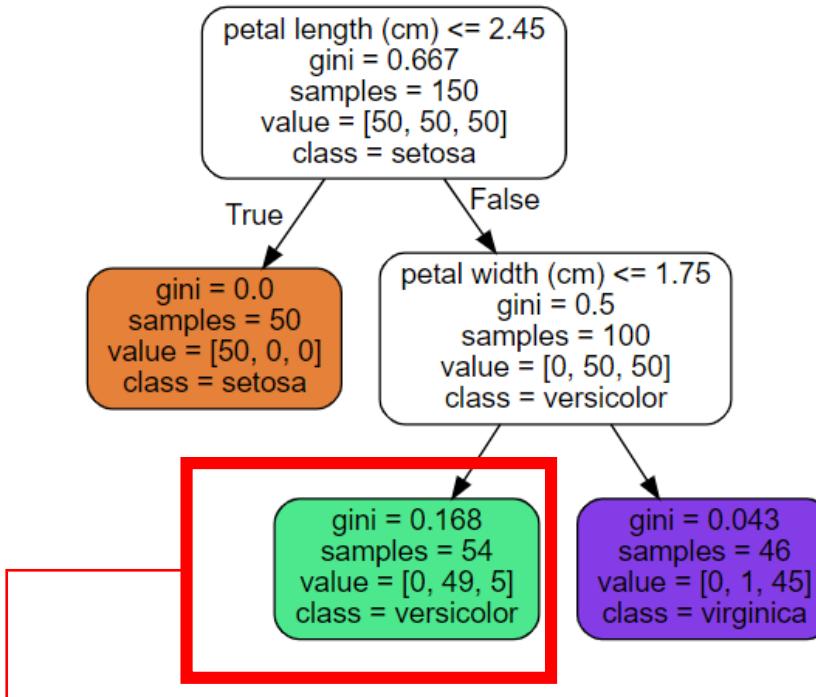
- 엔트로피의 정의

$$H_i = - \sum_{k=1}^n p_{i,k} \log_2(p_{i,k})$$

- 머신러닝에서

- ✓ 어떤 세트가 한 클래스의 샘플만 있는 경우는 엔트로피가 0이다.
- ✓ Gini와 엔트로피 중 어떤 것을 사용해도 실제로는 큰 차이가 없다.
- ✓ Gini 불순도가 계산이 조금 더 빠르기 때문에 기본값으로 좋다.

# 붓꽃의 결정트리의 엔트로피 계산



$$\rightarrow -\frac{49}{54} \log_2 \left( \frac{49}{54} \right) - \frac{5}{54} \log_2 \left( \frac{5}{54} \right) \approx 0.445.$$

# 결정 트리 학습과 시각화

- 붓꽃 세트에서 DecisionTreeClassifier를 훈련하는 코드

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

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```

# 결정 트리 학습과 시각화

- 시각화를 위하여, `export_graphviz()` 함수를 사용

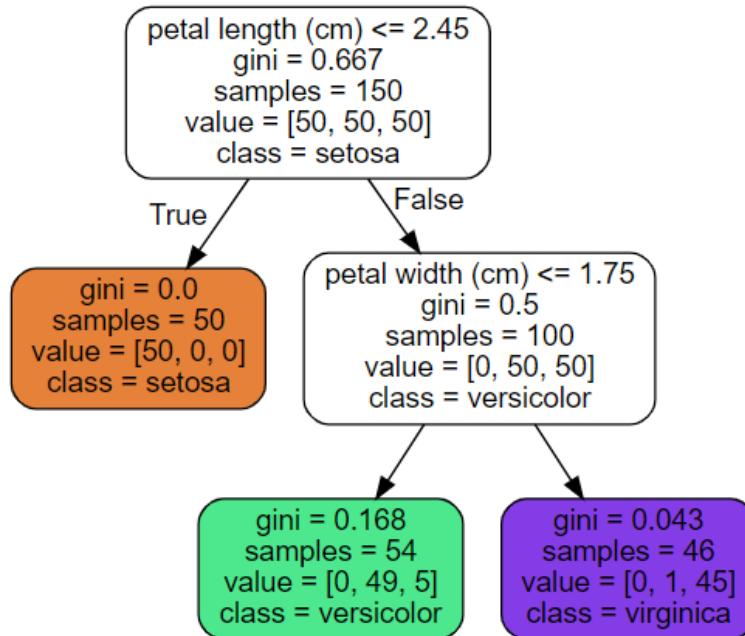
```
from sklearn.tree import export_graphviz

export_graphviz(
    tree_clf,
    out_file=image_path("iris_tree.dot"),
    feature_names=iris.feature_names[2:],
    class_names=iris.target_names,
    rounded=True,
    filled=True
)
```

# 결정 트리 학습과 시각화

- 시각화를 위하여, `export_graphviz()` 함수를 사용

```
Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```



# 예측하기

- 새로 발견한 꽃의 품종을 분류하려 있다고 가정해보자
  - ✓ 루트노드에서 시작 ~ 깊이가 0인 맨 꼭대기 노드
  - ✓ 이 노드에서 꽃의 길이가 2.45cm 보다 짧은지 검사
    - 왼쪽 : 리프노드(Leaf) ~ 자식 노드를 가지지 않는 노드 → setosa를 예측함.
  - ✓ 다른 꽃을 발견했고, 길이가 2.45보다 길경우.
    - 왼쪽: 다시 꽃잎의 너비가 1.75보다 작은지를 검사 → Versicolor (깊이2)
    - 그렇지않으면 Virginica (깊이2, 오른쪽)

# 결정트리 특성

- 장점
  - ✓ 데이터의 전처리가 거의 필요없다.
  - ✓ 특성 스케일을 맞추거나, 원점에 맞추는 작업은 필요가 없다.
- 노드의 속성은 얼마나 많은 훈련 샘플이 적용되었는지 헤아린 것이다.
  - ✓ 100개의 훈련 샘플의 꽃의 길이가 2.45cm보다 길고(깊이1,오른쪽)
    - 그 중에 54개 샘플이 1.75보다 짧고 (깊이2, 왼쪽)
- 노드의 value 속성은 노드에서 각 클래스에 있는 훈련 샘플 개수다.
  - ✓ 맨 오른쪽 아래 노드 Setosa=0, Versicolor=1, Virginica=45개
- 결정트리는 화이트 모델로 직관적이고 이해하기 쉽다.
- 램덤 포레스트나 신경망은 블랙박스 모델이다.

# 결정트리 특성

- 지니 gini 속성 불순도(impurity) 측정
- 한 노드에 모든 샘플이 같은 클래스에 속해 있으면 gini=0
  - ✓ 왼쪽 노드는 순수하다. gini=0.0
- 지니 불순도 측정

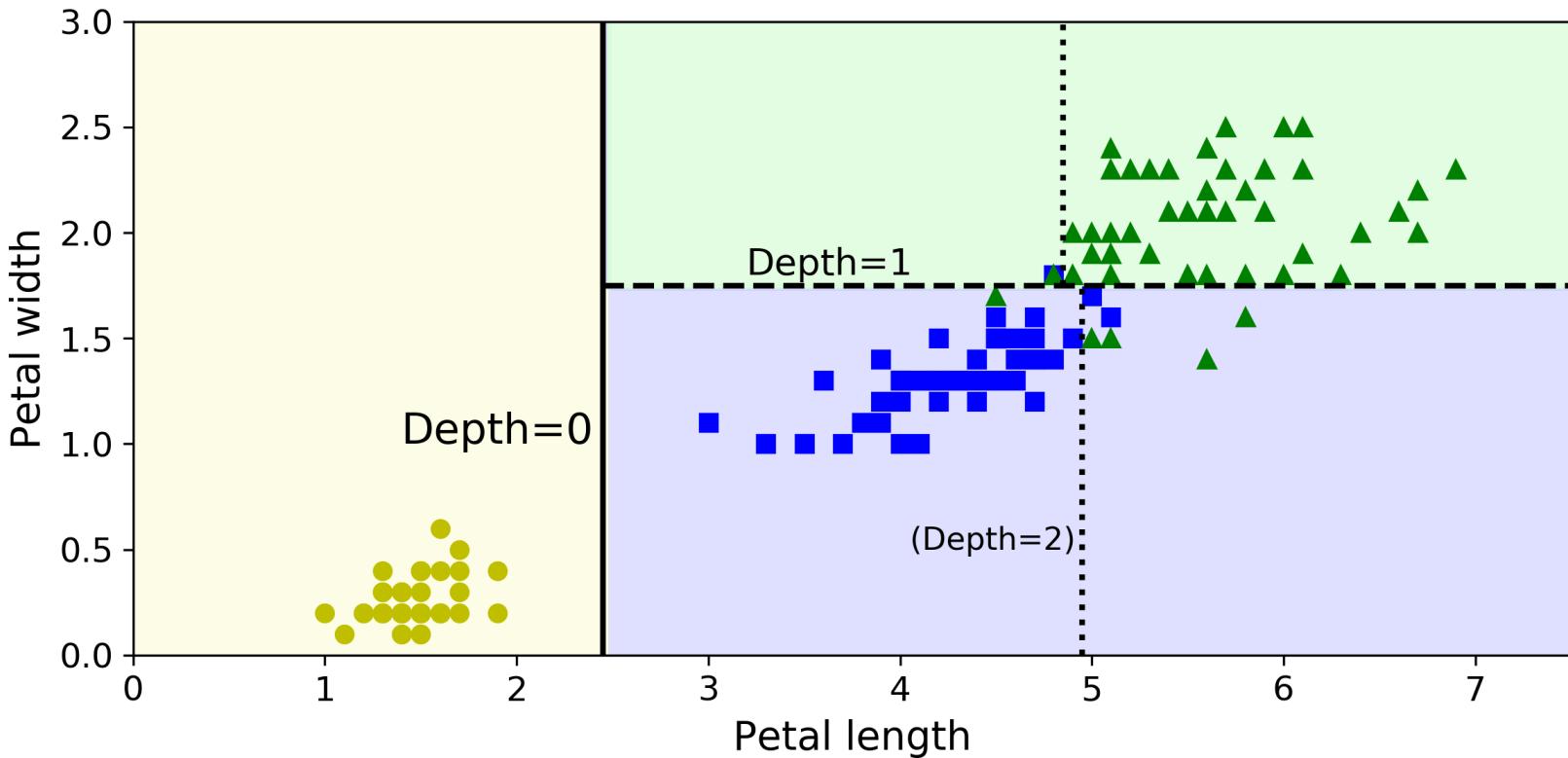
*Equation 6-1. Gini impurity*

$$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$$

- $p_{i,k}$  is the ratio of class  $k$  instances among the training instances in the  $i^{\text{th}}$  node.
- ✓ 깊이 2, 왼쪽 versicolor의 경우

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168.$$

# 결정트리 경계



굵은 수직선 ~ 루트 노드의 결정경계

오른쪽은 깊이 1로 나누어지면 ( $\text{max\_depth}=2$ 로 설정으로 더 나누지 않음)  
만일  $\text{max\_depth}=3$ 으로 설정하면, 수직(점선)으로 각각 결정 경계를 추가함.

# 사이킷런은 CART 알고리즘 사용

- CART
  - ✓ Classification and Regression Tree 알고리즘
  - ✓ 먼저 훈련세트를 하나의 특성  $k$ 의 임곗값  $t_k$ 를 사용해 두개의 서브셋으로 나눔
    - 크기에 따른 가중치가 적용된 가장 순수한 서브셋으로 나눌 수 있는  $k, t_k$  짝을 찾음
    - 이 알고리즘이 최소화해야 하는 비용함수

*Equation 6-2. CART cost function for classification*

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where  $\begin{cases} G_{\text{left/right}} & \text{measures the impurity of the left/right subset,} \\ m_{\text{left/right}} & \text{is the number of instances in the left/right subset.} \end{cases}$

- CART 알고리즘은 탐욕적(Greedy) 알고리즘으로 현재 단계의 분할이 몇 단계를 거쳐 가장 낮은 불순도로 이어질 수 있는지를 고려하지 않음  
→ 종종 납들할 만한 솔루션으로 최적의 트리는 아님.

# 계산의 복잡도

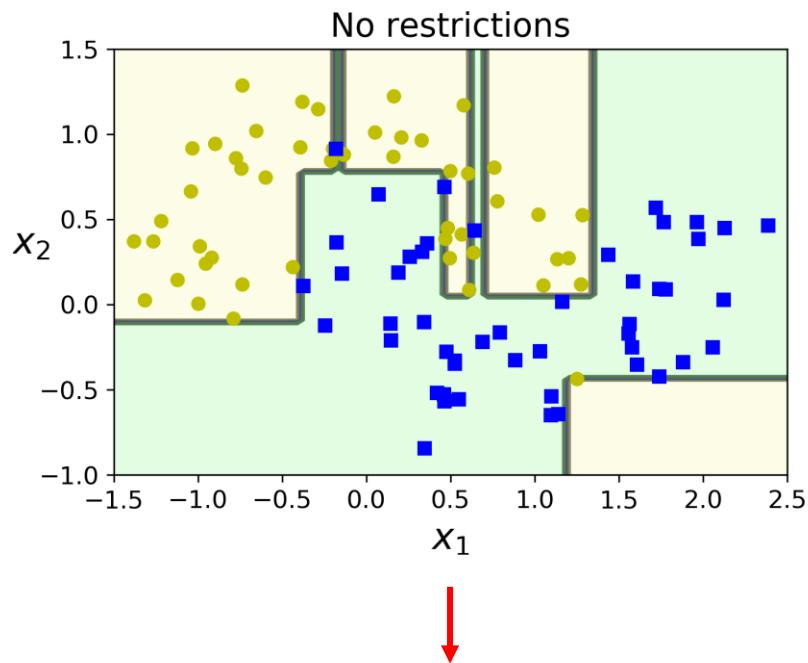
- 예측을 하려면 결정 트리를 노드에서부터 리프 노드까지 탐색함.
  - ✓ 각 노드에 1개의 특성값만 확인하기 때문에  $O(\log_2(M))$ 의 계산 복잡도는 특성 수와 무관하게  $O(\log_2(M))$ .
  - ✓ 그래서 큰 훈련세트를 다룰 때 예측속도가 빠르다.

# 결정트리 규제 매개변수

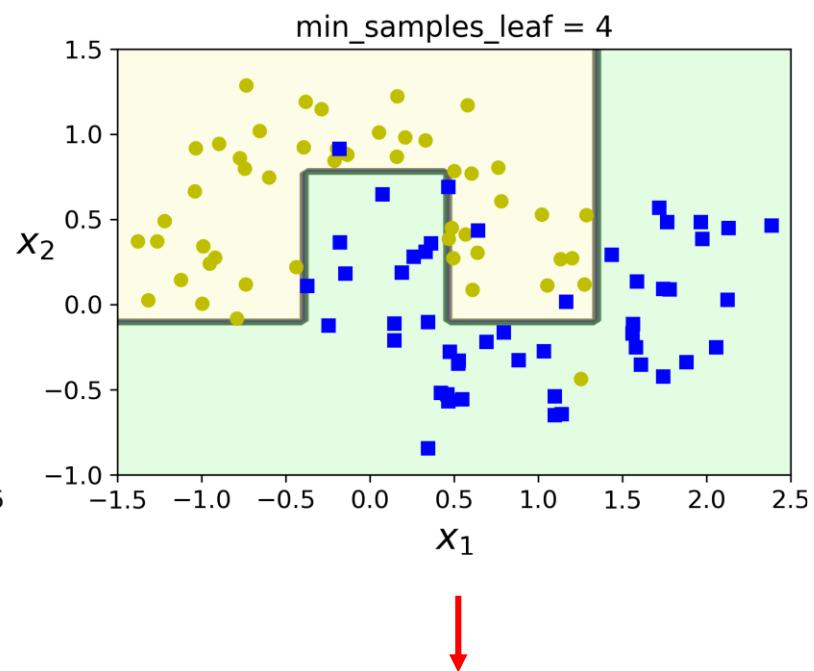
- 결정 트리는 훈련 데이터에 대한 제약 사항이 거의 없다.
  - ✓ (선형 모델은 데이터가 선형일 거라는 가정을 한다.)
  - ✓ 제한을 두지 않으면 트리가 훈련 데이터에 아주 깊게 맞추려고해서 과적합 발생하기 쉽다.
  - ✓ 결정트리는 비파라미터 모델
    - 훈련도기 전에 파라미터 수가 결정되지 않는다.
    - 모델 구조가 데이터에 맞추져서 고정되지 않고 자유롭다.
  - ✓ 참고로 파라미터 모델은 미리 정의된 파라미터 수를 가지므로 자유도가 제한되고 과대적합될 위험이 줄어든다.
- 규제
  - ✓ max\_depth 매개변수로 조절
  - ✓ min\_samples\_split

# 결정트리의 규제 예제

- 매개변수 `min_samples_leaf`를 사용한 규제 예제



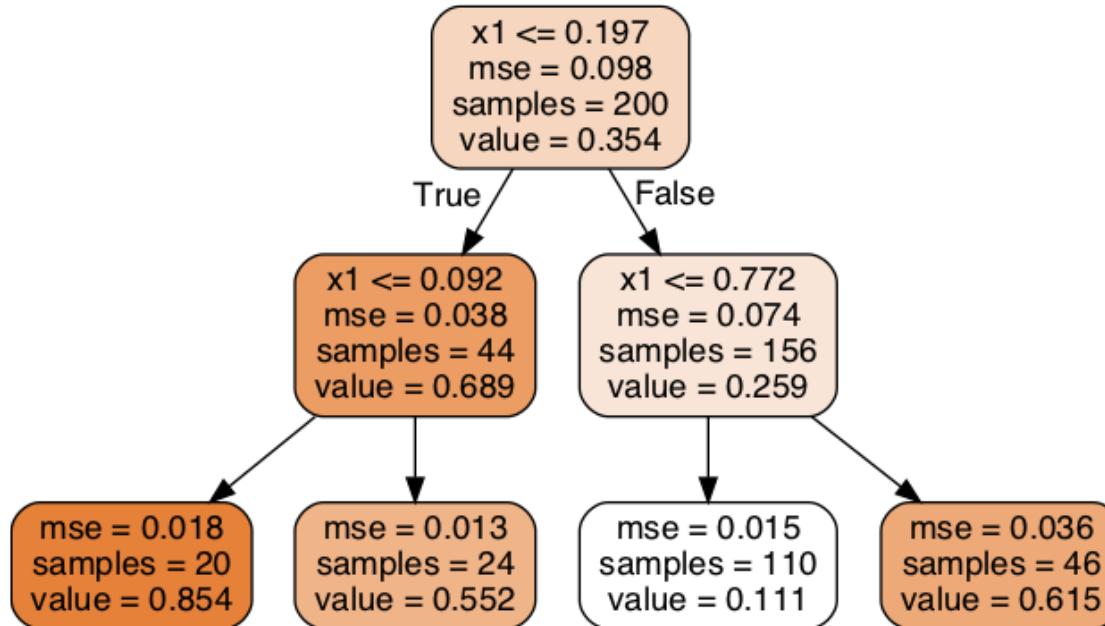
과적합



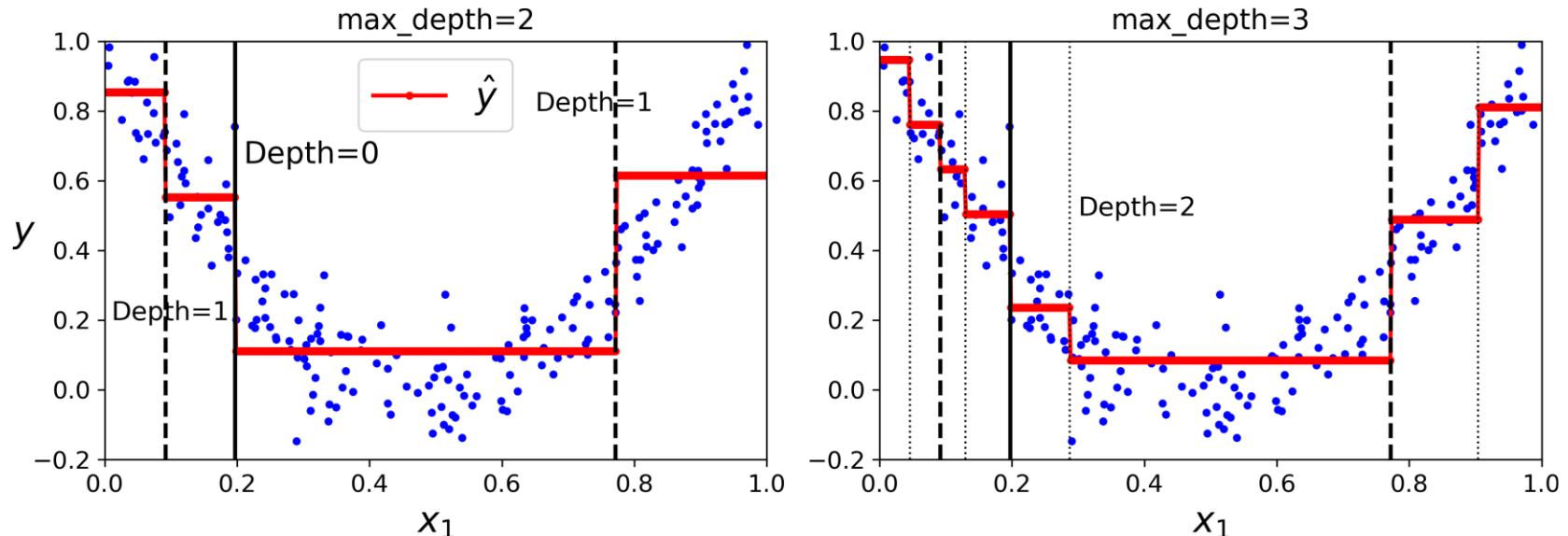
일반화에 적합

# 회귀

- DecisionTreeRegressor를 사용하여 회귀
  - 회귀 트리 max\_depth=2일 경우
  - 훈련 데이터  $y=4(x-0.2)*(x-0.2)$ ,  $x1=0.6$ 일 때 예측해보자.



# 2개의 결정트리 회귀 모델 예측



# CART 회귀 알고리즘

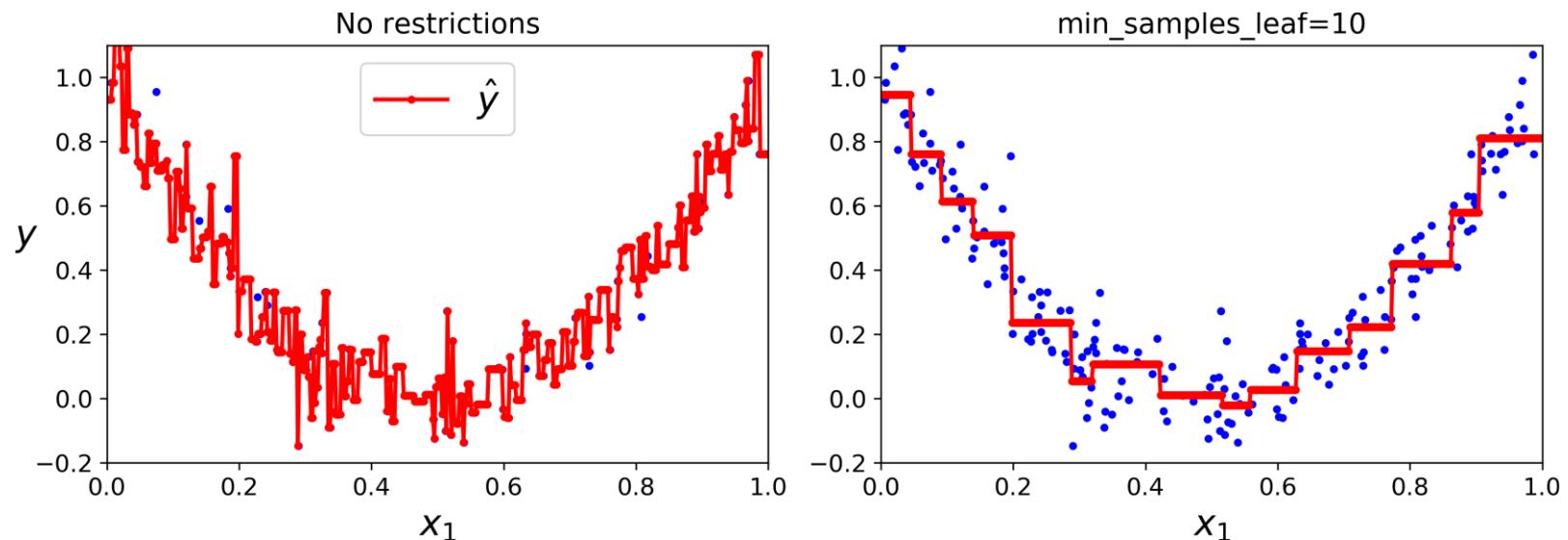
- 훈련 세트 불순도를 최소화하는 방향으로 분류하는 대신에 균제곱오차 (MSE)를 최소화하도록 분할하는 것을 제외하고 분류와 같다.
- CART 알고리즘이 최소화하기 위한 비용함수

*Equation 6-4. CART cost function for regression*

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}} \quad \text{where} \quad \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

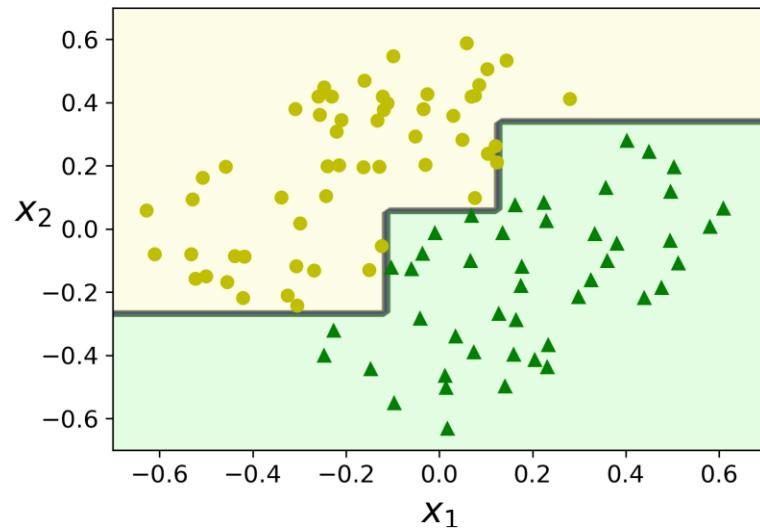
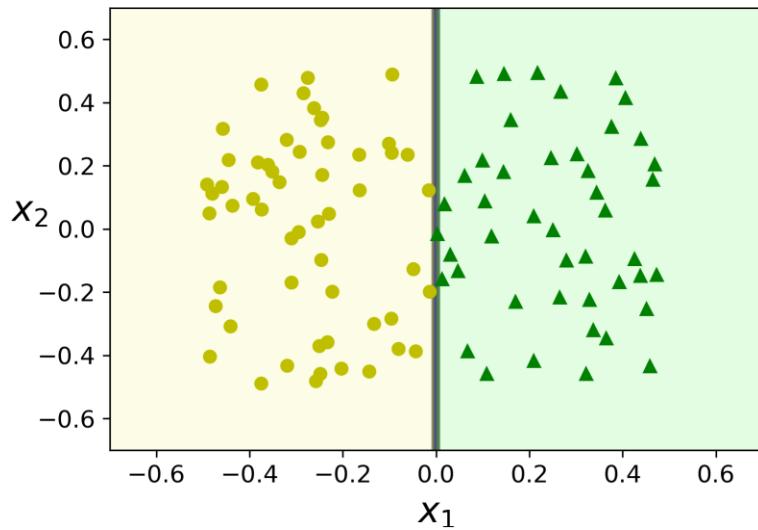
# 결정트리 회귀도 과적합에 주의

- 규제  $\text{min\_samples\_leaf}=10$ 으로 할 경우



# 결정트리 불안정성(1)

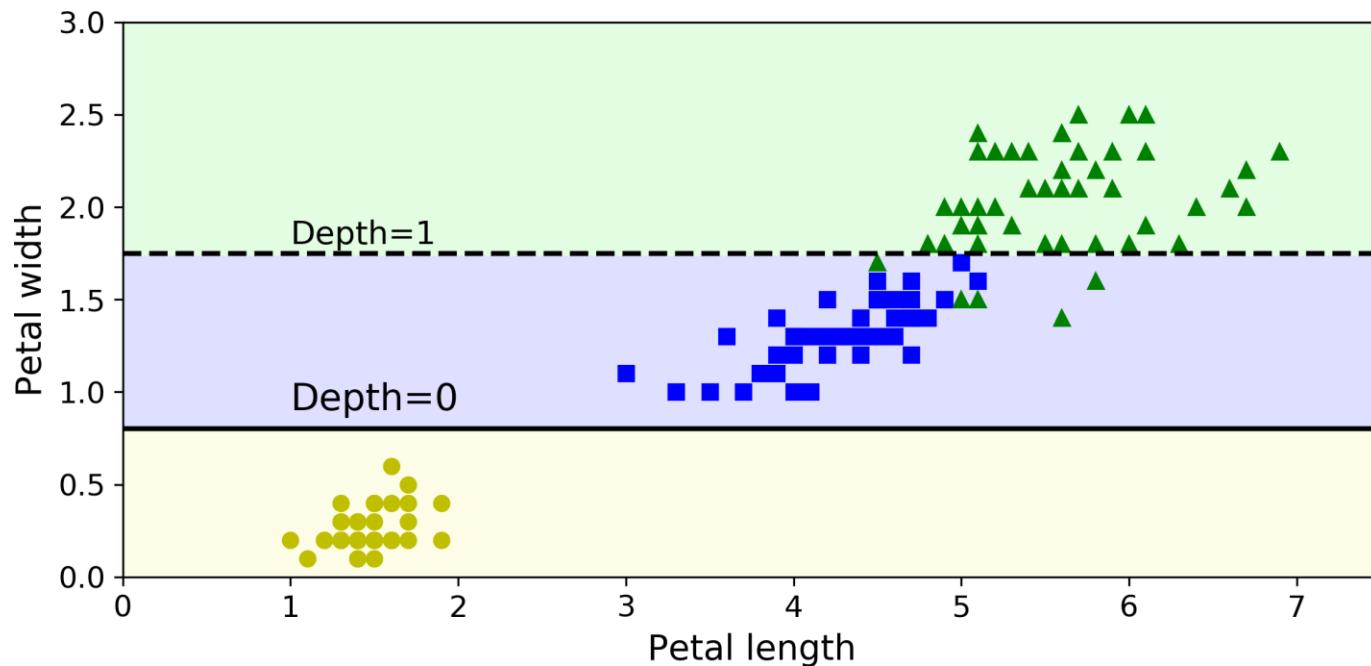
- 결정트리의 결정경계는 수직 축으로 분할
  - ✓ 그래서 훈련세트의 회전에 민감하다.



- ✓ 이를 해결하기 위해서 PCA 기법을 사용해보자

# 결정트리 불안정성(2)

- 결정트리의 주된 문제는 훈련데이터에 있는 작은 변화에도 민감하다.
  - ✓ (예제로) Versicolor 꽃 데이터를 제거하고 훈련하면, 이걸에 만든 결정트리와는 매우 다른 모습이다.
  - 사이킷런에서 사용하는 훈련알고리즘은 확률적이기 때문에 같은 훈련데이터에서도 다른 모델을 얻게 될 수 있다.

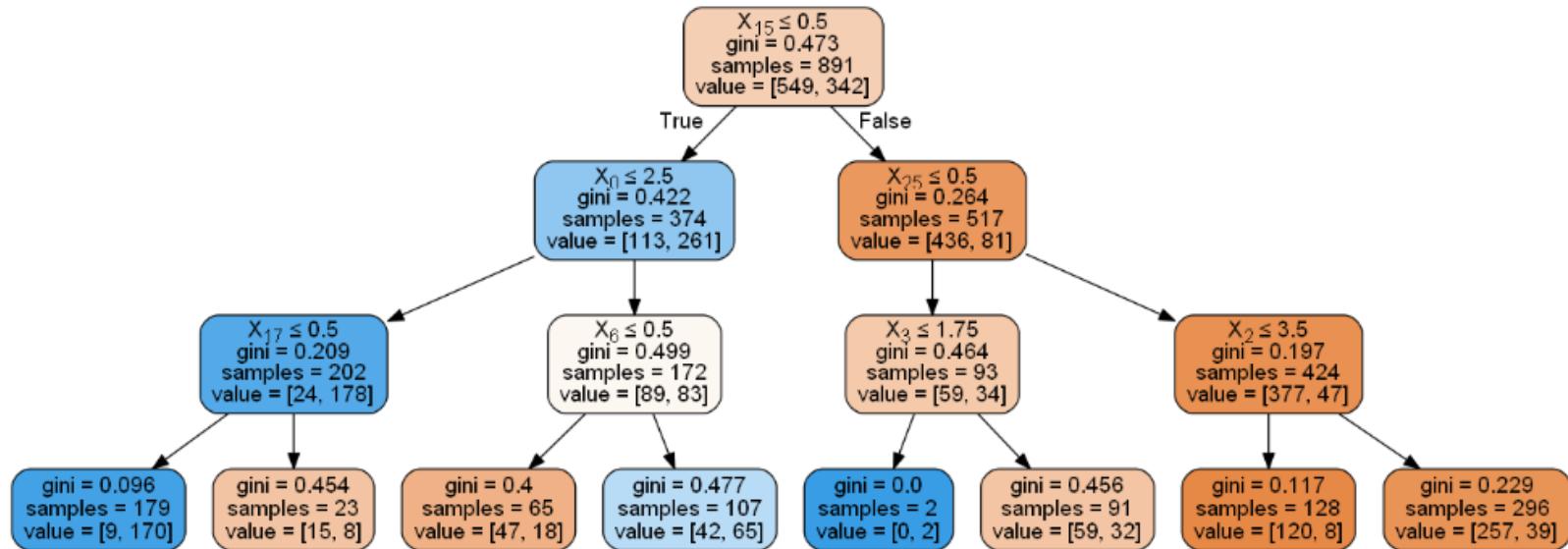




# 결정트리 소개 (2)

루트노드, 리프노드, 가지노드, gini를 이해해보자

(예) 결정트리 : 타이타닉 생존자 예측



## 결정트리 분류에 적용 (1)

학생 60명에서 방과후에 축구하는 사람을 예측하기

#### 사용 데이터 설명 및 문제 설명

- 데이터는 60명이 학생이 있고,
  - 성별 (M/F), 반 (IX/X), 키 (5/6 피트) 즉 150과 180cm, 몸무게 (50/58)
  - 이중 30명은 방과후에 축구를 한다.
  - 풀어하 할 문제는 방과후에 누가 축구를 하는지 예측하라!

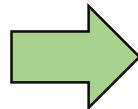
# 결정트리 분류에 적용 (2)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 5 columns):
Weight    60 non-null int64
Height    60 non-null int64
Class     60 non-null object
Sex       60 non-null object
label     60 non-null object
dtypes: int64(2), object(3)
memory usage: 2.4+ KB
```

```
df.head()
```

	Weight	Height	Class	Sex	label
0	50	5	IX	M	P
1	58	6	IX	M	NP
2	50	5	IX	M	P
3	58	6	IX	M	NP
4	50	5	IX	M	NP



- 카테고리컬한 데이터를 숫자로 바꾸어라.
- 축구하면 (P=1), 안하면(NP)를 0
- new\_label 변수로 1이면 방과후에 축구를 한 것

```
code = {'P': 1, 'NP': 0}
df['new_label'] = df['label'].map(code)
```

```
new_s = pd.get_dummies(df.Sex)
new_c = pd.get_dummies(df.Class)
df[new_s.columns] = new_s
df[new_c.columns] = new_c
```

판다스 df.get\_dummies는 카타로그 데이터를 0과 1의 값es 갖는 별도의 데이터를 만듬

```
df.head()
```

	Weight	Height	Class	Sex	label	new_label	F	M	IX	X
0	50	5	IX	M	P	1	0	1	1	0
1	58	6	IX	M	NP	0	0	1	1	0
2	50	5	IX	M	P	1	0	1	1	0
3	58	6	IX	M	NP	0	0	1	1	0
4	50	5	IX	M	NP	0	0	1	1	0

# 결정트리 분류에 적용 (3)

사이킷런의 의사결정분류 라이브러리를 이용하여 적용해보자 ¶

```
from sklearn import tree

model = tree.DecisionTreeClassifier(random_state = 23)

feature = new_df.drop(['new_label'], axis = 1)

label = new_df.new_label

model.fit(feature, label)

DecisionTreeClassifier(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_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=23, splitter='best')
```

# 결정트리 분류에 적용 (4)

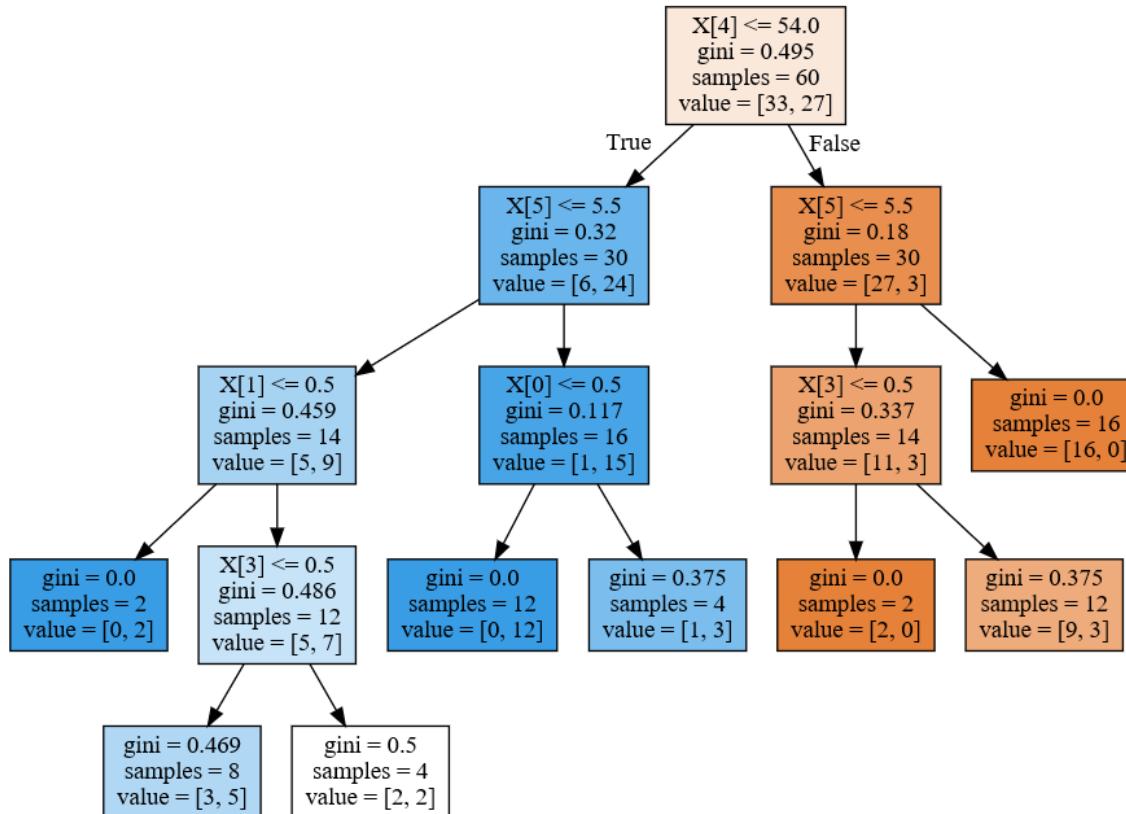
About   Download   Gallery   Documentation   Theory and Publications   License  
Resources   Credits   FAQ   Contact   Twitter   Issues/Bugs



## Graphviz - Graph Visualization Software

```
import graphviz
graph = tree.export_graphviz(model, out_file=None, filled=True)
graphviz.Source(graph)
```

# 결정트리 분류에 적용 (5)



```
model.fit(feature, label)
```

```
DecisionTreeClassifier(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_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort='deprecated',
random_state=23, splitter='best')
```

# 결정트리 분류에 적용 (6)

중요 변수의 이해

**criterion**: 지니 혹은 엔트로피 알고리즘 선택 'Gini' 혹은 'Entropy'.

**max\_depth**: 위의 예제의 경우 4이다. 이 숫자가 크면 오버피팅 될 수 있다.

**max\_features**: 최적의 분할을 고려할때 초대 피처 개수, 디폴트는 None으로 데이터 셋트를 사용하여 분할 수행

**max\_leaf\_nodes**: 말단 리프 노드의 최대 갯수

**min\_samples\_leaf**: 말단 노드인 리프노드가 되기 위한 최소한의 샘플 데이터 수

**min\_samples\_split**: 노드를 분할 하기 위한 최소한의 샘플 데이터의 수. 과적합을 제어에 도움이 됨

- 디폴트는 20이고, 작게 설정할 수록 분할 노드가 많아져서 과적함 증가

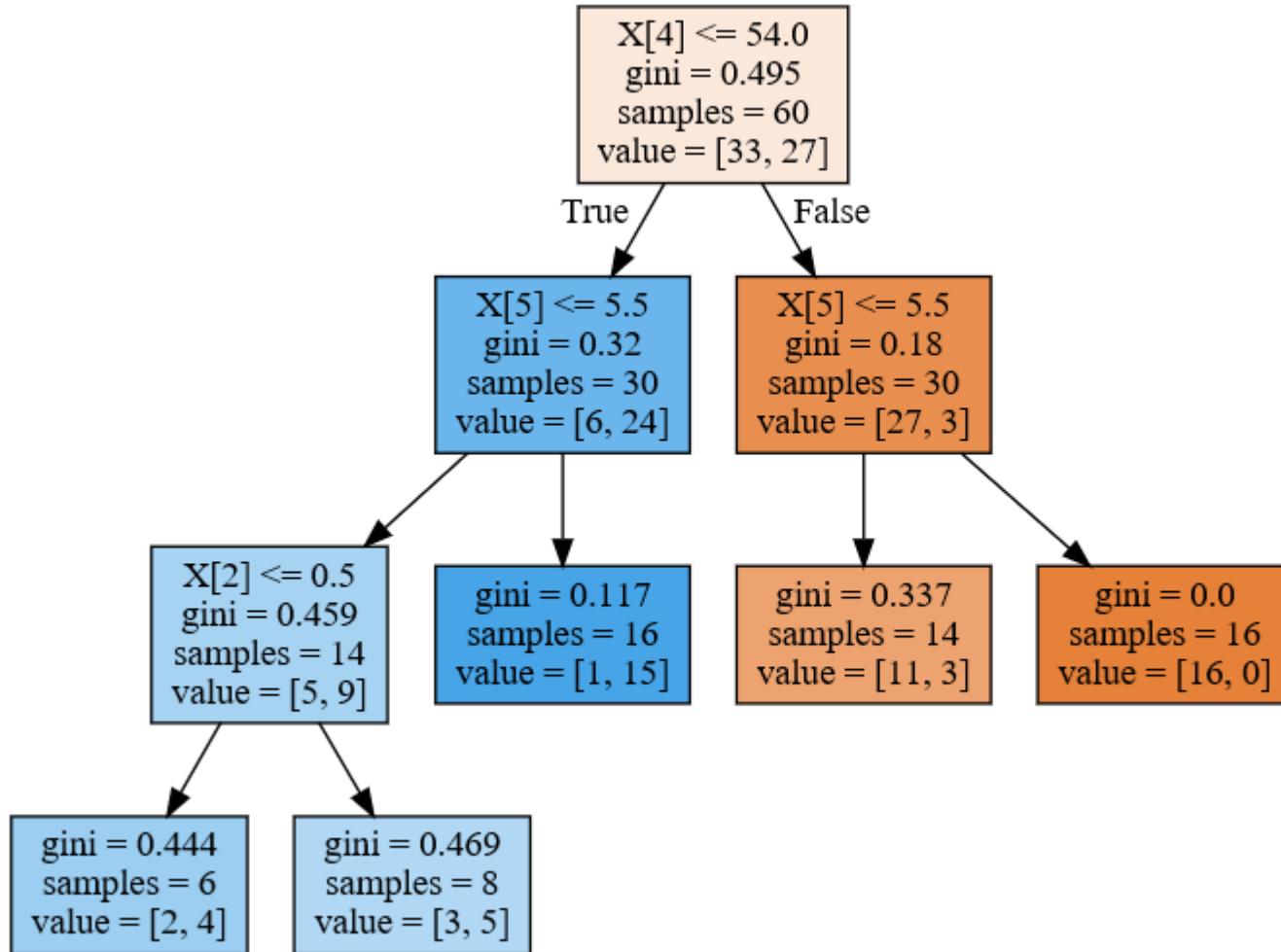
# 결정트리 분류에 적용 (7)

연습문제 min\_samples\_leaf 개수를 5로 했을때 차이점을 논의하시오

```
from sklearn import tree
model1 = tree.DecisionTreeClassifier(min_samples_leaf = 5, random_state = 23)
model1.fit(feature, label)
```

```
DecisionTreeClassifier(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_impurity_split=None,
                      min_samples_leaf=5, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=23, splitter='best')
```

# 결정트리 분류에 적용 (8)



# 결정트리 분류에 적용 (9)

## min\_samples\_split

```
from sklearn import tree
model2 = tree.DecisionTreeClassifier(min_samples_split = 16, random_state = 23)
model2.fit(feature, label)

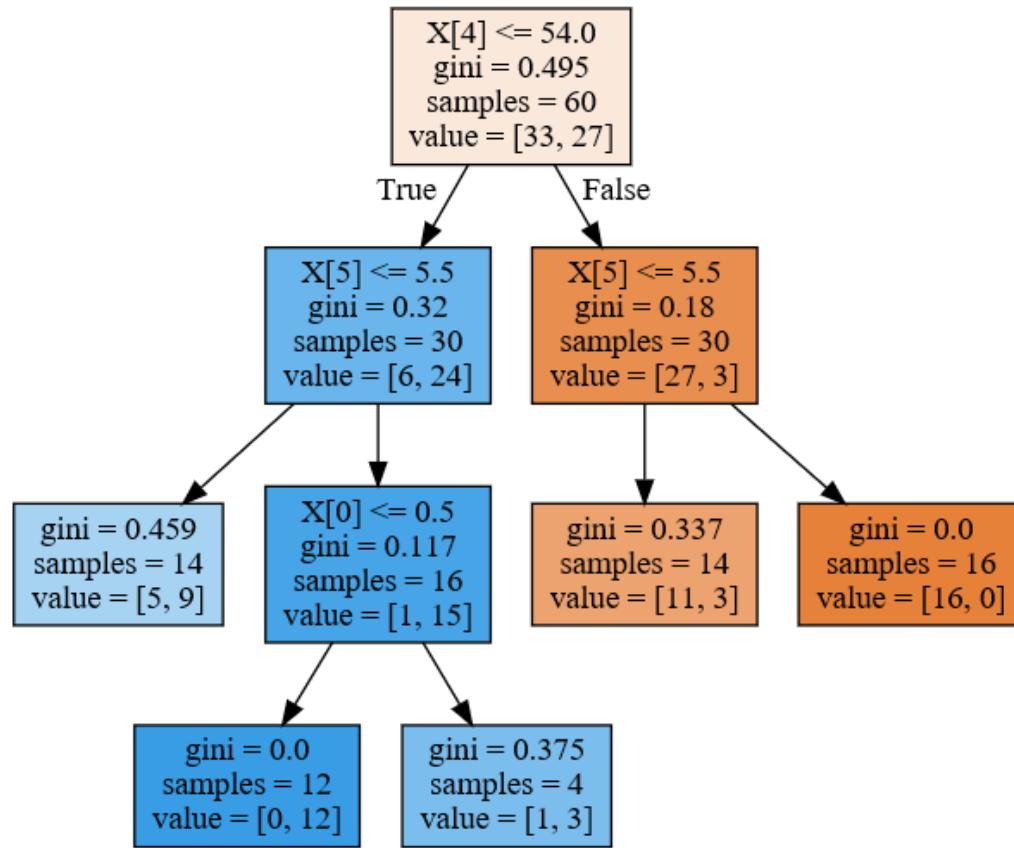
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                      max_depth=None, max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=16,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=23, splitter='best')
```

(실습 해보기)

min\_samples\_split를 4~20 사이를 변화시키면서,

그래프의 변화를 이해하라

# 결정트리 분류에 적용 (10)



트리의 변화를 확인하여라.

# 결정트리 분류에 적용 (11)

## Grid Search CV

```
from sklearn.model_selection import GridSearchCV

param1 = {'min_samples_leaf': [2,3,4,5],
          'min_samples_split': [2,3,5,10,12,14],
          'max_depth': [2,3,4,5,6],
          'criterion': ['gini', 'entropy'],
          'max_features':[2,3,4]}

CV = GridSearchCV(model, param1)

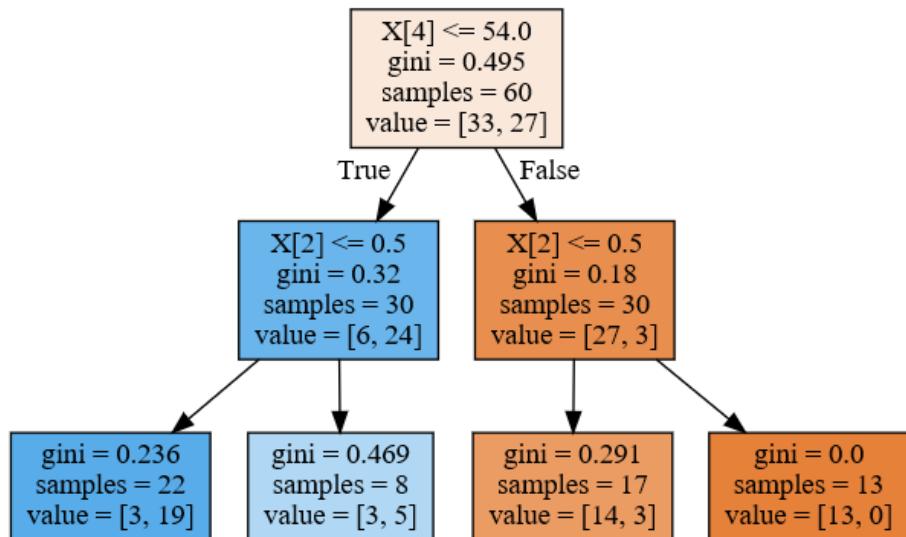
CV.fit(feature, label)

GridSearchCV(cv=None, error_score=nan,
            estimator=DecisionTreeClassifier(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_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=23,
                                              splitter='best'),
            iid='deprecated', n_jobs=None,
            param_grid={'criterion': ['gini', 'entropy'],
                        'max_depth': [2, 3, 4, 5, 6],
                        'max_features': [2, 3, 4],
                        'min_samples_leaf': [2, 3, 4, 5],
                        'min_samples_split': [2, 3, 5, 10, 12, 14]},
            pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
            scoring=None, verbose=0)
```

중요 :

- 1) Grid Search는 무엇인가?
- 2) 파라미터 공간을 조정해보자
- 3) 관심이 더 있으면 Random Seach
- 4) Bayesian Optimization Search
- 5) NAS(Neural architecture search)
- 6) Hyperparameter search(HPO)쪽으로 주제는 현재 AI에서 핫 이슈임.

# 결정트리 분류에 적용 (12)



중요:

- 1) Grid Search에서 얻은 각각 파라미터 최적의 값은 무엇인가?
- 2) 결정 트리 정확도 85%를 얻었다.
- 3) 정확도 85%의 의미와 데이터 양과의 관계를 생각해보아라
- 4) 교차검증은 83.3%를 얻었다. 의미는?

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

accuracy_score(best.predict(feature), label)

0.85

cross_val_score(best, feature, label, cv=5)

array([0.83333333, 0.91666667, 0.91666667, 0.66666667, 0.91666667])
```

소결) 정확도 85%는 꽤 좋은 성적이고 교차 검증 성적도 좋다

# 결정트리 회귀에 적용 (01)

## 1차원 선형 곡선을 결정트리로 회귀(Regression) 예측하기

```
import numpy as np
import pandas as pd

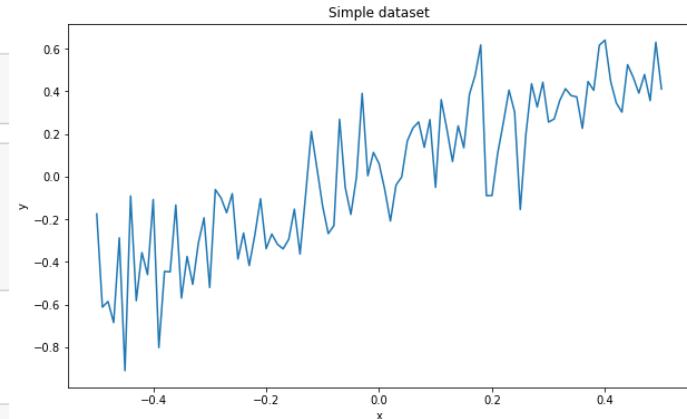
from sklearn import tree
import graphviz

import matplotlib.pyplot as plt
import seaborn as sns
```

### 예제로 1차 함수 형태의 데이터셋에서 회귀 트리 만들어 보기

```
# 1차 함수 형태는 y = x + random
nPoints = 100
xPlot = [(float(i)/float(nPoints) - 0.5) for i in range(nPoints + 1)]
x = [[s] for s in xPlot]
np.random.seed(1)
y = [s + np.random.normal(scale=0.2) for s in xPlot]
```

```
# plt.rcParams.update({'font.size': 14})
plt.rcParams['figure.figsize'] = (10.0, 6.0)
plt.plot(xPlot,y)
#plt.axis('tight')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simple dataset')
plt.show()
```



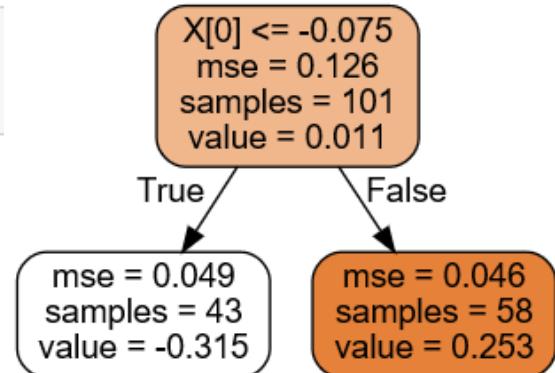
# 결정트리 회귀에 적용 (2)

max\_depth=1

```
model = tree.DecisionTreeRegressor(max_depth=1)

model.fit(x, y)

DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=1,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort='deprecated',
                     random_state=None, splitter='best')
```



- 아래 블럭 다이아그램에서 루트 노드는 -0.075을로 기준 (split 점)
- 이 split 값을 기준으로 2개의 그룹으로 분류
- 아래 2개의 박스 중에서 왼쪽으로는 43개의 샘플, 오른쪽에는 58개의 샘플로 분류
- 만일 테스트 값이  $x=0.20$ 이라면, 예측 값은  $y=0.253$  임

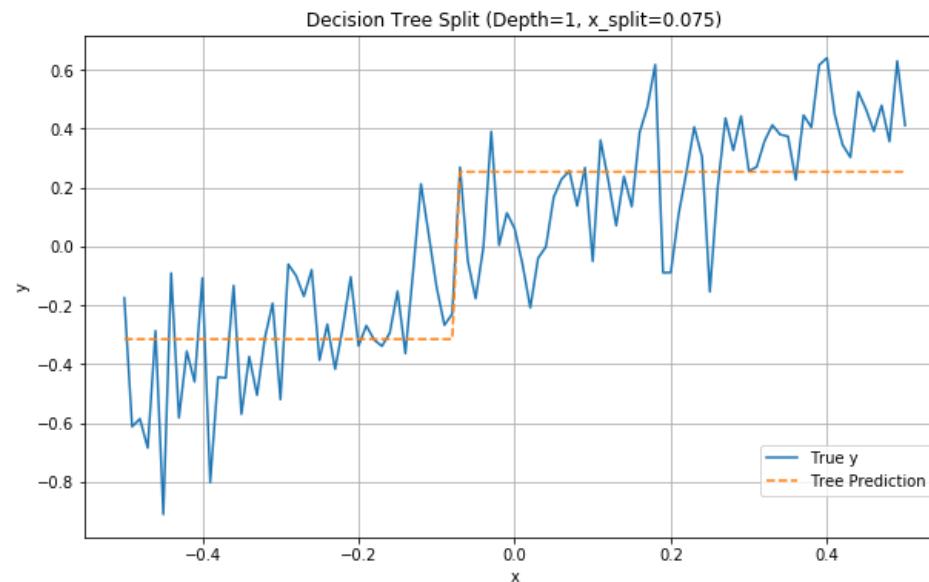
```
#draw the decision tree result with graphviz
graph = tree.export_graphviz(model, out_file = None, rounded = True, filled = True)
graphviz.Source(graph)
```

# 결정트리 회귀에 적용 (3)

```
#compare prediction from tree with true values
yHat = model.predict(x)

plt.figure()
plt.plot(xPlot, y, label='True y')
plt.plot(xPlot, yHat, label='Tree Prediction', linestyle='--');

plt.legend(bbox_to_anchor=(1,0.2))
plt.axis('tight')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.title('Decision Tree Split (Depth=1, x_split=0.075)')
plt.show()
```



# 결정트리 회귀에 적용 (4)

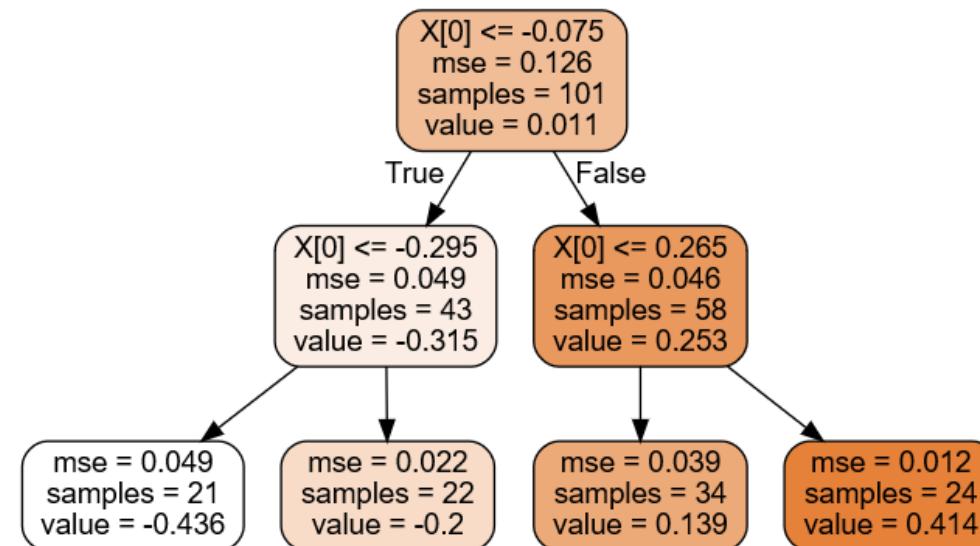
연습문제 : 의사결정 나무 depth=2로 증가 시켜라

```
# 결정나무 다시 설정 max_depth=2
model2 = tree.DecisionTreeRegressor(max_depth=2)

model2.fit(x, y)

DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort='deprecated',
                     random_state=None, splitter='best')

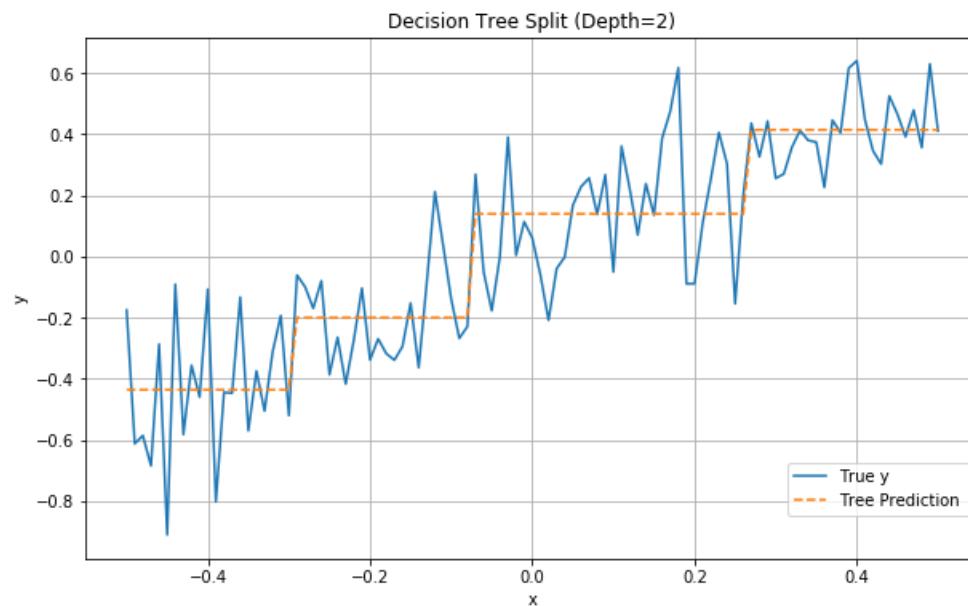
graph = tree.export_graphviz(model2, out_file = None, rounded = True, filled = True)
graphviz.Source(graph)
```



# 결정트리 회귀에 적용 (5)

```
#compare prediction from tree with true values
yHat = model2.predict(x)

plt.figure()
plt.plot(xPlot, y, label='True y')
plt.plot(xPlot, yHat, label='Tree Prediction ', linestyle='--')
plt.legend(bbox_to_anchor=(1,0.2))
plt.axis('tight')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.title('Decision Tree Split (Depth=2)')
plt.show()
```



split 포인트는 2개가 더 생김

- $x=-0.295$
- $x=0.264$

# 결정트리 회귀에 적용 (6)

## Split Points 찾기는 어떻게하나?

- 트리는 예측값의 제곱 오차를 최소화(MSE)한다.
- 생각해보면, 임의의 split 값이 주어지면 2개의 그룹 중에 1개로 선택된다. 각각의 그룹의 평균은 MSE를 최소화하는 값이 된다.
- 아래 예제를 보자

```
: sse = []
xMin = []

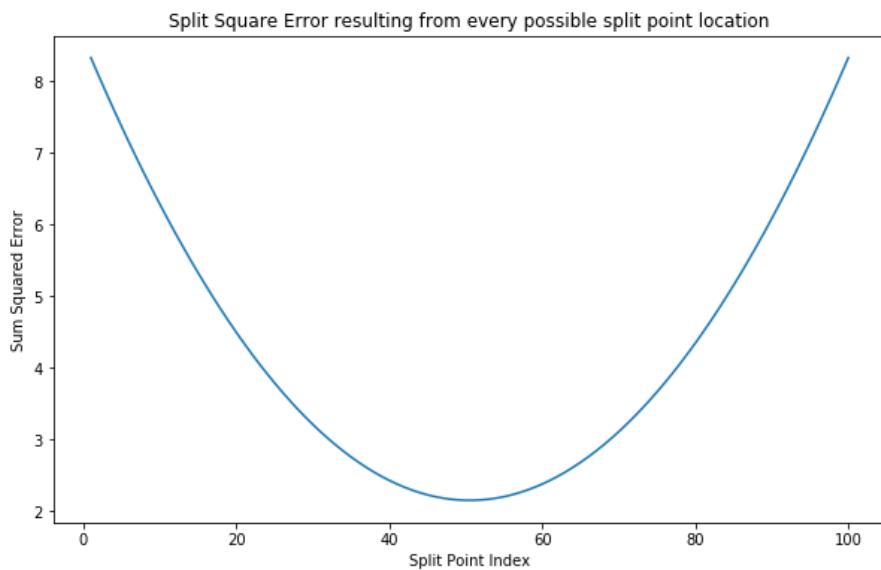
for i in range(1, len(xPlot)):          # 왼쪽과 오른쪽으로 나눔
    lhList = list(xPlot[0:i])
    rhList = list(xPlot[i:len(xPlot)]) # 각각의 사이드에서 평균을 구하기
    lhAvg = sum(lhList) / len(lhList)
    rhAvg = sum(rhList) / len(rhList)
    # 제곱 오차 합을 구하기
    lhSse = sum([(s - lhAvg) * (s - lhAvg) for s in lhList])
    rhSse = sum([(s - rhAvg) * (s - rhAvg) for s in rhList])
    # 합치고 최소값을 찾기
    sse.append(lhSse + rhSse)
    xMin.append(max(lhList))
```

# 결정트리 회귀에 적용 (7)

```
#SSE is sum of squared error
plt.plot(range(1, len(xPlot)), sse)

plt.xlabel('Split Point Index')
plt.ylabel('Sum Squared Error')

plt.title('Split Square Error resulting from every possible split point location')
plt.show()
```



SSE 함수에서 최소값을 찾고, 최소값의 위치를 찾은 다음, 그 값에 해당하는 최소값

결과는 잘 알려진 포물선 모양

- 최소값은



```
minSse = min(sse)
idxMin = sse.index(minSse)
print(xMin[idxMin])
```

-0.010000000000000009

# 결정트리 회귀에 적용 (8)

## 멀티 변수 트리 학습은 어떻게?

- 알고리듬은 MSE의 최소값에 기여하는 모든 가능한 split point를 찾는다.

## 결정트리의 과적합 (Overfitting) 문제

- 데이터는 적은데 너무 많은 split point를 고려해보면
- 학습은 잘 되는데, 예측할때 틀릴 수가 많다.
- depth를 높여서 실습해보자

중요:

- 과적합(Overfitting)이란 무엇인가
- 과적합은 언제 일어나는가?
- 과적합을 피하기 위한 방법은 무엇인가?

# 결정트리 회귀에 적용 (9)

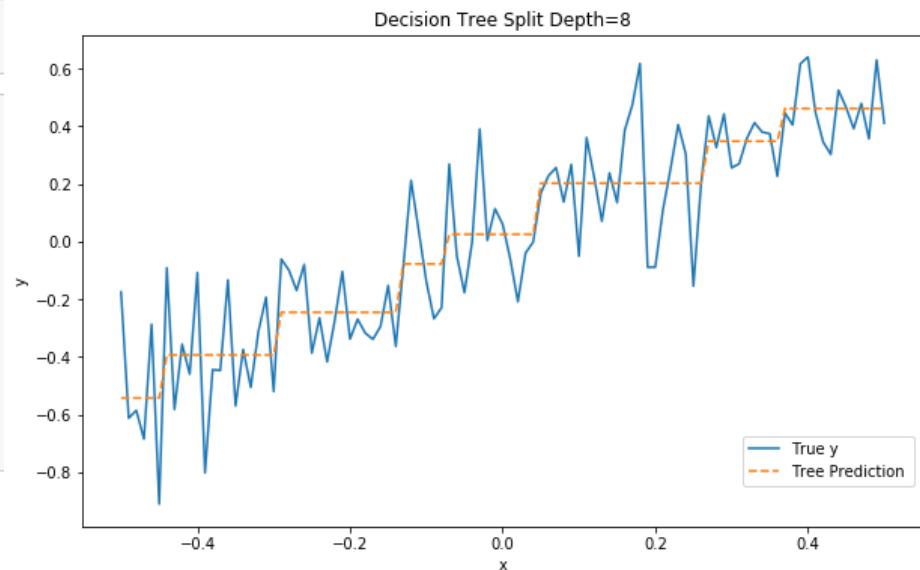
## 연습문제2: max\_depth를 3로 높여서 예측해보기

```
: model3 = tree.DecisionTreeRegressor(max_depth=3)
model3.fit(x, y);

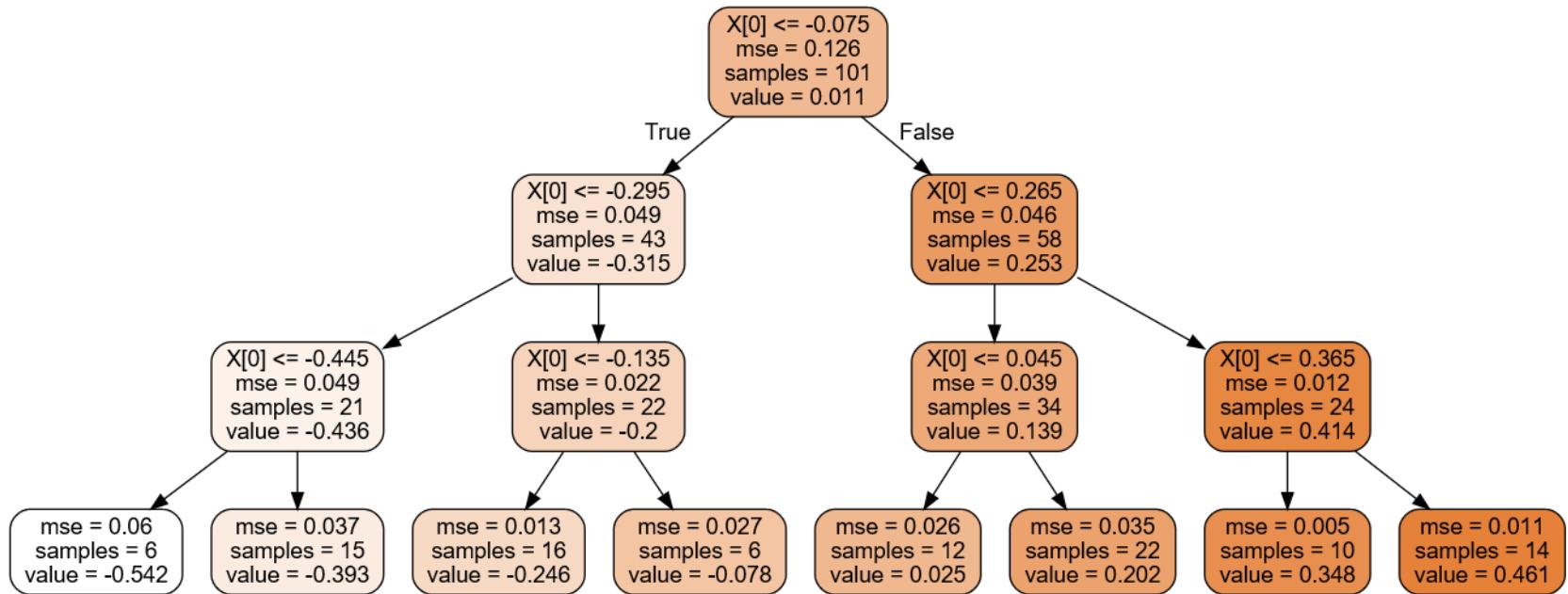
: #compare prediction from tree with true values
yHat = model3.predict(x)

plt.figure()
plt.plot(xPlot, y, label='True y')
plt.plot(xPlot, yHat, label='Tree Prediction ', linestyle='--')
plt.legend(bbox_to_anchor=(1,0.2))

plt.axis('tight')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Tree Split Depth=8')
plt.show()
```



# 결정트리 회귀에 적용 (10)



# 결정트리 회귀에 적용 (11)

- 데이터 숫자를 늘려서 확인해보기
  - 데이터 개수를 nPoints = 200로 2개 증가 시킴
  - max\_depth=1
  - max\_depth=2
  - max\_depth=4 일때 결정트리의 예측정확도를 구해라?

# 타이타닉 침몰과 생존자 데이터

- 타이타닉 승객의 정보를 사용하여 생존할 확률을 예측하는 문제
- Kaggle Competition에서 다뤘던 주제로,
- TFlearn의 기본 신경망 튜토리얼로 활용 중 <http://tflearn.org/tutorials/quickstart.html>

- ✓ 1912년 4월 15일, 타이타닉호 침몰.
- ✓ 탑승자 전제 2224명 중 1502명 사망.
- ✓ 특정 그룹의 사람들은 생존 확률이 높았음.



# 타이나닉 생존자 예측해보자

## 1. 라이브러리 와 데이터 로딩

```
import pandas as pd
import numpy as np # linear algebra

import re # 이들을 이용하여 피처 엔지니어링 할때 사용할

from sklearn import tree
import graphviz

# 가시화
import seaborn as sns # visualization
import matplotlib.pyplot as plt # visualization
```

pandas로 타이타닉 데이터 읽어오기

데이터의 70%는 훈련에 활용하고

나머지 30%는 시험에 사용한다.

```
df_train = pd.read_csv('./input/train.csv',sep=',') # importing train dataset
df_test = pd.read_csv('./input/test.csv',sep=',') # importing test dataset
```

데이터 정보 확인해보자

판다스의 df.head()

df.describe()

# 타이타닉 원본 train.csv 파일 읽기

```
df_train.head(9) # seeing the first 6 rows from train dataset
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	113803	53.1000	C123	S
3	4	1	1	Allen, Mr. William Henry	male	35.0	1	0	373450	8.0500	NaN	S
4	5	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
5	6	0	3	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
6	7	0	1	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
7	8	0	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
8	9	1	3									

# 타이타닉 데이터 형식 및 의미

컬럼명	의미
pclass	1, 2, 3등석 정보를 각각 1, 2, 3으로 저장
survived	생존 여부. survived(생존), dead(사망)
name	이름
sex	성별. female(여성), male(남성)
age	나이
sibsp	함께 탑승한 형제 또는 배우자의 수
parch	함께 탑승한 부모 또는 자녀의 수
ticket	티켓 번호
fare	티켓 요금
cabin	선실 번호
embarked	탑승한 곳. C(Cherbourg), Q(Queenstown), S(Southampton)

```
titanic.describe() # Seeing a summary of numeric columns
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000
mean	655.000000	2.294882	29.881138	0.498854	0.385027	33.295479
std	378.020061	0.837836	14.413493	1.041658	0.865560	51.758668
min	1.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	328.000000	2.000000	21.000000	0.000000	0.000000	7.895800
50%	655.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	982.000000	3.000000	39.000000	1.000000	0.000000	31.275000
max	1309.000000	3.000000	80.000000	8.000000	9.000000	512.329200

# Train.csv 파일 확인

데이터의 특성 'Survived'는 labeling(라벨)이다.

- 데이터는 numerical
- categorical
- text features
- 결손데이터 missing values ('NaN')

데이터의 형태(shape)을 보자

```
: # Seeing shape of each dataset
print('train shape:',df_train.shape)

train shape: (891, 12)

: print('test shape:',df_test.shape)

test shape: (418, 11)
```

라벨, 즉 'Survived' 컬럼은 train data로 부터 별도로 다룬다.(다중에 쓸것)

- 결손 데이터도 처리해주자

```
: survived = df_train['Survived'] # saving 'Survived' column from train dataset to be used latter

: df_train2 = df_train.drop('Survived',axis=1) # dropping 'Survived' column from train dataset to join datasets
```

## 주의

\*\* df\_train2 = df\_train.drop()로 오버라이트 하였음. 데이터 개수가 다름

# 학습데이터 전처리 (1)

```
titanic.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S
2	3	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
3	4	1	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
4	5	3									

전처리 1. 티켓을 없애자. 왜?

```
titanic = titanic.drop('Ticket',axis=1) # dropping Ticket column from titanic dataset
```

```
titanic.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	71.2833 7.9250	C85 NaN	C S
2	3	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	C123	S
3	4	1	Allen, Mr. William Henry	male	35.0	0	0	8.0500	NaN	S
4	5	3								

# 학습데이터 전처리 (2)

## 결손 데이터 파악하기 및 전체 갯수는

```
titanic.isnull().sum()
```

```
PassengerId      0  
Pclass           0  
Name             0  
Sex              0  
Age            263  
SibSp            0  
Parch            0  
Fare             1  
Cabin          1014  
Embarked         2  
dtype: int64
```

```
len(titanic)
```

```
1309
```

```
(titanic.isnull().sum() / len(titanic)).sort_values(ascending = True)
```

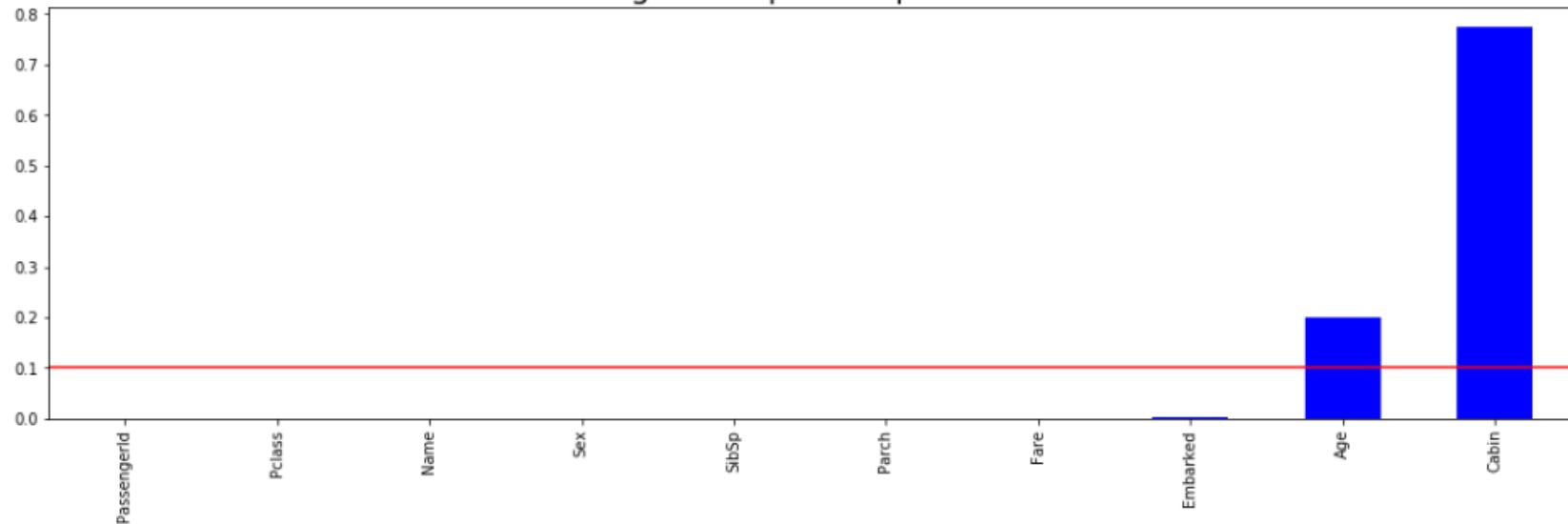
```
PassengerId    0.000000  
Pclass         0.000000  
Name           0.000000  
Sex            0.000000  
SibSp          0.000000  
Parch          0.000000  
Fare           0.000764  
Embarked       0.001528  
Age            0.200917  
Cabin          0.774637  
dtype: float64
```

# 학습데이터 전처리 (3)

```
plt.subplots(0,0, figsize = (18,5)) # defining figure size  
  
ax = (titanic.isnull().sum()/len(titanic)).sort_values(ascending = True).plot.bar(color = 'blue')  
  
plt.axhline(y=0.1, color='r', linestyle='-')  
  
plt.title('Missing values percent per columns', fontsize = 20) # plotting a title
```

Text(0.5, 1.0, 'Missing values percent per columns')

Missing values percent per columns



# 학습데이터 전처리 (4)

## titanic의 특성

- 2 개 숫자 데이터 **age** 와 **Fare**
- 2 개의 카타고리컬 데이터 **Cabin**, **Embarked**

```
# replacing na's of Age using linear regression method
titanic['Age'] = titanic['Age'].interpolate(method="linear", limit_direction="forward")
```

```
# replacing na's of Age using linear regression method
titanic['Fare'] = titanic['Fare'].interpolate(method="linear", limit_direction="forward")
```

## NaN을 'U'로 unKnown Port로

Now let's replace na's from Embarked variables by 'U' to indicate an unknown Port of Embarkation

```
# replacing na's of Embarked from U
titanic['Embarked'] = titanic['Embarked'].fillna('U')
titanic['Embarked'] = titanic['Embarked'].astype('category') # Converting into categorys
```

```
titanic.isna().sum() # verifying na's
```

```
PassengerId      0
Pclass            0
Name              0
Sex               0
Age               0
SibSp             0
Parch             0
Fare              0
Cabin          1014
Embarked         0
dtype: int64
```

# 학습데이터 상관관계 (1)

## 데이터 상관관계

```
num_vars = titanic[['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']][1:891] # taking numerical variables to correlation plot
```

```
num_vars.head(3)
```

	Pclass	Age	SibSp	Parch	Fare	
1	1	38.0	1	0	71.2833	
2	3	26.0	0	0	7.9250	
3	1	35.0	1	0	53.1000	

## 타겟과 합치기

```
cor_df = pd.concat([num_vars,survived],axis=1) # join numerical variables and our target
```

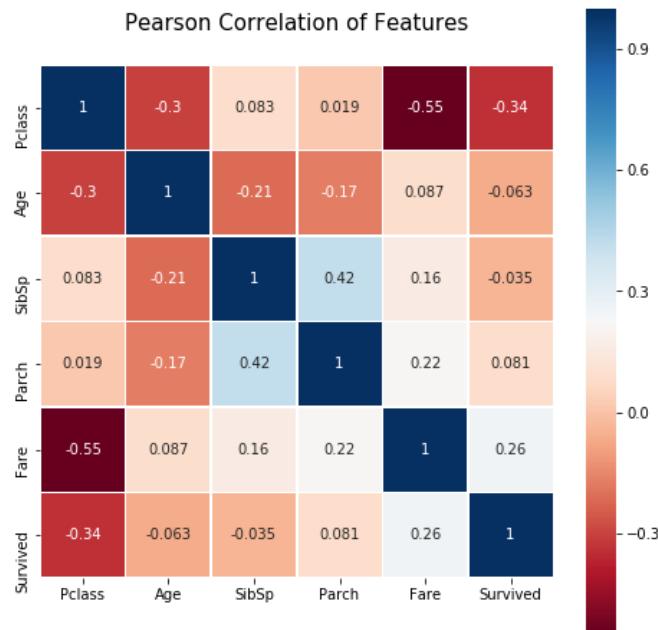
```
cor_df.head(5)
```

	Pclass	Age	SibSp	Parch	Fare	Survived
0	NaN	NaN	NaN	NaN	NaN	0
1	1.0	38.0	1.0	0.0	71.2833	1
2	3.0	26.0	0.0	0.0	7.9250	1
3	1.0	35.0	1.0	0.0	53.1000	1
4	3.0	35.0	0.0	0.0	8.0500	0

# 학습데이터 상관관계 (2)

seaborn으로 상관관계를 시각화 하자

```
colormap = plt.cm.RdBu  
  
plt.figure(figsize=(8,8)) # defining plot size  
  
plt.title('Pearson Correlation of Features', y=1.05, size=15) # plotting a title to our plot  
  
sns.heatmap(cor_df.astype(float).corr(), linewidths=0.5,vmax=1,  
            square=True, cmap=colormap, linecolor='white', annot=True);
```

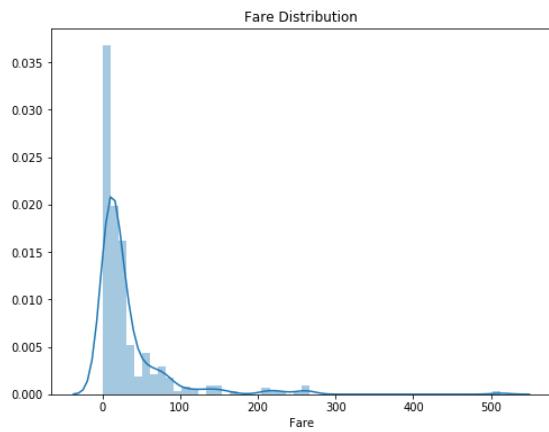


변수들 사이에 높은 상관관계는 없다.

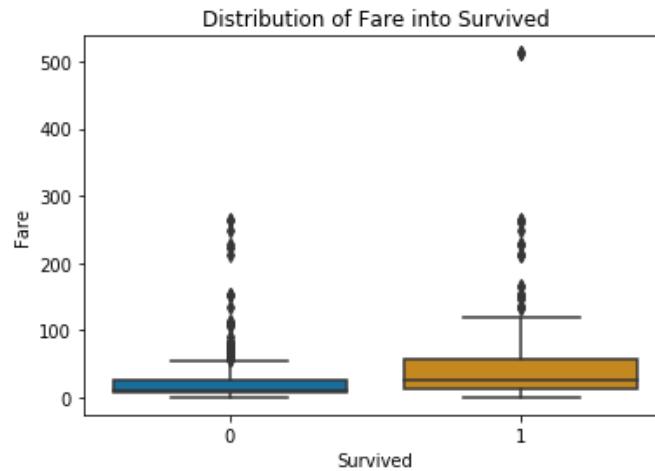
let's take a look into Age, Fare, Parch and Pclass with boxplots

# 학습데이터 상관관계 (3)

타켓 라벨 생존과 관계는



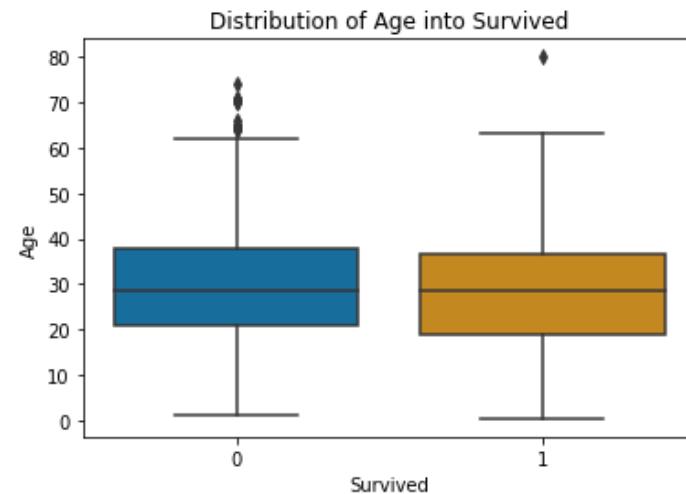
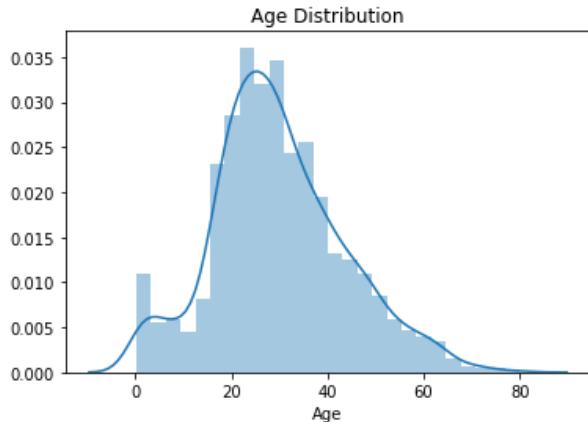
```
: plt.figure(figsize=(6,4)) # defining plot size  
sns.boxplot(y='Fare',x='Survived',data=cor_df, # boxplotting  
palette="colorblind")  
plt.title('Distribution of Fare into Survived') # defining a title
```



많은 outliers 가 있으며, 가격 100 이하에 집중된 승객임.

- 나이도 고려해보자

# 학습데이터 상관관계 (4)



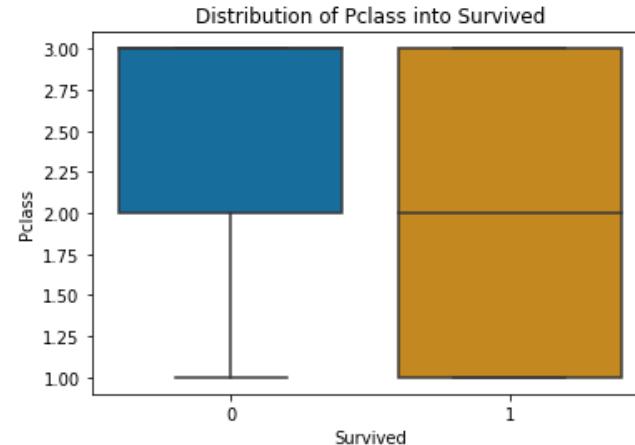
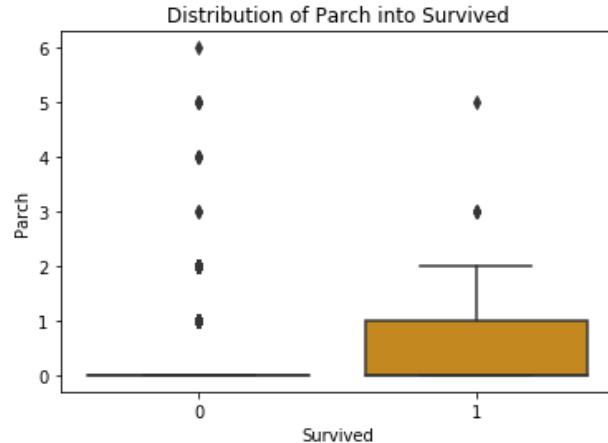
## 분석

- 나이에 대해서는 정규분포를 보여주고 있지만, 완벽하지는 않다.

## 분석

- 대다수 승객은 20에서 40대임을 알 수 있다.
- 60살 이상에서 아웃레이어가 보이며,

# 학습데이터 상관관계 (5)



## 함께 탑승한 부모 또는 자녀 수 (Parch)

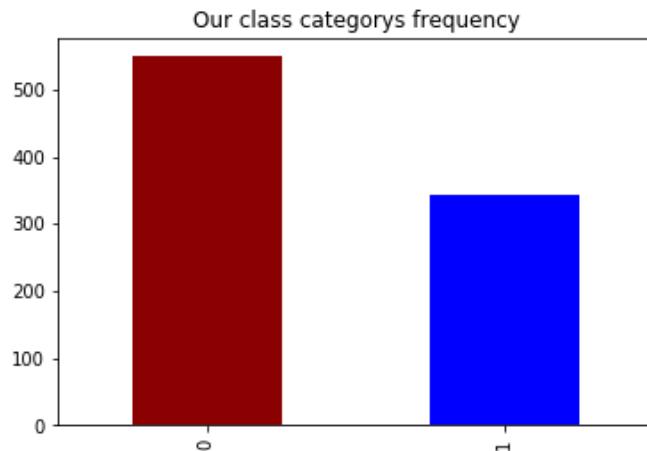
- 파취가 1 이하에서 생존 확률이 높다.

## 분석

- 모든 클래스에서 생존하였다.
- 하지만 2등석, 3등석 객실에서 사망을 하였다.

# 학습데이터 상관관계 (6)

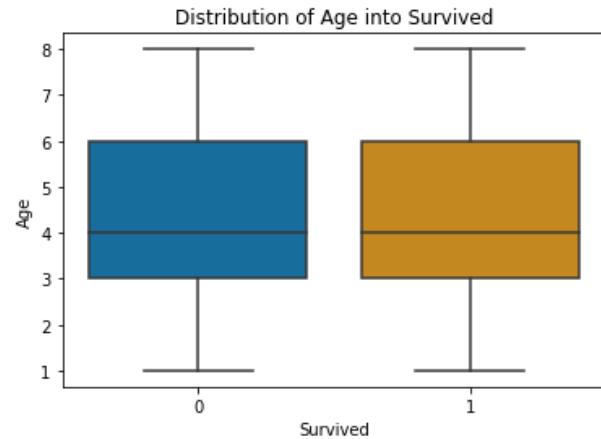
```
plt.figure(figsize=(6,4))
survived.value_counts().plot(kind='bar',color=['darkred','blue']) # counting category values and plotting
plt.title('Our class categorys frequency'); # defining a title
```



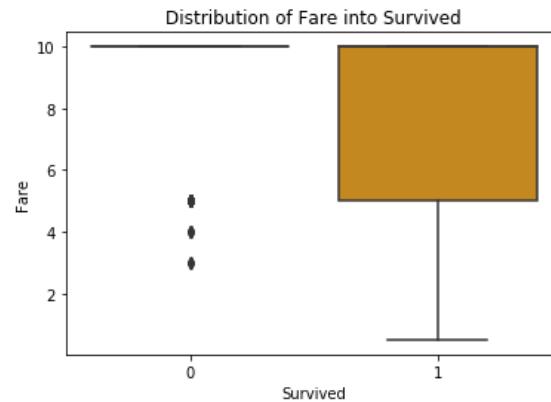
## 생존 분석

- 사망자가 더 많았다.
- 생존자 예측이 더 어렵다. 사망자 예측은 잘 된다.

# 피처엔지니어링 분석(1)



- outlier가 없다. 3-6 그룹에 집중된 데이터.



가격은 5개로 구분하면,

- 생존자의 가격대가 잘 보인다.

# 피처엔지니어링 분석(2)

```
gender = {'male':0,'female':1} # creating a dictionary to storage numeric representation of male and female
data = [titanic] # passing as a list

for dataset in data: # loop
    dataset['Sex'] = dataset['Sex'].map(gender) # mapping by gender
```

## 이름 변수에 대한 작업

```
data = [titanic] # passing titanic dataset as a list

for dataset in data: # loop to go through the list
    dataset['Title'] = dataset.Name.str.extract('([A-Za-z]+)\.', expand = False) # extracting titles from Name column

    #Replace title with more common one
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess', 'Capt', 'Col', 'Don', 'Dr',
                                                'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
    dataset['Title'] = dataset['Title'].fillna('unknow') # fill na's
    dataset['Title'] = dataset['Title'].astype('category') # converting into categories

titanic = titanic.drop('Name',axis=1) # Dropping 'Name' column
```

```
titanic.head() # Verifying result
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked	Title
0	1	3	0	6	1	0	10.0	NaN	S	Mr
1	2	1	1	3	1	0	5.0	C85	C	Mrs
2	3	3	1	5	0	0	10.0	NaN	S	Miss
3	4	1	1	3	1	0	5.0	C123	S	Mrs
4	5	3	0	3	0	0	10.0	NaN	S	Mr

# 피처엔지니어링 분석(3)

```
# creating a variables for passengers that have a family aboard
titanic['family_aboard'] = np.where((titanic['Parch']>=1) & (titanic['SibSp']>=1),1,0)

# creating a variable to represent couples with no childrens
titanic['no_child_couple'] = np.where((titanic['Parch']==0) & (titanic['SibSp']==1),1,0)

# creating a variables to describe the family size
titanic['family_small'] = np.where((titanic['Parch']<=1) & (titanic['SibSp']<=1),1,0)
titanic['family_median'] = np.where((titanic['Parch']>1) & (titanic['SibSp']>1) & (titanic['Parch']<=2) & (titanic['SibSp']<=2),1,0)
titanic['family_large'] = np.where((titanic['Parch']>2) & (titanic['SibSp']>2),1,0)
```

Now let's drop the original 'Parch' and 'SibSp' variables

```
titanic = titanic.drop(['Parch', 'SibSp'],axis=1) # drooping the 'Parch' and 'SibSp' columns
```

```
titanic.head()
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	Deck	family_aboard	no_child_couple	family_small	family_median	family_large
0	1	3	0	6	10.0	S	Mr	H	0	1	1	0	0
1	2	1	1	3	5.0	C	Mrs	C	0	1	1	0	0
2	3	3	1	5	10.0	S	Miss	H	0	0	1	0	0
3	4	1	1	3	5.0	S	Mrs	C	0	1	1	0	0
4	5	3	0	3	10.0	S	Mr	H	0	0	1	0	0

```
len(titanic)
```

1309

# 피처엔지니어링 분석(4)

## One-Hot-Encoding

원핫인코딩은 숫자에 대상하것을 주어진 영역에 표시

## OneHot Encoding

workclass	State-gov	Self-emp-not-inc	Private
State-gov	1	0	0
Self-emp-not-inc	0	1	0
Private	0	0	1
Private	0	0	1
Private	0	0	1

# 피처엔지니어링 분석(5)

```
titanic.head() # seeing the first 8 rows
```

	PassengerId	Pclass	Sex	Age	Fare	family_aboard	no_child_couple	family_small	family_median	family_large	...	Rare	A	B	C	D	E	F	G	H	T
0	1	3	0	6	10.0	0	1	1	0	0	0 ...	0	0	0	0	0	0	0	0	1	0
1	2	1	1	3	5.0	0	1	1	0	0	0 ...	0	0	0	1	0	0	0	0	0	0
2	3	3	1	5	10.0	0	0	1	0	0	0 ...	0	0	0	0	0	0	0	0	0	1
3	4	1	1	3	5.0	0	1	1	0	0	0 ...	0	0	0	1	0	0	0	0	0	0
4	5	3	0	3	10.0	0	0	1	0	0	0 ...	0	0	0	0	0	0	0	0	0	1

5 rows × 28 columns

When we use get.dummies function maybe some of the new columns are created with the same name, let's verify and solve this

```
titanic.columns # seeing columns names
```

```
Index(['PassengerId', 'Pclass', 'Sex', 'Age', 'Fare', 'family_aboard',
       'no_child_couple', 'family_small', 'family_median', 'family_large', 'C',
       'Q', 'S', 'U', 'Master', 'Miss', 'Mr', 'Mrs', 'Rare', 'A', 'B', 'C',
       'D', 'E', 'F', 'G', 'H', 'T'],
      dtype='object')
```

# 사이킷런 모델링을 위한 전처리(1)

## 모델링을 위한 데이터 전처리

```
train = titanic[0:891] # taking the first 891 rows to train  
test = titanic[891:1310]
```

Let's see if works well

```
print('train shape:',train.shape)  
print('test shape:',test.shape)
```

```
train shape: (891, 28)  
test shape: (418, 28)
```

```
# Saving PassengerId of test dataset to create submission dataset later  
passengerId = test['PassengerId']  
  
# Drooping PassengerId column from train and test  
train = train.drop('PassengerId',axis=1)  
test = test.drop('PassengerId',axis=1)
```

```
x_train = train # defining train dataset  
y_train = survived # defining target to train  
x_test = test # defining test dataset
```

# 사이킷런 모델링을 위한 전처리(2)

```
x_train.head()
```

	Pclass	Sex	Age	Fare	family_aboard	no_child_couple	family_small	family_median	family_large	C_1	...	Rare	A	B	C_2	D	E	F	G	H	T
0	3	0	6	10.0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	3	5.0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
2	3	1	5	10.0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	1	1	3	5.0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4	3	0	3	10.0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

5 rows × 27 columns

```
y_train.head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

# 사이킷런 모델 및 Score(학습)

사이킷런으로 모델링하자. 85% 정확도를 목표로 ¶

```
model = tree.DecisionTreeClassifier() # creating the model  
model.fit(x_train,y_train) # training the model on train dataset  
  
DecisionTreeClassifier(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_impurity_split=None,  
                      min_samples_leaf=1, min_samples_split=2,  
                      min_weight_fraction_leaf=0.0, presort='deprecated',  
                      random_state=None, splitter='best')  
  
model.score(x_train,y_train) # seeing accuracy  
0.9068462401795735
```

헉! 90.68% 정확도라니. 미쳤다.

# 숙제 : 타이타닉의 예측 정확도는?

```
y_pred = pd.DataFrame(model.predict(x_test)) # predicting on test dataset

# creating the submission file
y_pred['Survived'] = y_pred[0]
y_pred.drop(0, axis=1, inplace=True)
y_pred['PassengerId'] = passengerId
y_pred_Dtrees = y_pred

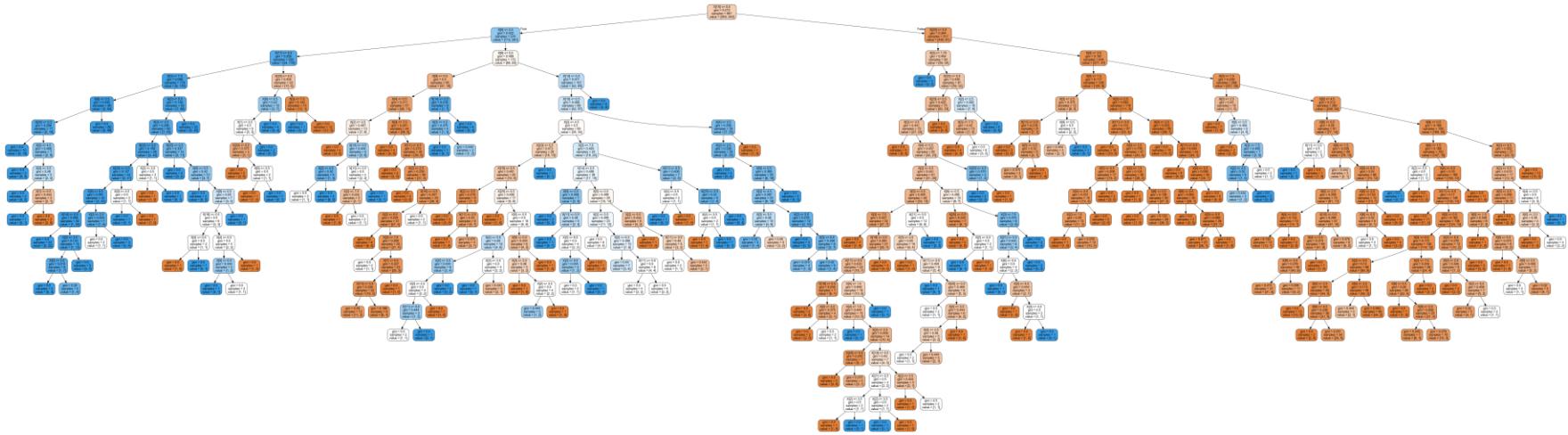
y_pred_Dtrees.to_csv('Decision_tree_model.csv', index=False) # exporting submission file
```

숙제 1:

타이타닉 데이터를 max\_depth=1일 때,  
앞의 90.68% 정확도를 활용하여,  
테스트(test) 데이터에 대한 y\_pred 정확도를 예측하시오.

# 타이타닉 결정 트리 : 뭔가 이상해?

```
graph = tree.export_graphviz(model, out_file = None, rounded = True, filled = True)
graphviz.Source(graph)
```



문제:

위의 그래프에서 max\_depth는 얼마인가?

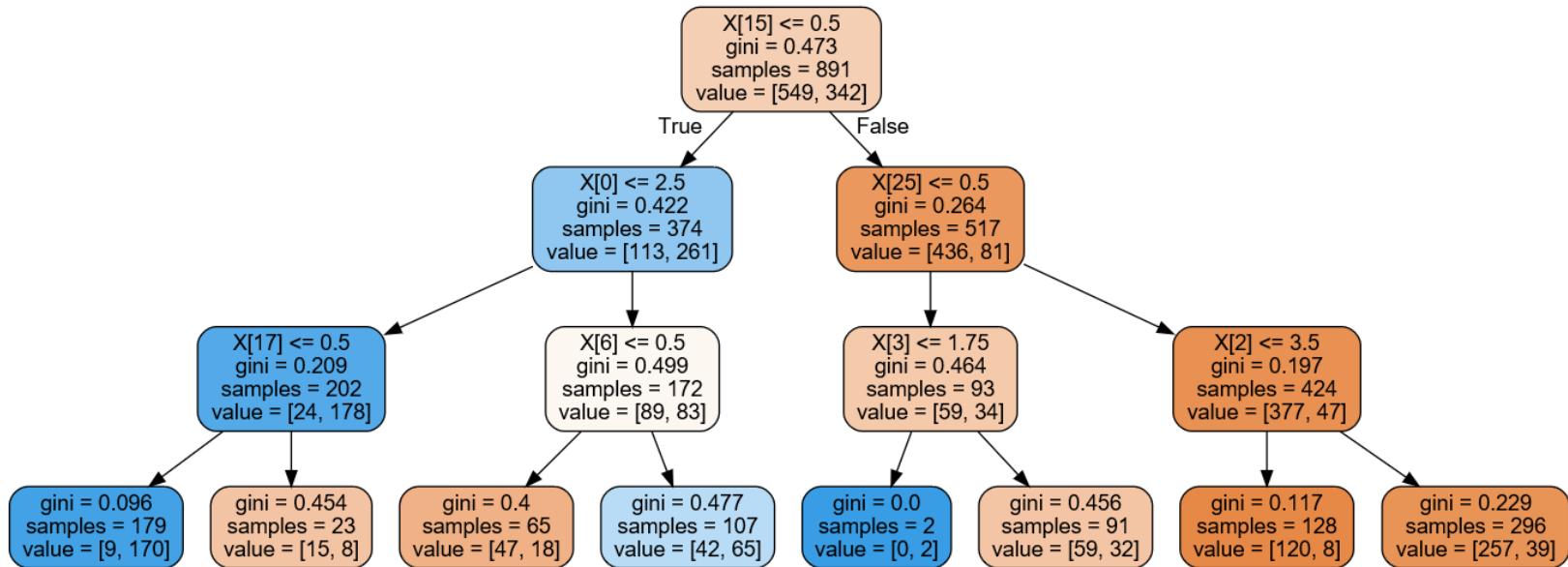
# 파라미터 max\_depth=3 적용(1)

```
model3 = tree.DecisionTreeClassifier(criterion = "gini", max_depth = 3) # creating the model  
  
model3.fit(x_train,y_train) # training the model on train dataset  
  
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',  
                      max_depth=3, max_features=None, max_leaf_nodes=None,  
                      min_impurity_decrease=0.0, min_impurity_split=None,  
                      min_samples_leaf=1, min_samples_split=2,  
                      min_weight_fraction_leaf=0.0, presort='deprecated',  
                      random_state=None, splitter='best')  
  
model3.score(x_train,y_train) # seeing accuracy  
0.8249158249158249
```

82.49% pruning을 한것은 과적합을 방지한 것인가?

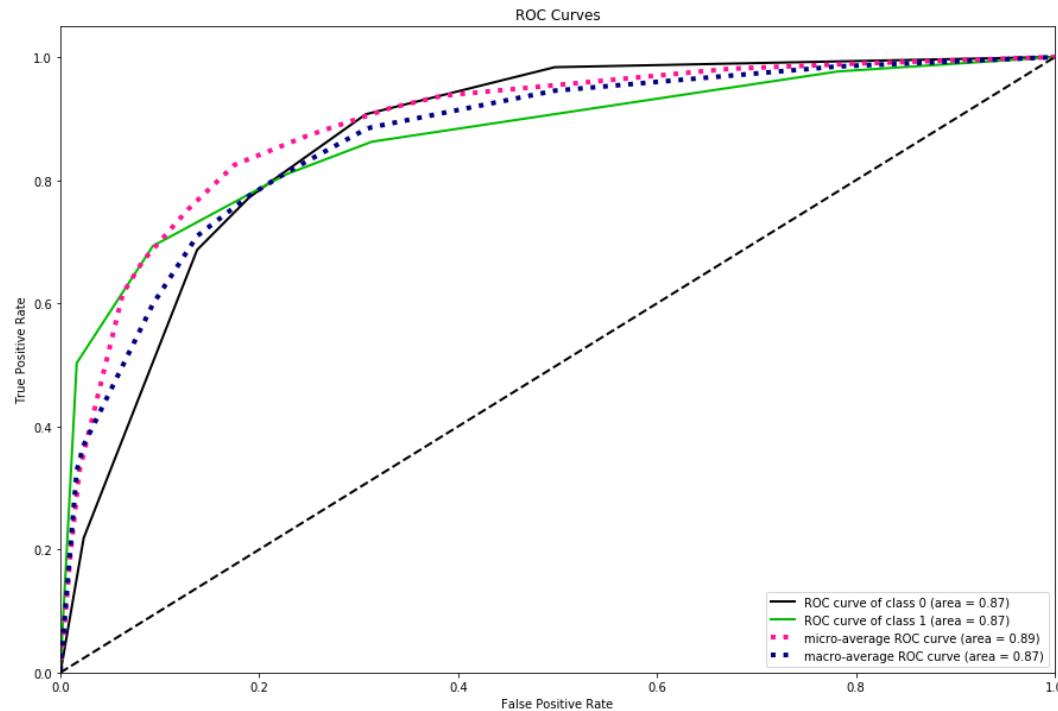
```
graph = tree.export_graphviz(model3, out_file = None, rounded = True, filled = True)  
graphviz.Source(graph)
```

# 파라미터 max\_depth=3 적용(2)



소결: 괜 좋아졌군요. 82.49% 정확도!

# ROC 곡선 (1)



```
# ROC 커브 그리는 패키지
import scikitplot as skplt # ROC curve

y_probs = model3.predict_proba(x_train) # predicting probabilities to plot ROC curve
skplt.metrics.plot_roc(y_train, y_probs, figsize=(15,10)) # creating the plot
plt.show() # showing the plot
```

# 생존자 예측 결과

```
y_pred = pd.DataFrame(model3.predict(x_test))

# creating the submission file
y_pred['Survived'] = y_pred[0]
y_pred.drop(0, axis=1, inplace=True)
y_pred['PassengerId'] = passengerId
y_pred_Dtrees2 = y_pred

y_pred_Dtrees2.to_csv('Decision_tree_pruned.csv', index=False)
```

The screenshot shows a Microsoft Excel window with two files listed in the title bar: "Decision\_tree\_model.csv" and "Decision\_tree\_pruned.csv". The "Decision\_tree\_pruned.csv" file is currently open, showing its contents in a table format. The table has two columns: "Survived" and "PassengerId". The data starts from row 2 and continues to row 21. The "Survived" column contains binary values (0 or 1) representing survival status, and the "PassengerId" column contains passenger IDs ranging from 892 to 911.

	A	B	C	D	E	F	G	H	I
1	Survived	PassengerId							
2	0	892							
3	0	893							
4	0	894							
5	0	895							
6	1	896							
7	0	897							
8	0	898							
9	0	899							
10	1	900							
11	0	901							
12	0	902							
13	0	903							
14	1	904							
15	0	905							
16	1	906							
17	1	907							
18	0	908							
19	0	909							
20	0	910							
21	0	911							

# 승객의 생존 예측

## Prediction for Passengers

```
passengers_set_1 = titanic_df[titanic_df.pclass == 1].iloc[:4,:].copy()
passengers_set_2 = titanic_df[titanic_df.pclass == 2].iloc[:4,:].copy()
passengers_set_3 = titanic_df[titanic_df.pclass == 3].iloc[:2,:].copy()
passenger_set = pd.concat([passengers_set_1,passengers_set_2,passengers_set_3])
passenger_set
```

	pclass	name	sex	age	sibsp	parch	ticket	fare
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500
323	2	Abelson, Mr. Samuel	male	30.0000	1	0	P/PP 3381	24.0000
324	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0000	1	0	P/PP 3381	24.0000
325	2	Aldworth, Mr. Charles Augustus	male	30.0000	0	0	248744	13.0000
526	2	Andrew, Mr. Edgardo Samuel	male	18.0000	0	0	231945	11.5000
600	3	Abbing, Mr. Anthony	male	42.0000	0	0	C.A. 5547	7.5500
601	3	Abbott, Master. Eugene Joseph	male	19.0000	0	2	C.A. 2673	20.2500

1등석  
(25백만원)

2등석  
(4백만원)

3등석  
(1백만원)

# 머신러닝이란?

- 머신러닝
  - ✓ 데이터로 부터 학습하도록 컴퓨터를 프로그래밍하는 과학
  - ✓ 어떤 작업 T에 대한 컴퓨터 프로그램의 성능을 P로 측정했을 때 경험 E로 인해 성능이 향상됐다면, 이 컴퓨터 프로그램은 작업 T와 성능 측정 P에 대한 경험 E로 학습한 것

# Thank You!

[www.ust.ac.kr](http://www.ust.ac.kr)

# Gini impurity

# CART cost function for classification

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where  $\begin{cases} G_{\text{left/right}} & \text{measures the impurity of the left/right subset,} \\ m_{\text{left/right}} & \text{is the number of instances in the left/right subset.} \end{cases}$

# CART cost function for regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}}$$

where  $\begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$