第2章　线性模型

刘家锋

哈尔滨工业大学

第2章　线性模型

2.1 一元线性回归

例2.1 薪资预测(2-1 Salary.ipynb)

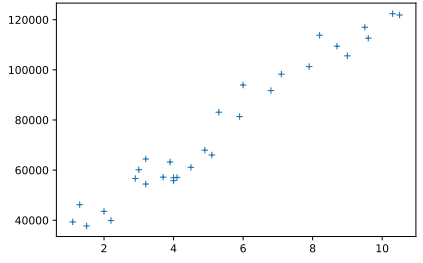
- **薪资预测**
  - 员工的薪资与工作经验相关，根据数据构建预测模型

编号	经验	薪资	编号	经验	薪资	编号	经验	薪资
1	1.1	39343	11	3.9	63218	21	6.8	91738
2	1.3	46205	12	4	55794	22	7.1	98273
3	1.5	37731	13	4	56957	23	7.9	101302
4	2	43525	14	4.1	57081	24	8.2	113812
5	2.2	39891	15	4.5	61111	25	8.7	109431
6	2.9	56642	16	4.9	67938	26	9	105582
7	3	60150	17	5.1	66029	27	9.5	116969
8	3.2	54445	18	5.3	83088	28	9.6	112635
9	3.2	64445	19	5.9	81363	29	10.3	122391
10	3.7	57189	20	6	93940	30	10.5	121872

载入和显示数据

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv("Salary_Data.csv")

x = data['YearsExperience'].to_numpy()
y = data['Salary'].to_numpy()

print("x:Years Experience\n", x)
print("y:Salary\n", y)

plt.plot(x,y,'+')
```

```
x:Years Experience
 [ 1.1  1.3  1.5  2.   2.2  2.9  3.   3.2  3.2  3.7  3.9  4.   4.   4.1
  4.5  4.9  5.1  5.3  5.9  6.   6.8  7.1  7.9  8.2  8.7  9.   9.5  9.6
 10.3 10.5]
y:Salary
 [ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
  57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
  81363.  93940.  91738.  98273. 101302. 113812. 109431. 105582. 116969.
 112635. 122391. 121872.]
[ 1.1  1.3  1.5  2.   2.2  2.9  3.   3.2  3.2  3.7  3.9  4.   4.   4.1
  4.5  4.9  5.1  5.3  5.9  6.   6.8  7.1  7.9  8.2  8.7  9.   9.5  9.6
 10.3 10.5]
```



一元线性回归

- **一元线性回归问题**
  - 输入x是一元的标量，回归问题希望学习一组参数w, b，使得输入训练数据x_i，线性模型的输出尽量接近标记y_i：

$$f(x_i) = wx_i + b \quad \rightarrow \quad y_i$$

  - 线性回归可以通过最小化训练集上的平方误差，学习一组最优的模型参数：

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

  这种模型求解的方法称为“最小二乘法”

最小二乘法求解

优化的目标函数：

$$E(w,b)=\sum_{i=1}^{m}\left(f(x_i)-y_i\right)^2=\sum_{i=1}^{m}\left(wx_i+b-y_i\right)^2$$

分别对w和b求导数，并求极值：

$$\frac{\partial E(w,b)}{\partial w}=2\left(w\sum_{i=1}^{m}x_i^2-\sum_{i=1}^{m}(y_i-b)x_i\right)=0$$

$$\frac{\partial E(w,b)}{\partial b}=2\left(mb-\sum_{i=1}^{m}(y_i-wx_i)\right)=0$$

得到：

$$w^*=\frac{\sum_{i=1}^{m}y_i(x_i-\bar{x})}{\sum_{i=1}^{m}x_i^2-\frac{1}{m}\left(\sum_{i=1}^{m}x_i\right)^2}\qquad b^*=\frac{1}{m}\sum_{i=1}^{m}(y_i-wx_i)$$

一元最小二乘回归代码(LinearRegression_1.py)

```
import numpy as np

class LinearRegression:
    def __init__(self):
        self.w = 0
        self.b = 0

    def fit(self,x,y):
        self.w = (y*(x-x.mean())).sum() / ((x**2).sum() - ((x.sum())**2)/x.size)
        self.b = (y-self.w*x).mean()

    def predict(self,x):
        return(self.w*x + self.b)
```

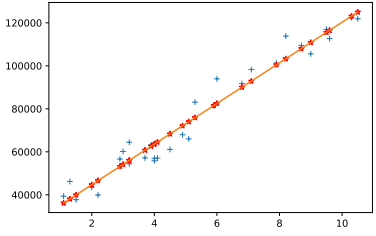
代码测试

```
if __name__ == '__main__':
    x = np.array([1,2,3,4,5,6,7,8,9,10])
    y = 1.45*x - 2.5

    LR = LinearRegression()
    LR.fit(x,y)

    print("\n w* = ",LR.w,", b* = ",LR.b)

    yy = LR.predict(x)

    print("\ny:\n",y)
    print("Predict y:\n",yy,"\n")
```

```
 w* =  1.45 ,  b* =  -2.5

y:
 [-1.05  0.4   1.85  3.3   4.75  6.2   7.65  9.1  10.55 12.  ]
Predict y:
 [-1.05  0.4   1.85  3.3   4.75  6.2   7.65  9.1  10.55 12.  ]
```

测试Salary数据(2-1 Salary.ipynb)

```
from LinearRegression_1 import LinearRegression

LR = LinearRegression()
LR.fit(x,y)
predict_y = LR.predict(x)

plt.plot(x,y,'+')
plt.plot(x,predict_y,'r*')
plt.plot(x,predict_y)

plot.show()
```



回归的质量评价

- **决定系数(coefficient of determination)**
  - 利用数据学习得到的模型质量如何？是否能够很好地模型化数据？
  - 方式一：计算回归的平方误差

$$E = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - wx_i - b)^2$$

缺点是与y的绝对值大小有关，无法反应相对回归误差的大小

  - 方式二：决定系数

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2}, \qquad \bar{y} = \frac{1}{m} \sum_{i=1}^{m} y_i$$

值越接近于1，回归的质量越好，越接近于0越差

R^2的代码

- 计算R^2的代码

```
def score(self,x,y):
    predict_y = self.w*x + self.b
    E = ((y-predict_y)**2).sum()
    R2 = 1 - E/(((y-y.mean())**2).sum())

    return(R2)
```

- 测试代码

```
print(LR.score(x,y))
```

2.2 多元线性回归

多元线性回归

- **基本形式**
  - 需要预测的y可能与很多因素相关，例如薪资不仅与工作年限有关，也与学历、职位等等有关
  - 多元线性回归是多个因素的线性组合：

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = \sum_{i=1}^{d} w_ix_i + b$$

  - 可以写成向量形式：

$$f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + b$$

其中：

$$\mathbf{x} = (x_1, \cdots, x_d)^t, \qquad \mathbf{w} = (w_1, w_2, \cdots, w_d)^t$$

权向量$\mathbf{w}$和偏置b是需要学习的模型参数

模型的学习

- **多元线性回归问题**
  - 参数$\mathbf{w}, b$的学习仍然需要训练数据$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_m, y_m)\}$
  - 学习的目标是使得输入训练数据$\mathbf{x}_i$，模型的输出尽量接近标记y_i：

$$f(\mathbf{x}_i) = \mathbf{w}^t \mathbf{x}_i + b \quad \rightarrow \quad y_i$$

  - 需要求解最小二乘问题：

$$\begin{aligned}(\mathbf{w}^*, b^*) &= \arg\min_{\mathbf{w},b} \sum_{i=1}^{m} (f(\mathbf{x}_i) - y_i)^2 \\ &= \arg\min_{\mathbf{w},b} \sum_{i=1}^{m} (\mathbf{w}^t \mathbf{x}_i + b - y_i)^2\end{aligned}$$

优化目标的矩阵形式

优化的目标函数可以进一步写成矩阵形式：

权值向量$\mathbf{w}$和偏置b表示为一个向量：

$$\hat{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix}$$

输入数据和输出数据分别表示为矩阵和向量：

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^t & 1 \\ \mathbf{x}_2^t & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^t & 1 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

优化的目标函数可以写成矩阵形式：(参见“补充数学知识”)

$$E(\hat{\mathbf{w}}) = \|X\hat{\mathbf{w}} - \mathbf{y}\|^2 = \sum_{i=1}^{m} (\mathbf{w}^t \mathbf{x}_i + b - y_i)^2$$

矩阵形式求解

优化的目标函数：

$$E(\hat{\mathbf{w}}) = \|X\hat{\mathbf{w}} - \mathbf{y}\|^2 = (X\hat{\mathbf{w}} - \mathbf{y})^t(X\hat{\mathbf{w}} - \mathbf{y})$$
$$= \hat{\mathbf{w}}^t X^t X\hat{\mathbf{w}} - 2\hat{\mathbf{w}}^t X^t\mathbf{y} + \mathbf{y}^t\mathbf{y}$$

对$\hat{\mathbf{w}}$求导数，并求极值：(参见“补充数学知识”)

$$\frac{\partial E(\hat{\mathbf{w}})}{\partial \hat{\mathbf{w}}} = 2X^t X\hat{\mathbf{w}} - 2X^t\mathbf{y} = 0$$

当X^tX为可逆矩阵时：

$$\hat{\mathbf{w}}^* = (X^tX)^{-1}X^t\mathbf{y}$$

其中，$(X^tX)^{-1}X^t$称为X的伪逆矩阵

多元最小二乘回归代码(LinearRegression_d.py)

```
import numpy as np

class LinearRegression:
    def __init__(self):
        self.w = 0
        self.b = 0

def fit(self,X,y):
    (m,d) = X.shape
    EX = np.append(X,np.ones([m,1]),axis=1)
    pinvX = np.matmul( np.linalg.inv(np.matmul(EX.T,EX)), EX.T )

    wHat = np.matmul(pinvX,y)
    self.w = wHat[0:-1]
    self.b = wHat[d-1]

def predict(self,x):
    return(np.matmul(x,self.w) + self.b)
```

代码测试

```
if __name__ == '__main__':
    x1 = np.array([[1.1,2.2,3.1,4.5,5.4,6.2,
                    7.8,8.4,9.5,10]]).T
    x2 = np.array([[10,9,8,7,6,5,4,3,2,1]]).T
    y = 1.45*x1 - 2.31*x2 - 2.5 \
           + np.random.normal(0,0.1,(10,1))

    X = np.append(x1,x2,axis=1)

    LR = LinearRegression()
    LR.fit(X,y)

    print("\n w* = ",LR.w,"\nb* = ",LR.b)

    yy = LR.predict(X)

    print("\ny Predict_y\n",np.append(y,yy,axis=1))
    print("\n R2 = ", LR.score(X,y))
```

```
w* =  [[ 1.43961535]
 [-2.32461601]]
b* =  [-2.32461601]

      y          Predict_y
 [[-23.98797077 -23.98719923]
 [-20.02585314 -20.07900634]
 [-16.54169829 -16.45873652]
 [-12.24042344 -12.11865903]
 [ -8.59425316  -8.49838921]
 [ -5.14853377  -5.02208092]
 [ -0.47325001  -0.39408036]
 [  2.86716881   2.79430486]
 [  6.74065659   6.70249775]
 [  9.66044284   9.74692143]]

 R2 =  0.9999409145649466
```

sklearn中的线性回归(TestSklearn.py)

```
import numpy as np
from sklearn.linear_model import LinearRegression

x1 = np.array([[1.1,2.2,3.1,4.5,5.4,6.2,7.8,8.4,9.5,10]]).T
x2 = np.array([[10,9,8,7,6,5,4,3,2,1]]).T
y = 1.45*x1 - 2.31*x2 - 2.5 + np.random.normal(0,0.1,(10,1))

X = np.append(x1,x2,axis=1)

LR = LinearRegression()
LR.fit(X,y)

print("\n w* = ",LR.coef_,"\nb* = ",LR.intercept_)

yy = LR.predict(X)

print("\n y Predict_y\n", np.append(y,yy,axis=1) )
print("\n R2 = ", LR.score(X,y))
```

例2.2 Boston房价预测

- **房价预测**
  - 根据房屋的属性预测房价

编号	属性	含义
x_1	per capita crime rate by town	城镇人均犯罪率
x_2	proportion of residential land zoned	住宅用地的比例
x_3	proportion of non-retail business acres	非零售客户业务比例
x_4	Charles River dummy variable	是否靠近Charles River
x_5	nitric oxides concentration	氮氧化物浓度
x_6	average number of rooms per dwelling	住宅平均房间数
x_7	proportion of owner-occupied units	自住单位的比例
x_8	distances to employment centres	距离就业中心的距离
x_9	accessibility to radial highways	可用高速公路数
x_{10}	full-value property-tax rate	税率
x_{11}	pupil-teacher ratio by town	学校师生比
x_{12}	the proportion of blacks by town	黑人居民比例
x_{13}	lower status of the population	人口密度
y	Median value of owner-occupied homes	自有房屋的价值中位数

载入Boston Housing数据(2-2 HousePrice.ipynb)

```
import pandas as pd
import numpy as np

data = pd.read_csv("Boston_Housing.csv",\
           header=None,delimiter=r"\s+")

X = data.iloc[:,0:13].to_numpy()
y = data.iloc[:,13].to_numpy().reshape(-1,1)

print("\nShape of X:", X.shape)
print("First two rows of X:\n", X[0:2,])
print("\nFirst ten row of y:\n", y[0:10])
```

```
Shape of X: (506, 13)
First two rows of X:
 [[6.3200e-03 1.8000e+01 2.3100e+00 0.0000e+00 5.3800e-01 6.5750e+00
  6.5200e+01 4.0900e+00 1.0000e+00 2.9600e+02 1.5300e+01 3.9690e+02
  4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 6.4210e+00
  7.8900e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9690e+02
  9.1400e+00]]

First ten row of y:
 [[24. ]
 [21.6]
 [34.7]
 [33.4]
 [36.2]
 [28.7]
 [22.9]
 [27.1]
 [16.5]
 [18.9]]
```

原始属性线性预测

```
from sklearn.linear_model import LinearRegression

LR = LinearRegression()
LR.fit(X,y)
predict_y = LR.predict(X)

print(" y predict_y\n", np.append(y,predict_y,axis=1))
print(LR.score(X,y))
```

```
  y            predict_y
[[24.         30.00384338]
[21.6        25.02556238]
[34.7        30.56759672]
...
[23.9        27.6274261 ]
[22.         26.12796681]
[11.9        22.34421229]]

Score: 0.7406426641094094
```

二次函数回归

- **属性扩展**
  - 保留属性1次项：$x_1, \cdots, x_{13}$
  - 增加属性2次项：$x_1^2, \cdots, x_{13}^2$
  - 增加属性交叉项：$x_1x_2, x_1x_3, \cdots, x_{12}x_{13}$
  - 构成104维的扩展属性
- **属性扩展**
  - 使用扩展属性线性回归，等价于实现了二次回归：

$$\begin{aligned}\hat{y} =& w_1x_1 + w_2x_2 + \cdots + w_{13}x_{13} \\ &+ w_{14}x_1^2 + w_{15}x_2^2 + \cdots + w_{26}x_{13}^2 \\ &+ w_{27}x_1x_2 + w_{28}x_1x_3 + \cdots + w_{103}x_{11}x_{13} + w_{104}x_{12}x_{13} + b\end{aligned}$$

属性扩展

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2,include_bias=False)

X_extend = poly.fit_transform(X)

print("Shape of Extened Features:",X_extend.shape)
```

```
Shape of Extened Features: (506, 104)
```

二次回归预测

```
from sklearn.linear_model import LinearRegression

LR = LinearRegression()
LR.fit(X_extend,y)
predict_y = LR.predict(X_extend)

print(" y predict_y\n", np.append(y,predict_y,axis=1))
print("\n Score:", LR.score(X_extend,y))
```

```
  y           predict_y
[[24.         24.7918342 ]
[21.6        22.70683946]
[34.7        32.63160085]
...
[23.9        22.83472267]
[22.         20.99475095]
[11.9        16.03188822]]

Score: 0.9289961714593017
```

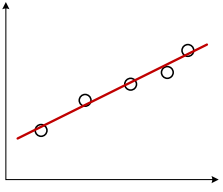
2.3 线性二分类

二分类问题

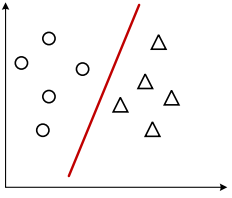
- **训练数据集**
  - 输入数据为向量$\mathbf{x}$，预测输出y标记类别：

$$y = \begin{cases} 0, & \mathbf{x}\text{是反例} \\ 1, & \mathbf{x}\text{是正例} \end{cases}$$

  - 训练数据集$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_m, y_m)\}$，$y_i \in \{0, 1\}$
- **线性回归与线性分类**
  - 线性函数同样可以用于解决两分类问题



线性回归



线性分类

最小二乘求解

- **将分类问题视为回归问题**
  - 将类别标记y_i作为输入$\mathbf{x}_i$时的“期望”输出
  - 按照线性回归的方式，学习权值向量$\mathbf{w}$和偏置b
  - 分类时，根据$z = \mathbf{w}^t\mathbf{x} + b \lessgtr 0.5$判别
- **最小二乘求解**
  - 平方误差损失函数优化：

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

  - 伪逆解：

$$\hat{\mathbf{w}}^* = (X^tX)^{-1}X^t\mathbf{y}$$

例2.3 线性二分类(2-3 Linear-classification.ipynb)

- **生成一组仿真数据**
  - 使用sklearn中的make_blobs函数，生成两个类别的数据
  - 每个类别的数据服从一个高斯分布

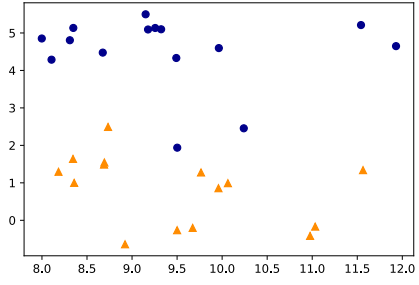
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X,y = make_blobs(centers=2, random_state=4, \
                 n_samples=30)
print("y:", y)
print("X:", X)
```

```
y: [0 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0]
X: [[ 9.76566918  1.27689813]
 [ 8.35760513  0.99907772]
 [ 9.96346605  4.59676542]
 [11.0329545  -0.16816717]
 [11.54155807  5.21116083]
 [ 9.95926647  0.85665806]
 [ 8.69289001  1.54322016]
 [ 8.1062269   4.28695977]
 [ 8.30988863  4.80623966]
 [11.93027136  4.64866327]
 [ 9.67284681 -0.20283165]
 [ 8.34810316  5.13415623]
 [ 8.67494727  4.47573059]
 [ 9.17748385  5.09283177]
 [10.24028948  2.45544401]
 [ 8.68937095  1.48709629]
 [ 8.92229526 -0.63993225]
```

显示二分类数据

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X,y=make_blobs(centers=2,random_state=4,\
               n_samples=30)
print("y:", y)

id0 = np.where(y==0)
id1 = np.where(y==1)
print("Class 0:", id0)
print("Class 1:", id1)

plt.plot(X[id0,0],X[id0,1],'^',\
         color='darkorange')
plt.plot(X[id1,0],X[id1,1],'o',\
         color="darkblue")
plt.show()
```

```
y: [0 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0]
Class 0: (array([ 0,  1,  3,  5,  6, 10, 15, 16, 20, 21, 23, 24, 25, 26, 29],
      dtype=int64),)
Class 1: (array([ 2,  4,  7,  8,  9, 11, 12, 13, 14, 17, 18, 19, 22, 27, 28],
      dtype=int64),)
```



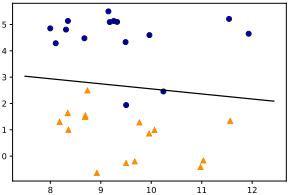
最小二乘分类

```
from sklearn.linear_model import LinearRegression

m = X.shape[0]
LR = LinearRegression()
LR.fit(X,y)
output_y = LR.predict(X)
print("output_y:\n", output_y)

id1 = np.where(output_y>0.5)
predict_y = np.zeros((m,),dtype=int)
predict_y[id1] = 1

print("\n y vs. predict_y:")
print(y)
print(predict_y)
```

```
output_y:
 [ 0.21040467  0.09021125  0.94609321 -0.05280296  1.14719108  0.12649037
  0.22355466  0.7999672   0.92231782  1.0403301  -0.11770245  0.99577164
  0.86528692  1.02166172  0.48861107  0.21111013 -0.24509014  0.86824335
  1.03377683  0.91931996  0.14786233  0.43306182  1.02901332  0.16028557
 -0.13843219  0.22970283 -0.10955641  0.34417848  1.10937381  0.29976401]

 y vs. predict_y:
[0 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0]
[0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0]
```

显示分类边界

```
eps = 0.5
x_min,x_max = X[:,0].min()-eps,X[:,0].max()+eps

xx = np.linspace(x_min, x_max, 1000)
yy = -1*(LR.coef_[0]*xx+LR.intercept_-0.5)/LR.coef_[1]

id0 = np.where(y==0); id1 = np.where(y==1)

plt.plot(X[id0,0],X[id0,1],'^',color='darkorange')
plt.plot(X[id1,0],X[id1,1],'o',color="darkblue")
plt.plot(xx,yy,'black')
plt.show()
```



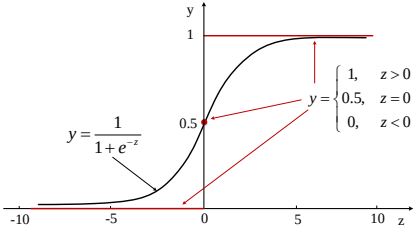
对数几率回归

- **阶跃函数与Sigmoid函数**
  - 线性分类需要使用一个阶跃函数将输出映射为类别标记：

$$y = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0.5 \\ 1, & z > 0 \end{cases}, \qquad z = \mathbf{w}^t \mathbf{x} + b$$

  - 阶跃函数不连续，可以使用Sigmoid函数近似阶跃函数：

$$y = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(\mathbf{w}^t \mathbf{x} + b)}}$$



对数几率回归

- **Logistic Regression**
  - 使用Sigmoid函数代替阶跃函数，并将其视为正例类别的“后验概率”：

$$P(y=1|\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}^t\mathbf{x}+b)}} = \frac{e^{\mathbf{w}^t\mathbf{x}+b}}{1+e^{\mathbf{w}^t\mathbf{x}+b}}$$

  - 显然，反例类别的后验概率：

$$P(y=0|\mathbf{x}) = 1 - P(y=1|\mathbf{x}) = \frac{1}{1+e^{\mathbf{w}^t\mathbf{x}+b}}$$

对数几率回归

- **判别方法**
  - 如果正例类别的后验概率大于反例类别，应该判别为正例，否则判别为反例：

$$y = \begin{cases} 0, & P(y=0|\mathbf{x}) > P(y=1|\mathbf{x}) \\ 1, & P(y=0|\mathbf{x}) < P(y=1|\mathbf{x}) \end{cases}$$

  - 计算后验概率之比的对数：

$$\ln \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \mathbf{w}^t\mathbf{x} + b \begin{cases} < 0, & y=0 \\ > 0, & y=1 \end{cases}$$

    得到与阶跃函数相同的判别条件

对数几率回归

- **参数的学习**
  - 给定数据集$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$，学习模型的参数$\mathbf{w}, b$
  - 希望学习到的参数使得数据集D中样本总的后验概率最大
  - 极大似然估计，优化对数似然函数(详细内容见第5章)：

$$(\mathbf{w}^*, b^*) = \arg\max_{\mathbf{w},b} \sum_{i=1}^{m} \ln P(y = y_i|\mathbf{x}_i; \mathbf{w}, b)$$

- **类别的后验概率**
  - 类别的后验概率满足Bernoulli分布：

$$P(y = y_i|\mathbf{x}_i) = P(y = 1|\mathbf{x}_i)^{y_i} \times P(y = 0|\mathbf{x}_i)^{1-y_i}$$

极大似然估计

简化符号，令：

$$\hat{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix}, \qquad \hat{\mathbf{x}}_i = \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$$

将Logistic Regression代入后验概率的对数：

$$\begin{aligned} \ln P(y = y_i | \hat{\mathbf{x}}_i; \hat{\mathbf{w}}) &= y_i \ln P(y = 1 | \hat{\mathbf{x}}_i; \hat{\mathbf{w}}) + (1 - y_i) \ln P(y = 0 | \hat{\mathbf{x}}_i; \hat{\mathbf{w}}) \\ &= y_i \left[\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i - \ln \left(1 + e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}\right) \right] - (1 - y_i) \ln \left(1 + e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}\right) \\ &= y_i \hat{\mathbf{w}}^t \hat{\mathbf{x}}_i - \ln \left(1 + e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}\right) \end{aligned}$$

极大似然估计等价于最小化：

$$\hat{\mathbf{w}}^* = \arg \min_{\hat{\mathbf{w}}} \sum_{i=1}^{m} \left[-y_i \hat{\mathbf{w}}^t \hat{\mathbf{x}}_i + \ln \left(1 + e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}\right) \right]$$

无法直接根据极值点条件得到$\hat{\mathbf{w}}^*$的解析解，需要采用梯度法迭代优化(参见“补充数学知识”)

对数几率回归

对数似然的优化目标函数：

$$l(\hat{\mathbf{w}}) = \sum_{i=1}^{m} \left[-y_i \hat{\mathbf{w}}^t \hat{\mathbf{x}}_i + \ln\left(1 + e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}\right) \right]$$

计算梯度：

$$\begin{aligned} \nabla l(\hat{\mathbf{w}}) &= \sum_{i=1}^{m} \left[-y_i \hat{\mathbf{x}}_i + \frac{e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}}{1 + e^{\hat{\mathbf{w}}^t \hat{\mathbf{x}}_i}} \hat{\mathbf{x}}_i \right] \\ &= \sum_{i=1}^{m} \left(P(y = 1 | \hat{\mathbf{x}}_i; \hat{\mathbf{w}}) - y_i \right) \hat{\mathbf{x}}_i \end{aligned}$$

对数几率回归

Algorithm 1 Logistic Regression的梯度学习

Input: 数据集$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$，学习率$\eta$，收敛精度$\theta$

Output: 模型的参数$\hat{\mathbf{w}}^* = (\mathbf{w}^*; b^*)$

随机初始化$\hat{\mathbf{w}}_0$，$k = 0$，$l_0 = 0$；

repeat

　计算后验概率：$P(y = 1|\hat{\mathbf{x}}_i; \hat{\mathbf{w}}_k), \quad i = 1, \cdots, m$

　学习参数：

$$\hat{\mathbf{w}}_{k+1} \leftarrow \hat{\mathbf{w}}_k - \eta \sum_{i=1}^m \left(P(y = 1|\hat{\mathbf{x}}_i; \hat{\mathbf{w}}_k) - y_i\right) \hat{\mathbf{x}}_i$$

　计算对数似然：

$$l_{k+1} = \sum_{i=1}^m \ln P(y = y_i|\hat{\mathbf{x}}_i; \hat{\mathbf{w}}_{k+1})$$

　$k \leftarrow k + 1$

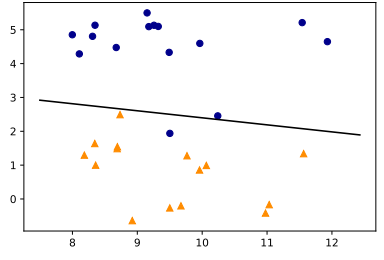
until $|l_k - l_{k-1}| < \theta$

return $\hat{\mathbf{w}}^* = \hat{\mathbf{w}}_k$；

例2.3(续) 对数几率回归(2-3 Linear-classification.ipynb)

```
from sklearn.linear_model import LogisticRegression

CLF = LogisticRegression().fit(X, y)
predict_y = CLF.predict(X)
print("y vs. predict_y:")
print(y), print(predict_y)
print("\nw and b:\n", CLF.coef_,CLF.intercept_)

xx = np.linspace(x_min, x_max, 1000)
yy = -1*(CLF.coef_[0][0]*xx+CLF.intercept_)\
          /CLF.coef_[0][1]

id0 = np.where(y==0)
id1 = np.where(y==1)

plt.plot(X[id0,0],X[id0,1],'^',color='darkorange')
plt.plot(X[id1,0],X[id1,1],'o',color="darkblue")
plt.plot(xx,yy,'black')
```

```
y vs. predict_y:
[0 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0]
[0 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0]

w and b:
 [[0.34745716 1.67046518]] [-7.48104489]
```



例2.4 糖尿病分类预测

- **糖尿病预测**
  - 根据相关因素预测，预测患者是否患有糖尿病

编号	属性	含义
x_1	Number of times pregnant	怀孕次数
x_2	Plasma glucose concentration	血浆葡萄糖浓度
x_3	Diastolic blood pressure	舒张压
x_4	Triceps skin fold thickness	三头肌皮下脂肪
x_5	2-Hour serum insulin	2小时血清胰岛素
x_6	Body mass index	体重指数
x_7	Diabetes pedigree function	糖尿病家族遗传因素
x_8	Age	年龄
y	Class variable	是否患有糖尿病

例2.4 糖尿病分类预测(2-4 Diabetes.ipynb)

```
import numpy as np
import pandas as pd

data = pd.read_csv("Diabetes.csv")

X = data.iloc[:,0:8].to_numpy(); y = data.iloc[:,8].to_numpy()

print("X:",X.shape); print(X[0:4,:])
print("\ny:",y.shape); print(y[1:100])
```

```
X: (768, 8)
[[6.00e+00 1.48e+02 7.20e+01 3.50e+01 0.00e+00 3.36e+01 6.27e-01 5.00e+01]
 [1.00e+00 8.50e+01 6.60e+01 2.90e+01 0.00e+00 2.66e+01 3.51e-01 3.10e+01]
 [8.00e+00 1.83e+02 6.40e+01 0.00e+00 0.00e+00 2.33e+01 6.72e-01 3.20e+01]
 [1.00e+00 8.90e+01 6.60e+01 2.30e+01 9.40e+01 2.81e+01 1.67e-01 2.10e+01]]

y: (768,)
[0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1
 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0
 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1]
```

线性分类器学习与预测

```
from sklearn.linear_model import LogisticRegression

CLF = LogisticRegression().fit(X, y)
predict_y = CLF.predict(X)

print("y == predict_y:"); print((y==predict_y)[0:100])
print("Accuate rate:", CLF.score(X,y))
```

```
y == predict_y:
[ True  True  True  True  True  True False False  True False  True  True
  False  True  True False False False  True False  True  True  True False
   True False  True  True False  True  True  True  True  True  True  True
  False False False  True False False  True  True False  True  True  True
  False  True  True  True  True  True False  True  True  True False  True
   True  True  True  True False  True False  True  True  True False  True
   True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True False  True  True
   True  True  True False]
Accuate rate: 0.7825520833333334
```

$$\text{分类正确率(score)} = \frac{\text{正确分类样本数}}{\text{总样本数}}$$

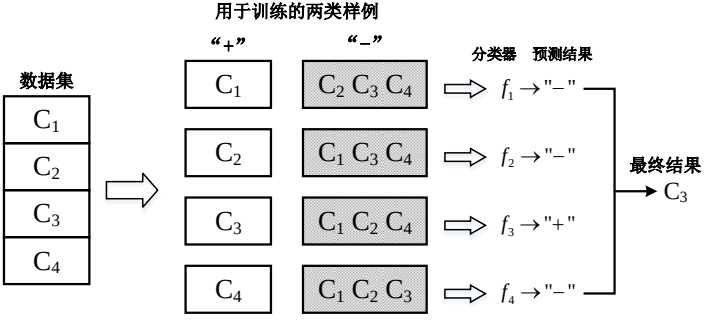
2.4 线性多分类

多分类问题

- **多分类的学习**
  - 将二分类扩展到多分类问题，有N个类别$C_1, \cdots, C_N$；
  - 给定数据集$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_m, y_m)\}, y_i \in \{1, \cdots, N\}$
  - 学习一组线性模型，用于区分N个类别；
- **学习方法**
  - 拆分法：多分类转化为多个二分类问题
  - 输出编码：用向量作为类别标记，同时学习多个线性模型

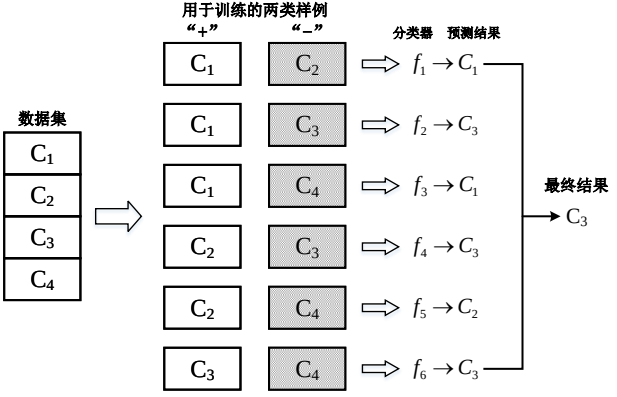
拆分方式一

- **“一对其余”方式：One vs. Rest**
  - 每次将一类作为正例，其余的作为反例，学习N个分类器
  - 测试阶段，如果仅有一个分类器预测为正例，则得到最终分类结果



拆分方式二

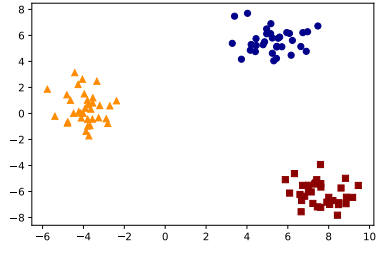
- **“一对一”方式：One vs. One**
  - 学习$N(N-1)/2$个分类器，每个分类器区分两个类别
  - 测试阶段，产生$N(N-1)/2$个二分类结果，投票获得多分类结果



例2.5 多类别分类(2-5 Multiclass-classification.ipynb)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X,y = make_blobs(centers=3,random_state=26,\
           n_samples=100)

print("y:", y)

id0 = np.where(y==0)
id1 = np.where(y==1)
id2 = np.where(y==2)

plt.plot(X[id0,0],X[id0,1],'^',color='darkorange')
plt.plot(X[id1,0],X[id1,1],'o',color="darkblue")
plt.plot(X[id2,0],X[id2,1],'s',color="darkred")
plt.show()
```

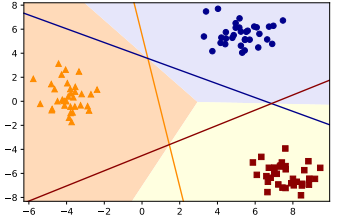
```
y: [1 2 0 2 0 0 1 1 1 0 2 1 0 2 1 0 1 1 0 0 2 0 1 1 1 1 0 1 2 0 1 2 1 0 1 2 2
 1 2 2 1 0 0 2 1 2 2 0 0 0 2 1 2 2 0 2 0 0 2 0 1 0 1 1 0 1 2 2 2 1 1 2 1 0 1
 2 0 2 2 0 0 1 2 0 0 0 2 2 2 0 1 0 2 2 0 1 2 1 2 1 0 2 1]
```



One vs. Rest方式

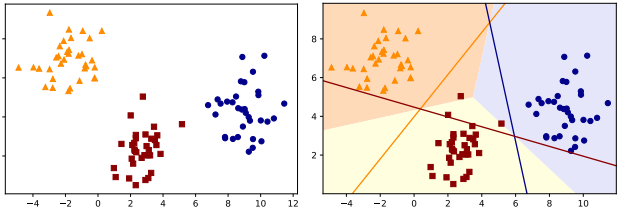
```
from sklearn.linear_model import LogisticRegression
from plot_decision_boundary import plot_decision_boundary

CLF = LogisticRegression(multi_class='ovr').fit(X, y)
print("w:\n", CLF.coef_);print("b:", CLF.intercept_)
'''部分省略'''

xx = np.linspace(x_min, x_max, 1000)
yy = -1*(CLF.coef_[0][0]*xx+CLF.intercept_[0])\
    /CLF.coef_[0][1], plt.plot(xx,yy,'darkorange')
yy = -1*(CLF.coef_[1][0]*xx+CLF.intercept_[1])\
    /CLF.coef_[1][1], plt.plot(xx,yy,'darkblue')
yy = -1*(CLF.coef_[2][0]*xx+ CLF.intercept_[2])\
    /CLF.coef_[2][1], plt.plot(xx,yy,'darkred')
plt.ylim((y_min,y_max))

plot_decision_boundary(CLF,axis=[x_min,x_max,y_min,y_max])
plt.show()
```

```
w:
 [[-1.22868189 -0.19464336]
 [ 0.68494304  1.19616244]
 [ 0.61140392 -0.96942375]]
b: [ 1.07530675 -4.4777967  -4.38128814]
```



另一组数据

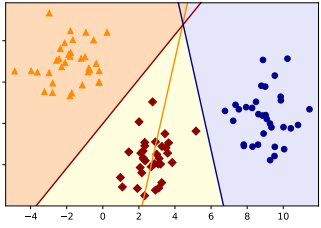
```
X,y = make_blobs(centers=3, random_state=38,n_samples=100)
print("y:", y)

id0 = np.where(y==0);id1 = np.where(y==1);id2 = np.where(y==2)

plt.plot(X[id0,0],X[id0,1],'^',color='darkorange')
plt.plot(X[id1,0],X[id1,1],'o',color="darkblue")
plt.plot(X[id2,0],X[id2,1],'s',color="darkred")
plt.show()
```



One vs. One方式

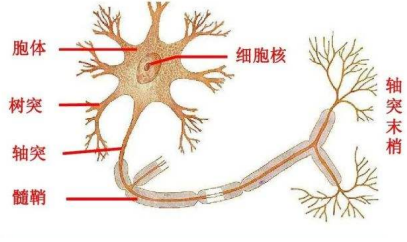
```
from sklearn.multiclass import OneVsOneClassifier

CLF = OneVsOneClassifier(LogisticRegression()).fit(X,y)
clf0=CLF.estimators_[0]; clf1=CLF.estimators_[1];
clf2=CLF.estimators_[2]

print( "w and b:")
print(clf0.coef_,clf0.intercept_)
print(clf1.coef_,clf1.intercept_)
print(clf2.coef_,clf2.intercept_)

'''部分省略'''
xx = np.linspace(x_min, x_max, 1000)
yy = -1*(clf0.coef_[0][0]*xx+clf0.intercept_)\
       /clf0.coef_[0][1]
plt.plot(xx,yy,'darkorange')
'''部分省略'''
plot_decision_boundary(CLF,axis=[x_min,x_max,y_min,y_max])
```

```
w and b:
[[ 1.0887506  -0.27997323]] [-2.36131102]
[[ 1.23309652 -1.14458655]] [4.56172557]
[[-1.62002574 -0.41305829]] [10.84242497]
```



线性模型与神经元

- **生物神经元**
  - 神经元细胞由细胞体，一条轴突和若干条树突构成
  - 轴突与其它神经元的树突相连，向其它神经元传输生物电信号
  - 突触之间的连接强度是不同的，并会发生变化

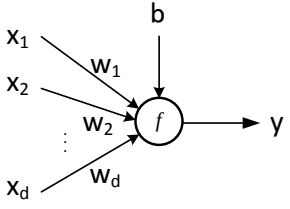


线性模型与神经元

- **人工神经元**
  - 人工神经元有多个输入端和一个输出端
  - 输出端与其它神经元的输入相连，每一个连接上有不同的权值
  - 输出y与输入$\mathbf{x}$之间的关系可以表示为：

$$y = f(\mathbf{w}^t\mathbf{x} + b) = f\left(\sum_{i=1}^{d} w_i x_i + b\right)$$

其中，f称为激活函数，$\mathbf{w}$为权值向量，b为偏置或阈值



线性模型与神经元

- **神经元与线性模型**
  - 人工神经元本质上就是一个线性模型
  - 对于回归问题，可以使用线性的激活函数：

$$y = \mathbf{w}^t\mathbf{x} + b = \sum_{i=1}^{d} w_i x_i + b$$

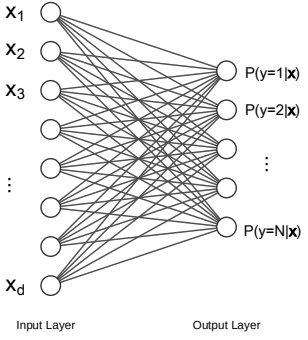
  - 对于二分类问题，可以使用Sigmoid激活函数：

$$y = f(\mathbf{w}^t\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^t\mathbf{x}+b)}}$$

    输出表示的是正例类别的后验概率：$P(y = 1|\mathbf{x})$

线性网络

- **线性神经网络**
  - 由多个神经元构成的网络可以解决多分类问题
  - 输入层神经元输入样本的属性值，传输给输出层神经元
  - 输出层神经元实现线性分类，第j个输出表示输入$\mathbf{x}$属于第j个类别的后验概率



线性网络

- **网络的概率输出**
  - 网络输出的是类别的后验概率，需要满足归一化要求

$$\sum_{j=1}^{N} P(y=j|\mathbf{x}) = 1$$

  - 输出层神经元使用Sigmoid函数无法满足归一化要求
  - 需要将Logistic Regression扩展为Multinomial Logistic Regression

多项式逻辑回归(Multinomial Logistic Regression)

以第N类为“基准”，定义其它类与第N类后验概率比值的对数为线性函数：

$$\ln \frac{P(y=1|\mathbf{x})}{P(y=N|\mathbf{x})} = \mathbf{w}_1\mathbf{x} + b_1, \cdots, \ln \frac{P(y=N-1|\mathbf{x})}{P(y=N|\mathbf{x})} = \mathbf{w}_{N-1}\mathbf{x} + b_{N-1}$$

因此：

$$P(y=j|\mathbf{x}) = P(y=N|\mathbf{x})e^{\mathbf{w}_j\mathbf{x}+b_j}, \qquad j=1,\cdots,N-1$$

第N类的后验概率：

$$P(y=N|\mathbf{x}) = 1 - \sum_{j=1}^{N-1} P(y=j|\mathbf{x}) = 1 - P(y=N|\mathbf{x}) \sum_{j=1}^{N-1} e^{\mathbf{w}_j\mathbf{x}+b_j}$$

可以得到：

$$P(y=N|\mathbf{x}) = \frac{1}{1+\sum_{j=1}^{N-1} e^{\mathbf{w}_j\mathbf{x}+b_j}}$$

$$P(y=j|\mathbf{x}) = \frac{e^{\mathbf{w}_j\mathbf{x}+b_j}}{1+\sum_{j=1}^{N-1} e^{\mathbf{w}_j\mathbf{x}+b_j}}, \qquad j=1,\cdots,N-1$$

Softmax

- 由Multinomial Logistic Regression到Softmax
  - 多项式逻辑回归的优点是只需要学习$N-1$个线性函数的参数，缺点是第N类的激活函数形式不统一
  - Softmax采用统一的函数形式，每个类别后验概率的对数定义为一个线性函数
  - 引入归一化因子Z，保证后验概率之和为1：

$$\ln P(y=j|\mathbf{x}) = \mathbf{w}_j^t\mathbf{x} + b_j - \ln Z, \qquad j = 1, \cdots, N$$

可以得到：

$$Z = \sum_{j=1}^{N} e^{\mathbf{w}_j^t\mathbf{x}+b_j}$$

  - Softmax激活函数为：

$$P(y=1|\mathbf{x}) = \frac{e^{\mathbf{w}_1^t\mathbf{x}+b_1}}{\sum_{j=1}^{N} e^{\mathbf{w}_j^t\mathbf{x}+b_j}}, \cdots, P(y=N|\mathbf{x}) = \frac{e^{\mathbf{w}_N^t\mathbf{x}+b_N}}{\sum_{j=1}^{N} e^{\mathbf{w}_j^t\mathbf{x}+b_j}}$$

线性网络的学习

- **类别标记的One-Hot编码**
  - 使用一个One-Hot的向量$\mathbf{y}_i$，表示训练样本的类别标记y_i：

$$y_i = k \qquad \Longrightarrow \qquad \mathbf{y}_i = (\overbrace{0, \cdots, 1, \cdots, 0}^{N})^t$$

    向量$\mathbf{y}_i$的第k个元素为1，其它元素为0
  - 后验概率的对数似然函数：

$$L(\mathbf{w}_1, \cdots, \mathbf{w}_N, b_1, \cdots, b_N) = \sum_{i=1}^{m} \sum_{j=1}^{N} y_{ij} P(y = j | \mathbf{x}_i)$$

    其中，y_{ij}为第i个训练样本One-Hot向量的第j个元素
  - 模型学习一般采用梯度法最小化对数似然函数的负数，称为交叉熵损失函数优化

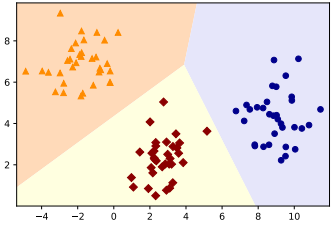
Softmax方式

```
CLF = LogisticRegression(multi_class='multinomial').fit(X,y)

print("w:\n", CLF.coef_)
print("b:\n",CLF.intercept_)

py0= np.matmul(X,CLF.coef_[0].reshape(-1,1))\
                + CLF.intercept_[0]
py1= np.matmul(X,CLF.coef_[1].reshape(-1,1))\
                + CLF.intercept_[1]
py2= np.matmul(X,CLF.coef_[2].reshape(-1,1))\
                + CLF.intercept_[2]

py = np.append(py0,py1,axis=1)
py = np.append(py,py2,axis=1)
py = np.append(py,y.reshape(-1,1),axis=1)
print("\n Posteriors Label:\n", py[0:10,:])
```

```
w:
 [[-1.20411311  0.72769566]
 [ 1.41035104  0.10440383]
 [-0.20623793 -0.83209949]]
b:
 [-0.29124979 -6.20635117  6.49760096]

   Posteriors                              Label:
 [[-8.16941513  7.04738254  1.12203258  1.        ]
 [-2.45568131 -0.97838571  3.43406702  2.        ]
 [-0.97391235 -2.62340029  3.59731264  2.        ]
 [-9.9872922   7.46413215  2.52316005  1.        ]
 [ 7.37460606 -9.49762599  2.12301993  0.        ]
 [ 7.34909209 -8.71756069  1.3684686   0.        ]
 [-1.53379348 -2.65507483  4.18886831  2.        ]
 [-0.46197283 -4.68466851  5.14664134  2.        ]
 [ 6.16700709 -6.08773191 -0.07927517  0.        ]
 [-1.29468811 -2.80500788  4.099696    2.        ]]
```

Softmax方式

```
id0 = np.where(y==0)
id1 = np.where(y==1)
id2 = np.where(y==2)

plt.plot(X[id0,0],X[id0,1],'^',color='darkorange')
plt.plot(X[id1,0],X[id1,1],'o',color="darkblue")
plt.plot(X[id2,0],X[id2,1],'D',color="darkred")

eps = 0.5
x_min,x_max=X[:,0].min()-eps, X[:,0].max()+eps
y_min,y_max=X[:,1].min()-eps, X[:,1].max()+eps

plot_decision_boundary(CLF,axis=[x_min,x_max,y_min,y_max])

plt.show()
```



2.5 模型性能测试

性能度量

- **模型性能的评价**
  - 性能度量(performance measure)是衡量模型好坏的评价标准，不同的机器学习任务需要使用不同的评价指标
  - 模型的性能好坏不能只用个别样本来评价
  - 需要以样本集合$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_m, y_m)\}$上的平均表现来度量模型的性能
  - 样本集合D称为测试集合，样本数量m越大，性能度量的越准确

回归任务的性能度量

- **均方误差(MSE:Mean Squared Error)**
  - 回归任务常用平方误差度量模型$f(\mathbf{x})$在测试集D上的性能：

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\left(f(\mathbf{x}_i)-y_i\right)^2$$

- **决定系数(coefficient of determination)**
  - 决定系数可以度量回归模型在测试集D上的相对误差大小：

$$R^2=1-\frac{\sum_{i=1}^{m}\left(f(\mathbf{x}_i)-y_i\right)^2}{\sum_{i=1}^{m}\left(y_i-\bar{y}\right)^2},\qquad \bar{y}=\frac{1}{m}\sum_{i=1}^{m}y_i$$

分类任务的性能度量

- **分类错误率**
  - 测试集D中被错误分类样本的占比：

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f(\mathbf{x}_i)\neq y_i\right)$$

其中，$\mathbb{I}(\alpha)$为指示函数

$$\mathbb{I}(\alpha)=\begin{cases}1, & \alpha=true\\ 0, & \alpha=false\end{cases}$$

- **分类精度**
  - 测试集D中被正确分类样本的占比：

$$acc(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f(\mathbf{x}_i)=y_i\right)=1-E(f;D)$$

性能评价方法

- **使用什么样的测试集评价模型的性能？**
  - 模型学习的目的是要对未来新的未知数据进行预测，模型性能的好坏应该体现对未来数据的预测能力
  - 使用训练数据作为测试集，无法准确评价模型的性能
  - 最好的评估方法是使用训练集S学习模型，使用另外的测试集T评估模型的性能
  - 数据集D既用于训练又用于测试，需要适当的划分：

$$D = S \cup T, \qquad S \cap T = \Phi$$

- **常用的划分方法**
  - 留出法(hold-out)；
  - 交叉验证法(cross validation)
  - 自助法(bootstrap)

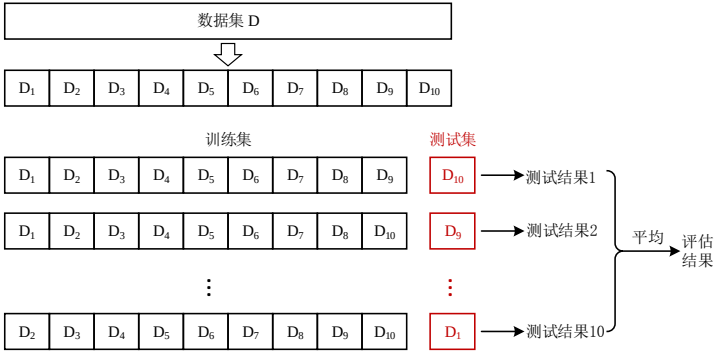
留出法

- **数据集的划分原则**
  - 保持数据分布一致性，例如分类问题中应该不同类别样本的比例是一致的
  - 测试集不宜太大或太小，一般选择样本总数的20% ~ 30%

- **评估过程**
  - 按比例将数据集D随机划分为S和T
  - 训练集S学习模型，T测试模型的性能
  - 重复划分和测试若干次，计算平均性能

交叉验证法

- **k-折交叉验证法**
  - 数据集D随机划分为k个子集
  - $k-1$个子集用于训练，1个用于测试



自助法

- **自助法的过程**
  - 从数据集D中有放回地随机抽样m个样本，构成训练集S
  - 从数据集D中有放回地随机抽样m个样本，构成测试集T
  - 重复若干次，计算平均的评估结果
- **自助法的优点**
  - 数据集D中约有30%的样本没有出现在训练集S中：

$$P(\mathbf{x} \notin S) \approx \lim_{m\to\infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e} \approx 0.368, \quad \forall \mathbf{x} \in D$$

  - S和T的样本数与D相同，测试可以进行任意多次
  - 评估结果更接近最终模型的性能
  - 自助法更适用于样本数m较小时，样本数多时留出法和交叉验证更常用

例2.6 模型性能评价(2-6 Model-Evaluation.ipynb)

- **留出法评价Boston房价数据的回归模型**
  - 数据集：Boston Housing数据
  - 属性：104维的2次扩展属性
  - 样本集划分：随机保留25%的样本作为测试集，其余75%的数据作为训练集
  - 数据集划分函数：

    X_train, X_test, y_train, y_test = train_test_split(X_extend, y, random_state=0)

  - 评价指标：决定系数R^2

数据集准备

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

data = pd.read_csv("Boston_Housing.csv",header=None,delimiter=r"\s+")

X = data.iloc[:,0:13].to_numpy()
y = data.iloc[:,13].to_numpy().reshape(-1,1)

poly = PolynomialFeatures(degree=2,include_bias=False)
X_extend = poly.fit_transform(X)

print("Shape of Extened Features:", X_extend.shape)
print("Shape of y:", y.shape)
```

```
Shape of Extened Features: (506, 104)
Shape of y: (506, 1)
```

留出法模型测试

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X_train,X_test,y_train,y_test = train_test_split(X_extend,y,random_state=0)
print("Shape of train set:", X_train.shape,y_train.shape)
print("Shape of test set:", X_test.shape,y_test.shape)

LR = LinearRegression().fit(X_train, y_train)

print("\nLinear Regression:")
print("\t Train set score: {:.2f}".format(LR.score(X_train, y_train)))
print("\t Test set score: {:.2f}\n".format(LR.score(X_test, y_test)))
```

```
Shape of train set: (379, 104) (379, 1)
Shape of test set: (127, 104) (127, 1)

Linear Regression:
        Train set score: 0.95
        Test set score: 0.61
```

模型改进

- **线性回归模型**
  - 模型学习优化平方误差函数：

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w},b} \frac{1}{m} \sum_{i=1}^{m} (\mathbf{w}^t \mathbf{x}_i + b - y_i)^2$$

- **岭回归(Ridge Regression)**
  - 模型学习优化平方误差与正则项：

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w},b} \frac{1}{m} \sum_{i=1}^{m} (\mathbf{w}^t \mathbf{x}_i + b - y_i)^2 + \alpha \|\mathbf{w}\|_2^2$$

岭回归

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=100).fit(X_train, y_train)

print("Ridge Linear Regression:")
print("\t Training set score: {:.2f}".format(ridge.score(X_train, y_train)))
print("\t Test set score: {:.2f}\n".format(ridge.score(X_test, y_test)))
```

```
Ridge Linear Regression:
        Training set score: 0.93
        Test set score: 0.76
```