有监督学习-线性模型

Supervised Learning - Linear Model

目录

线性模型

线性回归

- 目标：找到 $f(x) = wx_i + b$ 使得 $f(x_i) \simeq y_i$

- 对于离散属性：

  - 若有顺序，则转换为连续数字

  - 若无序，则转换为k维数据

    - 例如：属性为颜色的种类，分别为红色、黄色、绿色

    则将颜色属性转换为红色、黄色、绿色三列，值分别为0或1

线性回归

- 目标：找到 $f(x) = wx_i + b$ 使得 $f(x_i) \simeq y_i$

- 解决思路：令均方根误差最小

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

- 所以我们需要找到可以使得J(w,b)取得最小值的w, b

- 对J(w, b)中的w与b分别求偏导数

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

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

线性回归

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

$$\frac{\partial E(w,b)}{\partial w}=\sum_{i=1}^{m}\cdot 2\cdot(y_i-wx_i-b)\cdot(-x_i)=2\sum_{i=1}^{m}(x_i^2w-y_ix_i+bx_i)$$

$$=2\left(w\cdot\sum_{i=1}^{m}x_i^2-\sum_{i=1}^{m}(y_i-b)x_i\right)$$

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

线性回归

- 令偏导数为0则可以求的

$$w = \frac{\sum_{i=1}^{m} y_{i(x_i - \bar{x})}}{\sum_{i=1}^{m} x^2} - 1/m * (\sum_{i=1}^{m} x_i)^2$$

$$b = 1/m * \sum_{i=1}^{m} (y_i - wx_i)$$

- 上述的方法又称为最小二乘法
  - 一种数学优化的技术，通过最小化误差的平方来寻找最优解
- J又称为损失函数或cost function或loss function

多元线性回归

- 如果有多个x呢?
- 多元回归问题
- 数据集可以表示为，X是一个m行n列的矩阵

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} & 1 \\ x_{21} & x_{22} & \dots & x_{21} & 1 \\ \dots & \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} & 1 \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \dots & \dots \\ x_m & 1 \end{pmatrix}$$

多元线性回归

- 目标：找到 $f(x) = Xw + b$ 使得 $f(X) \simeq y$

  $x_i = x_{i1}, x_{i2}, \ldots, x_{in}$

- 令 $w' = (w; b)$ ，则数据可以表示为， w' 是一个n+1维的列向量

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} & 1 \\ x_{21} & x_{22} & \cdots & x_{21} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} & 1 \end{pmatrix}, \quad y = (y_1, y_2, \ldots, y_m)$$

- 同样使用最小二乘法，求值得使 $J(w')$ 最小的 w'

  $J(w') = (y - Xw')^T (y - Xw')$

- 对 w' 求导，有 $\frac{\partial J(w')}{\partial w'} = 2X^T(Xw' - y))$

多元线性回归

- 令导数为零就可以得到参数 w'

  - X^TX 正定或者满秩的时候 w' 有唯一解

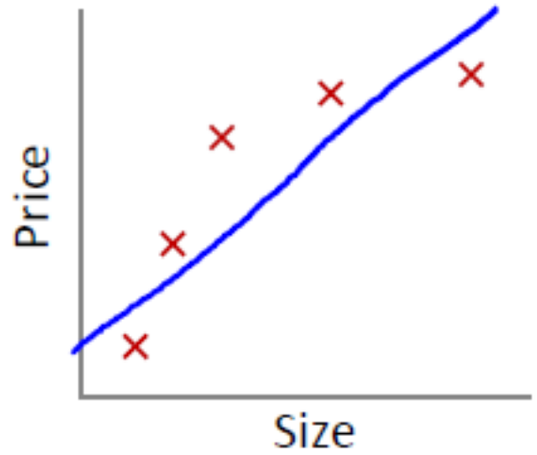
    $w' = (X^TX)^{-1}X^Ty$
  - 若 X^TX 不满秩的时候 w' 会有多个解

- 此时，需要引入正则化(regularization)来帮助我们

正则化

正规化

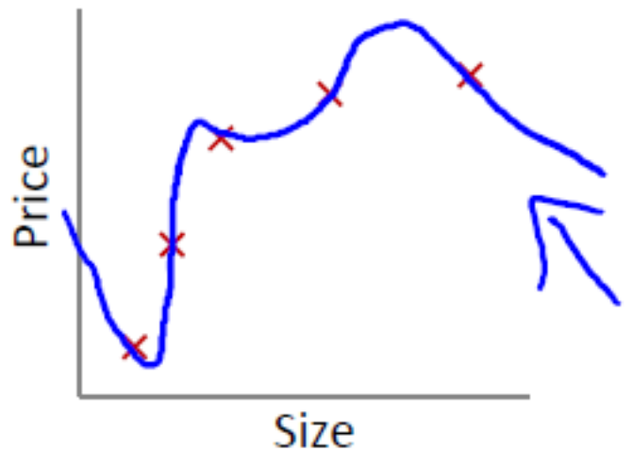
- 从拟合说起，以房价预测来举例



$\rightarrow \theta_0 + \theta_1 x$

"Underfit" "High bias"

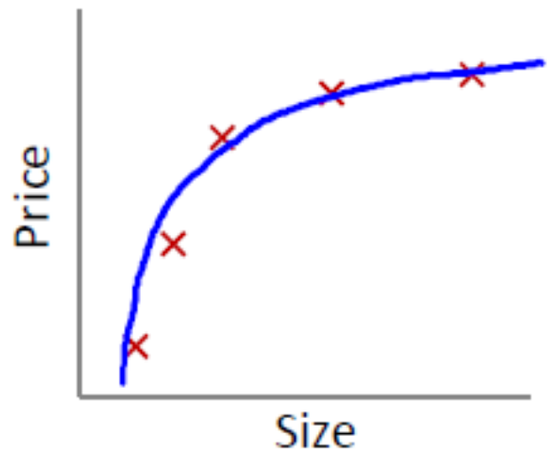
欠拟合



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"Overfit" "High variance"

过拟合



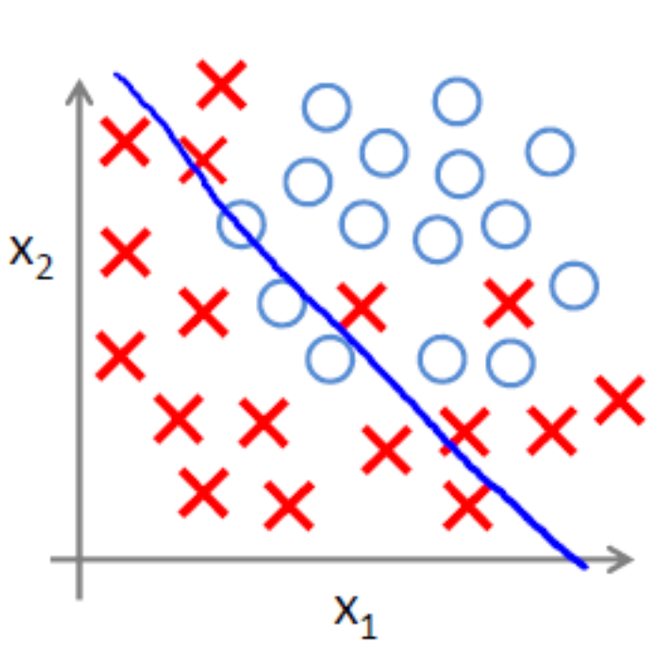
$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

合适的拟合

正规化

- 例子2

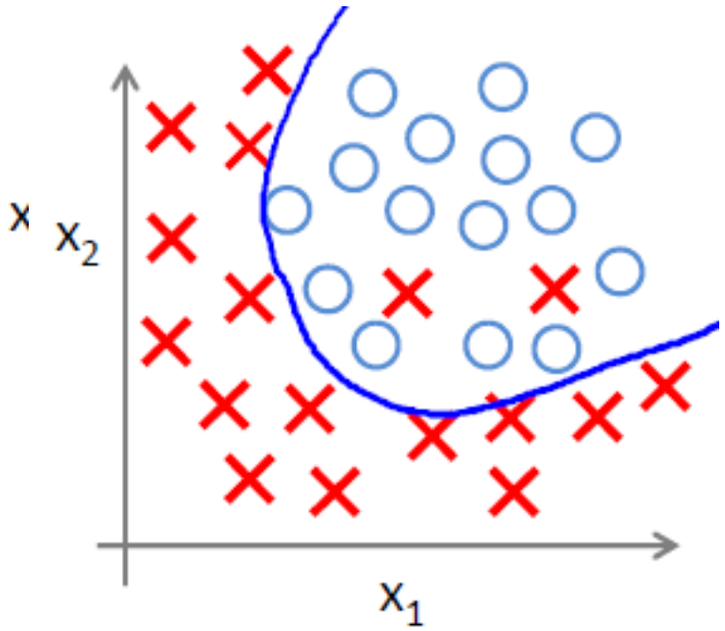


$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

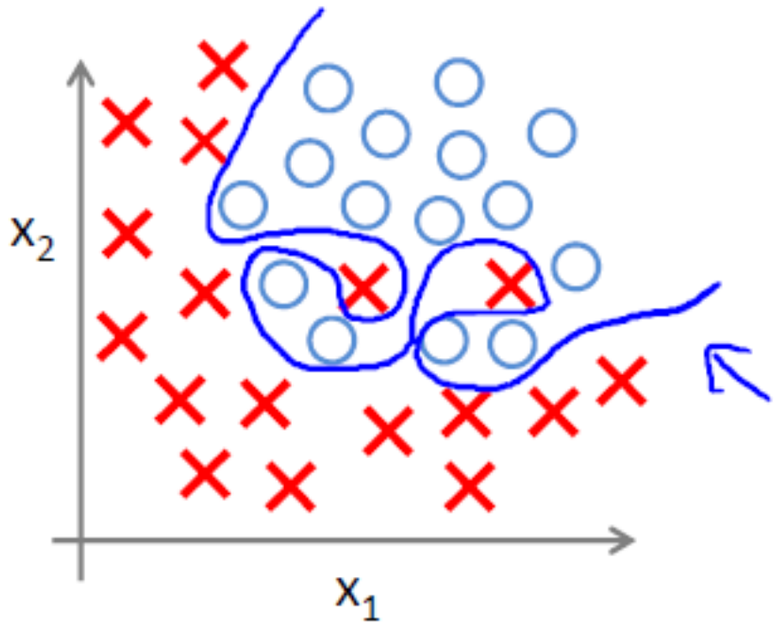
"Underfit"

欠拟合



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

合适的拟合



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

"Overfit"

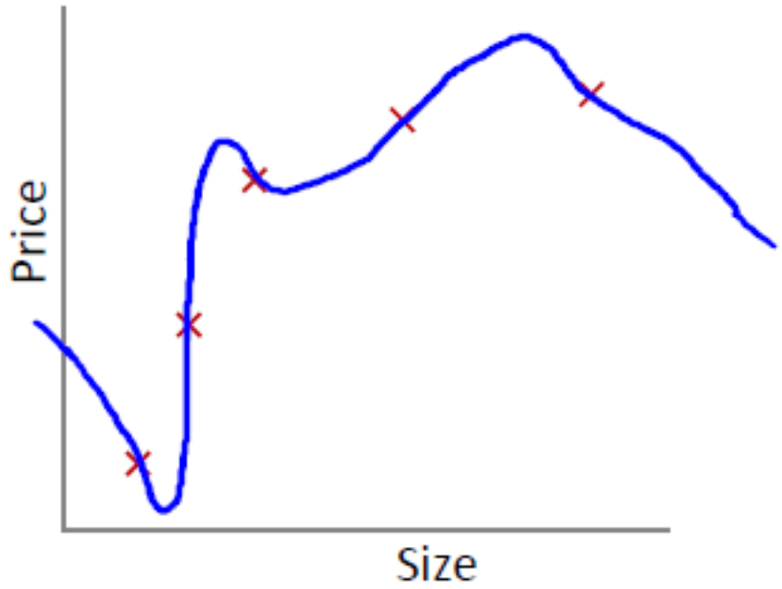
过拟合

如何解决过拟合

- 过拟合往往源于特征数目太多

- 对于房价问题，如果有100维属性，那么模型会对训练集拟合的很好

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

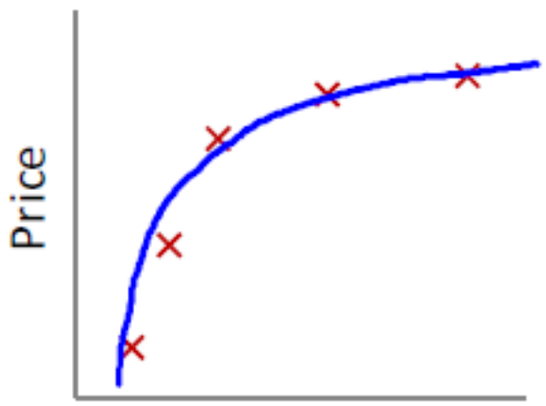


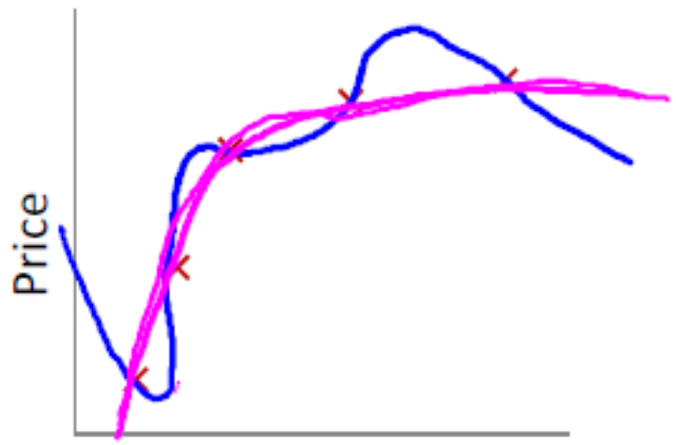
如何解决过拟合

- 解决过拟合的问题
    - 方法1: 特征抽取，降低特征数
        - 人工挑选
        - 使用降维算法或者其他特征提取算法（PCA，RandomForest等）
    - 正则化
        - 保留所有特征，但是降低参数的量或者值
        - 正则化的好处：当特征很多的时候，每一个特征都会对y做一份自己的贡献

损失函数

- 依然从房价预测开始考虑





- 直观看来，我们只需要消除 x^3 x^4 就可以解决我们的问题

- 所以，只需要让 θ_3 , θ_4 约等于0就可以了

损失函数

- 那么我们可以在损失函数中对 θ_3，θ_4 添加一个很大的惩罚系数就可以了

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

- 这样在我们最小化损失函数的时候，就可以对3与4进行惩罚，使他们变得很小

- 对所有参数都取小一点的优点是可以防止过拟合

损失函数

- 经过正则化的损失函数，我们可以定义为

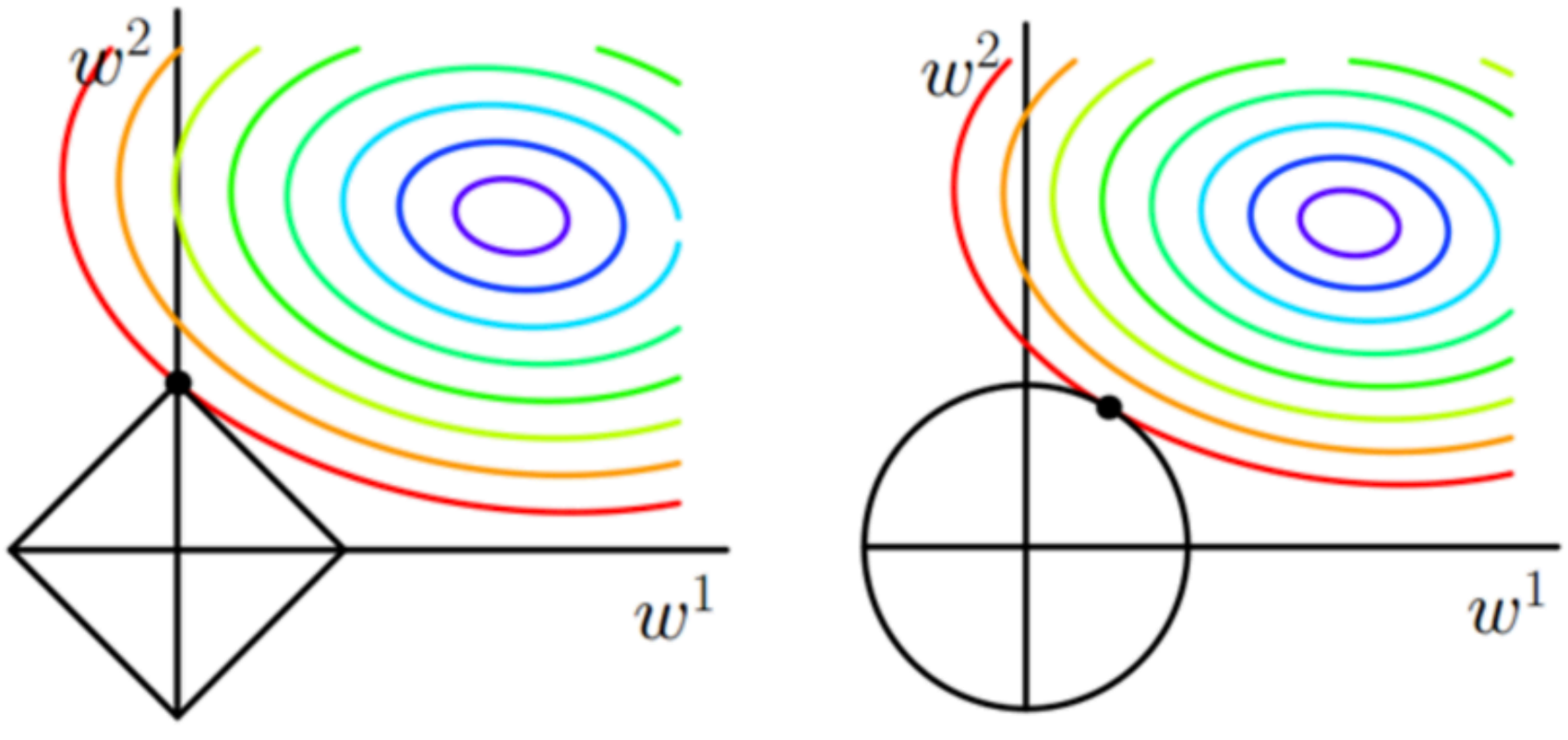
$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta - y^i)^2 + \lambda \sum_{j=1}^{n} g(\theta)]$$

- $g(\theta)$ 为正则化函数

- λ 为正则化参数，我们的目标依然是最小化 $J(\theta)$

L1与L2正则化

- L1正则化：　$g(\theta) = ||\theta||_1 = |\theta_0| + |\theta_1| + \ldots + |\theta_n|$
  - 更倾向于获得更加稀疏的参数
- L2正则化：　$g(\theta) = ||\theta||_2^2 = \theta_1^2 + \theta_2^2 + \ldots + \theta_n^2$
  - 更倾向于选择更多的参数



广义线性模型

广义线性模型

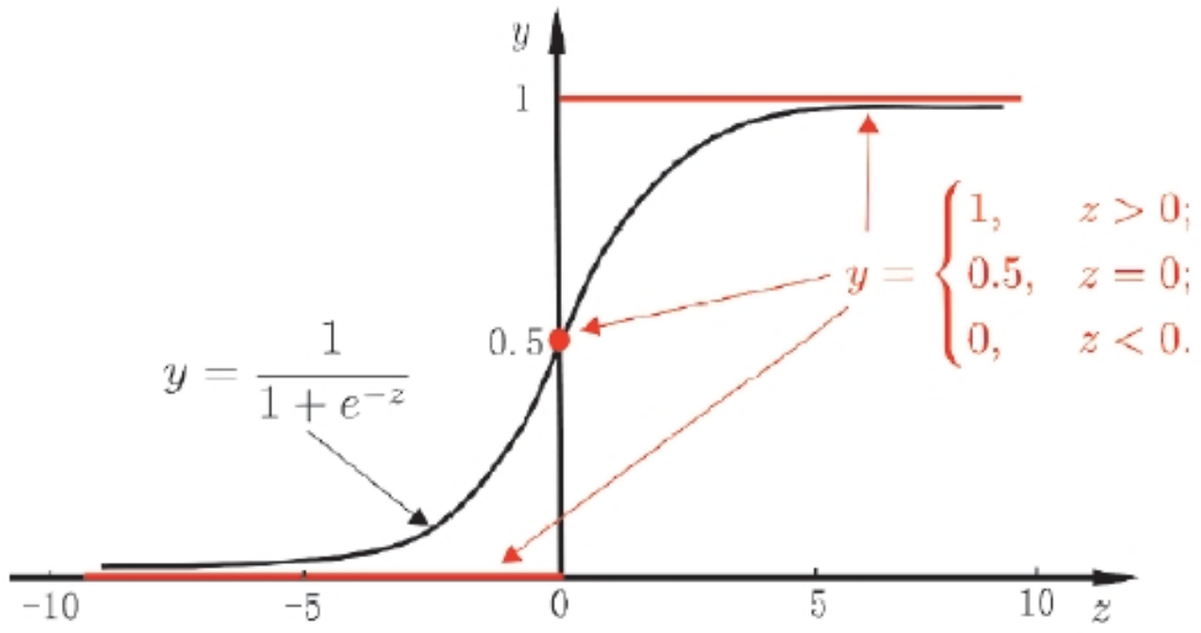
- 一般的函数表达：　$y = g(w^T x + b)$

- $g(x)$ 为一个单调可微的函数，通常叫做连接函数或者联系函数

二分类任务

- 线性回归的输出为 $z = w^T x + b$ ，期望将目标y分为0或者1
- 所以我们要找到z与y的联系函数
- 在广义线性模型中，常用sigmod函数作为联系函数，用来解决二分类问题

$$s(x) = \frac{1}{1 + e^{-x}}$$



二分类任务

- 这种模型我们常称为逻辑回归模型(logistic regression)

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

- 这样我们可以将回归任务转换为分类任务

- 逻辑回归优点：
  - 可以得到类别的近似概率预测
  - 可以直接使用现有的优化方法求解

二分类任务

- 令

$$\theta = (w, b),$$

$$p(y = 1 \,|\, x; \theta) = h(x; \theta), \quad p(y = 0 \,|\, x; \theta) = 1 - h(x; \theta)$$

则有 $p(y \,|\, x; \theta) = (h(x; \theta))^{y}(1 - h(x; \theta))^{(1-y)}$

- 似然函数

$$L(\theta) = \prod_{i=1}^{m} p(y_i \,|\, x_i; \theta) = \prod_{i=1}^{m} (h(x_i; \theta))_{i}^{y}(1 - h(x_i; \theta))^{(1-y_i)}$$

- 对数似然函数（最大化该函数）

$$\imath(\theta) = \sum_{i=1}^{m} y_i ln h(x_i; \theta) + (1 - y_i) ln(1 - h(x_i; \theta)))$$

二分类任务

$$L(\theta) = \prod_{i=1}^{m} (h(x_i;\theta))^{y_i} (1-h(x_i;\theta))^{(1-y_i)}$$

$$= \sum_{i=1}^{m} y_i \ln (h(x_i;\theta)) + \sum_{i=1}^{m} (1-y_i) \cdot \ln (1-h(x_i;\theta))$$

$$= \sum_{i=1}^{m} y_i \ln h(x_i;\theta) + (1-y_i) \cdot \ln (1-h(x_i;\theta))$$

高阶连续函数可以使用梯度下降算法求最优解

优化算法

梯度下降（Gradient descent）

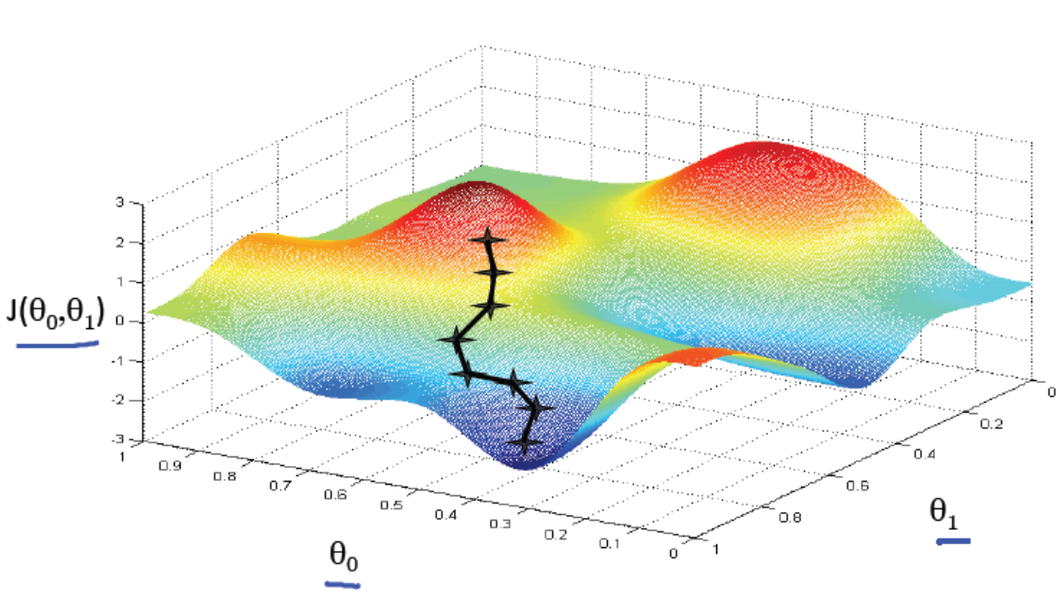
- 应用场景：求最小值问题
- 例如，给定函数 $J(\theta_1, \theta_2)$ ，求其最小值
- 梯度：在该方向是函数下降最快的方向

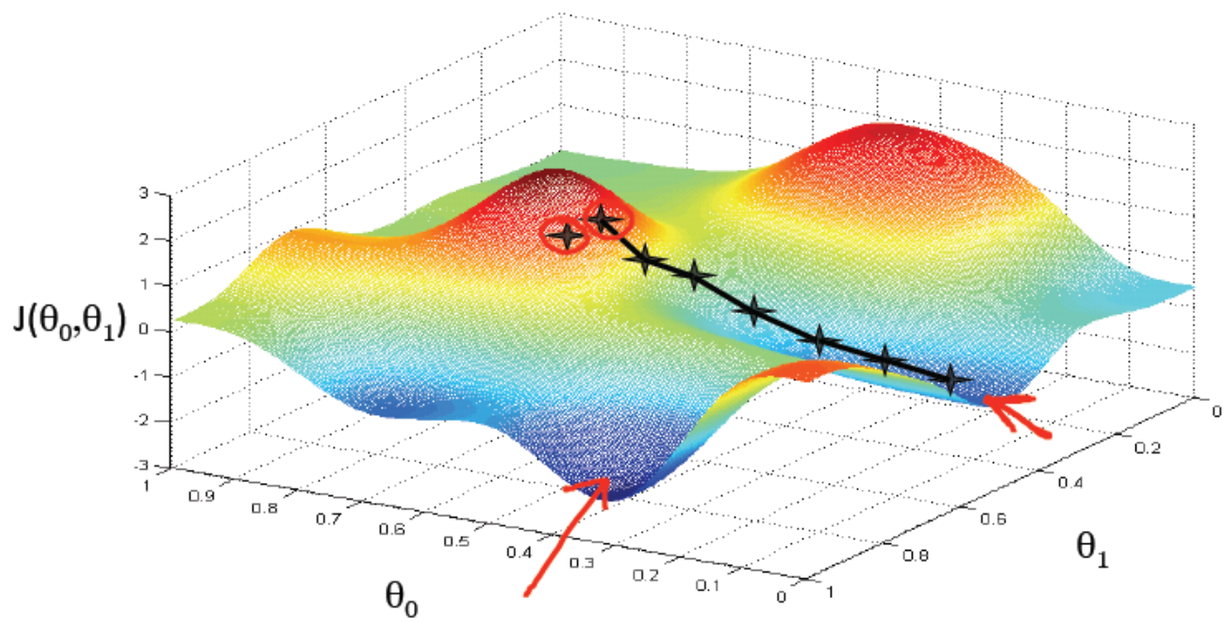
梯度下降（Gradient descent）

- 方法框架：
  - 初始化 θ_1 与 θ_2
  - 每次改变这两个参数，使J递减，一直到减小到我们满意的数值
- 初始值不同，得到的最小值会不同

梯度下降（Gradient descent）

- 初始值不同，得到的最小值会不同





梯度下降（Gradient descent）

- 重复下面过程知道收敛，注意同时更新两个参数

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(simultaneously update $j = 0$ and $j = 1$)

}

learning rate

derivative

- α 为学习率，控制了学习太快。
  - 太小，会收敛过慢
  - 太大，会错过局部最优

优化算法

- 最小二乘法
  - 优点:
    - 高效。比梯度下降这样的迭代算法简单
  - 缺点:
    - 特征数大的时候计算量非常大
    - 如果模型不是线性关系的话，无法使用最小二乘法，而梯度下降仍然可以使用

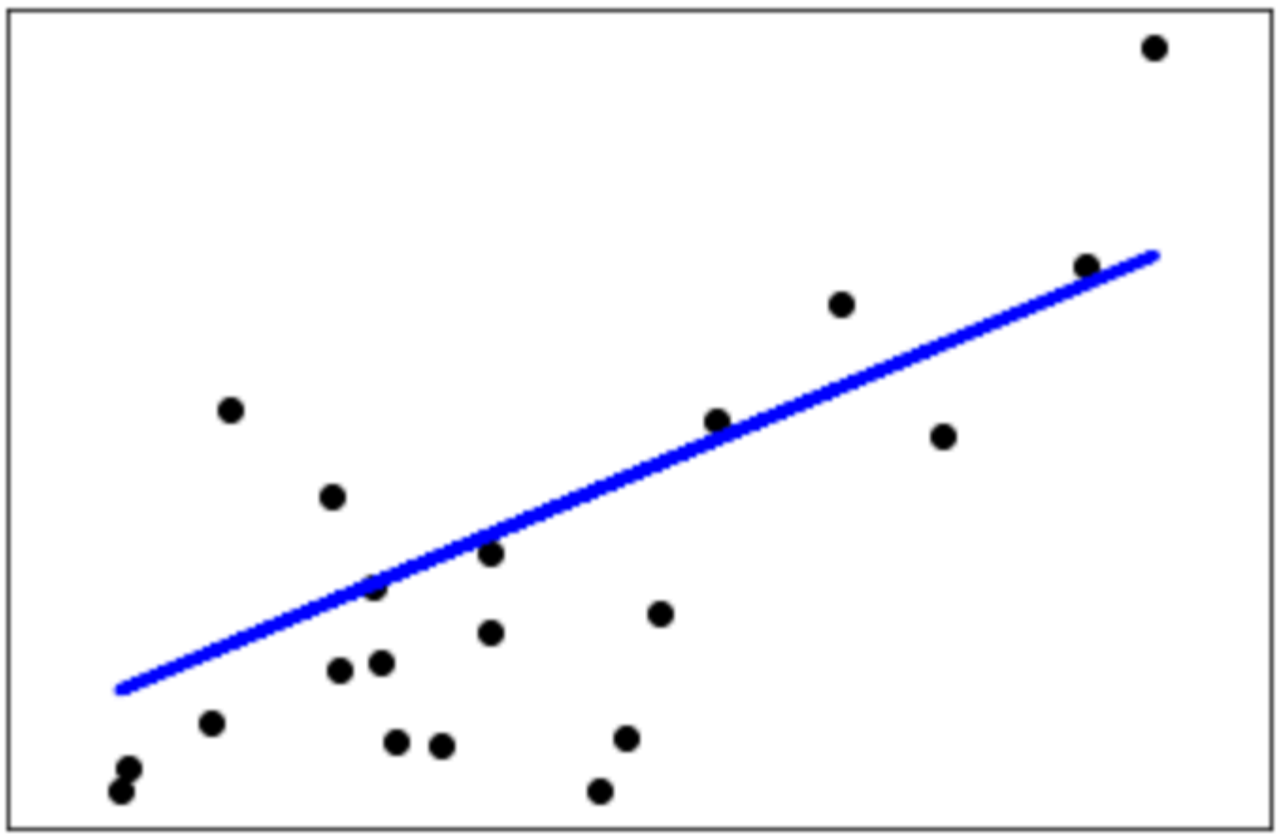
优化算法

- 梯度下降
  - 优点:
    - 如果是凸函数，理论上是全剧最优，如果不是凸函数则是局部最优
- 缺点:
  - 每次迭代都需要过所有的训练数据

实践

实践

- 使用sklearn中自带的diabetes数据集进行回归实验



实践

1. 引入包

sklearn自带一部分公开数据集，包含在datasets模块中 引入均方根误差与r2_score模块

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error
```

2. 加载数据

28]:
```
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
```

31]:
```
# Use only one feature
# 选择第二列
diabetes_X = diabetes.data[:, 2]
# diabets_X.shape (442,)
diabetes_X = diabetes_X[:, np.newaxis]
# np.newaxis 的作用是在当前添加一个维度
# diabetes_X.shape (442, 1)

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

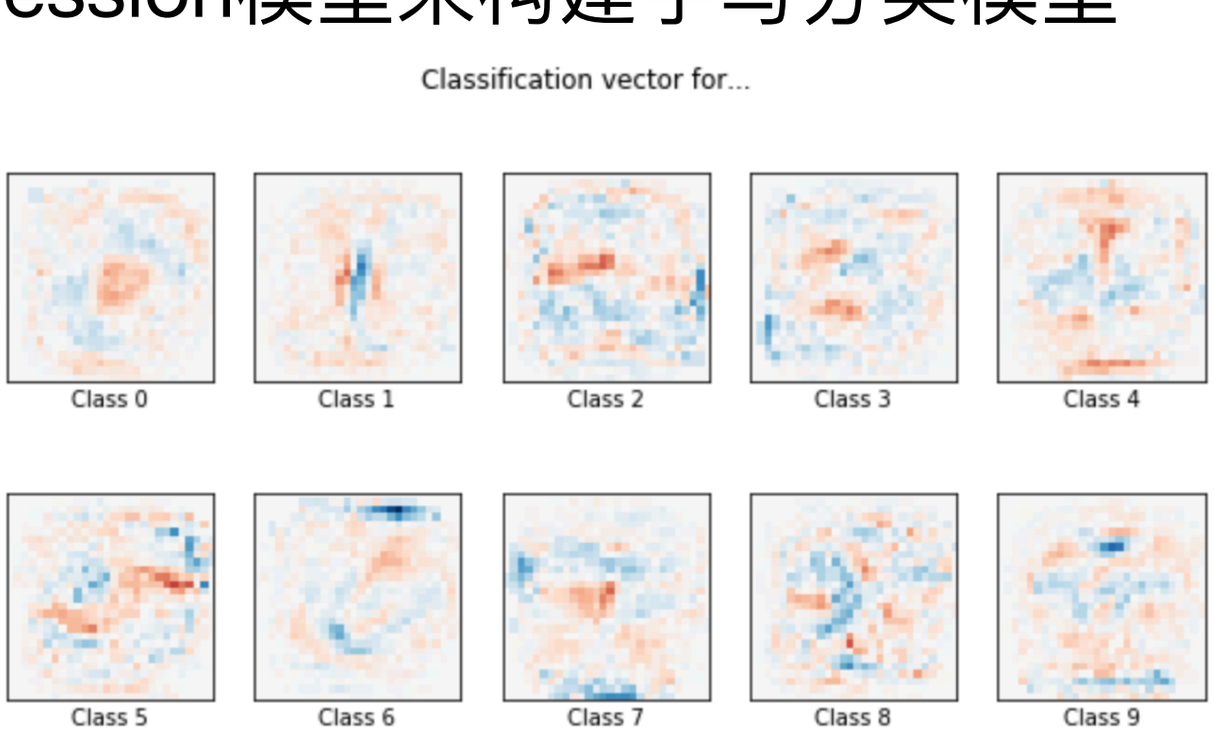
实践

[14]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# 获得系数(w)
# 系数保存在regr对象的coef_中
print('Coefficients: \n', regr.coef_)
# 获得截距(b)
print('Intercept: \n', regr.intercept_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
# do not show axis
plt.xticks(())
plt.yticks(())

plt.show()
```

```
Coefficients:
 [938.23786125]
Intercept:
 152.91886182616167
Mean squared error: 2548.07
```

实践

- 使用逻辑回归对手写字体进行分类
- 要求：
  - 需要用matplotlib对训练数据进行可视化
  - 使用sklearn的LogisticRegression模型来构建手写分类模型
    - 使用L2正则化
    - 使用梯度下降优化算法
- 对模型参数可视化



实践

```
LogisticRegression(
    penalty='l2', # 正则化系数，可选值'l1'与'l2'，默认为l2
    tol=0.0001,   # 终止迭代的误差范围
    C=1.0,        # 正则化系数λ的倒数，必须为正数，默认为1。和SVM中的C一样，值越小，代表正则化越强。
    fit_intercept=True, # 是否存在截距（b）
    class_weight=None,  # 是否对不同的类别划分不同的权重，数据不平衡时使用
    solver='warn', # 优化方法
    max_iter=100, # 最大迭代次数
    multi_class='warn', # 'ovr'(one-vs-rest)，不管是多少类的回归，都当作二分类来处理,默认为'ovr'
                        #  'multinomial'连接函数为softmax，直接输出多分类
)

solver:
1. liblinear：使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。可以使用L1与L2正则化
2. lbfgs：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。只可以使用L2正则化
3. newton-cg：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。只可以使用L2正则化
4. sag：即随机平均梯度下降，是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度。只可以使用L2正则化
```

source code:
https://github.com/scikit-learn/scikit-learn/blob/7b136e9/sklearn/linear_model/logistic.py#L998

实践

[2]:
```
%matplotlib inline
```

使用罗辑回归对手写字体进行识别，训练数据使用MNIST数据集。
MNIST 数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST).
训练集 (training set) 由来自 250 个不同人手写的数字构成, 其中 50% 是高中学生, 50% 来自人口普查局 (the Census Bureau) 的工作人员. 测试集(test set) 也是同样比例的手写数字数据.
每张图片的尺寸是28*28的单通道灰度图
每张图片的标号保存在第一列中

1. 导入包

[15]:
```
import time
import matplotlib.pyplot as plt
import numpy as np

#from sklearn.datasets import fetch_openml
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

实践

2. 加载数据

```
raw_data = np.loadtxt('./mnist_train.csv', delimiter=',')
print('Data load Done.')

# 类的标号保存在第一列
X = raw_data[:, 1:]
y = raw_data[:, 0]
print(X.shape)
print(y.shape)

train_samples = 5000
# 转换成1行
#X = X.reshape((X.shape[0], -1))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=train_samples, test_size=10000)

# 对数据进行标准化
scaler = StandardScaler()
# 调用fit_transform将数据缩放到统一尺度，u为均值，s为标准差
# z = (x - u) / s
X_train = scaler.fit_transform(X_train)
# 使用同样的标准对测试集进行缩放
X_test = scaler.transform(X_test)
```

实践

3.训练集可视化

[17]:
```
1 images_and_labels = list(zip(X[:8, ], y[:8, ]))
2 for index, (image, label) in enumerate(images_and_labels):
3     plt.subplot(2, 4, index + 1)
4     # 不显示坐标轴
5     plt.axis('off')
6     plt.imshow(image.reshape(28, 28))
7     plt.title('Training: %i' % label)
8
```

实践

4. 模型训练

```
t0 = time.time()

# C为正则化系数λ的倒数，通常默认为1
# tol为收敛条件
clf = LogisticRegression(C=1,
                         multi_class='ovr',
                         penalty='l2',
                         solver='sag',
                         tol=0.01
                        )
print(clf)
clf.fit(X_train, y_train)

# 获得模型准确度
score = clf.score(X_test, y_test)
print("Test score with L2 penalty: %.4f" % score)

coef = clf.coef_.copy()
plt.figure(figsize=(10, 5))
scale = np.abs(coef).max()
for i in range(10):
    l1_plot = plt.subplot(2, 5, i + 1)
    l1_plot.imshow(coef[i].reshape(28, 28), interpolation='nearest',
                   cmap=plt.cm.RdBu, vmin=-scale, vmax=scale)
    l1_plot.set_xticks(())
    l1_plot.set_yticks(())
    l1_plot.set_xlabel('Class %i' % i)
plt.suptitle('Classification vector for...')

run_time = time.time() - t0
print('Example run in %.3f s' % run_time)
plt.show()
```