Small data technique

Practice Session 13

Changho Suh

January 26, 2024

Recap: DTs

A decision-based model of the tree structure.

Training algorithm: **CART**

Hyperparameters:

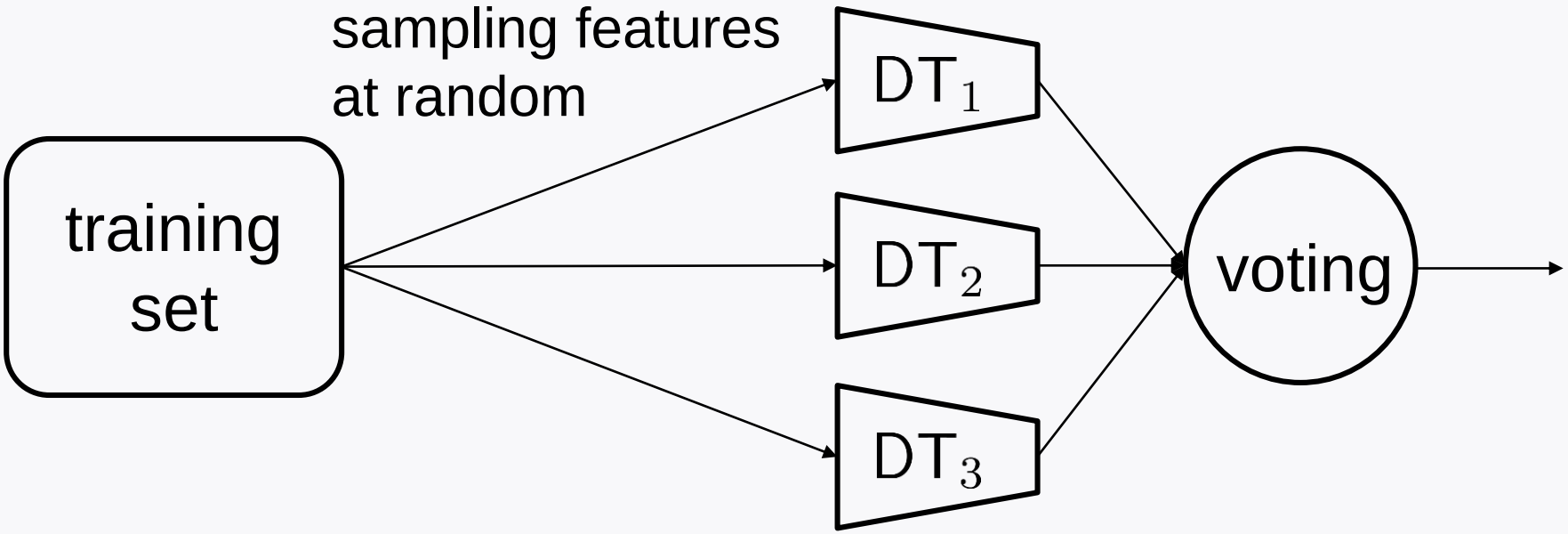
“max_depth” “min_samples_split”

“max_leaf_nodes” “min_samples_leaf”

Challenge: Sensitive to small variations in training data

Recap: RFs

An ensemble of DTs, each trained on the random subspace method



Hyperparameters: **“max_features”** **“n_estimators”**

A measure for interpretation: **Feature importance**

Outline

1. Implement **decision tree classifier**

   Task: Iris plants classification

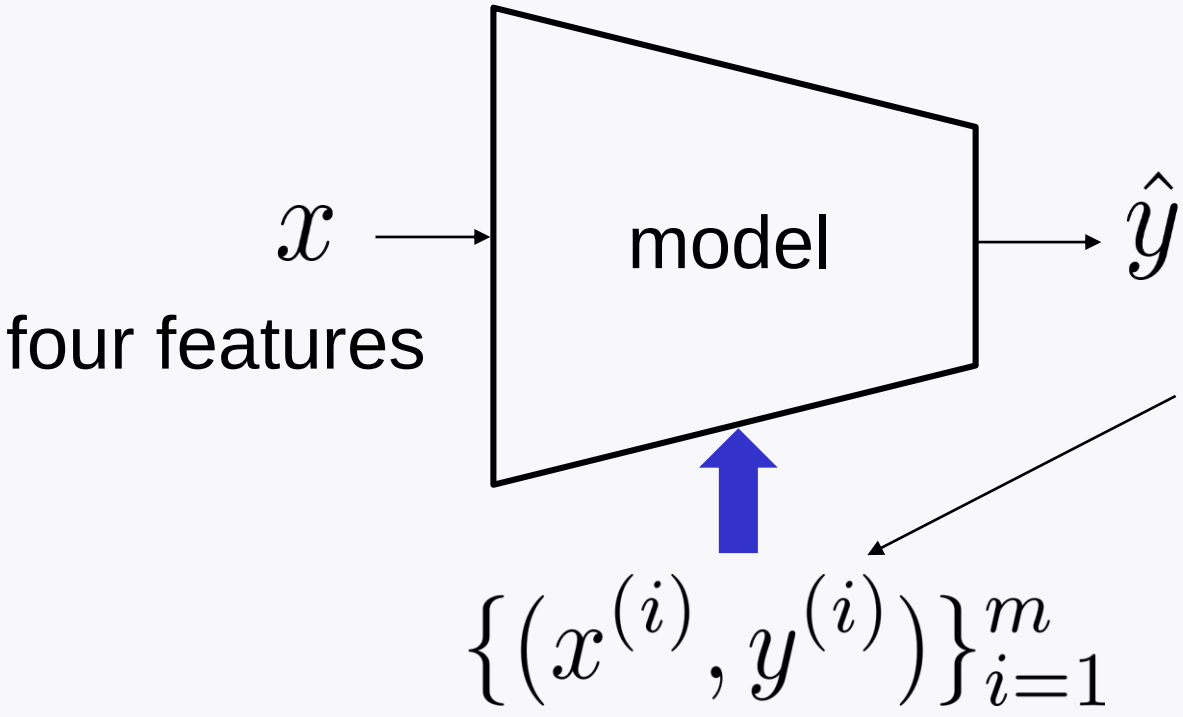
2. Implement **decision tree regressor**

   Task: California housing price prediction

3. Implement **random forests**

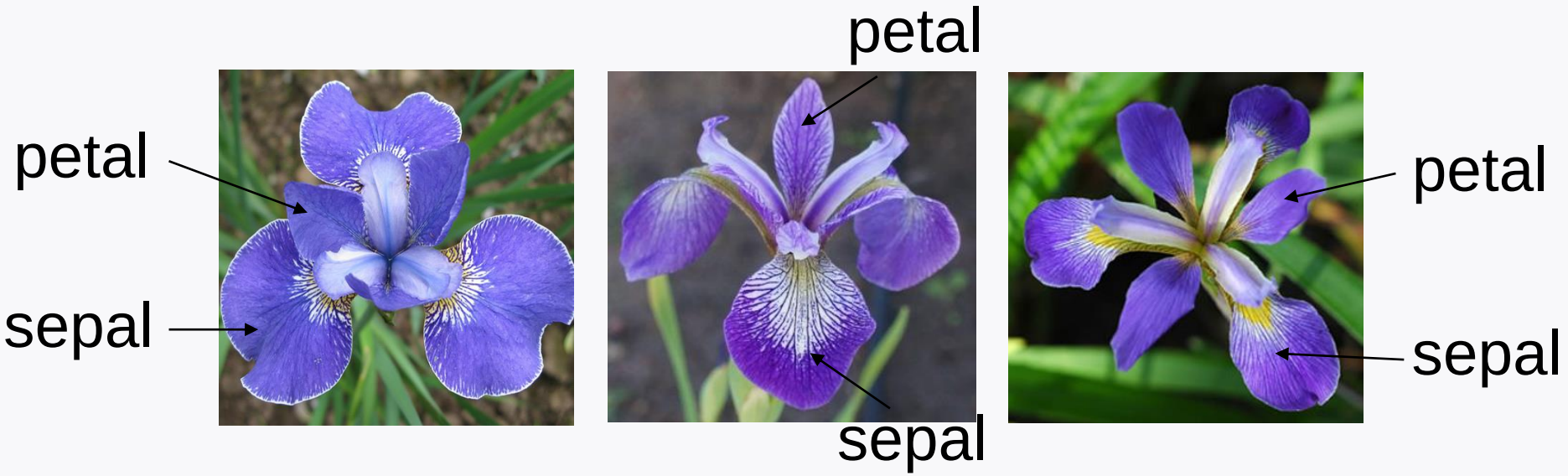
   Tasks: Iris plants classification
   California housing price prediction
   Handwritten digit classification

Iris plants classification



Four features

Class: setosa (0) versicolor (1) virginica (2)



Focus on petal length & width:

x_1 : petal length

x_2 : petal width

Load Iris dataset

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

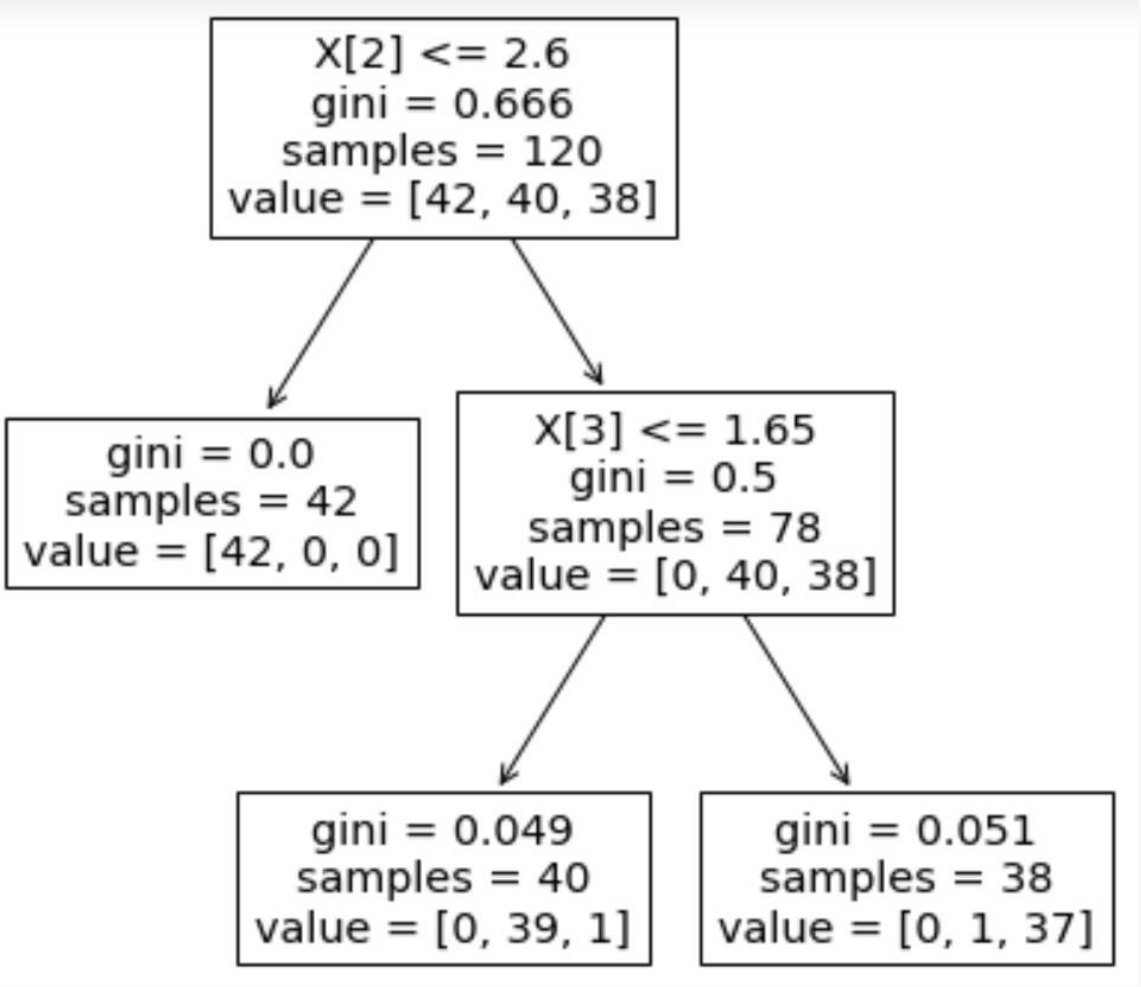
Decision tree classifier

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# train-test data split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

tree_clf = DecisionTreeClassifier(max_depth=2)

# training
tree_clf.fit(X_train, y_train)

# evaluation
test_performance = tree_clf.score(X_test, y_test)
print(test_performance)
```

```
0.8666666666666667
```

Decision tree visualization

```
from sklearn.tree import plot_tree

plot_tree(tree_clf)
```



Feature importance

```
print(tree_clf.feature_importances_)
```

```
[0.         0.         0.53867896 0.46132104]
```

```
for name, score in zip(iris.feature_names, tree_clf.feature_importances_):
    print(name, ': ', score)
```

```
sepal length (cm) :  0.0
sepal width (cm) :  0.0
petal length (cm) :  0.5386789581249728
petal width (cm) :  0.4613210418750273
```

Feature importance visualization

```
print(tree_clf.feature_importances_)
```

```
[0.         0.         0.53867896 0.46132104]
```

```
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

0 1 2 3

```
sorted_idx = tree_clf.feature_importances_.argsort()
```

increasing order

```
print(sorted_idx)
```

⟶ `[0 1 3 2]`

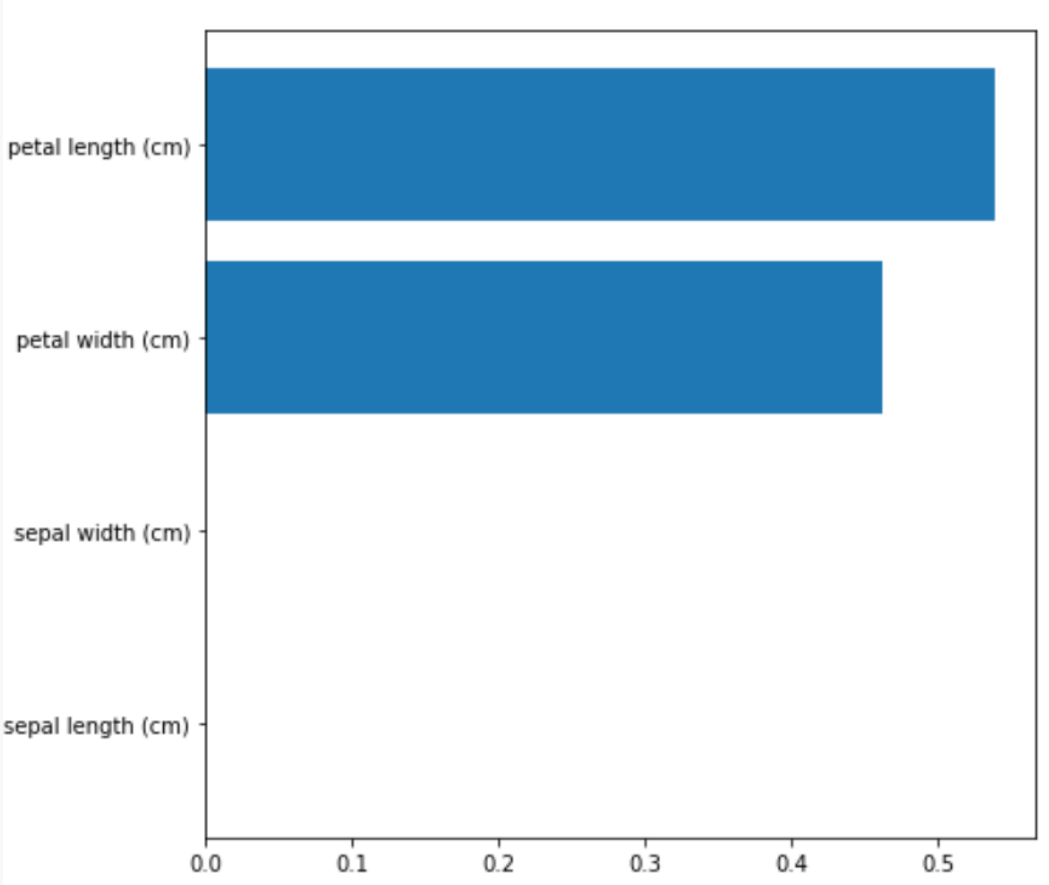
```
print(np.asarray(iris.feature_names)[sorted_idx])
```

```
['sepal length (cm)' 'sepal width (cm)' 'petal width (cm)'
 'petal length (cm)']
```

Feature importance visualization

```
import matplotlib.pyplot as plt
```

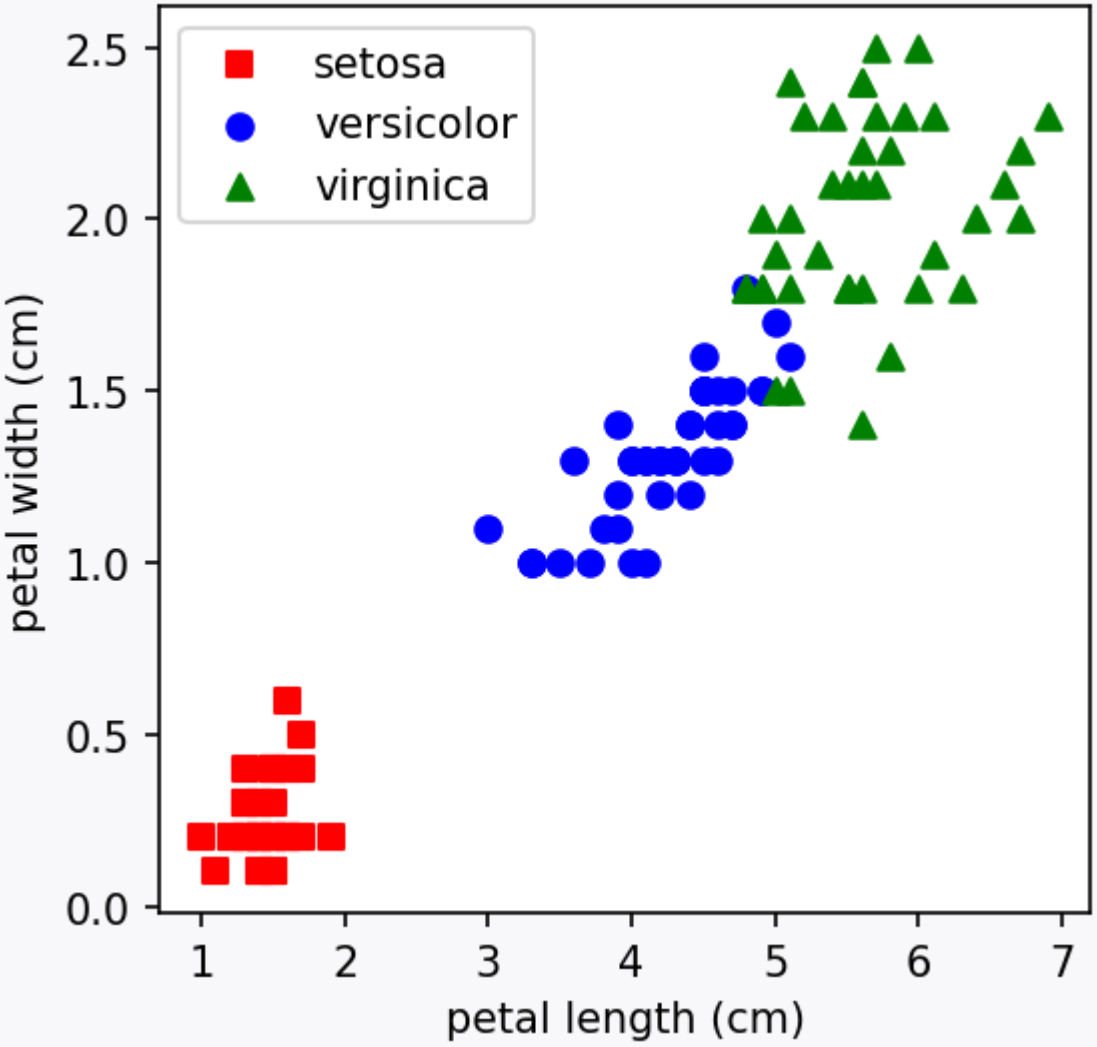
```
plt.barh(np.asarray(iris.feature_names)[sorted_idx],
         tree_clf.feature_importances_[sorted_idx])
```



Scatter plot

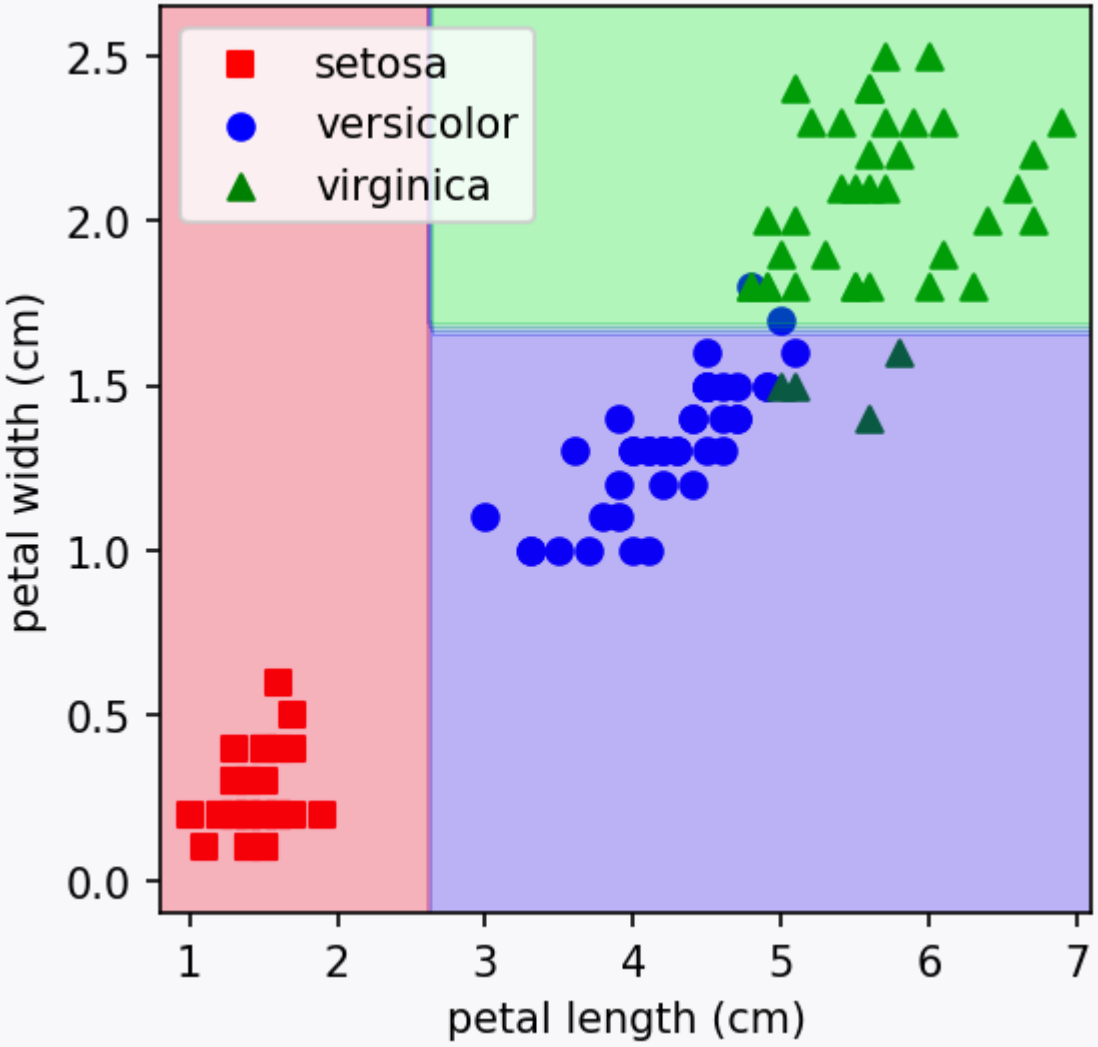
```
import matplotlib.pyplot as plt

X_y_0 = X_train[y_train==0] # setosa
X_y_1 = X_train[y_train==1] # versicolor
X_y_2 = X_train[y_train==2] # virginica

plt.scatter(X_y_0[:, 2], X_y_0[:, 3],
            c='red',label='setosa',marker='s')
plt.scatter(X_y_1[:, 2], X_y_1[:, 3],
            c='blue',label='versicolor',marker='o')
plt.scatter(X_y_2[:, 2], X_y_2[:, 3],
            c='green', label='virginica',marker='^')

plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])

plt.legend()
plt.show()
```

Scatter plot



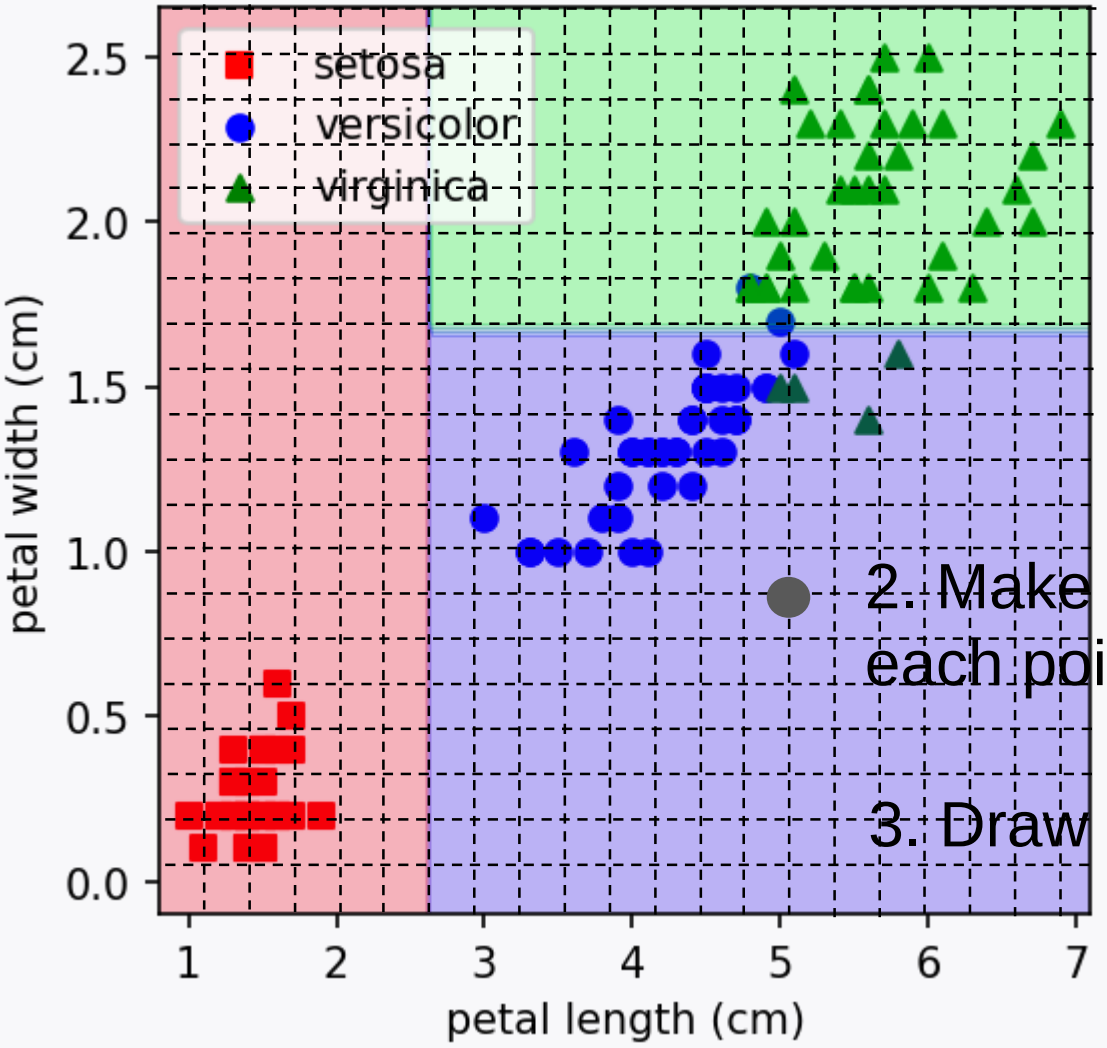
Decision boundary visualization

Suppose we want to draw:



How to draw decision boundary?

1. Construct a mesh grid



2. Make a prediction for each point in the grid

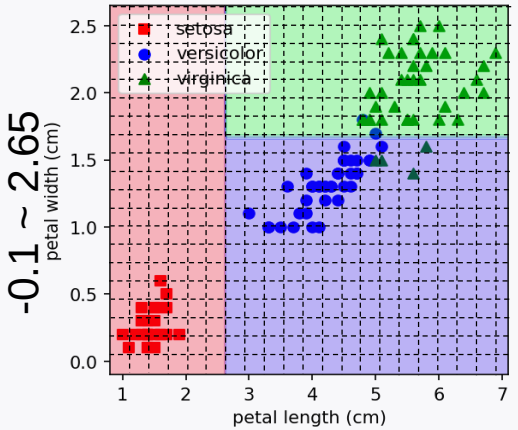
3. Draw a colored map.

1. Construct a mesh grid

petal length (0.9 ~ 7.05)

```
zz_min, zz_max = X_train[:, 2].min() - 0.2, X_train[:, 2].max() + 0.2
ww_min, ww_max = X_train[:, 3].min() - 0.2, X_train[:, 3].max() + 0.2
```

petal width (-0.1 ~ 2.65)



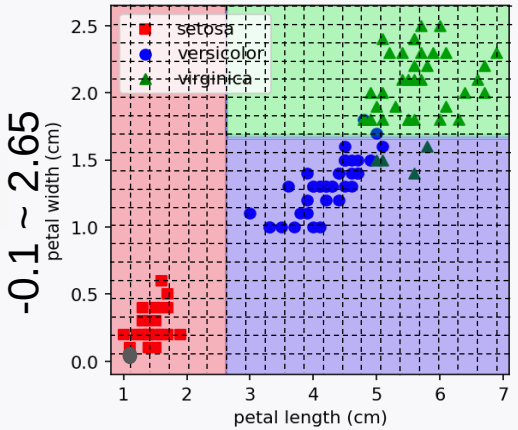

```
zz, ww = np.meshgrid(np.arange(zz_min, zz_max, 0.05),
                     np.arange(ww_min, ww_max, 0.05),indexing='ij')
```

```
[[0.9  0.9  0.9  ... 0.9  0.9  0.9 ]
 [0.95 0.95 0.95 ... 0.95 0.95 0.95]
 [1.   1.   1.   ... 1.   1.   1.  ]
 ...
 [6.95 6.95 6.95 ... 6.95 6.95 6.95]
 [7.   7.   7.   ... 7.   7.   7.  ]
 [7.05 7.05 7.05 ... 7.05 7.05 7.05]]
```

```
[[-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 ...
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]]
```

2. Make a prediction for each point

```
zz, ww = np.meshgrid(np.arange(zz_min, zz_max, 0.05),
                     np.arange(ww_min, ww_max, 0.05),indexing='ij')
```



-0.1 ~ 2.65

0.9 ~ 7.05

```
[[0.9  0.9  0.9  ... 0.9  0.9  0.9 ]
 [0.95 0.95 0.95 ... 0.95 0.95 0.95]
 [1.   1.   1.   ... 1.   1.   1.  ]
 ...
 [6.95 6.95 6.95 ... 6.95 6.95 6.95]
 [7.   7.   7.   ... 7.   7.   7.  ]
 [7.05 7.05 7.05 ... 7.05 7.05 7.05]]
```

```
[[-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 ...
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]]
```

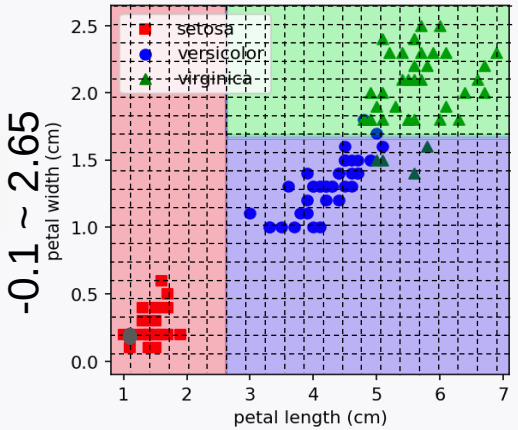
```
xx = np.zeros_like(zz)
yy = np.zeros_like(zz)
```

xx, yy, zz, ww

Wish to obtain prediction for [0, 0, 0.9, -0.1]

2. Make a prediction for each point

```
zz, ww = np.meshgrid(np.arange(zz_min, zz_max, 0.05),
                     np.arange(ww_min, ww_max, 0.05),indexing='ij')
```



-0.1 ~ 2.65

0.9 ~ 7.05

```
[[0.9  0.9  0.9  ... 0.9  0.9  0.9 ]
 [0.95 0.95 0.95 ... 0.95 0.95 0.95]
 [1.   1.   1.   ... 1.   1.   1.  ]
 ...
 [6.95 6.95 6.95 ... 6.95 6.95 6.95]
 [7.   7.   7.   ... 7.   7.   7.  ]
 [7.05 7.05 7.05 ... 7.05 7.05 7.05]]
```

```
[[-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 ...
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]]
```

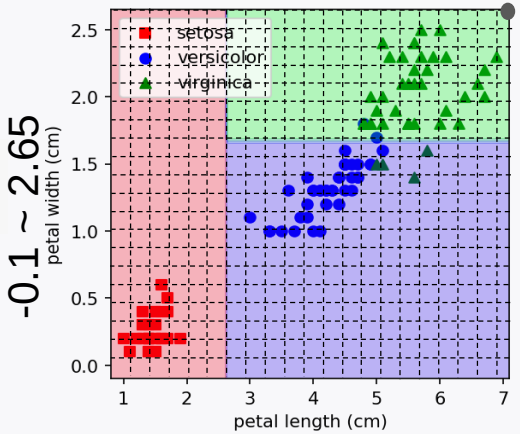
```
xx = np.zeros_like(zz)
yy = np.zeros_like(zz)
```

xx, yy, zz, ww

Wish to obtain prediction for [0, 0, 0.9, -0.05]

2. Make a prediction for each point

```
zz, ww = np.meshgrid(np.arange(zz_min, zz_max, 0.05),
                     np.arange(ww_min, ww_max, 0.05),indexing='ij')
```




```
[[0.9  0.9  0.9  ... 0.9  0.9  0.9 ]
 [0.95 0.95 0.95 ... 0.95 0.95 0.95]
 [1.   1.   1.   ... 1.   1.   1.  ]
 ...
 [6.95 6.95 6.95 ... 6.95 6.95 6.95]
 [7.   7.   7.   ... 7.   7.   7.  ]
 [7.05 7.05 7.05 ... 7.05 7.05 7.05]]
```

```
[[-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 ...
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]]
```

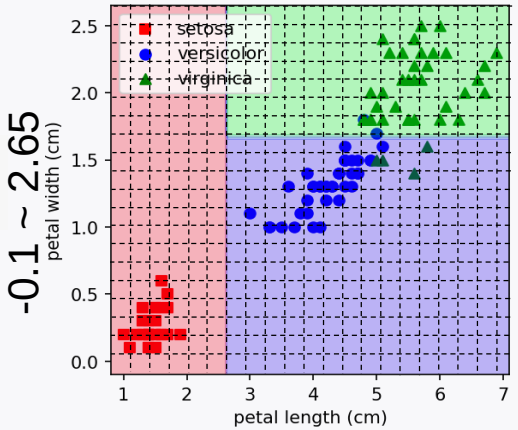
```
xx = np.zeros_like(zz)
yy = np.zeros_like(zz)
```

xx, yy, zz, ww

Wish to obtain prediction for [0, 0, 7.05, 2.65]

2. Make a prediction for each point

```
zz, ww = np.meshgrid(np.arange(zz_min, zz_max, 0.05),
                     np.arange(ww_min, ww_max, 0.05),indexing='ij')
```



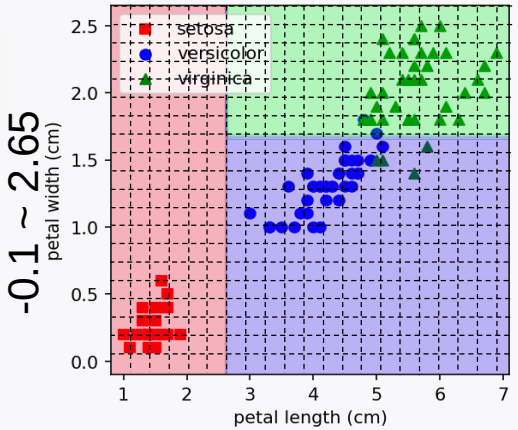

```
[[0.9  0.9  0.9  ... 0.9  0.9  0.9 ]
 [0.95 0.95 0.95 ... 0.95 0.95 0.95]
 [1.   1.   1.   ... 1.   1.   1.  ]
 ...
 [6.95 6.95 6.95 ... 6.95 6.95 6.95]
 [7.   7.   7.   ... 7.   7.   7.  ]
 [7.05 7.05 7.05 ... 7.05 7.05 7.05]]
```

```
[[-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 ...
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]
 [-0.1  -0.05  0.   ...  2.55  2.6   2.65]]
```

```
Z = tree_clf.predict(np.c_[xx.reshape(-1, 1),
                           yy.reshape(-1, 1),
                           zz.reshape(-1, 1),
                           ww.reshape(-1, 1)])
Z = Z.reshape(zz.shape)
```

3. Draw a colored map

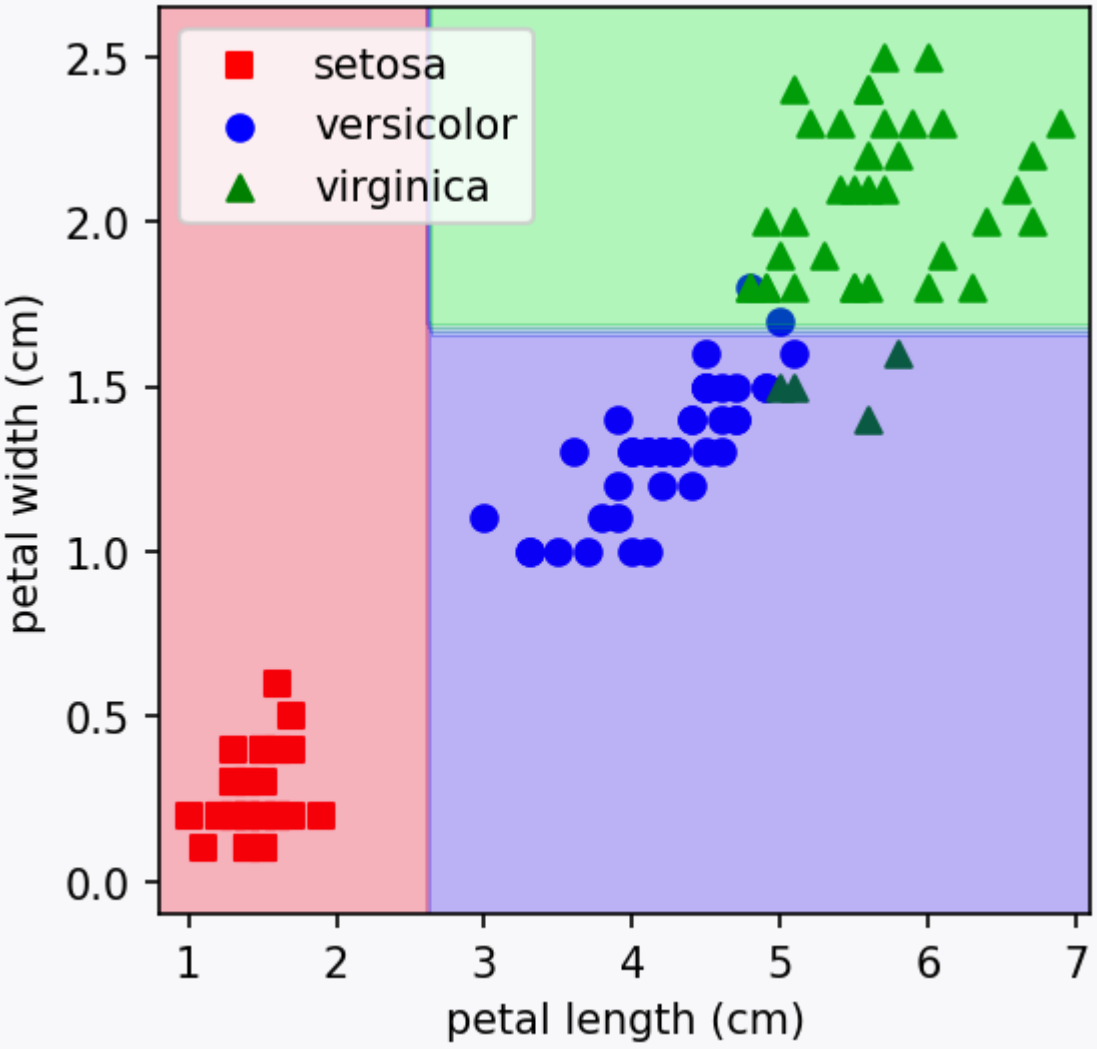
```
zz, ww = np.meshgrid(np.arange(zz_min, zz_max, 0.05),
                     np.arange(ww_min, ww_max, 0.05),indexing='ij')
xx = np.zeros_like(zz)
yy = np.zeros_like(zz)

Z = tree_clf.predict(np.c_[xx.reshape(-1, 1),
                           yy.reshape(-1, 1),
                           zz.reshape(-1, 1),
                           ww.reshape(-1, 1)])
Z = Z.reshape(zz.shape)
```




```
from matplotlib.colors import LinearSegmentedColormap

colors = [(1, 0, 0), (0, 0, 1), (0, 1, 0)]

plt.contourf(zz, ww, Z,
             cmap=LinearSegmentedColormap.from_list('rgb', colors),
             alpha=0.3)
plt.show()
```

transparency

Decision boundary visualization



Look ahead

Implementation of **decision tree regressor**

Task: California housing price prediction