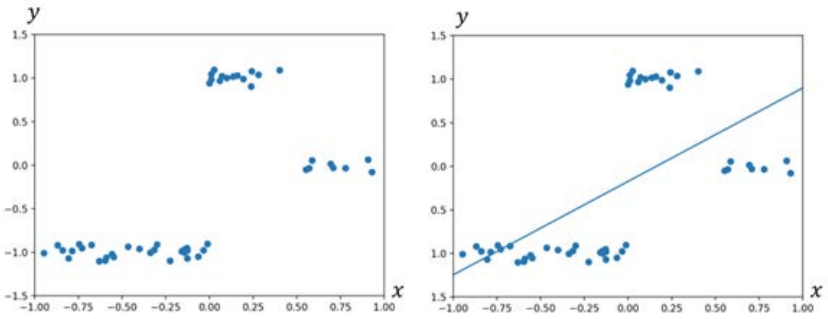


(a) (b)

图 7.1　例 7.1 的数据采样及线性回归拟合

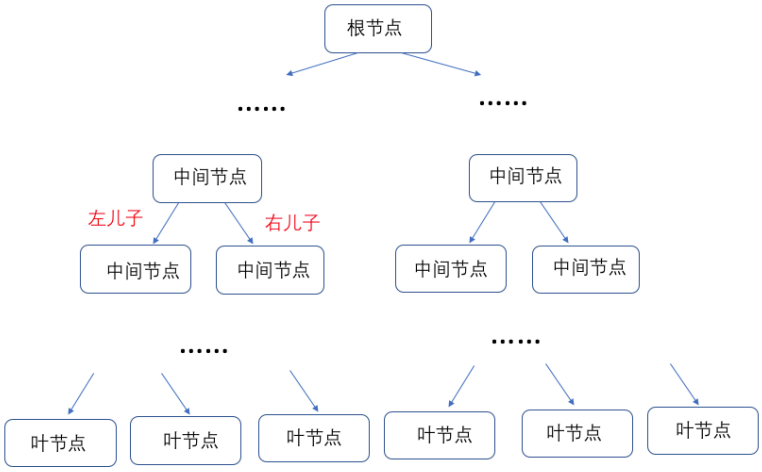
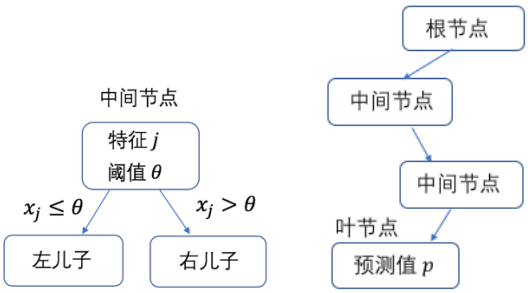

根节点
.....
.....
中间节点
中间节点
左儿子
右儿子
中间节点
中间节点
中间节点
中间节点
.....
.....
叶节点
叶节点
叶节点
叶节点
叶节点
叶节点

图 7.2　二叉树结构



(a) (b)

图 7.3　决策树及搜索步骤图示

决策树节点数据结构

Node:

j：特征下标

θ: 阈值

p：标签预测值

$left$：左儿子

$right$：右儿子

决策树模型的递归算法

$T(node, \boldsymbol{x})$:

If $node.left = NULL$ and $node.right = NULL$：

Return $node.p$

Else：

$j = node.j$

If $x_j \leq node.\theta$:

Return $T(node.left, \boldsymbol{x})$

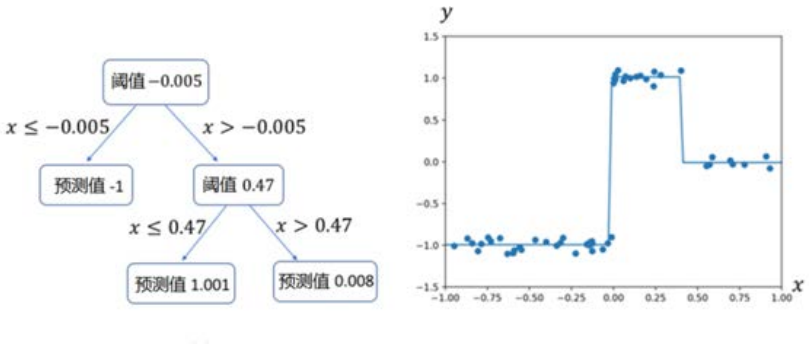
Else:

Return $T(node.right, \boldsymbol{x})$

决策树模型

$h(\boldsymbol{x}) = T(root, \boldsymbol{x})$

图 7.4　决策树模型



(a) (b)

图 7.5　例 7.1 问题的决策树模型及其拟合效果

决策树回归算法

输入：m 个训练数据 $S=\{(\boldsymbol{x}^{(1)},y^{(1)}),\ (\boldsymbol{x}^{(2)},y^{(2)}),\dots,\ (\boldsymbol{x}^{(m)},y^{(m)})\}$

模型假设：$H_{tree}=\{\,h(\boldsymbol{x}):h\text{ 为决策树模型}\,\}$

$$\min_{h\in H_{tree}}\frac{1}{m}\sum_{i=1}^{m}\left(h(\boldsymbol{x}^{(i)})-y^{(i)}\right)^2$$

图 7.6　决策树回归算法描述

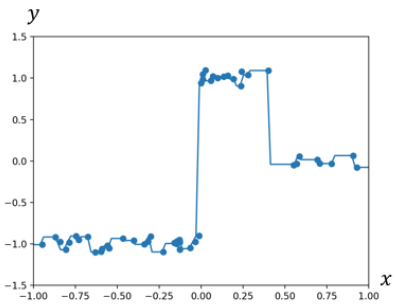


图 7.7　例 7.1 问题的过度拟合

决策树回归算法（带深度限制）

输入：m 个训练数据 $S=\left\{\left(\boldsymbol{x}^{(1)},y^{(1)}\right),\ \left(\boldsymbol{x}^{(2)},y^{(2)}\right),\ldots,\ \left(\boldsymbol{x}^{(m)},y^{(m)}\right)\right\}$

模型假设：$H_{tree}^{d}=\left\{h(\boldsymbol{x}):h\text{ 为深度不超过 }d\text{ 的决策树}\right\}$

$$\min_{h\in H_{tree}^{d}}\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right)^{2}$$

图 7.8　限制深度的决策树回归算法描述

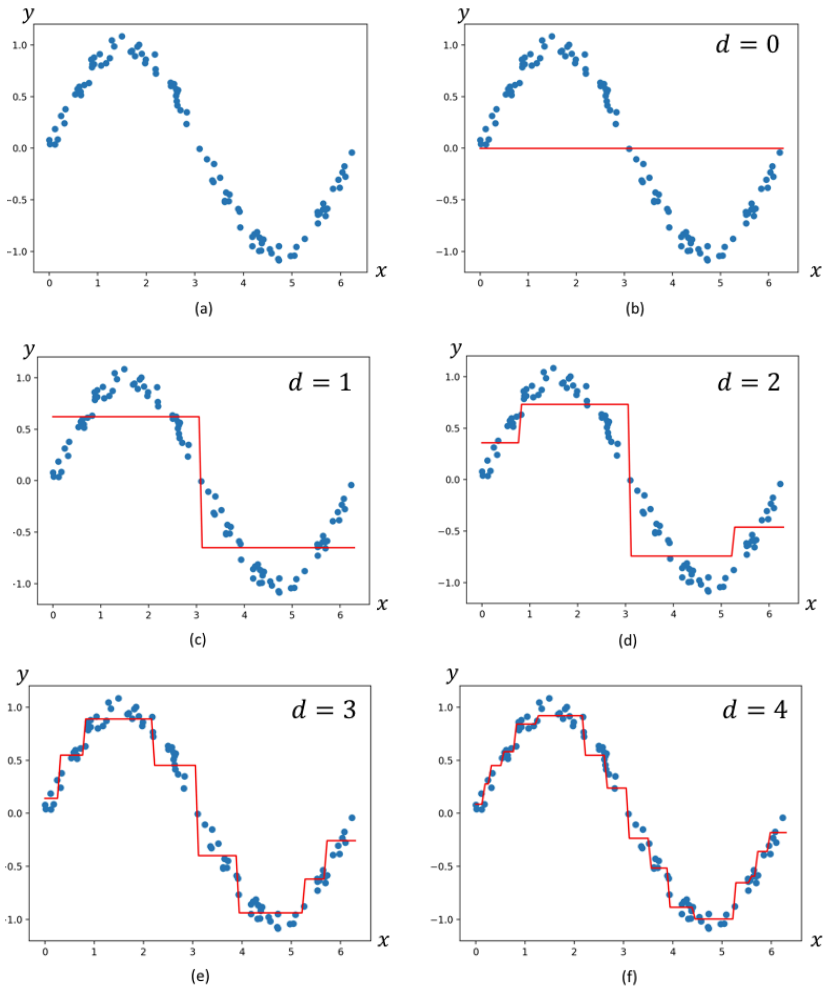
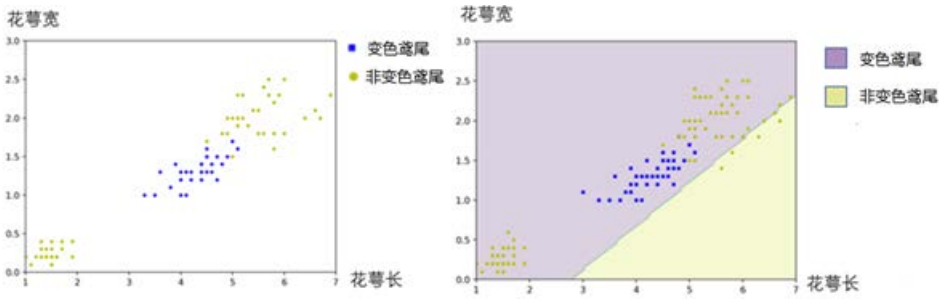


图 7.9　训练数据分布及决策树模型拟合



(a) (b)

图 7.10　训练数据正负采样分布及 Logistic 回归拟合

决策树分类算法（带深度限制）

任务：k 元分类问题的概率预测任务

输入：m 个训练数据 $S=\{(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}),\ (\boldsymbol{x}^{(2)},\boldsymbol{y}^{(2)}),\ldots,\ (\boldsymbol{x}^{(m)},\boldsymbol{y}^{(m)})\}$

模型假设：$H_{tree}^{d}=\{\,h(\boldsymbol{x}):h$ 为深度不超过 d 的决策树 $\}$

$$\min_{h\in H_{tree}^{d}} -\frac{1}{m}\sum_{i=1}^{m}\langle \boldsymbol{y}^{(i)},\log h(\boldsymbol{x}^{(i)})\rangle$$

图 7.11　概率预测任务的决策树分类算法

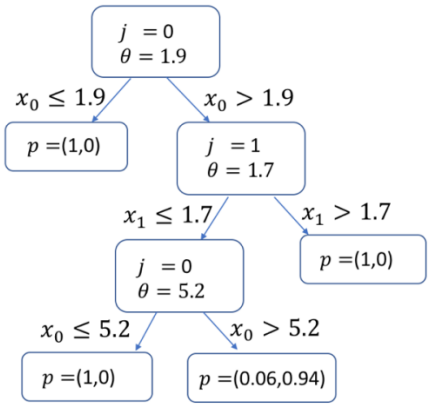


图 7.12　变色鸢尾识别问题的决策树

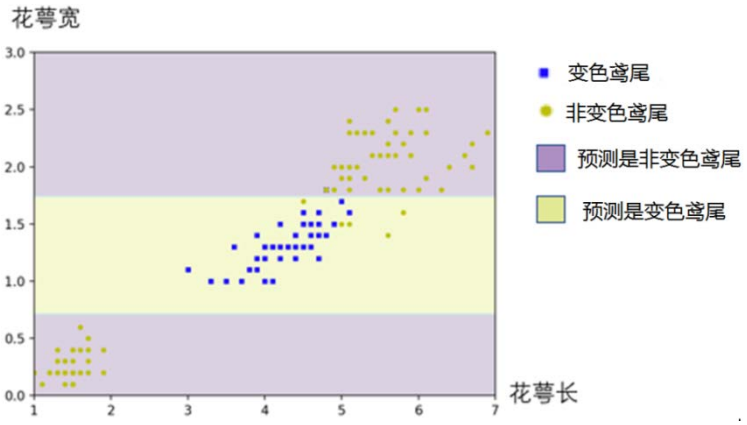


图 7.13　变色鸢尾花预测的决策树模型

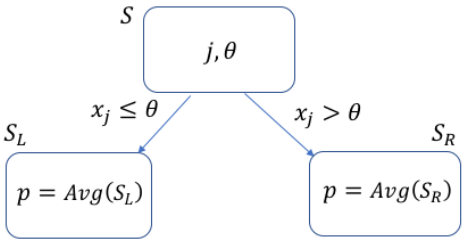


图 7.14 深度为 1 的决策树

决策树回归的 CART 算法

$GenerateTree(S, d)$:

$root =$ new Node

If $d = 0$ Or $|S| < 2$:

$root.p = Avg(S)$

Else:

For $j = 1,2, \ldots, n$:

For θ in $\{x_j: \boldsymbol{x} \in S\}$:

$S_L(j, \theta) = \{(\boldsymbol{x}, y) \in S : x_j \leq \theta\}$

$S_R(j, \theta) = \{(\boldsymbol{x}, y) \in S: x_j > \theta\}$

$MSE(j, \theta) = \frac{|S_L(j,\theta)|}{|S|} Var(S_L(j, \theta)) + \frac{|S_R((j,\theta)|}{|S|} Var(S_R(j, \theta))$

$j^*, \theta_j^* = \text{argmax}_{j,\theta}\, MSE(j, \theta)$

$root.j = j^*$

$root.\theta = \theta^*$

$root.left = GenerateTree(S_L(j^*, \theta^*), d - 1)$

$root.right = GenerateTree(S_R(j^*, \theta^*), d - 1)$

Return $root$

图 7.15　决策树回归的 CART 算法

决策树分类的 CART 算法

$GenerateTree(S, d)$:

$root =$ new node

If $d = 0$ Or $|S| < 2$:

$root.p = Avg(S)$

Else:

For $j = 1,2, \ldots, n$:

For θ in $\{x_j: \boldsymbol{x} \in S\}$:

$S_L(j, \theta) = \{(\boldsymbol{x}, \boldsymbol{y}) \in S : x_j \leq \theta\}$

$S_R(j, \theta) = \{(\boldsymbol{x}, \boldsymbol{y}) \in S: x_j > \theta\}$

$CE(j, \theta) = \frac{|S_L(j,\theta)|}{|S|} Entropy(S_L(j, \theta)) + \frac{|S_R(j,\theta)|}{|S|} Entropy(S_R(j, \theta))$

$j^*, \theta_j^* = \text{argmax}_{j,\theta}\, CE(j, \theta)$

$root.j = j^*$

$root.\theta = \theta^*$

$root.left = GenerateTree(S_L(j^*, \theta^*), d - 1)$

$root.right = GenerateTree(S_R(j^*, \theta^*), d - 1)$

Return $root$

图 7.16 决策树分类的 CART 算法

machine_learning.lib.tree_node

```
1  class Node:
2      j = None
3      theta = None
4      p = None
5      left = None
6      right = None
```

图 7.17　决策树节点数据结构

machine_learning.lib.decision_tree_base

```
1  import numpy as np
2  from machine_learning.lib.tree_node import Node
3
4  class DecisionTreeBase:
5      def __init__(self, max_depth, get_score, feature_sample_rate=1.0):
6          self.max_depth = max_depth
7          self.get_score = get_score
8          self. feature_sample_rate = feature_sample_rate
9
10     def split_data(self, j, theta, X, idx):
11         idx1, idx2 = list(), list()
12         for i in idx:
13             if X[i][j] <= theta:
14                 idx1.append(i)
15             else:
16                 idx2.append(i)
17         return idx1, idx2
18
19       def get_random_features(self, n):
20         shuffled = np.random.permutation(n)
21         size = int(self.feature_sample_rate * n)
22         return shuffled[:size]
23
24     def find_best_split(self, X, y, idx):
25         m, n = X.shape
26         best_score, best_j, best_theta = float("inf"), -1, float("inf")
27         best_idx1, best_idx2 = list(), list()
28         selected_j = self.get_random_features(n)
29         for j in selected_j:
30             thetas = set([x[j] for x in X])
31             for theta in thetas:
32                 idx1, idx2 = self.split_data(j, theta, X, idx)
33                 if min(len(idx1), len(idx2)) == 0:
34                     continue
35                 score1, score2 = self.get_score(y, idx1), self.get_score(y, idx2)
36                 w = 1.0 * len(idx1) / len(idx)
37                 score = w * score1 + (1 - w) * score2
```

```
38                          if score < best_score:
39                              best_score, best_j, best_theta = score, j, theta
40                              best_idx1, best_idx2 = idx1, idx2
41              return best_j, best_theta, best_idx1, best_idx2, best_score
42
43      def generate_tree(self, X, y, idx, d):
44          r = Node()
45          if d == 0 or len(idx) == 1:
46              r.p = np.average(y[idx], axis=0)
47              return r
48          j, theta, idx1, idx2, score = self.find_best_split(X, y, idx)
49          current_score = self.get_score(y, idx)
50          if score >= current_score:
51              return r
52          r.j, r.theta = j, theta
53          r.left, r.right = self.generate_tree(X, y, idx1, d-1), self.generate_tree(X, y, idx2,
d-1)
54          return r
55
56      def fit(self, X, y):
57          self.root = self.generate_tree(X, y, range(len(X)), self.max_depth)
58
59      def get_prediction(self, r, x):
60              if r.left == None and r.right == None:
61                  return r.p
62              if x[r.j] <= r.theta:
63                  return self.get_prediction(r.left, x)
64              else:
65                  return self.get_prediction(r.right, x)
66
67      def predict(self, X):
68              y = list()
69              for i in range(len(X)):
70                  y.append(self.get_prediction(self.root, X[i]))
71              return np.array(y)
```

图 7.18　决策树 CART 算法的基类

machine_learning.lib.decision_tree_regressor

```
1  import numpy as np
2  from machine_learning.lib.decision_tree_base import DecisionTreeBase
3
4  def get_var(y, idx):
5      y_avg = np.average(y[idx]) * np.ones(len(idx))
6      return np.linalg.norm(y_avg - y[idx], 2) ** 2 / len(idx)
7
8  class DecisionTreeRegressor(DecisionTreeBase):
9      def __init__(self, max_depth=0, feature_sample_rate=1.0):
10         super().__init__(
11             max_depth = max_depth,
12             feature_sample_rate = feature_sample_rate,
13             get_score = get_var)
```

图 7.19　基于 CART 算法基类的决策树回归算法

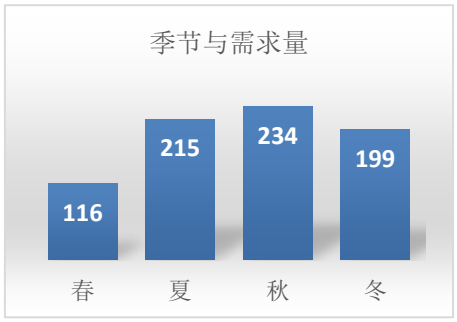


图 7.20　四个季节的共享单车需求量

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from machine_learning.lib.decision_tree_regressor import DecisionTreeRegressor
from sklearn.metrics import r2_score

def get_data():
    df = pd.read_csv("./bike.csv")
    df.datetime = df.datetime.apply(pd.to_datetime)
    df['hour'] = df.datetime.apply(lambda x: x.hour)
    y = df['count'].values
    df.drop(['datetime','casual','registered','count'], 1, inplace = True)
    X = df.values
    return X, y

X, y = get_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
model = DecisionTreeRegressor(max_depth = 2)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print('r2= {}'.format(get_r2(y_test, y_pred)))
```

图 7.21 共享单车需求量问题的决策树回归算法

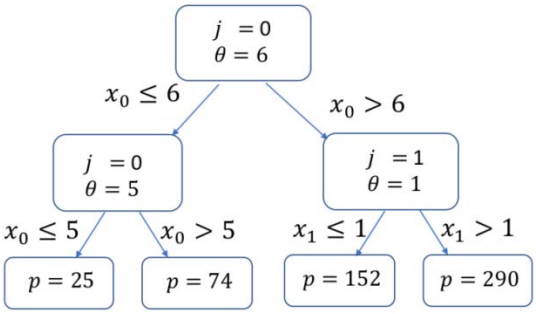


图 7.22　共享单车需求量问题的决策树模型

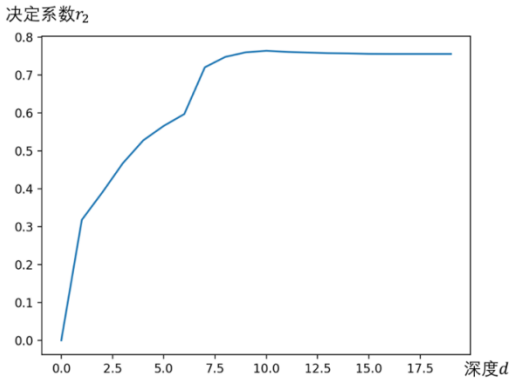


图 7.23　决策树模型的决定系数与深度之间的关系

machine_learning.lib.decision_tree_classifier

```
1  import numpy as np
2  from machine_learning.lib.decision_tree_base import DecisionTreeBase
3
4  def get_entropy(y, idx):
5      _ , k = y.shape
6      p = np.average(y[idx], axis=0)
7      return - np.log(p + 0.001 * np.random.rand(k)).dot(p.T)
8
9  class DecisionTreeClassifier(DecisionTreeBase):
10     def __init__(self, max_depth=0, feature_sample_rate=1.0):
11         super().__init__(max_depth = max_depth,
12                 feature_sample_rate = feature_sample_rate,
13                 get_score = get_entropy)
14
15     def predict_proba(self, X):
16         return super().predict(X)
17
18     def predict(self, X):
19         proba = self.predict_proba(X)
20         return np.argmax(proba, axis=1)
```

图 7.24　基于 CART 算法基类的决策树分类算法

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from machine_learninglib.decision_tree_classifier import DecisionTreeClassifier

def get_data():
    df = pd.read_csv("./voice.csv")
    y = (df['label'].values=='male').astype(np.int)
    df.drop(['label'], 1, inplace = True)
    X = df.values
    return X, y.reshape(-1,1)

X, y = get_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
encoder = OneHotEncoder()
y_train = encoder.fit_transform(y_train).toarray()
model = DecisionTreeClassifier(max_depth = 5)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy = {}".format(accuracy))
```

图 7.25　男女声识别问题的决策树算法

machine_learning.lib.random_forest_classifier

```
1  import numpy as np
2  from machine_learning.lib.decision_tree_classifier import DecisionTreeClassifier
3
4  class RandomForestClassifier:
5      def __init__(self, num_trees,max_depth, feature_sample_rate,
6                   data_sample_rate, random_state = 0):
7          self.max_depth, self.num_trees = max_depth, num_trees
8          self.feature_sample_rate = feature_sample_rate
9          self.data_sample_rate = data_sample_rate
10         self.trees = []
11         np.random.seed(random_state)
12
13     def get_data_samples(self, X, y):
14         shuffled_indices = np.random.permutation(len(X))
15         size = int(self.data_sample_rate * len(X))
16         selected_indices = shuffled_indices[:size]
17         return X[selected_indices], y[selected_indices]
18
19     def fit(self, X, y):
20         for t in range(self.num_trees):
21             X_t, y_t = self.get_data_samples(X, y)
22             model = DecisionTreeClassifier(
23                 max_depth = self.max_depth,
24                 feature_sample_rate = self.feature_sample_rate)
25             model.fit(X_t, y_t)
26             self.trees.append(model)
27
28     def predict(self, X):
29         y = []
30         for i in range(len(X)):
31             preds = [np.asscalar(tree.predict(X[i].reshape(1,-1))) for tree in self.trees]
32             y.append(max(set(preds), key=preds.count))
33         return np.array(y)
```

图 7.26　随机森林分类算法

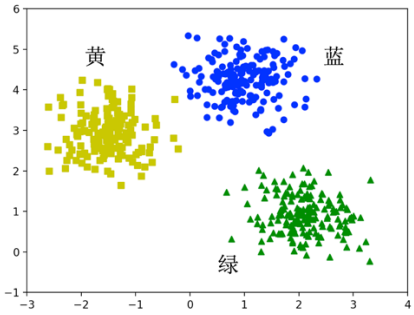
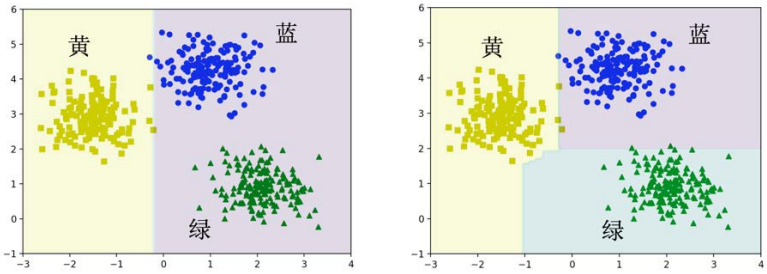


图 7.27　墨渍数据采样

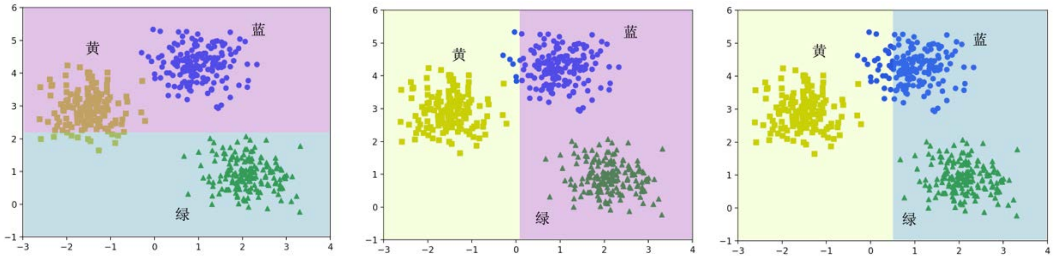
```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from machine_learning.lib.decision_tree_classifier import DecisionTreeClassifier
from machine_learning.lib.random_forest_classifier import RandomForestClassifier
from sklearn.metrics import accuracy_score

X, y = make_blobs(n_samples=1000, centers=3, random_state=0, cluster_std=1.0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
encoder = OneHotEncoder()
y_train = encoder.fit_transform(y_train.reshape(-1,1)).toarray()

tree = DecisionTreeClassifier(max_depth=1)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
print("decision tree accuracy= {}".format(accuracy_score(y_test, y_pred)))

forest = RandomForestClassifier(max_depth=1, num_trees=100,
                                feature_sample_rate=0.5, data_sample_rate=0.1)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)
print("random forest accuracy= {}".format(accuracy_score(y_test, y_pred)))
```

图 7.28　墨渍数据分类的决策树和随机森林算法



(a) (b)

图 7.29　决策树与随机森林的预测结果比较



(a) (b) (c)

图 7.30　随机森林中 3 棵决策树采样的划分结果

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from machine_learning.lib.decision_tree_classifier import DecisionTreeClassifier
from machine_learning.lib.random_forest_classifier import RandomForestClassifier
from sklearn.metrics import accuracy_score

def get_data():
    df = pd.read_csv("./voice.csv")
    y = (df['label'].values=='male').astype(np.int)
    df.drop(['label'], 1, inplace = True)
    X = df.values
    return X, y.reshape(-1,1)

X, y = get_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
encoder = OneHotEncoder()
y_train = encoder.fit_transform(y_train).toarray()

tree = DecisionTreeClassifier(max_depth = 5)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
print("tree accuracy= {}".format(accuracy_score(y_test, y_pred)))

m, n = X.shape
forest = RandomForestClassifier(max_depth = 5, num_trees = 100,
    feature_sample_rate = 1.0 / np.sqrt(n), data_sample_rate = 0.2)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)
print("forest accuracy= {}".format(accuracy_score(y_test, y_pred)))
```

图 7.31　声识别问题的随机森林算法

machine_learning.lib.random_forest_regressor

```
1  import numpy as np
2  from machine_learning.lib.decision_tree_regressor import DecisionTreeRegressor
3
4  class RandomForestRegressor:
5      def __init__(self, num_trees, max_depth, feature_sample_rate,
6                   data_sample_rate, random_state = 0):
7          self.max_depth, self.num_trees = max_depth, num_trees
8          self.feature_sample_rate = feature_sample_rate
9          self.data_sample_rate = data_sample_rate
10         self.trees = []
11         np.random.seed(random_state)
12
13     def get_data_samples(self, X, y):
14         shuffled_indices = np.random.permutation(len(X))
15         size = int(self.data_sample_rate * len(X))
16         selected_indices = shuffled_indices[:size]
17         return X[selected_indices], y[selected_indices]
18
19     def fit(self, X, y):
20         for t in range(self.num_trees):
21             X_t, y_t = self.get_data_samples(X, y)
22             model = DecisionTreeRegressor(max_depth = self.max_depth,
23                                           feature_sample_rate = self.feature_sample_rate)
24             model.fit(X_t, y_t)
25             self.trees.append(model)
26
27     def predict(self,X):
28         preds = np.array([tree.predict(X) for tree in self.trees])
29         return np.average(preds, axis=0)
```

图 7.32　随机森林回归算法

```
import pandas as pd
from sklearn.model_selection import train_test_split
from machine_learning.lib.decision_tree_regressor import DecisionTreeRegressor
from machine_learning.lib.random_forest_regressor import RandomForestRegressor
from sklearn.metrics import r2_score

def get_data():
    df = pd.read_csv("./bike.csv")
    df.datetime = df.datetime.apply(pd.to_datetime)
    df['hour'] = df.datetime.apply(lambda x: x.hour)
    y = df['count'].values
    df.drop(['datetime','casual','registered','count'], 1, inplace = True)
    X = df.values
    return X, y

X, y = get_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=3)

tree = DecisionTreeRegressor(max_depth = 8)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
print("tree r2 = {}".format(r2_score(y_test, y_pred)))

forest = RandomForestRegressor(max_depth = 8, num_trees=100,
                    feature_sample_rate=1.0, data_sample_rate=0.2)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)
print("forest r2= {}".format(r2_score(y_test, y_pred)))
```

图 7.33　共享单车问题的决策树算法和随机森林算法

machine_learning.lib.gbdt

```
1  import numpy as np
2  from machine_learning.lib.decision_tree_regressor import DecisionTreeRegressor
3
4  class GBDT:
5      def __init__(self, num_trees, max_depth):
6          self.max_depth = max_depth
7          self.num_trees = num_trees
8          self.trees = []
9
10     def fit(self, X, y):
11         r = y
12         for t in range(self.num_trees):
13             model = DecisionTreeRegressor(max_depth = self.max_depth)
14             model.fit(X, r)
15             self.trees.append(model)
16             pred = model.predict(X)
17             r = r - pred
18
19     def predict(self,X):
20         preds = np.array([tree.predict(X) for tree in self.trees])
21         return np.sum(preds, axis=0)
```

图 7.34　梯度提升决策树算法

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from machine_learning.lib.decision_tree_regressor import DecisionTreeRegressor
from machine_learning.lib.random_forest_regressor import RandomForestRegressor
from machine_learning.lib.gbdt import GBDT
from sklearn.metrics import r2_score

housing = fetch_california_housing()
X = housing.data
y = housing.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

tree = DecisionTreeRegressor(max_depth = 5)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_train)
print("tree r2 = {}".format(get_r2(y_test, y_pred)))

forest = RandomForestRegressor(max_depth = 5, num_trees = 100)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)
print("forest r2 = {}".format(r2_score(y_test, y_pred)))

gbdt = GBDT(max_depth = 5, num_trees=2)
gbdt.fit(X_train, y_train)
y_pred = gbdt.predict(X_test)
print("gbdt r2 = {}".format(get_r2(y_test, y_pred)))
```

图 7.35　房价预测的梯度提升决策树算法