上节课回顾

- Caffe
- Tensorflow
- PyTorch

Caffe使用流程

1. 数据准备以及格式转换（现成脚本程序）

2. 定义网络文件（train_val.prototxt）

3. 定义训练中的超参数（solver.prototxt）

4. 训练模型（三种方式运行）

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

TensorFlow

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/gpu:0')
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y,grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

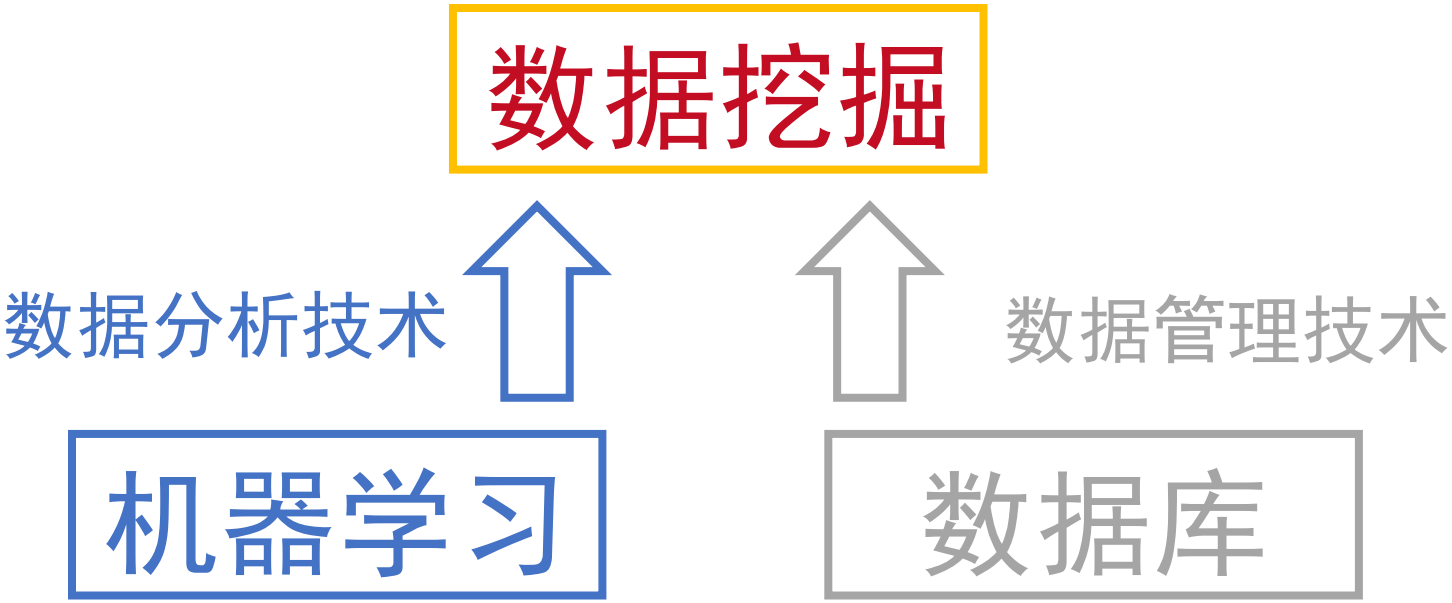
深度学习软件框架建议

- **TensorFlow** 是当前最流行的，适用大部分工作
- **PyTorch** 适合学术研究
- **Caffe**, **TensorFlow** 适合工程项目
- **TensorFlow** or **Caffe2** 适合移动端

深度学习及应用

一番外篇之机器学习

机器学习与数据挖掘



机器学习简介

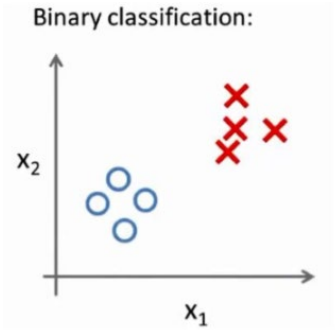
- 机器学习中的“学习(Learning)”指的是从数据中学得模型的过程，又叫“训练(Training)”。这个过程通过执行某个学习算法完成，训练过程中使用的数据称为“训练数据”，其中每一个样本称为一个“训练样本”，训练样本组成的集合称为“训练集”。

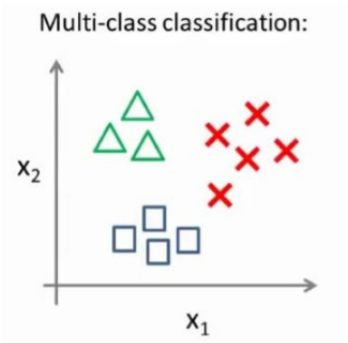
- 机器学习的目标是使学得的模型能很好地适用于“新样本”（或测试数据），而不仅仅在训练样本上工作的很好，我们希望学得模型适用于新样本（未见过的）的能力称为“泛化能力”。

回归、分类与聚类

根据模型输出，可以将任务分为三类：若预测的是连续值，此类任务称为“回归”，若预测的是离散值，则称之为“分类”任务；若不知道数据内在规律也没有数据标签，可以进行“聚类”任务，即将任务数据分为若干个组，每组称为一个“簇”，模型自动将数据集分为不同的“簇”。

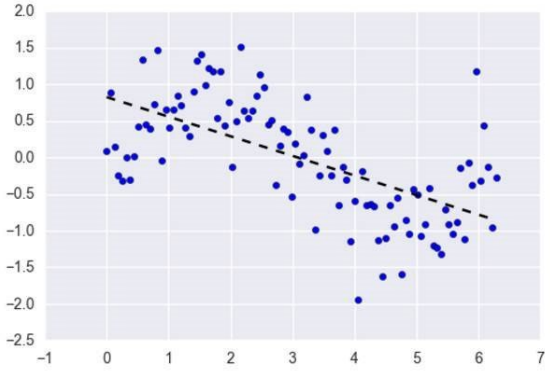
在分类任务中，对只涉及两个类别的“二分类”任务，通常称其中一个类为正类，另一个类为反类；涉及多个类别时，则称为多分类。



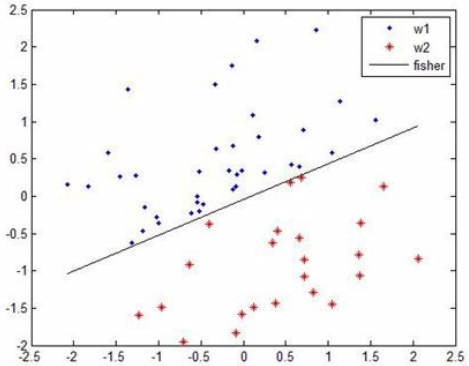


回归、分类与聚类

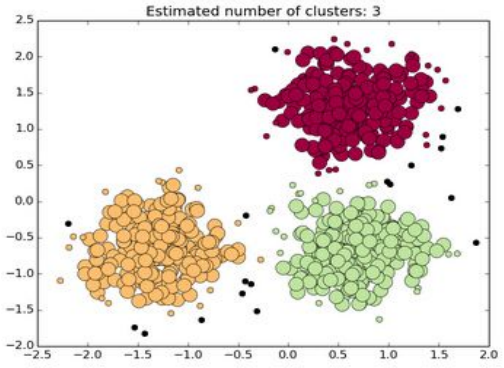
根据模型输出，可以将任务分为三类：若预测的是连续值，此类任务称为“回归”，若预测的是离散值，则称之为“分类”任务；若不知道数据内在规律也没有数据标签，可以进行“聚类”任务，即将任务数据分为若干个组，每组称为一个“簇”，模型自动将数据集分为不同的“簇”。



回归任务



分类任务



聚类任务

基本术语-任务

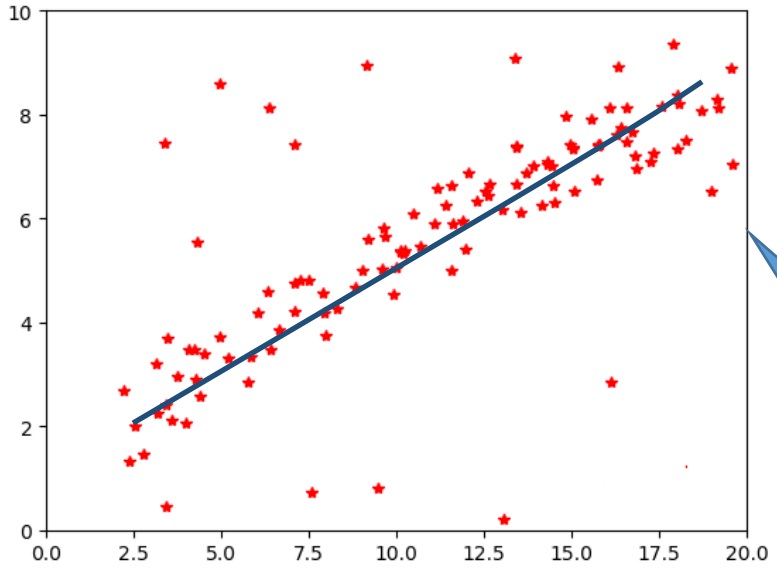
- 预测目标：
  - 分类：离散值
    - ✓二分类：好瓜；坏瓜
    - ✓多分类：冬瓜；南瓜；西瓜
  - 回归：连续值
    - ✓ 瓜的成熟度
  - 聚类：无标记信息

基本术语-任务

- 有无标记信息
  - 监督学习：分类、回归
  - 无监督学习：聚类
  - 半监督学习：两者结合

线性回归与逻辑回归

问题引出：收入预测



收入随工作年限变化情况

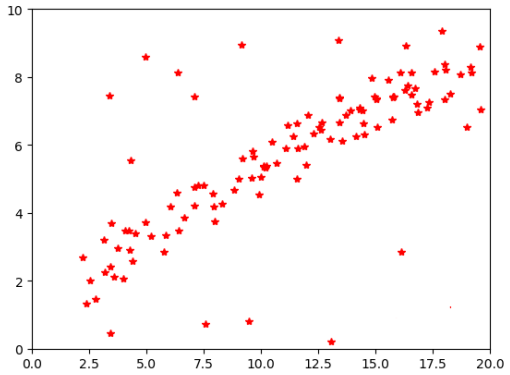
一个简单的想法

使用这样的一条线来拟合数据，获得模型用来预测，即为机器学习中回归问题的一个实例

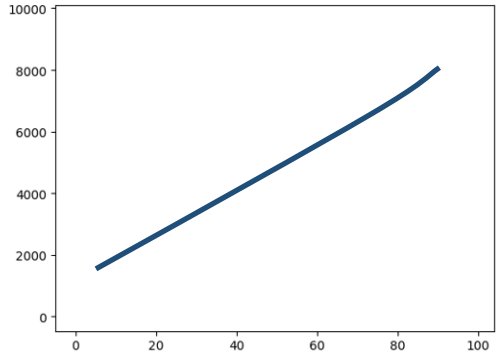
1. 线性回归模型

$$h_\theta(x) = \theta_0 + \theta_1 x$$

如何找到合适的θ呢，我们需要找到一个评价模型参数的方法，用来迭代更新模型参数θ_0，θ_1



$h_\theta(X)$



此处的h即为我们想要从数据中学习的模型，使用的算法是线性回归

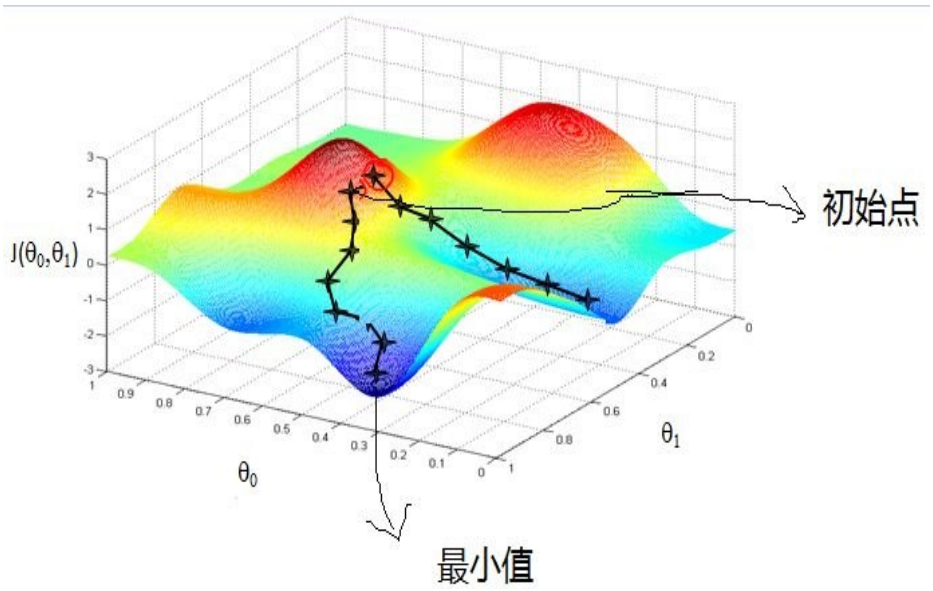
代价函数

- 为了找到合适的参数θ_0，θ_1，需要有一个方法来评判参数，来指导算法去进行参数调整，这样的评价参数的方法（函数），便被称作代价函数(cost function)。
- 线性回归中的代价函数：
  - 均方误差：$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$
- 也就是说，代价函数的值越小，便说明模型越好。这就给了我们更新参数的指导，即向着使代价函数降低的方向更新θ_0，θ_1

梯度下降

$$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$$

$$\frac{d\mathrm{J}(\theta_0,\theta_1)}{d\theta_{\mathrm{i}}}$$



根据梯度信息更新参数

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

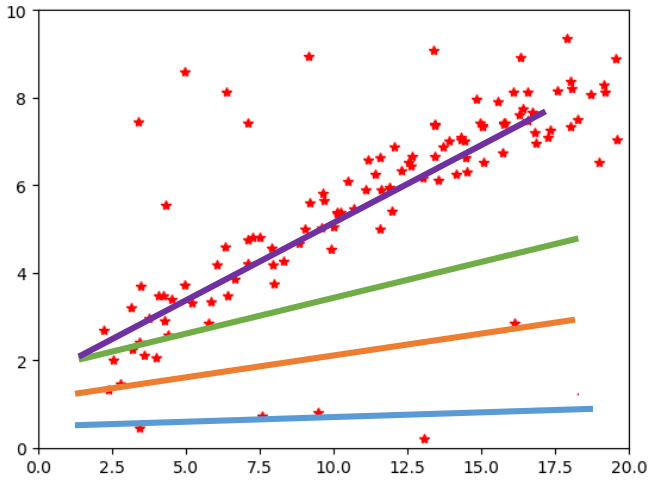
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

update θ_0 and θ_1 simultaneously

注：α为学习率，超参数（需要人为设置）

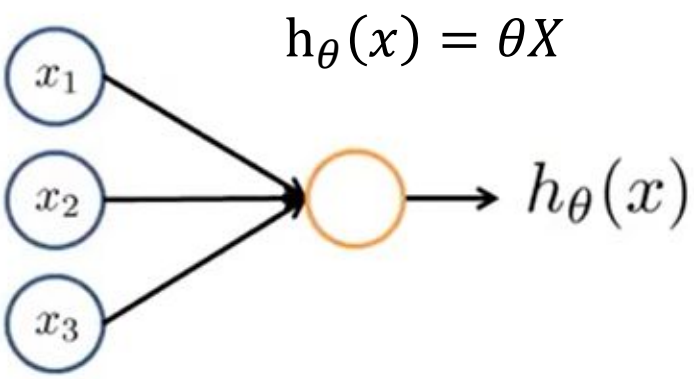
迭代更新线性模型



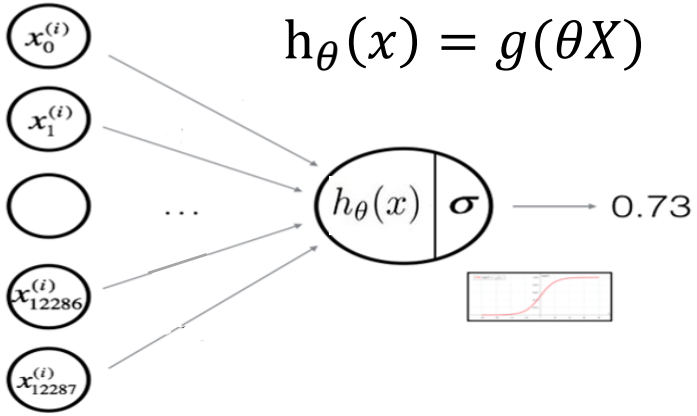
训练过程中不断的迭代来更新参数 θ_0, θ_1， 直至收敛

2. 逻辑回归

- 逻辑回归虽然名中有“回归”二字，但它却是用来处理分类问题的。它和线性模型十分相似，唯一的区别就是引入了非线性函数σ。



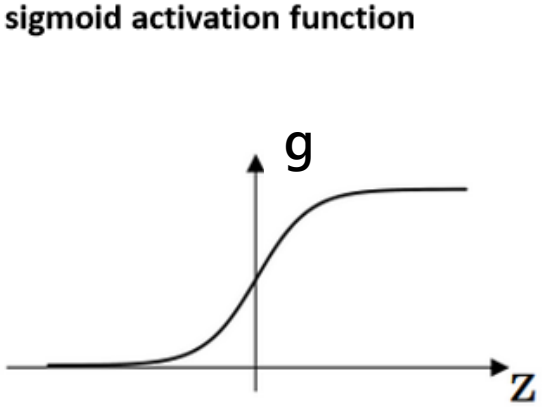
线性模型



逻辑回归

逻辑回归模型

- 逻辑回归的模型假设为：$\mathrm{h}_\theta(x) = g(\theta X)$，其中$g$代表一个逻辑函数，公式为$g(z) = \frac{1}{1+e^{-z}}$，它的图像如下图所示，因形如S型，它的名字叫做Sigmoid函数。。
- 故逻辑回归的完整形式为$\mathrm{h}_\theta(x) = \frac{1}{1+e^{-\theta X}}$。



Sigmoid函数的优点如下：
①引入非线性，且求导容易
②限制输出范围，适合分类任务
③对临界点细微变化敏感，利于分类

使用逻辑回归进行分类

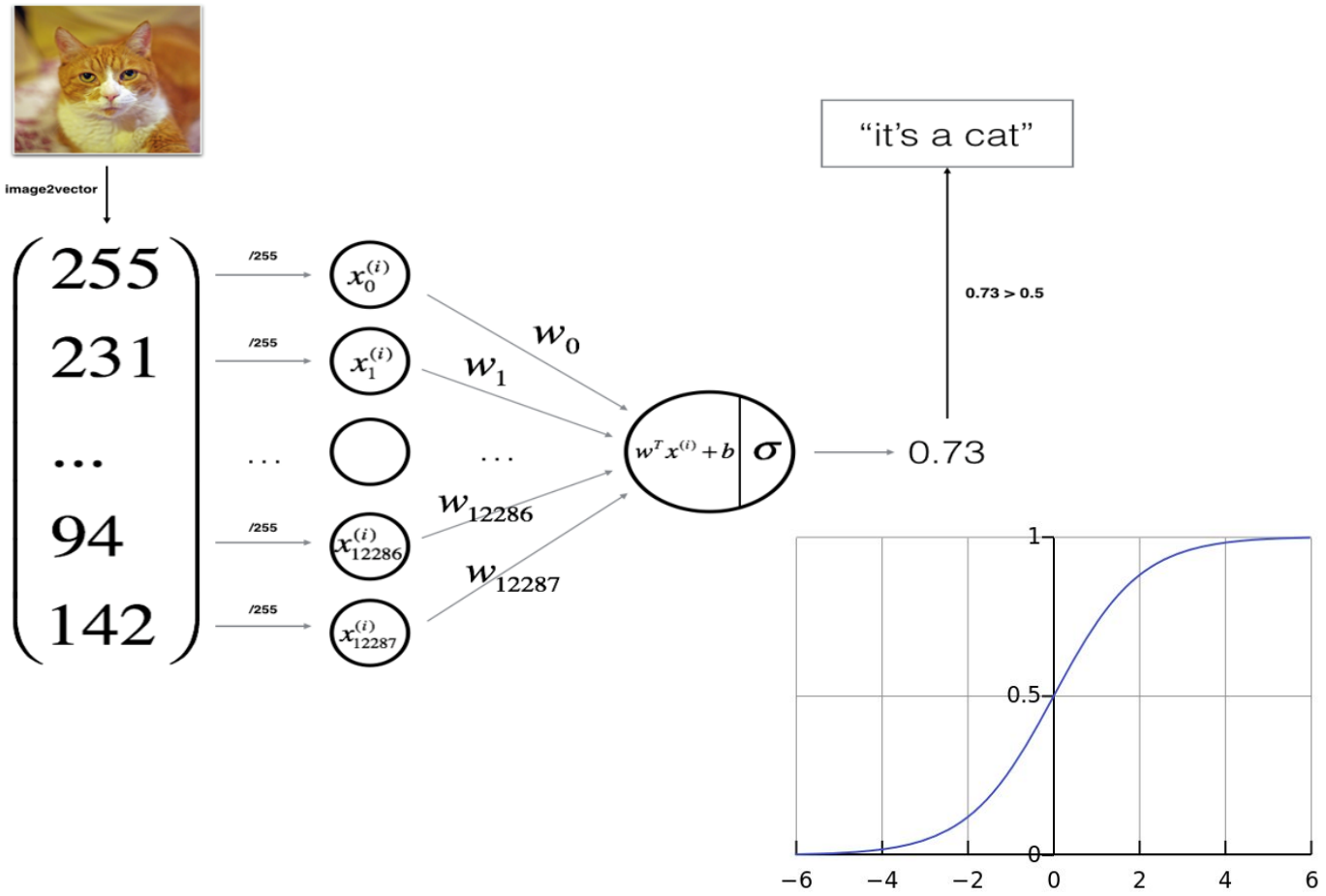
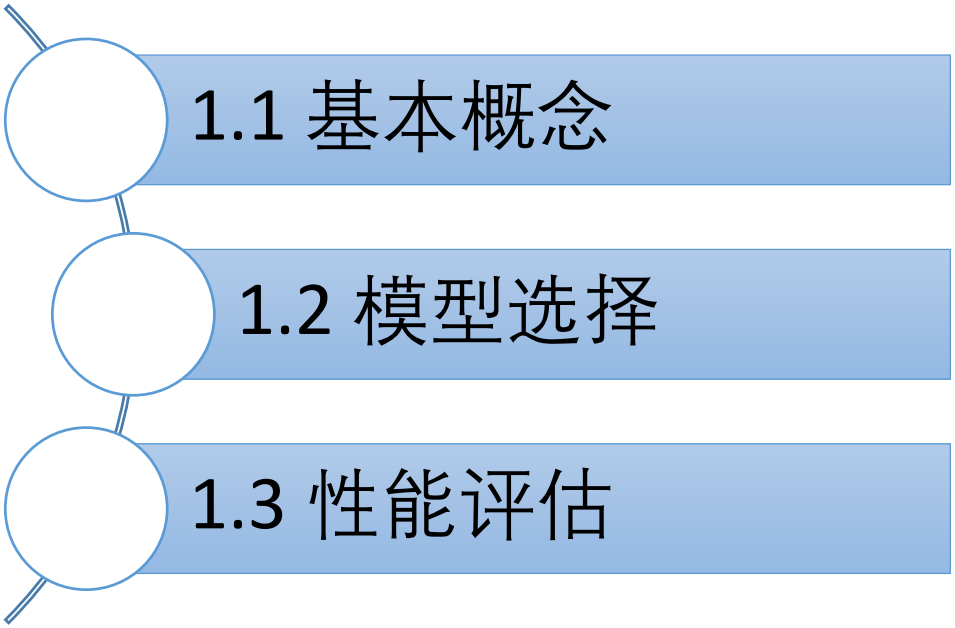
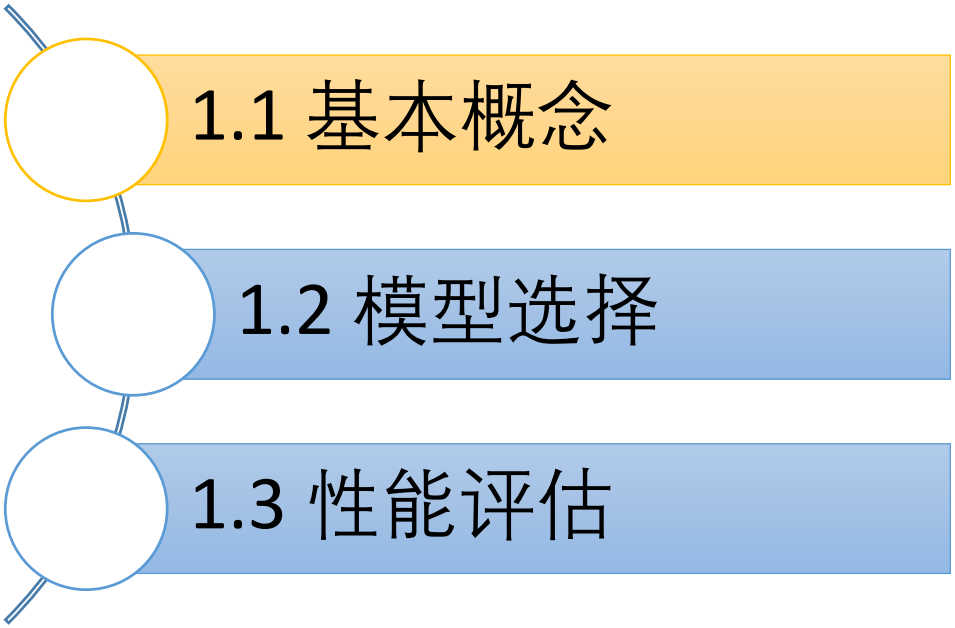


image2vector
255
231
...
94
142
/255
/255
/255
/255
$x_0^{(i)}$
$x_1^{(i)}$
$x_{12286}^{(i)}$
$x_{12287}^{(i)}$
w_0
w_1
w_{12286}
w_{12287}
$w^T x^{(i)} + b$
σ
0.73
0.73 > 0.5
"it's a cat"
1
0.5
0
−6
−4
−2
0
2
4
6

模型选择与评估


1.1 基本概念
1.2 模型选择
1.3 性能评估


1.1 基本概念
1.2 模型选择
1.3 性能评估

1.1 基本概念

- 1.1.1 泛化性能
- 1.1.2 模型评估目标
- 1.1.3 数据集前提假设
- 1.1.4 欠拟合与过拟合
- 1.1.5 偏差与方差
- 1.1.6 超参数
- 1.1.7 超参数调整
  - 模型复杂度d
  - 正则化参数λ
  - 训练集大小
  - 神经网络

1.1.1 泛化性能Generalization Performance

- 如何评判一个模型的好坏?
- 在**未知的数据**中能够做出准确的预测
- **“泛化性能”**
  - 泛化精度(generalization accuracy)
  - 泛化误差(generalization error)
- 如何“测量”?

1.1.2 模型评估目标

- **估计泛化性能**，即模型对未来(看不见的)数据的预测性能
- 根据评估调整学习算法，**选择合适的参数，**使模型达到最佳性能
- **比较不同的算法**，找出最适合解决当前问题的学习算法

1.1.3 数据集的前提假设Assumptions

- **i.i.d.(independent and identically distributed)** 独立同分布
- **测试误差(test error)**是**泛化误差(generalization error)** 的**无偏估计**
- **训练误差(training error)**是对模型性能**有偏差的估计**

1.1.4 欠拟合underfit与过拟合overfit

- 过拟合：

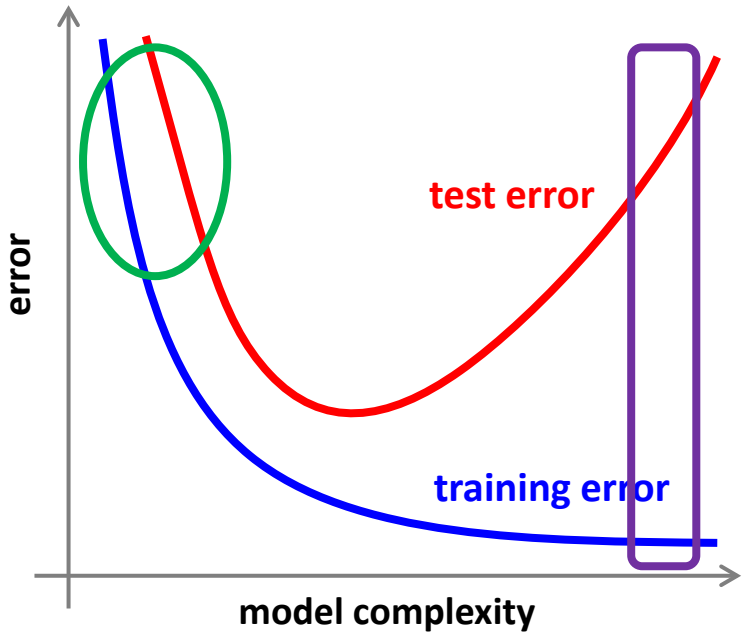
学习器把训练样本学习的“太好”，将训练样本本身的特点当做所有样本的一般性质，导致泛化性能下降

- 欠拟合：

对训练样本的一般性质尚未学好

1.1.4 欠拟合underfit与过拟合overfit

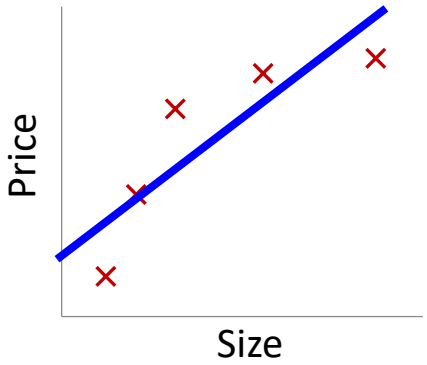
- 判断模型对训练集和测试集**预测结果性能**的术语



- **欠拟合underfit**：
  训练误差和测试误差都很大
- **过拟合overfit**：
  训练误差和测试误差差距大
  (测试误差更高)

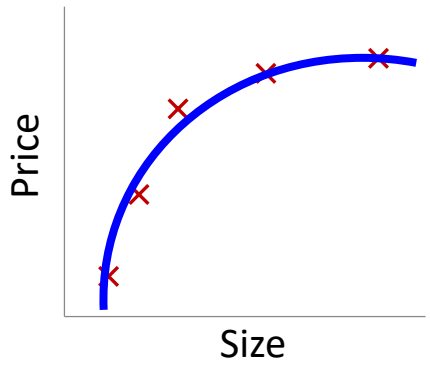
1.1.4 欠拟合underfit与过拟合overfit

线性回归（房价预测）



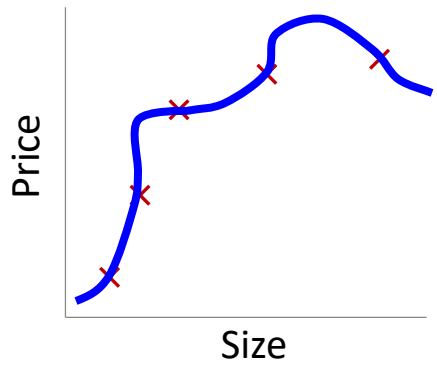
$\theta_0 + \theta_1 x$

欠拟合



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

适合



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

过拟合

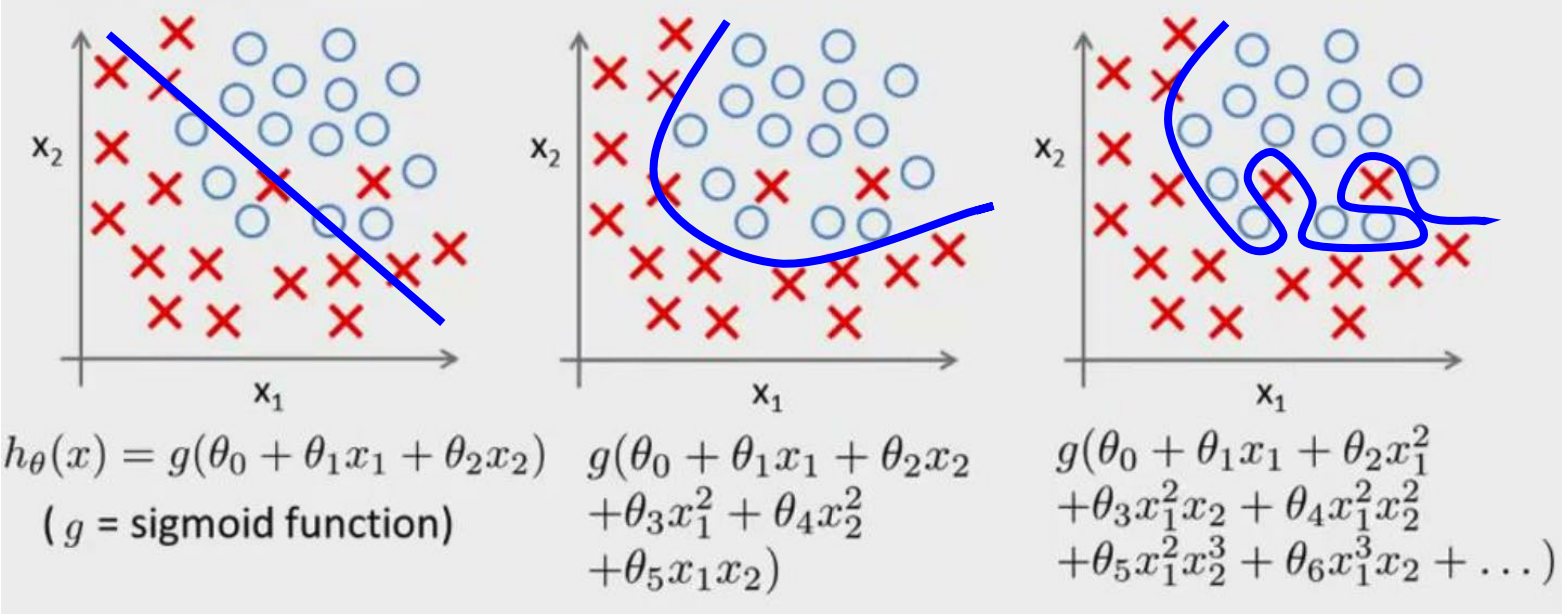
过拟合Overfitting：使用大量特征学习，预测结果

在训练集的loss很小 $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$

但在新样本的测试中，效果不好

1.1.4 欠拟合underfit与过拟合overfit

逻辑回归（分类任务）

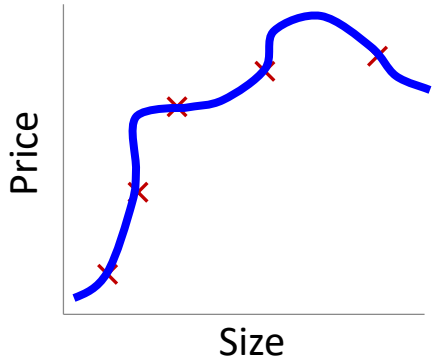


欠拟合　　适合　　过拟合

1.1.4 欠拟合underfit与过拟合overfit

- 解决过拟合

$x_1 =$ 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}



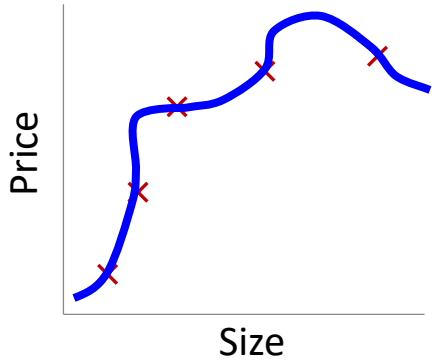
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

1.1.4 欠拟合underfit与过拟合overfit

- 解决过拟合

1. 减少参数量

- 留下合适的特征
- 选择合适的算法



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

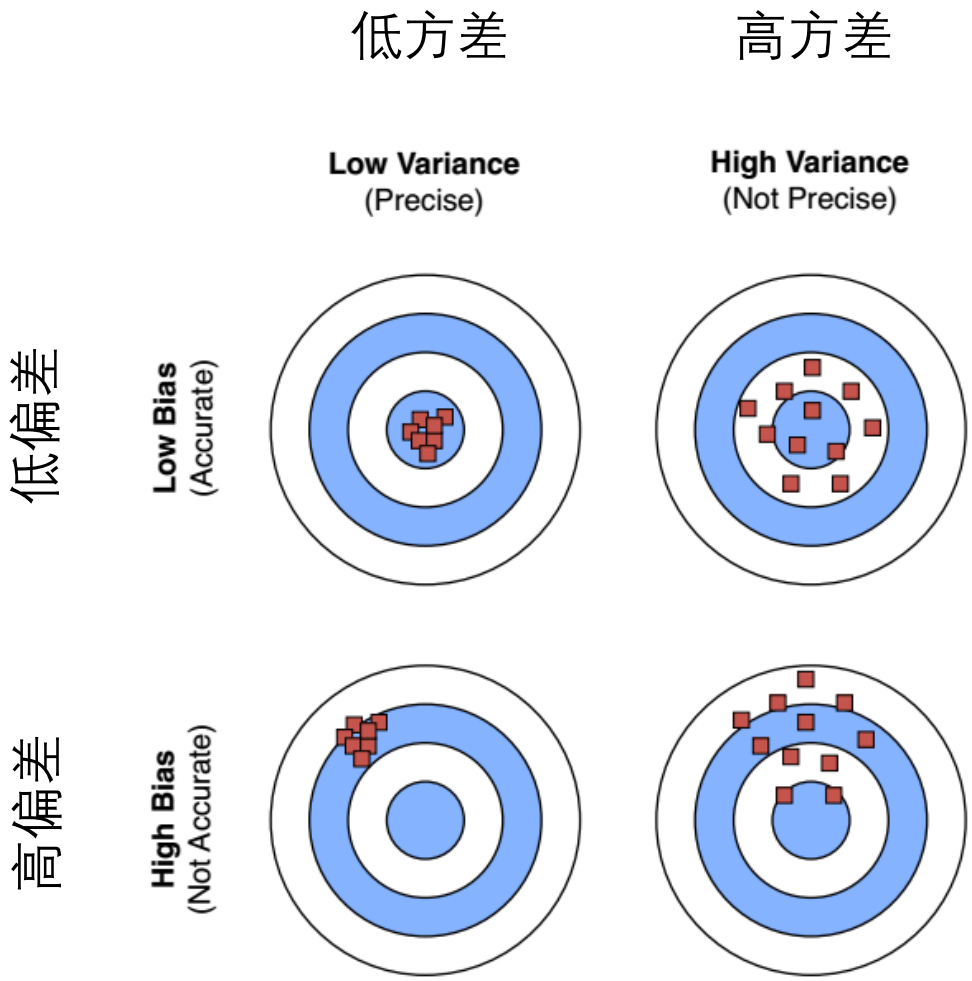
2. 正则化

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \quad + \lambda \sum_{i=0}^{m} \theta_i^2$$

“奥卡姆剃刀”是一种常用的、自然科学研究中最基本的原则，即“若有多个假设与观察一致，选最简单的那个.

1.1.4 欠拟合与过拟合——直观理解

1.1.5 偏差Bias与方差Variance

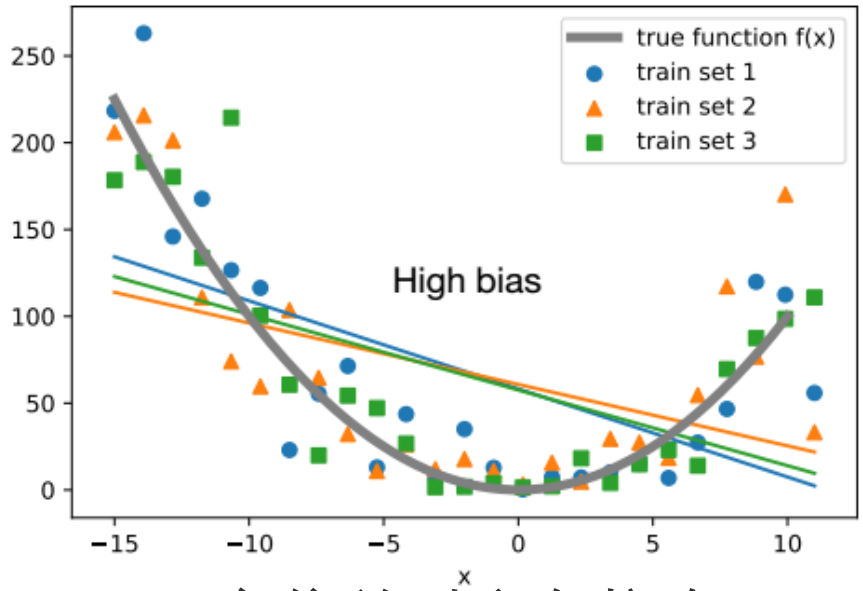


偏差：衡量模型期望输出与真实标记的差别。

方差：衡量模型对于给定值的预测稳定性。

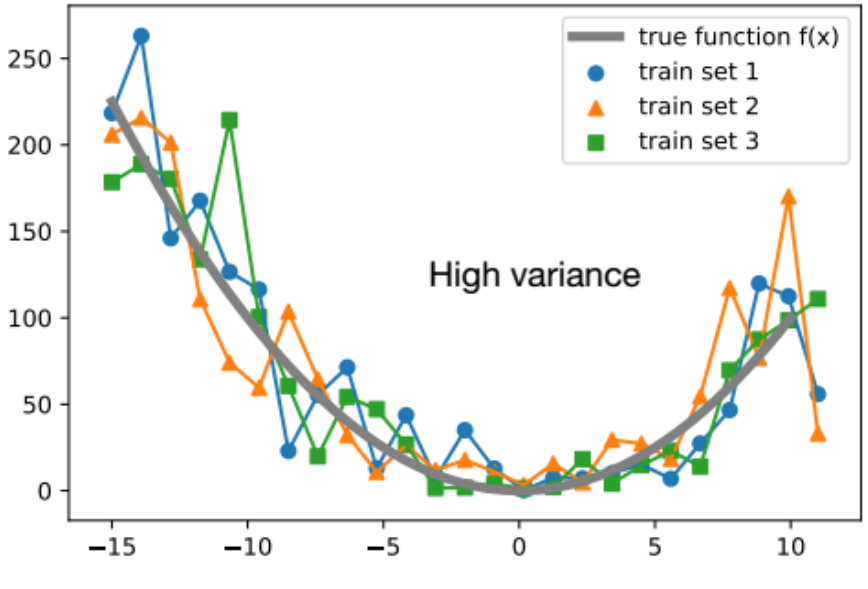
1.1.5 偏差Bias与方差Variance

线性回归模型



高偏差对应欠拟合

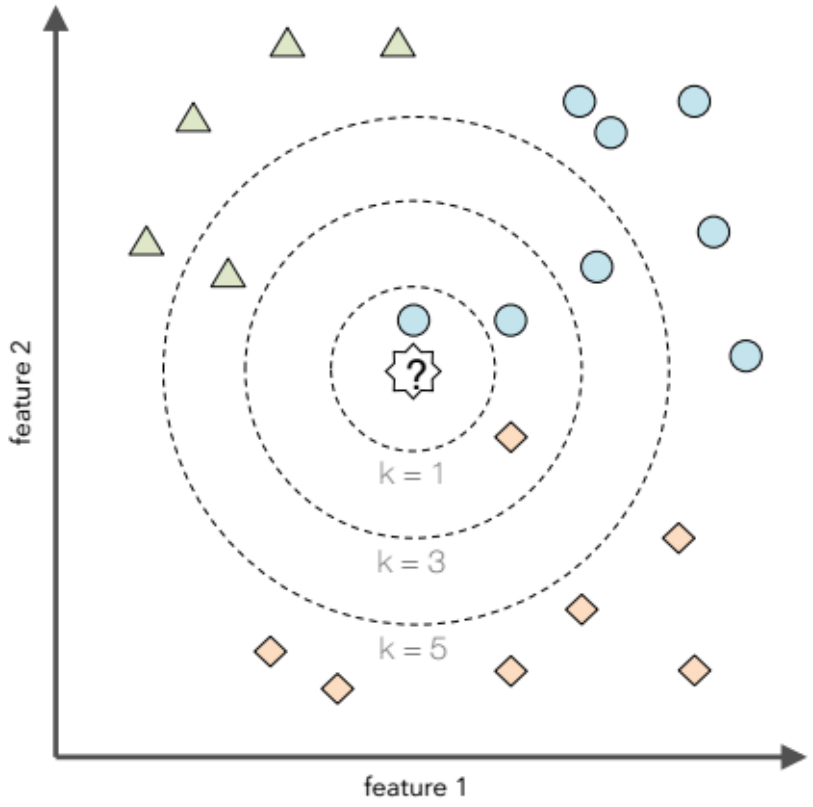
决策树模型



高方差对应过拟合

- 偏差度量了学习算法期望预测与真实结果的偏离程度；即刻画了学习算法本身的拟合能力；
- 方差度量了同样大小训练集的变动所导致的学习性能的变化；即刻画了数据扰动所造成的影响；

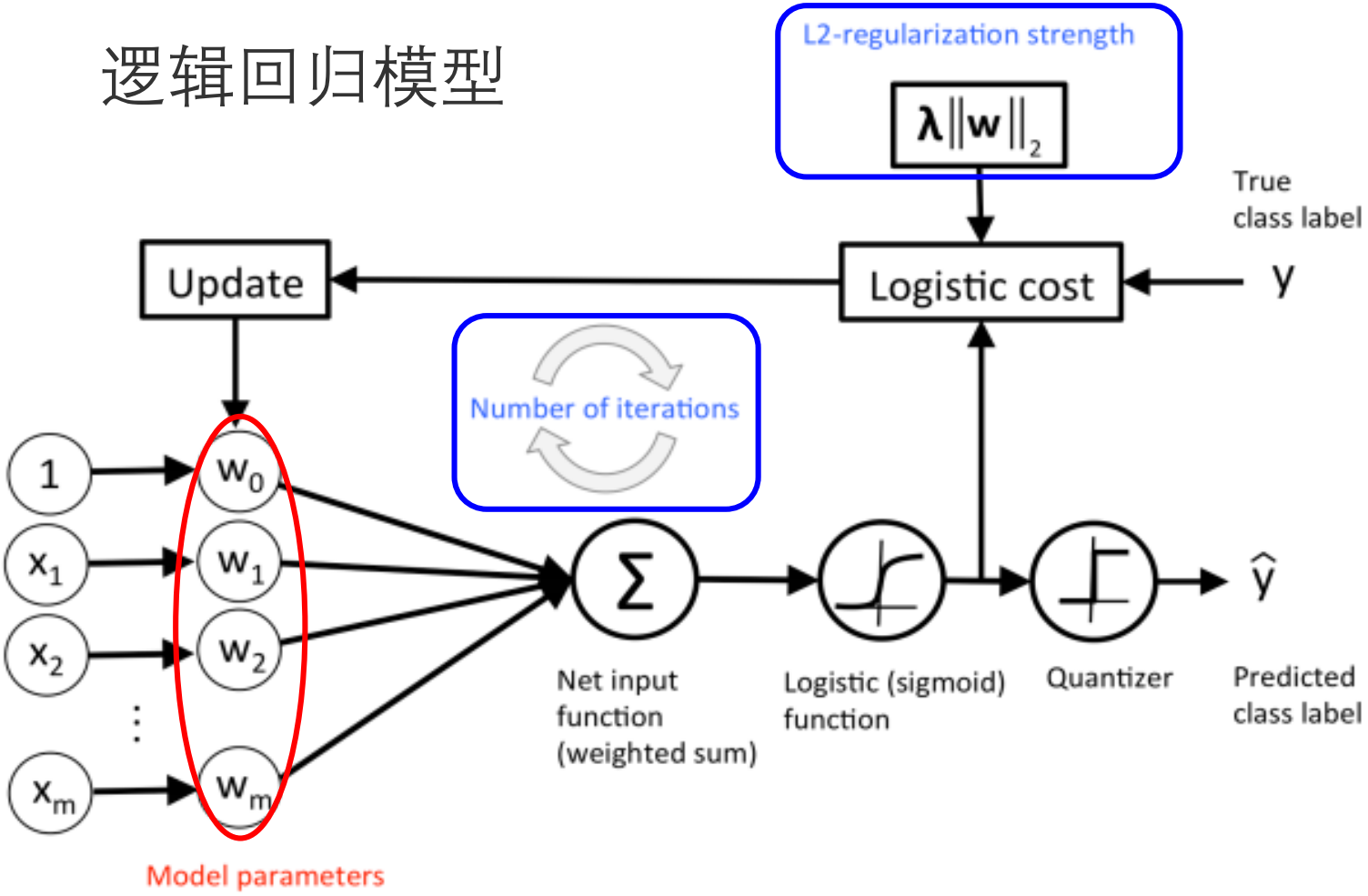
1.1.6 超参数Hyperparameters



- **超参数**：学习算法本身的参数。需要预先定义，不能从模型的训练中得出
- 如学习率、迭代次数

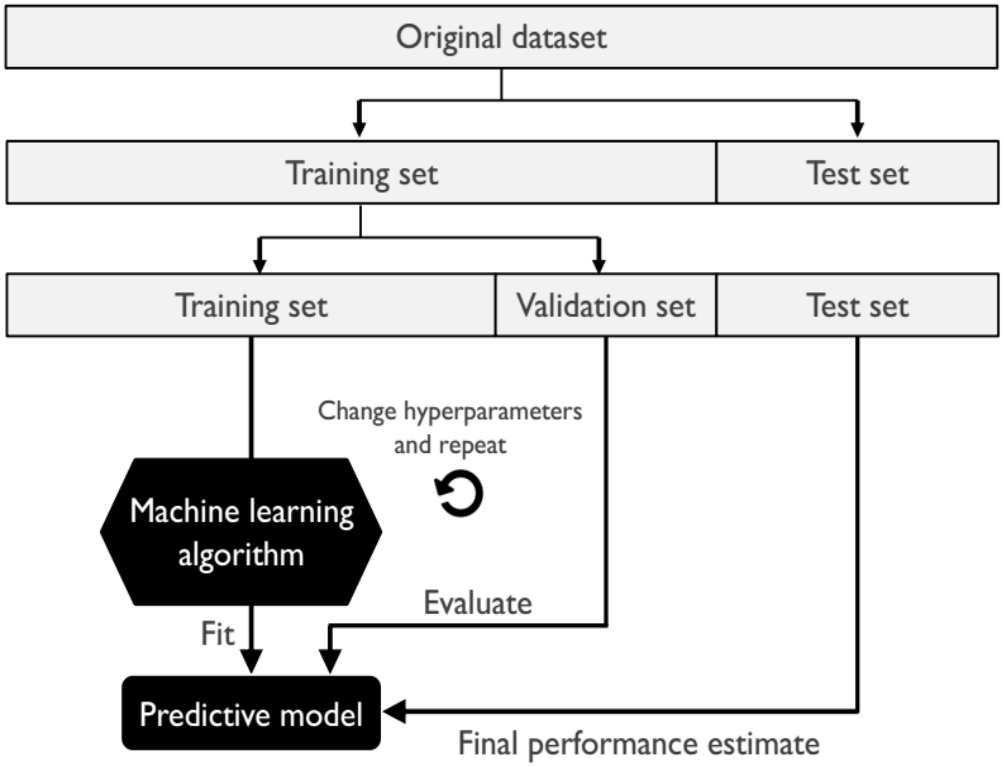
1.1.6 超参数Hyperparameters

逻辑回归模型



1.1.7 超参数调整Hyperparameter Tuning

- 通过设置不同的值，训练不同的模型和选择更好的测试值来决定



Training error:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

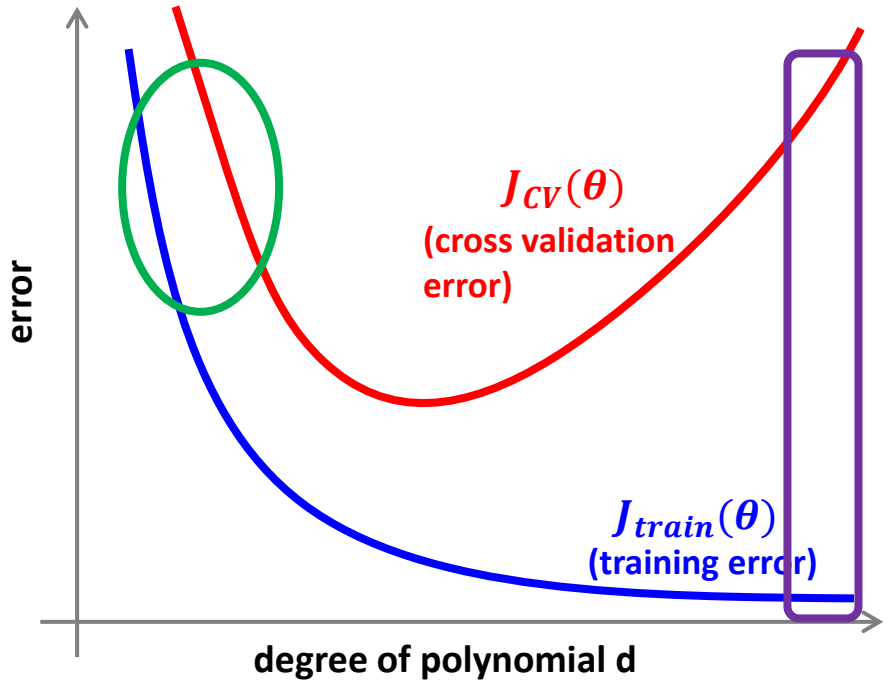
Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

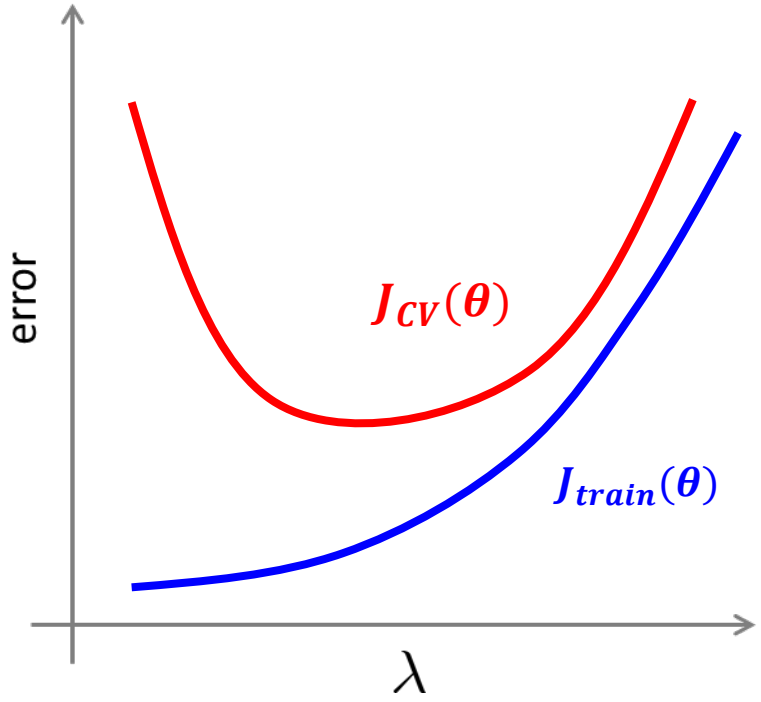
1.1.7 超参数调整——模型复杂度d



- **Bias (underfit):**
  - 模型复杂度低
  - $J_{\text{train}}(\theta)$较高
  - $J_{\text{train}}(\theta) \approx J_{cv}(\theta)$
- **Variance (overfit):**
  - 模型复杂度高
  - $J_{\text{train}}(\theta)$较低
  - $J_{\text{train}}(\theta) \ll J_{cv}(\theta)$

1.1.7 超参数调整——正则化参数λ

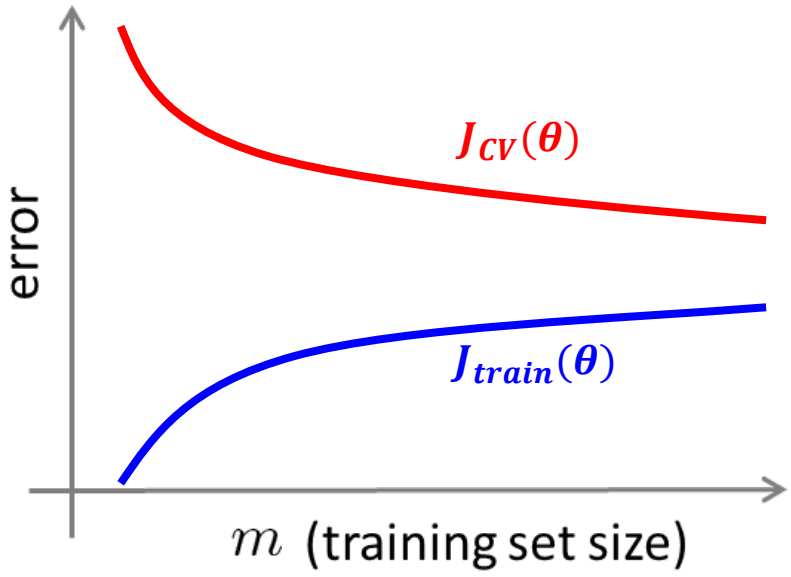
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

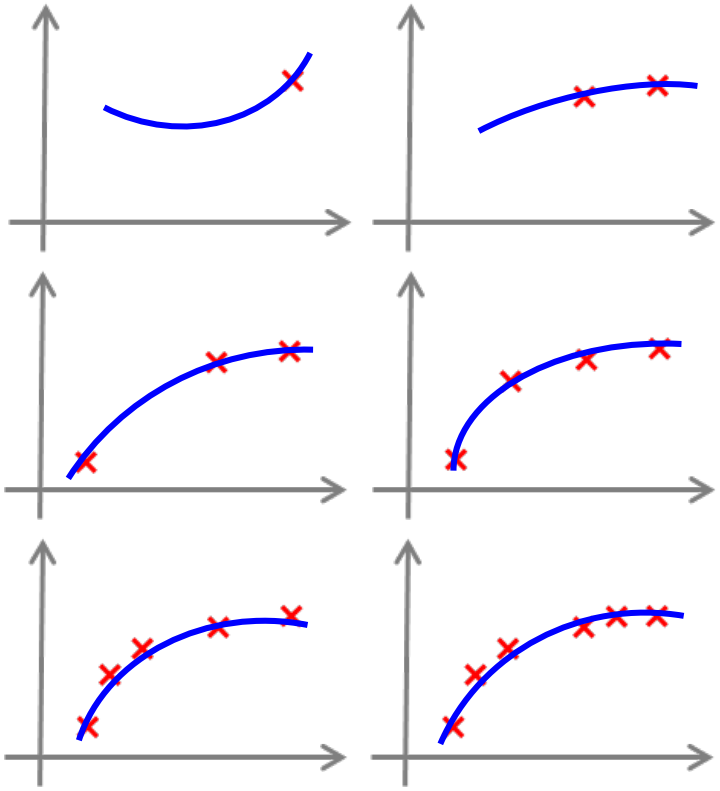


- 当 λ 较大时，参数变小，模拟函数变简单，出现欠拟合
- 当 λ较小时，参数变大，模拟函数更复杂，出现过拟合

1.1.7 超参数调整——训练集大小

- 假设函数: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

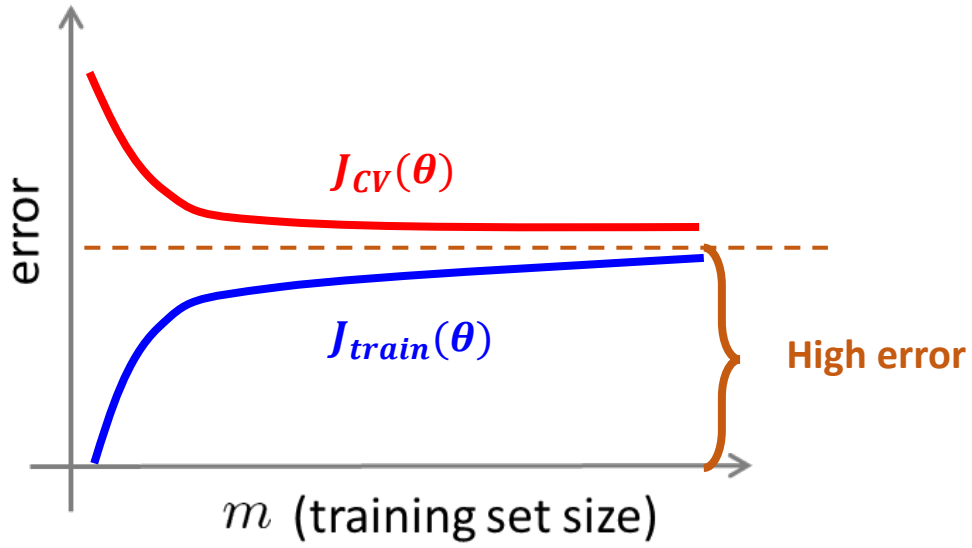


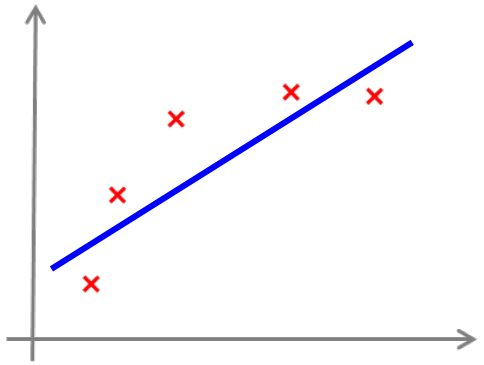


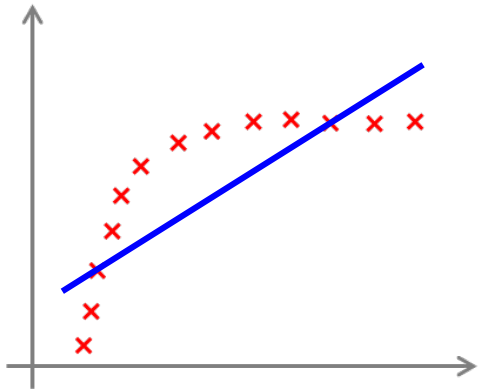
- 训练误差随训练集增大而逐渐增加
- 验证误差随训练集增大而逐渐减小

1.1.7 超参数调整——训练集大小

- 假设函数: $h_\theta(x) = \theta_0 + \theta_1 x$



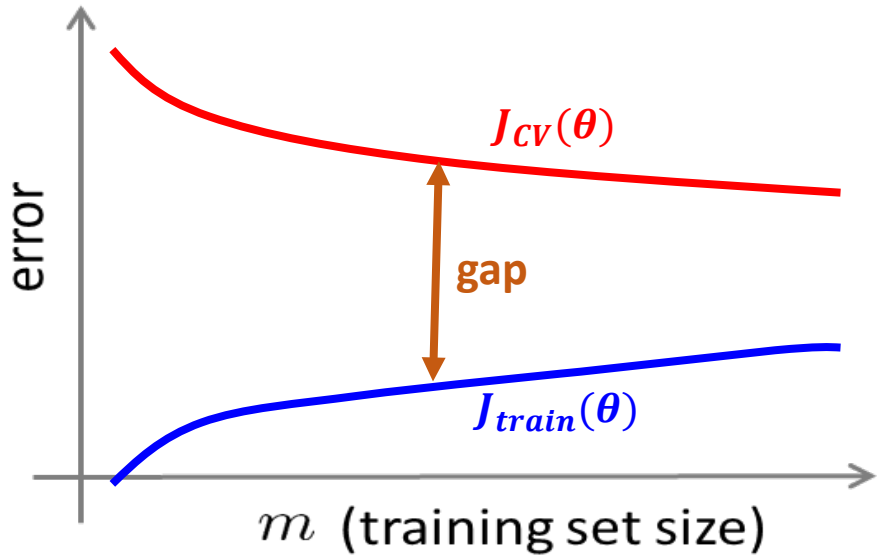


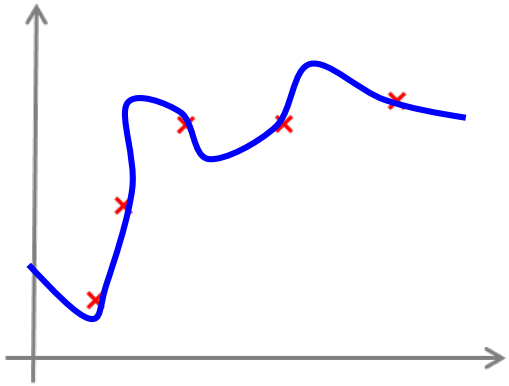


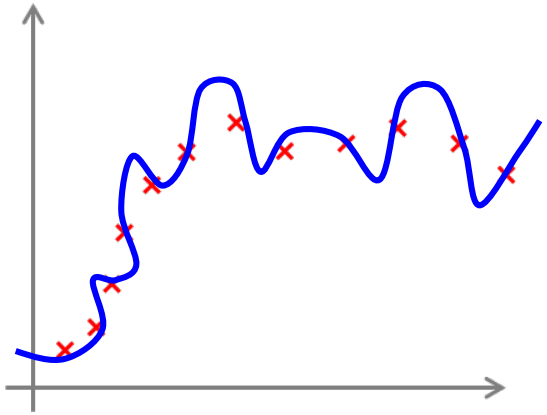
- 在欠拟合（高偏差）的情况下，增大训练集样本，不会提升性能

1.1.7 超参数调整——训练集大小

- 假设函数: $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{100} x^{100}$
  (不考虑正则化参数λ)







- 在过拟合（高方差）状态下，继续增大训练集，两条曲线逐渐逼近，交叉验证集的误差逐渐减小，对算法性能有帮助

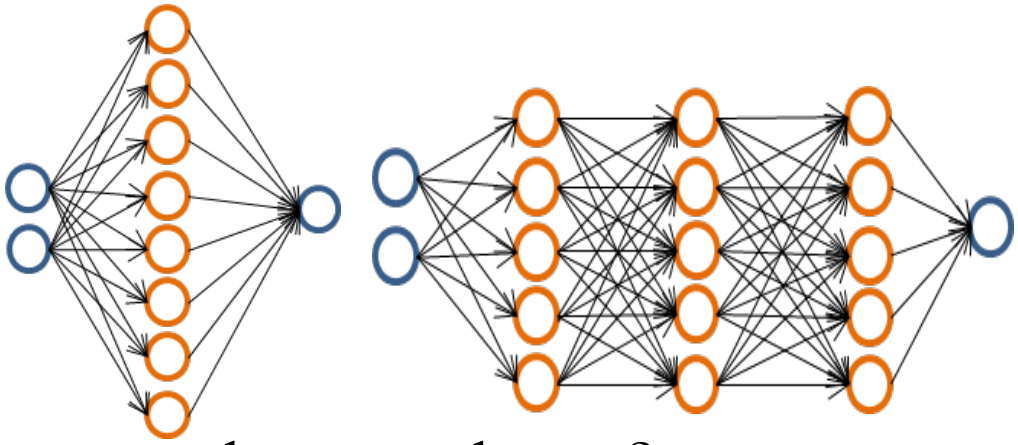
1.1.7 超参数调整——神经网络

“小”的神经网络

- 层次少，参数少
- 更可能欠拟合
- 计算代价小

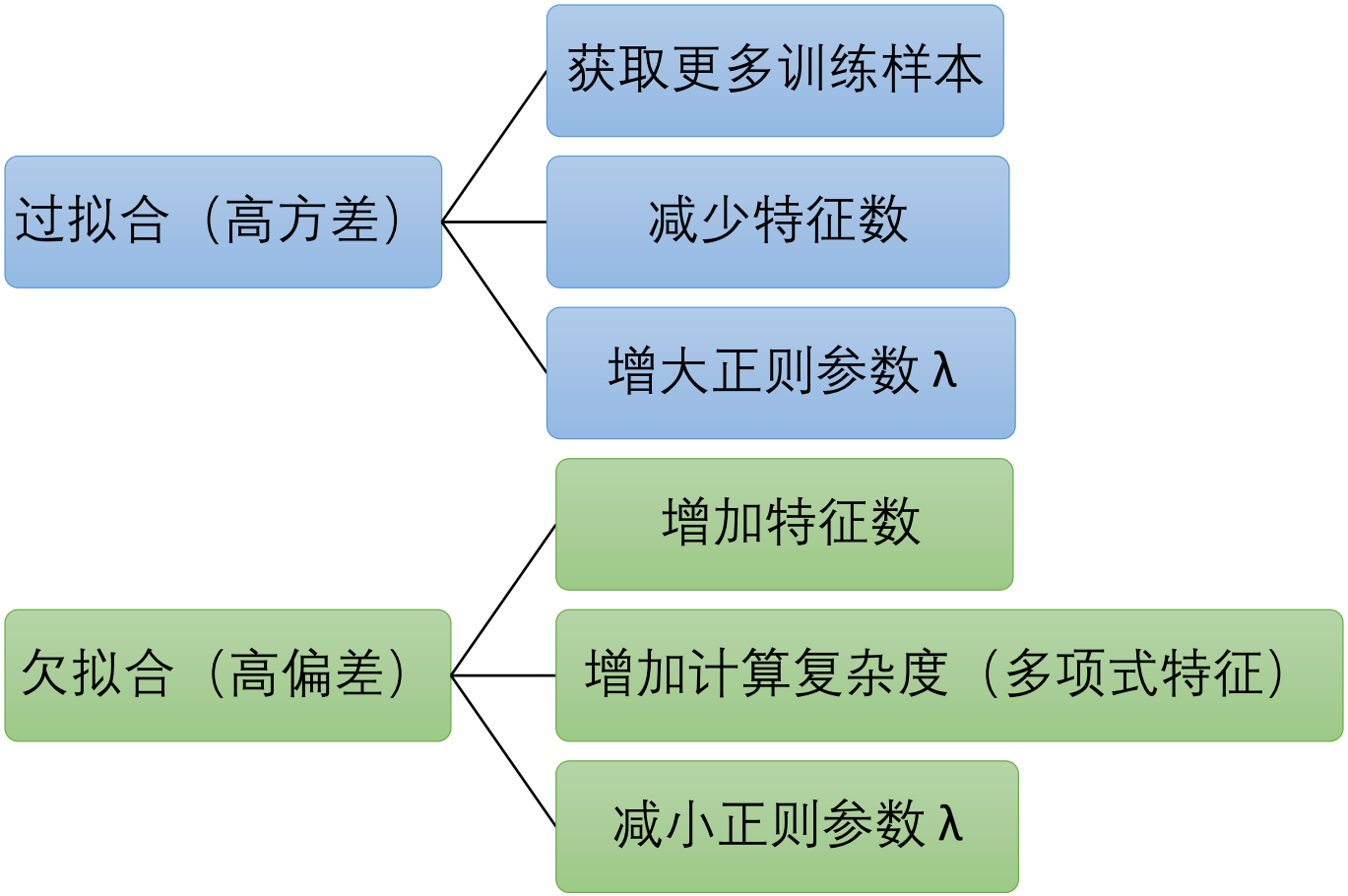
“大”的神经网络

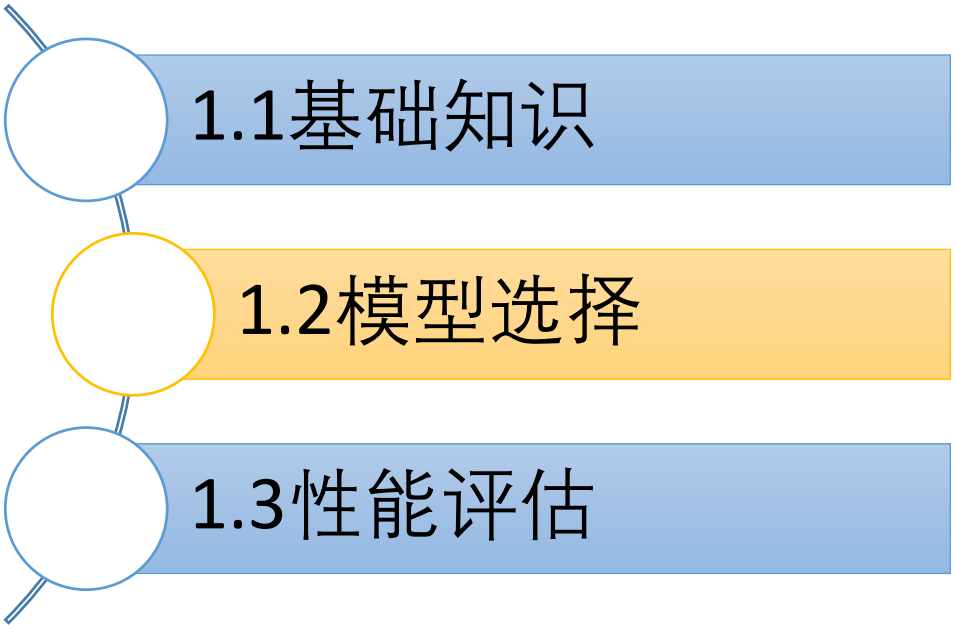
- 层次多，参数多
- 更可能过拟合（正则化调整）
- 计算代价大



larger or deeper?

1.1.7 超参数调整——小结


过拟合（高方差）
获取更多训练样本
减少特征数
增大正则参数 λ
欠拟合（高偏差）
增加特征数
增加计算复杂度（多项式特征）
减小正则参数 λ


1.1基础知识
1.2模型选择
1.3性能评估

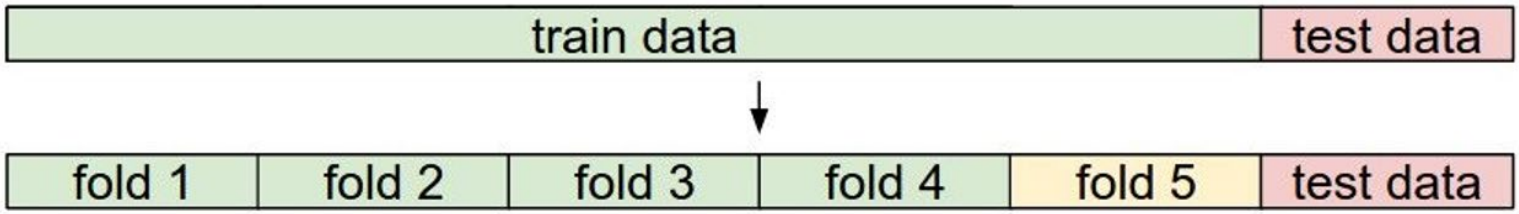
1.2 模型选择

- 1.2.1 留出法
  - 概述
  - 留出法评估
  - 三向留出法
  - 重复留出法
- 1.2.2 交叉验证
  - 概述
  - k-折交叉验证评估
- 1.2.3 特殊交叉验证
  - k=2
  - k=n LOOCV
  - 嵌套交叉验证
- 1.2.4 总结

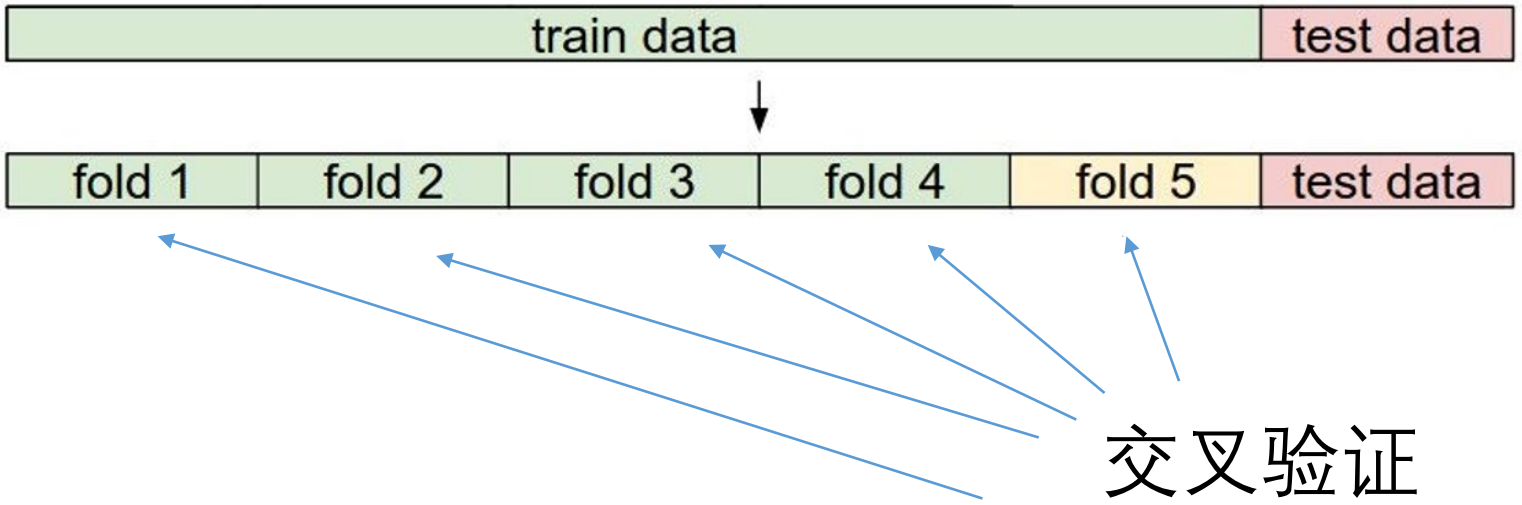
数据集划分

- 数据集的拆分：训练集，测试集

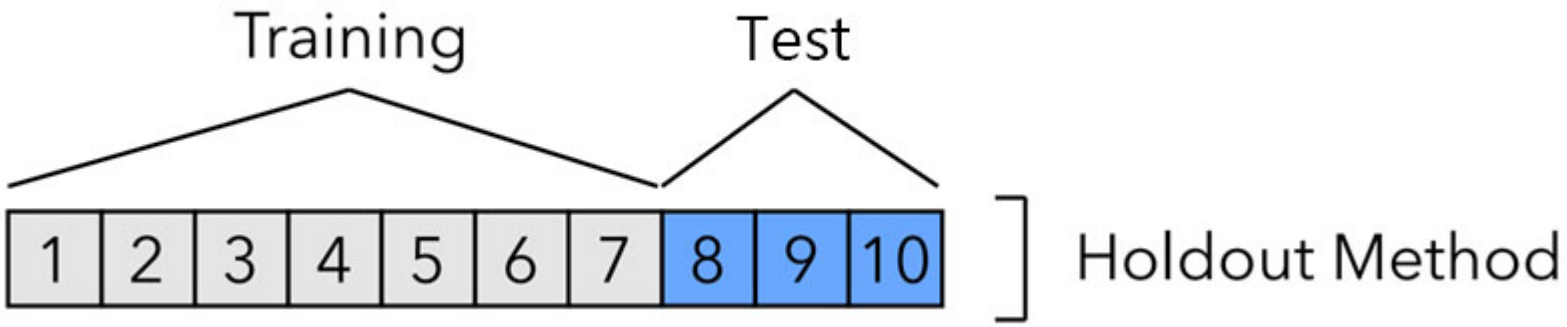
训练集（训练集+验证集），测试集



交叉循环验证取平均

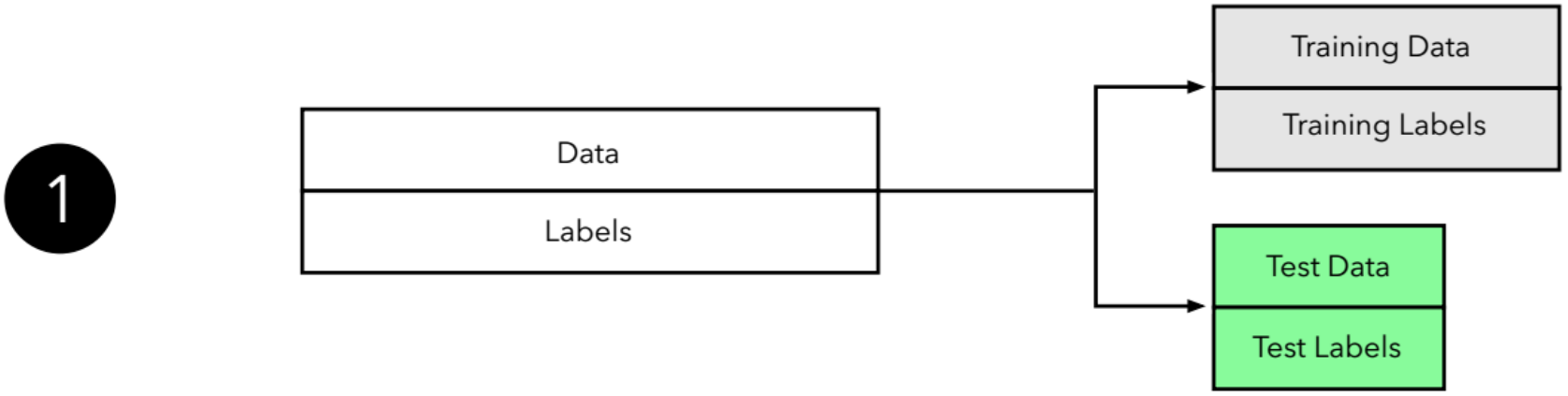

train data
test data
fold 1
fold 2
fold 3
fold 4
fold 5
test data
交叉验证

1.2.1 留出法Hold-out——概述



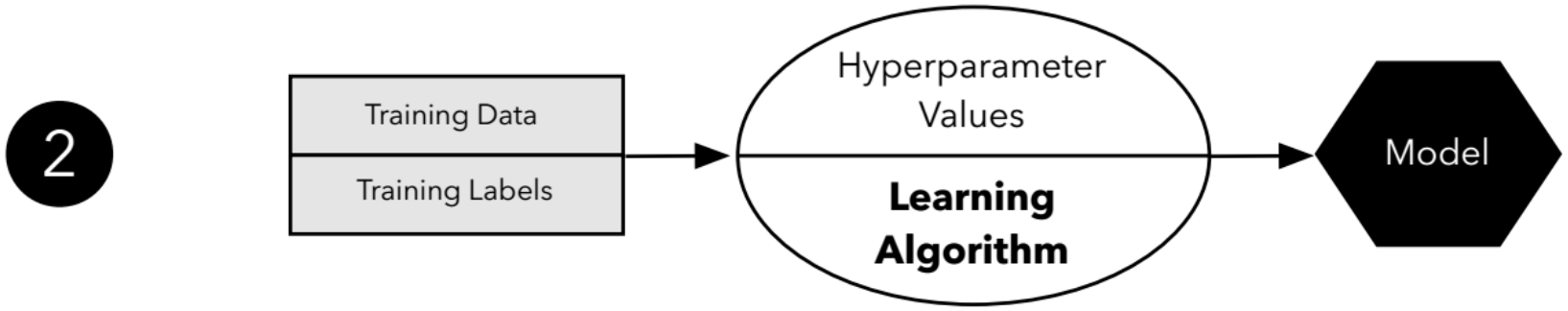
- 将数据集分割成**训练集**和**测试集**
- 一个简单的**随机子抽样**过程
- 假设所有的数据都来自于**相同的概率分布**(相对于每个分类)
- 实践中，一般随机选择**2/3作为训练集**，**1/3作为测试集**

1.2.1 留出法——评估Holdout evaluation



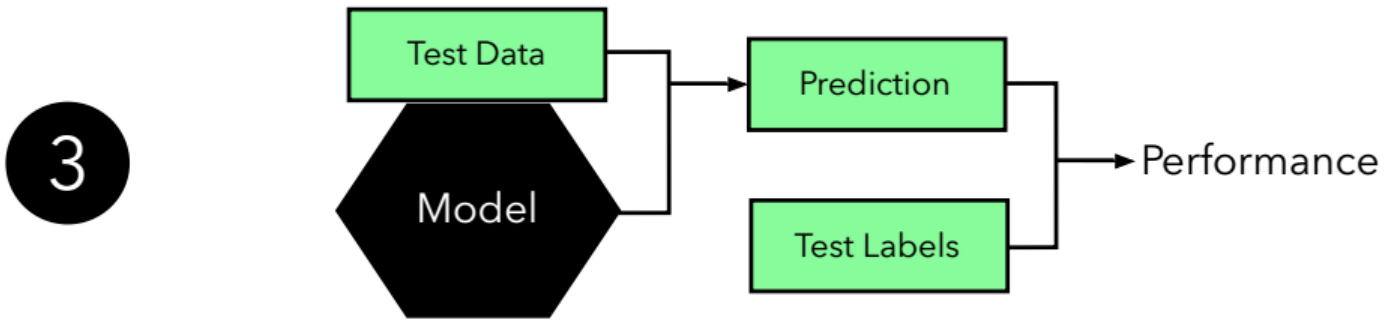
- 将可用数据随机划分为两个子集:一个**训练集**和一个**测试集**

1.2.1 留出法——评估Holdout evaluation



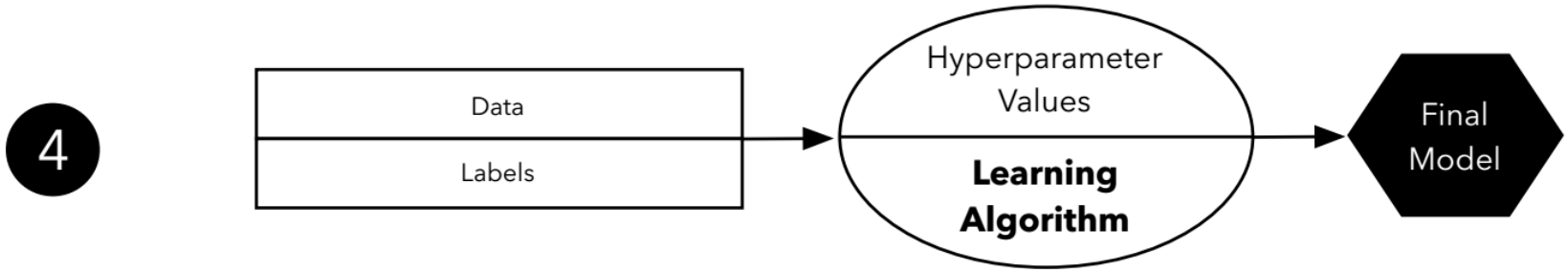
- 对于**训练集数据**，选择学习算法并**固定超参数**值，训练模型
- 超参数为**预先设定好的**经验值

1.2.1 留出法——评估Holdout evaluation



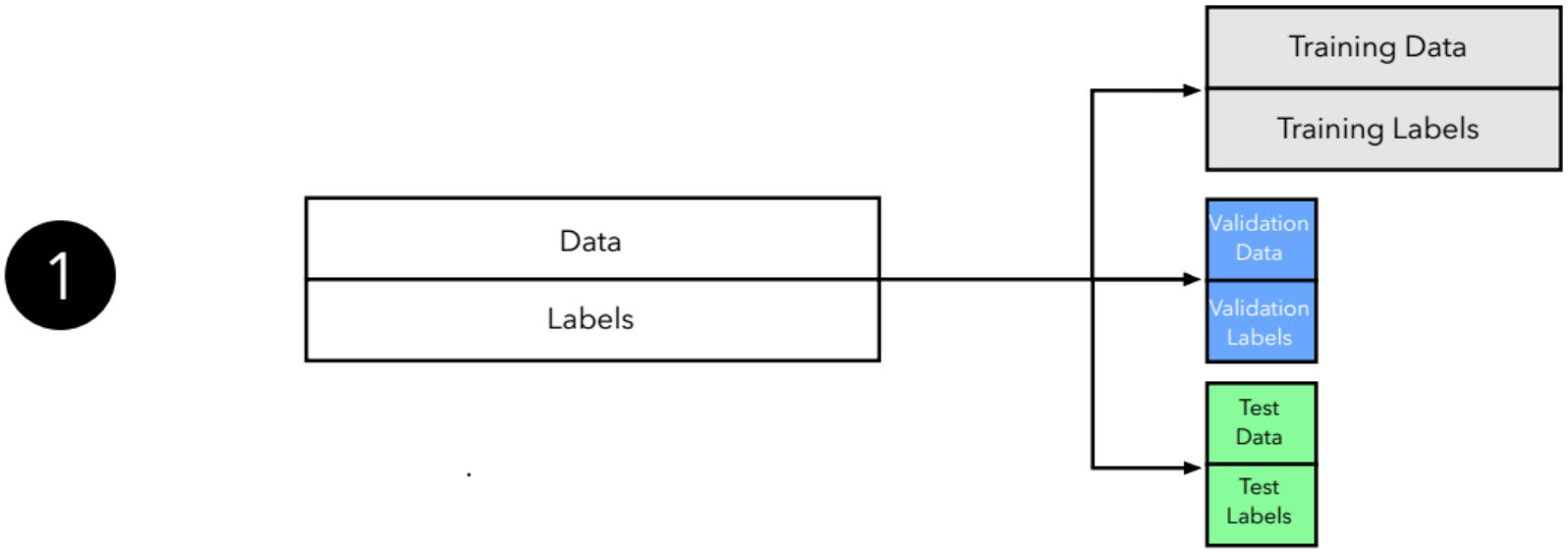
- 使用训练好的模型**预测测试集数据的类标签**
- 将预测的类标签与测试集正确的类标签进行**比较**
- 估计模型的**精度或误差**

1.2.1 留出法——评估Holdout evaluation



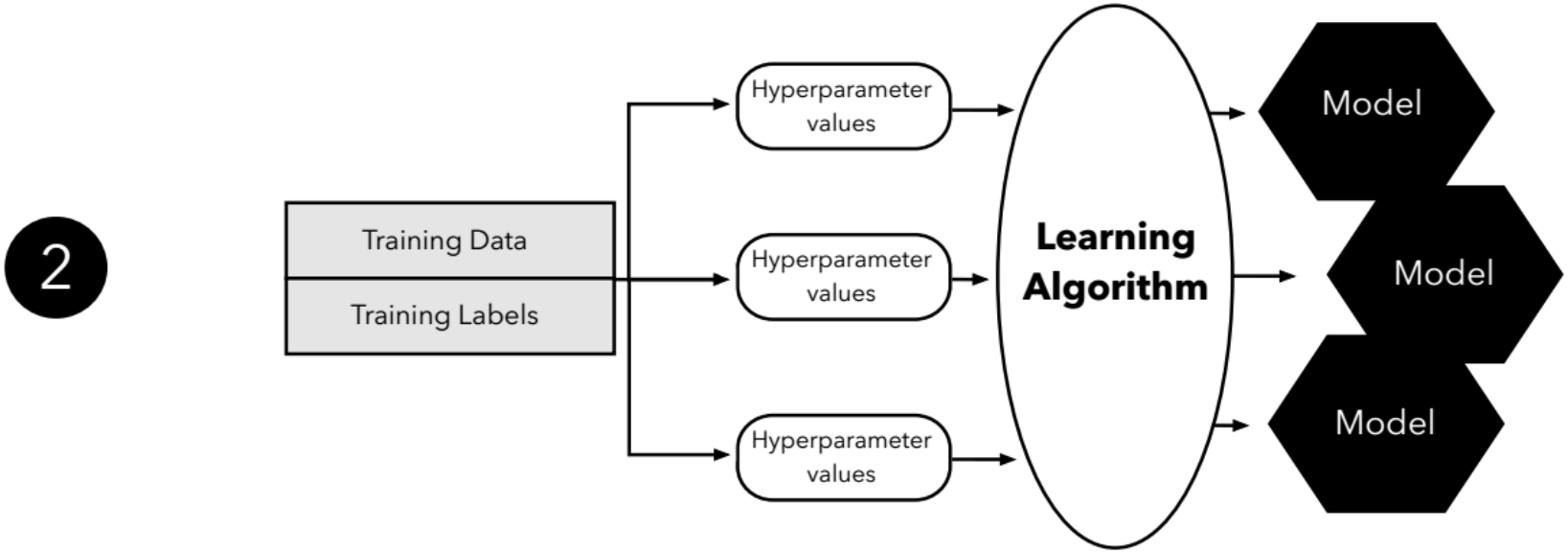
- 将测试集与训练集**合并**，**重新训练模型**

1.2.1 留出法——三向留出法Three-Way Holdout Method（超参数调优Hyperparameter Tuning）



- 将数据集分为三个部分:**训练集**、**验证集（校验集）**和**最终用于评估模型的测试集**

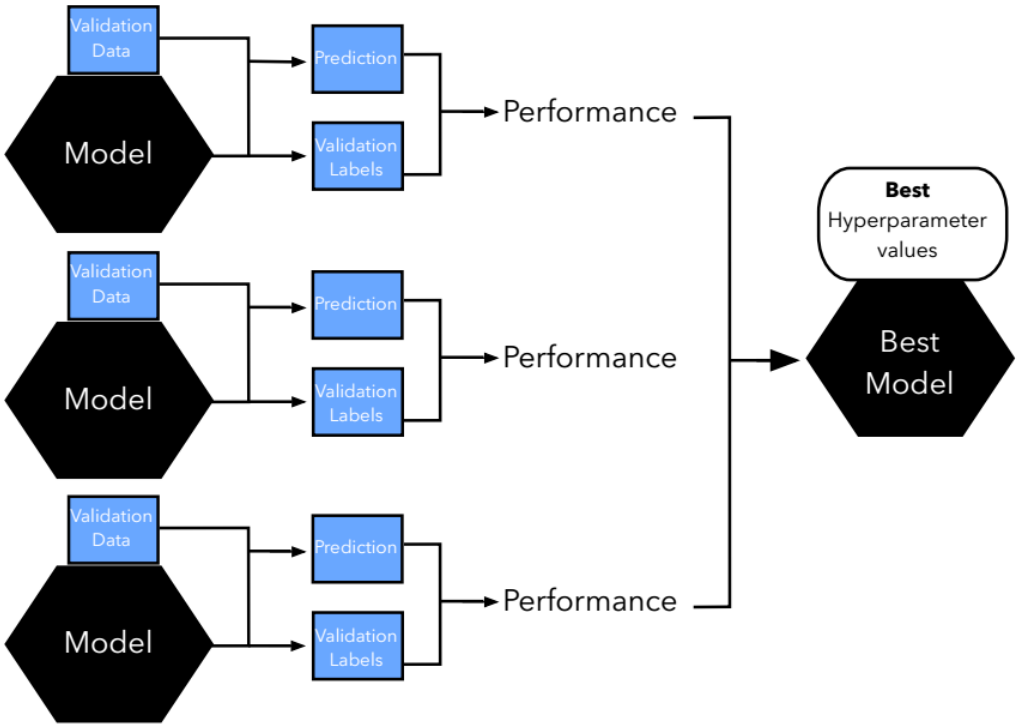
1.2.1 留出法——三向留出法Three-Way Holdout Method（超参数调优Hyperparameter Tuning）



- 超参数调优阶段，使用**不同超参数**设置的学习算法来拟合模型到训练数据

1.2.1 留出法——三向留出法Three-Way Holdout Method（超参数调优Hyperparameter Tuning）

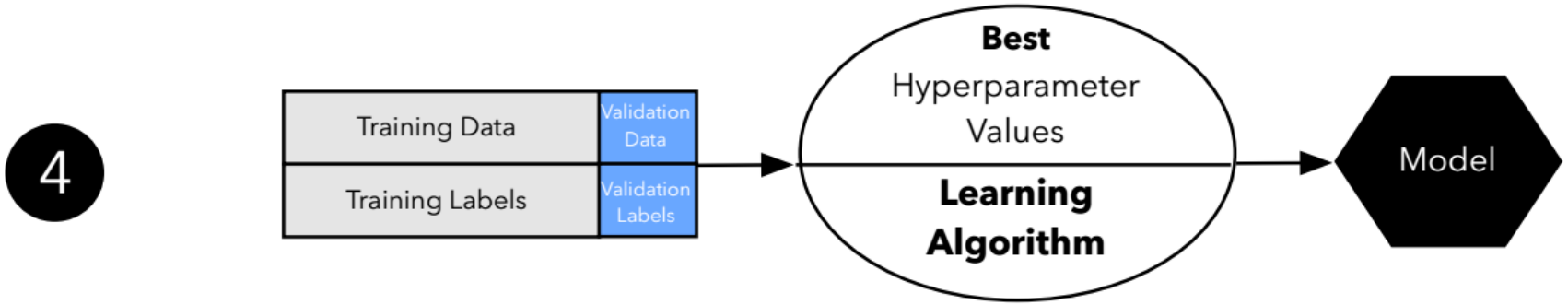
3



- 一般合并2和3步
- 在继续下一个模型之前，先计算当前模型的性能
- 以避免将所有拟合的模型都保存在内存中

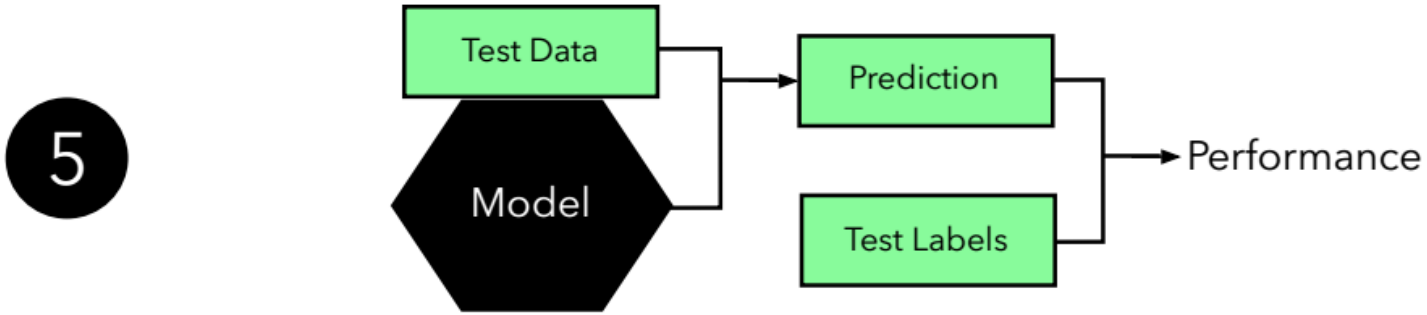
- 在验证集上评估模型的性能,选择与**最佳性能**相关的超参数设置

1.2.1 留出法——三向留出法Three-Way Holdout Method（超参数调优Hyperparameter Tuning）



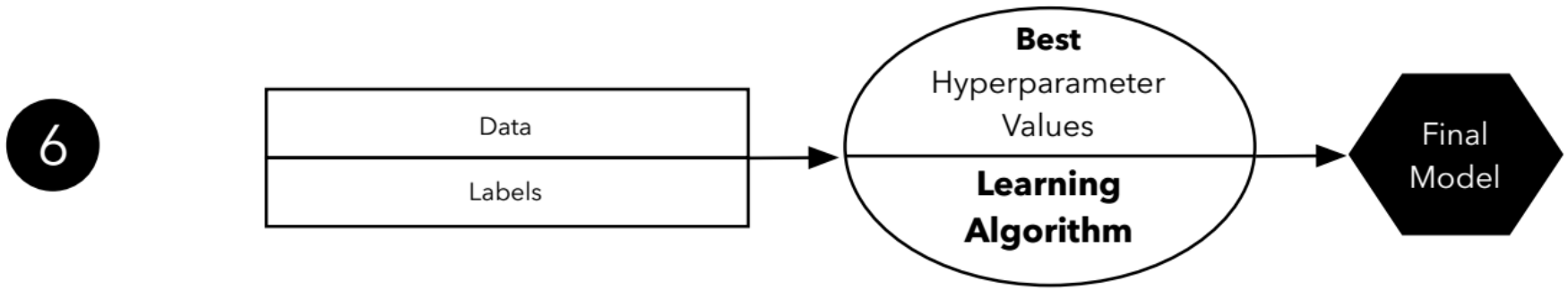
- 在选择模型之后**合并训练集和验证集**，并使用最好的超参数设置，拟合模型到这个更大的数据集

1.2.1 留出法——三向留出法Three-Way Holdout Method（超参数调优Hyperparameter Tuning）



- 使用**独立的测试集**来估计模型的泛化性能

1.2.1 留出法——三向留出法Three-Way Holdout Method（超参数调优Hyperparameter Tuning）



- 利用**所有的数据**（合并训练和测试集）拟合模型（可选步骤）

1.2.1 留出法——重复留出法Repeated Holdout

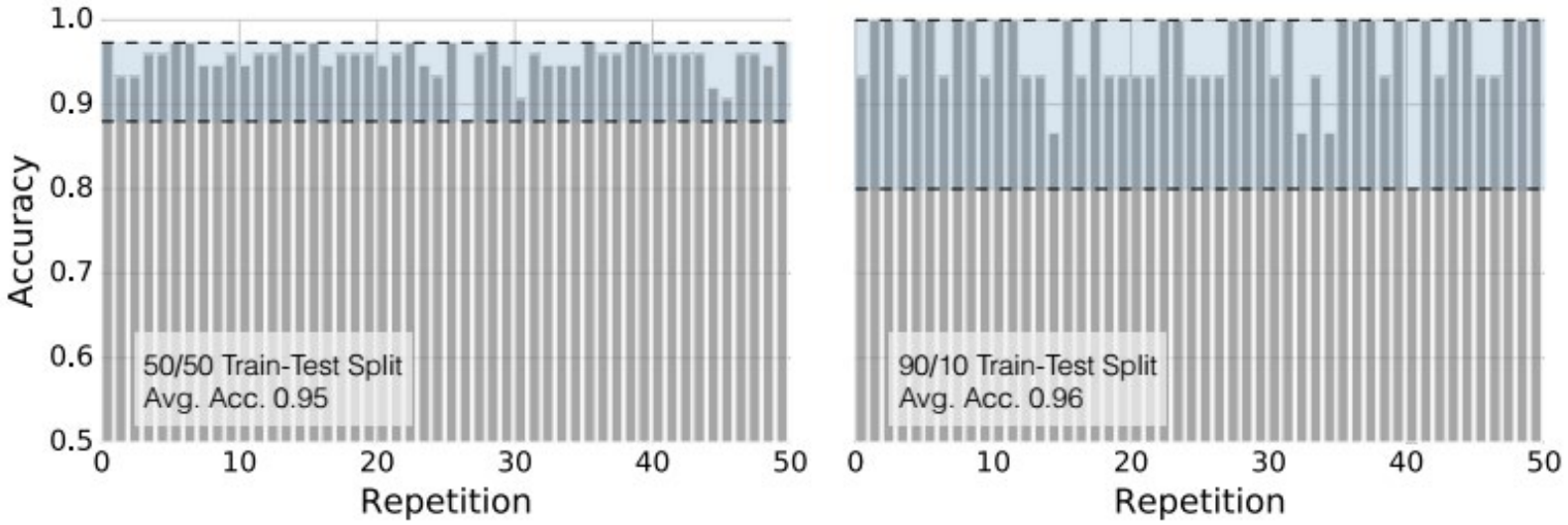
- 用不同的随机种子重复留出法k次，并计算这些k次重复的平均性能

$$\mathrm{ACC}_{avg} = \frac{1}{k}\sum_{j=1}^{k} ACC_j$$

- 其中，$\mathrm{ACC}_j = 1 - \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i)$

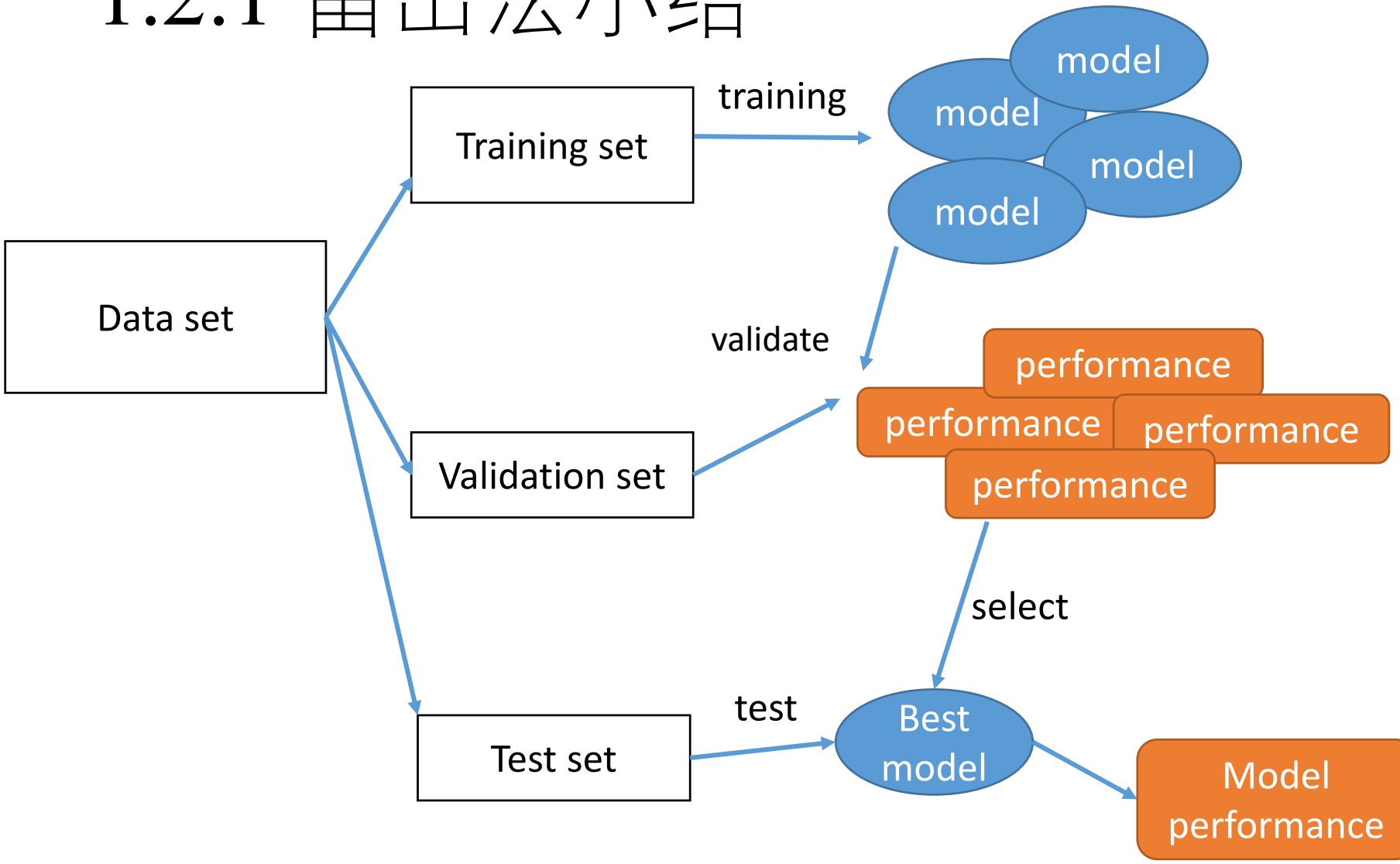
- 表示在大小为m的第j个测试集上的精度估计

1.2.1 留出法——重复留出法Repeated Holdout



- 随着**测试集减小**，估计的**方差增加**（蓝色区域）
- **缩小训练集**的大小，测试集**误差**小幅度**增加**

1.2.1 留出法小结


Data set
Training set
training
model
model
model
model
validate
Validation set
performance
performance
performance
performance
select
Test set
test
Best model
Model performance