

train / test data의 분리

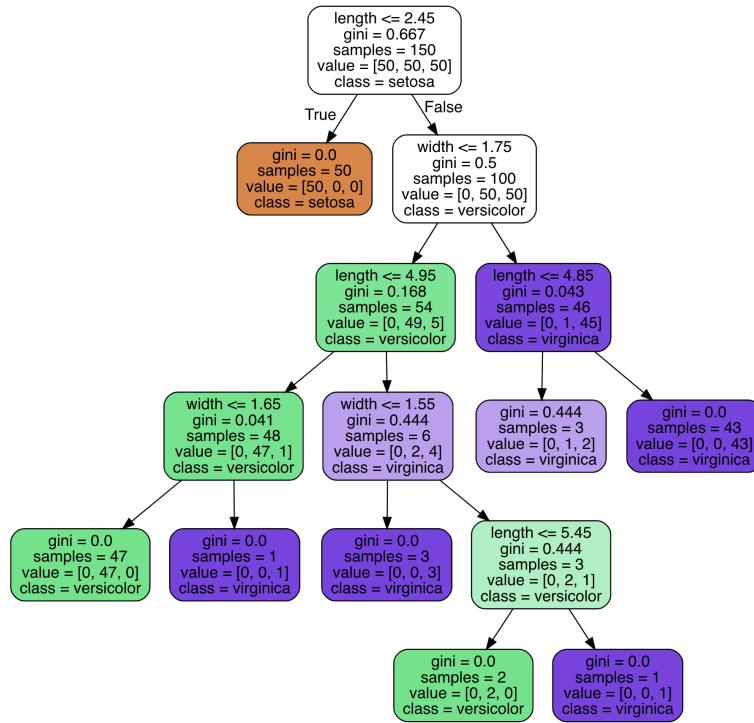
PinkLAB Edu

29 October, 2025

Table of Contents

1	Tree model visualization	3
2	mlxtend 설치	4
3	iris의 품종을 분류하는 결정나무 모델이 어떻게 데이터를 분류했는지 확인해보자	5
4	결정 경계를 확인해보자	6
5	머신러닝의 일반적 절차에 대해	7
5.1	Accuracy가 높다고? 믿을 수 있을까?	7
5.2	지도 학습	8
5.3	데이터의 분리 (훈련 / 검증 / 평가)	8
5.4	새롭게 다시 시작	9
5.5	데이터를 훈련 / 테스트로 분리	9
5.6	훈련용 / 테스트용이 잘 분리 되었을까?	9
5.7	그럴 때 쓰는 옵션 stratify	10
5.8	방금 전처럼 다시 train 데이터만 대상으로 결정나무 모델을 만들어 보자	10
5.9	train 데이터에 대한 accuracy 확인	11
5.10	iris 꽃을 분류하는 결정나무 모델	11
5.11	훈련데이터에 대한 결정경계를 확인해보자	12
5.12	테스트 데이터에 대한 accuracy	12
5.13	결과	13
5.14	잔기술 하나 - 전체 데이터에서 관찰해보기	14
5.15	결과	14
5.16	feature를 네 개	15
5.17	전체 특성을 사용한 결정나무 모델	16
5.18	모델을 사용하는 방법은?	16
5.19	주요 특성 확인하기	17
5.20	간단한 zip과 언패킹~	18
5.20.1	리스트를 튜플로 zip	18
5.20.2	튜플을 dict으로	18
5.20.3	한번에~	18
5.20.4	unpacking 인자를 이용한 역변환	18

1 Tree model visualization



- 이 모델이 이전에 엄청 accuracy가 좋았던 모델이다

2 mlxtend 설치

- (ml) jt@ubuntu:~/dev_ws\$ pip3 install mlxtend
Collecting mlxtend
 Downloading mlxtend-0.23.4-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: scipy>=1.2.1 in /home/jt/venv/ml/lib/python3.12/site-packages (from mlxtend) (1.15.1)
Requirement already satisfied: numpy>=1.16.2 in /home/jt/venv/ml/lib/python3.12/site-packages (from mlxtend) (2.2.2)
Requirement already satisfied: pandas>=0.24.2 in /home/jt/venv/ml/lib/python3.12/site-packages (from mlxtend) (2.2.3)
Requirement already satisfied: scikit-learn>=1.3.1 in /home/jt/venv/ml/lib/python
• sklearn에 없는 몇몇 유용한 기능을 가지고 있다

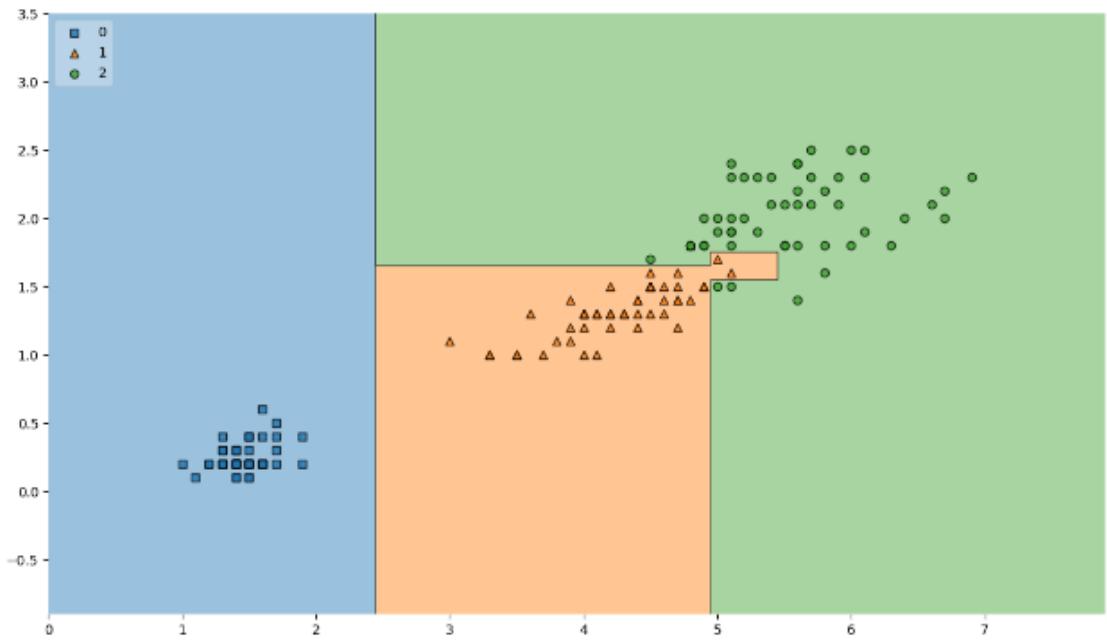
3 iris의 품종을 분류하는 결정나무 모델이 어떻게 데이터를 분류했는지 확인해보자

```
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 8))
plot_decision_regions(X=iris.data[:, 2:], y=iris.target,
                      clf=iris_tree, legend=2)
plt.show()
```

Python

4 결정 경계를 확인해보자



- 우리가 여기서 느껴야 할 것은?

5 머신러닝의 일반적 절차에 대해...

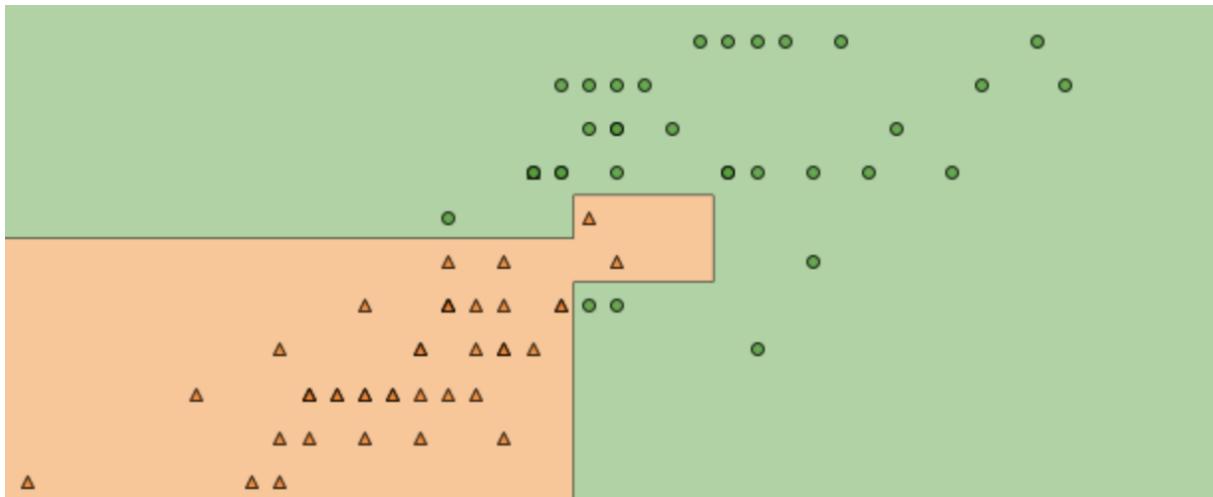
5.1 Accuracy가 높다고? 믿을 수 있을까?

```
from sklearn.metrics import accuracy_score
y_pred_tr = iris_tree.predict(iris.data[:, 2:])
accuracy_score(iris.target, y_pred_tr)
```

Python

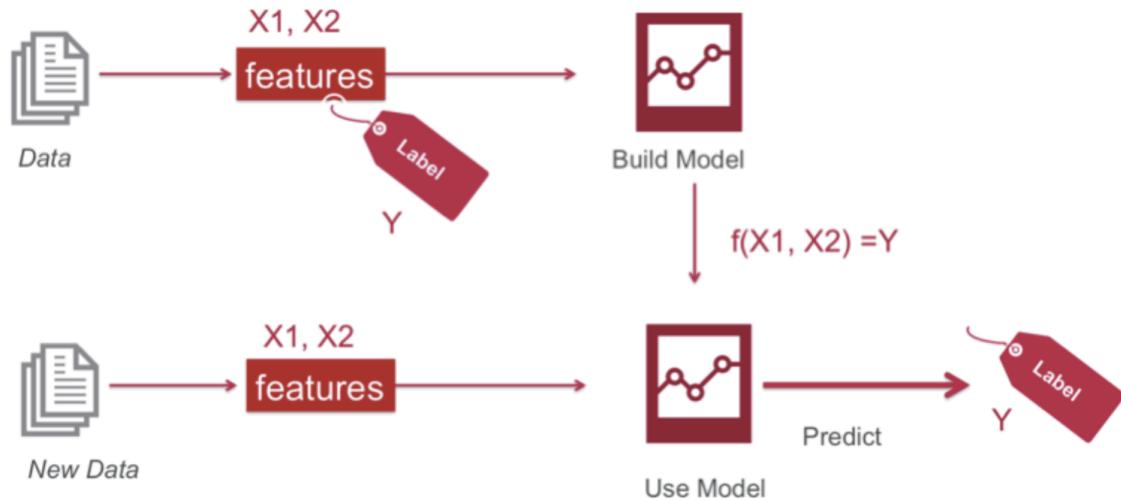
0.9933333333333333

- Acc가 높게 나왔다고 해도 좀 더 들여다 볼 필요가 당연히 있다



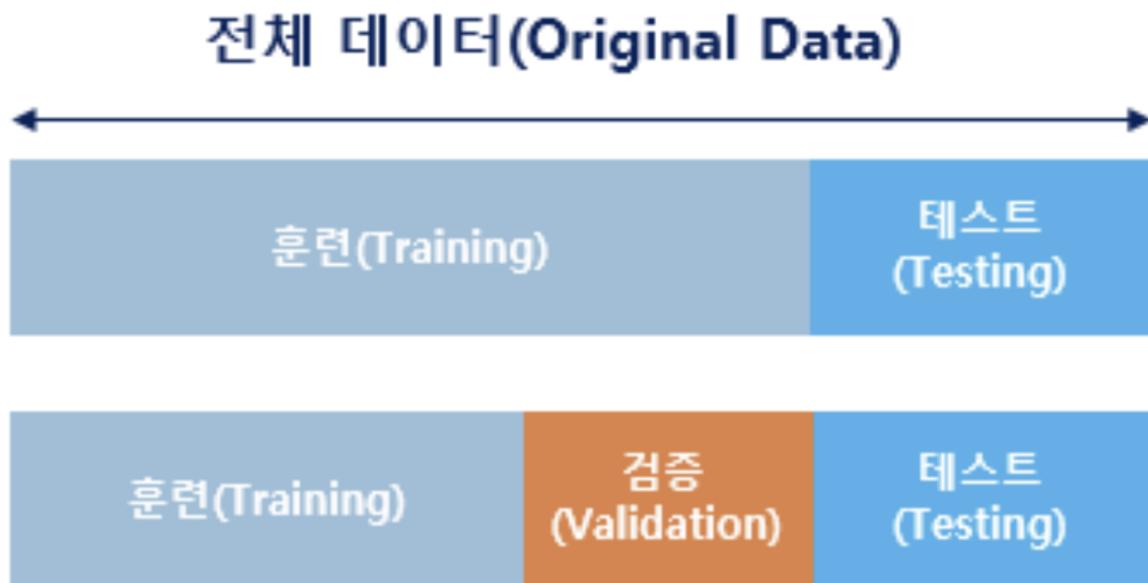
- 저 경계면은 올바른 걸까?
- 저 결과는 내가 가진 데이터를 벗어나서 일반화할 수 있는 걸까?
- 어차피 얻은(혹은 구한) 데이터는 유한하고 내가 얻은 데이터를 이용해서 일반화를 추구하게 된다.
- 이때 복잡한 경계면은 모델의 성능을 결국 나쁘게 만든다.

5.2 지도 학습



- 학습 대상이 되는 데이터에 정답(label)을 붙여서 학습 시키고
- 모델을 얻어서 완전히 새로운 데이터에 모델을 사용해서 “답”을 얻고자 하는 것

5.3 데이터의 분리 (훈련 / 검증 / 평가)



- 확보한 데이터 중에서 모델 학습에 사용하지 않고 빼둔 데이터를 가지고 모델을 테스트한다.

5.4 새롭게 다시 시작

```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
✓ 0.0s
```

Python

5.5 데이터를 훈련 / 테스트로 분리

```
from sklearn.model_selection import train_test_split

features = iris.data[:, 2:]
labels = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    features, labels, test_size=0.2, random_state=13
)
✓ 0.0s
```

Python

- 8:2 확률로 특성(features)과 정답(labels)를 분리해서

5.6 훈련용 / 테스트용이 잘 분리 되었을까?

```
import numpy as np

np.unique(y_test, return_counts=True)
✓ 0.0s
```

Python

```
(array([0, 1, 2]), array([ 9,  8, 13]))
```

- 문제가 각 클래스(setosa, versicolor, virginica) 별로 동일 비율이 아니다

5.7 그럴 때 쓰는 옵션 stratify

```
from sklearn.model_selection import train_test_split

features = iris.data[:, 2:]
labels = iris.target

X_train, X_test, y_train, y_test = train_test_split(features, labels,
                                                    test_size=0.2,
                                                    stratify = labels,
                                                    random_state = 13)

✓ 0.0s Python

import numpy as np

np.unique(y_test, return_counts = True)
✓ 0.0s Python

(array([0, 1, 2]), array([10, 10, 10]))
```

5.8 방금 전처럼 다시 train 데이터만 대상으로 결정나무 모델을 만들어 보자

```
from sklearn.tree import DecisionTreeClassifier

iris_tree = DecisionTreeClassifier(max_depth=2, random_state=13)
iris_tree.fit(X_train, y_train)

✓ 0.0s Python
```

DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=13)

- 학습할 때마다 일관성을 위해 random_state만 고정해서~
- 모델을 단순화시키기 위해 max_depth를 조정

5.9 train 데이터에 대한 accuracy 확인

```
from sklearn.metrics import accuracy_score
y_pred_tr = iris_tree.predict(X_train)
accuracy_score(y_train, y_pred_tr)
```

✓ 0.0s

Python

0.95

- iris 데이터는 단순해서 acc가 높게 나타남

5.10 iris 꽃을 분류하는 결정나무 모델

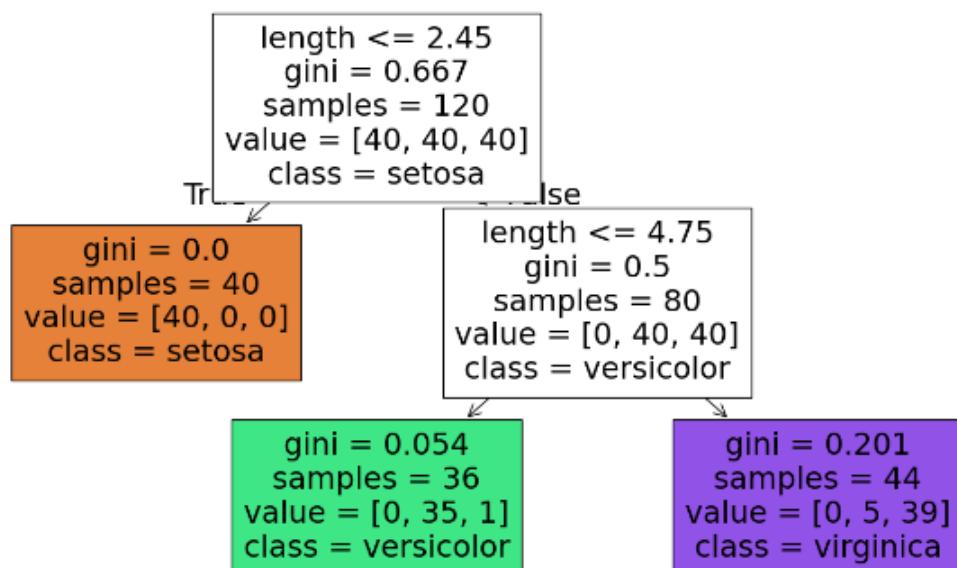
```
import matplotlib.pyplot as plt
from sklearn import tree

fig = plt.figure(figsize=(15, 8))
= tree.plot_tree(
    iris_tree,
    feature_names=["length", "width"],
    class_names=list(iris.target_names),
    filled=True,
)

```

✓ 0.2s

Python



- 이 안에 표현된 값들을 이제 이해할 수 있겠지...

5.11 훈련데이터에 대한 결정경계를 확인해보자

```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions

plt.figure(figsize=(12, 8))
plot_decision_regions(X=X_train, y=y_train, clf=iris_tree, legend=2)
plt.show()
```

✓ 0.3s Python



5.12 테스트 데이터에 대한 accuracy

```
y_pred_test = iris_tree.predict(X_test)
accuracy_score(y_test, y_pred_test)
```

✓ 0.0s Python

0.9666666666666667

5.13 결과

```
plt.figure(figsize=(12, 8))
plot_decision_regions(X=X_test, y=y_test, clf=iris_tree, legend=2)
plt.show()
```

✓ 0.2s

Python



5.14 잔기술 하나 - 전체 데이터에서 관찰해보기

```
scatter_highlight_kwarg = {"s": 150, "label": "Test data", "alpha": 0.9}
scatter_kwarg = {"s": 120, "edgecolor": None, "alpha": 0.7}

plt.figure(figsize=(12, 8))
plot_decision_regions(
    X=features,
    y=labels,
    X_highlight=X_test,
    clf=iris_tree,
    legend=2,
    scatter_highlight_kwarg=scatter_highlight_kwarg,
    scatter_kwarg=scatter_kwarg,
    contourf_kwarg={"alpha": 0.2},
)
plt.show()
```

✓ 0.2s

Python

5.15 결과



5.16 feature를 네 개..

```
features = iris.data
labels = iris.target

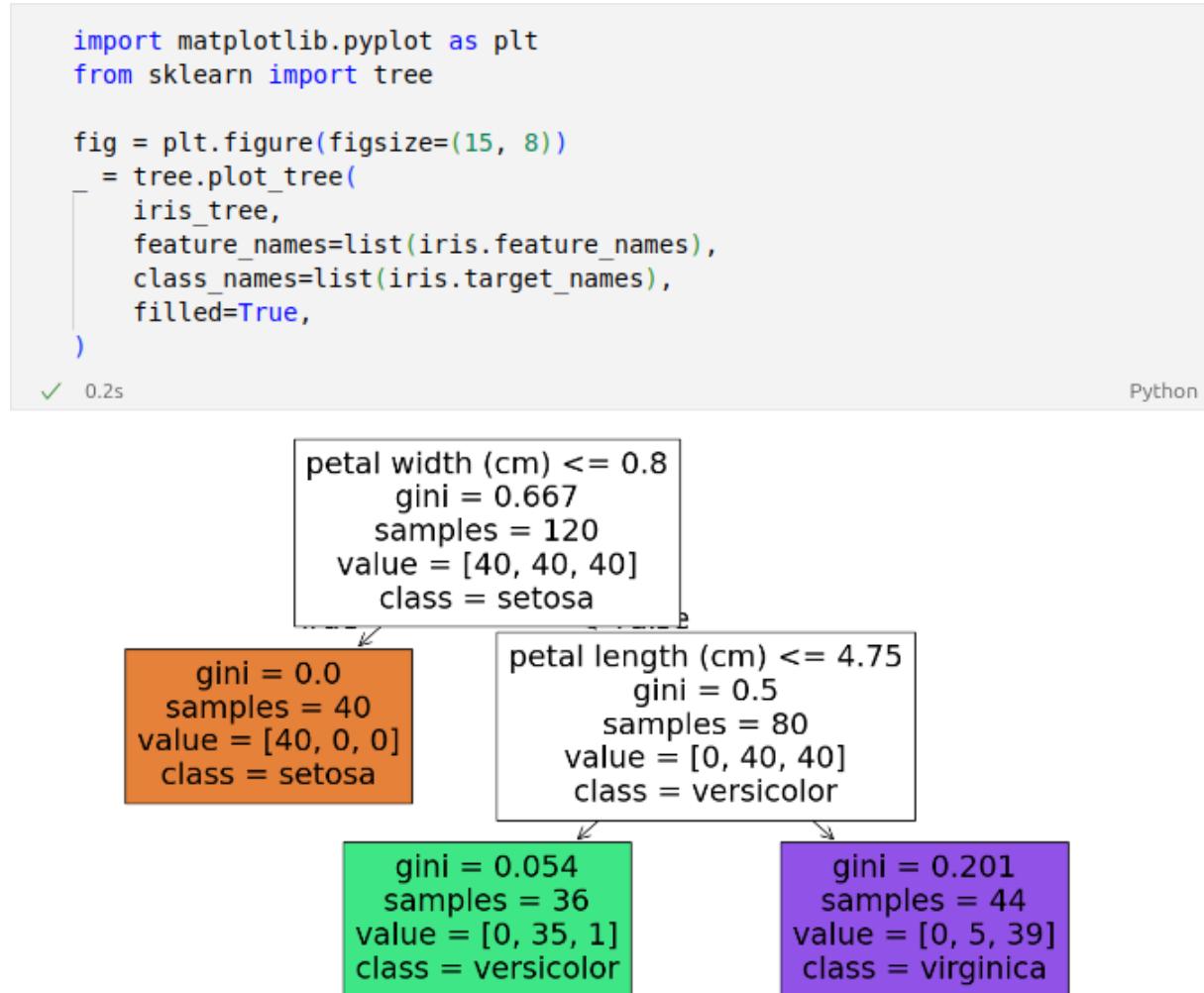
X_train, X_test, y_train, y_test = train_test_split(
    features, labels, test_size=0.2, stratify=labels, random_state=13
)
iris_tree = DecisionTreeClassifier(max_depth=2, random_state=13)
iris_tree.fit(X_train, y_train)
```

✓ 0.0s

Python

```
▼      DecisionTreeClassifier      ⓘ ⓘ
DecisionTreeClassifier(max_depth=2, random_state=13)
```

5.17 전체 특성을 사용한 결정나무 모델



5.18 모델을 사용하는 방법은?

- 길가다가 주운 iris가 sepal과 petal의 length, width가 각각 [4.3, 2.0, 1.2, 1.0]이라면

```

test_data = [[4.3, 2.0, 1.2, 1.0]]
iris_tree.predict_proba(test_data)

```

✓ 0.0s Python

```

array([[0.          , 0.97222222, 0.02777778]])

```

- 각 클래스별 확률이 아니라 범주 값을 바로 알고 싶다면

```

iris.target_names
✓ 0.0s Python
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

test_data = [[4.3, 2.0, 1.2, 1.0]]
iris.target_names[iris_tree.predict(test_data)]
✓ 0.0s Python
array(['versicolor'], dtype='<U10')

```

5.19 주요 특성 확인하기

```

iris_tree.feature_importances_
✓ 0.0s Python
array([0.          , 0.          , 0.42189781, 0.57810219])

iris_clf_model = dict(zip(iris.feature_names,
                           iris_tree.feature_importances_))
iris_clf_model
✓ 0.0s Python
{'sepal length (cm)': np.float64(0.0),
 'sepal width (cm)': np.float64(0.0),
 'petal length (cm)': np.float64(0.421897810218978),
 'petal width (cm)': np.float64(0.578102189781022)}

```

- 여기서 잠깐 Black box / White box 모델
- Tree계열 알고리즘은 특성을 파악하는데 장점을 가진다

5.20 간단한 zip과 언패킹~

5.20.1 리스트를 튜플로 zip

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

✓ 0.0s

Python

```
pairs = [pair for pair in zip(list1, list2)]
pairs
```

✓ 0.0s

Python

[('a', 1), ('b', 2), ('c', 3)]

5.20.2 튜플을 dict으로

```
dict(pairs)
```

✓ 0.0s

Python

{'a': 1, 'b': 2, 'c': 3}

5.20.3 한번에~

```
dict(zip(list1, list2))
```

✓ 0.0s

Python

{'a': 1, 'b': 2, 'c': 3}

5.20.4 unpacking 인자를 이용한 역변환

```
a, b = zip(*pairs)
```

```
print(list(a))
print(list(b))
```

✓ 0.0s

Python

['a', 'b', 'c']
[1, 2, 3]