第3章 线性模型

中国矿业大学 计算机科学与技术学院

3.1基本形式

给定示例$\boldsymbol{x} = (x_1; x_2; \ldots; x_d)$，线性模型(linear model)试图学得一个通过属性的线性组合来进行预测的函数

$$f(\boldsymbol{x}) = \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_d x_d + b$$

向量形式：

$$f(\boldsymbol{x}) = \boldsymbol{\omega}^T \boldsymbol{x} + b$$

$$\boldsymbol{\omega} = (\omega_1; \omega_2; \ldots; \omega_d)$$

简单、容易建模、可解释性好

例1，徐州房价预测。

表3-1 房价数据

编号	面积（m^2）	售价（万元）
1	85	250.93
2	100	293.97
3	120	366.56
4	125	400.10
5	150	471.72
……	……	……

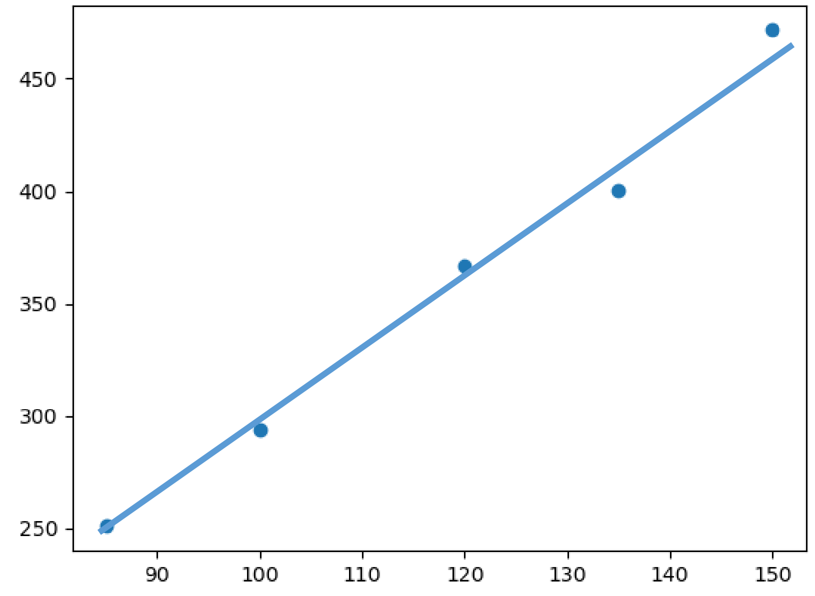


图3-1 样本分布

```
import torch
import matplotlib.pyplot as plt

x = torch.Tensor([85, 100, 120, 135, 150])
y = torch.Tensor([250.93, 293.97, 366.56, 400.10, 471.72])

plt.scatter(x.numpy(), y.numpy())
plt.show()
```

$y = kx + b$ → $y = \omega x + b$ → $y = \omega_1 x + \omega_0$ → $\hat{y} = \omega_1 x + \omega_0$

3.2 线性回归

表3-2 $\hat{y}_i$与y_i对照表

x_i	y_i	$\hat{y}_i$
85	300	$\omega_1 * 85+\omega_0$
100	380	$\omega_1 *100+\omega_0$
120	450	$\omega_1 *120+\omega_0$
125	500	$\omega_1 *125+\omega_0$
150	600	$\omega_1 *150+\omega_0$

- **损失函数 (loss function)**

$$L(\omega_1, \omega_0) = \sum_{i=1}^{5}(\hat{y}_i - y_i)^2 = \sum_{i=1}^{5}(\omega_1 x_i + \omega_0 - y_i)^2$$

$$\omega_1^*, \omega_0^* = \underset{\omega_1,\ \omega_0}{\arg\min}\, L(\omega_1, \omega_0)$$

凸优化问题

- **凸函数(Concave Function)**：对于区间$[a,b]$上定义的函数f，若它对区间中任意两点x_1、x_2均有$f(\frac{x_1+x_2}{2}) \leq \frac{f(x_1)+f(x_2)}{2}$，则称$f$为区间$[a,b]$上的凸函数。
- U形曲线的函数，如$f(x) = x^2$，通常是凸函数。
- 对于实数集上的函数，可通过求二阶导数来判别：若二阶导数在区间上非负，则称为凸函数；若二阶单数在区间上恒大于0，则称为严格凸函数。

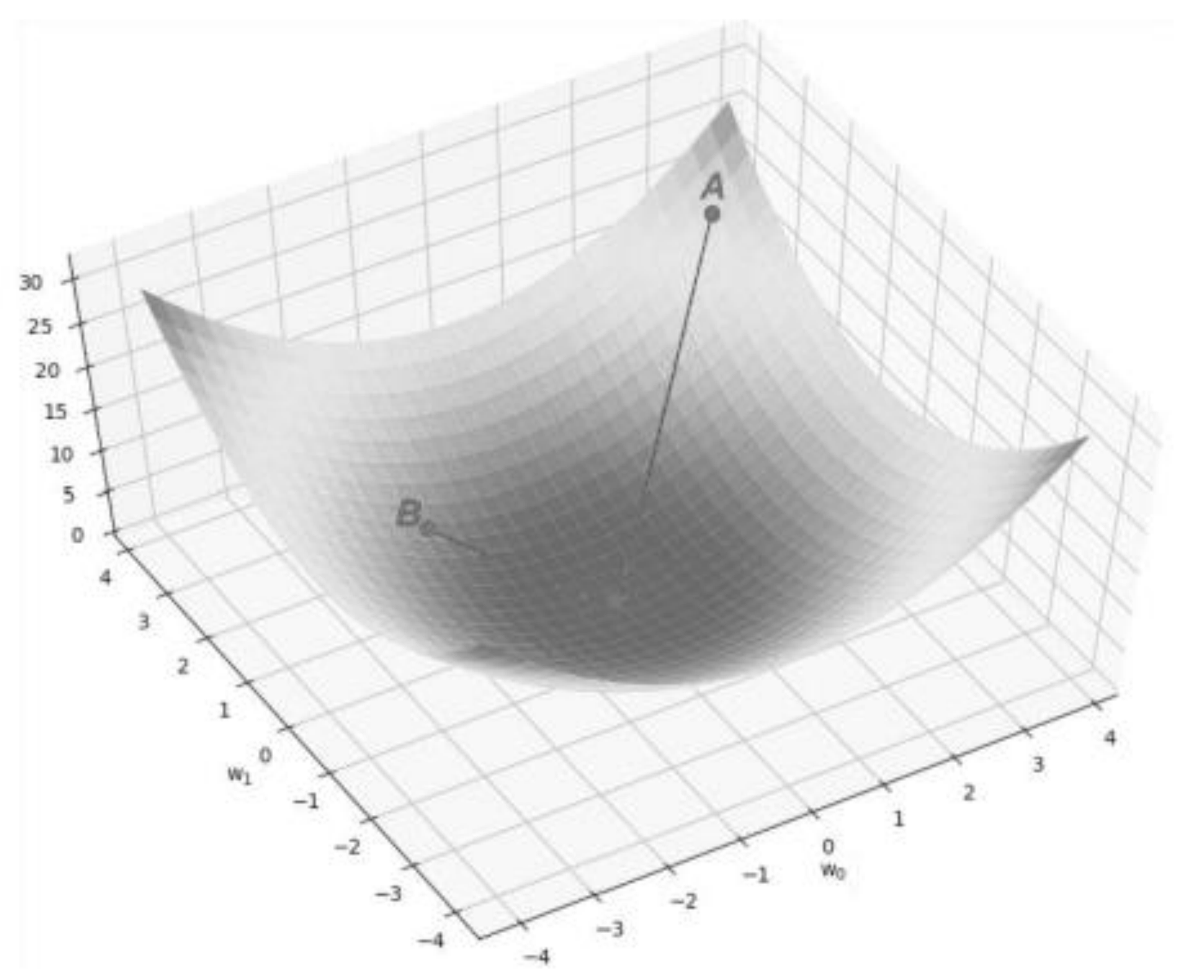


图3-2 损失函数L的三维曲面

3.2 线性回归

损失函数L的梯度

$$\nabla L = (\frac{\partial L}{\partial \omega_1}, \frac{\partial L}{\partial \omega_0})$$

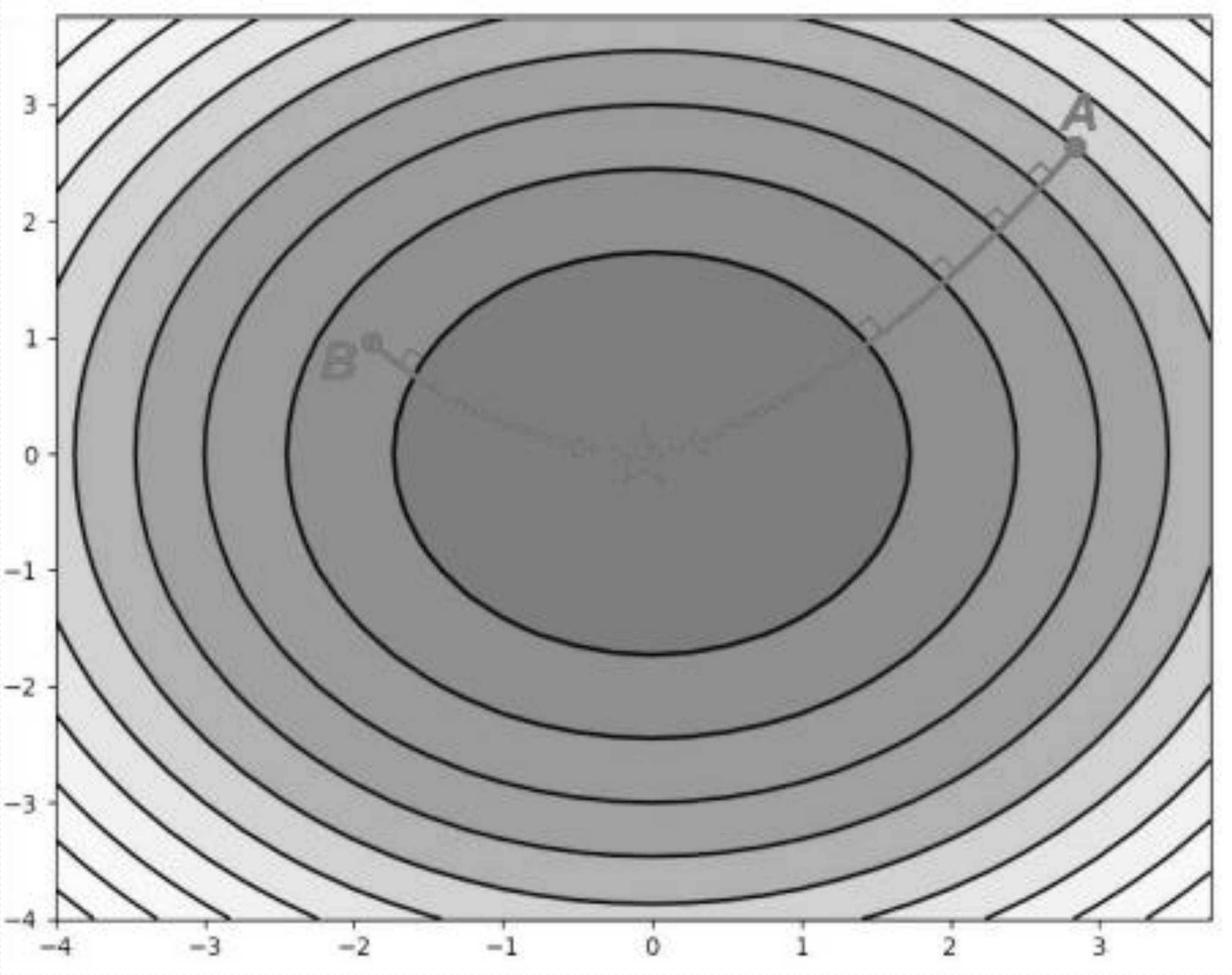


图3-3 梯度下降法

梯度下降法

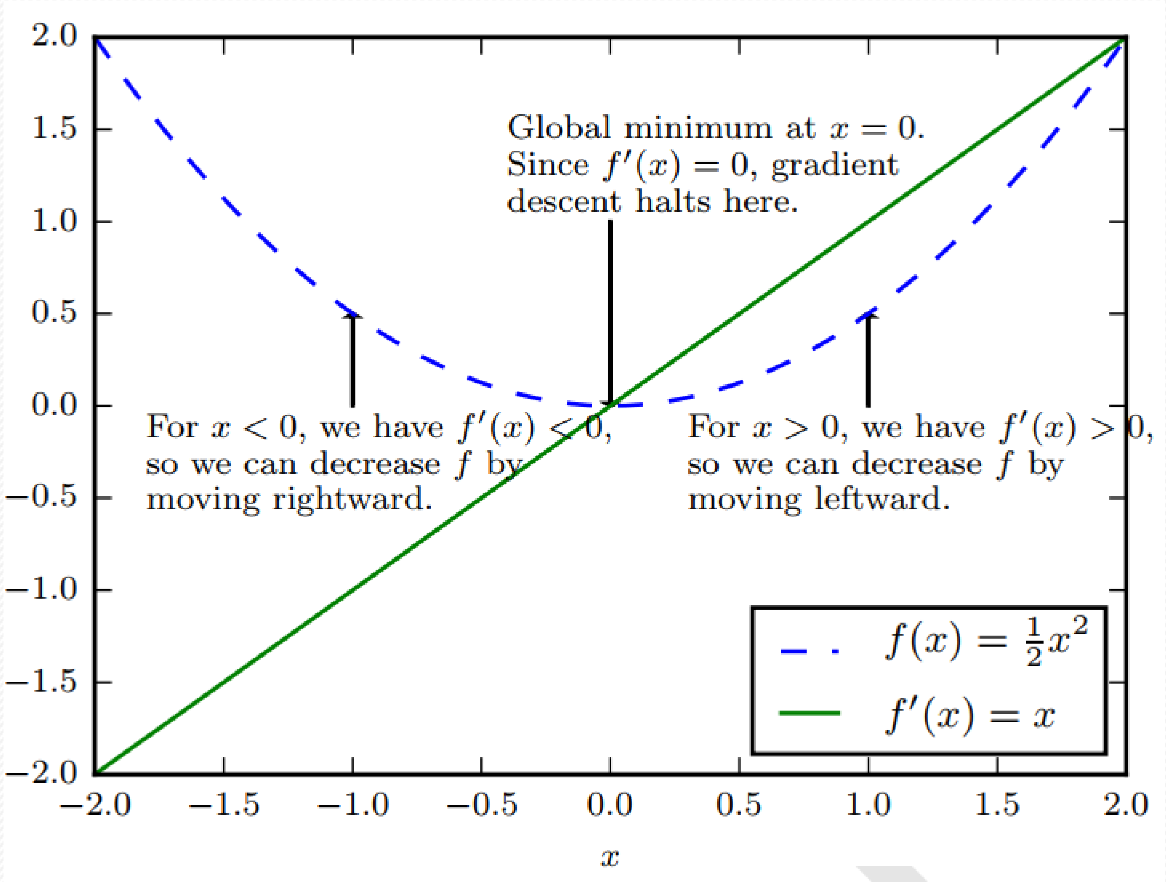
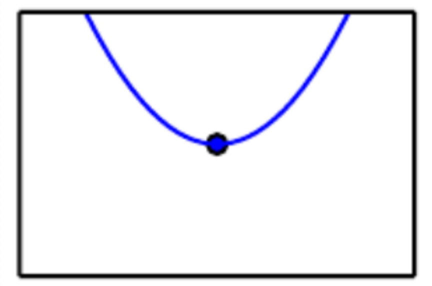


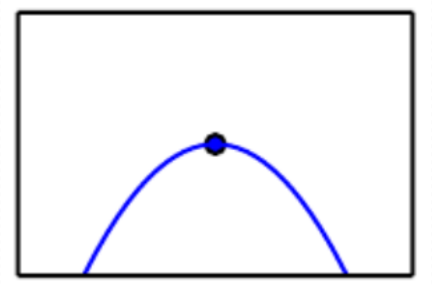
图3-4: 梯度下降。沿着函数的下坡方向（导数反方向）直到最小。

梯度下降法的工作原理

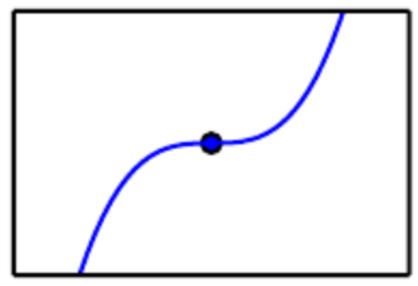
$$f(x - \eta * f'(x)) < f(x)$$



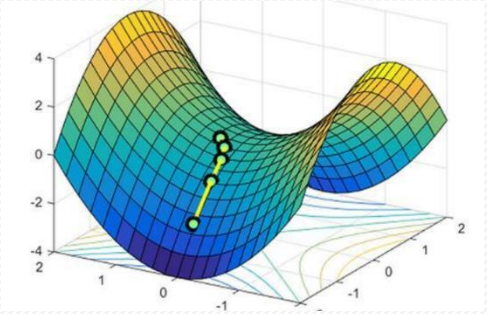
局部极小点



局部极大点



鞍点



鞍点

各种临界点

梯度下降法

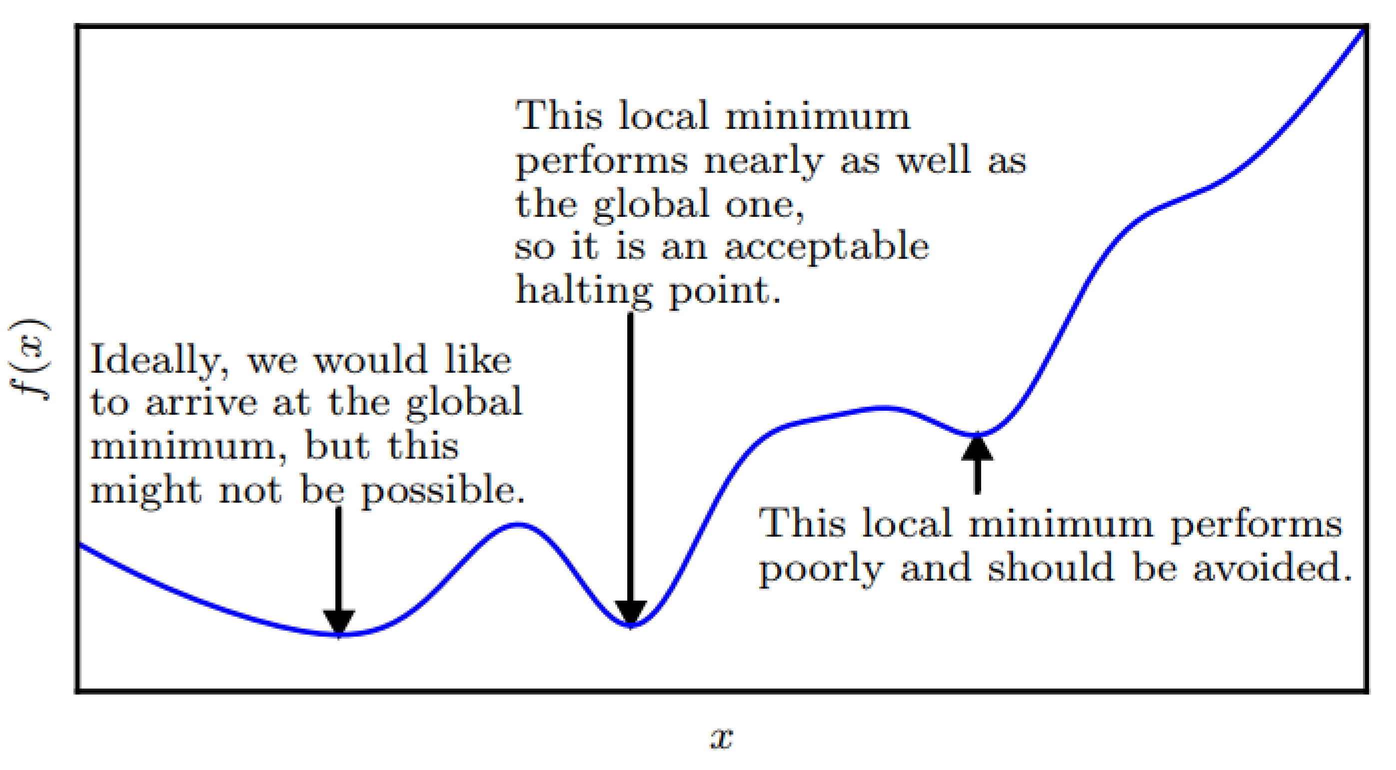


图3-6 近似最小化

当存在多个局部极小点或平坦区域时，优化算法可能无法找到全局最小点。在深度学习的背景下，即使找到的解不是真正最小的，但只要它们对应于代价函数的值显著低，通常就能接受这样的解。

梯度下降法

二维情形下的梯度下降法

$$\omega^{t+1} = \omega^{t} - \eta \frac{dL}{d\omega} \qquad (\eta>0)$$

三维情形下的梯度下降法

标量表示

$$\omega_1^{t+1} = \omega_1^{t} - \eta \frac{\partial L}{\partial \omega_1}$$

$$\omega_0^{t+1} = \omega_0^{t} - \eta \frac{\partial L}{\partial \omega_0}$$

向量表示

$$\boldsymbol{\omega} = (\omega_1, \omega_0)$$

$$\boldsymbol{\omega}^{t+1} = \boldsymbol{\omega}^{t} - \eta \frac{\partial L}{\partial \boldsymbol{\omega}}$$

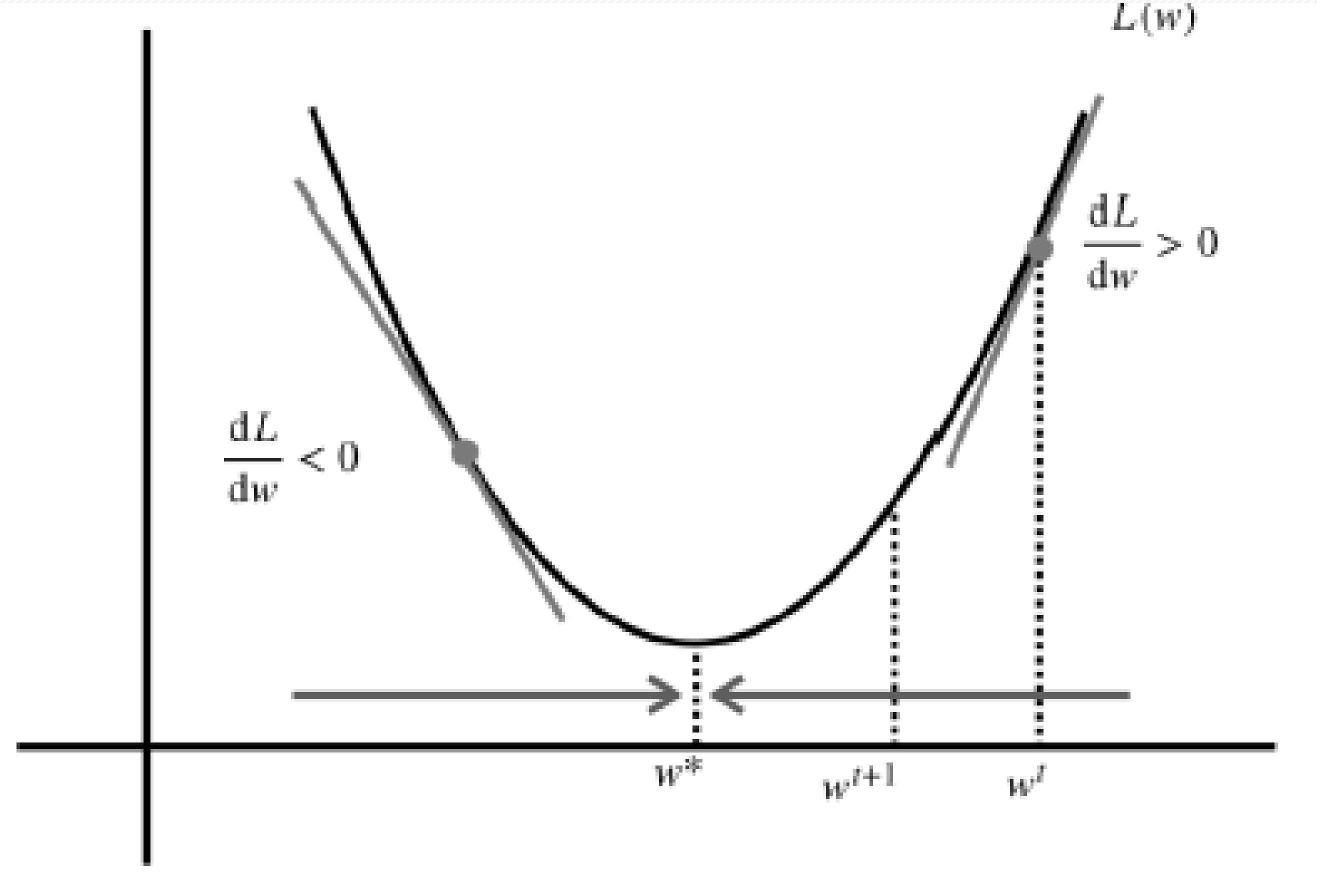


图3-5 二维情形下的梯度下降

3.2 线性回归(linear regression)

给定数据集$D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$，其中$\boldsymbol{x}_i=(x_{i1};x_{i2};\ldots;x_{id})$，$y_i\in\mathbb{R}$。

对于输入属性数目只有一个的最简单情形：$D=\{(x_i,y_i)\}_{i=1}^m$，其中$x_i\in\mathbb{R}$.

回归模型：$f(x_i)=\omega x_i+b$，使得$f(x_i)\approx y_i$

令均方误差最小化，有

$$(\omega^*,b^*)=\underset{(\omega,b)}{\arg\min}\sum_{i=1}^{m}(f(x_i)-y_i)^2$$

$$=\underset{(\omega,b)}{\arg\min}\sum_{i=1}^{m}(\omega x_i+b-y_i)^2$$

对 $E_{(\omega,b)}=\sum_{i=1}^m(\omega x_i+b-y_i)^2$ 进行最小二乘参数估计(LSM, least square method)。

3.2线性回归

$$E_{(\omega,b)}=\sum_{i=1}^{m}(\omega x_i+b-y_i)^2$$

分别对ω和b求导:

$$\frac{\partial E_{(\omega,b)}}{\partial\omega}=2\left(\omega\sum_{i=1}^{m}x_i^2-\sum_{i=1}^{m}(y_i-b)x_i\right)$$

$$\frac{\partial E_{(\omega,b)}}{\partial b}=2\left(mb-\sum_{i=1}^{m}(y_i-\omega x_i)\right)$$

令导数为 0, 得到闭式(closed-form)解:

$$\omega=\frac{\sum_{i=1}^{m}y_i(x_i-\bar{x})}{\sum_{i=1}^{m}x_i^2-\frac{1}{m}\left(\sum_{i=1}^{m}x_i\right)^2}$$

其中, $\bar{x}=\frac{1}{m}\sum_{i=1}^{m}x_i$

$$b=\frac{1}{m}\sum_{i=1}^{m}(y_i-\omega x_i)$$

多元(multi-variate)线性回归

$f(\boldsymbol{x}_i) = \boldsymbol{\omega}^T \boldsymbol{x}_i + b$使得$f(\boldsymbol{x}_i) \approx y_i$　　其中，$\boldsymbol{x}_i = (x_{i1}; x_{i2}; \ldots; x_{id}), y_i \in \mathbb{R}$

把$\boldsymbol{\omega}$和b吸收入向量形式$\hat{\boldsymbol{\omega}} = (\boldsymbol{\omega}; b)$，数据集表示为

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} & 1 \\ x_{21} & x_{22} & \ldots & x_{2d} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \ldots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1^T & 1 \\ \boldsymbol{x}_2^T & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_m^T & 1 \end{pmatrix}$$

$$\mathbf{y} = (y_1; y_2; \ldots, y_m)$$

同样，采用最小二乘法求解，有

$$\hat{\boldsymbol{\omega}}^* = \arg\min_{\hat{\boldsymbol{\omega}}} (\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\omega}})^T (\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\omega}})$$

或者

$$\hat{\boldsymbol{\omega}}^* = \arg\min_{\hat{\boldsymbol{\omega}}} |\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\omega}}|^2$$

多元(multi-variate)线性回归

令 $\boxed{E_{\hat{\boldsymbol{\omega}}} = (\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\omega}})^T(\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\omega}})}$ ，对$\hat{\boldsymbol{\omega}}$求导：

$$\boxed{\frac{\partial E_{\hat{\boldsymbol{\omega}}}}{\partial \hat{\boldsymbol{\omega}}} = 2\boldsymbol{X}^T(\mathbf{X}\hat{\boldsymbol{\omega}} - \boldsymbol{y})}$$ 令其为零，可得$\hat{\boldsymbol{\omega}}$

然而，麻烦来了：涉及矩阵求逆！

- 若$\mathbf{X}^T\boldsymbol{X}$满秩或正定，$\boxed{\hat{\boldsymbol{\omega}}^* = (\mathbf{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$ $\boxed{f(\hat{\boldsymbol{x}}_i) = \hat{\boldsymbol{x}}_i^T(\mathbf{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$
- 则若$\mathbf{X}^T\boldsymbol{X}$不满秩，则可解出多个$\hat{\boldsymbol{\omega}}$

此时需求助于归纳偏好，或引入正则化(regularization)项。

3.2 线性回归

```
# 生成矩阵X
def Construct_X(x):
    x0 = torch.ones(x.numpy().size)  #产生值为1的(列)向量
    X = torch.stack((x, x0), dim=1)  #将两个向量横向拼合（新增维度 ）
    return X

x = torch.Tensor([85, 100, 120, 125, 150])
y = torch.Tensor([250.93, 293.97, 366.56, 400.10, 471.72])
X = Construct_X(x)

# 定义权重向量w（模型参数，随机初始化，要求能够自动微分）
w = torch.rand(2, requires_grad=True )

inputs = X
target = y
```

3.2 线性回归

```
def train(epochs=1, learning_rate=0.01):
    for epoch in range(epochs):
        # 前向传播
        output = inputs.mv(w)  # 计算输出：y'=Xw
        loss = (target - output).pow(2).sum()  # 计算损失：L=∑(y-y')^2
        print(w.data)

        loss.backward()  # 反向传播（获得梯度）
        w.data -= learning_rate * w.grad  # 更新权重
        w.grad.zero_()  # 清空grad的值

        if epoch % 1000 == 0:
            draw(output, loss)

    return w, loss
w, loss = train(10000, learning_rate=1e-5)
```

3.2 线性回归

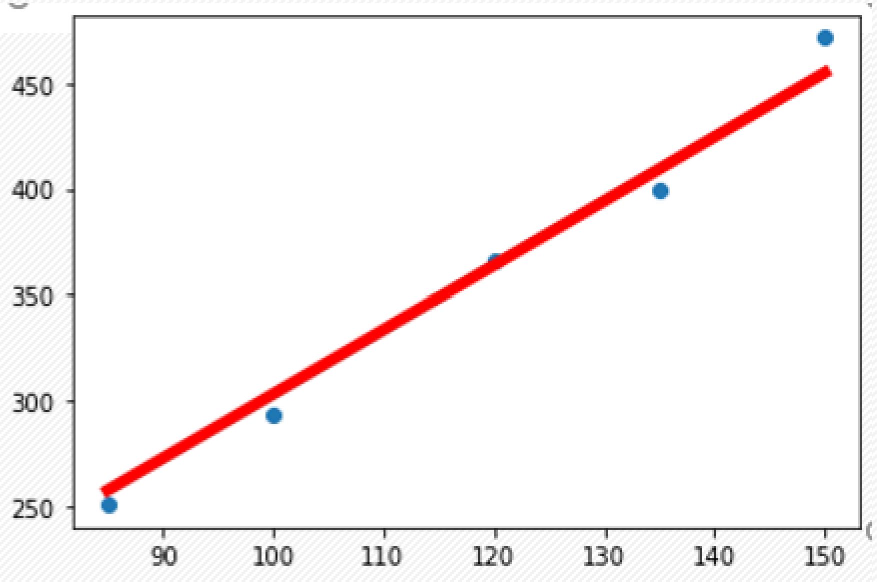
```
#绘图
def draw(output,loss):
    plt.cla()
    plt.scatter(x.numpy(), y.numpy())

    plt.plot(x.numpy(), output.data.numpy(),'r-', lw=5)
    plt.text(0.5, 0,'Loss=%s' % (loss.item()),fontdict={'size':20,'color':'red'})

    plt.pause(0.005)
```

```
w,loss = train(10000,learning_rate = 1e-4)
```

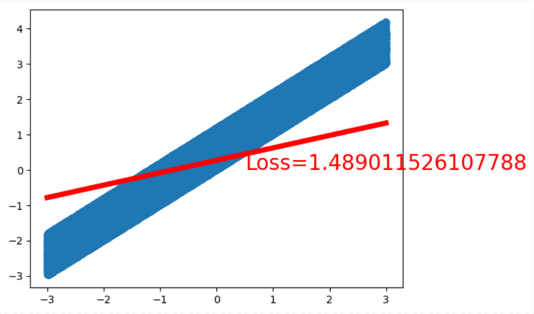
```
print("final loss:",loss.item())
print("weights:",w.data)
```

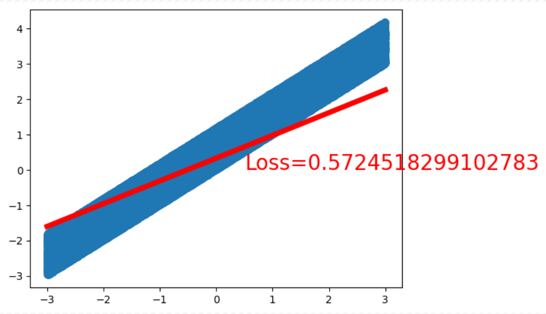


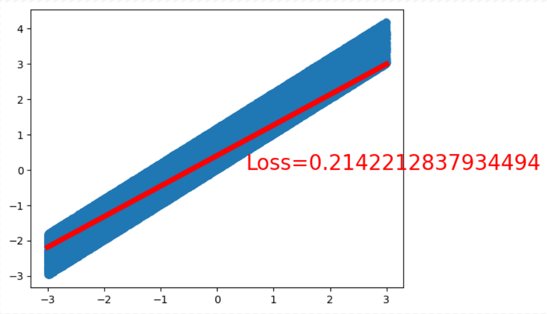
大规模数据实例

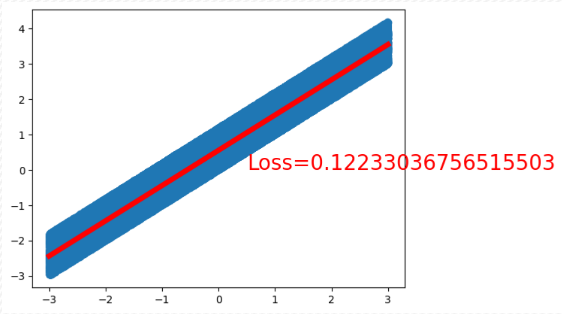
```
#构建特征矩阵X
def Construct_X(x):
        x0 = torch.ones_like(x)
        X = torch.stack((x, x0), dim=1)
        return X

x = torch.linspace(-3, 3, 100000)  #产生含有100000个元素的向量
X = Construct_X(x)
y = x +1.2*torch.rand(x.size())  #假设真实函数是y=x，增加一些误差
w = torch.rand(2)   #随机初始化权重向量w
```









线性回归的scikit-learn实现

■ 普通线性回归

class sklearn.linear_model.LinearRegression (fit_intercept=True, normalize=False, n_jobs=1)

■ Lasso回归

class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic’)

■ Ridge回归

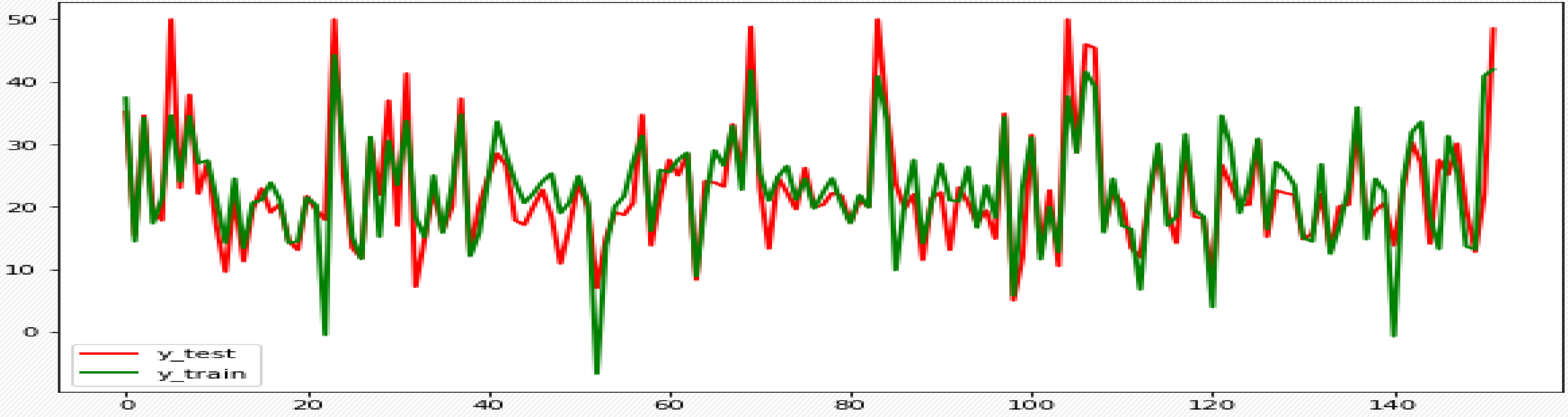
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)

■ ElasticNet回归

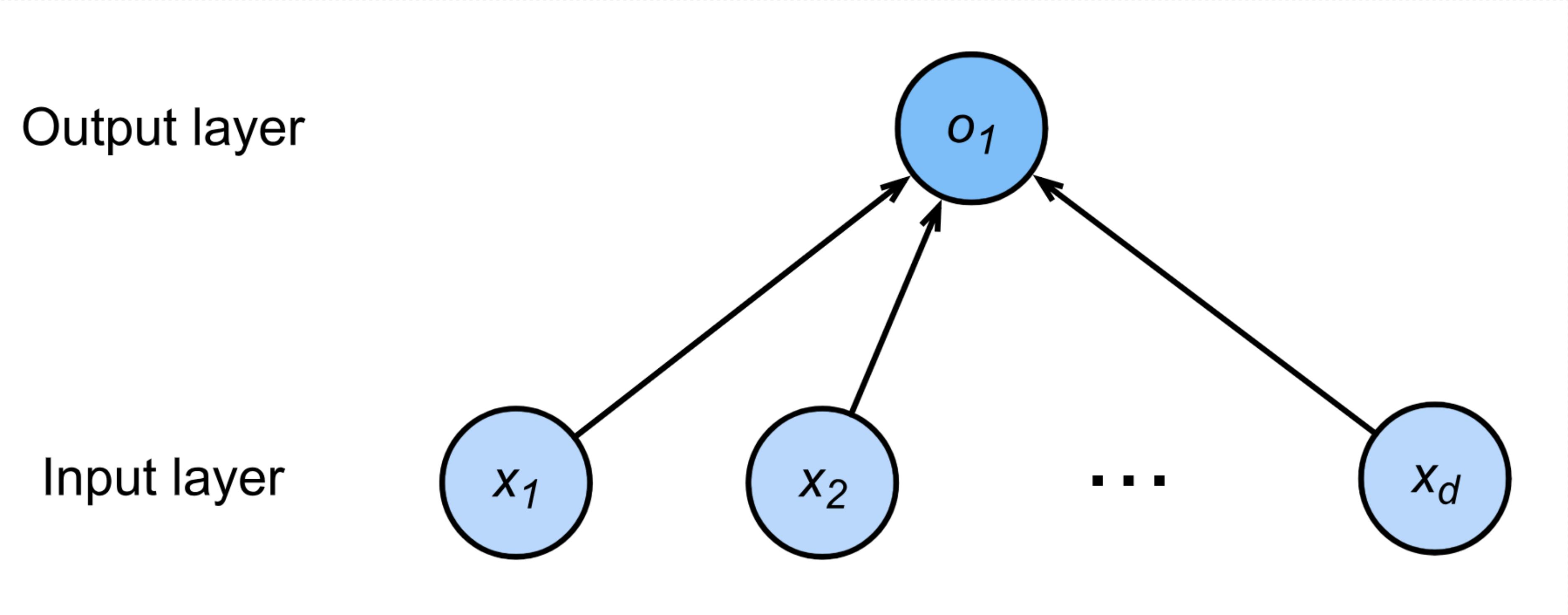
class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, normalize=False, precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')

例3.2，波士顿房价线性回归分析

CRIM：城镇人均犯罪率；
INDUS：城镇非商业土地的比例；
NOX：一氧化氮浓度；
AGE：1940 年之前建成的自用房屋比例；
RAD：距离高速公路的便利指数；
PTRATIO：城镇中师生比例；
LSTAT：人口中地位低下者的比例。
ZN：住宅用地比例；
CHAS：查理斯河变量（如果边界是河流，则为1；否则为0）；
RM：平均房间数；
DIS：到波士顿五个中心区域的加权距离；
TAX：不动产税率；
B：城镇中黑人的比例。
MEDV：自住房的平均房价，以千美元计。



线性方法是一个单层神经网络



可以堆叠多个层来获得深层神经网络。

线性模型的变化

对于样例$(\boldsymbol{x}, y)$, $y \in \mathbb{R}$，若希望线性模型的预测值逼近真实标记y，

则得到线性回归模型：$\boxed{y = \boldsymbol{\omega}^T \boldsymbol{x} + b}$

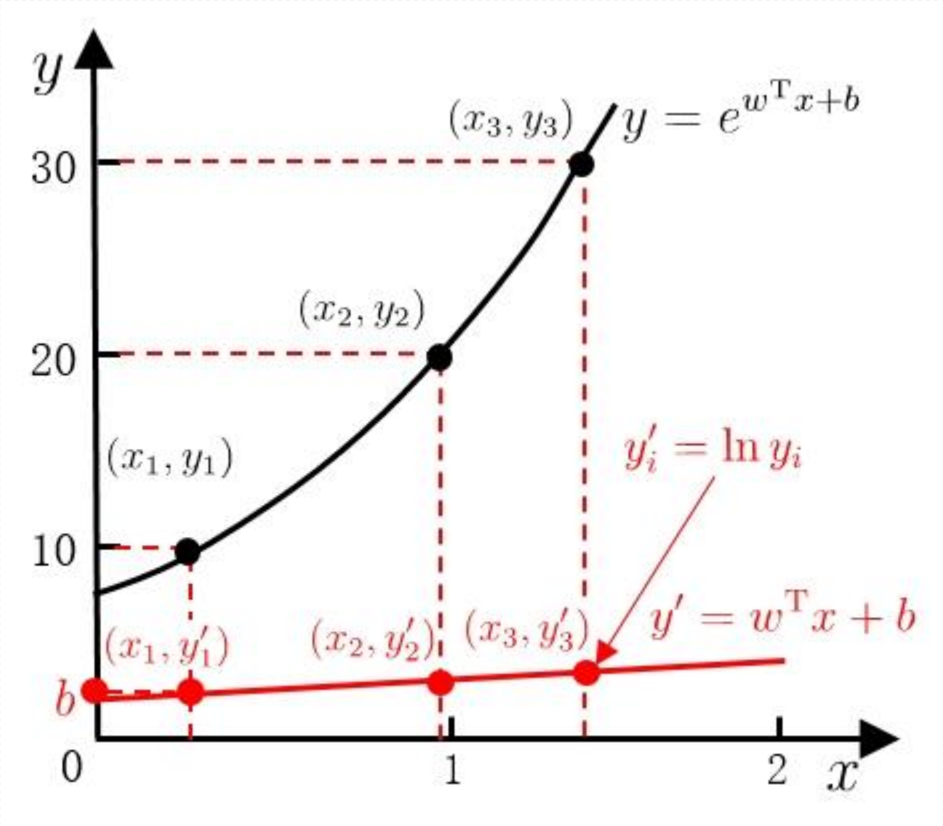
能否令预测值逼近y的衍生物？

$$\ln y = \boldsymbol{\omega}^T \boldsymbol{x} + b$$

则得到对数线性回归(log-linear regression)

实际是在用$e^{\boldsymbol{\omega}^T \boldsymbol{x}+b}$逼近$y$

$$y = e^{\boldsymbol{\omega}^T \boldsymbol{x}+b}$$



广义(generalized)线性模型

更一般地，考虑单调可微函数 $g(.)$，令:

$$y = g^{-1}(\boldsymbol{\omega}^T \boldsymbol{x} + b)$$

得到广义线性模型，其中函数$g(.)$称为**联系函数**(link function)

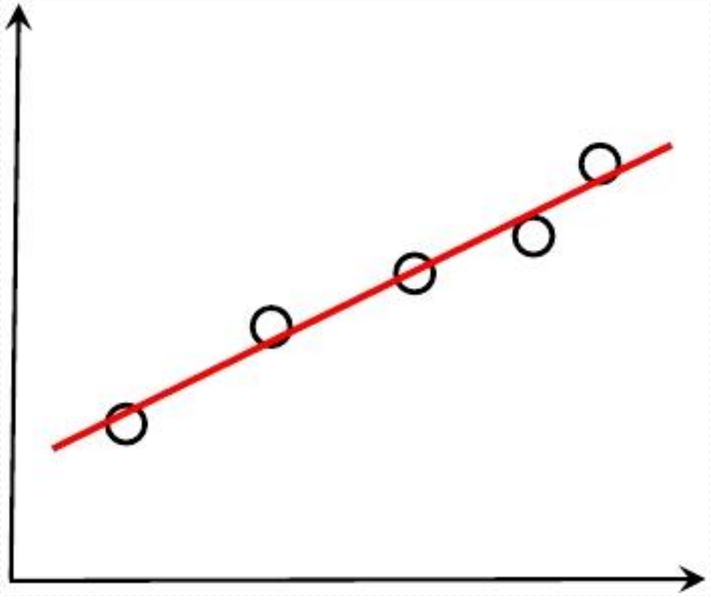
令$g(.) = \ln(.)$，则得到**对数**线性回归

$$\ln \mathrm{y} = \boldsymbol{\omega}^T \boldsymbol{x} + b$$

…… ……

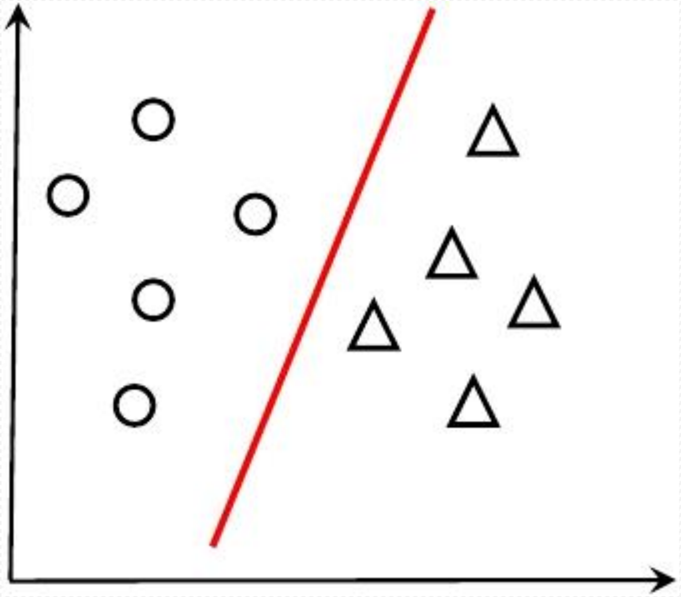
线性模型做“分类”

回归



广义线性模型：
通过“联系函数”
例如，对率回归

分类



如何“直接”做分类?

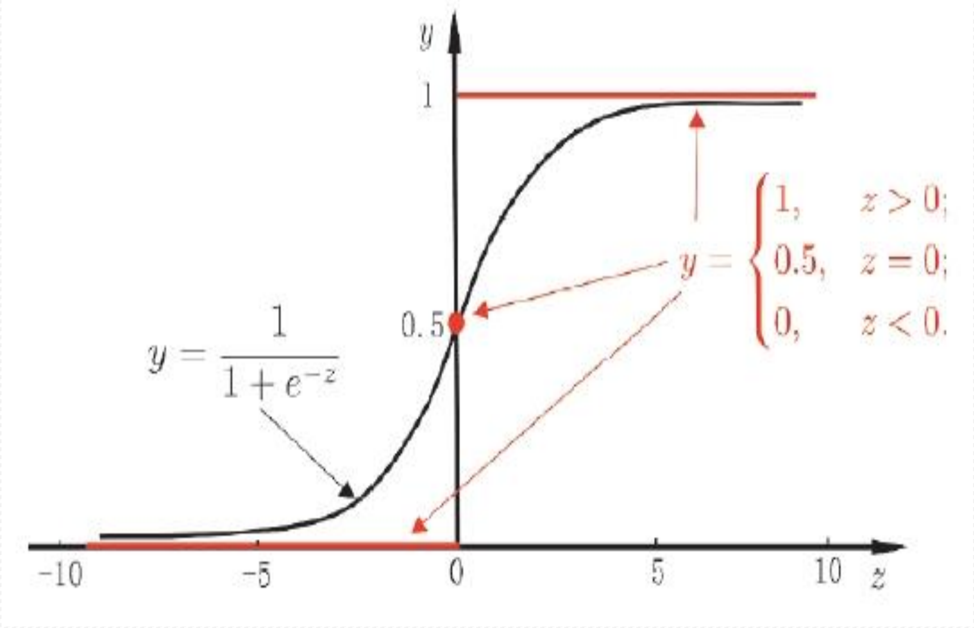
3.3 对数几率回归

方法：找到一个单调可微函数将分类任务的真实标记y与线性回归模型的预测值联系起来。

线性回归模型产生的实值输出$z = \boldsymbol{\omega}^T \boldsymbol{x} + b$

理想的“单位阶跃函数”(unit-step function)

$$y = \begin{cases} 1, & z > 0 \\ 0.5, & z = 0 \\ 0, & z < 0 \end{cases}$$



性质不好，需找“替代函数”

单调可微

$$y = \frac{1}{1 + e^{-z}}$$

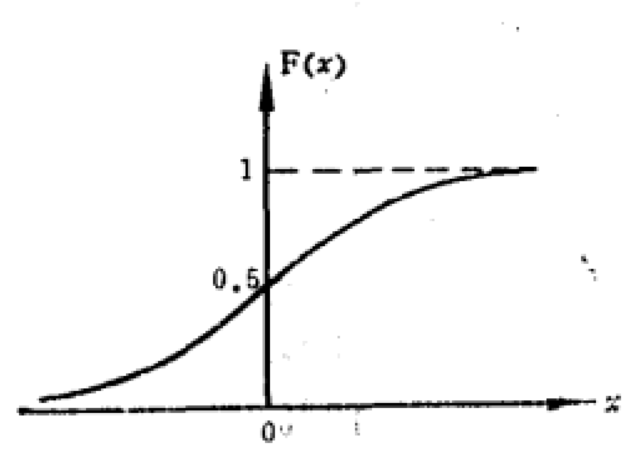
对数几率函数(logistic function)
简称“对率函数”

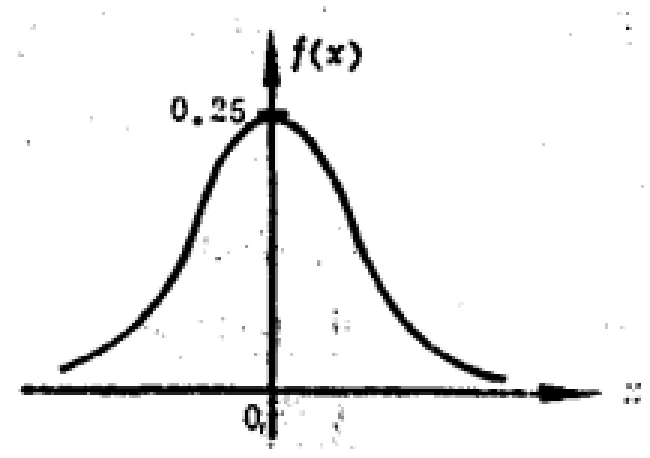
Logistic distribution

定义3.1 （Logistic分布） 设X是连续随机变量，X服从Logistic分布是指X具有下列分布函数和密度函数。

$$F(X) = P(X \le x) = \frac{1}{1 + e^{-(x-\mu)/\gamma}}$$

$$f(x) = F'(x) = \frac{e^{-(x-\mu)/\gamma}}{\gamma[1 + e^{-(x-\mu)/\gamma}]^2}$$





3.3 对率回归（二项Logistic回归）

定义 （对率回归模型）对率回归模型是如下的条件概率分布

$$P(Y=1|x)=\frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}=\frac{e^{(\boldsymbol{w}^T\boldsymbol{x}+b)}}{1+e^{(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

$$P(Y=0|x)=1-P(Y=1|x)=\frac{1}{1+e^{(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

得 $$\frac{P(Y=1|x)}{P(Y=0|x)}=e^{(\boldsymbol{w}^T\boldsymbol{x}+b)}$$ **几率**(odds)，反映了$\boldsymbol{x}$作为正例的相对可能性

因此 $$\log\frac{P(Y=1|x)}{P(Y=0|x)}=\boldsymbol{w}^T\boldsymbol{x}+b$$ “对数几率” (log odds, 亦称 logit)

模型参数估计

若将 y_i 看作类后验概率估计$P(y_i=1|\boldsymbol{x}_i)$, 则

$$p(y_i=1|\boldsymbol{x}_i)=\frac{1}{1+e^{-(\omega^T x_i+b)}}=\frac{e^{\omega^T x_i+b}}{1+e^{\omega^T x_i+b}}$$

$$p(y_i=0|\boldsymbol{x}_i)=1-p(y_i=1|\boldsymbol{x}_i)=\frac{1}{1+e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b}}$$

给定数据集$\{(\boldsymbol{x}_i,y_i)\}_{i=1}^m$，定义似然函数

$$L(\boldsymbol{\omega},b)=\prod_{i=1}^m[p(y_i=1|\boldsymbol{x}_i)]^{y_i}[p(y_i=0|\boldsymbol{x}_i)]^{1-y_i}$$

$$\ln L(\boldsymbol{\omega},b)=\sum_{i=1}^m\{y_i\ln[p(y_i=1|\boldsymbol{x}_i)]+(1-y_i)\ln[p(y_i=0|\boldsymbol{x}_i)]\}$$

$$\ln L(\boldsymbol{\omega},b)=\sum_{i=1}^m\{y_i\ln\frac{e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b}}{1+e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b}}+(1-y_i)\ln\frac{1}{1+e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b}}\}$$

$$\ln L(\boldsymbol{\omega},b)=\sum_{i=1}^m\{y_i(\boldsymbol{\omega}^T\boldsymbol{x}_i+b)-y_i\ln\left(1+e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b}\right)+(y_i-1)\ln(1+e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b})\}$$

$$\ln L(\boldsymbol{\omega},b)=\sum_{i=1}^m\{y_i(\boldsymbol{\omega}^T\boldsymbol{x}_i+b)-\ln(1+e^{\boldsymbol{\omega}^T\boldsymbol{x}_i+b})\}$$

模型参数估计

$$\frac{\partial}{\partial \boldsymbol{\omega}} \ln L(\boldsymbol{\omega}, b) = \frac{\partial}{\partial \boldsymbol{\omega}} \sum_{i=1}^{m} \{y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b) - \ln(1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b})\}$$

$$= \sum_{i=1}^{m} (y_i \boldsymbol{x}_i - \frac{\boldsymbol{x}_i e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}{1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}})$$

$$= \sum_{i=1}^{m} (y_i - \frac{e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}{1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}})\, \boldsymbol{x}_i$$

$$\frac{\partial}{\partial b} \ln L(\boldsymbol{\omega}, b) = \frac{\partial}{\partial b} \sum_{i=1}^{m} \{y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b) - \ln(1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b})\}$$

$$= \sum_{i=1}^{m} (y_i - \frac{e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}{1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}})$$

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta(\sum_{i=1}^{m} (y_i - \frac{e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}{1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}))\boldsymbol{x}_i$$

$$b \leftarrow b - \eta(\sum_{i=1}^{m} (y_i - \frac{e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}{1 + e^{\boldsymbol{\omega}^T \boldsymbol{x}_i + b}}))$$

例3.3，鸢尾花数据集

http://archive.ics.uci.edu/ml/datasets/Iris

- 埃德加·安德森在加拿大加斯佩半岛上提取了鸢尾属花朵的地理变异数据。数据集X包含了150个样本(M=150)，均属于鸢尾属下的三个亚属(3个类别)，分别是山鸢尾(setosa)、变色鸢尾(versicolor)和维吉尼亚鸢尾(virginica)。数据集通过四个特征维度(N=4)描述鸢尾属花朵样本：花萼长度(sepal length)、花萼宽度(sepal width)、花瓣长度(petal length)、花瓣宽度(petal width)

Iris setosa

Iris versicolor

Iris virginica

第一步：描述任务

■问题：

- 150个样本。
- 4个特征维度：花萼长度 sepal length(cm)、花萼宽度 sepal width(cm）、花瓣长度 petal length(cm)、花瓣宽度petal width (cm)。
- 3个类别：0-setosa，1-versicolor, 2-virginica。

■目标：

- 获得logistic分类器，在给定样本特征的未知标签数据上（如x = [3 2 5 3]) ，可以准确预测标签y。

第二步：观察数据集

```
from sklearn.datasets import load_iris
import pandas as pd
iris=load_iris()
iris.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names', 'filename', 'data_module'])
```

```
type(iris['data'])
```

```
numpy.ndarray
```

```
iris['data'].shape
```

```
(150, 4)
```

```
iris['feature_names']
```

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

```
iris['target_names']
```

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

```
iris['target'].shape
```

```
(150,)
```

```
iris_df=pd.DataFrame(iris["data"],columns=iris["feature_names"])
iris_df.head(2)
```

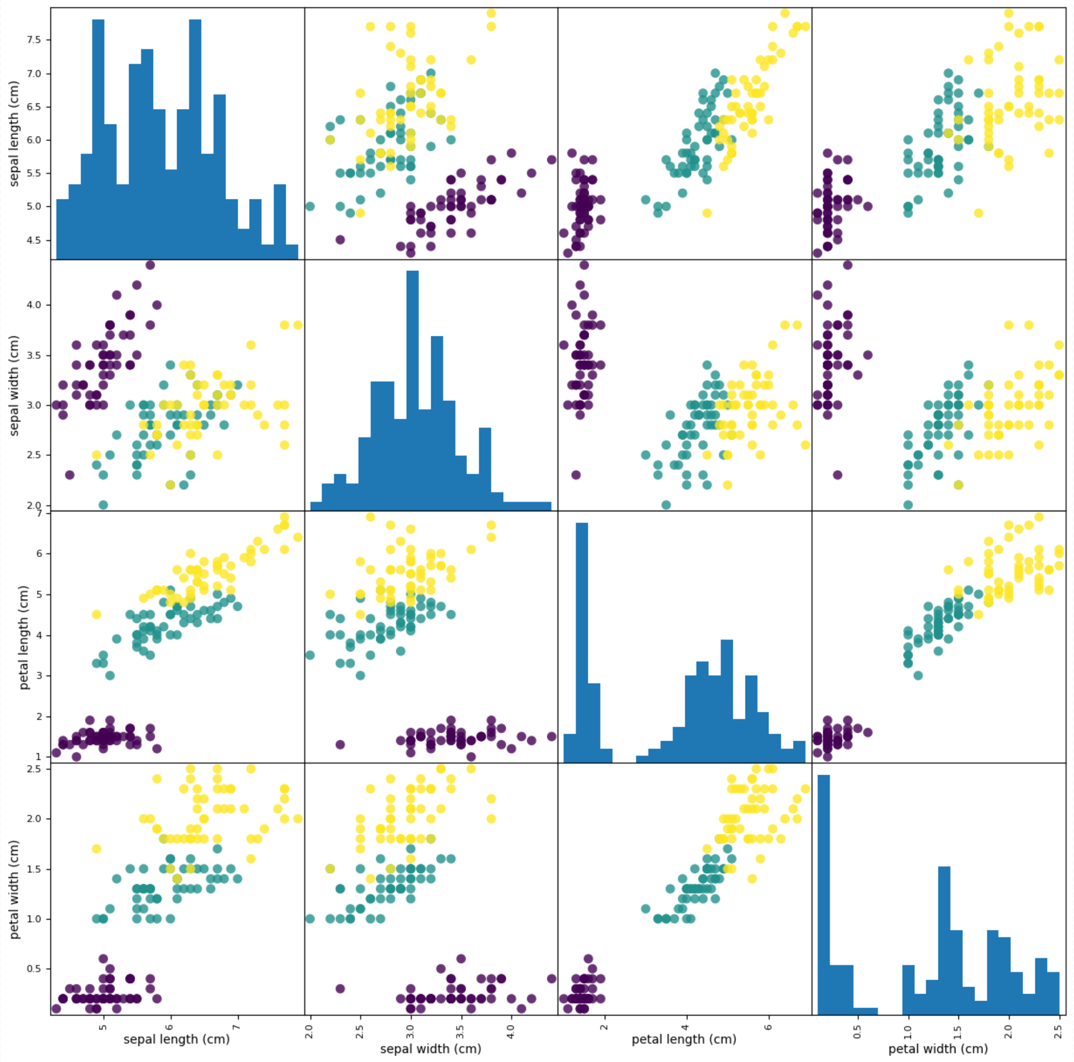
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

```
print(iris_df.isnull().sum())
```

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
```

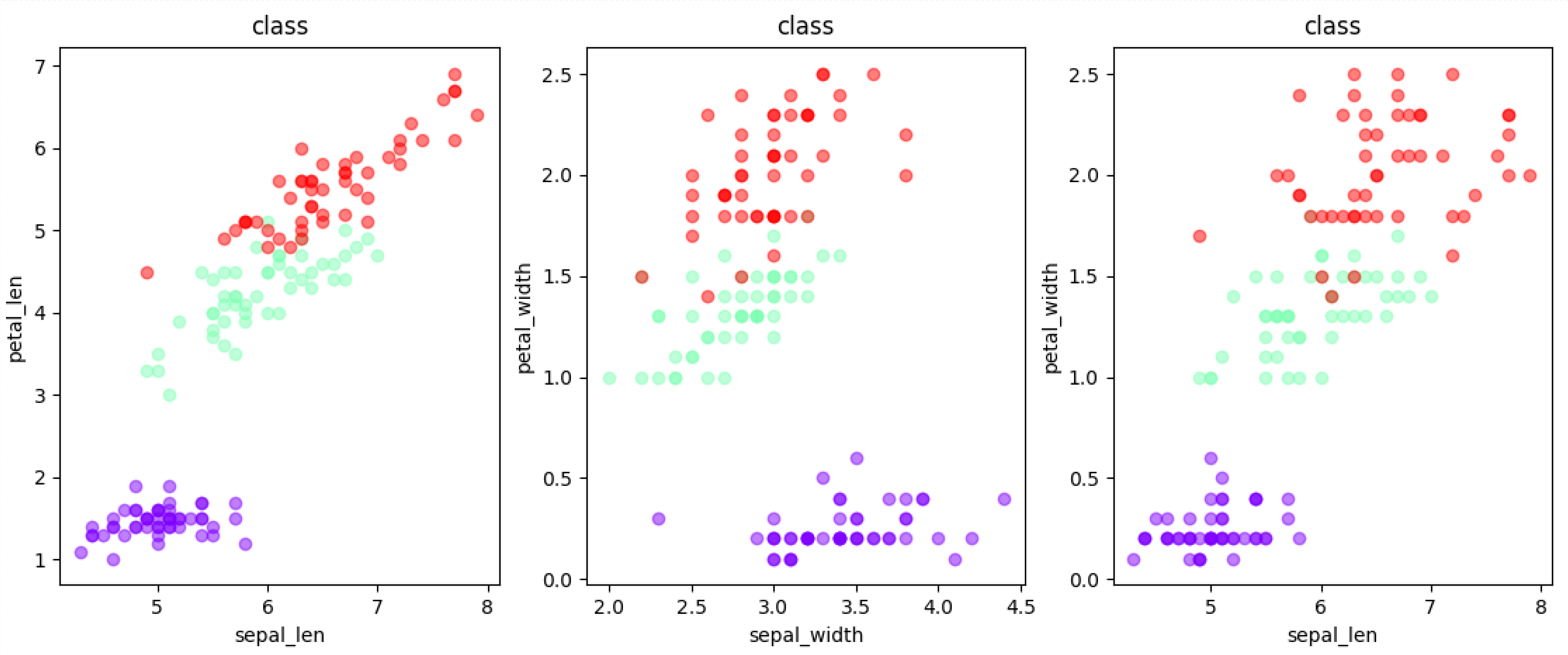
第二步：观察数据集

```
grr = pd.plotting.scatter_matrix(iris_df, c=iris["target"], figsize=(15, 15), marker='o', hist_kwds={'bins': 20}, s=60, alpha=.8)
```



可视化显示

iris_vis.txt



第三步：训练和评估模型

```
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()
iris_x = iris.data
iris_y = iris.target

x_train, x_test, y_train, y_test = train_test_split(iris_x, iris_y, test_size=0.3, random_state=0)
print(y_train)

classifier = LogisticRegression(solver='lbfgs',multi_class='auto')
classifier.fit(x_train, y_train)

y_predict = classifier.predict(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_predict))
```

例3.4，威斯康辛州乳腺癌数据（自实现）

- 包含了威斯康辛州记录的569个病人(其中357个良性，212个恶性)的乳腺癌恶性/良性（1/0）数据，以及与之对应的30个维度的生理指标数据。
- 字段中包含mean的代表平均值，包含se的代表标准差（standard error），包含worst代表最大值（3个最大值的平均值）。每张图像都计算了相应的特征，得出了30个特征值。（实际上是10个特征值的3个维度：平均、标准差、最大值）。这些特征值都保留了4位数字。字段中没有缺失的值。

例3.5，泰坦尼克号数据集

泰坦尼克号数据集

和平时期死伤人数最为惨重的一次海难。

泰坦尼克号数据集

三等舱乘客大部分时间都遭到船员留置于下层甲板，他们只能自己想办法穿越障碍，导致其中的许多乘客受困，而大部分下层甲板都充满了海水。在登艇时一般都遵循女人和小孩优先的原则，大部分男性乘客和船员都留在船上。

Age/sex	Class/crew	Number aboard	Number saved	Number lost	Percentage saved	Percentage lost
Children	First Class	6	5	1	83%	17%
	Second Class	24	24	0	100%	0%
	Third Class	79	27	52	34%	66%
Women	First Class	144	140	4	97%	3%
	Second Class	93	80	13	86%	14%
	Third Class	165	76	89	46%	54%
	Crew	23	20	3	87%	13%
Men	First Class	175	57	118	33%	67%
	Second Class	168	14	154	8%	92%
	Third Class	462	75	387	16%	84%
	Crew	885	192	693	22%	78%
Total		**2224**	**710**	**1514**	**32%**	**68%**

谁能活下来？

3.4 线性判别分析(Linear Discriminant Analysis, LDA)

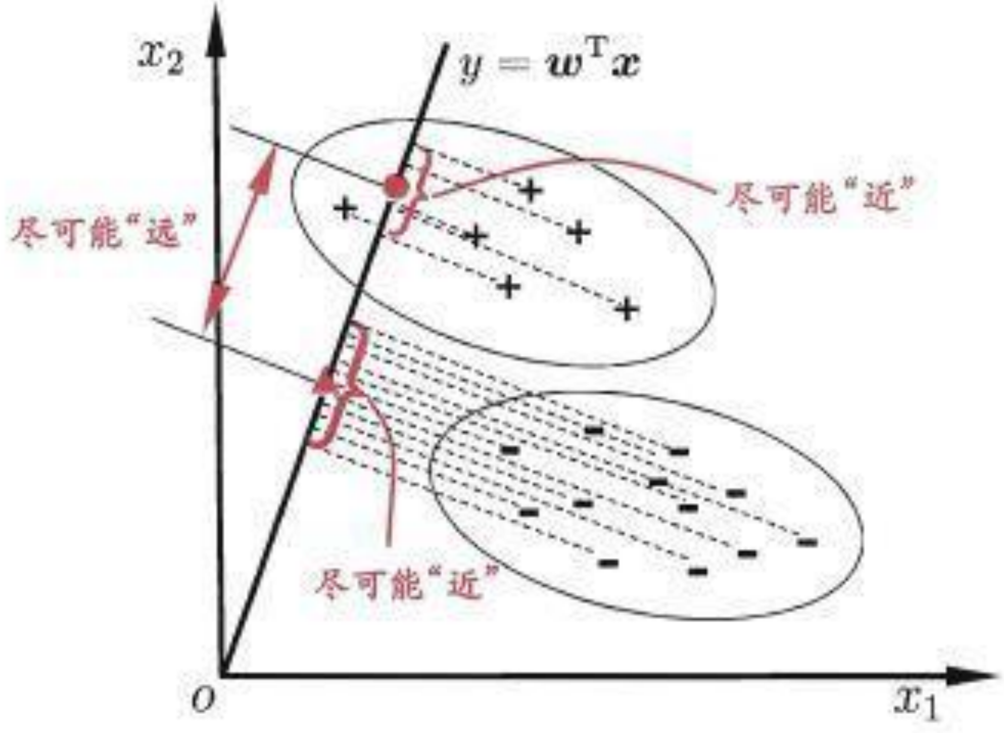


图 3.3 LDA 的二维示意图. "+"、"−"分别代表正例和反例, 椭圆表示数据簇的外轮廓, 虚线表示投影, 红色实心圆和实心三角形分别表示两类样本投影后的中心点.

将样例投影到一条直线（低维空间），因此也被视为一种"监督降维"技术。

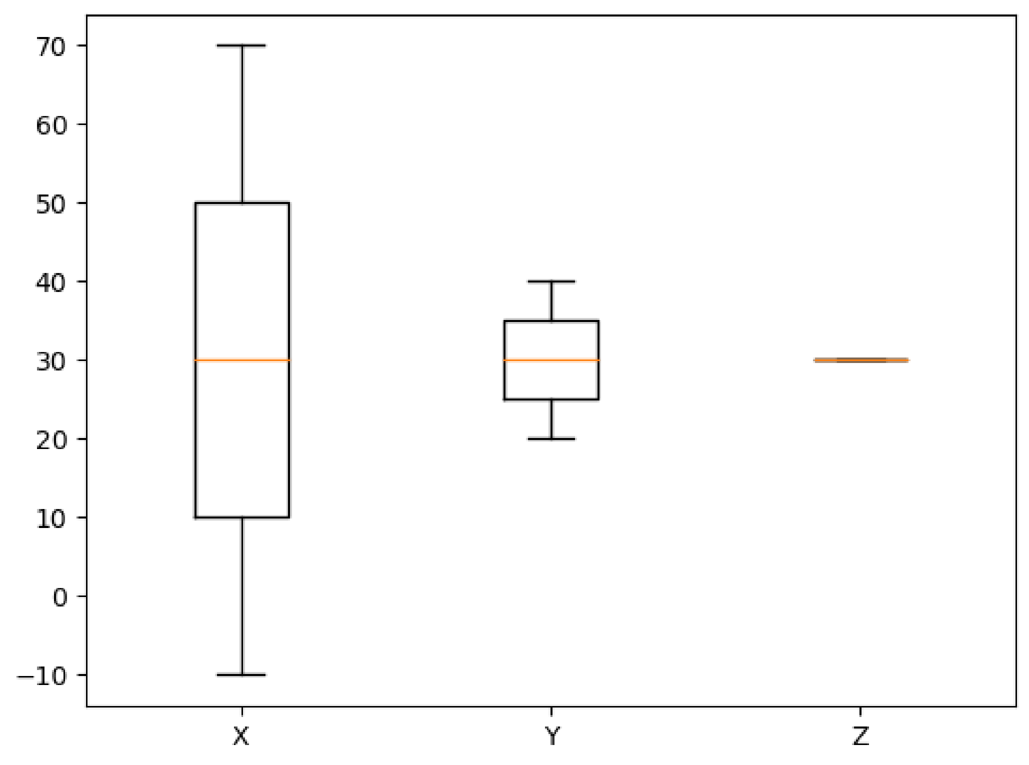
方差（variance）

3组数据

向量	内容
X	[-10, 10, 30, 50, 70]
Y	[20, 25, 30, 35, 40]
Z	[30, 30, 30, 30, 30]

3组数据的统计量

统计量	X	Y	Z
计数	5	5	5
均值	30	30	30
标准差	28.28	7.07	0
最小值	-10	20	30
上四分位点	10	25	30
中位数	30	30	30
下四分位点	50	35	30
最大值	70	40	30



3组数据的箱型图

```
from matplotlib import pyplot as plt
X=[-10, 10, 30, 50, 70]
Y=[20, 25, 30, 35, 40]
Z=[30, 30, 30, 30, 30]
plt.boxplot((X, Y, Z),labels=["X","Y","Z"])
plt.show()
```

例3.6，协方差矩阵及其计算

方差

$$\mathrm{Var}(\boldsymbol{x}) = \frac{\sum_{i=1}^{n}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^2}{n-1}$$

协方差

$$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{y}_i - \overline{\boldsymbol{y}})}{n-1}$$

假设有 4 个样本，每个样本有两个特征，分别表示如下：

$$\boldsymbol{x}_1 = (1,2), \boldsymbol{x}_2 = (3,6), \boldsymbol{x}_3 = (4,2), \boldsymbol{x}_4 = (5,2)$$

所有样本信息的矩阵表示：

$$\boldsymbol{Z} = \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 4 & 2 \\ 5 & 2 \end{bmatrix}$$

例3.6（续），协方差矩阵及其计算——手工计算

用两个变量空间$\boldsymbol{x}$，$\boldsymbol{y}$分别表示两个特征对应的向量

$$\boldsymbol{x}=\begin{bmatrix}1\\3\\4\\5\end{bmatrix} \quad \boldsymbol{y}=\begin{bmatrix}2\\6\\2\\2\end{bmatrix} \quad \text{Cov}(\boldsymbol{z})=\begin{bmatrix}Cov(\boldsymbol{x},\boldsymbol{x}) & Cov(\boldsymbol{x},\boldsymbol{y})\\Cov(\boldsymbol{y},\boldsymbol{x}) & Cov(\boldsymbol{y},\boldsymbol{y})\end{bmatrix}$$

首先，计算出$\boldsymbol{x}$，$\boldsymbol{y}$两个特征向量的平均值：$\overline{\boldsymbol{x}}=3.25,\overline{\boldsymbol{y}}=3$

然后，计算协方差矩阵的各个元素：

$$\text{Cov}(\boldsymbol{x},\boldsymbol{x})=\frac{(1-3.25)^2+(3-3.25)^2+(4-3.25)^2+(5-3.25)^2}{4-1}=2.9167$$

$$\text{Cov}(\boldsymbol{x},\boldsymbol{y})=\frac{(1-3.25)(2-3)+(3-3.25)(6-3)+(4-3.25)(2-3)+(5-3.25)(2-3)}{4-1}=-0.3333$$

$$\text{Cov}(\boldsymbol{y},\boldsymbol{x})=\frac{(2-3)(1-3.25)+(6-3)(3-3.25)+(2-3)(4-3.25)+(2-3)(5-3.25)}{4-1}=-0.3333$$

$$\text{Cov}(\boldsymbol{y},\boldsymbol{y})=\frac{(2-3)^2+(6-3)^2+(2-3))^2+(2-3))^2}{4-1}=4$$

$$\text{Cov}(\boldsymbol{Z})=\begin{bmatrix}2.9167 & -0.3333\\-0.3333 & 4\end{bmatrix}$$

例3.6（续），协方差计算——numpy实现

```
import numpy as np

a=np.array([1,3,4,5])
b=np.array([2,6,2,2])
z=np.stack((a,b))
np.cov(z)
```

```
array([[ 2.91666667, -0.33333333],
       [-0.33333333,  4.        ]])
```

$$\mathrm{Cov}(z)=\begin{bmatrix}2.9167 & -0.3333\\ -0.3333 & 4.0000\end{bmatrix}$$

LDA的目标

给定数据集$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}, y_i \in\{0,1\}$

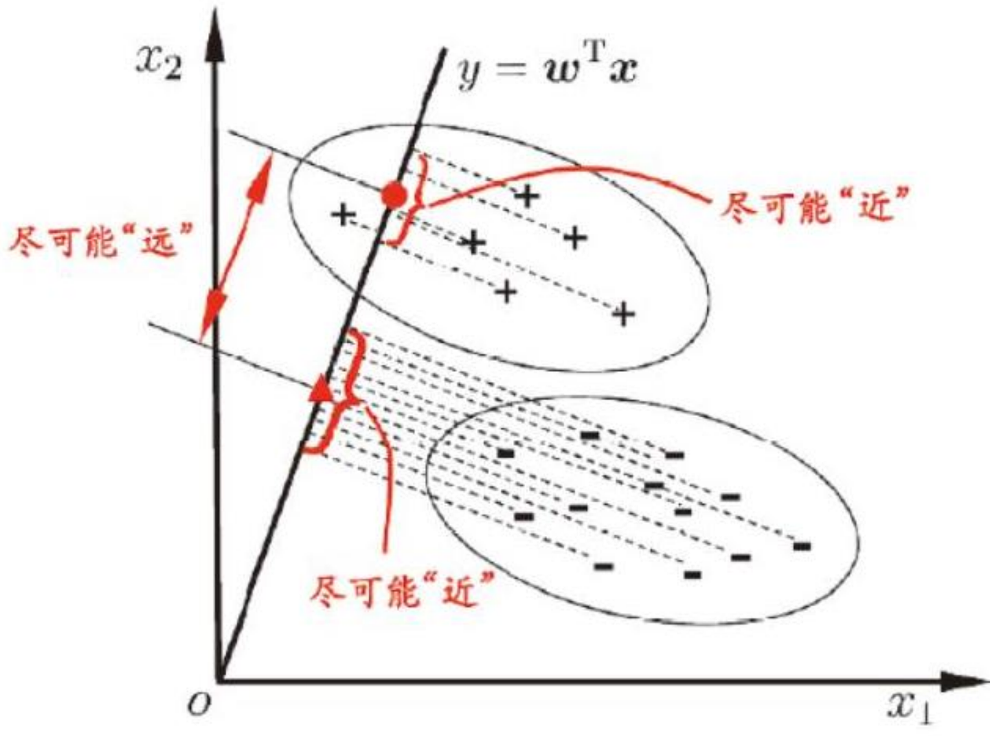
第 i 类示例的集合$\boldsymbol{X}_i$

第 i类示例的均值向量$\boldsymbol{\mu}_i$

第 i类示例的协方差矩阵$\boldsymbol{\Sigma}_i$

两类样本的中心在直线上的投影：$\boldsymbol{\omega}^T\boldsymbol{\mu}_0$和$\boldsymbol{\omega}^T\boldsymbol{\mu}_1$

两类样本投影后的协方差： $\boldsymbol{\omega}^T\boldsymbol{\Sigma}_0\boldsymbol{\omega}$和$\boldsymbol{\omega}^T\boldsymbol{\Sigma}_1\boldsymbol{\omega}$



同类样例的投影点尽可能接近， $\boldsymbol{\omega}^T\boldsymbol{\Sigma}_0\boldsymbol{\omega}+\boldsymbol{\omega}^T\boldsymbol{\Sigma}_1\boldsymbol{\omega}$尽可能小

异类样例的投影点尽可能远离， $\left\|\boldsymbol{\omega}^T\boldsymbol{\mu}_0-\boldsymbol{\omega}^T\boldsymbol{\mu}_1\right\|_2^2$尽可能大

于是，最大化

$$J=\frac{\left\|\boldsymbol{\omega}^T\boldsymbol{\mu}_0-\boldsymbol{\omega}^T\boldsymbol{\mu}_1\right\|_2^2}{\boldsymbol{\omega}^T\boldsymbol{\Sigma}_0\boldsymbol{\omega}+\boldsymbol{\omega}^T\boldsymbol{\Sigma}_1\boldsymbol{\omega}}=\frac{\boldsymbol{\omega}^T(\mu_0-\mu_1)(\mu_0-\mu_1)^T\boldsymbol{\omega}}{\boldsymbol{\omega}^T(\boldsymbol{\Sigma}_0+\boldsymbol{\Sigma}_1)\boldsymbol{\omega}}$$

LDA的目标

类内散度矩阵 (within-class scatter matrix)

$$S_\omega = \Sigma_0 + \Sigma_1$$

$$= \sum_{x \in X_0} (x - \mu_0)(x - \mu_0)^T + \sum_{x \in X_1} (x - \mu_1)(x - \mu_1)^T$$

类间散度矩阵 (between-class scatter matrix)

$$S_b = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T$$

LDA的目标：最大化广义瑞利商 (generalized Rayleigh quotient)

$$J = \frac{\omega^T S_b \omega}{\omega^T S_\omega \omega}$$

求解思路

令 $\omega^T S_\omega \omega=1$ ，最大化广义瑞利商等价形式为

$$J = \frac{\omega^T S_b \omega}{\omega^T S_\omega \omega} \equiv \begin{array}{l} \min\limits_{\omega} -\omega^T S_b \omega \\ \text{s.t. } \omega^T S_\omega \omega = 1 \end{array}$$

运用拉格朗日乘子法，有 $S_b\omega = \lambda S_\omega \omega$

$S_b\omega$的方向恒为$\mu_0 - \mu_1$，不妨令 $S_b\omega = \lambda(\mu_0 - \mu_1)$

$$\omega = S_\omega^{-1}(\mu_0 - \mu_1)$$

实践中通常是进行奇异值分解 $S_\omega = U\Sigma V^T$

然后 $S_\omega^{-1} = V\Sigma^{-1}U^T$

推广到多类

假定有 N 个类

全局散度矩阵 $\boldsymbol{S}_t = \boldsymbol{S}_b + \boldsymbol{S}_\omega = \sum_{i=1}^{m}(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$

类内散度矩阵 $\boldsymbol{S}_\omega = \sum_{i=1}^{m}\boldsymbol{S}_{\omega_i}$ $\boldsymbol{S}_{\omega_i} = \sum_{\boldsymbol{x}\in X_i}(\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^T$

类间散度矩阵 $\boldsymbol{S}_b = \boldsymbol{S}_t - \boldsymbol{S}_\omega = \sum_{i=1}^{N} m_i(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$

多分类**LDA**有多种实现方法：采用$\boldsymbol{S}_b, \boldsymbol{S}_w, \boldsymbol{S}_t$中的任何两个

例如，$\max_{\boldsymbol{W}} \frac{tr(\boldsymbol{W}^T\boldsymbol{S}_b\boldsymbol{W})}{tr(\boldsymbol{W}^T\boldsymbol{S}_\omega\boldsymbol{W})}$ ⇒ $\boldsymbol{S}_b\boldsymbol{W} = \lambda\boldsymbol{S}_\omega\boldsymbol{W}$

$\boldsymbol{W} \in \mathbb{R}^{d\times(N-1)}$

$\boldsymbol{W}$的闭式解是$\boldsymbol{S}_\omega^{-1}\boldsymbol{S}_b$的$d'(d' \le N-1)$个最大广义特征值所对应的特征向量组成的矩阵

LDA 算法流程

输入：数据集$D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\dots,(\boldsymbol{x}_N,y_N)\}$，其中任意样本$\boldsymbol{x}_i$为$d$维向量，$y_i\in\{C_1,C_2,\dots,C_k\}$，降维到的维度$d'$。

输出：降维后的样本集d'

1）计算类内散度矩阵$\boldsymbol{S}_{\boldsymbol{\omega}}$

2）计算类间散度矩阵 $\boldsymbol{S}_b$

3）计算矩阵$\boldsymbol{S}_{\boldsymbol{\omega}}^{-1}\boldsymbol{S}_b$

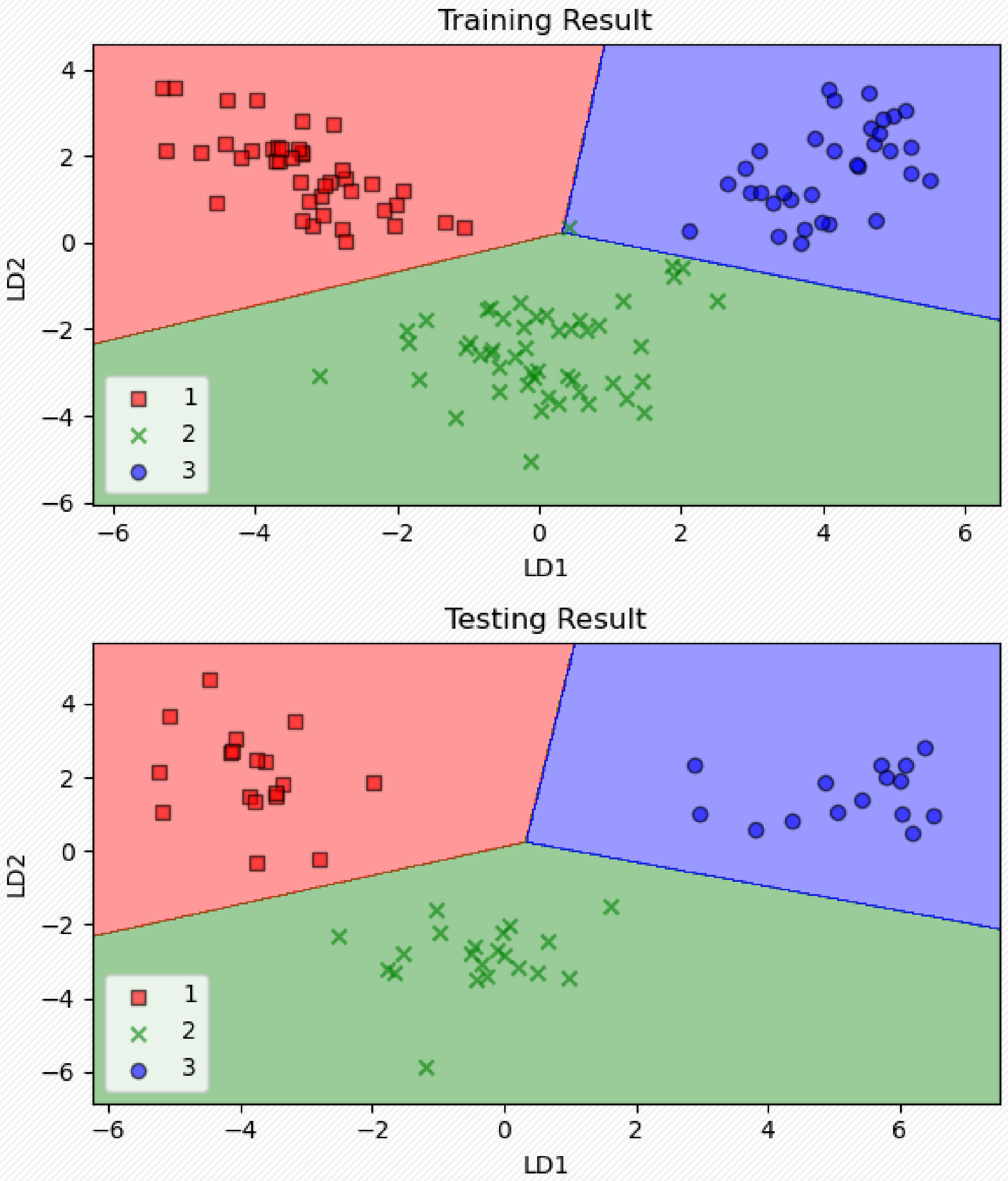
4）计算$\boldsymbol{S}_{\boldsymbol{\omega}}^{-1}\boldsymbol{S}_{\boldsymbol{b}}$ 的最大的d'个特征值和对应的d'个特征向量（$\boldsymbol{\omega}_1,\boldsymbol{\omega}_2,\dots\boldsymbol{\omega}_{d'}$），得到投影矩阵 $\boldsymbol{W}\in\mathbb{R}^{n\times d'}$

5）对样本集中的每一个样本特征$\boldsymbol{x}_i$，转化为新样本$\boldsymbol{z}_i=\boldsymbol{W}^T\boldsymbol{x}_i$

6）得到输出样本集 $\boldsymbol{D}'=\{(\boldsymbol{z}_1,y_1),(\boldsymbol{z}_2,y_2),\dots,(\boldsymbol{z}_m,y_m)\}$

例3.7，基于sklearn的线性判别分析(LDA)代码实现

- **Wine Data Set**由**Stefan Aeberhard**捐助，是对意大利同一地区种植的葡萄酒进行化学分析的结果，这些葡萄酒来自三个不同的品种。该分析确定了三种葡萄酒中每种葡萄酒中含有的**13**种成分的数量。不同种类的酒品的成分有所不同，通过对成分分析可以对不同的葡萄酒进行分类。
- 原始数据集共有**178**个样本数、**3**种数据类别、每个样本的有**13**个属性，分别是：酒精、苹果酸、灰、灰分的碱度、镁、总酚、黄酮类化合物、非黄烷类酚类、原花色素、颜色强度、色调、稀释葡萄酒的**OD280 / OD315**、脯氨酸。



LDA算法小结

■主要优点

1）降维过程中使用类别的先验知识，而PCA则无法使用类别先验知识。

2）LDA在样本分类信息依赖均值而不是方差的时候，比PCA之类的算法较优。

■主要缺点

1）LDA不适合对非高斯分布样本进行降维，PCA也有这个问题。

2）LDA降维最多降到类别数$k-1$的维数，如果降维的维度大于$k-1$，则不能使用。当然目前有一些LDA的进化算法可以绕过这个问题。

3）LDA在样本分类信息依赖方差而不是均值的时候，降维效果不好。

4）LDA可能过拟合数据。

3.5 多分类学习

■**拆解法**：将多分类任务拆成若干个二分类任务求解。对问题进行拆分，然后为拆分的每个二分类任务训练一个分类器；在测试时，对这些分类器的预测结果进行集成以及获得最终的多分类结果。

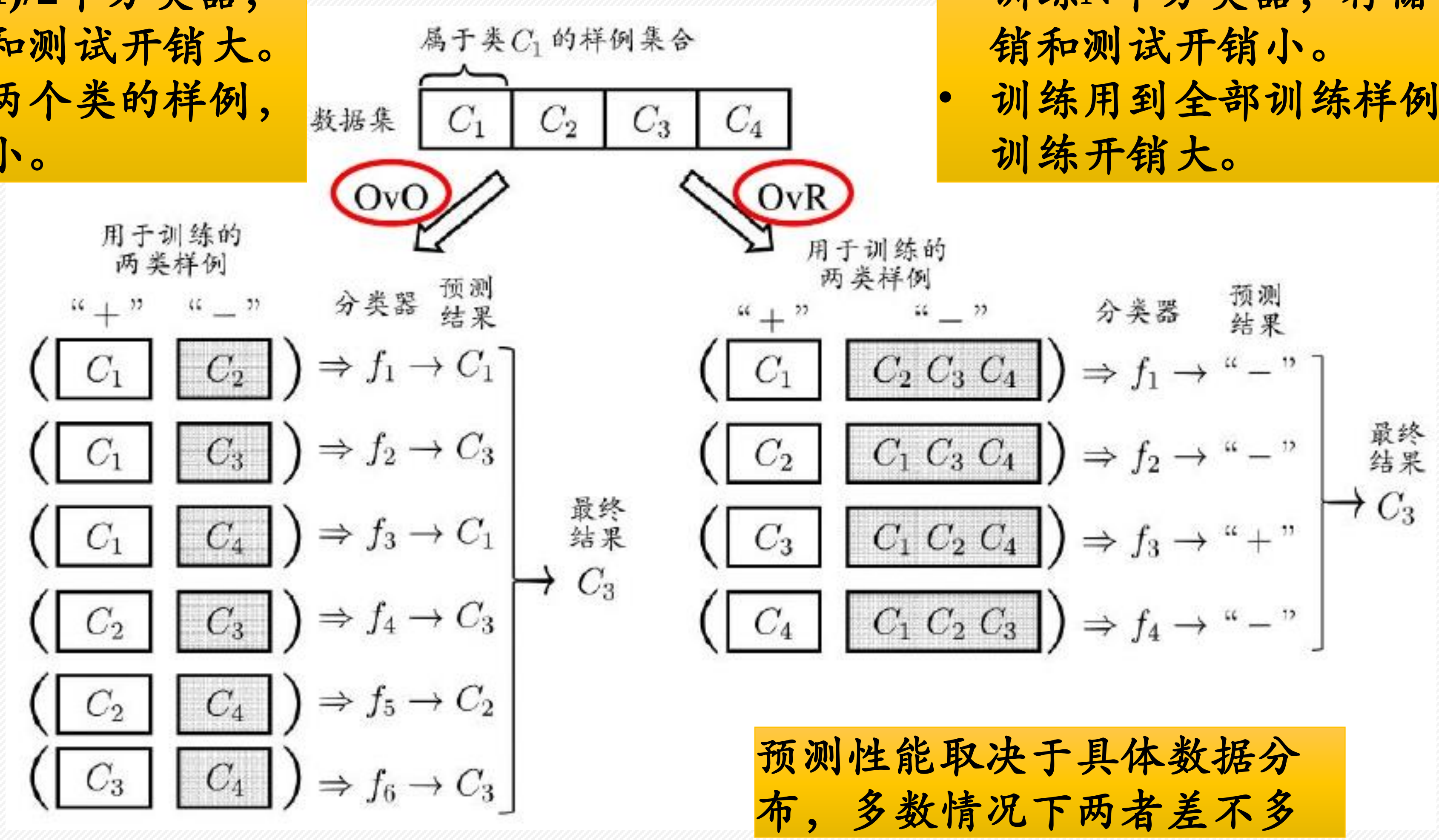
■**经典的拆分策略**

- **一对一**（OvO）：将N个类别两两配对，产生$N(N-1)/2$个二分类任务。
- **一对其余**（OvR）：每次将一个类的样例作为正例、所有其他类的样例作为反例来训练N个分类器。
- **多对多**（MvM）：每次将若干个类作为正类，若干个其他类作为反类。

3.5 多分类学习

- 训练N(N-1)/2个分类器，存储开销和测试开销大。
- 训练只用两个类的样例，训练开销小。

- 训练N个分类器，存储开销和测试开销小。
- 训练用到全部训练样例，训练开销大。



预测性能取决于具体数据分布，多数情况下两者差不多

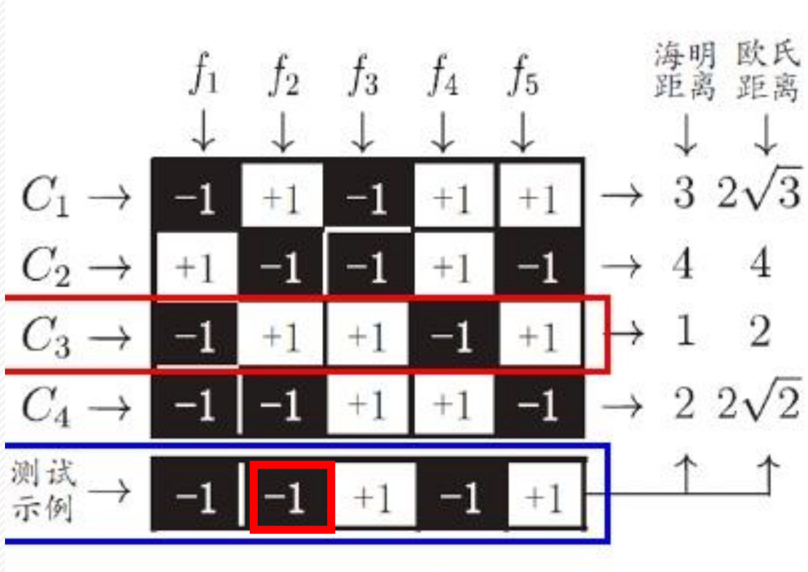
纠错输出码 (ECOC)

多对多(Many vs Many, MvM): 将若干类作为正类，若干类作为反类。
常见方法：**纠错输出码**(Error Correcting Output Code, ECOC)

① **编码**：对 N 个类别做 M 次划分，每次将一部分类别划为正类，一部分划为反类。从而形成一个二分类训练集。这样一共产生 M 个训练集，可训练出 M 个分类器。

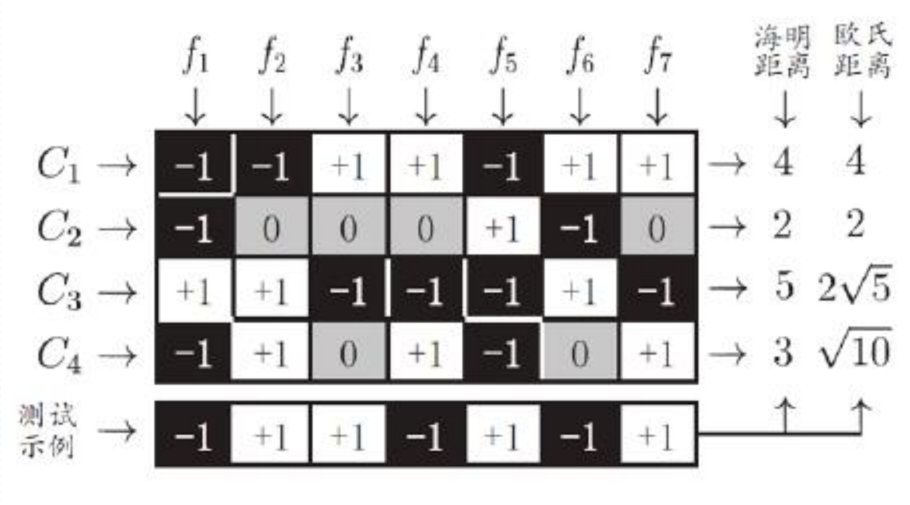
② **解码**： M 个分类器分别对测试样本进行预测，这些预测标记组成一个编码。将这个预测编码与每个类别各自的编码进行比较，返回其中距离最小的类别作为最终预测结果。

纠错输出码 (ECOC)



(a) 二元ECOC码

[Dietterich and Bakiri,1995]



(b) 三元ECOC码

[Allwein et al. 2000]

- **ECOC编码对分类错误有一定容忍和修正能力，编码越长、纠错能力越强**
- 对同等长度的编码，理论上来说，任意两个类别间的编码距离越远，则纠错能力越强

3.6 类别不平衡问题

■**基本假设**：不同类别的训练样本数目相当。

■**类别不平衡问题（class imbalance）**：分类任务中不同类别的训练样例数目差别很大的情况。

■类别不平衡问题不仅仅出现在原始数据集D中，而且也有可能出现在经过OvR，MvM策略后产生的二分类任务中。

3.6 类别不平衡问题

若 $\frac{y}{1-y} > 1$ 则预测为正例 ⇒ 若 $\frac{y}{1-y} > \frac{m^+}{m^-}$ 则预测为正例

基本策略

—— "再缩放" (rescaling)：

$$\frac{y'}{1-y'} = \frac{y}{1-y} \times \frac{m^-}{m^+}$$

然而，精确估计 $\frac{m^-}{m^+}$ 通常很困难！

常见类别不平衡学习方法：

- **欠采样 (undersampling)**
  例如：EasyEnsemble
- **过采样 (oversampling)**
  例如：SMOTE
- **阈值移动 (threshold-moving)**

优化学习算法

优化：改变x以最小化或最大化某个函数$f(x)$的值。通常以最小化$f(x)$为例。

- 梯度下降算法
  - batch梯度下降算法
  - stochastic梯度下降算法
  - minibatch梯度下降算法
- 动量法(Momentum)
- Nesterov加速梯度下降法
- AdaGrad
- RMSProp
- Adadelta
- Adam
- AdaMax
- Nadam

(1) 梯度下降算法(Gradient Descent, GD)

对于多元函数$f(\boldsymbol{x})$, **梯度**定义为对每个自变量的偏导数构成的向量。

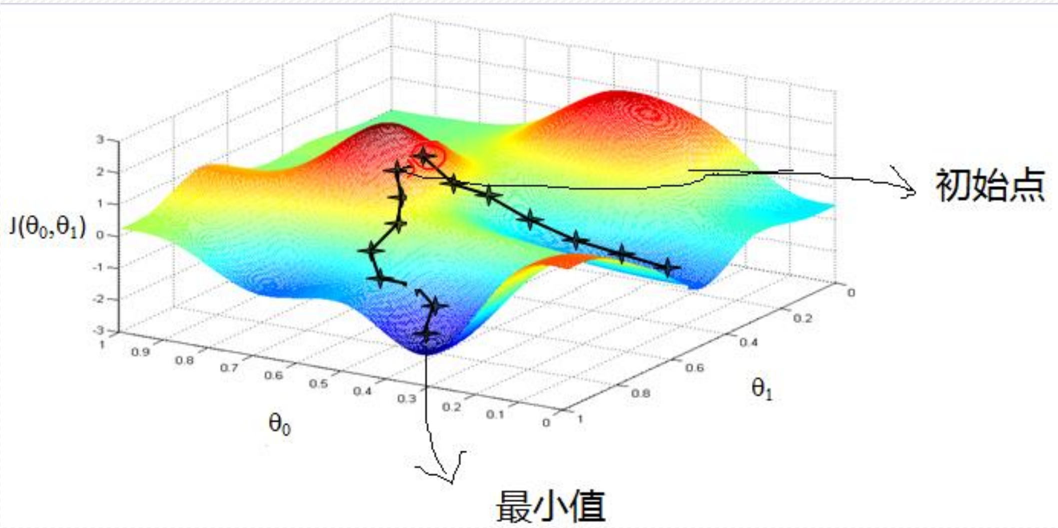
$$f'(\boldsymbol{x}) = \nabla f(x) = [\nabla f(x_1), \dots, \nabla f(x_n)]^T$$

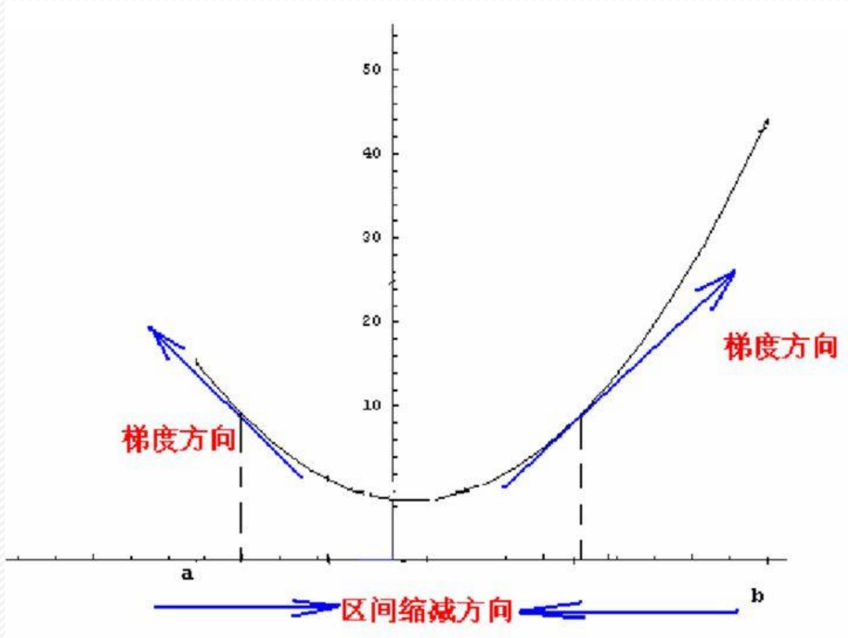
考察$f(\boldsymbol{x}+\Delta\boldsymbol{x})$在$\boldsymbol{x}$处的泰勒展开，则,

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) = f(x) + f'(\boldsymbol{x})^T\Delta\boldsymbol{x} + \boldsymbol{o}(\Delta\boldsymbol{x})$$

$$\Delta\boldsymbol{x} = -\gamma f'(\boldsymbol{x})$$

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) < f(x)$$





(1) 梯度下降算法(Gradient Descent, GD)

给定训练集$\{\boldsymbol{x}_i, y_i\}_{i=1}^{m}$，给定模型$f$，损失函数为$L(Y, f(X;\boldsymbol{\theta}))$，学习率为$\gamma$

随机初始化参数$\boldsymbol{\theta}_0$

for t=1 to .... do

  计算损失$L \leftarrow \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_{t-1}\right), \boldsymbol{y}^{(i)}\right)$

  计算梯度：$\boldsymbol{g}_t \leftarrow \frac{\partial L}{\partial \boldsymbol{\theta}}$

  更新参数：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \times \boldsymbol{g}_t$

end while

缺点：数据规模大时，模型计算所有样本的时间增加；随着数据维度的增加，梯度计算的复杂度也会增加。

梯度下降算法(Gradient Descent, GD)

随机梯度下降(Stochastic Gradient Descent, SGD)

每次从训练集中随机选择1个样本， 计算其对应的损失和梯度，进行参数更新，反复迭代。

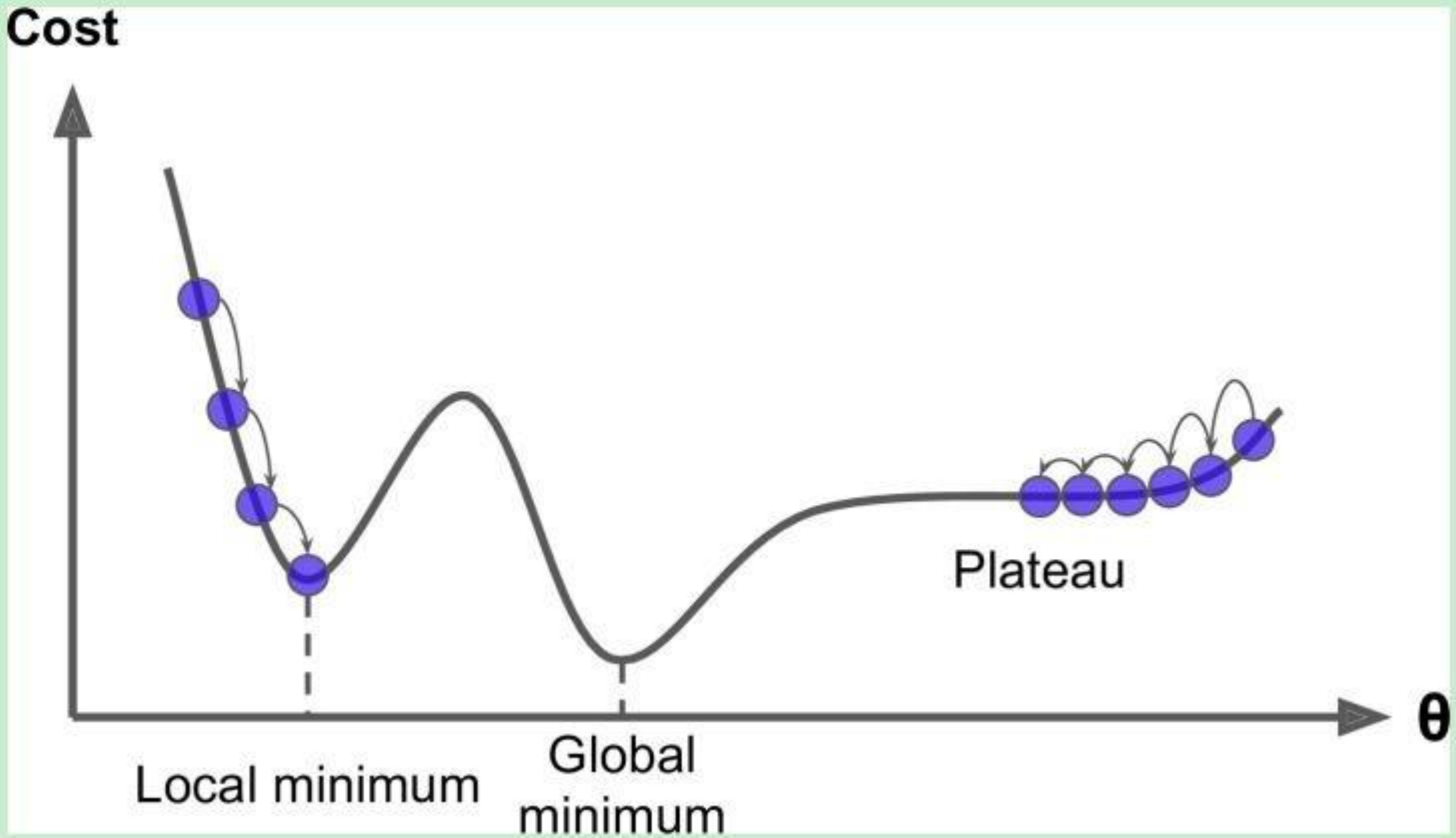
是否能保证一定收敛呢? 可以证明，这种算法的收敛性是可以保证的。
极大地减少计算量，提高计算效率；
存在着一定的不确定性，因此收敛速率相比批梯度下降得更慢。

小批量梯度下降(Mini-Batch Gradient Descent)

每次随机抽取B个样本，以它们的梯度均值作为梯度的近似估计。

(2)动量法(Momentum)

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \times \boldsymbol{g}_t$$



(2)动量法(Momentum)

■ 使用动量的随机梯度下降(SGD)

Require: 学习速率η， 动量因子μ

Require: 初始参数$\boldsymbol{\theta}_0$，初始速度$\boldsymbol{m}_{\mathbf{0}}$

while 没有达到停止准则 do

　　从训练数据中抽取m个样本$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$，对应目标为 $\boldsymbol{y}^{(i)}$ 。

　　计算梯度：$\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}_{t-1}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_{t-1}\right), \boldsymbol{y}^{(i)}\right)$

　　动量更新：$\boldsymbol{m}_t \leftarrow \gamma \boldsymbol{m}_{t-1} + \eta * \boldsymbol{g}_t$

　　应用更新：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t$

end while

在实践中，γ的一般取值为0.9 ~0.99。

(2)动量法(Momentum)

$$m_0 = 0$$

$$m_1 = \gamma m_0 + \eta g = \eta g$$

$$m_2 = \gamma m_1 + \eta g = (1+\gamma)\eta g$$

$$m_3 = \gamma m_2 + \eta g = (1+\gamma+\gamma^2)\eta g$$

$$\cdots$$

$$m_\infty = (1+\gamma+\gamma^2+\cdots)\eta g$$

$$= \frac{1}{1-\gamma}\eta g$$

(3) Nesterov加速梯度下降法

使用Nesterov动量的随机梯度下降(SGD)

Require: 学习速率η， 动量因子γ

Require:初始参数$\boldsymbol{\theta}_0$，初始速度$\boldsymbol{m}_0$

while 没有达到停止准则 do

从训练数据中抽取m个样本$\{\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)},\ldots,\boldsymbol{x}^{(m)}\}$，对应目标为 $\boldsymbol{y}^{(i)}$ 。

应用临时更新：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta * \gamma * \boldsymbol{m}_{t-1}$

计算梯度(在临时点)：$\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\widetilde{\boldsymbol{\theta}}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_t\right),\boldsymbol{y}^{(i)}\right)$

速度更新：$\boldsymbol{m}_t \leftarrow \gamma \boldsymbol{m}_{t-1} + \boldsymbol{g}_t$

应用更新：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta * \boldsymbol{m}_t$

end while

(4) AdaGrad(adaptive sub-gradient descent)

自适应梯度下降算法，能够针对参数更新的频率调整更新幅度——对于更新频繁且更新量大的参数，适当减小步长；对于更新不频繁的参数，适当增大步长。

AdaGrad 算法

Require: 全局学习率η、 初始参数$\boldsymbol{\theta}_0$，小常数δ（为数值稳定，设为10^{-7}），初始化梯度累积变量$s_0 = 0$

while 没有达到停止准则 do

从训练数据中抽取m个样本$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$，对应目标为$\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}_{t-1}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_{t-1}), \boldsymbol{y}^{(i)})$

累积平方梯度：$s_t \leftarrow s_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t$

应用更新：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\delta + \sqrt{s_t}} \boldsymbol{g}_t$

end while

(5) RMSProp

思路：让梯度积累值$\boldsymbol{g}$不要一直变大，而是按照一定的比率衰减，这样其含义就不再是梯度的积累项了，而是梯度的平均值：

RMSProp 算法

参数：学习率μ、梯度积累衰减率ρ
常数 δ，通常设为10^{-6}（用于被小数除时的数值稳定）
初始参数$\boldsymbol{\theta}_0$

while 停止条件未满足 do

从训练数据中抽取m个样本$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$，对应目标为 $\boldsymbol{y}^{(i)}$ 。

计算梯度：$\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}_{t-1}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_{t-1}\right), \boldsymbol{y}^{(i)}\right)$

累积平方梯度：$s_t = \rho s_{t-1} + (1-\rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t$

应用更新：$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\mu}{\sqrt{\delta + s_t}} \boldsymbol{g}_t$

end while

(5) Adam(Adaptive Moment Estimation)

参数：学习率η，矩估计的指数衰减速率，一阶动量衰减系数ρ_1和 ρ_2 在 [0,1)内（默认0.9 和 0.999），用于数值稳定的小常数 δ（默认：10^{-8}），初始参数 $\boldsymbol{\theta}$

初始化：一阶和二阶矩变量 $s=0, r=0$，时间步 $t=0$

while 没有达到停止准则 **do**

从训练数据中抽取m个样本$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t+1$

更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

修正一阶矩的偏差：$\hat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{1-\rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} = \frac{\boldsymbol{r}}{1-\rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} = -\eta \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$（逐元素应用操作）

end while

(6) Adadelta

参数：学习率ρ，常量ϵ，初始值x_1
初始化：$\mathrm{E}[g^2]_0 = 0$，$\mathrm{E}[\Delta x^2]_0 = 0$
for t=1:T do 停止条件未满足
　　计算梯度：$\boldsymbol{g}_t$
　　累积梯度：$\mathrm{E}[g^2]_t = \rho\mathrm{E}[g^2]_{t-1} + (1-\rho)\mathrm{E}[g^2]_t$
　　计算更新：$\mathrm{RMS}[g]_t = \sqrt{E[g^2]_t + \epsilon}$
　　累计更新：$\mathrm{E}[\Delta x^2]_t = \rho\mathrm{E}[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$
　　计算更新：$\mathrm{RMS}[\Delta x]_t = \sqrt{E[\Delta x^2]_t + \epsilon}$
　　计算更新：$\Delta x_t = -\frac{\mathrm{RMS}[\Delta x]_t}{\mathrm{RMS}[g]_t}\boldsymbol{g}_t$
　　更新参数：$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \Delta x_t$
end for

PyTorch 中的优化器

类	算法名称
torch.optim.Adadelta()	Adadelta算法
torch.optim.Adagrad()	Adagrad算法
torch.optim.Adam()	Adam算法
torch.optim.Adamax()	Adamax算法
torch.optim.ASGD()	平均随机梯度下降算法
torch.optim.LBFGSQL()	BFGS算法
torch.optim.RMSprop()	RMSprop算法
torch.optim.Rprop()	弹性反向传播算法
torch.optim.SGD()	随机梯度下降算法

CLASS torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False, *, maximize=False)

例，优化器的使用

```
import torch.nn as nn
#建立一个测试网络类
class TestNet(nn.Module):
    def __init__(self):
        super (TestNet,self).__init__()
        ##定义隐藏层
        self.hidden=nn.Sequential(nn.Linear(13,10), nn.ReLU())
        #定义预测回归层
        self.regression = nn.Linear(10, 1)
    #定义网络的向前传播路径
    def forward(self, x):
        x = self.hidden(x)
        output = self.regression(x)
        #输出为output
        return output

#并输出网络结构
model = TestNet()
```

```
from torch.optim import Adam
#使用方式1：为不同的层定义统-的学习率
optimizer = Adam(model.parameters(), lr=0.001)
```

```
#使用方式2：为不同的层定义不同的学习率
optimizer = Adam(
[{"params": model.hidden.parameters(), "lr":0.0001},
{"params": model.regression.parameters(),"lr": 0.01}],
lr=1e-2)
```

```
#对目标函数进行优化时通常的格式
for input, target in dataset:
    optimizer.zero_grad() #梯度清零
    output = model(input) #计算预测值
    loss = loss_fn(output, target) #计算损失
    loss.backward() #损失后向传播
    optimizer.step() #更新网络参数
```

优化器学习率调整

(1) lr_scheduler.LambdaLR(optimizer, lr_ lambda, last_epoch=-1): 不同的参数组设置不同的学习调整策略，last_ epoch参数用于设置何时开始调整学习率。

(2) lr._scheduler.StepLR(optimizer, step_size, gamma=0.1, last_epoch=-1): 等间隔调整学习率，学习率会每经过step_size指定间隔调整为原来的gamma倍。

(3) lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.1, last_epoch=-1):按照设定的间隔调整学习率。milestones参数通常使用一个列表来指定需要调整学习率的epoch数值，学习率会调整为原来的gamma倍。

(4) lr_scheduler.ExponentialLR(optimizer, gamma, last_epoch=-1): 按照指数衰减调整学习率，学习率调整公示为lr = lr* $gamma^{epoch}$。

(5) lr_scheduler.CosineAnnealingLR(optimizer, T_max, eta_min=0, last_epoch=-1):以余弦函数为周期，并在每个周期最大值时调整学习率，T_max表示在max个epoch后重新设置学习率，eta_min表示最小学习率，即每个周期的最小学习率不会小于eta_min。

```
scheduler = . . .   #设置学习率调整方式
for epoch in range (100) :
    train(...)
     validate(... )
     scheduler.step() #更新学习率
```

The end