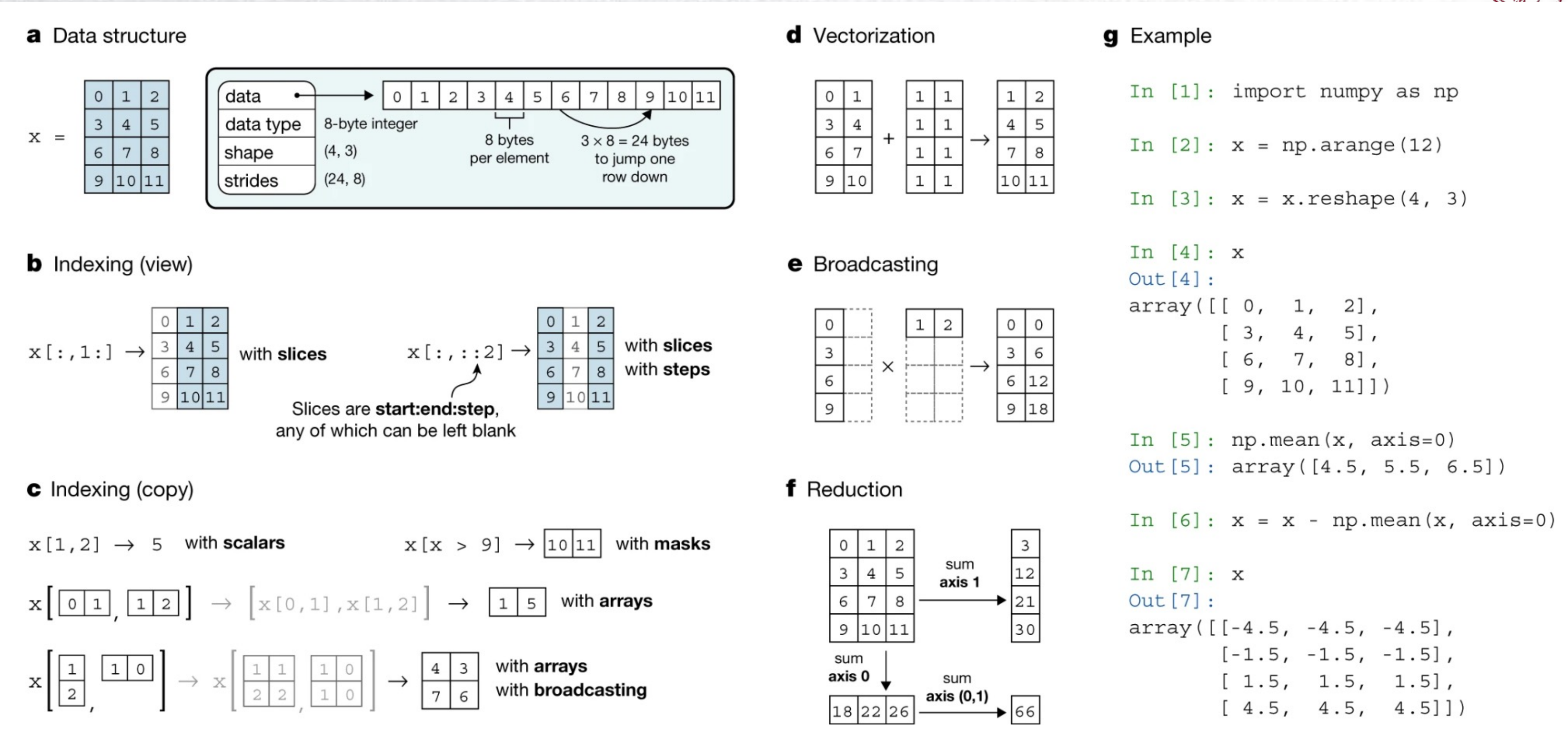
《人工智能与Python程序设计》—— 回归分析

人工智能与Python程序设计 教研组

Numpy



a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → with masks
x[0,1],x[1,2]
with arrays
with arrays with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])

练习题：如何使用普通乘法和broadcasting、reduction机制实现矩阵乘法?

$$C_{M\times N} = A_{M\times K} B_{K\times N}$$

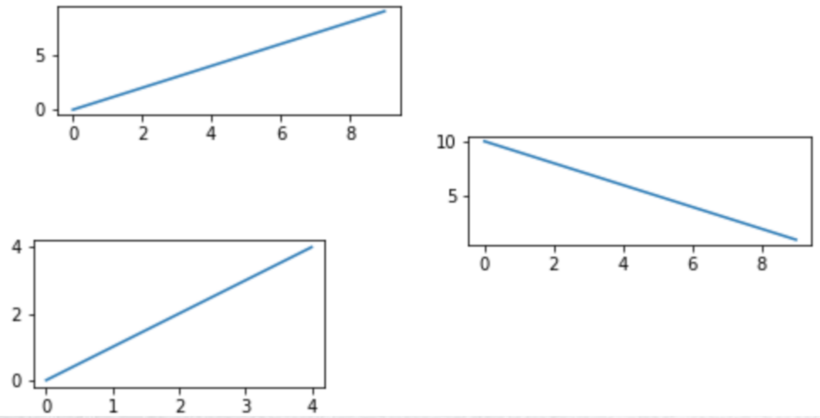
$$c_{ij} = \sum_{k=1}^{K} a_{ik} \cdot b_{kj}$$

需要计算$M\times K\times N$ 次$a_{ik} \cdot b_{kj}$

使用Matplotlib库进行图表绘制

- 绘制：
  - 散点图plt.scatter
  - 折线图plt.plot
  - 直方图plt.hist
  - 等高线图plt.contour
- 绘制多个子图：

```
# 绘图环境的设置
plt.figure(figsize=(8,4)) # 设置画布大小
plt.subplot(321) # 在一个三行两列的绘图区域中选择子区域1开始绘图
plt.plot(range(10), range(10)) # x=0..9, y=0..9

plt.subplot(324) # 在一个三行两列的绘图区域中选择子区域4开始绘图
plt.plot(range(10), range(10,0,-1))

# 直接在画布上按照[左边缘位置, 下边缘位置, 宽度, 高度]选择绘图区域
plt.axes([0.1, 0.1, 0.3, 0.3])
plt.plot(range(5), range(5))
plt.show()
```



如何绘制 $L(w, b)$的等高线图?

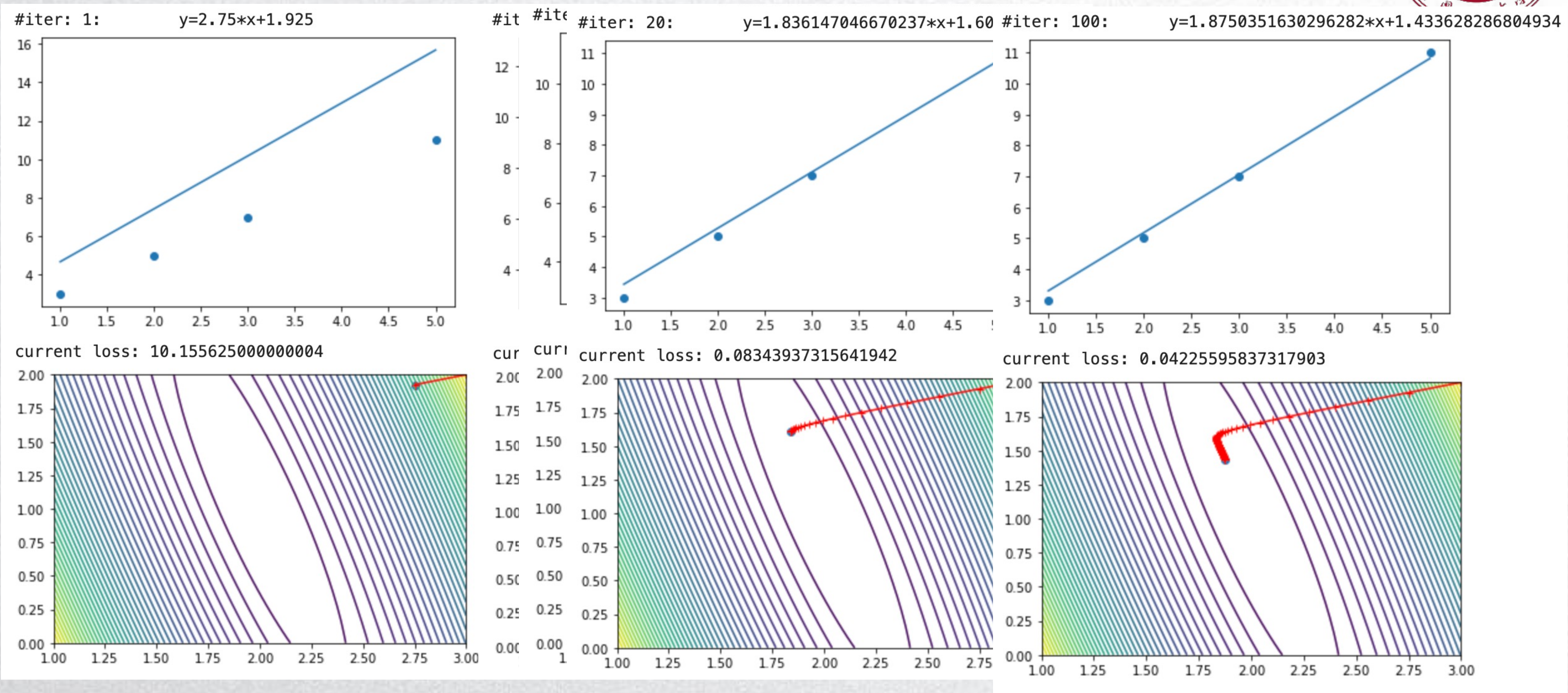
- 先生成(w, b)的坐标网格（meshgrid）

- 对网格中每一个点（j, k)计算$L(w_{j,k}, b_{j,k}; \boldsymbol{x}, \boldsymbol{y})$

$$L\left(w_{j,k}, b_{j,k}; \boldsymbol{x}, \boldsymbol{y}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(w_{j,k} \cdot x_i + b_{j,k} - y_i\right)^2$$

  – 充分利用向量化和广播机制

一元线性回归的可视化



#iter: 1:
y=2.75*x+1.925
current loss: 10.155625000000004
#iter: 20:
y=1.836147046670237*x+1.60
current loss: 0.08343937315641942
#iter: 100:
y=1.8750351630296282*x+1.433628286804934
current loss: 0.04225595837317903

一些常用名词

- 数据：
  - 一般为空间中点的集合，每个点以 (x_i, y_i) 的形式出现
  - 一般称 x_i 为“特征”， y_i 为对应的“真实值”或“观测值”
- **任务（task）**：找到由 x 得到 y 的方法，即给定 x，预测 y
  - 回归 vs 分类
- **模型（model）**：
  - 猜想 x 到 y 的映射由 $f(x)$ 给出
  - 模型即 $f(x)$ 的具体形式，带有待定参数
- **损失函数（loss function）**：根据已有数据，评价参数质量的指标
- **训练（优化）（optimization）**：找到最好的 $f(x)$ 参数的过程，即最小化损失函数的过程
- 预测：给出 x，计算 $f(x)$ 的过程
- 测试：评价预测结果

模型

- 一元线性回归:
  - $y = f(x) = w \cdot x + b$
    - 自变量x和参数 w和 b均为一元变量（标量）
  - $\hat{y}_i = w \cdot x_i + b$

- 多元线性回归:
  - $y = f(x) = \boldsymbol{x} \cdot \boldsymbol{w}$
    - 在$\boldsymbol{x}$后面加上一维值恒为1的特征，$\boldsymbol{x} \leftarrow [\boldsymbol{x}, 1]$
    - 自变量$\boldsymbol{x}$为d+1维行向量（1 x (d+1)）
    - 参数$\boldsymbol{w}$为d维列向量（(d+1) x 1)
  - $\widehat{y}_i = \boldsymbol{x}_i \cdot \boldsymbol{w}$

$$\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,j}, \dots, x_{i,d}, 1)$$

$$\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_j \\ \dots \\ w_d \\ w_{d+1} \end{pmatrix}$$

损失函数

- 使用任何一组参数都可以得到一组预测值$\hat{y}$，需要一个标准来对预测结果进行度量：定量化一个目标函数式度量预测结果的好坏。
- MSE(mean square error)均方误差：线性模型应用于训练样本x_i，得到一组预测值$\hat{y}_i$，将预测值和真实值 y_i之间的平均的平方距离定义为损失函数：

$$L = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

N为训练集样本总数

- 一元线性回归：

$$L(w, b) = \frac{1}{N}\sum_{i=1}^{N}(w \cdot x_i + b - y_i)^2$$

- 多元线性回归：

$$L(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)^2$$

优化：梯度下降法

- 1. 随机选择一组初始参数(w^0, b^0)
- 2. 对于可微函数，梯度方向时函数增长速度最快的方向，梯度的反方向是函数减少最快的方向。计算损失函数在当前参数点处的梯度。
- 3. 从当前参数点向梯度相反的方向移动，移动步长为:

$$w^{t+1} = w^t - \boxed{lr}\frac{\partial L(w,b)}{\partial w}(w^t)$$

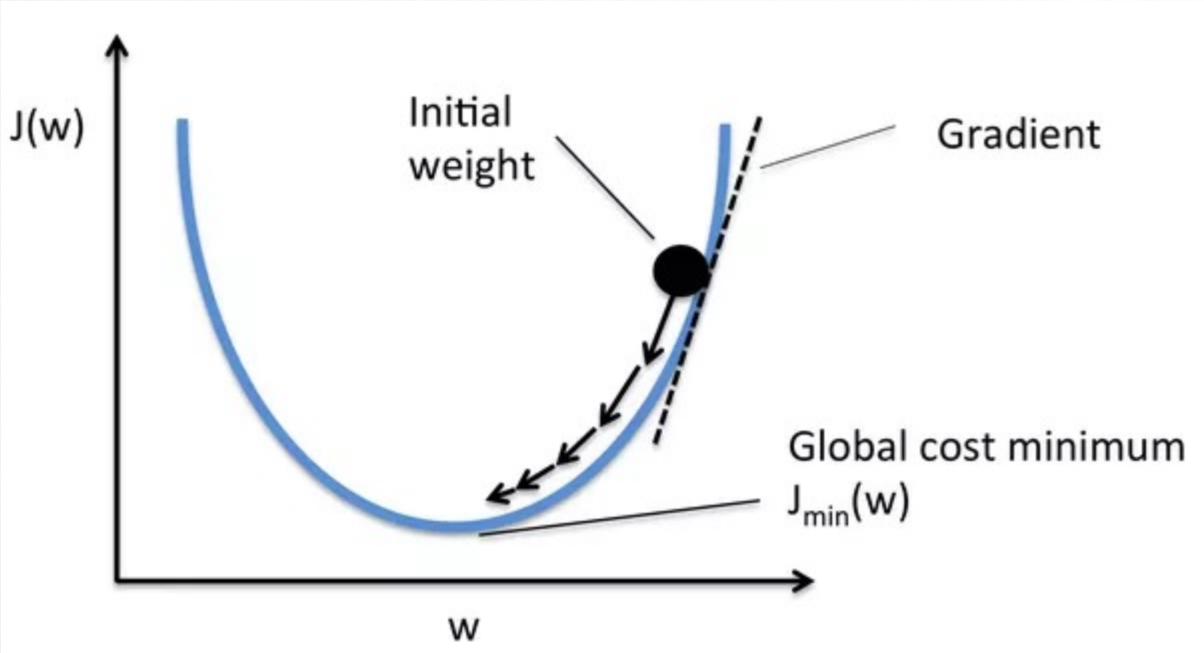
$$b^{t+1} = b^t - \boxed{lr}\frac{\partial L(w,b)}{\partial b}(b^t)$$

学习率

- 4. 循环迭代步骤 3， 直到前后两次迭代得到的(w^t, b^t)差值足够小，即参数基本不再变化，说明此时损失函数已经达到局部最小值。
- 5. 输出(w^t, b^t)，即为使得损失函数最小时的参数取值 。

梯度下降法

- 一元线性回归



优化

- 一元线性回归：
  - 对w求导：

$$\frac{\partial L(w,b)}{\partial w}=\frac{1}{N}\sum_{i=1}^{N}2(wx_i+b-y_i)x_i$$

$$\frac{\partial L(w,b)}{\partial w}=\frac{2}{N}\boldsymbol{x}^{\mathrm{T}}(w\boldsymbol{x}+b\mathbf{1}-\mathbf{y})$$

$\boldsymbol{x}$、$\mathbf{1}$、 $\boldsymbol{y}$为$N\times1$的向量（列向量）

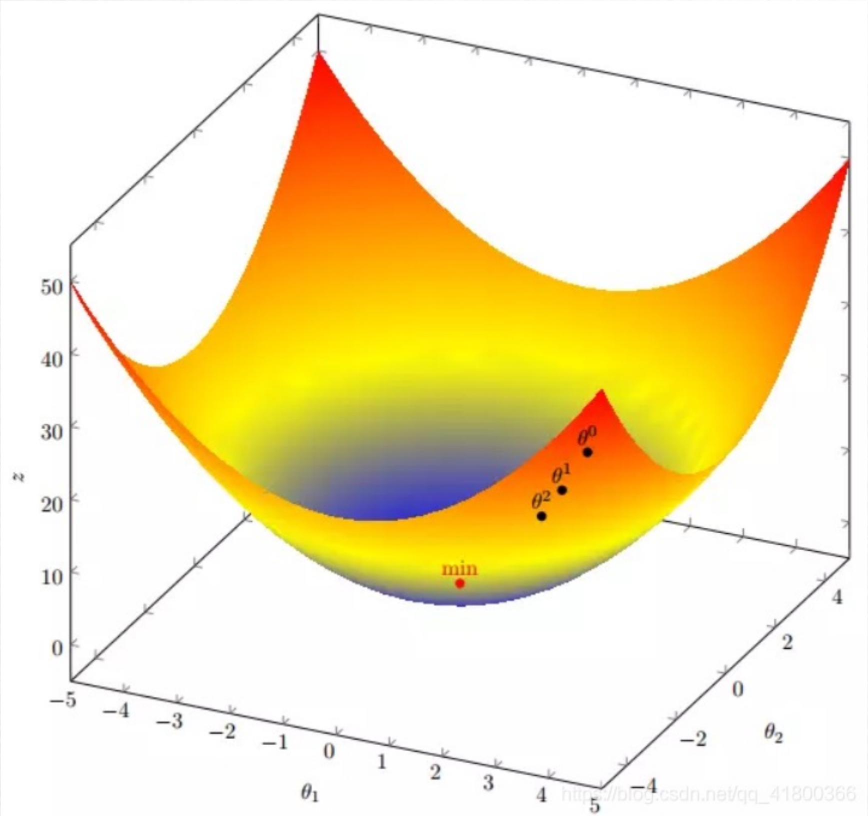
  - 对b求导：

$$\frac{\partial L(w,b)}{\partial b}=\frac{2}{N}\sum_{i=1}^{N}(wx_i+b-y_i)=\frac{2}{N}\sum_{i=1}^{N}1\times(wx_i+b-y_i)$$

$$\frac{\partial L(w,b)}{\partial b}=\frac{2}{N}\mathbf{1}^{\mathrm{T}}(w\boldsymbol{x}+b\mathbf{1}-\mathbf{y})$$

$\boldsymbol{x}$、$\mathbf{1}$、 $\boldsymbol{y}$为$N\times1$的向量（列向量）

梯度下降法



50
40
30
20
10
0
z
min
θ^0
θ^1
θ^2
−5 −4 −3 −2 −1 0 1 2 3 4 5
θ_1
−4 −2 0 2 4
θ_2
https://blog.csdn.net/qq_41800366

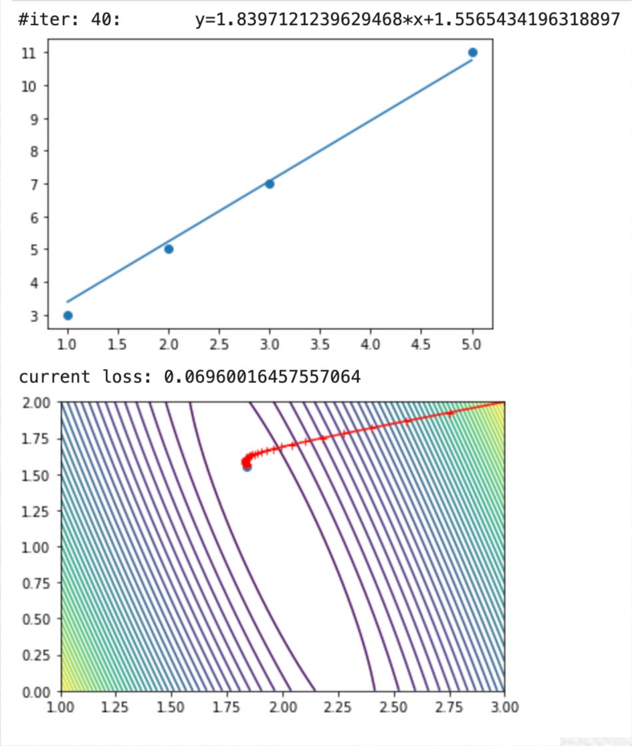
优化

- 一元线性回归：
- 1. 随机选择一组初始参数(w^0, b^0)
- 2. 对于可微函数，梯度方向时函数增长速度最快的方向，梯度的反方向是函数减少最快的方向。计算损失函数在当前参数点处的梯度。
- 3. 从当前参数点向梯度相反的方向移动，移动步长为：

$$w^{t+1} = w^t - lr\frac{\partial L(w,b)}{\partial w}(w^t)$$

$$b^{t+1} = b^t - lr\frac{\partial L(w,b)}{\partial b}(b^t)$$

- 4. 循环迭代步骤 3， 直到前后两次迭代得到的 (w^t, b^t) 差值足够小，即参数基本不再变化，说明此时损失函数已经达到局部最小值。
- 5. 输出(w^t, b^t)，即为使得损失函数最小时的参数取值 。



思考

- 多元线性回归：
  - 自变量x是向量，因变量y是数字
  - 自变量$\boldsymbol{x}$是向量，因变量$\boldsymbol{y}$也是向量

- 怎样用梯度下降法训练多元线性回归模型？

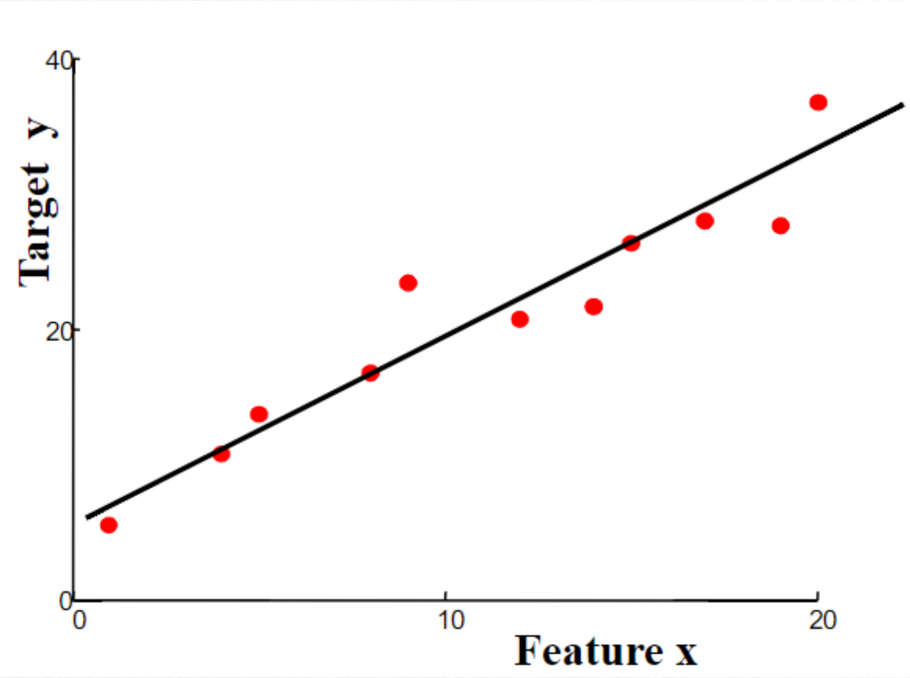
Python AI
Numpy与科学计算

提纲

- 一元线性回归
- 多元线性回归
- 逻辑回归

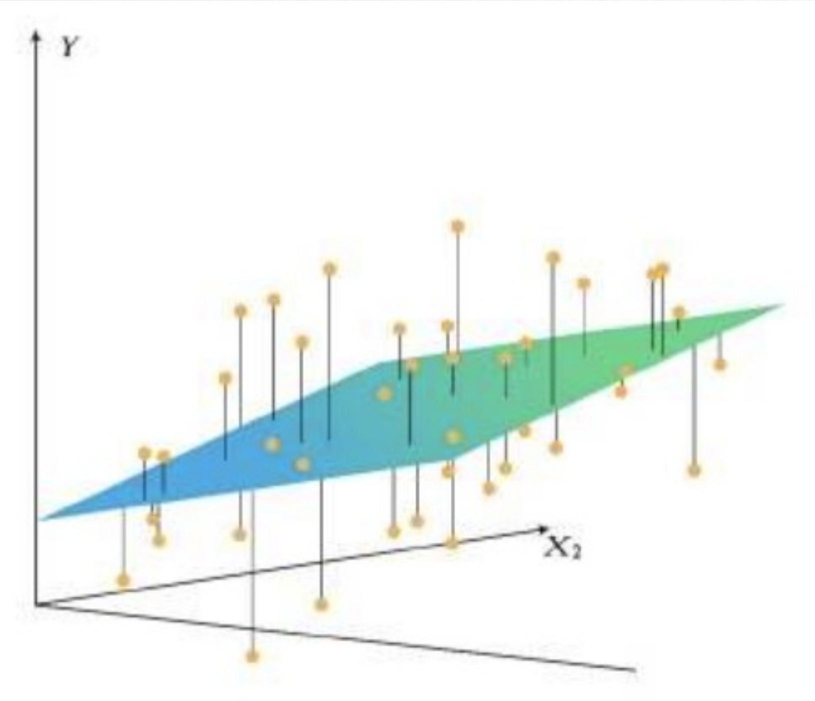
一元线性回归

- 只有一个自变量（特征x的维度为1，因此w的维度也是1）



多元线性回归

- 有多个自变量（特征x的维度大于1，对应的w的维度也大于1）



$$\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,j}, \ldots, x_{i,d})$$

$$\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_j \\ \cdots \\ w_d \end{pmatrix}$$

多元线性回归

- 假设目标值与特征之间线性相关，即因变量与自变量满足一个多元一次方程。
- 训练集 $D=\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$
- 将特征 $\boldsymbol{x}$ 和对应的目标 y 建模为多元一次方程：

$$\hat{y}=\boldsymbol{x}\cdot\boldsymbol{w}+b$$

 其中，$\boldsymbol{w}$和b为模型的参数。

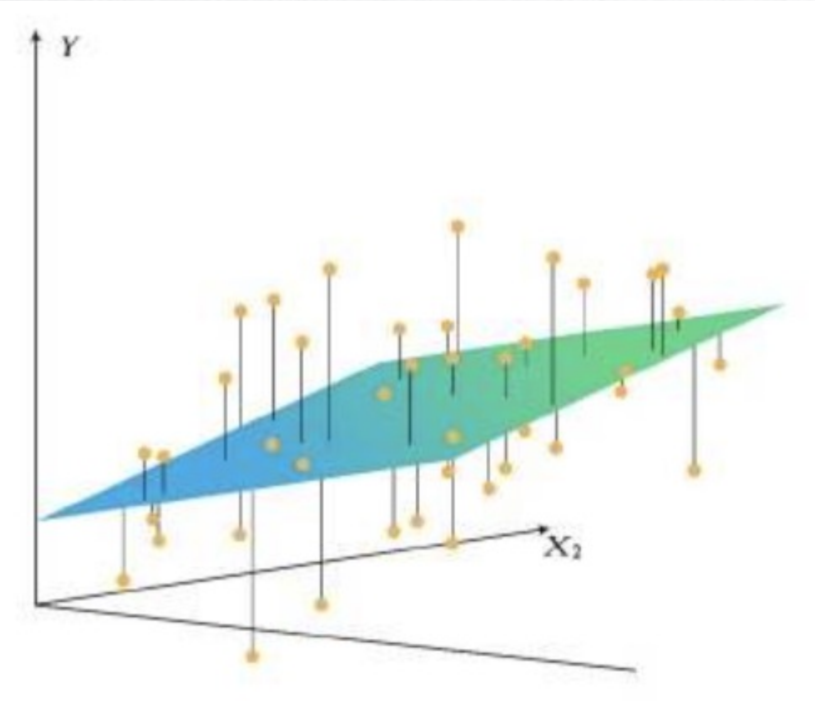
- 为了简化符号表示，在x后面加上一维值恒为1的特征，$\boldsymbol{x} \leftarrow [\boldsymbol{x}, 1]$, 同时$\boldsymbol{w}$增加一维，则可以把$b$融入$\boldsymbol{w}$

$$\hat{y}=\boldsymbol{x}\cdot\boldsymbol{w}$$

- 为了学习这两个参数，根据已知的训练样本点（自变量x和因变量y都是已知的）拟合这个多元一次方程，求解参数。

多元线性回归

- 有多个自变量（特征x的维度大于1，对应的w的维度也大于1）



$$\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,j}, \dots, x_{i,d}, 1)$$

$$\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_j \\ \cdots \\ w_d \\ w_{d+1} \end{pmatrix}$$

损失函数

- 将线性预测函数式代入MSE损失函数$L=\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i-y_i)^2$，将需要求解的参数$\boldsymbol{w}$看做是损失函数$L$的自变量：

$$L(\boldsymbol{w})=\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i\cdot\boldsymbol{w}-y_i)^2$$

- 通过最小化损失函数求解最优参数：

$$(\boldsymbol{w}^*)=\arg\min_{\boldsymbol{w}}\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_{\boldsymbol{i}}\cdot\boldsymbol{w}-y_i)^2$$

求梯度

$$(\boldsymbol{w}^*) = \arg\min_{\boldsymbol{w}} \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)^2$$

- 分别对参数**w**求导：

$$\frac{\partial L}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} x_{i,1}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)$$

$$\frac{\partial L}{\partial w_2} = \frac{2}{N}\sum_{i=1}^{N} x_{i,2}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)$$

....

$$\frac{\partial L}{\partial w_{d+1}} = \frac{2}{N}\sum_{i=1}^{N} x_{i,d+1}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)$$

求梯度

$$(\boldsymbol{w}^*) = \arg\min_{\boldsymbol{w}} \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)^2$$

- 分别对参数**w**求导：

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \cdots \\ \frac{\partial L}{\partial w_{d+1}} \end{pmatrix} = \begin{pmatrix} \frac{2}{N}\sum_{i=1}^{N} x_{i,1}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i) \\ \frac{2}{N}\sum_{i=1}^{N} x_{i,2}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i) \\ \cdots \\ \frac{2}{N}\sum_{i=1}^{N} x_{i,d+1}(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i) \end{pmatrix} = \frac{2}{N}\sum_{i=1}^{N} \boldsymbol{x}_i^T(\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)$$

梯度的矩阵形式

$$
\begin{array}{c}
x_1 \\ \\ \\ x_N
\end{array}
\begin{bmatrix}
x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(d)} & 1 \\
x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(d)} & 1 \\
\vdots & \vdots & & \vdots & \vdots \\
x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(d)} & 1
\end{bmatrix}
\qquad
\begin{bmatrix}
y_1 \\ y_2 \\ \vdots \\ y_N
\end{bmatrix}
$$

$$\boldsymbol{X} \qquad \mathbf{y}$$

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{2}{N}\sum_{i=1}^{N} \boldsymbol{x}_i^T (\boldsymbol{x}_i \cdot \boldsymbol{w} - y_i)$$ ，其中$\boldsymbol{x}_i$为行向量

- 矩阵形式：

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{2}{N}\mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{w} - \mathbf{y})$$

注意：此处**X**的维度是$N\times(d+1)$, **w**维度为$(d+1)\times 1$

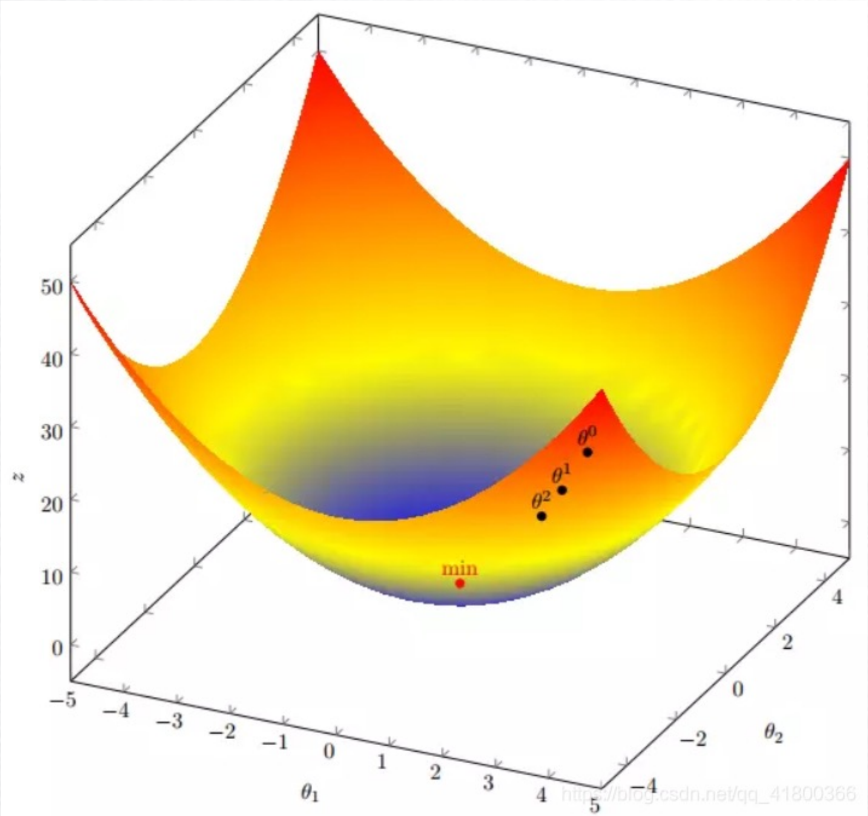
优化：梯度下降法

- 1. 随机选择一组初始参数($\boldsymbol{w}^0$)
- 2. 对于可微函数，梯度方向时函数增长速度最快的方向，梯度的反方向是函数减少最快的方向。计算损失函数在当前参数点处的梯度。
- 3. 从当前参数点向梯度相反的方向移动，移动步长为:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - lr\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}(\boldsymbol{w}^t)$$

- 4. 循环迭代步骤 3， 直到前后两次迭代得到的($\boldsymbol{w}^t$)差值足够小，即参数基本不再变化，说明此时损失函数已经达到局部最小值。
- 5. 输出($\boldsymbol{w}^t$)，即为使得损失函数最小时的参数取值 。

梯度下降法



50
40
30
20
10
0
z
θ^0
θ^1
θ^2
min
−5
−4
−3
−2
−1
0
1
2
3
4
5
θ_1
4
2
0
−2
−4
θ_2
https://blog.csdn.net/qq_41800366

示例：梯度下降法（类的定义和初始化）

```python
class MultiLinearRegression(object):
    def __init__(self, dim_in, learning_rate=0.01, max_iter=100, seed=None):
        """
        一元线性回归类的构造函数：
        参数 学习率：learning_rate
        参数 最大迭代次数：max_iter
        参数 seed：产生随机数的种子
        从正态分布中采样w的初始值
        """
        np.random.seed(seed)
        self.lr = learning_rate
        self.max_iter = max_iter
        self.w = np.random.normal(1, 0.1, [dim_in+1, 1]) # w 的维度为输入维度+1
        self.loss_arr = []
```

示例：梯度下降法

```python
def fit(self, x, y):
    """
    类的方法：训练函数
    参数 自变量：x
    参数 因变量：y
    返回每一次迭代后的损失函数
    """
    #首先在x矩阵后面增加一列1
    x = np.hstack([x, np.ones((x.shape[0], 1))])
    for i in range(self.max_iter):
        self.__train_step(x, y)
        y_pred = self.predict(x)
        self.loss_arr.append(self.loss(y, y_pred))
```

```python
def __train_step(self, x, y):
    '''
    类的方法：单步迭代，即一次迭代中对梯度进行更新
    '''
    d_w = self.__calc_gradient(x, y)
    self.w = self.w - self.lr * d_w
    return self.w
```

```python
def loss(self, y_true, y_pred):
    '''
    类的方法：计算损失
    参数 真实因变量：y_true
    参数 预测因变量：y_pred
    返回：MSE损失
    '''
    return np.mean((y_true - y_pred) ** 2)
```

```python
def __calc_gradient(self, X, y):
    '''
    类的方法：分别计算对w和b的梯度
    '''
    N = X.shape[0]
    #diff = (X.dot(self.w) - y)
    diff = np.matmul(X, self.w) - y
    #print(type(X.T))
    grad = np.matmul(X.T, diff)# X.T 转置 x.transpose()
    d_w = (2 * grad) / N
    return d_w
```

```python
def __f(self, X, w):
    '''
    类的方法：计算一元线性回归函数在x处的值
    '''
    return np.matmul(X,w)

def predict(self, X):
    '''
    类的方法：预测函数
    参数：自变量：x
    返回：对x的回归值
    '''
    y_prd = self.__f(X, self.w)
    return y_prd
```

示例：梯度下降法（数据准备）

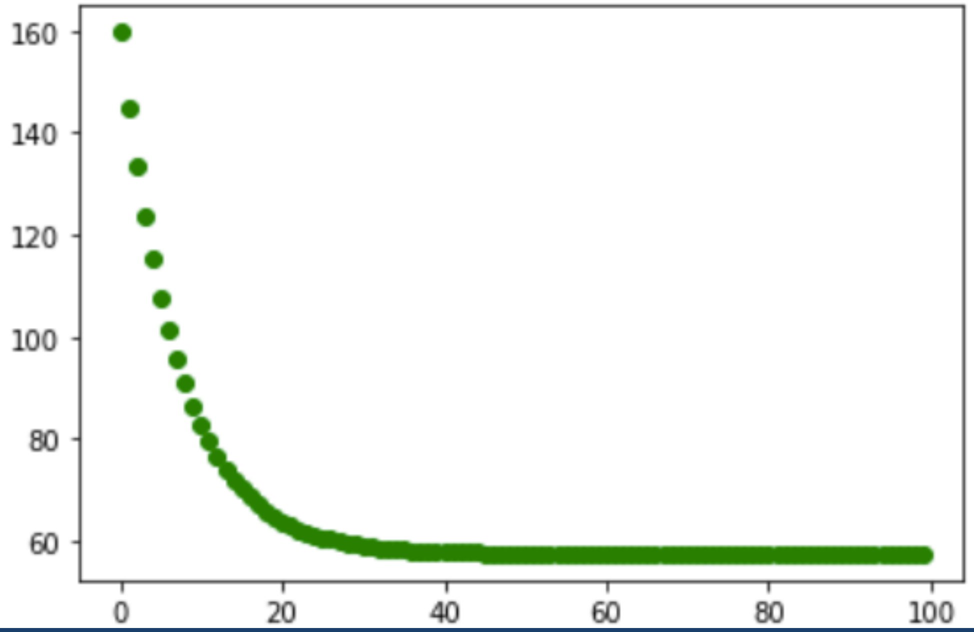
```
# data generation
np.random.seed(272)
data_size = 100
dim_in = 3
dim_out = 1
x = np.random.uniform(low=1.0, high=10.0, size=[data_size, dim_in])
map_true = np.array([[1.5], [-5.], [3.]])
y = x.dot(map_true) + np.random.normal(loc=0.0, scale=10.0, size=[data_size, dim_out])

# train / test split
shuffled_index = np.random.permutation(data_size)
x = x[shuffled_index, :]
y = y[shuffled_index, :]
split_index = int(data_size * 0.7)
x_train = x[:split_index, :]
y_train = y[:split_index, :]
x_test = x[split_index:, :]
y_test = y[split_index:, :]
```

示例：梯度下降法（训练模型）

```python
# train the liner regression model
regr = LinearRegressionMulti(dim_in, learning_rate=0.01, max_iter=100, seed=0)
regr.fit(x_train, y_train)
print(regr.w)

x_test_aug = np.hstack([x_test, np.ones((x_test.shape[0], 1))])
y_pred = regr.predict(x_test_aug)
res = y_pred - y_test



# plot the evolution of cost
plt.scatter(np.arange(len(regr.loss_arr)), regr.loss_arr, marker='o', c='green')
plt.show()
```

示例：梯度下降法

```
[[ 0.53989083]
 [-4.89837128]
 [ 3.30861944]
 [ 0.78811499]]
```



提纲

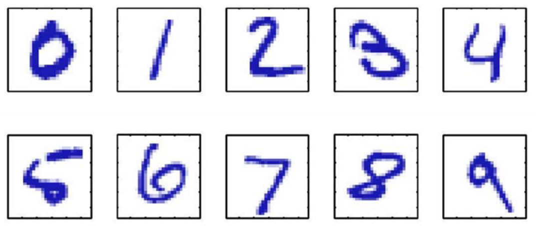
- 一元线性回归
- 多元线性回归
- **逻辑回归**

Python AI
Numpy与科学计算

回归与分类

- 都是监督学习中常见的任务类型

- 区别
  - 任务类型不同
  - 输出值不同
    - 回归模型一般给出**具体数值，输出值连续**
    - 分类模型一般给出**所属类别，输出值离散**
  - 使用的模型不同


Images are 28 x 28 pixels

输入：每一个图像表示为28*28维的向量$\mathbf{x} \in R^{784}$
学习一个分类器$f: \mathbf{x} \mapsto \{0,1,2,3,4,5,6,7,8,9\}$

标签	短信内容
0	商业秘密的秘密性那是维系其商业价值和垄断地位的前提条件之一
1	南口阿玛施新春第一批限量春装到店啦□ 春暖花开淑女裙、冰蓝色公主衫□气质粉小西装、冰丝女王长半裙
0	带给我们大常州一场壮观的视觉盛宴
0	有原因不明的泌尿系统结石等
0	23年从盐城拉回来的麻麻的嫁妆
1	感谢致电杭州萧山全金釜韩国烧烤店，本店位于金城路xxx号。韩式烧烤等，价格实惠、欢迎惠顾【全金釜韩国烧烤店】
1	(长期诚信在本市作各类资格职称（以及印 /章、牌、 ……等。祥：x x x x x x x x x x x x 李伟%

输入：每一个短信文本
学习一个分类器$f: \mathbf{x} \mapsto$ {正常短信，垃圾短信}

二分类问题与Logistic回归

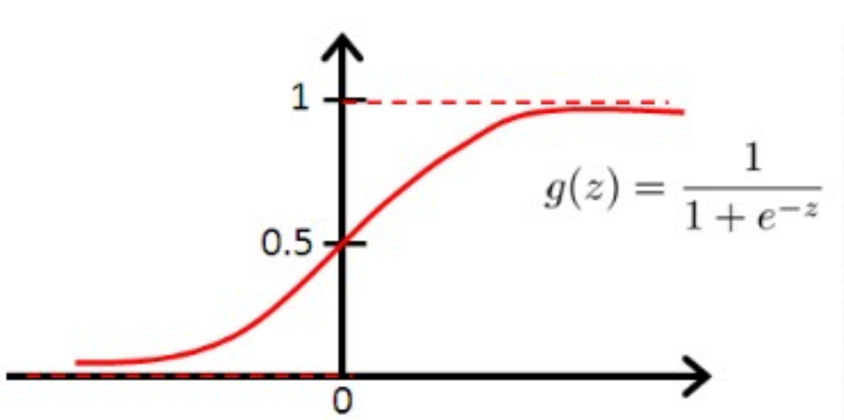
- 二分类：因变量y可能属于的两个类分别称为负类和正类，则因变量$y \in \{0,1\}$，其中0表示负类，1表示正类。
- 训练集 $D=\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)\}$

- Logistic回归：引入非线性变换sigmoid函数$g(z)$，把线性回归的输出值压缩到$(0,1)$之间：

$$\hat{y}=g(\boldsymbol{x}\boldsymbol{w})$$

其中，$\boldsymbol{w}$为模型的参数。

注意：已经在x后添加恒为1的特征，将参数b融入了$\mathbf{w}$

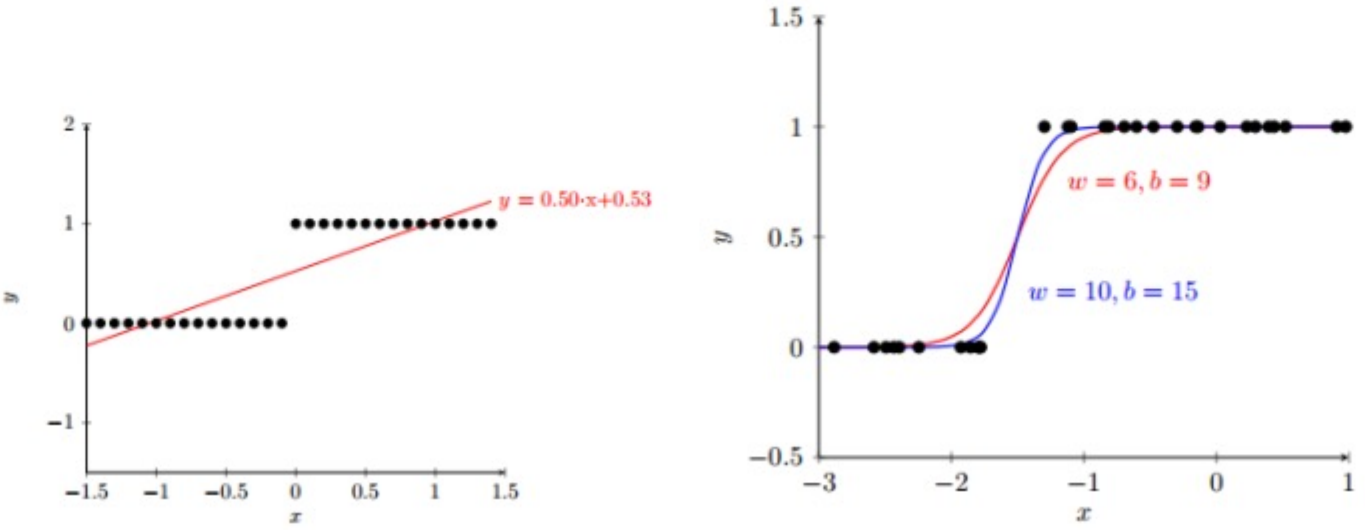


逻辑回归

- 为什么是“回归”？
  - 预测的是落入某一分类的概率

- 又叫“对数几率”回归
  - 设某事件发生概率为$p \in [0,1]$
    - 若采用线性回归，则无法保证输出在[0,1]范围内
  - 机率比（发生比、优势比）：$\frac{p}{1-p} \geq 0$
  - 取对数，令 $z = \ln\frac{p}{1-p} \in (-\infty, +\infty)$
  - 可以采用线性回归模型拟合：$z = f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{w}$

线性回归与逻辑回归



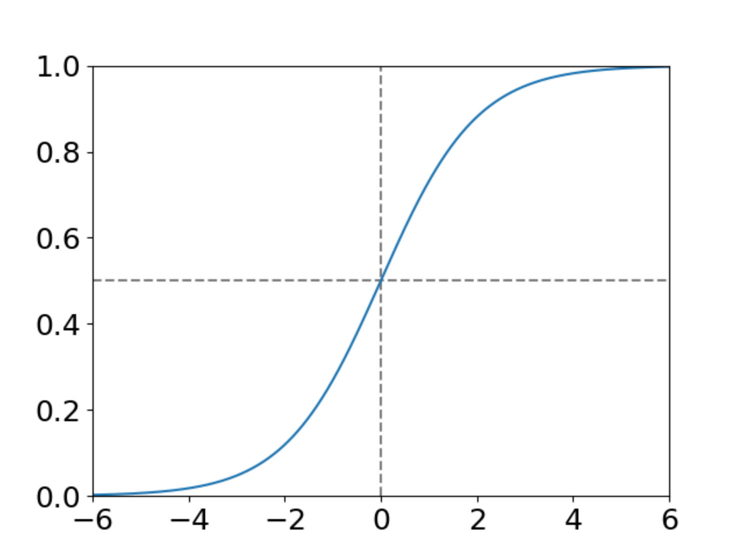
(a) 线性回归　　(b) Logistic 回归

逻辑回归

- 逻辑回归模型：

$$\boldsymbol{xw} = z = \ln\frac{p}{1-p} \Rightarrow p = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\boldsymbol{xw})}}$$

- Sigmoid函数$p = \frac{1}{1+e^{-z}} \in (0,1)$
  - 一般用于表示概率



逻辑回归损失函数的构造

- 什么样的参数 $\boldsymbol{w}$会使得观察到训练数据D=$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)\}$的概率最大?
  - 对于单个训练数据$(\boldsymbol{x}_i, y_i)$
    - 如果为正例： $y_i = 1$， 使得p_i最大
    - 如果为负例： $y_i = 0$， 使得$1 - p_i$最大

⇩

- 总体而言：使得$p_i^{y_i} \times (1 - p_i)^{1-y_i}$最大
  - 取对数：使得$y_i \ln p_i + (1 - y_i) \ln(1 - p_i)$ 最大
- 对于N个训练数据D=$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)\}$，使得它们的联合概率越大（负联合概率最小）

$$L(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

逻辑回归

- 损失函数

$$L(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \ln p_i + (1-y_i)\ln(1-p_i)]$$

  - 一般称为交叉熵损失函数
  - 应用梯度下降法：梯度（j 是维度索引）

$$\frac{\partial L(\boldsymbol{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_i^{(j)}(p_i - y_i)$$

其中$p_i = \frac{1}{1+e^{-z_i}} = \frac{1}{1+e^{-(\boldsymbol{x}_i\cdot\boldsymbol{w})}}$

逻辑回归梯度的具体推导

- Sigmoid函数导数：$p=g(z)=\frac{1}{e^{-z}+1}\Rightarrow\frac{dp}{dz}=\frac{dg}{dz}=g(z)\big(1-g(z)\big)=p(1-p)$
  - $\frac{dp_i}{dw^j}=\frac{dg(\boldsymbol{x}_i\boldsymbol{w})}{dw^j}=g(\boldsymbol{x}_i\boldsymbol{w})\big(1-g(\boldsymbol{x}_i\boldsymbol{w})\big)\frac{d(\boldsymbol{x}_i\boldsymbol{w})}{dw^j}=p_i(1-p_i)x_i^j$
- $\frac{\partial L(\boldsymbol{w})}{\partial w_j}=-\frac{\partial\frac{1}{N}\sum_{i=1}^{N}[y_i\ln p_i+(1-y_i)\ln(1-p_i)]}{\partial w_j}$
- $=-\frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i}{p_i}\frac{dp_i}{dw^j}-\frac{1-y_i}{1-p_i}\frac{dp_i}{dw^j}\right]$
- $=-\frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i}{p_i}-\frac{1-y_i}{1-p_i}\right]\frac{dp_i}{dw^j}$
- $=-\frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i}{p_i}-\frac{1-y_i}{1-p_i}\right]p_i(1-p_i)x_i^j$
- $=-\frac{1}{N}\sum_{i=1}^{N}[y_i(1-p_i)-(1-y_i)p_i]x_i^j$
- $=\frac{1}{N}\sum_{i=1}^{N}(p_i-y_i)x_i^j$

梯度的矩阵形式

x_1	$x_1^{(1)}$	$x_1^{(2)}$	$\cdots$	$x_1^{(d)}$	1
	$x_2^{(1)}$	$x_2^{(2)}$	$\cdots$	$x_2^{(d)}$	1
	$\vdots$	$\vdots$		$\vdots$	$\vdots$
x_N	$x_N^{(1)}$	$x_N^{(2)}$	$\cdots$	$x_N^{(d)}$	1

$\mathbf{X}$

y_1
y_2
$\vdots$
y_N

$\mathbf{y}$

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \left(\frac{1}{N}\sum_{i=1}^{N} x_i^{(1)}(p_i - y_i), \frac{1}{N}\sum_{i=1}^{N} x_i^{(2)}(p_i - y_i), \cdots, \frac{1}{N}\sum_{i=1}^{N} x_i^{(d+1)}(p_i - y_i)\right)^T$$

- 矩阵形式：

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{N}\mathbf{X}^{\mathrm{T}}(\boldsymbol{p} - \mathbf{y})$$

注意：此处**X**的维度是$N\times(d+1)$, $\boldsymbol{p} - \boldsymbol{y} = \begin{bmatrix} p_1 - y_1 \\ \vdots \\ p_N - y_N \end{bmatrix}$维度为$N\times 1$

逻辑回归

- 梯度下降
  1. 随机初始化参数 $\boldsymbol{w}$
  2. 计算
  $$p = \frac{1}{1 + e^{-(\boldsymbol{X}\boldsymbol{w})}}$$
  3. 更新参数 $\mathbf{w}$
  $$w \leftarrow w - \alpha \frac{1}{N} \mathbf{X}^{\mathrm{T}} (\boldsymbol{p} - \mathbf{y})$$
  4. 重复步骤2、3直至收敛

提纲

Python AI
Numpy与科学计算

- 一元线性回归
- 使用Matplotlib库进行图表绘制
- 多元线性回归
- 逻辑回归
- 作业

作业：用numpy实现逻辑回归

- 用numpy实现Logistic Regression类，用梯度下降法训练
  - 阅读多元线性回归示例代码，理解每个部分的含义
  - 在此基础上进行修改实现逻辑回归代码

  - 注意将计算向量化以提高计算速度，充分利用numpy在矩阵计算上的优势

作业：用numpy实现逻辑回归

- 用numpy实现Logistic Regression类，用梯度下降法训练
  - 实现类的方法 loss = fit(X,Y) 函数，用于训练
    - 输入：X为N * d维的训练数据，N为训练样本数，d为数据的维数，即X是N个d维数据点排成的矩阵；Y为N*1维的训练数据真实类别号，即N个类别号排成的向量；
    - 输出：loss为列表，长度为训练轮数，每个元素为对应轮的损失函数值。

$$\begin{array}{c} x_1 \\ \\ \\ x_N \end{array} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(d)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(d)} \\ \vdots & \vdots & & \vdots \\ x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(d)} \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$\boldsymbol{X} \qquad\qquad\qquad \boldsymbol{y}$$

作业：用numpy实现逻辑回归

- 用numpy实现Logistic Regression类，用梯度下降法训练
  - 实现类的方法 loss = fit(X,Y) 函数，用于训练
  - 实现步骤：
    1. 扩充 x_i ，增加常数1在末尾，并将所有 x_i 组合成 $\boldsymbol{X}$，将所有 y_i 组合成 $\boldsymbol{y}$
    2. 计算损失函数值，加入到loss列表结尾
    3. 计算损失函数对参数 W 的梯度
    4. 更新参数 W
    5. 重复上述过程直至收敛

作业：用numpy实现逻辑回归

- 用numpy实现Logistic Regression类，用梯度下降法训练
  - 实现类的方法 y_pred, y_pred_label = predict(X) 函数，用于测试
    - 输入： X为N * d维的测试数据， N为测试样本数;
    - 输出：y_pred维数为N*1，为模型的预测（回归）值，即属于类别1的概率；y_pred_label维数为N*1，为根据回归值得到的预测类别号，只能取0或1
    - y_pred_label的获得方法：若相应的y_pred小于0.5，则y_pred_label为0，否则为1

作业：用numpy实现逻辑回归

- 提交:
  - 把Jupyter Notebook中的空缺部分填好，并运行得到结果
  - 提交文件

- 提交期限：2周

- 提交平台：未来课堂

谢谢！