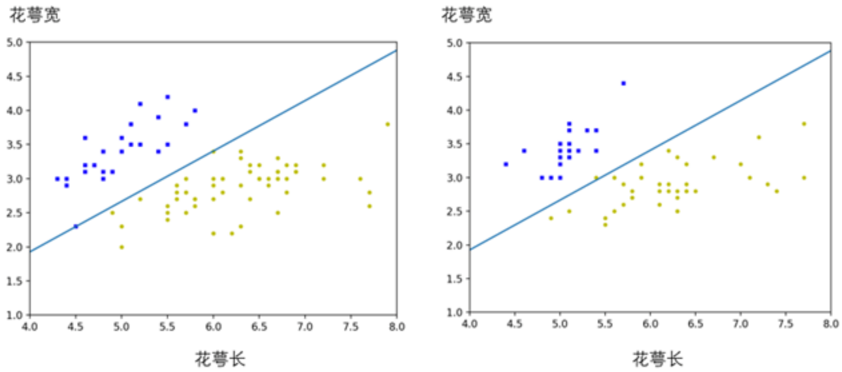
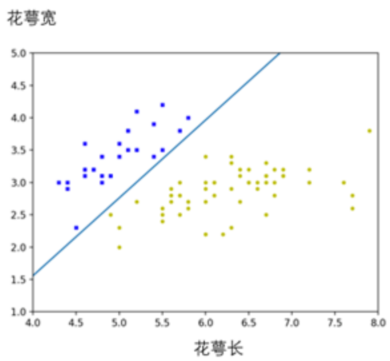
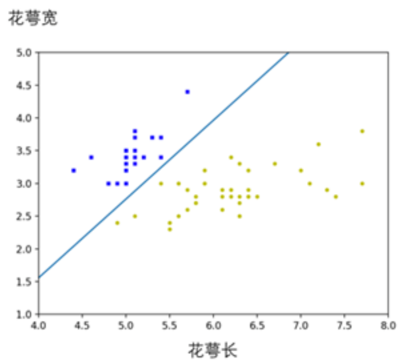


(a) (b)

图 6.1　感知器算法找出的分离直线



(a)



(b)

图 6.2　支持向量机算法找出的分离直线

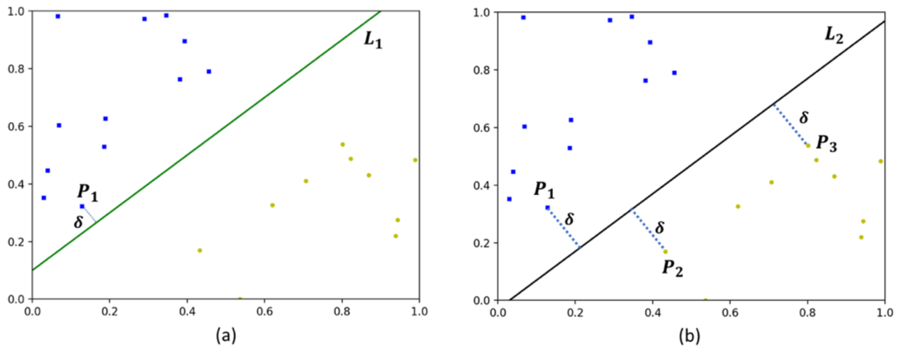


图 6.3　支持向量与间隔

支持向量机算法

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

前提：训练数据中正负采样存在分离平面

模型假设：$H = \{h_{\boldsymbol{w},b} : h_{\boldsymbol{w},b}(\boldsymbol{x}) = \mathrm{Sign}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\}$

计算如下优化问题的最优解 $\boldsymbol{w}^*, b^*$：

$$\min_{w,b} \quad \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{约束:} \quad y^{(i)}(\langle \boldsymbol{w}, \boldsymbol{x}^{(i)} \rangle + b) \geq 1, \quad i = 1,2,\dots,m$$

输出模型 $h_{\boldsymbol{w}^*,b^*}$

图 6.4　支持向量机算法描述

SMO 算法

$\boldsymbol{\lambda} = \mathbf{0}, b = 0$

For each i, j：$K_{i,j} = \langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle$

For $r = 1,2, \dots, N$:

For $i = 1,2, \dots, m$:

For $j = 1,2, \dots, m$:

$$\delta_j = \max\left\{L_{i,j},\ \min\left\{\lambda_j + \frac{E_j - E_i}{2K_{i,j} - K_{i,i} - K_{j,j}}, H_{i,j}\right\}\right\} - \lambda_j$$

$\lambda_j \leftarrow \lambda_j + \delta_j$

$\lambda_i \leftarrow \lambda_i - y^{(i)} y^{(j)} \delta_j$

If $\lambda_i > 0$:

$b = y^{(i)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,i}$

Else If $\lambda_j > 0$:

$b = y^{(j)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,j}$

$\boldsymbol{w} = \sum_{i=1}^{m} \lambda_i y^{(i)} \boldsymbol{x}^{(i)}$

Return $h(\boldsymbol{x}) = \mathrm{Sign}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$

图 6.5　SMO 算法描述

machine_learning.lib.svm_smo

```python
import numpy as np

class SVM:
    def get_H(self, Lambda, i, j, y):
        if y[i] == y[j]:
            return Lambda[i] + Lambda[j]
        else:
            return float("inf")

    def get_L(self, Lambda, i, j, y):
        if y[i] == y[j]:
            return 0.0
        else:
            return max(0, Lambda[j] - Lambda[i])

    def smo(self, X, y, K, N):
        m, n = X.shape
        Lambda = np.zeros((m,1))
        epsilon = 1e-6
        for r in range(N):
            for i in range(m):
                for j in range(m):
                    D_ij = 2 * K[i][j] - K[i][i] - K[j][j]
                    if abs(D_ij) < epsilon:
                        continue
                    E_i = K[:, i].dot(Lambda * y) - y[i]
                    E_j = K[:, j].dot(Lambda * y) - y[j]
                    delta_j = y[j] * (E_j - E_i) / D_ij
                    H_ij = self.get_H(Lambda, i, j, y)
                    L_ij = self.get_L(Lambda, i, j, y)
                    if Lambda[j] + delta_j > H_ij:
                        delta_j = H_ij - Lambda[j]
                        Lambda[j] = H_ij
                    elif Lambda[j] + delta_j < L_ij:
                        delta_j = L_ij - Lambda[j]
                        Lambda[j] = L_ij
                    else:
```

```
                    Lambda[j] += delta_j
                delta_i = - y[i] * y[j] * delta_j
                Lambda[i] += delta_i
                if Lambda[i] > epsilon:
                    b = y[i] - K[:, i].dot(Lambda * y)
                elif Lambda[j] > epsilon:
                    b = y[j] - K[:, j].dot(Lambda * y)
        self.Lambda = Lambda
        self.b = b

    def fit(self, X, y, N = 10):
        K = X.dot(X.T)
        self.smo(X, y, K, N)
        self.w = X.T.dot(self.Lambda * y)

    def predict(self, X):
        return np.sign(X.dot(self.w) + self.b)
```

图 6.6　支持向量机的 SMO 算法

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from machine_learning.lib.svm_smo import SVM
import matplotlib.pyplot as plt

def plot_figure(X, y, model):
    z = np.linspace(4, 8, 200)
    w = model.w
    b = model.b
    L = - w[0] / w[1] * z - b / w[1]
    plt.plot(X[:, 0][y[:, 0]==1], X[:, 1][y[:, 0]==1], "bs")
    plt.plot(X[:, 0][y[:, 0]==-1], X[:, 1][y[:, 0]==-1], "yo")
    plt.plot(z, L)
    plt.show()

iris = datasets.load_iris()
X= iris["data"][:, (0,1)]
y = 2 * (iris["target"]==0).astype(np.int).reshape(-1,1) - 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=5)

model = SVM()
model.fit(X_train, y_train, N=10)
plot_figure(X_train, y_train, model)
plot_figure(X_test, y_test, model)
```

图 6.7 山鸢尾预测问题的支持向量机算法

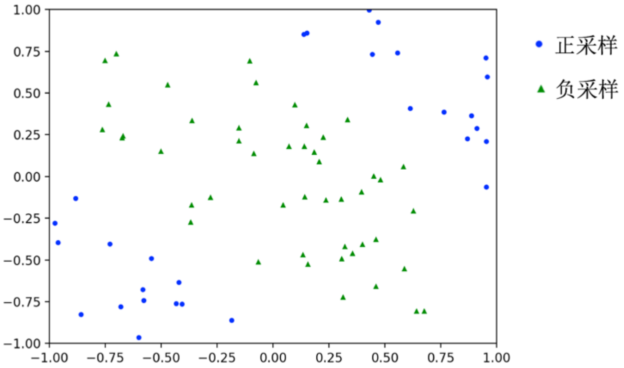


图 6.8　无线性边界的正负采样

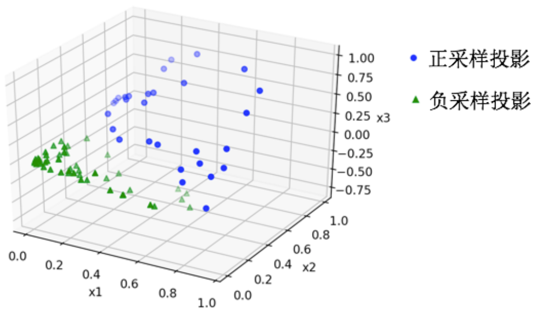
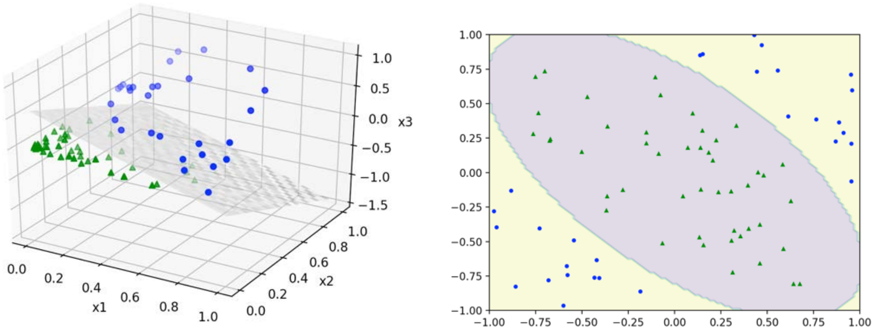


图 6.9　图 6.8 中训练数据在三维空间的投影



(a)　　　　　　　　　　(b)

图 6.10　三维空间中的分离平面及其在二维平面上的椭圆边界

带核函数的 SMO 算法

$\boldsymbol{\lambda} = \mathbf{0}$

For each i, j: $K_{i,j} = K_\phi\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$

For $r = 1,2, \ldots, N$:

For $i = 1,2, \ldots, m$:

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

$\delta_j = \max\left\{L_{i,j}, \min\left\{\lambda_j + \frac{E_j - E_i}{2K_{i,j} - K_{i,i} - K_{j,j}}, H_{i,j}\right\}\right\} - \lambda_j$

$\lambda_j \leftarrow \lambda_j + \delta_j$

$\lambda_i \leftarrow \lambda_i - y^{(i)} y^{(j)} \delta_j$

If $\lambda_i > 0$:

$b = y^{(i)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,i}$

Else If $\lambda_j > 0$:

$b = y^{(j)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,j}$

Return $h(\boldsymbol{x}) = \mathrm{Sign}\left(\sum_{t=1}^{m} \lambda_t y^{(t)} K_\phi(\boldsymbol{x}^{(t)}, \boldsymbol{x}) + b\right)$

图 6.11　带核函数的 SMO 算法描述

machine_learning.lib.kernel_svm

```
import numpy as np
from machine_learning.lib.svm_smo import SVM

class KernelSVM(SVM):
    def __init__(self, kernel = None):
        self.kernel = kernel

    def get_K(self, X_1, X_2):
        if self.kernel == None:
            return X_1.dot(X_2.T)
        m1, m2 = len(X_1), len(X_2)
        K = np.zeros((m1, m2))
        for i in range(m1):
            for j in range(m2):
                K[i][j] = self.kernel(X_1[i], X_2[j])
        return K

    def fit(self, X, y, N=10):
        K = self.get_K(X, X)
        self.smo(X, y, K, N)
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        K = self.get_K(X, self.X_train)
        return np.sign(K.dot(self.Lambda * self.y_train) + self.b)
```

图 6.12　带核函数的 SMO 算法

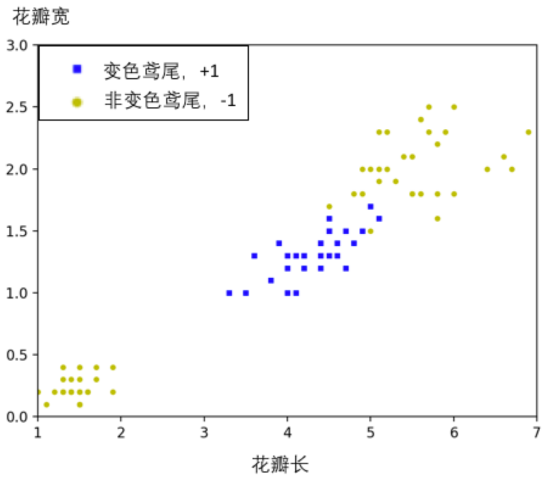


图 6.13　变色鸢尾识别问题的训练数据

```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from machine_learning.lib.kernel_svm import KernelSVM

def rbf_kernel(x1, x2):
    sigma = 1.0
    return np.exp(-np.linalg.norm(x1 - x2, 2) ** 2 / sigma)

iris = datasets.load_iris()
X= iris["data"][:,(2,3)]
y = 2 * (iris["target"]==1).astype(np.int).reshape(-1,1) - 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=5)
model = KernelSVM(kernel = rbf_kernel)
model.fit(X_train, y_train)

x0s = np.linspace(1, 7, 100)
x1s = np.linspace(0, 3, 100)
x0, x1 = np.meshgrid(x0s, x1s)
W = np.c_[x0.ravel(), x1.ravel()]
u = model.predict(W).reshape(x0.shape)
plt.plot(X_train[:, 0][y_train[:, 0]==1] , X_train[:, 1][y_train[:, 0]==1], "bs")
plt.plot(X_train[:, 0][y_train[:, 0]==-1], X_train[:, 1][y_train[:, 0]==-1], "yo")
plt.contourf(x0, x1, u, alpha=0.2)
plt.show()
```

图 6.14　变色鸢尾预测问题的核方法

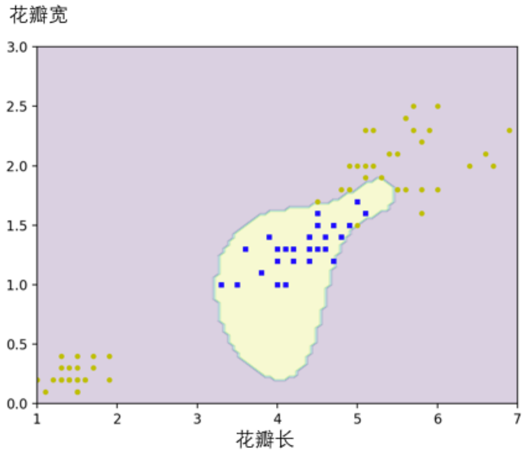


图 6.15　高斯核函数输出的鸢尾花正负采样的非线性边界

machine_learning.lib.soft_svm_smo

```
from machine_learning.lib.svm_smo import SVM

class SoftSVM(SVM):
    def __init__(self, C = 1000):
        self.C = C

    def get_H(self, Lambda, i, j, y):
        C = self.C
        if y[i] == y[j]:
            return min(C, Lambda[i] + Lambda[j])
        else:
            return min(C, C + Lambda[j] - Lambda[i])

    def get_L(self, Lambda, i, j, y):
        if y[i] == y[j]:
            return max(0, Lambda[i] + Lambda[j] - self.C)
        else:
            return max(0, Lambda[j] - Lambda[i])
```

图 6.16　软间隔支持向量机算法

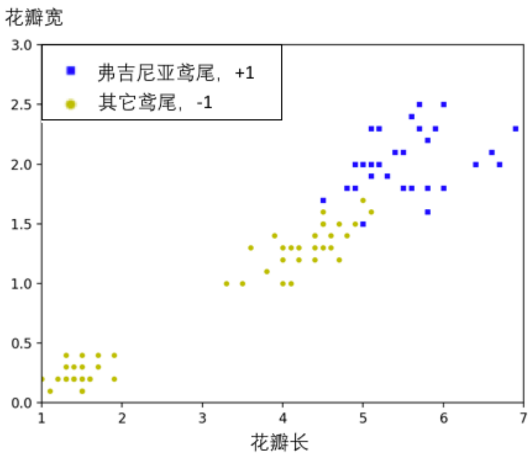


图 6.17　维吉尼亚鸢尾识别问题的训练数据分布

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from machine_learning.lib.soft_svm_smo import SoftSVM
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]
y = 2 * (iris["target"]==2).astype(np.int).reshape(-1,1) - 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=5)

model = SoftSVM(C=5.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy= {}".format(accuracy))
```

图 6.18　维吉尼亚鸢尾识别的软间隔支持向量机算法

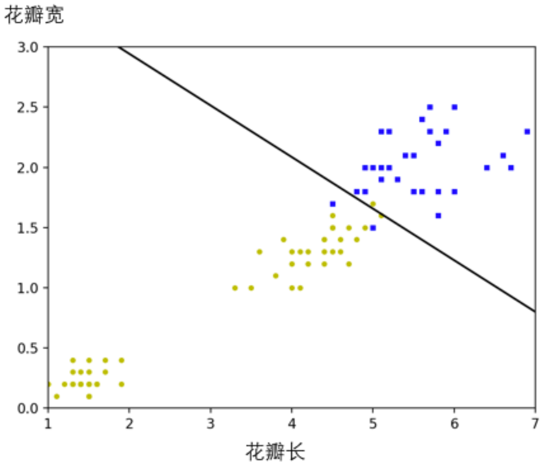


图 6.19　软间隔支持向量机算法输出的近似分离直线

machine_learning.lib.soft_svm_gd

```python
import numpy as np

class SoftSVM:
    def __init__(self, C = 1000):
        self.C = C

    def fit(self, X, y, eta=0.01, N=5000):
        m, n = X.shape
        w, b = np.zeros((n,1)), 0
        for r in range(N):
            s = (X.dot(w) + b) * y
            e = (s < 1).astype(np.int).reshape(-1,1)
            g_w = - 1 / m * X.T.dot(y * e) + 1 / (m * self.C ) * w
            g_b = - 1 / m * (y * e).sum()
            w = w - eta * g_w
            b = b - eta * g_b
        self.w = w
        self.b = b

    def predict(self, X):
        return np.sign(X.dot(self.w)+self.b)
```

图 6.20 软间隔支持向量机次梯度下降算法