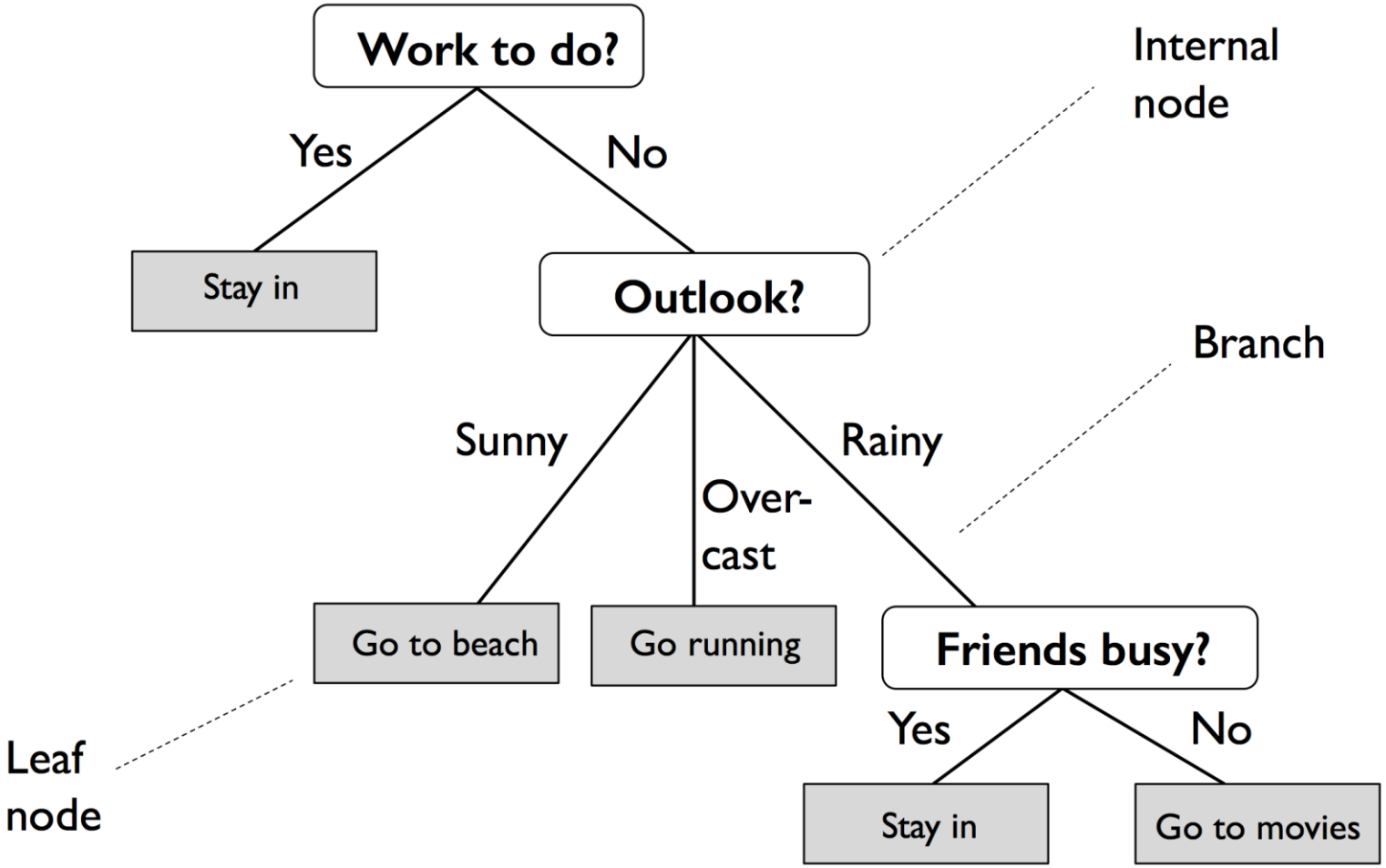
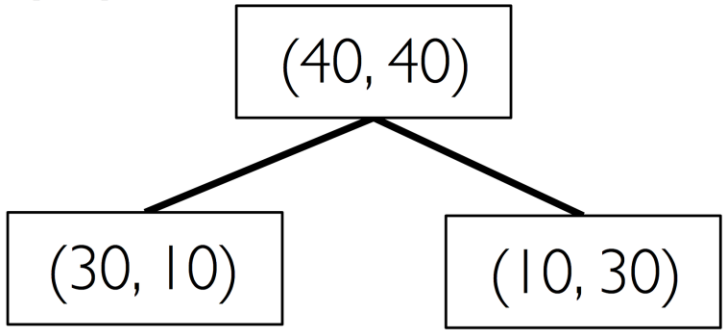
決定木

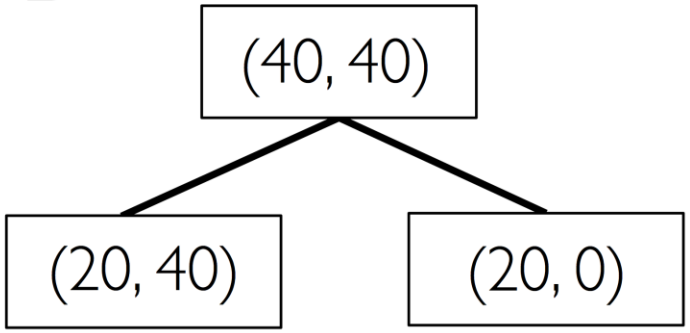

Work to do?
Yes
No
Stay in
Outlook?
Internal node
Branch
Sunny
Over-cast
Rainy
Go to beach
Go running
Friends busy?
Leaf node
Yes
No
Stay in
Go to movies

決定木

A



B



決定木

できるだけ高い効果が得られるように木を構築する。
不純度(impurity)

ジニ不純度
エントロピー
分類誤差

決定木

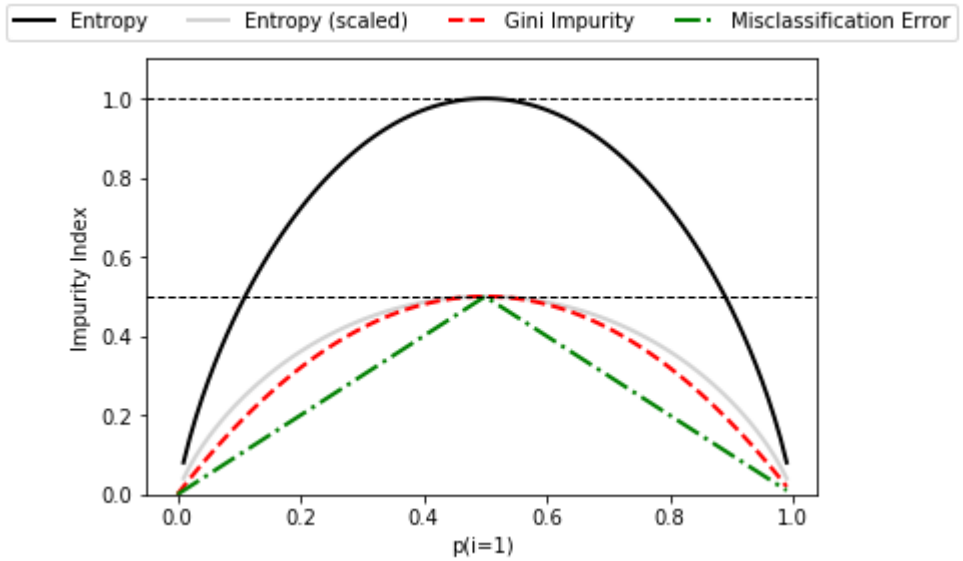
できるだけ高い効果が得られるように木を構築する。

不純度(impurity)

ジニ不純度

エントロピー

分類誤差



In [39]:
```
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

print('Class labels:', np.unique(y))
```

```
Class labels: [0 1 2]
```

In [92]:
```
iris.data[:10]
```

Out[92]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [99]:
```
iris.feature_names
```

Out[99]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [97]:
```
iris.target_names
```

Out[97]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

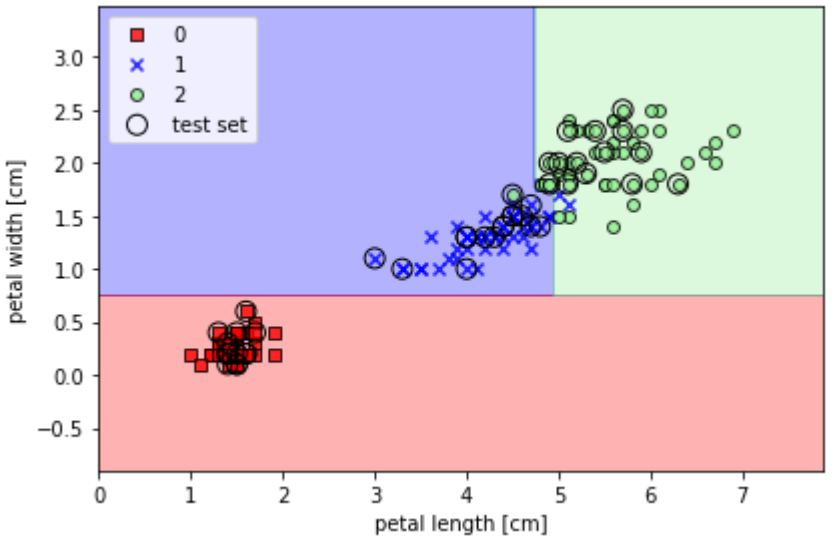
Splitting data into 70% training and 30% test data:

アヤメのデータ

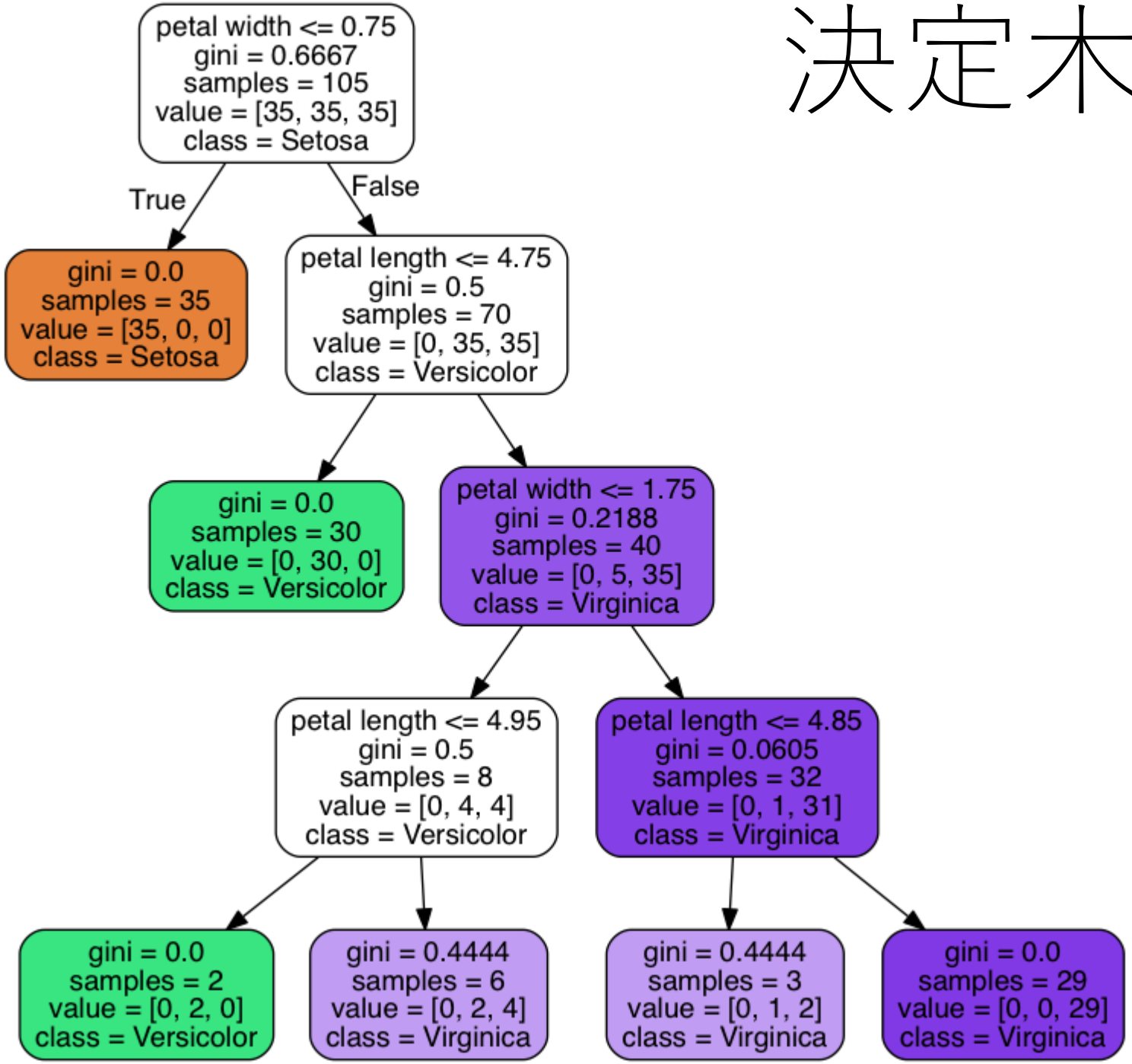
決定木

決定木の構築

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion='gini',
                              max_depth=4,
                              random_state=1)
tree.fit(X_train, y_train)

X_combined = np.vstack((X_train, X_test))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X_combined, y_combined,
                      classifier=tree, test_idx=range(105,
150))

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_20.png', dpi=300)
plt.show()
```



決定木



Petal width: 花弁の幅
Petal length: 花弁の長さ

Setosa: ヒオウギアヤメ
Versiclolor:　ブルーフラッグ(薬草)
Virginica:

Value: クラスの標本サイズ

決定木

決定木の構築

In [104]:
```
from sklearn.tree import export_graphviz
export_graphviz(tree,out_file='tree.dot')
```

決定木

決定木の構築

```
digraph Tree {
node [shape=box] ;
0 [label="X[1] <= 0.75¥ngini = 0.667¥nsamples = 105¥nvalue = [35, 35, 35]"] ;
1 [label="gini = 0.0¥nsamples = 35¥nvalue = [35, 0, 0]"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="X[0] <= 4.75¥ngini = 0.5¥nsamples = 70¥nvalue = [0, 35, 35]"] ;
0 -> 2 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
3 [label="gini = 0.0¥nsamples = 30¥nvalue = [0, 30, 0]"] ;
2 -> 3 ;
4 [label="X[1] <= 1.75¥ngini = 0.219¥nsamples = 40¥nvalue = [0, 5, 35]"] ;
2 -> 4 ;
5 [label="X[0] <= 4.95¥ngini = 0.5¥nsamples = 8¥nvalue = [0, 4, 4]"] ;
4 -> 5 ;
6 [label="gini = 0.0¥nsamples = 2¥nvalue = [0, 2, 0]"] ;
5 -> 6 ;
7 [label="gini = 0.444¥nsamples = 6¥nvalue = [0, 2, 4]"] ;
5 -> 7 ;
8 [label="X[0] <= 4.85¥ngini = 0.061¥nsamples = 32¥nvalue = [0, 1, 31]"] ;
4 -> 8 ;
9 [label="gini = 0.444¥nsamples = 3¥nvalue = [0, 1, 2]"] ;
8 -> 9 ;
10 [label="gini = 0.0¥nsamples = 29¥nvalue = [0, 0, 29]"] ;
8 -> 10 ;
}
```

決定木

決定木の構築

```
digraph Tree {
node [shape=box] ;
0 [label="X[1] <= 0.75¥ngini = 0.667¥nsamples = 105¥nvalue = [35, 35, 35]"] ;

1 [label="gini = 0.0¥nsamples = 35¥nvalue = [35, 0, 0]"] ;

0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;

2 [label="X[0] <= 4.75¥ngini = 0.5¥nsamples = 70¥nvalue = [0, 35, 35]"] ;

0 -> 2 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;

3 [label="gini = 0.0¥nsamples = 30¥nvalue = [0, 30, 0]"] ;

2 -> 3 ;
```

ランダムフォレスト

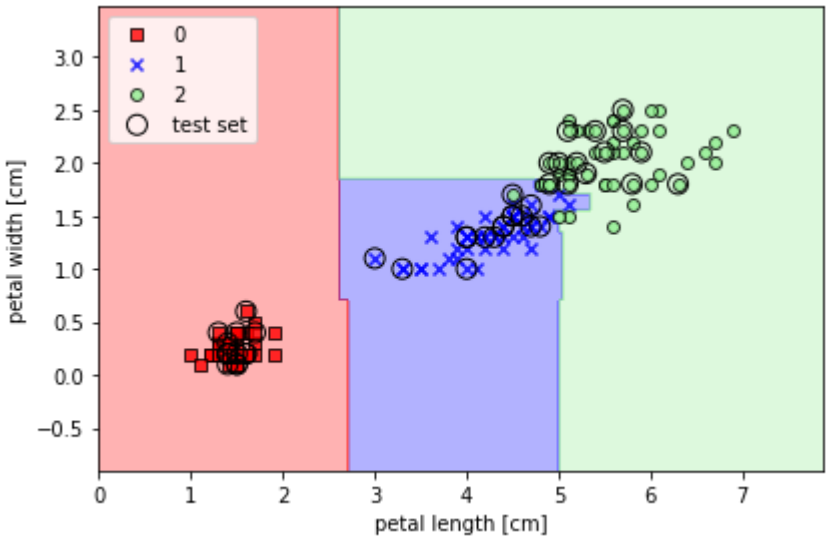
複数の決定木を結合する

1. サイズｎのランダムな「ブートストラップ」標本を抽出
2. Ｄ個の特徴量をランダムに非復元抽出する。
3. ノードを分割する
4. １－３を繰り返す。
５．多数決にしたがってクラスラベルを割り当てる。

ランダムフォレスト

複数の決定木を結合する

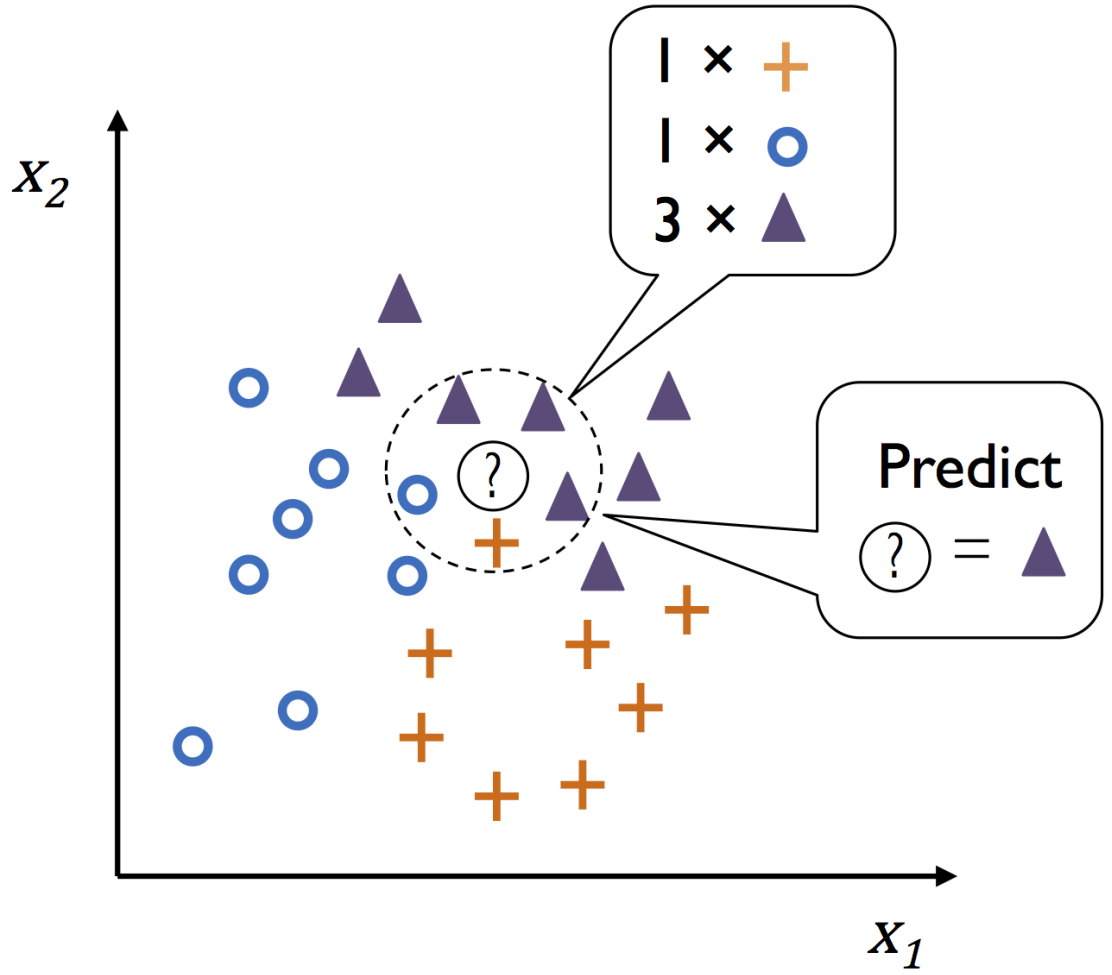
```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(criterion='gini',
                                n_estimators=25,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, y_train)

plot_decision_regions(X_combined, y_combined,
                      classifier=forest, test_idx=range(105,
150))

plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_22.png', dpi=300)
plt.show()
```



N_estimators: 木の数
N_jobs:用いるプロセッサーの数

K近傍法

訓練データを暗記する。



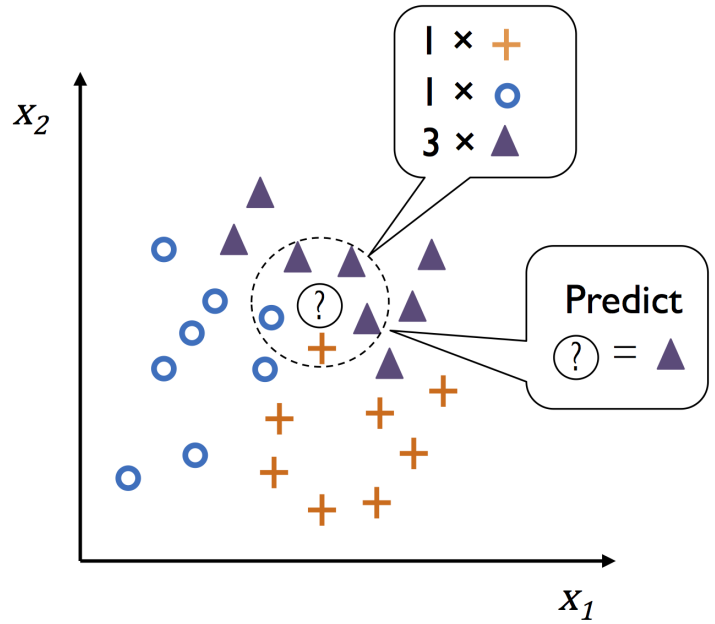
K近傍法

訓練データを暗記する。

１．Ｋの値と距離指標を選択する。

２．分類したいサンプルからｋ個の最近傍のデータ点を見つける。

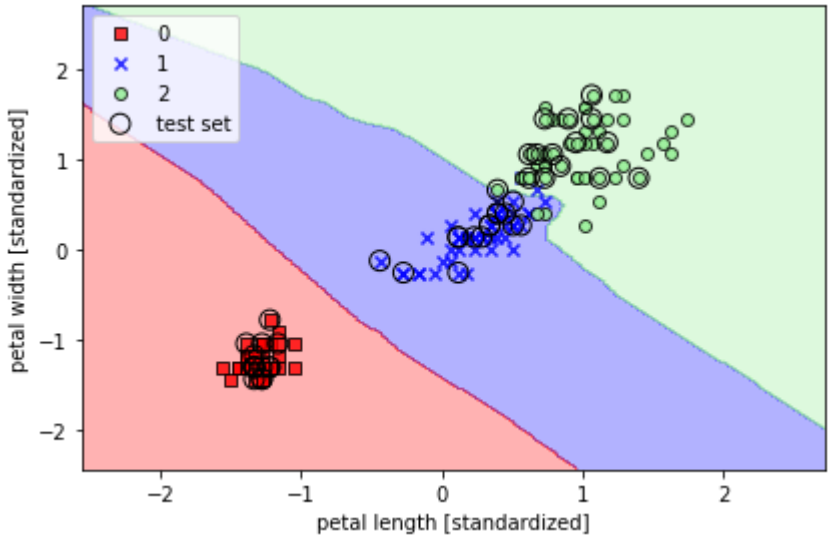
３．多数決でクラスラベルを割り当てる。



K近傍法

訓練データを暗記する。

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5,
                           p=2,
                           metric='minkowski')
knn.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=knn, test_idx=range(105, 150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_24.png', dpi=300)
plt.show()
```



K近傍法

次元の呪い

（じげんののろい、英: The curse of dimensionality）という言葉は、リチャード・ベルマンが使ったもので、（数学的）空間の次元が増えるのに対応して問題の算法が指数関数的に大きく（英語版）なることを表している。