线性回归算法

样本空间 $X \subseteq \mathbb{R}^n$

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

输出：线性模型 $h_{\boldsymbol{w}^*, b^*}(\boldsymbol{x})=\langle \boldsymbol{w}^*, \boldsymbol{x}\rangle+b^*$ ，使得 $\boldsymbol{w}^*, b^*$ 为如下优化问题的最优解

$$\min_{\boldsymbol{w} \in \mathbb{R}^n, b \in \mathbb{R}} \frac{1}{m} \sum_{i=1}^{m}\left(\langle \boldsymbol{w}, \boldsymbol{x}^{(i)}\rangle+b-y^{(i)}\right)^2$$

图 3.1　线性回归算法描述

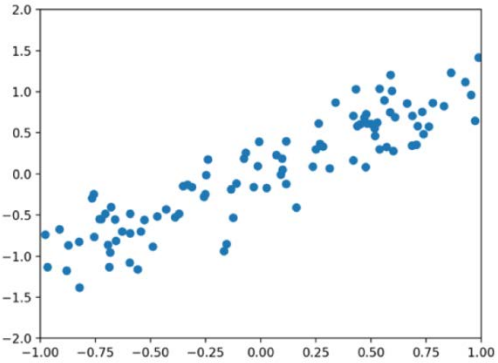


图 3.2　100 个训练数据的散点图

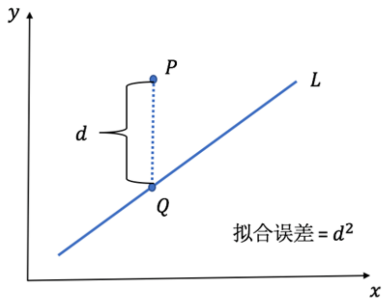


图 3.3　点 P的拟合误差

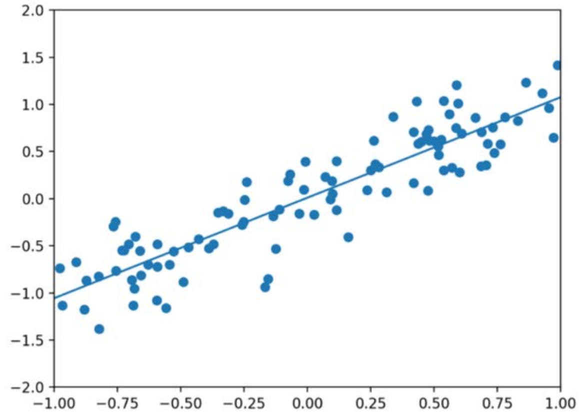


图 3.4　线性回归拟合训练数据

线性回归算法（简化记号）

样本空间 $X \subseteq \mathbb{R}^n$，每个样本 $\boldsymbol{x} \in X$ 首位是 1

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

输出：线性模型 $h(\boldsymbol{x}) = \langle \boldsymbol{w}^*, \boldsymbol{x} \rangle$，使得 $\boldsymbol{w}^*$ 为如下优化问题的最优解

$$\min_{\boldsymbol{w} \in \mathbb{R}^n} \frac{1}{m} \sum_{i=1}^{m} \left(\langle \boldsymbol{w}, \boldsymbol{x}^{(i)} \rangle - y^{(i)}\right)^2$$

图 3.5　简化记号的线性回归算法

平均值模型

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

输出：计算 $\bar{y}=\frac{1}{m}\sum_{i=1}^{m} y^{(i)}$，输出常数模型 h_{avg}，其中 $h_{\mathrm{avg}}(\boldsymbol{x})=\bar{y}$，$\forall \boldsymbol{x} \in X$

图 3.6　平均值模型

machine_learning.lib.linear_regression

```python
import numpy as np

class LinearRegression:
    def fit(self, X, y):
        self.w = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

    def predict(self, X):
        return X.dot(self.w)

def mean_squared_error(y_true, y_pred):
    return np.average((y_true - y_pred)**2, axis=0)

def r2_score(y_true, y_pred):
    numerator = (y_true - y_pred)**2
    denominator = (y_true - np.average(y_true, axis=0))**2
    return 1- numerator.sum(axis=0) / denominator.sum(axis=0))
```

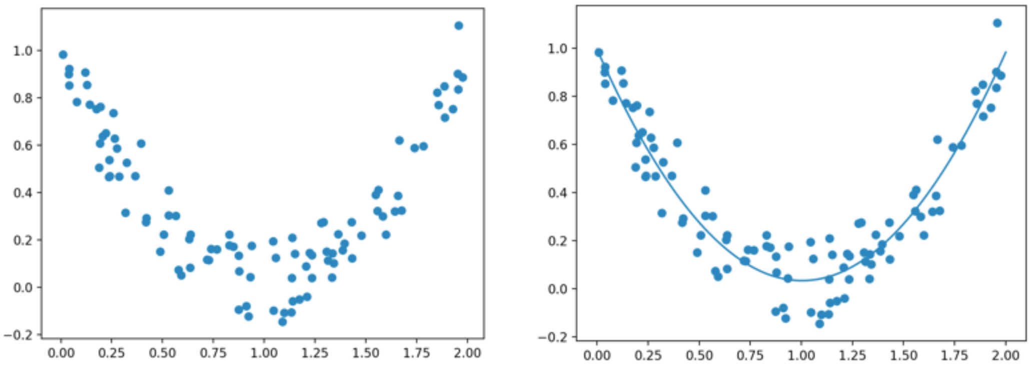
图 3.7　线性回归的正规方程算法

```python
import numpy as np
import machine_learning.lib.linear_regression as lib

def generate_samples(m):
    X = 2 * (np.random.rand(m, 1) - 0.5)
    y = X + np.random.normal(0, 0.3, (m, 1))
    return X, y

def process_features(X):
    m, n = X.shape
    X = np.c_[np.ones((m, 1)), X]
    return X

np.random.seed(0)
X_train, y_train = generate_samples(100)
X_train = process_features(X_train)
X_test, y_test = generate_samples(100)
X_test = process_features(X_test)

model = lib.LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = lib.mean_squared_error(y_test, y_pred)
r2 = lib.r2_score(y_test, y_pred)
print("mse = {} and r2 = {}".format(mse, r2))
```

图 3.8　散点的直线拟合

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
import machine_learning.lib.linear_regression as lib

def process_features(X):
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    m,n = X.shape
    X = np.c_[np.ones((m,1)), X]
    return X

housing = fetch_california_housing()
X = housing.data
y = housing.target.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X_train = process_features(X_train)
X_test = process_features(X_test)

model = lib.LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = lib.mean_squared_error(y_test, y_pred)
r2 = lib.r2_score(y_test, y_pred)
print("mse = {} and r2 = {}".format(mse, r2))
```

图3.9　房价预测问题的线性回归算法



(a)　　　　(b)

图 3.10 多项式模型拟合

```python
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from machine_learning.lib.linear_regression import LinearRegression
import matplotlib.pyplot as plt

def generate_samples(m):
    X = 2 * np.random.rand(m, 1)
    y = X**2 – 2 * X + 1 + np.random.normal(0, 0.1, (m, 1))
    return X, y

np.random.seed(0)
X, y = generate_samples(100)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)

plt.scatter(X, y)
W = np.linspace(0, 2, 300).reshape(300, 1)
W_poly = poly.fit_transform(W)
u = model.predict(W_poly)
plt.plot(W, u)
plt.show()
```

图 3.11　多项式回归

```python
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from machine_learning.lib.linear_regression import LinearRegression
import matplotlib.pyplot as plt

def generate_samples(m):
    X = 2 * (np.random.rand(m, 1) - 0.5)
    y = X + np.random.normal(0, 0.3, (m,1))
    return X, y

np.random.seed(100)
X, y = generate_samples(10)
poly = PolynomialFeatures(degree = 10)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)

plt.axis([-1, 1, -2, 2])
plt.scatter(X, y)
W = np.linspace(-1, 1, 100).reshape(100, 1)
W_poly = poly.fit_transform(W)
u = model.predict(W_poly)
plt.plot(W, u)
plt.show()
```

图 3.12　多项式回归拟合例 2.3 的数据

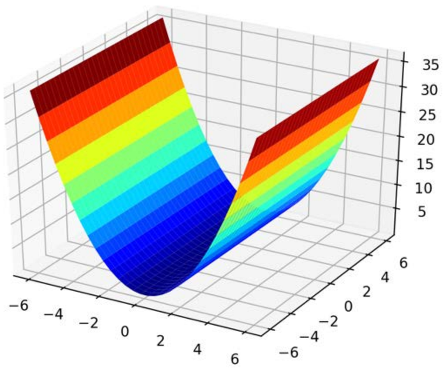


图 3.13　$F(w_1, w_2) = w_1^2$ 的 3 维图像

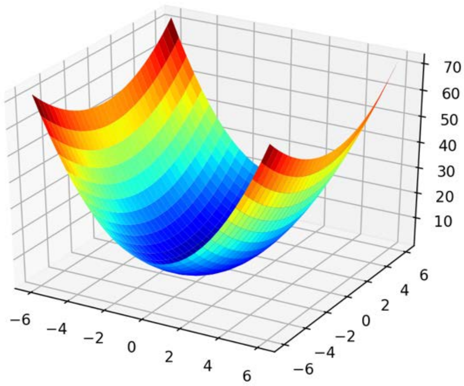


图 3.14　　例 3.7 正则化目标函数的 3 维图像

machine_learning.lib.ridge_regression

```python
import numpy as np

class RidgeRegression:
    def __init__(self, Lambda):
        self.Lambda = Lambda

    def fit(self, X, y):
        m, n = X.shape
        r = m * np.diag(self.Lambda * np.ones(n))
        self.w = np.linalg.inv(X.T.dot(X) + r).dot(X.T).dot(y)

    def predict(self, X):
        return X.dot(self.w)
```

图 3.15　岭回归算法

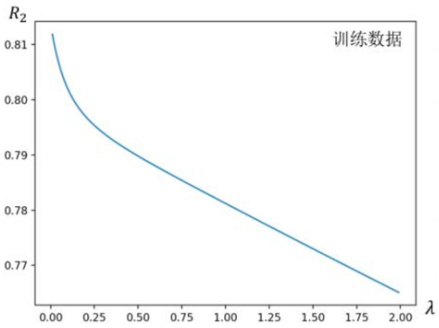
```python
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from machine_learning.lib.ridge_regression import RidgeRegression

def generate_samples(m):
    X = 2 * (np.random.rand(m, 1) - 0.5)
    y = X + np.random.normal(0, 0.3, (m, 1))
    return X, y

np.random.seed(100)
X, y = generate_samples(10)
poly = PolynomialFeatures(degree = 10)
X_poly = poly.fit_transform(X)
model = RidgeRegression(Lambda = 0.01)
model.fit(X_poly, y)

plt.axis([-1, 1, -2, 2])
plt.scatter(X, y)
W = np.linspace(-1, 1, 100).reshape(100, 1)
W_poly = poly.fit_transform(W)
u = model.predict(W_poly)
plt.plot(W, u)
plt.show()
```

图 3.16　多项式模型和岭回归模型

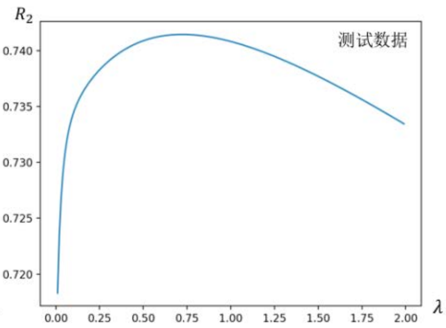
图 3.17 拟合岭回归模型

```python
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
import machine_learning.lib.linear_regression as lib
from machine_learning.lib.ridge_regression import RidgeRegression

def generate_samples(m):
    X = 2 * (np.random.rand(m, 1) - 0.5)
    y = X + np.random.normal(0, 0.3, (m,1))
    return X, y

np.random.seed(100)
poly = PolynomialFeatures(degree = 10)
X_train, y_train = generate_samples(30)
X_train = poly.fit_transform(X_train)
X_test, y_test = generate_samples(100)
X_test = poly.fit_transform(X_test)

Lambdas, train_r2s, test_r2s = [], [], []
for i in range(1, 200):
    Lambda = 0.01 * i
    Lambdas.append(Lambda)
    ridge = RidgeRegression(Lambda)
    ridge.fit(X_train, y_train)
    y_train_pred = ridge.predict(X_train)
    y_test_pred = ridge.predict(X_test)
    train_r2s.append(lib.r2_score(y_train, y_train_pred))
    test_r2s.append(lib.r2_score(y_test, y_test_pred))

plt.figure(0)
plt.plot(Lambdas, train_r2s)
plt.figure(1)
plt.plot(Lambdas, test_r2s)
plt.show()
```

图 3.17 拟合岭回归模型



(a)



(b)

图 3.18　正则化系数与决定系数的关系

向前逐步回归算法

StepwiseRegression($\boldsymbol{X}, \boldsymbol{y}$):

$A = \{1\},\ C = \{2, \dots, n\}$

For $i = 2, \dots, n$:

$mse_A = \min_{\boldsymbol{w} \in \mathbb{R}^{|A|}} \frac{1}{m} \|\boldsymbol{X}_A \boldsymbol{w} - \boldsymbol{y}\|^2$

For $j \in C$:

$mse_{A \cup j} = \min_{\boldsymbol{w} \in \mathbb{R}^{|A|+1}} \frac{1}{m} \left\|\boldsymbol{X}_{A \cup j} \boldsymbol{w} - \boldsymbol{y}\right\|^2$

$j^* = \operatorname{argmin}_{j \in C}\ mse_{A \cup j}$

If $mse_A / mse_{A \cup j^*}$ pass F test：

$A \leftarrow A \cup j^*$

$C \leftarrow C \setminus j^*$

Else：

Break

Return A

图 3.19　向前逐步回归算法描述

machine_learning.lib.stepwise_regression

```python
import numpy as np
from scipy.stats import f

class StepwiseRegression:
    def fit(self, X, y):
        return np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

    def compute_mse(self, X, y):
        w = self.fit(X,y)
        r = y - X.dot(w)
        return r.T.dot(r)

    def f_test(self, mse_A, mse_min, m):
        if mse_min > mse_A:
            return False
        F = mse_A / mse_min
        p_value = f.cdf(F, m, m)
        return p_value > 0.95

    def forward_selection(self, X, y):
        m,n = X.shape
        A, C = [0], [i for i in range(1,n)]
        while len(C) > 0:
            MSE_A = self.compute_mse(X[:, A], y)
            MSE_min, j_min = float("inf"), -1
            j_min = -1
            for j in C:
                MSE_j = self.compute_mse(X[:, A + [j]], y)
                if MSE_j < MSE_min:
                    MSE_min, j_min = MSE_j, j
            if self.f_test(MSE_A, MSE_min, m):
                A.append(j_min)
                C.remove(j_min)
            else:
                break
        self.w = self.fit(X[:, A], y)
        self.A = A

    def predict(self, X):
        return X[:, self.A].dot(self.w)
```

图 3.20　向前逐步回归算法实现

```python
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from machine_learning.lib.stepwise_regression import StepwiseRegression

def generate_samples(m):
    X = 2 * (np.random.rand(m, 1) - 0.5)
    y = X + np.random.normal(0, 0.3, (m,1))
    return X, y

np.random.seed(100)
X, y = generate_samples(10)
poly = PolynomialFeatures(degree = 10)
X_poly = poly.fit_transform(X)
model = StepwiseRegression()
model.forward_selection(X_poly, y)
print(model.A, model.w)
```

图 3.21　向前逐步回归与过度拟合

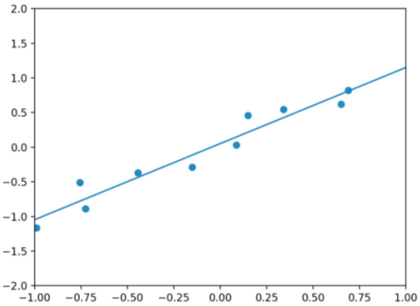


图 3.22　向前逐步回归拟合效果

```python
import numpy as np
from machine_learning.lib.stepwise_regression import StepwiseRegression

X= np.array(
        [[ 0.06,    0.34,    0.03]
        ,[ 0.44,    0.76,    0.28]
        ,[ 0.86,    0.44,    0.20]
        ,[ 0.26,    0.09,     0.25]])
y = np.array([[ 0.42], [ 1.32 ], [ 0.84], [ 0.61]])

model = StepwiseRegression()
model.forward_selection(X, y)
print(model.A, model.w)
```

图 3.23　向前逐步回归得到次优解

分段回归算法

StagewiseRegression($\boldsymbol{X}$, $\boldsymbol{y}$, N, η):

$\boldsymbol{w} = (0,0,\dots,0), t = 0$

While $t < N$:

$\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}$

$j^* = \mathrm{argmax}_{1 \le j \le n} \, |corr(\boldsymbol{X}_j, \boldsymbol{r})|$

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \cdot sign\left(corr(\boldsymbol{X}_{j^*}, \boldsymbol{r})\right) \cdot \boldsymbol{X}_{j^*}$

$t \leftarrow t + 1$

Return $\boldsymbol{w}$

图 3.24　分段回归算法描述

machine_learning.lib.stagewise_regression

```
import numpy as np

class StagewiseRegression:
    def feature_selection(self, X, y, N, eta):
        m, n = X.shape
        norms = np.linalg.norm(X, 2, axis=0).reshape(-1, 1)
        w = np.zeros(n)
        t = 0
        r = y
        while t < N:
            c = X.T.dot(r) / norms
            j_max = np.argmax(abs(c))
            delta = eta * np.sign(c[j_max])
            w[j_max] = w[j_max] + delta
            r = r - delta * X[:, j_max].reshape(-1,1)
            t = t + 1
        self.w = w
        return w

    def predict(self, X):
        return X.dot(self.w)
```

图 3.25　分段回归算法实现

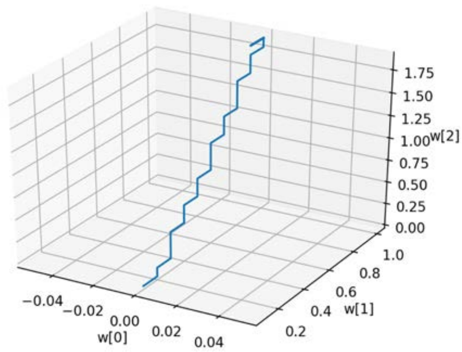


图 3.26　分段回归算法的搜索轨迹