머신러닝 데이터는 왜 나누나요?

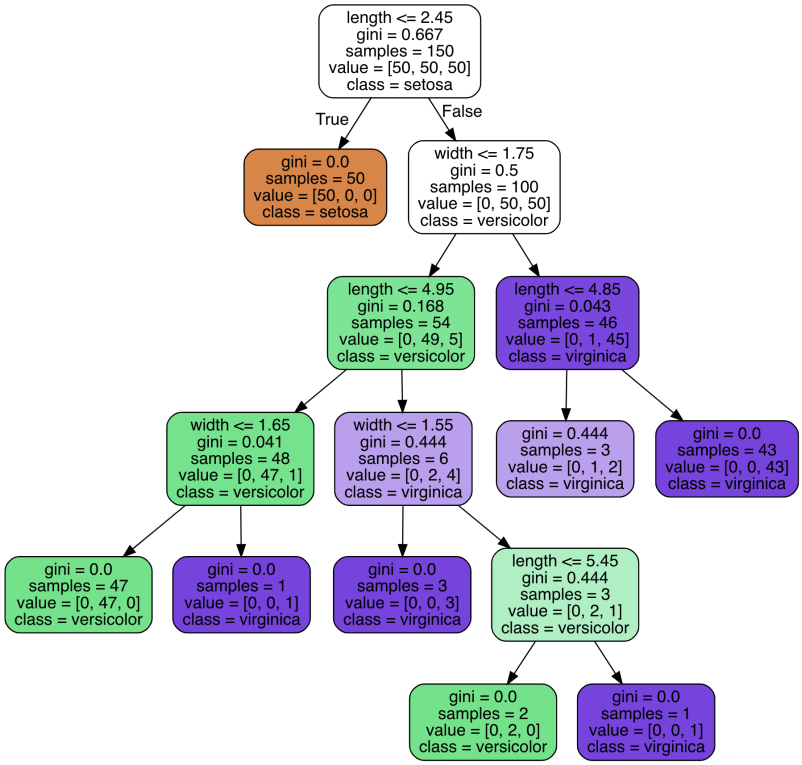
Auto Mobile Robot

Exported on 02/07/2024

Table of Contents

1 Back to Previous Story

1.1 Tree model visualization



- 이 모델이 이전에 엄청 accuracy가 좋았던 모델이다

1.2 mlxtend 설치

```
(fc) ~
(fc) ~ pip install mlxtend
Collecting mlxtend
  Downloading mlxtend-0.17.2-py2.py3-none-any.whl (1.3 MB)
     |████████████████████████████████| 1.3 MB 333 kB/s
Requirement already satisfied: matplotlib>=3.0.0 in ./opt/anaconda3/envs/fc/lib/python3.7/site-packages (from mlxtend)
 (3.1.3)
Requirement already satisfied: pandas>=0.24.2 in ./opt/anaconda3/envs/fc/lib/python3.7/site-packages (from mlxtend) (1
.0.1)
Requirement already satisfied: scipy>=1.2.1 in ./opt/anaconda3/envs/fc/lib/python3.7/site-packages (from mlxtend) (1.4
```

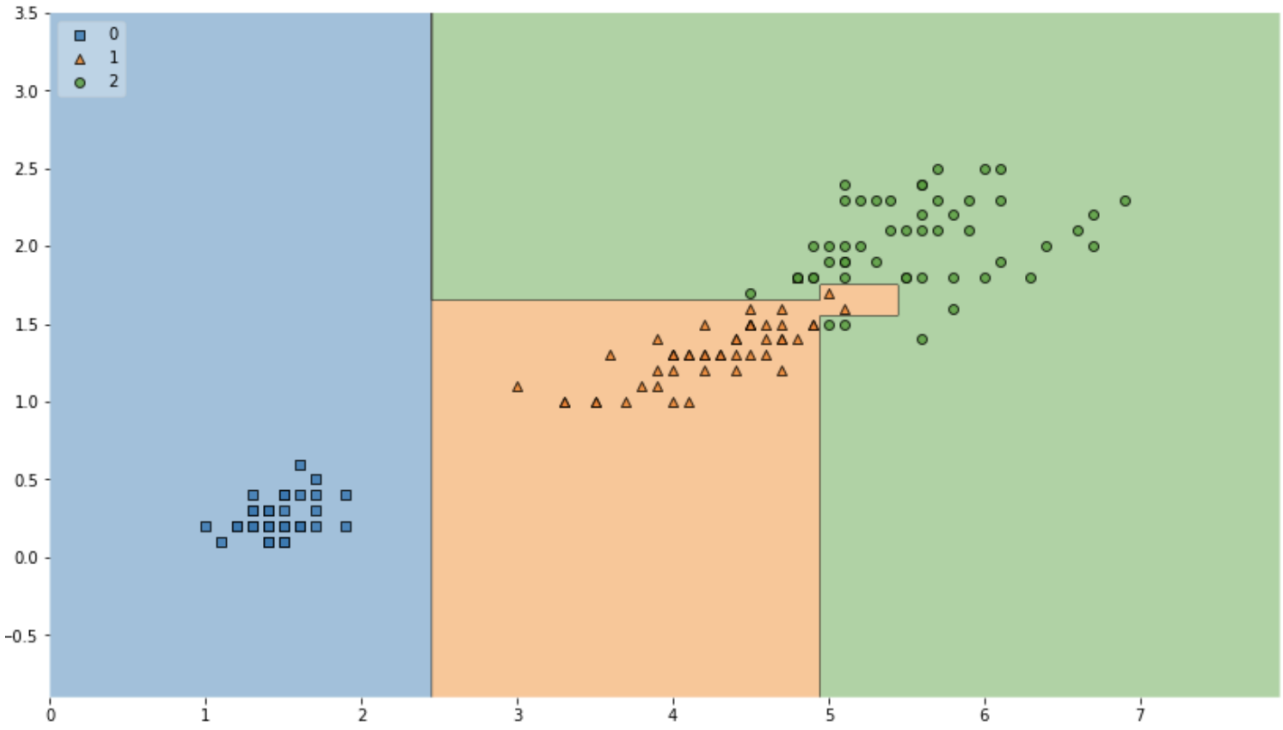
- sklearn에 없는 몇몇 유용한 기능을 가지고 있다

1.3 iris의 품종을 분류하는 결정나무 모델이 어떻게 데이터를 분류했는지 확인해보자

```
from mlxtend.plotting import plot_decision_regions

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

1.4 결정 경계를 확인해보자



- 우리가 여기서 느껴야 할 것은?

2 머신러닝의 일반적 절차에 대해...

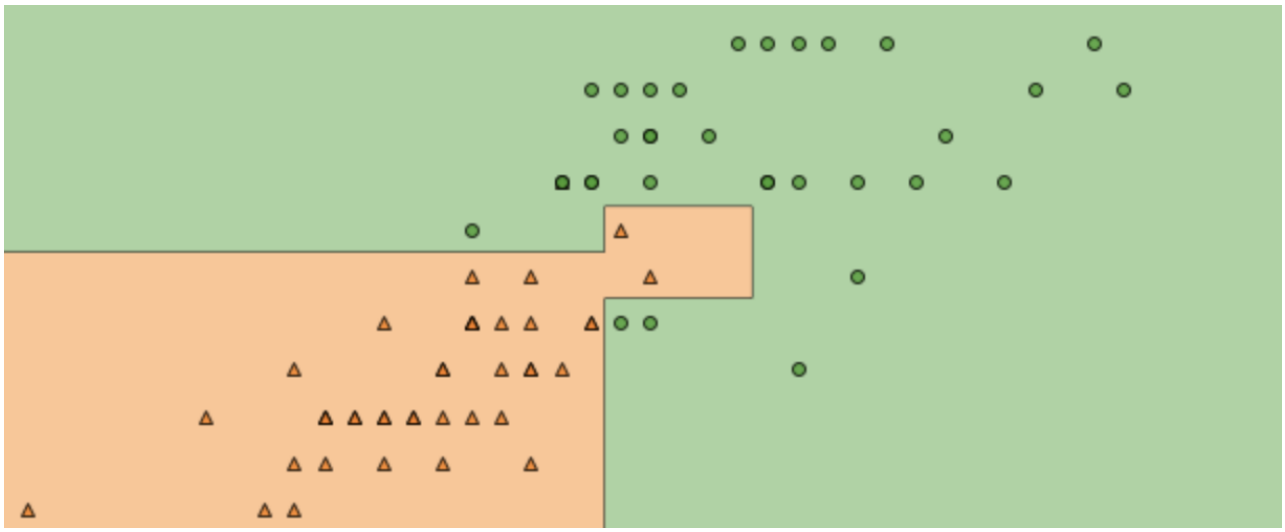
2.1 Accuracy가 높다고? 믿을 수 있을까?

```
from sklearn.metrics import accuracy_score

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

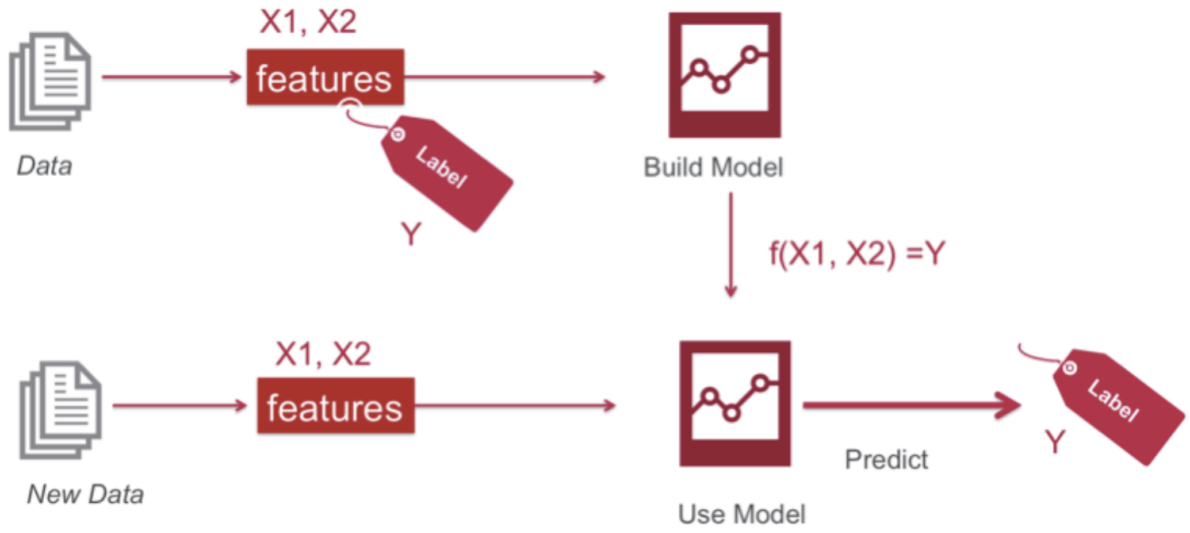
```
0.9933333333333333
```

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



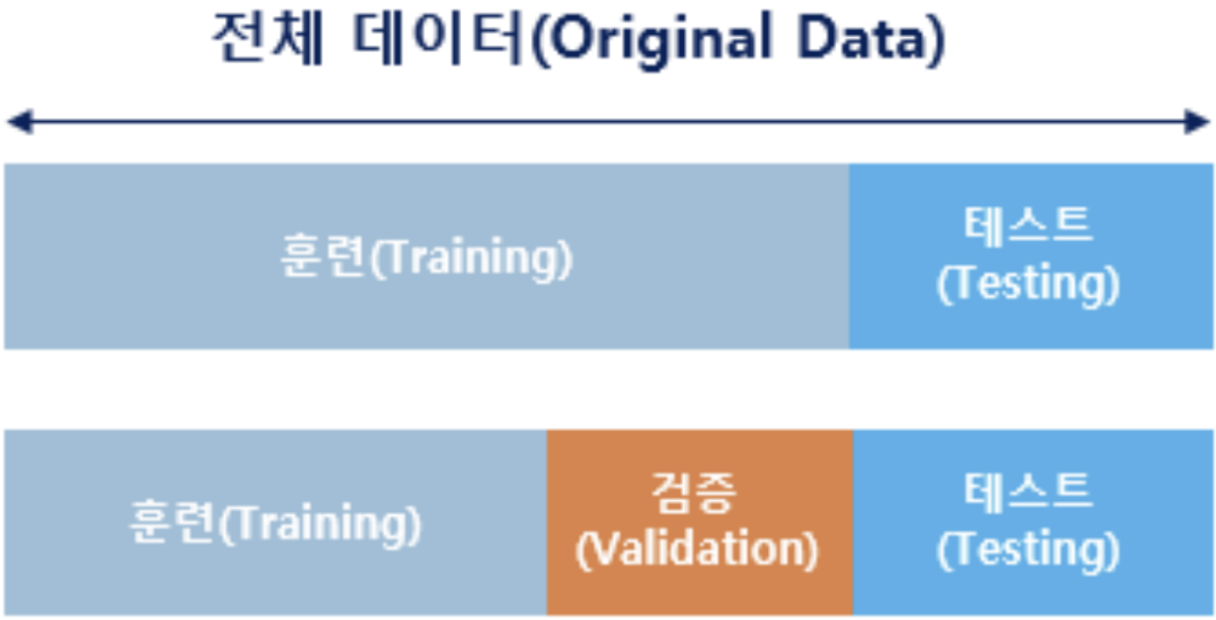
- 저 경계면은 올바른 걸까?
- 저 결과는 내가 가진 데이터를 벗어나서 일반화할 수 있는 걸까?
- 어차피 얻은(혹은 구한) 데이터는 유한하고 내가 얻은 데이터를 이용해서 일반화를 추구하게 된다.
- 이때 복잡한 경계면은 모델의 성능을 결국 나쁘게 만든다.

2.2 지도 학습



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

2.3 데이터의 분리 (훈련 / 검증 / 평가)



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

2.4 새롭게 다시 시작

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

iris = load_iris()
```

2.5 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)
```

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

2.6 훈련용 / 테스트용이 잘 분리 되었을까?

```
import numpy as np

np.unique(y_test, return_counts=True)
```

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

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

2.7 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)
```

```
import numpy as np

np.unique(y_test, return_counts=True)
```

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

2.8 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)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       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=13, splitter='best')
```

- 학습할 때 마다 일관성을 위해 random_state만 고정해서~
- 모델을 단순화시키기 위해 max_depth를 조정

2.9 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)
```

```
0.95
```

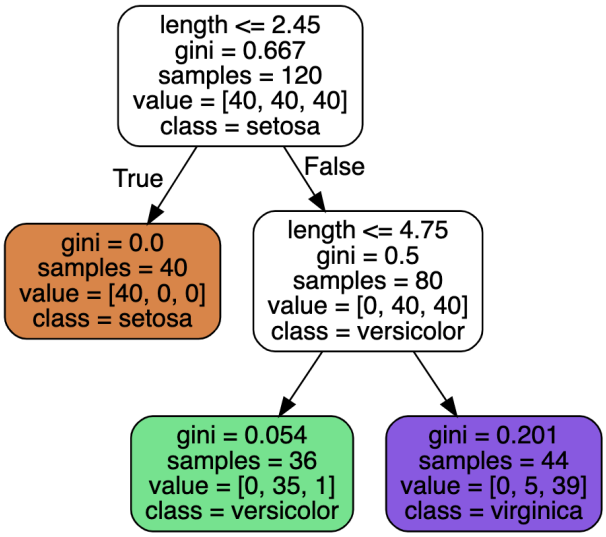
- iris 데이터는 단순해서 acc가 높게 나타남

2.10 모델 확인

```
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)
```

2.11 iris 꽃을 분류하는 결정나무 모델



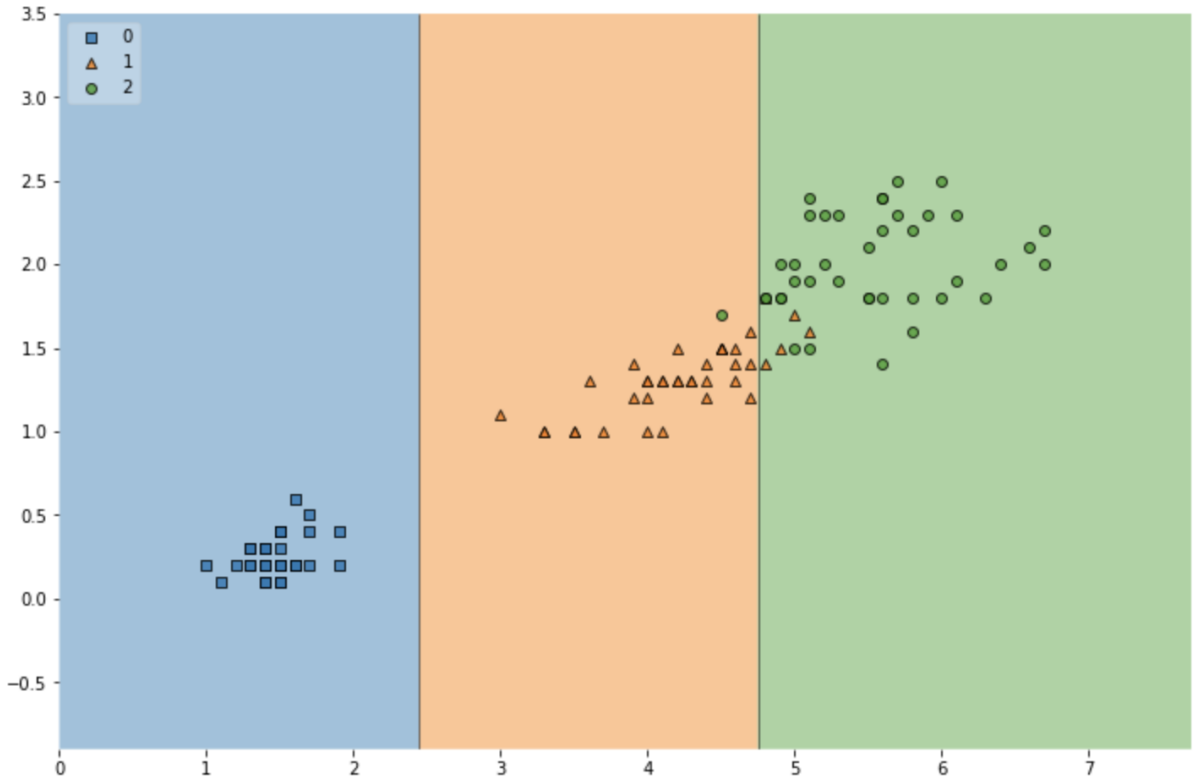
- 이 안에 표현된 값들을 이제 이해할 수 있겠지…

2.12 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()
```

2.13 훈련용 데이터에 대한 결정경계

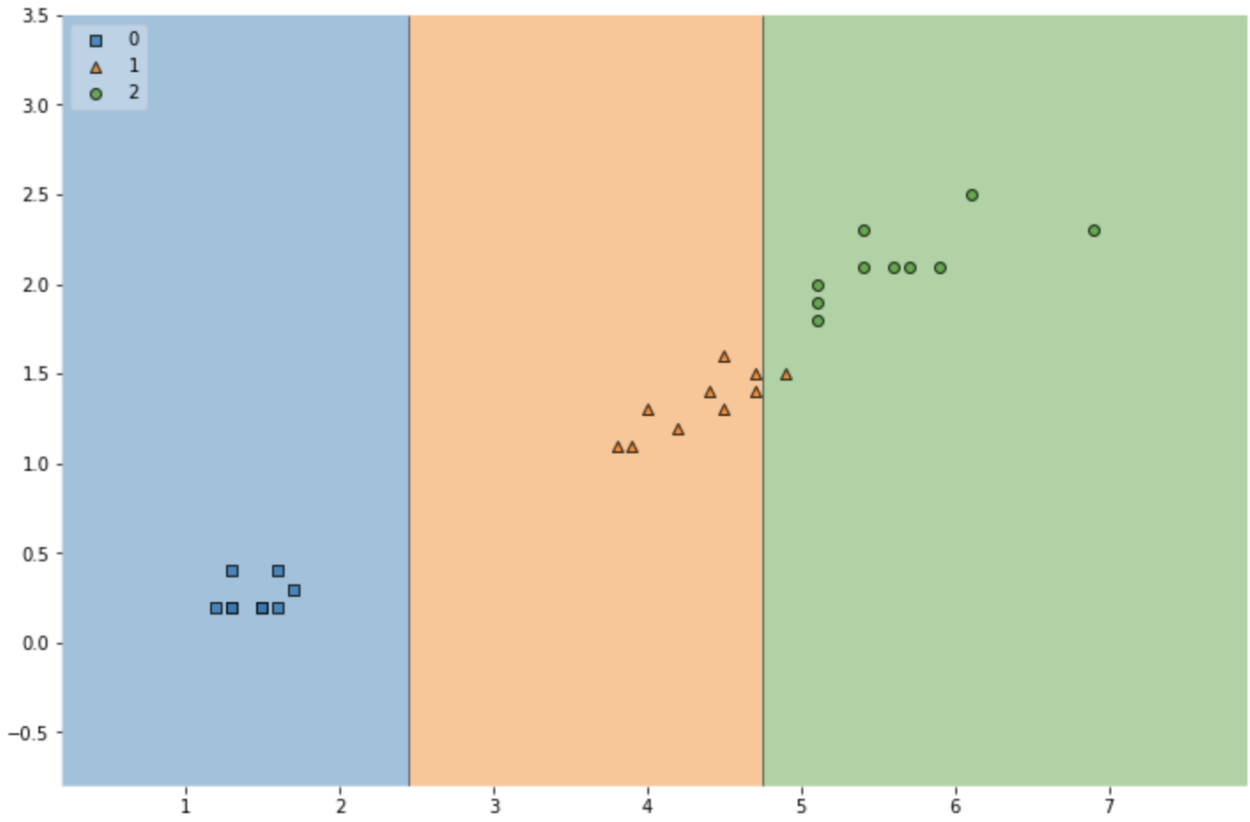


2.14 테스트 데이터에 대한 accuracy

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

```
0.9666666666666667
```

2.15 결과

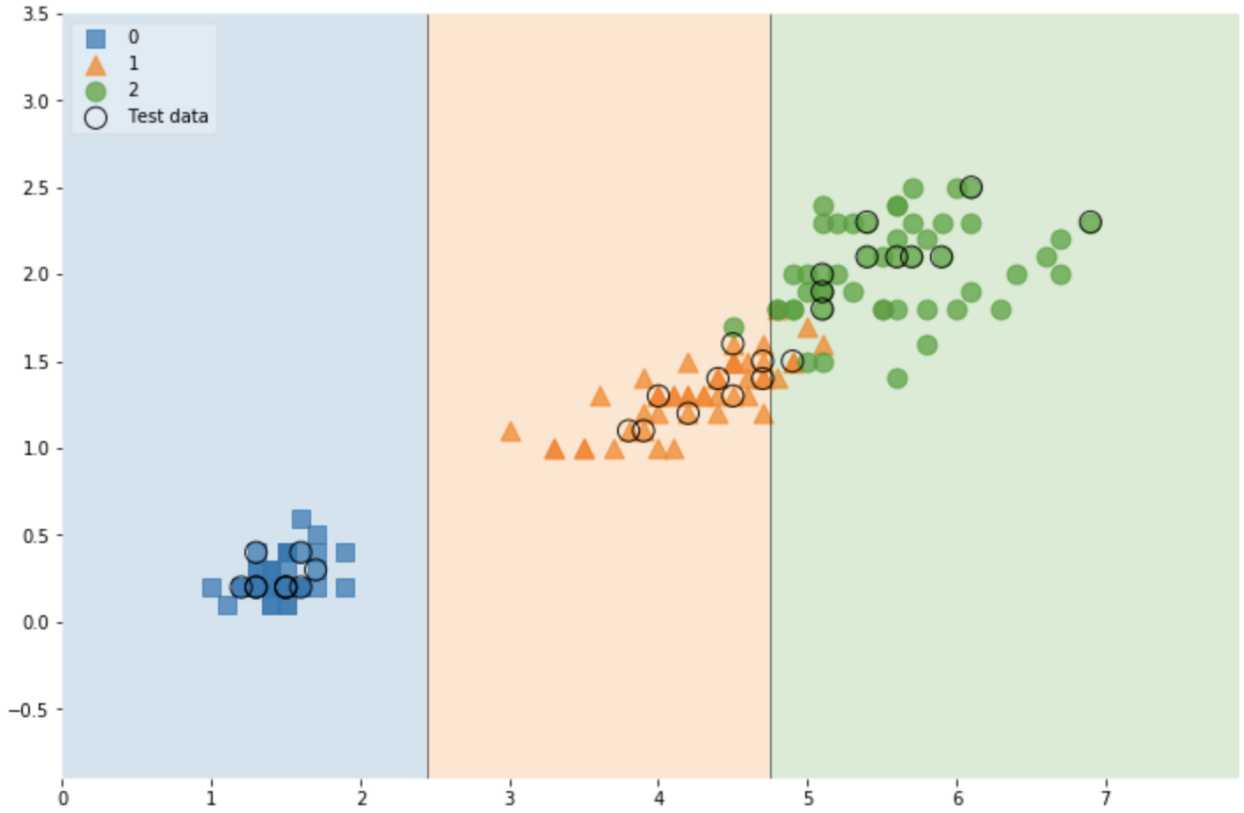


2.16 잔기술 하나 - 전체 데이터에서 관찰해보기

```
scatter_highlight_kwargs = {'s': 150, 'label': 'Test data', 'alpha': 0.9}
scatter_kwargs = {'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_kwargs=scatter_highlight_kwargs,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs={'alpha': 0.2})
plt.show()
```

2.17 결과

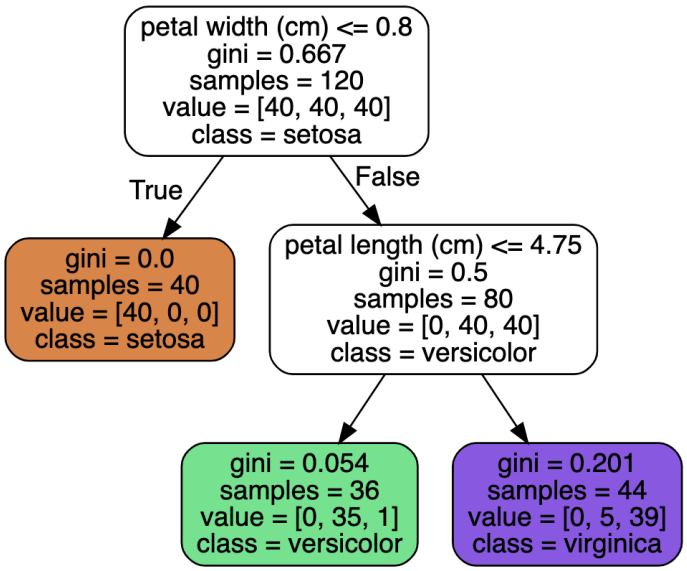


2.18 feature를 네 개..

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

fig = plt.figure(figsize=(15, 8))
_ = tree.plot_tree(iris_tree,
                   feature_names=list(iris.feature_names),
                   class_names = list(iris.target_names),
                   filled=True)
```

2.19 전체 특성을 사용한 결정나무 모델



2.20 모델을 사용하는 방법은?

- 길가다가 주운 iris가 sepal과 petal의 length, width가 각각 [4.3, 2. , 1.2, 1.0]이라면

```
test_data = [[4.3, 2. , 1.2, 1.0]]
iris_tree.predict_proba(test_data)
```

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

- 각 클래스별 확률이 아니라 범주 값을 바로 알고 싶다면

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
test_data = [[4.3, 2. , 1.2, 1.0]]
iris.target_names[iris_tree.predict(test_data)]
```

```
array(['versicolor'], dtype='<U10')
```

2.21 주요 특성 확인하기

```
iris_tree.feature_importances_
```

```
array([0.        , 0.        , 0.42189781, 0.57810219])
```

```
iris_clf_model =  dict(zip(iris.feature_names, iris_tree.feature_importances_))
iris_clf_model
```

```
{'sepal length (cm)': 0.0,
 'sepal width (cm)': 0.0,
 'petal length (cm)': 0.421897810218978,
 'petal width (cm)': 0.578102189781022}
```

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

2.22 간단한 zip과 언패킹~

2.22.1 리스트를 튜플로 zip

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

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

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

2.22.2 튜플을 dict으로

```
dict(pairs)
```

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

2.22.3 한번에~

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

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

2.22.4 unpacking 인자를 이용한 역변환

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

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

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