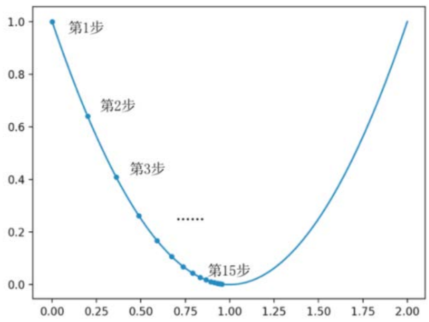


图 4.1 梯度下降算法的搜索过程

梯度下降算法

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

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

$\quad\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla F(\boldsymbol{w})$

Return $\boldsymbol{w}$

图 4.2 梯度下降算法

```python
N, eta = 20, 0.1
w = 0
for t in range(N):
    w = w - eta * 2 * (w - 1)
print(w)
```
图 4.3 目标函数 $(w-1)^2$ 的梯度下降算法

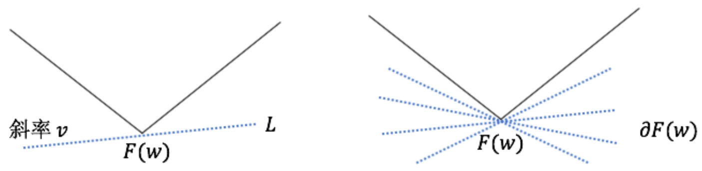
machine_learning.lib.linear_regression_gd

```
import numpy as np

class LinearRegression:
    def fit(self, X, y, eta, N):
        m, n = X.shape
        w = np.zeros((n,1))
        for t in range(N):
            e = X.dot(w) - y
            g = 2 * X.T.dot(e) / m
            w = w - eta * g
        self.w = w

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

图 4.4 线性回归的梯度下降算法



(a) (b)

图 4.5　次梯度集

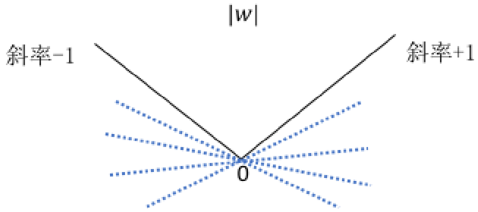


图 4.6　绝对值函数在0点的次梯度集

次梯度下降算法

$\boldsymbol{w} = \boldsymbol{0}$, $\boldsymbol{w}_{sum} = \boldsymbol{0}$

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

  Randomly pick $\boldsymbol{v} \in \partial F(\boldsymbol{w})$

  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{v}$

  $\boldsymbol{w}_{sum} \leftarrow \boldsymbol{w}_{sum} + \boldsymbol{w}$

Return $\overline{\boldsymbol{w}} = \boldsymbol{w}_{sum}/N$

图 4.7 次梯度下降算法

machine_learning.lib.lasso

```python
import numpy as np

class Lasso:
    def __init__(self, Lambda=1):
        self.Lambda = Lambda

    def fit(self, X, y, eta=0.1, N=1000):
        m,n = X.shape
        w = np.zeros((n,1))
        self.w = w
        for t in range(N):
            e = X.dot(w) - y
            v = 2 * X.T.dot(e) / m + self.Lambda * np.sign(w)
            w = w - eta * v
            self.w += w
        self.w /= N

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

图 4.8　Lasso 回归的次梯度下降算法

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

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 = Lasso(Lambda = 0.001)
model.fit(X_poly, y, eta=0.01, N=50000)

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 4.9　多项式模型的 Lasso 回归

随机梯度下降算法

$\boldsymbol{w} = \boldsymbol{0}$, $\boldsymbol{w}_{sum} = \boldsymbol{0}$

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

  Sample $\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \sim S$

  $\eta_t = \dfrac{\eta_0}{\eta_1 + t}$

  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \nabla l\left(h_{\boldsymbol{w}}\left(\boldsymbol{x}^{(i)}\right), \boldsymbol{y}^{(i)}\right)$

  $\boldsymbol{w}_{sum} \leftarrow \boldsymbol{w}_{sum} + \boldsymbol{w}$

Return $\overline{\boldsymbol{w}} = \boldsymbol{w}_{sum}/N$

图 4.10 随机梯度下降算法

machine_learning.lib.linear_regression_sgd

```
import numpy as np

class LinearRegression:
    def fit(self, X, y, eta_0=10, eta_1=50, N=3000):
        m, n = X.shape
        w = np.zeros((n,1))
        self.w = w
        for t in range(N):
            i = np.random.randint(m)
            x = X[i].reshape(1,-1)
            e = x.dot(w) - y[i]
            g = 2 * e * x.T
            w = w - eta_0 * g / (t + eta_1)
            self.w += w
        self.w /= N

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

图 4.11　线性回归的随机梯度下降算法

```python
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from machine_learning.lib.linear_regression_sgd import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

def process_features(X):
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    scaler = MinMaxScaler(feature_range=(-1,1))
    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 = LinearRegression()
model.fit(X_train, y_train, eta_0=10, eta_1=50, K=1000)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("mse = {}, r2 = {}".format(mse, score))
```

图 4.12　房价预测的随机梯度下降算法

```python
import numpy as np
from sklearn.datasets import make_regression
import machine_learning.lib.linear_regression_gd as gd
import machine_learning.lib.linear_regression_sgd as sgd

X, y = make_regression(n_samples=100, n_features=2, noise=0.1, bias=0, random_state=0)
y = y.reshape(-1,1)

model = gd.LinearRegression()
model.fit(X, y, eta=0.01, N=3000)
print(model.w)

model = sgd.LinearRegression()
model.fit(X, y, eta_0=10, eta_1=50, N=3000)
print(model.w)
```

图 4.13 梯度下降与随机梯度下降算法

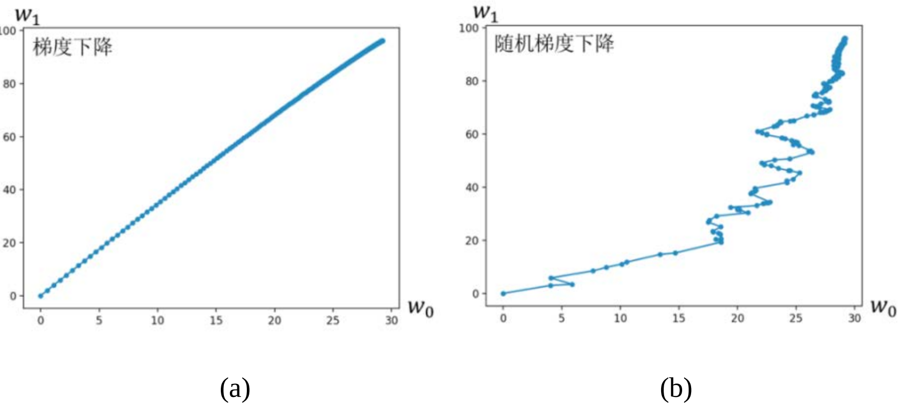


图 4.14 梯度下降与随机梯度下降收敛过程对比

小批量梯度下降算法

$\boldsymbol{w}=\mathbf{0}$, $\boldsymbol{w}_{sum}=\mathbf{0}$

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

Sample $\left(\boldsymbol{x}^{(i_1)},\boldsymbol{y}^{(i_1)}\right)$, $\left(\boldsymbol{x}^{(i_2)},\boldsymbol{y}^{(i_2)}\right),\ldots,\left(\boldsymbol{x}^{(i_B)},\boldsymbol{y}^{(i_B)}\right)\sim$ S

$\eta_t=\frac{\eta_0}{\eta_1+t}$

$\boldsymbol{w}\leftarrow\boldsymbol{w}-\eta_t\frac{1}{B}\sum_{r=1}^{B}\nabla l\left(h_{\boldsymbol{w}}\left(\boldsymbol{x}^{(i_r)}\right),\boldsymbol{y}^{(i_r)}\right)$

$\boldsymbol{w}_{sum}\leftarrow\boldsymbol{w}_{sum}+\boldsymbol{w}$

Return $\overline{\boldsymbol{w}}=\boldsymbol{w}_{sum}/N$

图 4.15　　小批量梯度下降算法

machine_learning.lib.linear_regression_mbgd

```python
import numpy as np

class LinearRegression:
    def fit(self, X, y, eta_0=10, eta_1=50, N=3000, B=10):
        m, n = X.shape
        w = np.zeros((n,1))
        self.w = w
        for t in range(N):
            batch = np.random.randint(low=0, high=m, size=B)
            X_batch = X[batch].reshape(B,-1)
            y_batch = y[batch].reshape(B,-1)
            e = X_batch.dot(w) - y_batch
            g = 2 * X_batch.T.dot(e) / B
            w = w - eta_0 * g / (t + eta_1)
            self.w += w
        self.w /= N

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

图 4.16　线性回归的小批量梯度下降算法

求函数 0 点的牛顿迭代算法

$w = 0$

While $|f(w)| > \epsilon$:

$w \leftarrow w - \frac{f(w)}{f'(w)}$

Return w

图 4.17　求解 $f(w) = 0$ 的牛顿迭代算法

```python
import numpy as np
import matplotlib.pyplot as plt

def f(w):
    return w ** 2

x, y = [], []
epsilon = 0.01
w = -1.5
while abs(f(w)) > epsilon:
    x.append(w)
    y.append(f(w))
    w = w - f(w) / (2 * w)
print(w)
```

图 4.18 求解 $f(w) = w^2$ 的 0 点

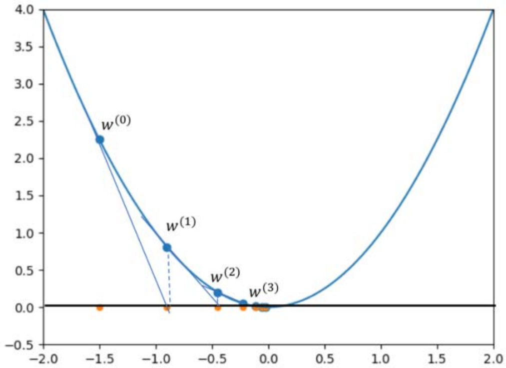


图 4.19　牛顿迭代法计算 $f(w)=w^2$ 的 0 点的过程

牛顿迭代法求解一元优化问题

$w = 0$

While $F'(w) > \epsilon$:

$$w \leftarrow w - \frac{F'(w)}{F''(w)}$$

Return w

图 4.20　一元优化问题的牛顿迭代算法

```python
def F(w):
    return w ** 2 - w + 1

def dF(w):
    return 2 * w - 1

epsilon = 0.01
w = 0
while abs(dF(w)) > epsilon:
    w = w - dF(w) / 2
print(w)
```

图 4.21 计算 $w^2 - w + 1$ 的最小值

牛顿迭代算法求解多元优化问题

$\boldsymbol{w} = \boldsymbol{0}$

While $\|\nabla F(\boldsymbol{w})\| > \epsilon$:

$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \nabla^2 F(\boldsymbol{w})^{-1} \nabla F(\boldsymbol{w})$

Return $\boldsymbol{w}$

图 4.22　多元优化问题的牛顿迭代算法

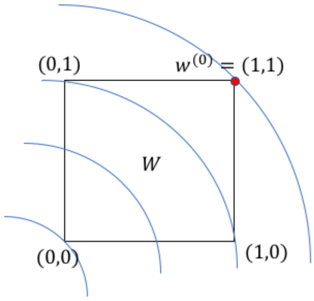


图 4.23　$F(\boldsymbol{w}) = w_1^2 + w_2^2$ 的等高线图

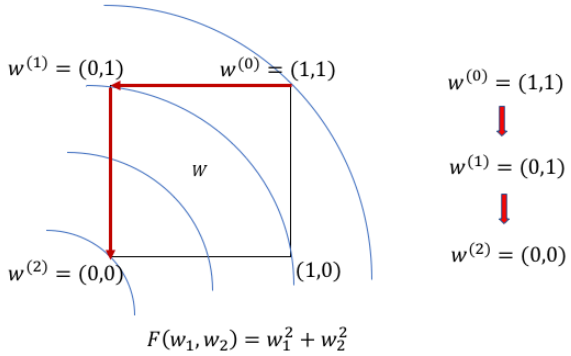


图 4.24　坐标下降算法的搜索过程

坐标下降算法

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

For $t = 1, \ldots, N$:

$j = t \ mod \ n$

$w_j^* = \underset{u \in \mathbb{R}, (u, \boldsymbol{w}_{-j}) \in W}{\operatorname{argmin}} F(u, \boldsymbol{w}_{-j})$

$\boldsymbol{w} \leftarrow (w_j^*, \boldsymbol{w}_{-j})$

Return $\boldsymbol{w}$

图 4.25　坐标下降算法

machine_learning.lib.lasso_cd

```python
import numpy as np

class Lasso:
    def __init__(self, Lambda=1):
        self.Lambda = Lambda

    def soft_threshold(self, t, x):
        if x>t:
            return x - t
        elif x >= -t:
            return 0
        else:
            return x + t

    def fit(self, X, y, N=1000):
        m,n = X.shape
        alpha = 2 * np.sum(X**2, axis=0) / m
        w = np.zeros(n)
        for t in range(N):
            j = t % n
            w[j] = 0
            e_j = X.dot(w.reshape(-1,1)) - y
            beta_j = 2 * X[:, j].dot(e_j) / m
            w[j] = self.soft_threshold(self.Lambda / alpha[j], -beta_j / alpha[j])
        self.w = w

    def predict(self, X):
        return X.dot(self.w.reshape(-1,1))
```

图 4.26　Lasso 回归的坐标下降算法