

의사 결정 트리 (Decision Tree)

학습 목표

- 의사 결정 트리 모델의 개념과 구현을 알아본다.

주요 내용

1. 의사 결정 트리
2. 데이터셋
3. 엔트로피
4. 노드 분할 방식
5. 의사 결정 트리 구현
6. scikit-learn 활용

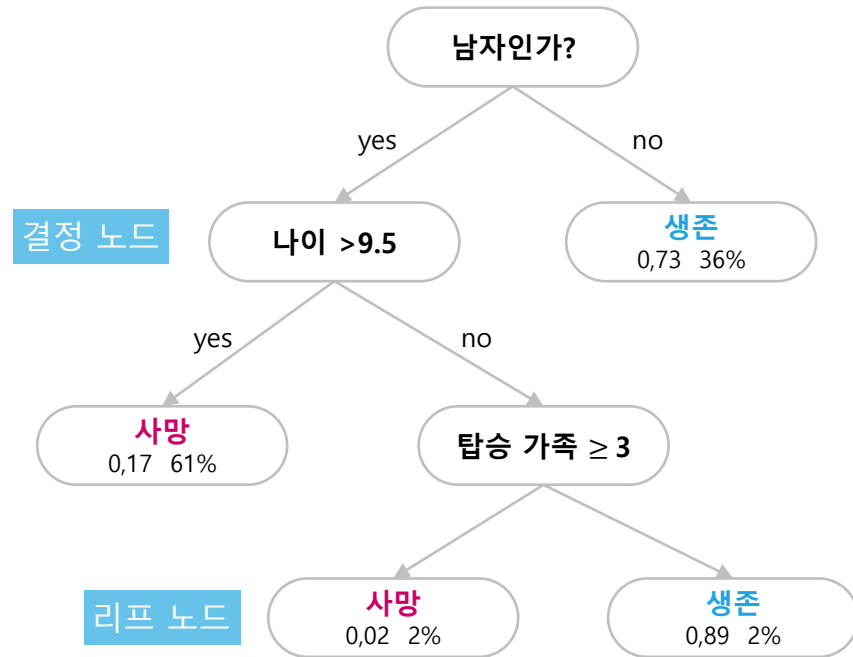


1. 의사 결정 트리



의사 결정 트리 (Decision Tree)

타이타닉호 탑승객의 생존 예측

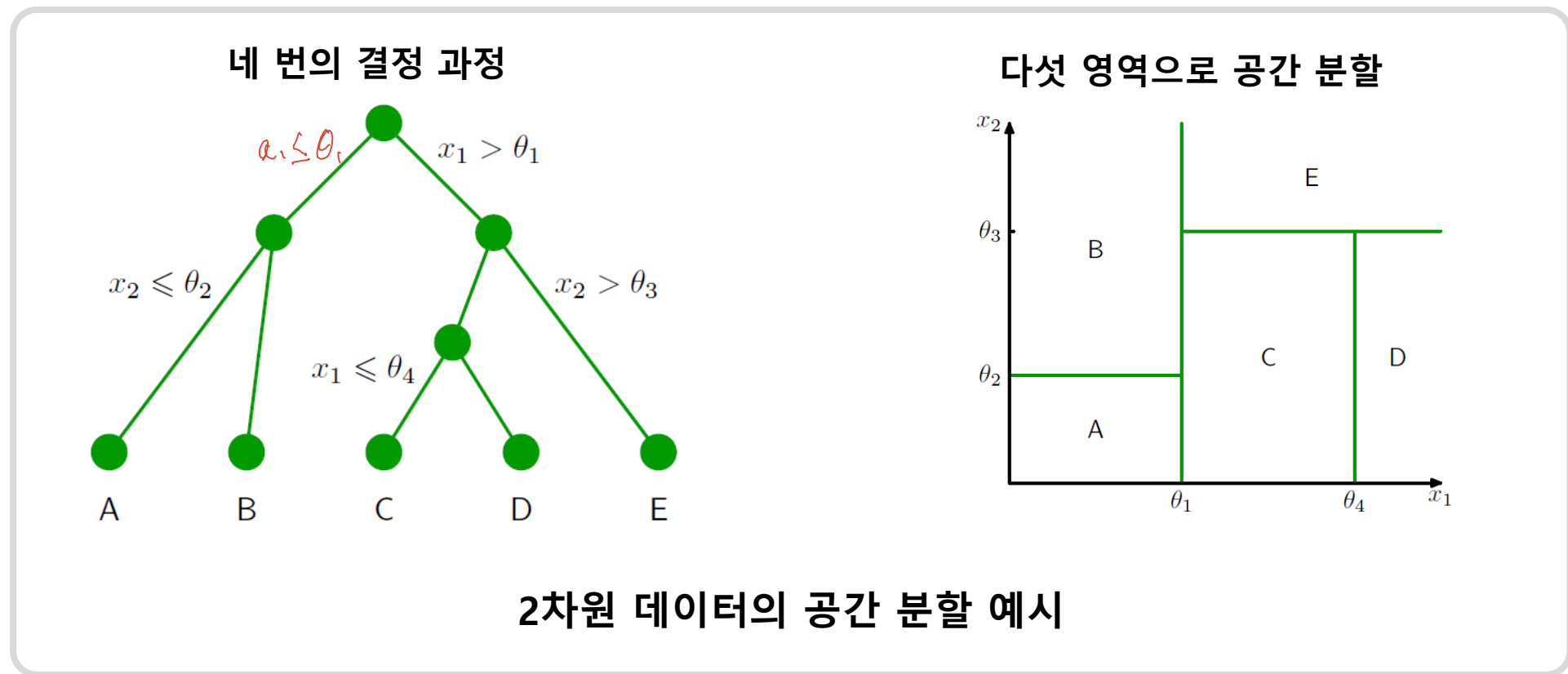


(생존 확률, 이 노드에 도달할 확률)

트리 구조 형태의 순차적인 결정을 통해
예측을 하는 분류/회귀 기법

결정 경계 (Decision Boundary)

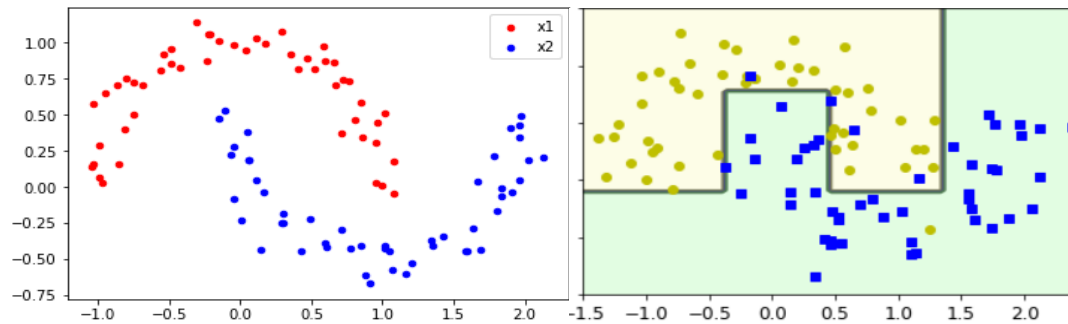
의사 결정 트리는 입력 데이터 공간의 좌표 축과 평행한 결정 경계로 나누는 방식



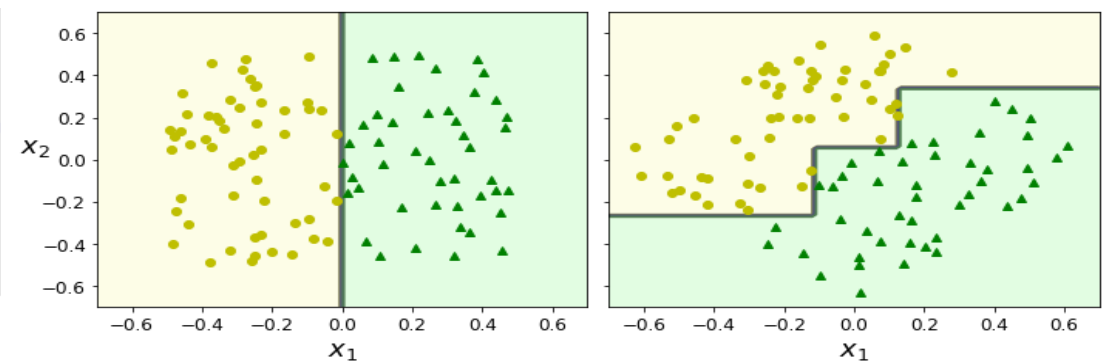
결정 경계 (Decision Boundary)

결정 경계가 입력 데이터의 좌표 축과 평행하기 때문에
최적의 분할 방식을 보장하지 못함

경계가 곡선일 때



데이터가 회전 했을 때



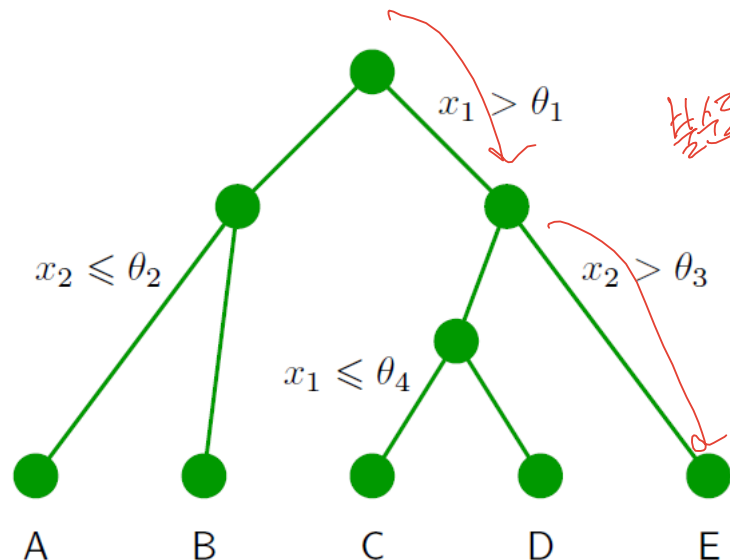
결정 경계의 방향이 기울어질수록 분할이 증가

모델 학습 방식

최적의 결정을 하는 의사 결정 트리를 어떻게 생성할 것인가?



데이터를 분석해서
의사 결정 트리를 자동으로 생성



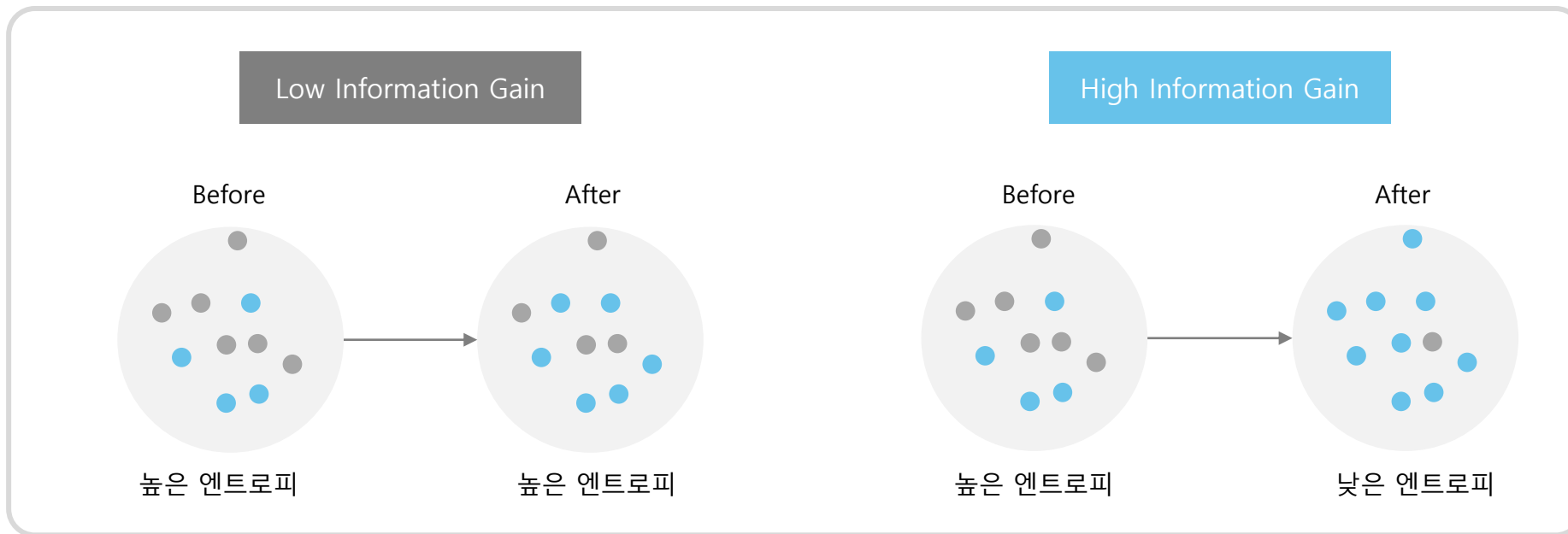
불순도
원하는 class의 개수
의 불균형

정보의 획득이 가장 큰 or 불확실성을 가장 많이 낮추는 조건을 찾자!

정보의 획득 (Information Gain)

정보의 획득은 엔트로피의 변화량을 말한다.

$$\text{Information Gain (IG)} = \text{Entropy(Before)} - \text{Entropy(After)}$$



클래스가 단일화될수록 엔트로피는 낮아지므로 정보의 획득이 높아진다.

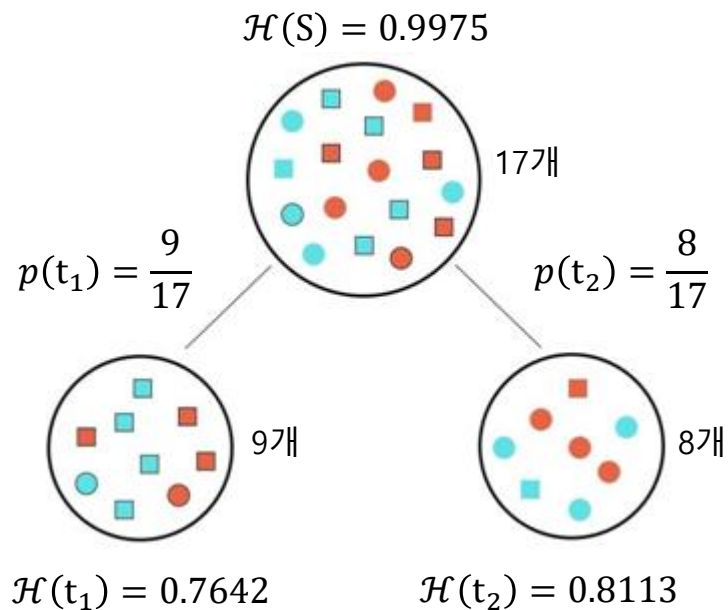
같은 클래스로 이뤄지는 비율이 높을수록 순도(purity)가 높아진다고 표현하기도 한다.

정보의 획득 (Information Gain)

의사 결정 트리에서 정보의 획득은

부모의 엔트로피와 자식들의 엔트로피의 가중 합산의 차로 표현할 수 있다.

$$\text{Information Gain (IG)} = \text{Entropy}(\text{Parent}) - \sum \text{Weight} * \text{Entropy}(\text{Child})$$



$$\text{IG}(S) = \mathcal{H}(S) - \sum_{t \in T} p(t) \mathcal{H}(t) \quad S = \bigcup_{t \in T} t$$

$\mathcal{H}(S)$: 집합 S 의 엔트로피

T : 집합 S 에서 분할된 부분 집합들

$p(t)$: 집합 S 에서 t 로 분할된 요소들의 비율

$\mathcal{H}(t)$: 부분집합 t 의 엔트로피

참고 정보량 (Self-Information)

확률 변수의 정보량은?

- 정보란 **놀라움의 정도**를 의미한다. 😲
- 발생 확률이 낮은 사건일수록 정보가 크다.

$$\frac{1}{p(x)}$$

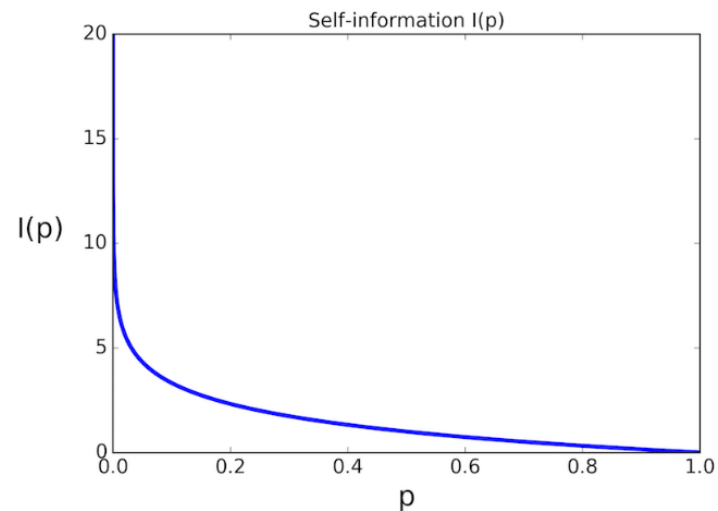
- 정보를 표현하는 Bit 수를 구해보자.

$$\log \frac{1}{p(x)} = -\log p(x)$$

정보량 (Self-Information)

확률 변수의 확률 값을 표현하기 위해 필요한 Bit 수

$$I(x) = -\log p(x)$$



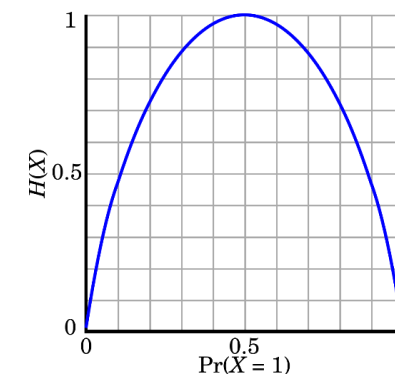
참고 엔트로피 (Entropy)

엔트로피 (Entropy)

- 확률 변수의 정보량의 기댓값
- 확률 분포가 얼마나 불확실한지 또는 랜덤한지를 나타냄

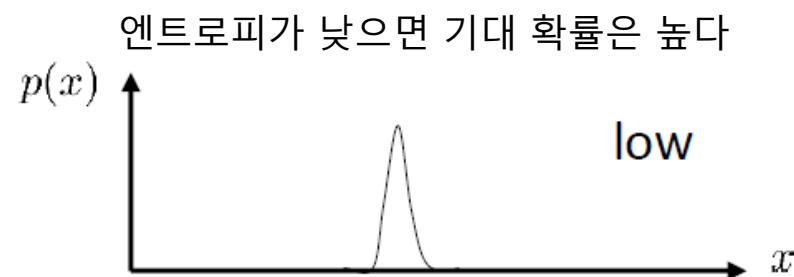
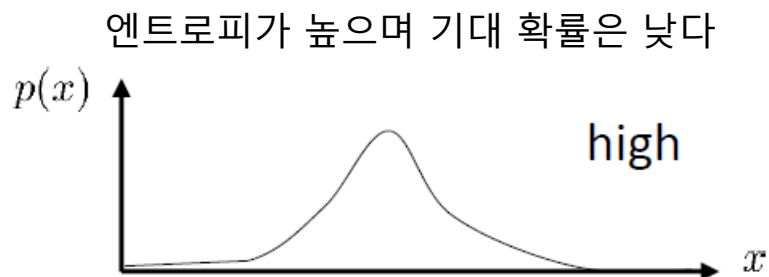
$$\mathcal{H}(p) = -\mathbb{E}_{x \sim p(x)}[\log p(x)] = -\int_x p(x) \log p(x) dx$$

$E(x) = \int p(x) \cdot x dx$ $= -\sum p(x) \log p(x)$



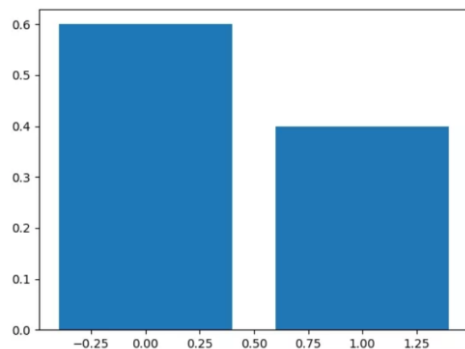
동전을 던졌을 때 앞면과
뒷면이 나올 확률의 엔트로피

확률 변수 x 가 얼마나 random한가?



참고 엔트로피 (Entropy)

베르누이 분포 (Bernoulli Distribution)



c_1 : 클래스 1 c_2 : 클래스 2

$$p(x; \mu) = \mu^x (1 - \mu)^{1-x}$$
$$x \in \{0, 1\}$$

클래스 1이 나올 확률 $p(x = 1) = \mu$

클래스 2가 나올 확률 $p(x = 0) = 1 - \mu$

만일 $\mu = 0.5$ 일 때 엔트로피는?

$$\mathcal{H}(p) = -\mathbb{E}_{x \sim p(x)}[\log p(x)]$$

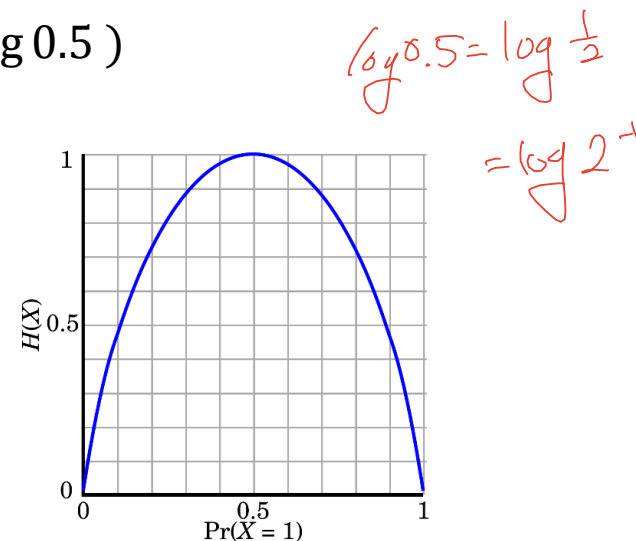
$$= -(p(x = 1) \log p(x = 1) + p(x = 0) \log p(x = 0))$$

$$= -(\mu \log \mu + (1 - \mu) \log(1 - \mu))$$

$$= -(0.5 \log 0.5 + 0.5 \log 0.5)$$

$$= -\log 2^{-1}$$

$$= 1$$



가지치기 (Pruning)

- 사전 가지치기 (pre-pruning)**
- 노드의 오차가 임계치 이하가 되면 더 이상 분할을 하지 않음
 - 분류 : 크로스 엔트로피 또는 지니 계수 (Gini Index)
 - 회귀 : 평균 제곱 오차 (MSE)

지니 계수가 속도가 더 빠르며 모델 성능은 비슷함

분류

크로스 엔트로피 (Cross Entropy)

$$Q_t(T) = \sum_{k=1}^K p_k(t) \ln p_k(t)$$

*위계화할수록
불확실성이 클수록
엔트로피가 커진다
즉, 불확실성이 클수록 엔트로피가 커진다*

지니 계수 (Gini Index)

$$Q_t(T) = \sum_{k=1}^K p_k(t)(1 - p_k(t))$$

= 1 - \sum p(A)^2

$t = 1, 2, \dots, |T|$: 리프 노드의 인덱스
 $|T|$: 리프 노드의 개수
 K : 클래스 개수

회귀

평균 제곱 오차 (MSE)

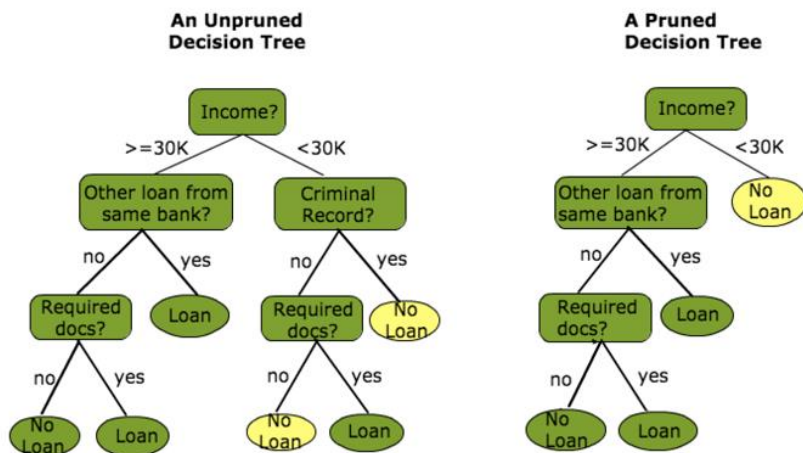
$$Q_t(T) = \sum_{n=1}^N (t_n - x_n)^2$$

N : 데이터 개수

가지치기 (Pruning)

사후 가지치기 (post-pruning)

- 리프 노드에 데이터 개수가 일정 개수 미만이 되도록 최대한 크게 만들고, 오차가 적은 노드에서 가지치기 하는 방식
- 과적합을 막는 정규화 기법이기도 함



노드를 분할해서 바로 오차를 줄이지 않더라도 분할을 더해 보면 오차가 줄어드는 경우가 많기 때문에, 큰 트리를 만들어서 과적합 시킨 후에 가지치기를 하는 것이 성능에 좋다.

가지치기 조건 (비용 함수)

모델의 복잡도를 작게 만드는 정규화 항

$$C(T) = \sum_{t=1}^{|T|} Q_t(T) + \lambda |T|$$

잘못된 것 같아
정규화 항

모델의 오차를 최소화 하는 항

$Q_t(T)$: 리프 노드에서의 오차 또는 비용

λ : 정규화 계수

$|T|$: 리프 노드의 개수

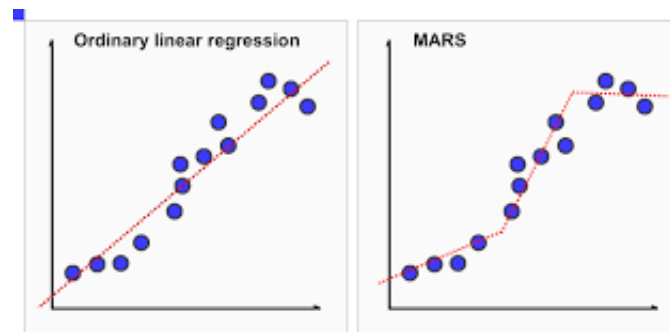
알고리즘 종류

- **ID3** (Iterative Dichotomiser 3)
- **C4.5** (successor of ID3)
- **C5.0** (successor of ID4)

데이터 마이닝에서 주로 사용

- 인공지능/기계학습 분야에서 개발
- 엔트로피, 정보이득 개념을 사용하여 노드 분할 기준을 결정

- **MARS** (Multivariate adaptive regression splines)



- **CART** (Classification And Regression Tree)
- **CHAID** (CHi-squared Automatic Interaction Detector)
- **조건부 추론 트리** (Conditional Inference Trees)

- 통계학 분야에서 개발 *필요를 위한 것*
- 카이스퀘어, T검정, F검정 등의 통계분석법을 사용해서 노드 분할 기준을 결정

ID3 알고리즘

ID3는 범주형 데이터만 분류할 수 있는 가장 간단한 의사 결정 트리 알고리즘

- 루트 노드 생성
- 다음 과정을 반복
 - (노드의 샘플이 모두 같은 클래스이면) 리프 노드로 만들고 해당 클래스로 분류
 - (더 이상 사용할 수 있는 속성이 없으면) 빈도수가 높은 클래스로 분류하고 **수행 종료**
 - (그 외) 정보획득(IG)이 가장 높은 속성을 선택해서 결정 노드 생성

탐욕 알고리즘 (Greedy Algorithm)

탐욕 방법 (Greedy Method) :

최적해를 찾기 위해 반복적으로 실행 과정에서 매번 의사 결정을 할 때마다 그 순간에 가장 좋은 것을 선택하는 방식

장점과 단점

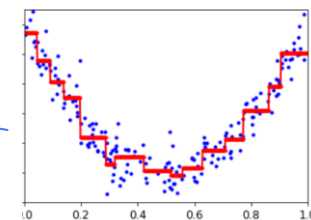
장점

- 적용하고 이해하기가 쉬움
- 예측 과정을 쉽게 설명하고 해석할 수 있음
- 숫자형 데이터와 범주형 데이터를 동시에 다룰 수 있음
- 변수 값이 누락되어도 사용할 수 있음
- 자료를 가공할 필요가 거의 없다.
- 큰 데이터셋에도 사용할 수 있음

단점

- 과적합/과소적합이 쉽게 생길 수 있음
- 데이터가 바뀌면 트리가 민감하게 바뀌게 됨
- 단계가 많은 특징의 분할로 편향될 수 있음
- 결정 경계가 입력 데이터의 좌표 축과 평행이라 최적의 해를 보장하지는 않음
- 트리가 커지면 결정의 직관성이 떨어짐
- 회귀 모델의 경우 불연속 함수가 생기게 됨

→ = 과적합 = 과소적합



과적합을 막기 위해 정규화 방식으로 랜덤 포레스트 or 속성의 랜덤 선택 방식을 적용

2. 데이터셋



예제 인터뷰 후보자 합격 예측

인터뷰 후보자의 데이터로 인터뷰의 긍정 평가를 예측하는
의사결정 트리 모델을 만들어보자.



데이터 NamedTuple

인터뷰 후보자 NamedTuple

```
from typing import NamedTuple, Optional

class Candidate(NamedTuple):
    level: str
    lang: str
    tweets: bool
    phd: bool
    did_well: Optional[bool] = None # allow unlabeled data
```

(직급, 선호하는 프로그래밍 언어, 트위터 사용, 박사 학위, 인터뷰 긍정 평가)

- level : 직급 {Junior, Mid, Senior}
- lang : 선호하는 프로그래밍 언어 {Java, Python, R, Java}
- tweets : 트위터 사용 {True, False}
- phd : 박사 학위 {True, False}
- did_well : 인터뷰 긍정 평가 {True, False}

데이터셋

데이터셋 (직급, 선호하는 프로그래밍 언어, 트위터 사용, 박사 학위, 인터뷰 긍정 평가)

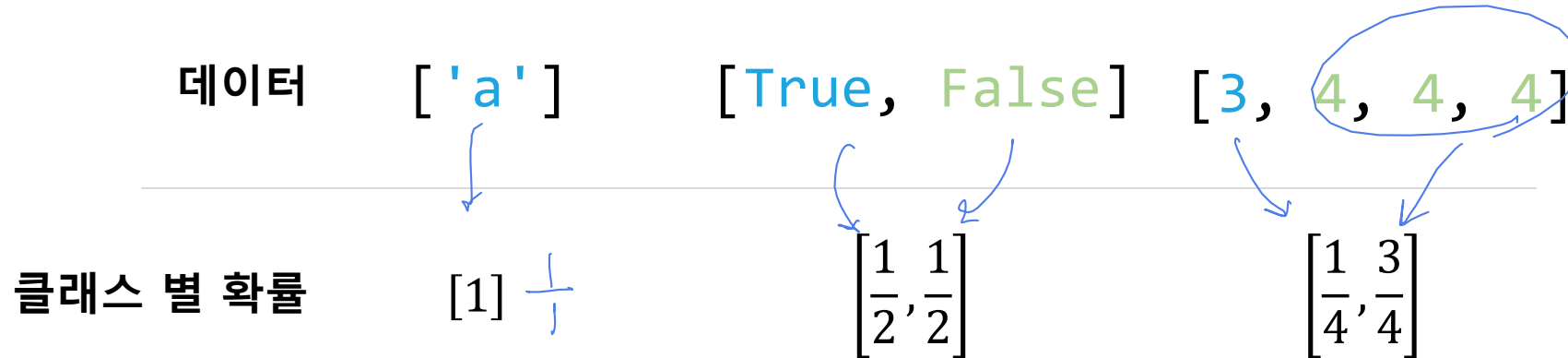
```
inputs = [Candidate('# level lang tweets phd did_well',
Candidate('Senior', 'Java', False, False, False),
Candidate('Senior', 'Java', False, True, False),
Candidate('Mid', 'Python', False, False, True),
Candidate('Junior', 'Python', False, False, True),
Candidate('Junior', 'R', True, False, True),
Candidate('Junior', 'R', True, True, False),
Candidate('Mid', 'R', True, True, True),
Candidate('Senior', 'Python', False, False, False),
Candidate('Senior', 'R', True, False, True),
Candidate('Junior', 'Python', True, False, True),
Candidate('Senior', 'Python', True, True, True),
Candidate('Mid', 'Python', False, True, True),
Candidate('Mid', 'Java', True, False, True),
Candidate('Junior', 'Python', False, True, False)
]
```

3. 엔트로피 구현



클래스 별 확률

데이터에서 클래스 별 확률은 각 클래스의 빈도수를 전체 데이터 개수로 나눈 값



클래스 별 확률 계산

```
from typing import Any
from collections import Counter


def class_probabilities(labels: List[Any]) -> List[float]:
    total_count = len(labels)
    return [count / total_count
            for count in Counter(labels).values()]
```

- 클래스 별 비율을 확률로 계산

데이터 엔트로피

데이터의 엔트로피는 클래스 별 확률 리스트에 대한 엔트로피 값

데이터	['a']	[True, False]	[3, 4, 4, 4]
클래스 별 확률	[1]	$\left[\frac{1}{2}, \frac{1}{2}\right]$	$\left[\frac{1}{4}, \frac{3}{4}\right]$
엔트로피	0	1	0.81



데이터 엔트로피 계산

```
def data_entropy(labels: List[Any]) -> float:  
    return entropy(class_probabilities(labels))
```

- 클래스 별 확률에 대한 엔트로피 계산

엔트로피 (Entropy)

엔트로피 계산

```
from typing import List
import math

def entropy(class_probabilities: List[float]) -> float:
    """Given a list of class probabilities, compute the entropy"""
    return sum(-p * math.log(p, 2)
               for p in class_probabilities)
```

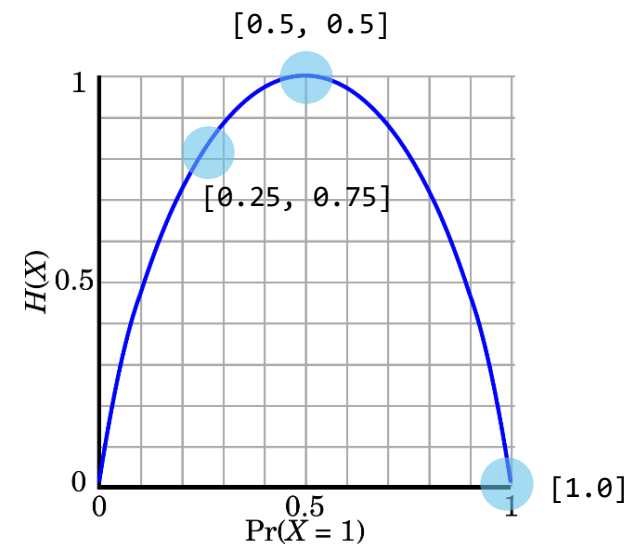
- 데이터의 클래스 별 확률 리스트를 받아서 엔트로피 계산

테스트

```
assert entropy([1.0]) == 0
assert entropy([0.5, 0.5]) == 1
assert 0.81 < entropy([0.25, 0.75]) < 0.82
```

- 확률이 1이면 엔트로피는 0
- 두 클래스의 확률이 0.5, 0.5이면 엔트로피는 1
- 두 클래스의 확률이 0.25, 0.75이면 엔트로피는 0.81과 0.82 사이 값

$$\mathcal{H}(p) = - \sum_x p(x) \log p(x)$$



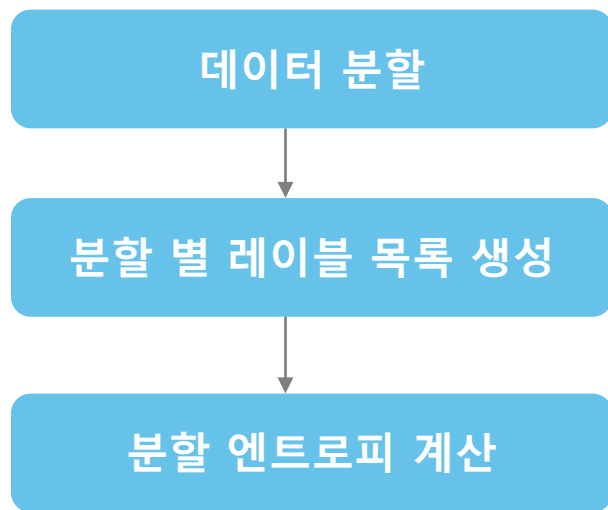
4. 노드 분할 방식



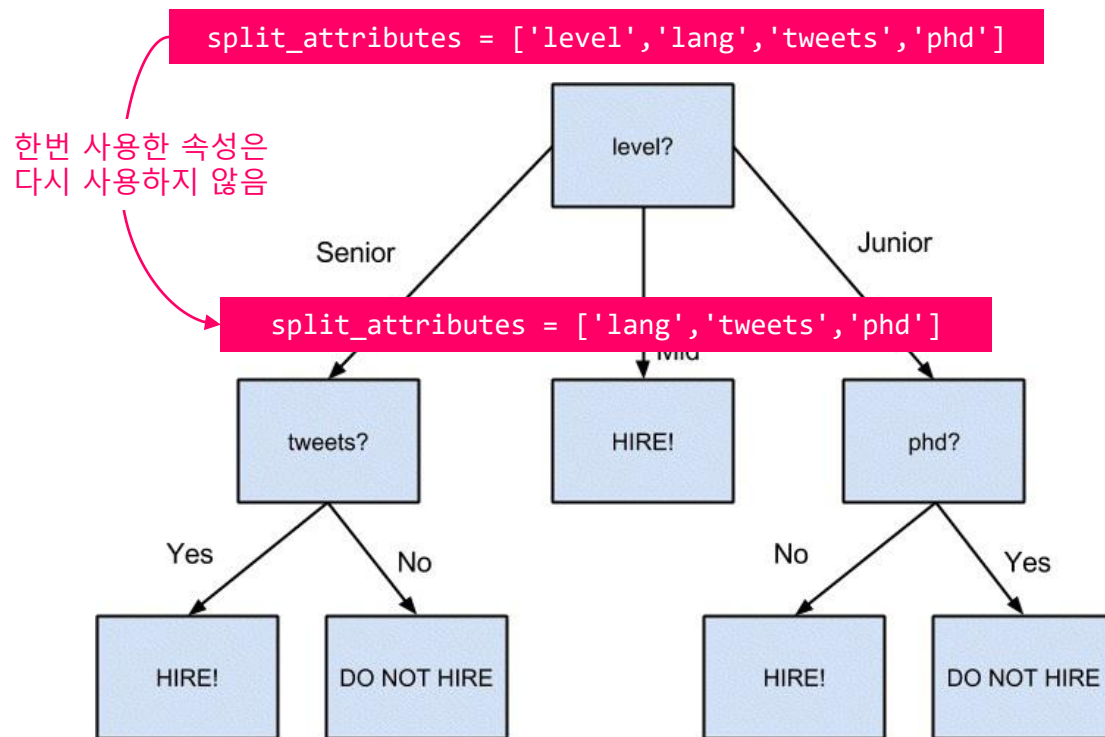
노드 분할 방식

결정 노드에서는 분할 엔트로피가 가장 작은 속성을 선택한다.

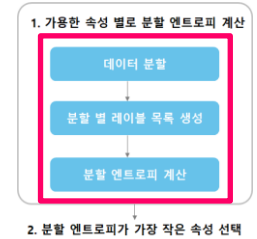
1. 가용한 속성 별로 분할 엔트로피 계산



2. 분할 엔트로피가 가장 작은 속성 선택



한 속성에 대해 분할 엔트로피 계산



level { Junior
mid
senior

```
def partition_entropy_by(inputs: List[Any],
                        attribute: str,
                        label_attribute: str) -> float:
    """Compute the entropy corresponding to the given partition"""
    # partitions consist of our inputs
    partitions = partition_by(inputs, attribute)

    # but partition_entropy needs just the class labels
    labels = [[getattr(input, label_attribute) for input in partition]
              for partition in partitions.values()]

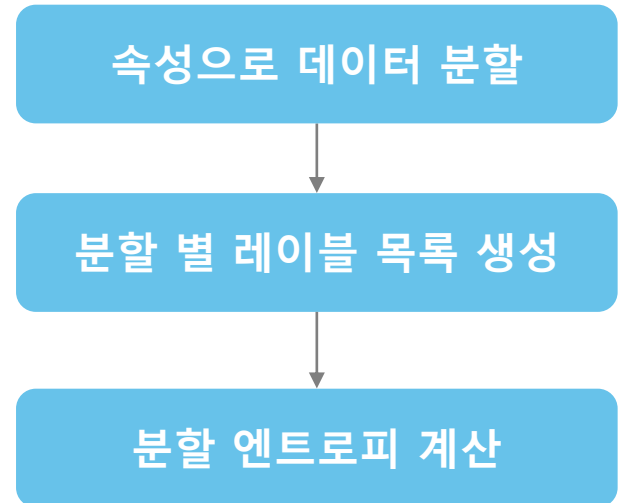
    return partition_entropy(labels)
```

각각의 level로 partition할 것

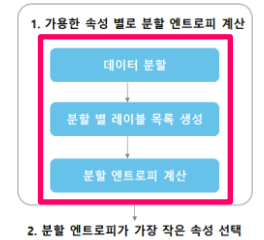
Partition entropy를 구해야 할 것.

List of list
[[T, F, T, ...],
[T, ...]]

- attribute : 데이터 분할에 사용할 데이터 속성
- label_attribute : 데이터의 레이블 속성
- getattr() : object의 속성(attribute) 값을 확인하는 함수



분할 엔트로피 계산 데이터 분할



범주형 속성(attribute) 값에 따라 데이터를 분할

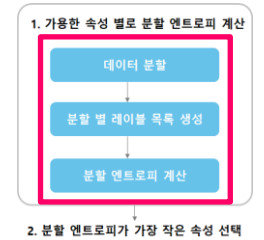
```
from typing import Dict, TypeVar
from collections import defaultdict

T = TypeVar('T') # generic type for inputs

def partition_by(inputs: List[T], attribute: str) -> Dict[Any, List[T]]:
    """Partition the inputs into lists based on the specified attribute."""
    partitions: Dict[Any, List[T]] = defaultdict(list)
    for input in inputs:
        key = getattr(input, attribute) # value of the specified attribute
        partitions[key].append(input) # add input to the correct partition
    return partitions
```

- TypeVar('T')은 Generic Type
- Partitions 딕셔너리 생성
- (attribute 값 기준으로) inputs 데이터를 분할해서 partitions 딕셔너리 구성
- getattr() : object의 속성(attribute) 값을 확인하는 함수

분할 엔트로피 계산 분할 엔트로피 (Partition Entropy)



분할 엔트로피

$$IG(S) = \mathcal{H}(S) - \sum_{t \in T} p(t) \mathcal{H}(t)$$

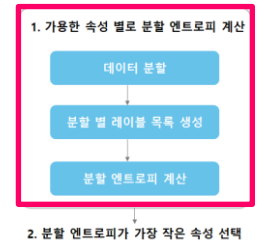
분할 엔트로피

```
def partition_entropy(subsets: List[List[Any]]) -> float:
    """Returns the entropy from this partition of data into subsets"""
    total_count = sum(len(subset) for subset in subsets)

    return sum(data_entropy(subset) * len(subset) / total_count
               for subset in subsets)
```

2/3-2/8-1

1차 분할 속성 탐색



데이터를 각 속성에 대해 분할 엔트로피를 계산해보면 1차 분할을 할 속성을 확인할 수 있다.

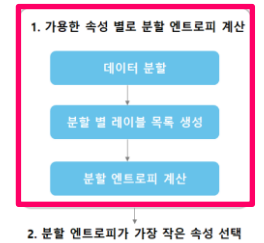
데이터 속성 별로 분할 엔트로피 확인

```
for key in ['level', 'lang', 'tweets', 'phd']:
    print(key, partition_entropy_by(inputs, key, 'did_well'))

assert 0.69 < partition_entropy_by(inputs, 'level', 'did_well') < 0.70
assert 0.86 < partition_entropy_by(inputs, 'lang', 'did_well') < 0.87
assert 0.78 < partition_entropy_by(inputs, 'tweets', 'did_well') < 0.79
assert 0.89 < partition_entropy_by(inputs, 'phd', 'did_well') < 0.90
```

직급이 가장 엔트로피를 낮추므로 직급을 기준으로 1차 분할을 한다.

분할 속성 탐색 2차 분할



직급을 기준으로 Senior, Mid, Junior 데이터에 대해 2차 분할 속성을 확인해 보자!

Senior 직급 데이터

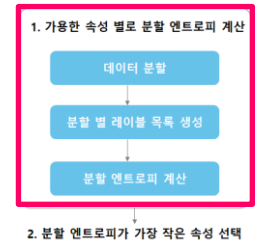
```
senior_inputs = [input for input in inputs if input.level == 'Senior']

assert 0.4 == partition_entropy_by(senior_inputs, 'lang', 'did_well')
assert 0.0 == partition_entropy_by(senior_inputs, 'tweets', 'did_well')
assert 0.95 < partition_entropy_by(senior_inputs, 'phd', 'did_well') < 0.96
```

- Senior 직급의 경우 '트위터 사용' 기준으로 분할 했을 때 엔트로피가 0
 - 트위터 사용하면 인터뷰 평가가 긍정
 - 트위터를 사용하지 않으면 인터뷰 평가가 부정

Senior 데이터는 '트위터 사용' 으로 2차 분할을 한다.

분할 속성 탐색 2차 분할



직급을 기준으로 Senior, Mid, Junior 데이터에 대해 2차 분할 속성을 확인해 보자!

Mid 직급 데이터

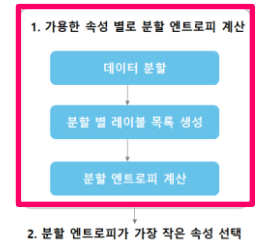
```
mid_inputs = [input for input in inputs if input.level == 'Mid']

print(partition_entropy_by(mid_inputs, 'lang', 'did_well'))
print(partition_entropy_by(mid_inputs, 'tweets', 'did_well'))
print(partition_entropy_by(mid_inputs, 'phd', 'did_well'))
```

0.0 • 분할 엔트로피가 모두 0
0.0 • Mid 데이터는 전체 데이터가 같은 클래스로 되어 있음을 알 수 있음
0.0

Mid 데이터는 더 이상 분할이 필요 없음

분할 속성 탐색 2차 분할



직급을 기준으로 Senior, Mid, Junior 데이터에 대해 2차 분할 속성을 확인해 보자!

Junior 직급 데이터

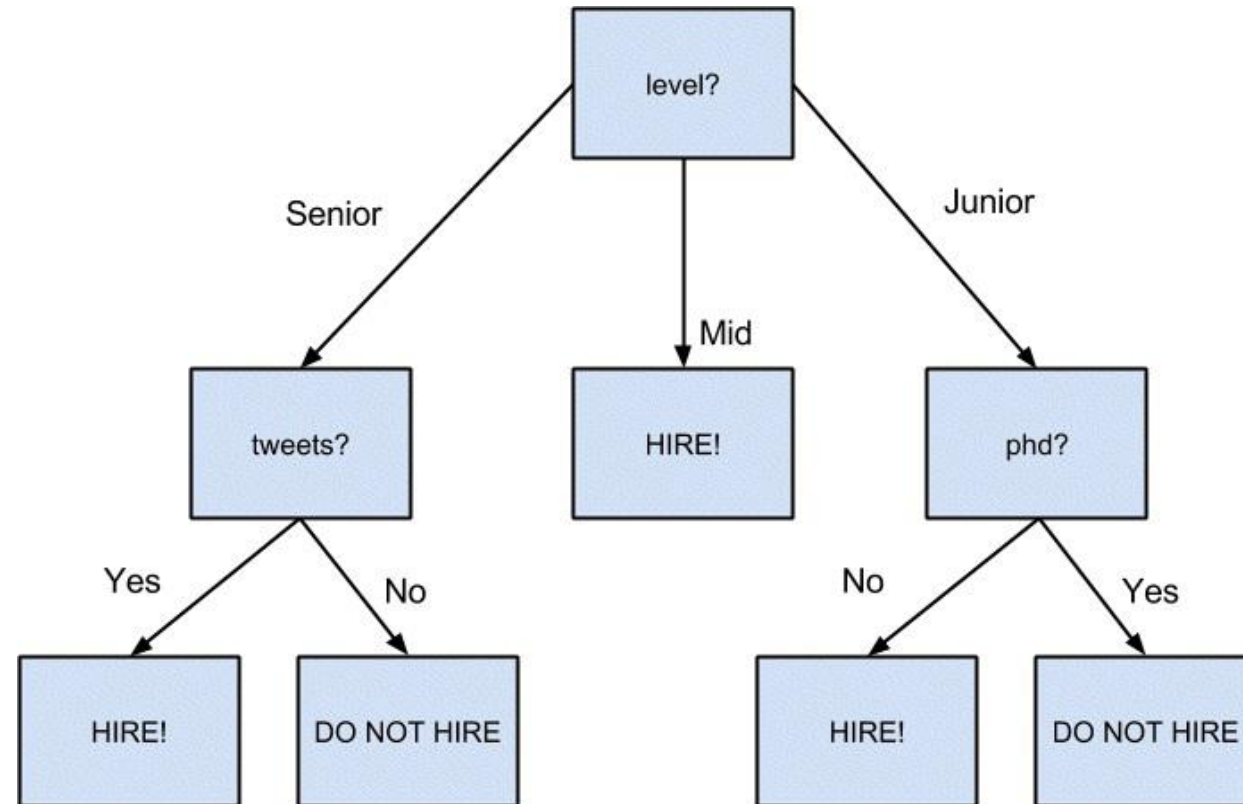
```
junior_inputs = [input for input in inputs if input.level == 'Junior']

print(partition_entropy_by(junior_inputs, 'lang', 'did_well'))
print( partition_entropy_by(junior_inputs, 'tweets', 'did_well'))
print(partition_entropy_by(junior_inputs, 'phd', 'did_well'))
```

0.9509775004326938 • '박사 학위 여부' 분할 엔트로피가 0
0.9509775004326938
0.0

Junior 데이터는 '박사 학위 여부' 기준으로 2차 분할을 한다.

최종 의사 결정 트리



5. 의사 결정 트리



리프/결정 노드 정의

리프 노드

```
from typing import NamedTuple, Union, Any

class Leaf(NamedTuple):
    value: Any
```

- 리프 노드는 분류될 클래스 값으로 정의됨 (클래스 타입은 Any로 어떤 타입이든 가능)

결정 노드

```
class Split(NamedTuple):
    attribute: str
    subtrees: dict
    default_value: Any = None
```

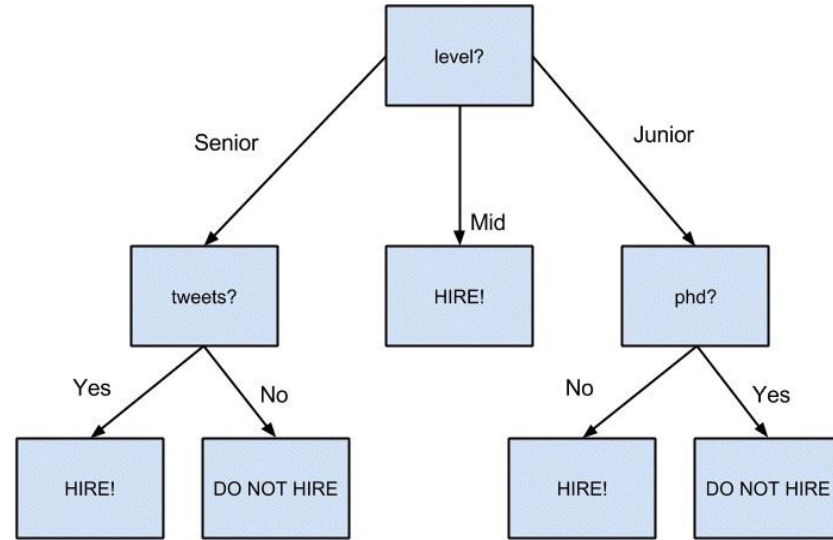
- Attribute : 분할 속성
- subtrees : 부트리를 구성하는 분할 속성의 값을 키로 하는 딕셔너리
- default_value : attribute 값의 범위에 없었던 값이 입력된 경우 반환

의사 결정 트리

```
DecisionTree = Union[Leaf, Split]
```

- 의사 결정 트리는 리프 노드 또는 결정 노드로 정의됨
- Union : 여러 타입을 허용하는 타입

트리 구성 예시



```
hiring_tree = Split('level', {
    'Junior': Split('phd', {
        False: Leaf(True),
        True: Leaf(False)
    }),
    'Mid': Leaf(True),
    'Senior': Split('tweets', {
        False: Leaf(False),
        True: Leaf(True)
    })
})
```

First, consider "level".
if level is "Junior", next look at "phd"
if "phd" is False, predict True
if "phd" is True, predict False

if level is "Mid", just predict True
if level is "Senior", look at "tweets"
if "tweets" is False, predict False
if "tweets" is True, predict True

Subtrees는
속성의 값을 key로하고
노드를 value로 하는
딕셔너리

결정 노드(Split)의
attribute subtrees

리프 노드(Leaf)의 value

클래스 분류

각 노드에서 노드에 지정된 속성(attribute)에 따라 같은 속성 값을 갖는 부트리로 분기

```
def classify(tree: DecisionTree, input: Any) -> Any:
    """classify the input using the given decision tree"""

    # If this is a leaf node, return its value
    if isinstance(tree, Leaf):
        return tree.value

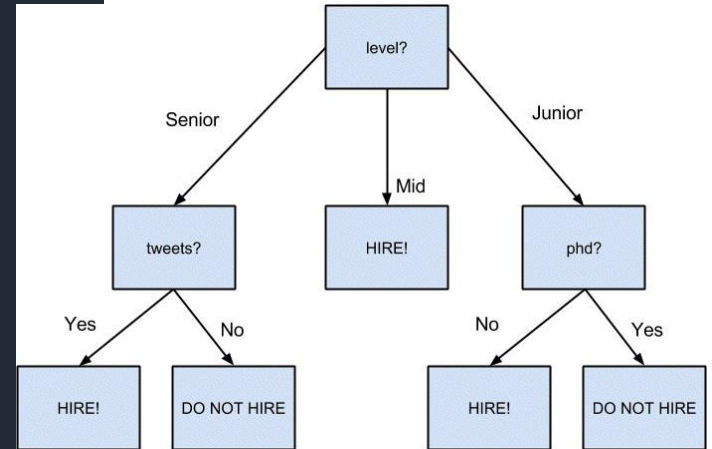
    # Otherwise this tree consists of an attribute to split on
    # and a dictionary whose keys are values of that attribute
    # and whose values are subtrees to consider next
    subtree_key = getattr(input, tree.attribute)

    if subtree_key not in tree.subtrees:
        return tree.default_value

    subtree = tree.subtrees[subtree_key]
    return classify(subtree, input)
```

리프 노드

결정 노드



- 리프 노드인 경우 : 클래스 반환
- 결정 노드인 경우 :
 - subtree_key : 노드의 분기 조건을 결정하는 속성(tree.attribute)에 해당하는 데이터(input)의 속성 값
 - 데이터의 속성 값이 분기 범위 밖에 있으면 default_value로 반환
 - 해당 속성 값에 해당하는 부트리로 분기해서 classify 함수 호출

build_tree_id3 의사 결정 트리 생성

ID3 알고리즘으로 의사 결정 트리 생성

```
def build_tree_id3(inputs: List[Any],  
                  split_attributes: List[str],  
                  target_attribute: str) -> DecisionTree:
```

- inputs : 데이터
- split_attributes : 분할 조건으로 사용할 속성 이름 목록
- target_attribute : 타겟 (레이블) 속성 이름

매개변수 list

list

build_tree_id3 의사 결정 트리 생성

각 노드에서 클래스 별로 데이터 개수를 세고 데이터가 가장 많은 클래스를 확인

(리프 노드의 value와 결정 노드의 default_value로 사용)

```
# Count target labels
label_counts = Counter(getattr(input, target_attribute)
                        for input in inputs)
most_common_label = label_counts.most_common(1)[0][0]
```

- label_counts : 데이터에서 레이블 별로 데이터 수를 셈
- most_common_label : 데이터 수가 가장 많은 레이블

label_counts : Counter({True: 9, False: 5})

label_counts.most_common(1) : [(True, 9)]

label_counts.most_common(1)[0][0]

가장 많은 레이블 찾기
가장 많은 레이블 찾기
가장 많은 레이블 찾기

build_tree_id3 리프 노드 생성

노드가 한 클래스로 되어 있거나 분할 조건에 사용할 속성이 없으면 리프 노드 생성

한 종류의 레이블(클래스)만 있다면 리프 노드 생성

```
# If there's a unique label, predict it
if len(label_counts) == 1:
    return Leaf(most_common_label)
```

- most_common_label을 value로 하는 리프 노드 생성

분할에 사용할 속성이 남아있지 않으면 리프 노드 생성

```
# If no split attributes left, return the majority label
if not split_attributes:
    return Leaf(most_common_label)
```

- most_common_label을 value로 하는 리프 노드 생성

build_tree_id3 데이터 분할

분할 엔트로피를 최소로 하는 속성을 구해서 데이터 분할

분할 엔트로피 계산 함수

```
# Otherwise split by the best attribute

def split_entropy(attribute: str) -> float:
    """Helper function for finding the best attribute"""
    return partition_entropy_by(inputs, attribute, target_attribute)
```

데이터 분할

```
best_attribute = min(split_attributes, key=split_entropy)
partitions = partition_by(inputs, best_attribute)
new_attributes = [a for a in split_attributes if a != best_attribute]
```

- best_attribute : 분할 엔트로피가 가장 작은 속성 선택
- partitions : 선택된 속성으로 데이터 데이터 분할
- new_attributes : 사용한 속성은 속성 목록에서 제거

build_tree_id3 결정 노드 생성

분할 별로 부트리 생성한 후 부트리의 부모인 결정 노드를 생성

각 분할 별로 부트리 생성

```
# recursively build the subtrees
subtrees = {attribute_value : build_tree_id3(subset,
                                              new_attributes,
                                              target_attribute)
            for attribute_value, subset in partitions.items()}
```

- 부트리는 key는 속성 값, value는 노드로 된 딕셔너리
- 부트리 노드를 생성하기 위해 각 분할에 대해 build_tree_id3를 재귀적으로 호출

결정 노드 생성

```
return Split(best_attribute, subtrees, default_value=most_common_label)
```

- default 값은 가장 빈도가 높은 레이블인 most_common_label로 지정

테스트

의사 결정 트리 모델 생성

```
tree = build_tree_id3(inputs,
                      ['level', 'lang', 'tweets', 'phd'],
                      'did_well')
```

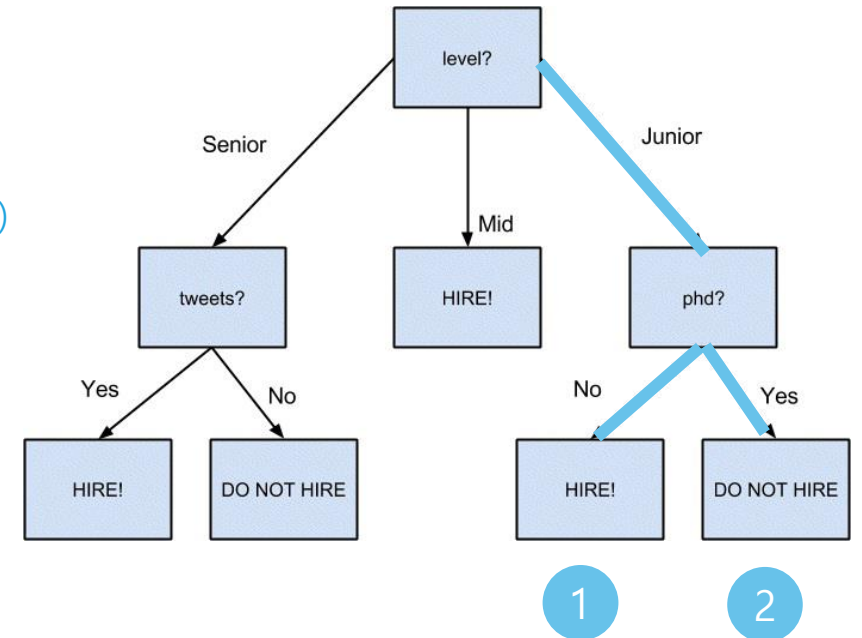
테스트 (직급, 선호하는 프로그래밍 언어, 트위터 사용, 박사 학위, 인터뷰 긍정 평가)

```
# Should predict True
assert classify(tree, Candidate("Junior", "Java", True, False))

# Should predict False
assert not classify(tree, Candidate("Junior", "Java", True, True))

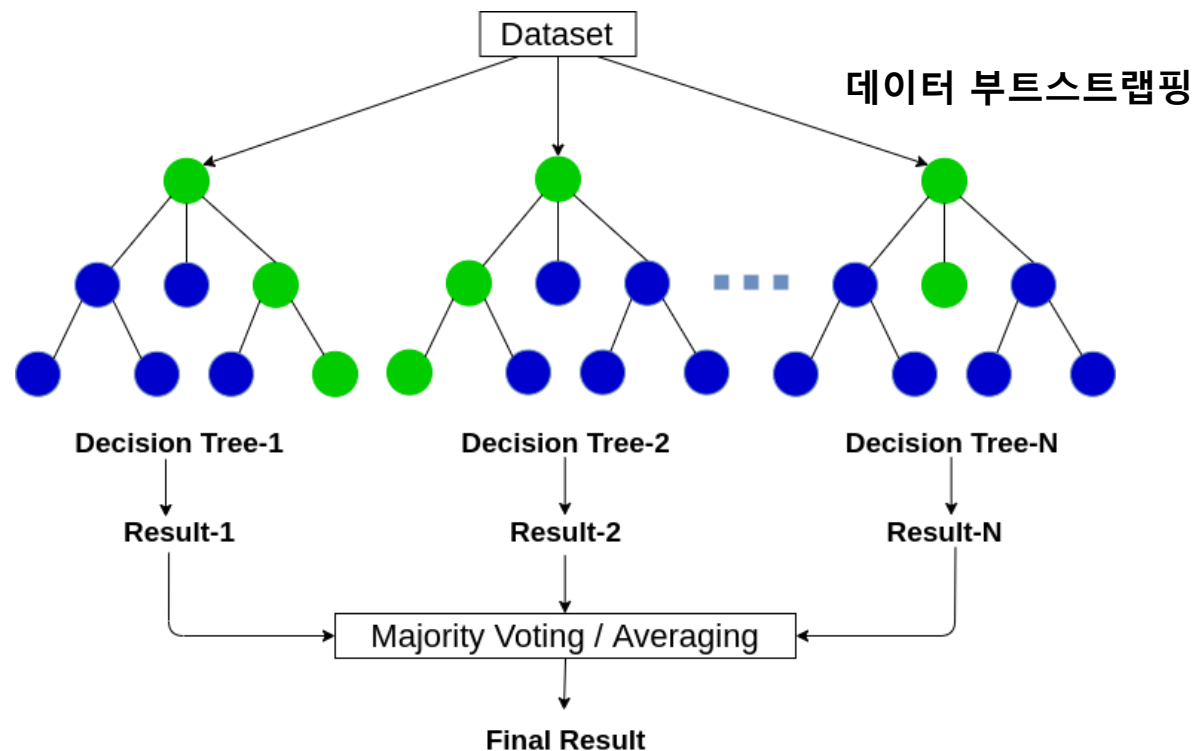
# Should predict True
assert classify(tree, Candidate("Intern", "Java", True, True))
```

- "Intern"은 루트 노드의 default 값인 빈도가 가장 높은 레이블인 True로 반환됨



정규화 랜덤 포레스트

의사 결정 트리를 동시에 여러 개를 실행해서 투표 또는 평균 방식으로 예측을 하는 방식



일종의 모델 앙상블 방식으로 여러 모델을 동시에 실행시킴으로써 편향을 제거하는 방법

정규화 속성의 랜덤 선택

속성을 랜덤하게 선택 한 후 그 중 최적의 속성을 선택하는 방식

```
# if there's already few enough split candidates, look at all of them
if len(split_candidates) <= self.num_split_candidates:
    sampled_split_candidates = split_candidates
# otherwise pick a random sample
else:
    sampled_split_candidates = random.sample(split_candidates,
                                             self.num_split_candidates)
# now choose the best attribute only from those candidates
best_attribute = min(sampled_split_candidates, key=split_entropy)
partitions = partition_by(inputs, best_attribute)
```

일종의 모델 앙상블 학습 방식

6. scikit-learn 활용

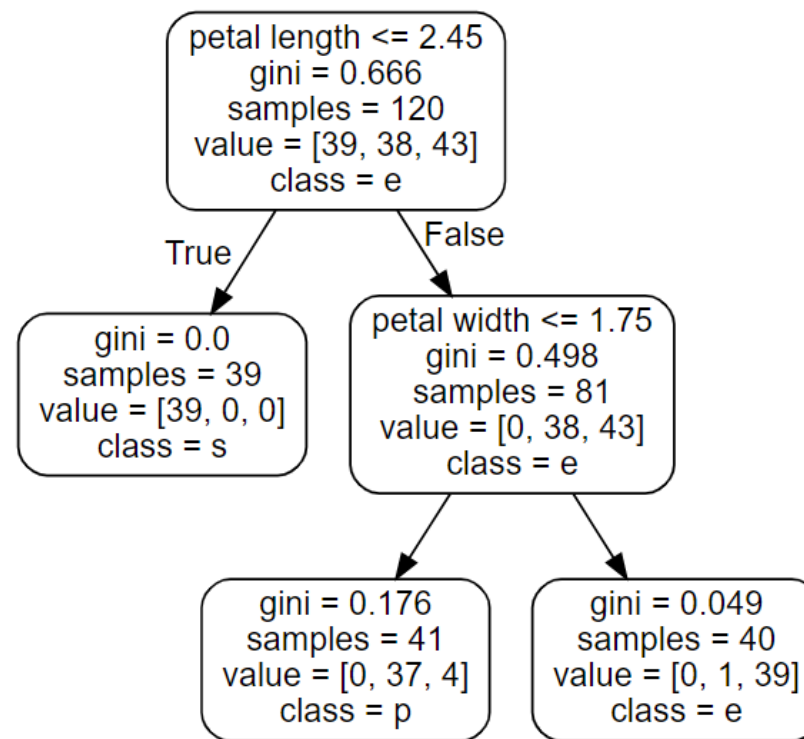


붓꽃 분류

머신러닝 패키지인 `scikit-learn`을 이용해서 의사 결정 트리로 세가지 붓꽃을 분류해보자.

kNN

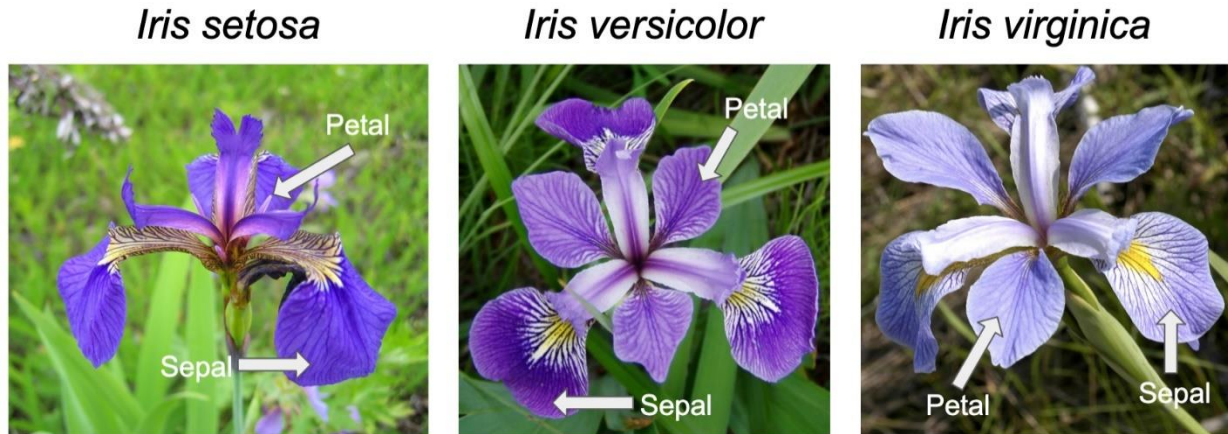
정확도 97.7%가 나옴



정확도 97%

붓꽃 데이터셋 (Iris dataset)

세가지 붓꽃 종에 대한 데이터셋



- 150개 (각 종별로 50개씩)
- 1936년 영국 통계학자이자 생물학자인 도널드 피셔 ([Ronald Fisher](#))의 논문에서 사용됨

속성	설명	타입
sepal length (cm)	꽃받침 길이	continuous
sepal width (cm)	꽃받침 폭	continuous
petal length (cm)	꽃잎 길이	continuous
petal width (cm)	꽃잎 폭	continuous
target	붓꽃 종류 • Iris Setosa • Iris Versicolour • Iris Virginica	multi-valued discrete

<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/>

패키지 설치

scikit-learn 설치

```
!pip install sklearn
```

Graphviz 설치

```
!pip install graphviz
```

- 의사 결정 트리를 출력하기 위해 시각화 툴인 Graphviz 설치
- 바이너리 설치 후 패키지 설치 (<https://graphviz.org/download/>)

패키지 импорт

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import learning_curve, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import graphviz
%matplotlib inline
```

데이터셋 다운로드

데이터 다운로드


```
import requests
import os

dataset_path = os.path.join('data', 'iris.data')
if os.path.exists(dataset_path) is False:
    data = requests.get("https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data")
    with open(dataset_path, "w") as f:
        f.write(data.text)
```

- URL에서 데이터를 다운로드해서 "iris.dat" 파일에 저장

데이터셋 읽기

데이터 읽기



```
import pandas as pd

column_names = ['sepal length', 'sepal width', 'petal length', 'petal width',
                'species']
class_names = ['Iris-setosa', 'Iris-virginica', 'Iris-versicolor']
dataset = pd.read_csv(dataset_path, names=column_names)
dataset.sample(5)
```

- csv 파일에 column 이름이 없으므로 이름을 지정해서 읽어 옴

	sepal length	sepal width	petal length	petal width	species
133	6.3	2.8	5.1	1.5	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
25	5.0	3.0	1.6	0.2	Iris-setosa
149	5.9	3.0	5.1	1.8	Iris-virginica
115	6.4	3.2	5.3	2.3	Iris-virginica

레이블 분리 및 데이터셋 분할

레이블 분리



```
X = dataset[dataset.columns[:-1]]
dataset['target'] = (dataset['species']=='Iris-setosa')*0 + \
                    (dataset['species']=='Iris-virginica')*1 + \
                    (dataset['species']=='Iris-versicolor')*2
y = dataset['target']
```

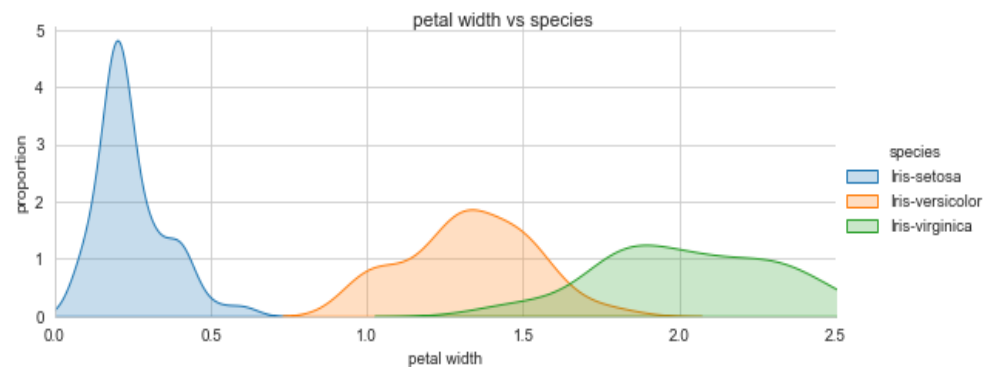
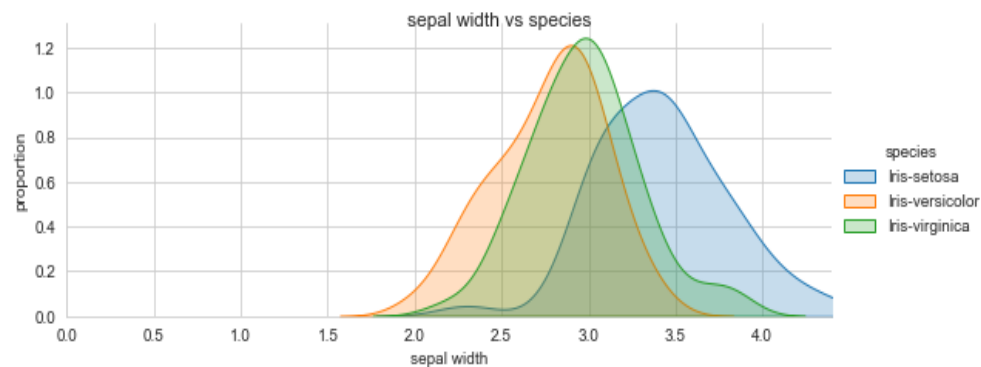
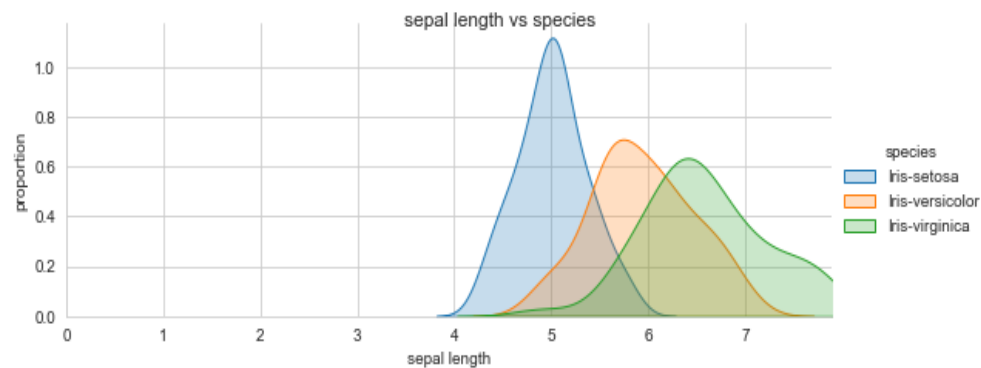
- 문자열로 되어 있는 Species를 숫자로 된 target으로 변환

데이터셋 분할

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
```

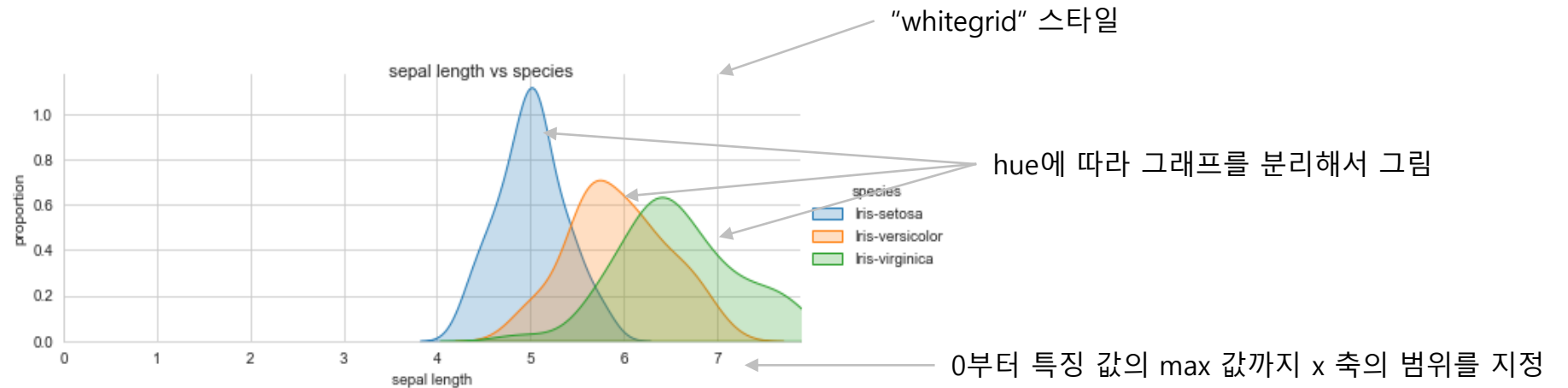
데이터 탐색 특징 별 분포

특징의 분포가 클래스 별로 잘 구분되는지 확인



```
for feature_name in X.keys():  
    plot_feature_by_label(dataset, feature_name, 'species', feature_name + ' vs species')
```

데이터 탐색 특징 별 분포



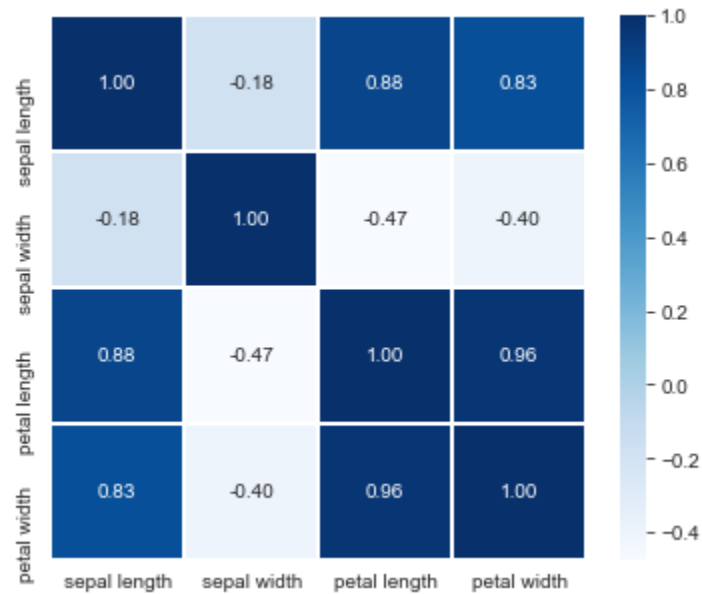
지정된 특징의 분포를 클래스 별로 분류해서 그림

```
def plot_feature_by_label(dataframe, feature_name, label_name, title):  
    sns.set_style("whitegrid")  
    ax = sns.FacetGrid(dataframe, hue=label_name, aspect=2.5)  
    ax.map(sns.kdeplot, feature_name, shade=True)  
    ax.set(xlim=(0, dataframe[feature_name].max()))  
    ax.add_legend()  
    ax.set_axis_labels(feature_name, 'proportion')  
    ax.fig.suptitle(title)  
    plt.show()
```

- `sns.FacetGrid`: 데이터셋을 row, column 변수 값에 따라 배열 형태로 그래프를 그리는 함수, hue로 지정된 값에 따라 별도의 그래프를 그려 줌
- `ax.map`: dataframe의 변수를 지정된 그래프 형태로 그림

차원 축소 전체 특징

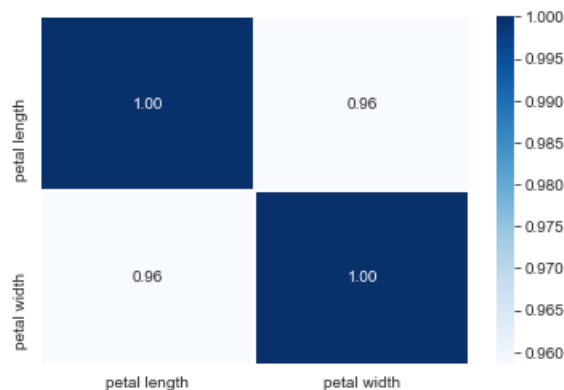
전체 특징에는 특징 간에 상관성 존재하는 상태



```
fig, ax = plt.subplots(figsize=(6, 5))
sns.heatmap(X_train.corr(), linewidths=.5, annot=True, fmt=".2f", cmap='Blues')
```

차원 축소 특징 선택

클래스를 변별력이 낮은 'sepal length', 'sepal width' 를 제거



변수 간에 상관성이 낮은 상태

변수 제거

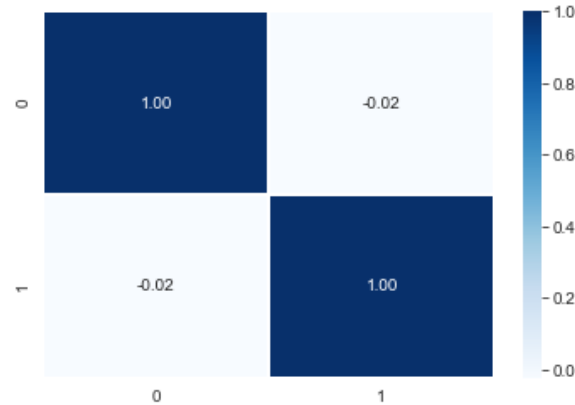
```
dataset2 = dataset.drop(['sepal length', 'sepal width'], axis=1)
```

데이터 분할/히트맵

```
X2 = dataset2[dataset2.columns[:-2]]  
y2 = dataset2.target  
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X2, y2, test_size=0.2)  
sns.heatmap(X_train2.corr(), linewidths=.5, annot=True, fmt=".2f", cmap='Blues')
```

차원 축소 특징 추출 (PCA)

PCA로 특징을 추출해서 2개의 특징만 남김



변수 간에 상관성이 낮은 상태

- PCA의 경우 특징 이름이 숫자로 생성됨

PCA 실행

```
from sklearn.decomposition import PCA
X3,y3 = X,y
variance_pct = 2
pca = PCA(n_components=variance_pct) # Create PCA object
X_transformed = pca.fit_transform(X3,y3) # Transform the initial features
```

DataFrame 생성/데이터 분할/히트맵

```
X3pca = pd.DataFrame(X_transformed)
X_train3, X_test3, Y_train3, Y_test3 = train_test_split(X3pca, y3, test_size=0.2)
sns.heatmap(X_train3.corr(), linewidths=.5, annot=True, fmt=".2f", cmap='Blues')
```

모델 훈련

세 모델을 의사 결정 트리로 훈련시켜서 성능을 확인해보자.

(단, 트리의 깊이는 2로 제한)

```
clf1 = tree.DecisionTreeClassifier(max_depth=2,min_samples_leaf=12)
clf1.fit(X_train, Y_train)
clf2 = tree.DecisionTreeClassifier(max_depth=2,min_samples_leaf=12)
clf2.fit(X_train2, Y_train2)
clf3 = tree.DecisionTreeClassifier(max_depth=2,min_samples_leaf=12)
clf3.fit(X_train3, Y_train3)
```

모델 훈련 및 성능 비교

```
print('Accuracy of Decision Tree classifier on original training set: {:.2f}'.format(clf1.score(X_train, Y_train)))
print('Accuracy of Decision Tree classifier on original test set: {:.2f}'.format(clf1.score(X_test, Y_test)))
print('Accuracy of Decision Tree classifier on reduced training set: {:.2f}'.format(clf2.score(X_train2, Y_train2)))
print('Accuracy of Decision Tree classifier on reduced test set: {:.2f}'.format(clf2.score(X_test2, Y_test2)))
print('Accuracy of Decision Tree classifier on PCA-
transformed training set: {:.2f}'.format(clf3.score(X_train3, Y_train3)))
print('Accuracy of Decision Tree classifier on PCA-transformed test set: {:.2f}'.format(clf3.score(X_test3, Y_test3)))
```

Accuracy of Decision Tree classifier on original training set: 0.97

Accuracy of Decision Tree classifier on original test set: **0.93**

Accuracy of Decision Tree classifier on reduced training set: 0.96

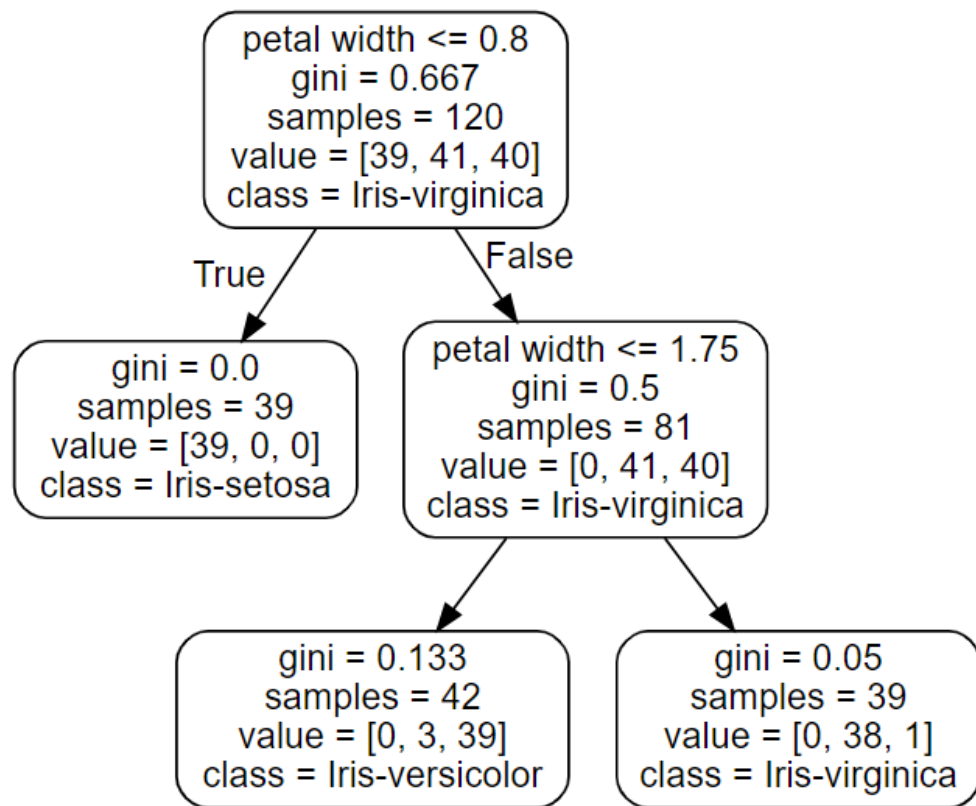
Accuracy of Decision Tree classifier on reduced test set: **0.93**

Accuracy of Decision Tree classifier on PCA-transformed training set: 0.93

Accuracy of Decision Tree classifier on PCA-transformed test set: **1.00**

특징 선택을 한 경우 97%의 정확도를 갖게 됨

의사 결정 트리 전체 특징

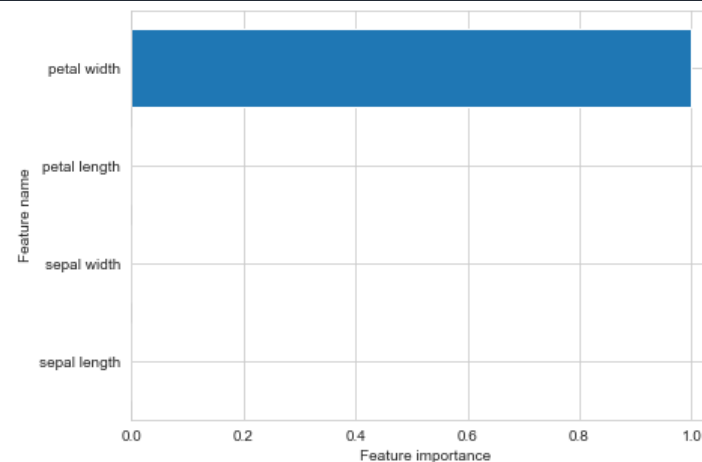


전체 특징으로 훈련했을 때 petal_width만 사용

```
plot_decision_tree(clf1, feature_names1, class_names)
```

주요 변수

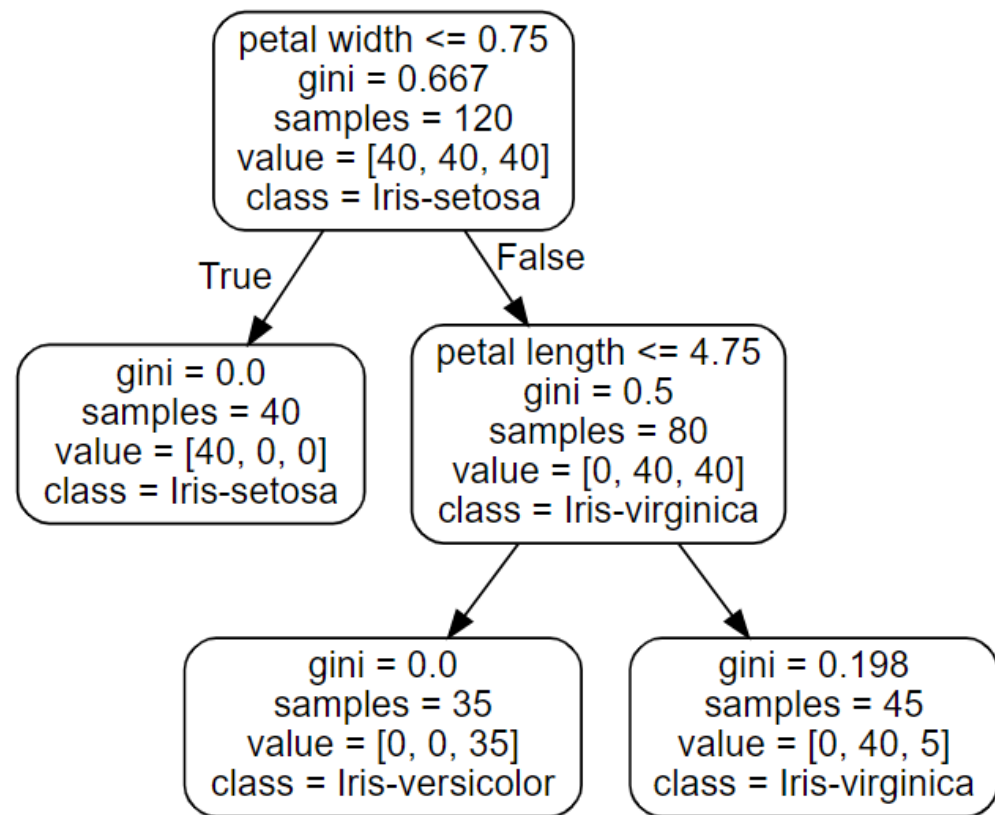
```
fig, ax = plt.subplots(figsize=(7, 5))  
plot_feature_importances(clf1, feature_names1)
```



Accuracy of Decision Tree classifier on original training set: 0.97

Accuracy of Decision Tree classifier on original test set: 0.93

의사 결정 트리 특징 선택

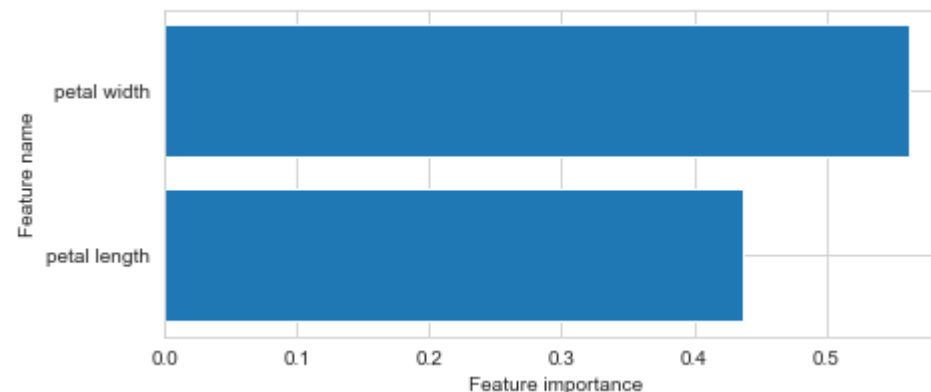


특징 선택을 했을 때는 꽃잎의 길이와 폭이
모두 활용되었음을 알 수 있음

```
plot_decision_tree(clf2, feature_names2, class_names)
```

주요 변수

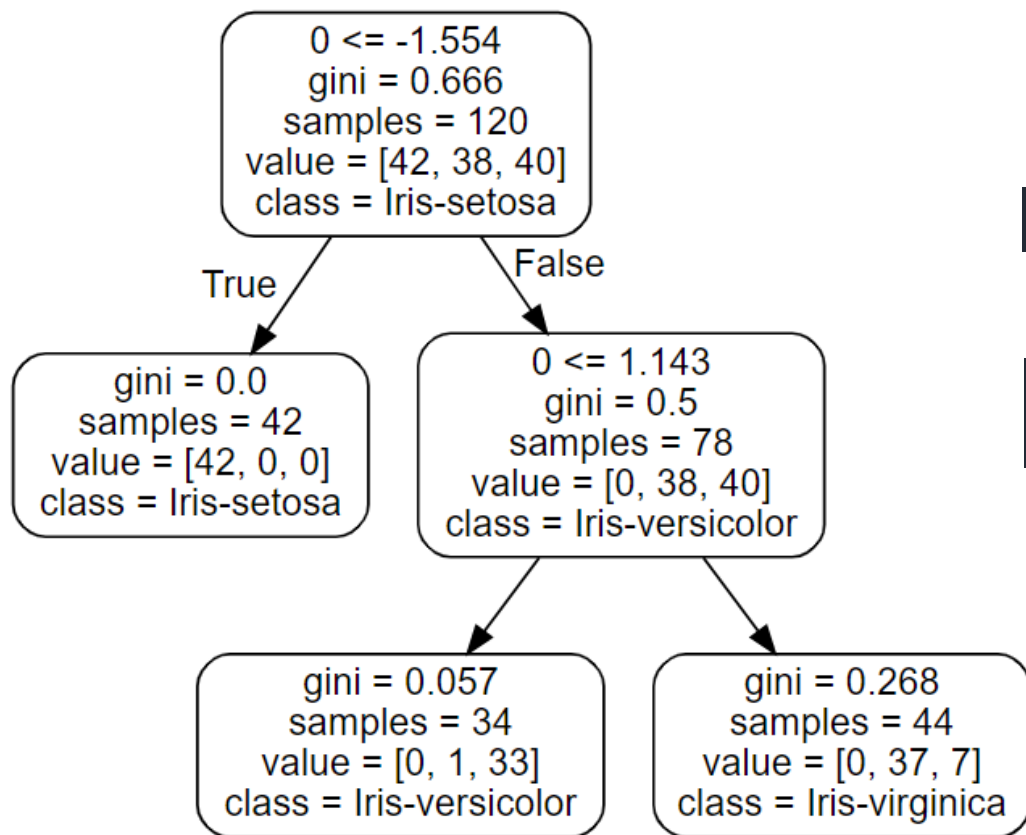
```
fig, ax = plt.subplots(figsize=(7, 3))  
plot_feature_importances(clf2, feature_names2)
```



Accuracy of Decision Tree classifier on reduced training set: 0.96

Accuracy of Decision Tree classifier on reduced test set: 0.93

의사 결정 트리 특징 추출

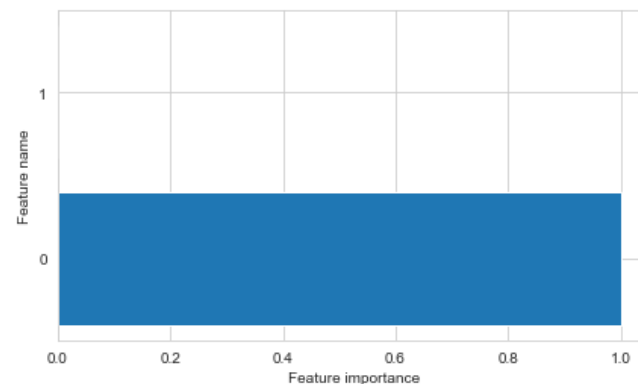


특징 추출을 했을 때 하나의 변수만 사용

```
plot_decision_tree(clf3, feature_names3, class_names)
```

주요 변수

```
fig, ax = plt.subplots(figsize=(7, 4))  
plot_feature_importances(clf3, feature_names3)
```



Accuracy of Decision Tree classifier on PCA-transformed training set: 0.93

Accuracy of Decision Tree classifier on PCA-transformed test set: **1.00**

의사 결정 트리 시각화

세 모델의 특징 이름을 추출

```
feature_names1 = X.columns.values  
feature_names2 = X2.columns.values  
feature_names3 = X3pca.columns.values # [0 1 2 3 4]
```

- PCA의 경우 특징 이름이 숫자로 생성됨

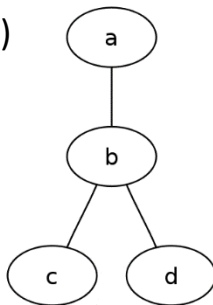
의사 결정 트리 그래프

의사 결정 트리를 DOT 포맷으로 추출해서 GraphBiz로 렌더링

```
def plot_decision_tree(tree_clf, feature_names, target_names):  
    dot_data = tree.export_graphviz(  
        tree_clf,  
        out_file=None,  
        feature_names=feature_names,  
        class_names=target_names,  
        filled=False,  
        rounded=True,  
        special_characters=False)  
    graph = graphviz.Source(dot_data)  
    return graph
```

DOT (graph description language)

```
graph graphname {  
    a -- b -- c;  
    b -- d;  
}
```



- `export_graphviz` : 의사 결정 트리를 DOT 포맷으로 추출
- `graphviz.Source` : DOT 소스 코드를 Graphviz로 렌더링

주요 특징 그래프

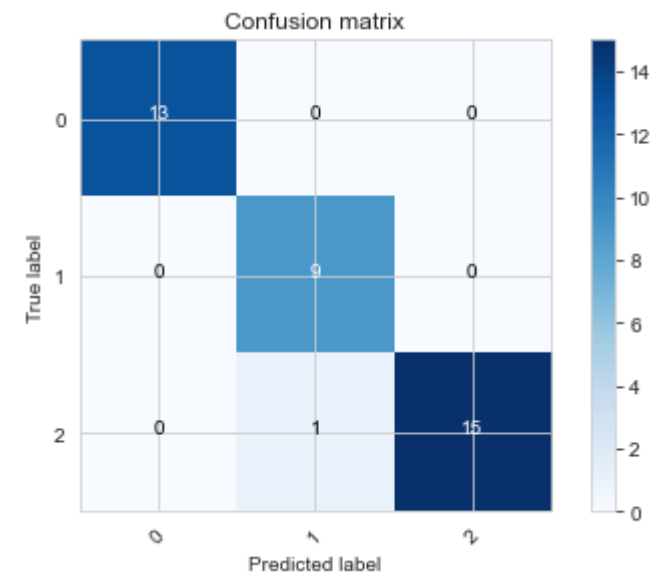
의사 결정 트리의 불순도(impurity)를 낮추는 변수를 주요 변수로 표현

```
def plot_feature_importances(clf, feature_names):  
    c_features = len(feature_names)  
    plt.barh(range(c_features), clf.feature_importances_)  
    plt.xlabel("Feature importance")  
    plt.ylabel("Feature name")  
    plt.yticks(np.arange(c_features), feature_names)  
    plt.show()
```

- `clf.feature_importances_` : 불순도(impurity) 누적 감소량 평균과 표준 편차로 계산됨

랜덤 포레스트

랜덤 포레스트 방식으로 훈련해 보자!



정확도 97%로 IRIS의 경우 특징이 많지 않기 때문에 성능이 개선되지 않음

모델 훈련 및 성능

모델 훈련

```
(X1, y1) = (X, y)
X_train1, X_test1, Y_train1, Y_test1 = train_test_split(X1, y1, random_state=0)
clf = RandomForestClassifier(max_features=4, random_state=0)
clf.fit(X_train1, Y_train1)
```

- 트리의 개수(`n_estimators`)는 default로 100개 생성
- `max_features` : 결정 노드에서 최대 몇 개의 속성을 이용해서 결정할 것인지 지정

모델 성능

```
print('Accuracy of Random Forest Classifier on training data: {:.2f}'.format(c  
lf.score(X_train1, Y_train1)))  
print('Accuracy of Random Forest Classifier on testing data: {:.2f}'.format(c  
lf.score(X_test1, Y_test1)))
```

Accuracy of Random Forest Classifier on training data: 1.00

Accuracy of Random Forest Classifier on testing data: 0.97

모델 평가

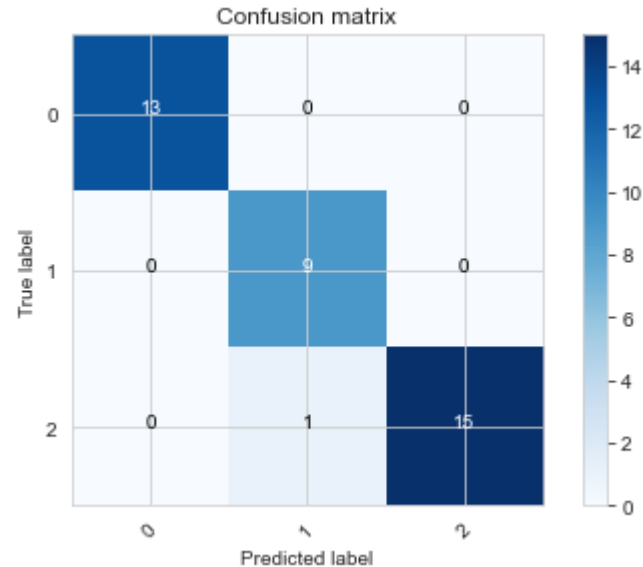
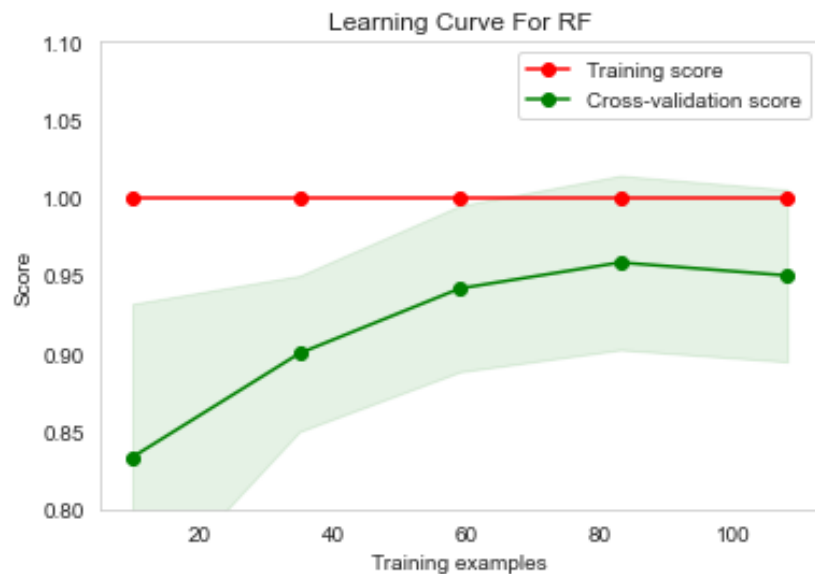
모델 평가

```
model = clf  
prediction = model.predict(X_test1)  
cnf_matrix = confusion_matrix(Y_test1, prediction)
```

모델 훈련 그래프 및 혼동 행렬

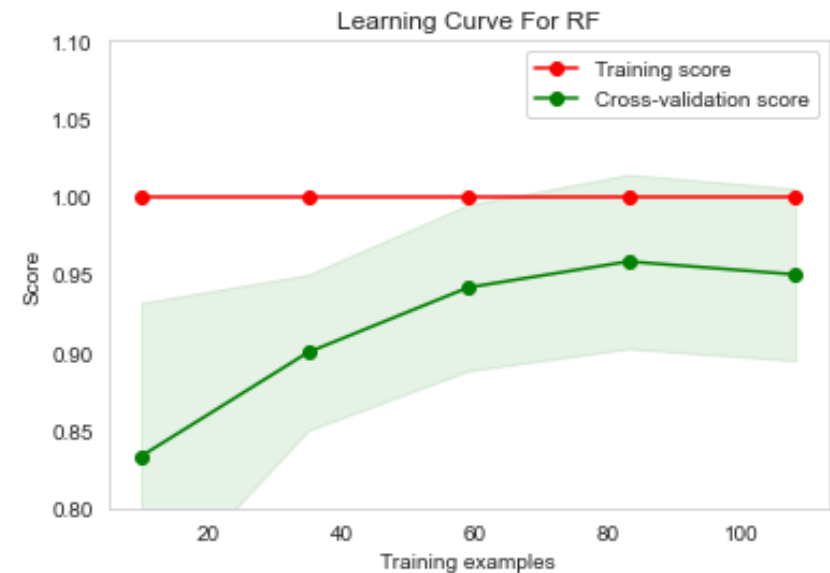
모델 훈련 그래프 및 혼동 행렬

```
plot_learning_curve(model, 'Learning Curve For RF', X_train, Y_train, (0.80,1.1), 10)
plt.show()
plot_confusion_matrix(cnf_matrix, classes=dict_characters, title='Confusion matrix')
plt.show()
```



학습 곡선 (Learning Curve)

훈련 데이터셋 크기 별로 교차 검증 (cross-validation)을
해서 훈련 및 테스트 성능을 반환



학습 곡선 데이터 생성

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
```

```
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
```


학습 곡선 (Learning Curve)

학습 곡선 데이터 그리기

```
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Cross-validation score")
plt.legend(loc="best")
return plt
```

평균, 표준편차

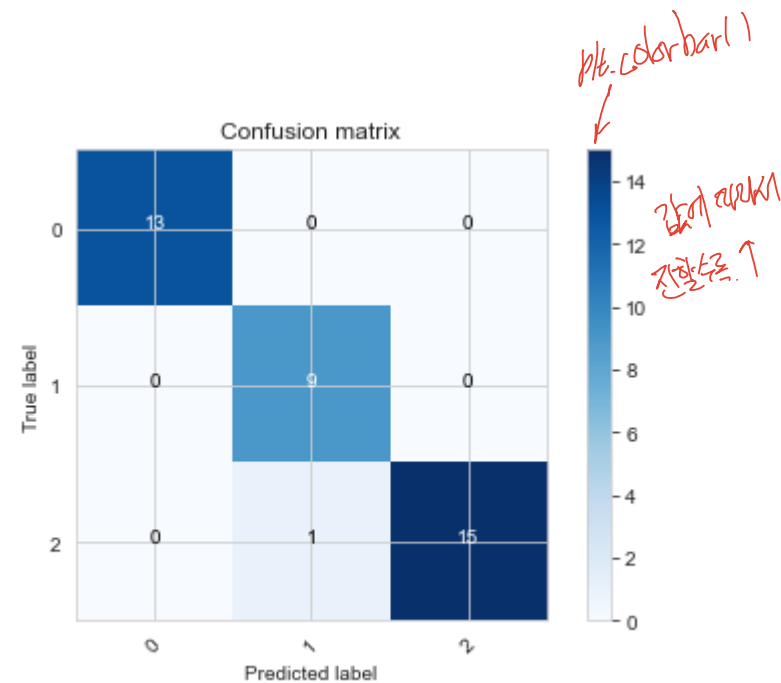
여기다 넣을

훈련

테스트

혼동 행렬

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```



- *tight_layout*: 서브 플롯이 그림(figure) 영역에 잘 들어가도록 파라미터를 자동으로 조정

Thank you!

