

1.4 机器学习算法的组成部分

CSDN学院

► 机器学习任务的一般步骤

- 确定特征
 - 可能是**最重要的**步骤! (收集训练数据)
- 确定模型
 - 目标函数/决策边界形状
- 模型训练：根据训练数据估计模型参数
 - 优化计算
- 模型评估：在校验集上评估模型预测性能
- 模型应用/预测



Deep Learning可学习特征 (对视觉/语音等非结构化数据尤其重要)

- 监督学习任务：给定带标签的训练样本 $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ ，学习到一个 $\mathbf{x} \rightarrow y$ 的映射 f ，从而对新输入的 \mathbf{x} 进行预测
- **模型**：对给定的输入 \mathbf{x} ，如何预测其标签 \hat{y}
 - 不同模型对数据的假设不同
 - 最简单的模型：**线性模型** $f(\mathbf{x}) = \sum_j w_j x_j = \mathbf{w}^T \mathbf{x}$
- 确定模型类别后，模型训练转化为求解模型**参数**
 - 如对线性模型参数为 $\boldsymbol{\theta} = \{w_j | j = 1, \dots, D\}$ ，其中 D 为特征维数
- 求解模型参数：目标函数最小化

► 非线性模型

- 线性模型非线性化

- 基函数： x^2 、 \log 、 \exp 、样条函数、决策树...
- 核化：将原问题转化为对偶问题，将对偶问题中的向量点积 $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ 换成核函数 $k(\mathbf{x}_i, \mathbf{x}_j)$

► 目标函数

- 目标函数通常包含两项：**损失函数**和**正则项**

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N L(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) + R(\boldsymbol{\theta})$$

参数 $\boldsymbol{\theta}$ 对应的损失函数
度量参数对应的模型与
训练数据的拟合程度

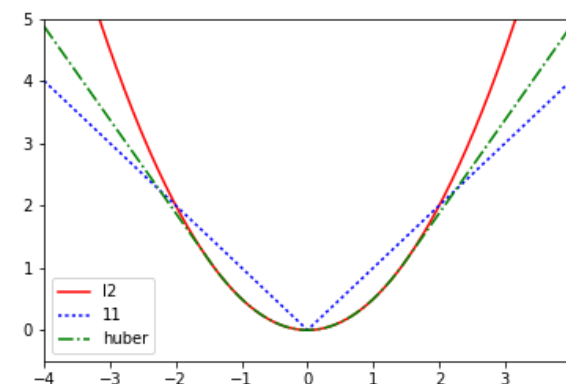
参数 $\boldsymbol{\theta}$ 对应的正则项
对模型的复杂度施加惩罚

► 损失函数—回归

- 损失函数：度量模型预测值与真值之间的差异
- 对回归问题：令残差 $r = f(\mathbf{x}) - y$
 - L2损失： $L_2(r) = \frac{1}{2}r^2$

对噪声不敏感

- L1损失： $L_1(r) = |r|$
- Huber损失： $L_\delta(r) = \begin{cases} \frac{1}{2}r^2 & \text{if } |r| \leq \delta \\ \delta|r| - \frac{1}{2}\delta^2 & \text{if } |r| > \delta \end{cases}$



► 损失函数——分类

- 损失函数：度量模型预测值与真值之间的差异

- 对分类问题

- 0-1损失: $l_{0/1}(y, f(x)) = \begin{cases} 1 & yf(x) < 0 \\ 0 & \text{otherwise} \end{cases}$

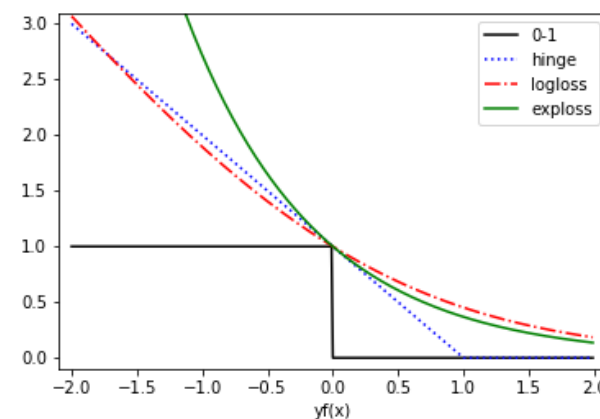
- Logistic损失: 亦称负log似然损失/logloss

- $l_{\log}(y, f(x)) = \log(1 + \exp(-yf(x)))$

- 指数损失: $l_{\exp}(y, f(x)) = \exp(-yf(x))$

- 合页损失: $l_{\text{hinge}}(y, f(x)) = \max(0, 1 - yf(x))$

- ...



- 不能只选择损失最小的模型，因为复杂的模型可能和训练数据拟合得非常好，在训练集上可以损失几乎为0
- 但我们真正关心的是测试数据上的性能
 - 测试数据和训练数据通常假设是来自同分布的独立样本，但只是分布相同，随机变量取值会变化
 - 训练数据中可能会有噪声，模型不应该将噪声包含在内
- 复杂模型（预测）不稳定：方差大
- 所以需要控制模型复杂度
 - 正则项：对复杂模型施加惩罚

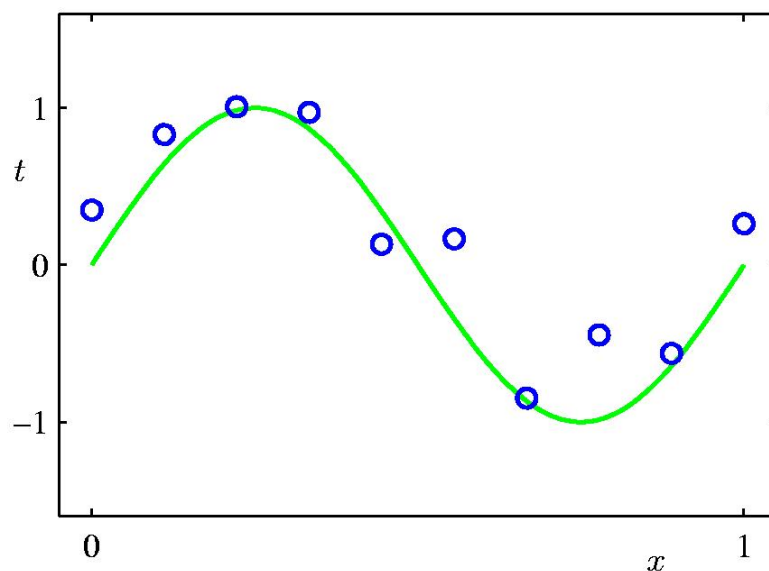
► 正则项的必要性

- 例：sin曲线多项式拟合

► 例：sin曲线拟合

$$Y = \sin(2\pi X) + \varepsilon$$

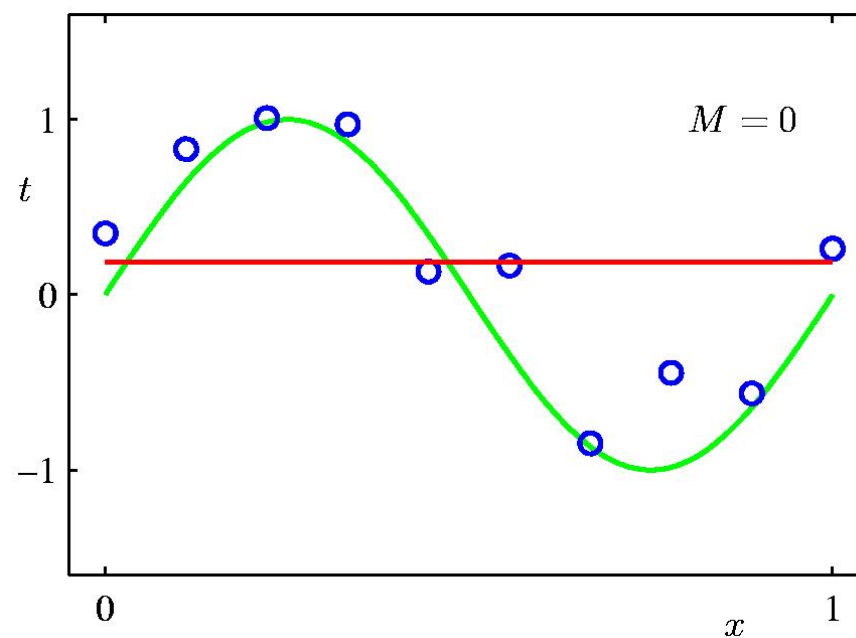
$$X \sim \text{Uniform}[0,1], \varepsilon \sim N(0, 0.3^2)$$



样本数 $N = 10$
用 M 阶多项式拟合：

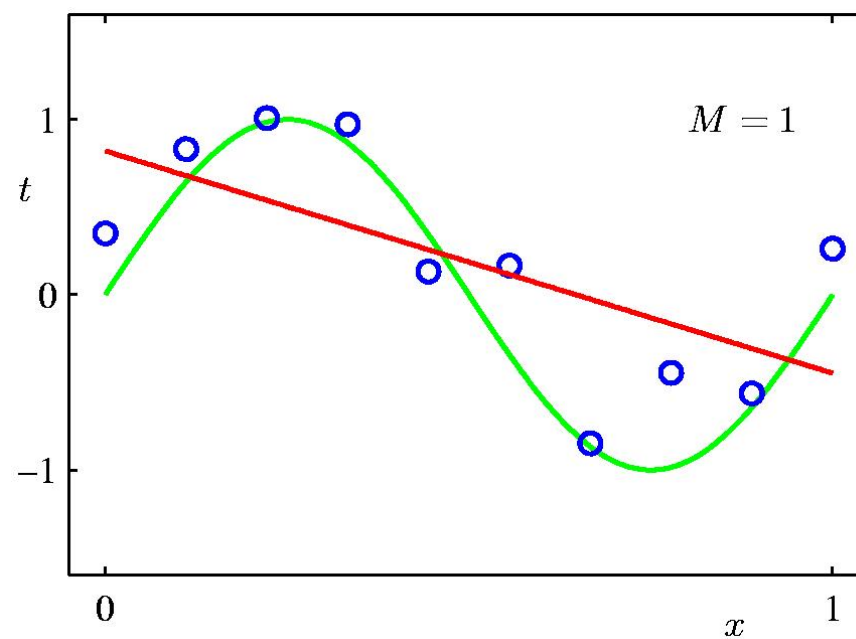
$$\hat{y} = \sum_{j=0}^M w_j x^j$$

► 例：sin曲线拟合（2）



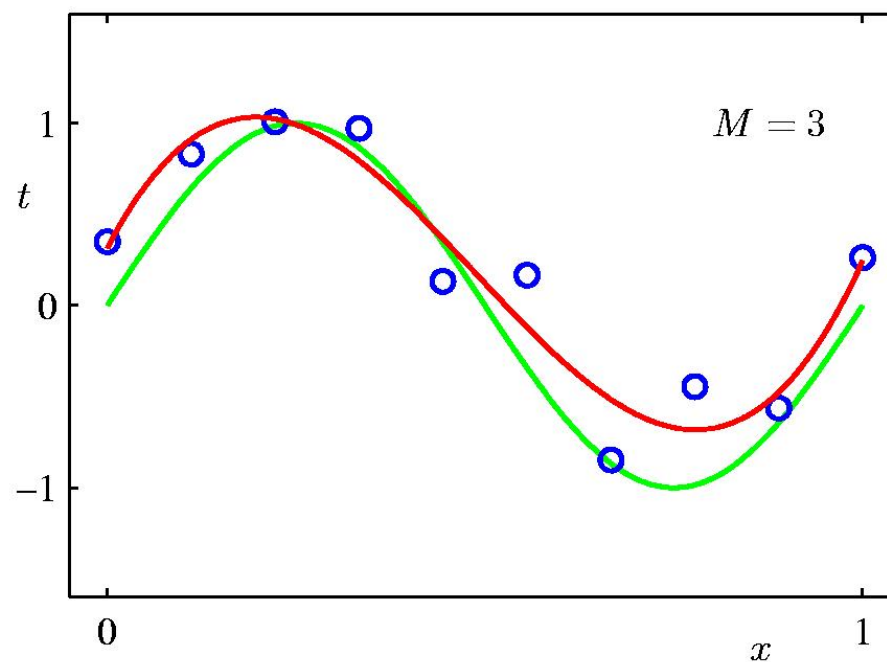
0阶多项式拟合

► 例：sin曲线拟合（3）



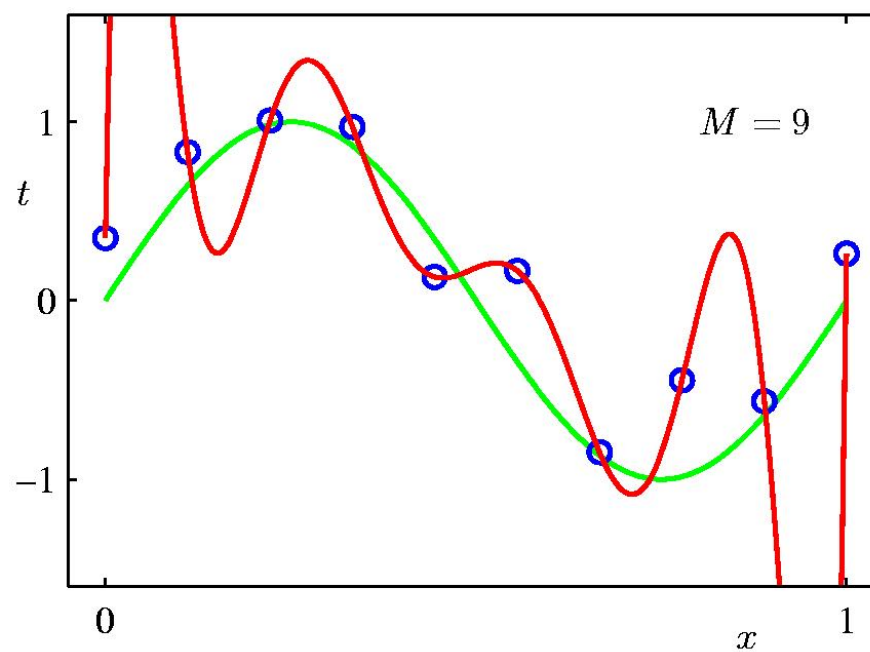
1阶多项式拟合

► 例：sin曲线拟合（4）



3阶多项式拟合

► 例：sin曲线拟合（5）

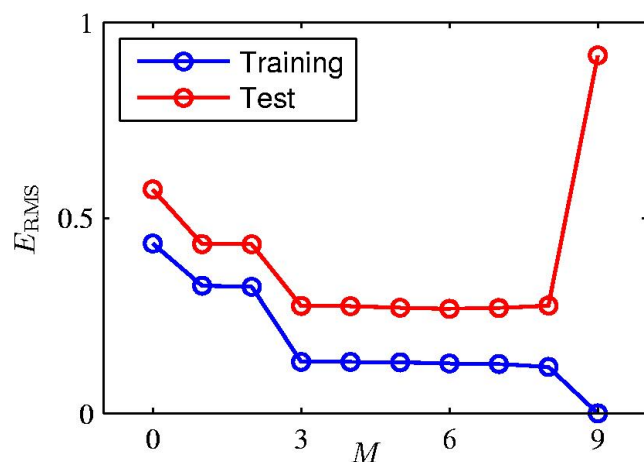


9阶多项式拟合

► 例：sin曲线拟合（6）

- 过拟合：

- 当模型复杂度增加时，训练误差继续下降，甚至趋近于0，而测试误差反而增大



$$E_{RMS} = \sqrt{\frac{1}{N} \sum_{i=0}^N (\hat{y}_i - y_i)^2}$$

► 训练集上的误差 \neq 测试集上的误差

- 推广性(**generalization**)：学习器在新的测试数据上表现
- 复杂的曲线不能泛化/推广到新数据上：根据特定输入调制得
太好，而不是真正建模 x 与 y 之间的关系
 - 被称为数据过拟合(**overfitting**)

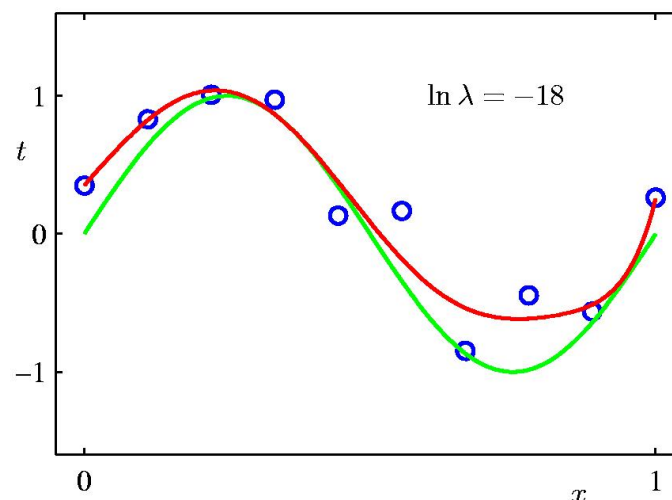
► 例：sin曲线拟合（7）

• 回归系数：

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

► 例：sin曲线拟合（8）

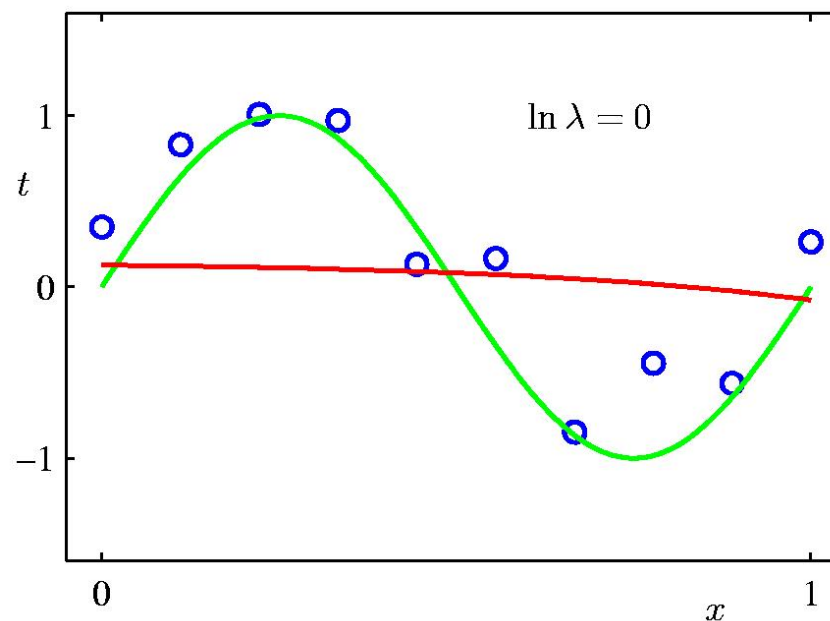
- 增加L2正则



岭回归：最小化 $RSS_{ridge}(\lambda) = \sum_{i=0}^N \left(\sum_{j=0}^M w_j x_i^j - y_i \right)^2 + \lambda \sum_{j=0}^M w_j^2$

► 例：sin曲线拟合（9）

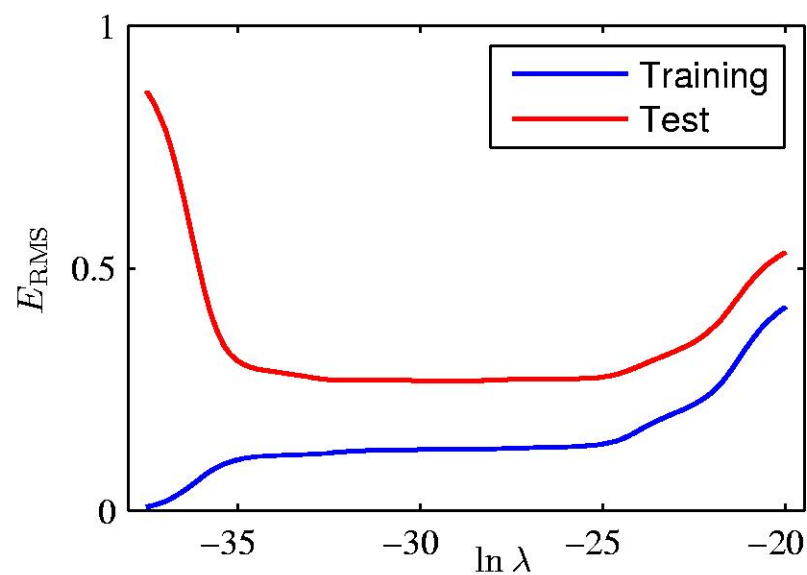
- 欠拟合：模型太简单 / 对复杂性惩罚太多



岭回归

► 例：sin曲线拟合（10）

- 不同正则参数下的训练误差vs.测试误差



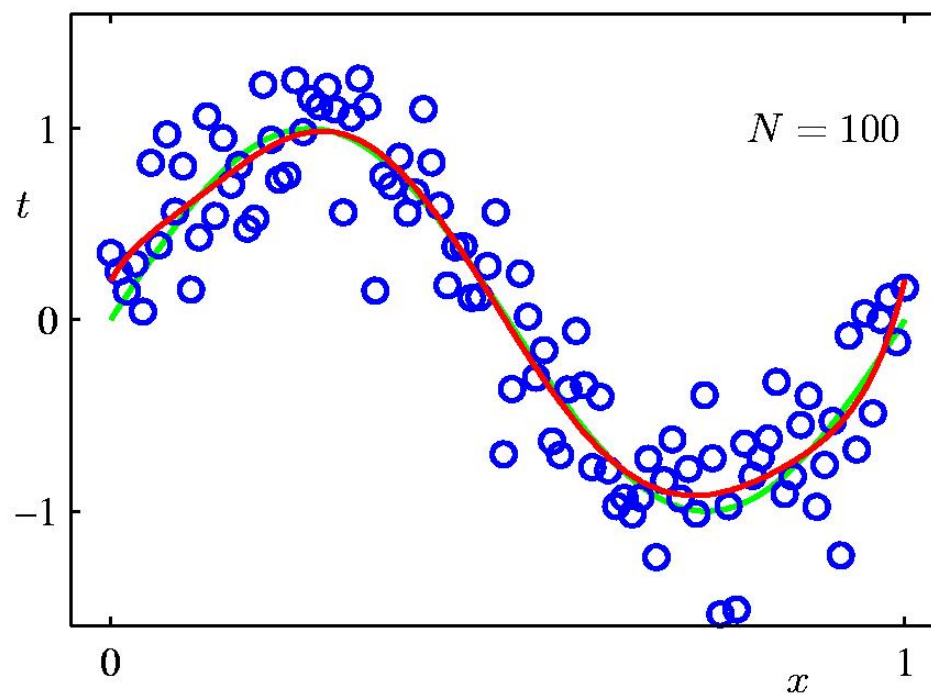
岭回归

► 例：sin曲线拟合（11）

• 岭回归系数

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

例：sin曲线拟合（12）



样本数目增多时，
可以考虑更复杂的
模型

9阶多项式拟合（不带正则），训练样本数 $N=100$

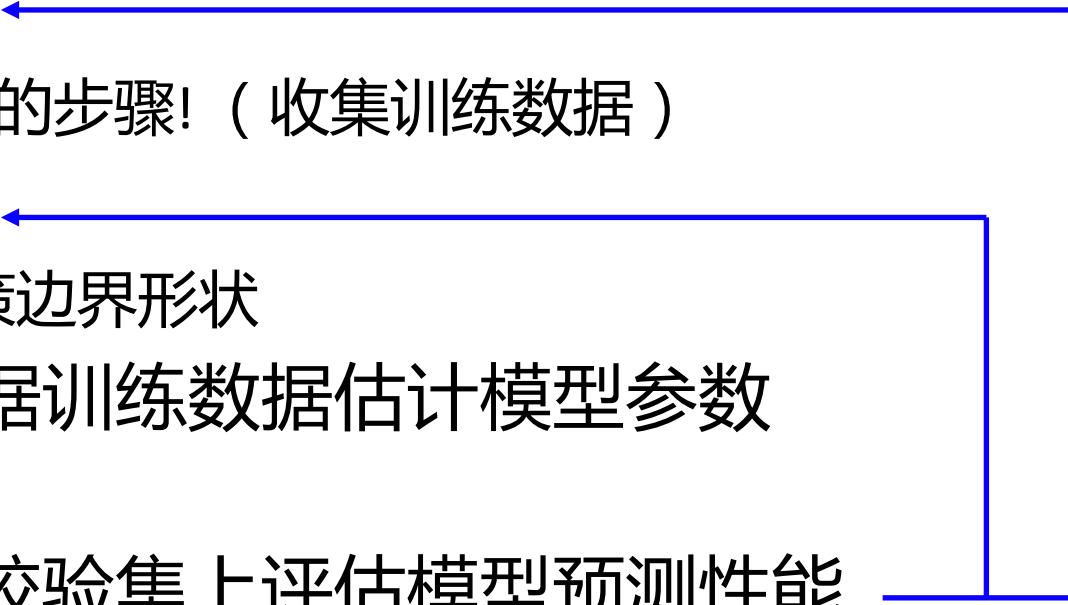
► 常用正则函数

- L2正则： $R(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_2^2 = \lambda \sum_{j=1}^D \theta_j^2$
- L1正则： $R(\boldsymbol{\theta}) = \lambda |\boldsymbol{\theta}| = \lambda \sum_{j=1}^D |\theta_j|$
- L0正则： $R(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_0$
 - 非0参数的数目
 - 不好优化，通常用L1正则近似

► 常见线性模型的损失和正则项组合

	L2损失	L1损失	Huber损失	Logistic损失	合叶损失	ϵ -insentive损失
L2正则	岭回归			L2正则 Logistic回归	SVM	SVR
L1正则	LASSO			L1正则 Logistic回归		
L2+L1正则	Elastic net					

► 机器学习任务的一般步骤

- 确定特征
 - 可能是最重要的步骤! (收集训练数据)
 - 确定模型
 - 目标函数/决策边界形状
 - **模型训练**：根据训练数据估计模型参数
 - 优化计算
 - 模型评估：在校验集上评估模型预测性能
 - 模型应用/预测
- 

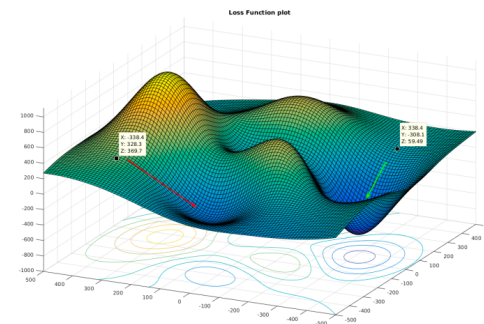


Deep Learning可学习特征 (对视觉/语音等非结构化数据尤其重要)

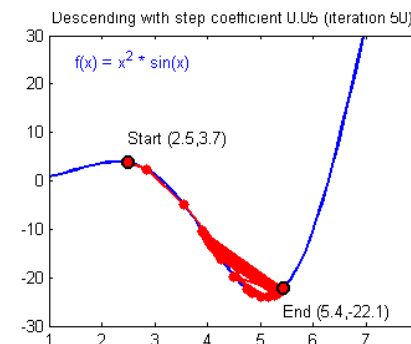
- 在训练数据上求目标函数极小值：优化
- 简单目标函数直接求解
 - 如小数据集上的线性回归
- 更复杂问题：凸优化
 - （随机）梯度下降
 - 牛顿法 / 拟牛顿法
 - ...

► 梯度下降 (Gradient Descent) 算法

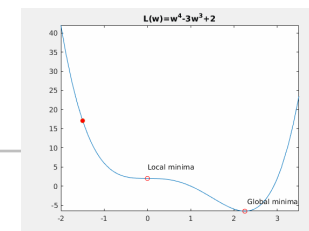
- 梯度下降 / 最速下降算法：快速寻找函数局部极小值
 - 将函数比作一座山，我们站在某个山坡上，往四周看，从哪个方向向下走一小步，能够下降的最快。这个方向就是梯度的方向
- 梯度下降算法：求函数 $J(\theta)$ 的最小值
 - 给定初始值 θ^0
 - 更新 θ ，使得 $J(\theta)$ 越来越小
 - $\theta^t = \theta^{t-1} - \eta \nabla_{\theta} J(\theta)$ （ η ：学习率 ）
 - 直到收敛到 / 达到预先设定的最大迭代次数



► 梯度下降算法 (cont.)



- 下降的步伐大小（学习率）非常重要：如果太小，收敛速度慢；如果太大，可能会出现overshoot the minimum的现象
 - 如果学习率取值后发现函数值增长了，则需要减小学习率的值
- 梯度下降求得的只是局部最小值
 - 二阶导数 >0 ，则目标函数为凸函数，局部极小值即为全局最小值
 - 随机选择多个初始值，得到函数的多个局部极小值点。多个局部极小值点的最小值为函数的全局最小值。



► 随机梯度下降

- 梯度下降算法每次学习都使用整个训练集，这样对大的训练数据集合，每次学习时间过长，对大的训练集需要消耗大量的内存。此时可采用随机梯度下降(Stochastic gradient descent, SGD),每次从训练集中随机选择一部分样本进行学习。
- 更多（随机）梯度下降算法的改进版
 - 动量(Momentum)
 - Nesterov accelerated gradient([NAG](#))
 - Adagrad
 - RMSprop
 - Adaptive Moment Estimation(Adam)...

► 模型选择与模型评估

- 同一个问题有不同的解决方案
 - 如线性回归 vs. 决策树
- 哪个更好？模型评估与模型选择
 - 在**新数据点**的预测误差最小
- 模型评估：已经选定最终的模型，估计它在新数据上的预测误差
- 模型选择：估计不同模型的性能，选出最好的模型

► 模型评估

[sklearn.model_selection.train_test_split](#)

- 当样本足够多时，可以将训练数据分成一部分数据做校验
 - 训练集：估计模型的参数
 - 校验集：估计模型的预测误差（模型评估）

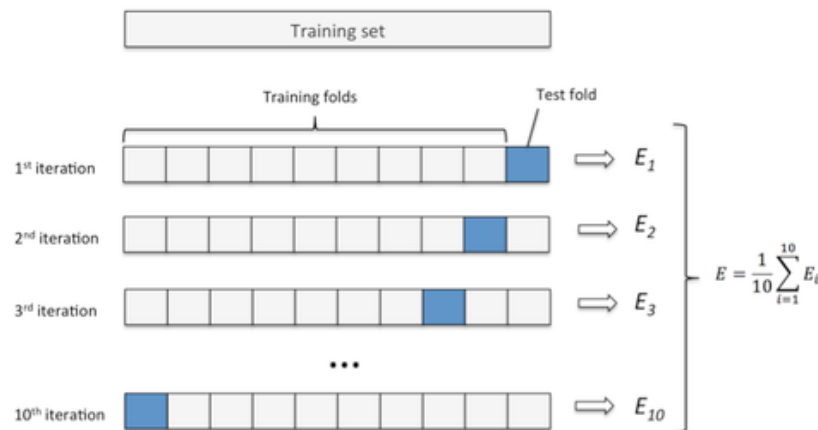


- 通常没有足够多样本，而且也很难说明多少数量的样本是足够的
 - 可通过[重采样技术](#)来模拟[校验集](#)：[交叉验证](#)和[bootstrap](#)是重采样技术的两个代表

► K-折交叉验证

[sklearn.model_selection.KFold](#)
[sklearn.model_selection.GridSearchCV](#)

- 交叉验证 (Cross Validation, CV) : 将训练数据分成容量大致相等的 K 份 (通常 $K=5/10$)



对每个 $k = 1, 2, \dots, K$,
留出第 k 份数据 , 其余 $K-1$ 份数据用于训练 ,
在留的出的第 k 份数据上计算的预测误差 $E_k(M)$

- 交叉验证估计的误差为 : $CV(M) = \frac{1}{K} \sum_{k=1}^K E_k(M)$

http://sklearn.lzjqsdd.com/modules/cross_validation.html

► 例：交叉验证

- `sklearn.model_selection.KFold(n_splits=3, shuffle=False, random_state=None)`

例：在 6 个样例的数据集上使用 3-fold 交叉验证

```
from sklearn.model_selection import Kfold
```

```
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [5, 6], [7,8]])
```

```
y = np.array([1, 2, 3, 4, 5, 6])
```

```
kf = KFold(n_splits=3)
```

```
for train_index, test_index in kf.split(X):
```

```
    print("TRAIN:", train_index, "TEST:", test_index)
```

```
    X_train, X_test = X[train_index], X[test_index]
```

```
    y_train, y_test = y[train_index], y[test_index]
```

可以通过使用 numpy 的索引创建训练/测试集合

输出：每个折叠由两个数组组成：

第一个为 *training set*

另一个作为 *test set*

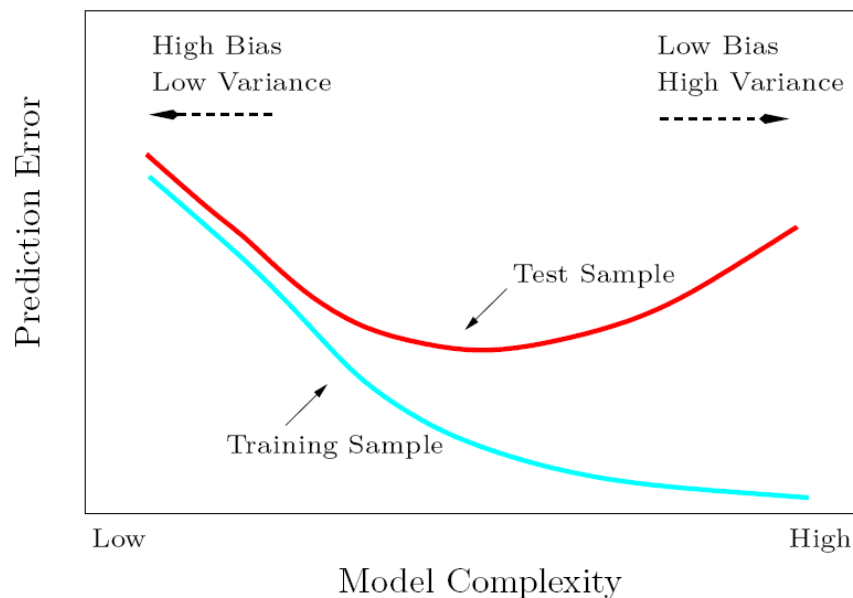
('TRAIN:', array([2, 3, 4, 5]), 'TEST:', array([0, 1]))

('TRAIN:', array([0, 1, 4, 5]), 'TEST:', array([2, 3]))

('TRAIN:', array([0, 1, 2, 3]), 'TEST:', array([4, 5]))

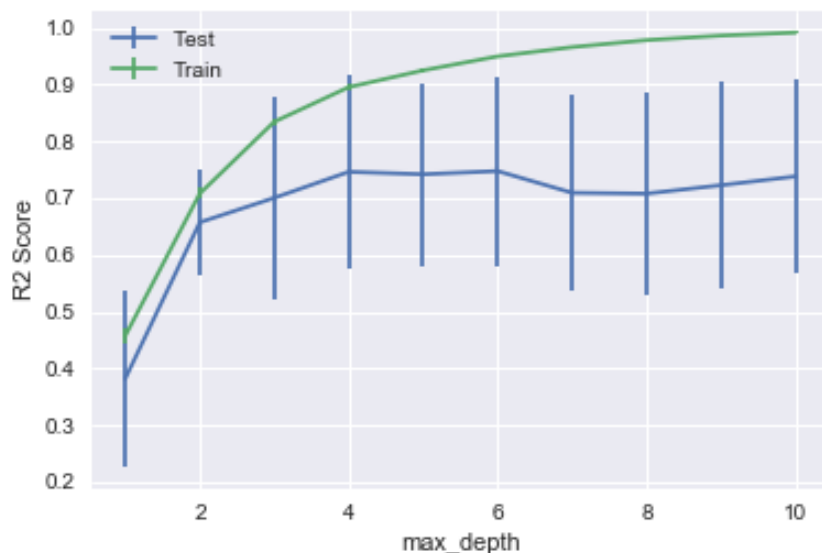
► 模型选择

- 模型选择：对多个不同的模型，计算其对应的误差 $CV(M)$ ，最佳模型为 $CV(M)$ 最小的模型。
- 模型复杂度和泛化误差的关系通常是U形曲线：



► 例：波士顿房价预测

- 评价指标：R2 Score（越大越好）
- 模型：决策树
 - 树的最大深度（max_depth）：1 ~ 10



训练集性能：深度越大，性能越好

测试集性能：

最佳参数：max_depth=6

最佳结果：mean: **0.74767**, std: 0.16583

► 小结：机器学习任务的一般步骤

- 确定特征
 - 可能是**最重要的**步骤! (收集训练数据)
 - 确定模型
 - 目标函数/决策边界形状
 - 模型训练：根据训练数据估计模型参数
 - 优化计算
 - 模型评估：在校验集上评估模型预测性能
- 