

—武大本本科生课程



第10讲 模型评估与选择

(Lecture 10 Model Evaluation and Selection)

武汉大学计算机学院机器学习课程组

2021.06

第8章 模型评估与选择

内容目录

8.1 模型选择的基本概念

8.2 模型评估方法

8.3 偏差-方差

8.4 模型性能度量

8.5 超参数的搜索

8.6 模型评估与选择实例(基于华为云ModelArts AI平台)
小结

8.1 模型选择的基本概念

1. 机器学习三要素

一个具体的机器学习方法由模型、策略和算法构成，称为统计学习方法的三要素：

$$\text{方法} = \text{模型} + \text{策略} + \text{算法}$$

(1) 模型：在监督学习过程中，模型就是所要学习的**条件概率分布或分类决策函数**

- **模型的假设空间**：所有可能的条件概率分布或决策函数。
 - 例如，如果决策函数是输入变量的线性函数，那么模型的假设空间就是所有线性函数构成的函数空间。这里“假设”是指一个具体的模型，在其函数形式确定后就对应于模型参数，因此模型的假设空间也称为模型的参数空间。
- 即使对模型的参数个数及其取值范围进行限制，模型空间中可能的模型个数仍是无穷多的。某种意义上机器学习就是：**在设定一定的优化目标函数条件下，在给定的训练数据约束下，在模型空间中有效地进行最优参数搜索。**

(2) 策略：判断模型拟合优度的准则

有了模型的结构形式或函数形式，还需要确定优化问题的目标函数 (即：如何确定模型参数的最优性)，以此指导最优模型参数的求取。

- **损失函数**：用来度量搜索到的模型在一个具体训练数据集上的最优性 (最优性与损失是简单的反向关系：**损失函数值越小，模型就越优**。——用损失大小反向表示模型最优性，是惯例)
- **风险函数**：用来度量在符合某种分布的训练数据集下，拟合模型的平均损失。

在假设空间 \mathcal{F} 中选取模型 f 作为决策函数，对于给定的输入 X ，由 $f(X)$ 给出相应的输出 Y ，**输出的预测值 $f(X)$ 与真实输出值 Y** 可能一致也可能不一致，用一个损失函数来度量预测 (分类) 错误的程度。**损失函数**是 $f(X)$ 和 Y 非负实值函数，**记作 $L(Y, f(X))$** 。

损失函数度量模型一次预测的好坏，风险函数度量平均意义下模型的好坏。

统计学习常用的损失函数有：

- 0-1损失函数 (0-1 loss function)

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases}$$

- 平方损失函数 (quadratic loss function)

$$L(Y, f(X)) = (Y - f(X))^2$$

- 绝对损失函数 (absolute loss function)

$$L(Y, f(X)) = |Y - f(X)|$$

- 对数损失函数 (logarithmic loss function)

$$L(Y, P(Y | X)) = -\log P(Y | X)$$

由于模型的训练数据本质上是来自某个总体的随机样本，因此，为更全面地衡量模型的最优性，需要对模型在同一总体下大量随机样本上的平均损失进行度量、比较。

- **模型 $f(X)$ 的期望风险 (Expected Risk)** 是模型的损失函数 $L(Y, f(X))$ 关于联合分布 $P(X, Y)$ 的期望：

$$\begin{aligned} R_{\text{exp}}(f) &= E_P[L(Y, f(X))] \\ &= \int_{\mathcal{X} \times \mathcal{Y}} L(y, f(x)) P(x, y) dx dy \end{aligned}$$

- **理论上，学习的优化策略是：搜索具有最小期望风险的模型。**
 - 由于联合分布 $P(X, Y)$ 是未知的， $R_{\text{exp}}(f)$ 不能直接计算。实际上，如果知道联合分布 $P(X, Y)$ ，可以从联合分布直接求出条件概率分布 $P(Y|X)$ ，也就不需要学习了。
- 联合分布未知，只能借助**大容量样本**和**大数定理**，利用最优计算技术，对机器学习问题进行求解。实践中一般做法是
 - 根据设定的优化目标，通过调整算法控制参数，对部分训练数据进行模型拟合；再根据在选定的模型性能指标上各模型在**验证集**上的表现，选择一个“**最优模型**”，然后用“最优模型”对应的算法超参，在一个稍大的训练集上重新进行模型拟合，在另一个没有参与模型拟合的测试集上评估该模型的泛化能力，与模型一起交付。

给定一个样本集

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

- 模型 $f(X)$ 关于样本集的**经验风险** (*Empirical Risk*), 记作 $R_{emp}(X, Y)$, 定义为

$$R_{emp}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$$

- **实践中，学习的优化策略：** 搜索具有**最小经验风险**的模型。
 - 期望风险 $R_{exp}(X, Y)$ 是模型关于联合分布的期望损失，经验风险 $R_{emp}(X, Y)$ 是模型关于样本的平均损失。根据大数定律，当样本容量趋于无穷时，经验风险 $R_{emp}(X, Y)$ 趋近期望风险 $R_{exp}(X, Y)$ 。所以一个很自然的想法是，用经验风险作为期望风险的近似估计。
 - 但是，由于**现实中样本数量有限**，甚至很少，用经验风险估计期望风险常常不理想，要对经验风险进行一定的矫正。这就关系到**监督学习的两个基本策略：经验风险最小化和结构风险最小化**。
 - 说明：机器学习用样本集一般被划分为起不同作用的子集，对应的经验风险也就有不同的名称：在拟合模型的训练集上经验风险称为**训练误差**，在验证模型的验证集上经验风险称为**验证误差**，在测试集上经验风险则为**测试误差** (**泛化误差**的估计)。

(A) 经验风险最小化策略:

经验风险最小化就是在训练样本上求解以下最优化问题:

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$$

- 当训练用样本容量足够大时, 经验风险最小化能保证有较好的学习效果, 在现实中被广泛采用。比如, 极大似然估计就是一种经验风险最小化方法。
- 当训练用的样本容量很小时, 经验风险最小化的学习效果未必很好, 容易产生**过拟合** (over-fitting) 现象。
- **模型过拟合**是指模型完美拟合训练数据, 从而导致泛化能力下降的一类现象。

(B) 结构风险最小化的策略

为防止过拟合提出了所谓的结构风险最小化策略。

- **结构风险**是在经验风险上加上表示模型复杂度的正则项(regularizer)或惩罚项(penalty term)，定义是

$$R_{\text{srn}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f)$$

其中 $J(f)$ 为模型复杂度的某个度量，如模型参数个数、参数向量的某类范数等，是定义在假设空间 \mathcal{F} 上的一个泛函。

模型 f 越复杂，复杂度 $J(f)$ 就越大。模型 f 越简单，复杂度 $J(f)$ 就越小。这样上式就实现了对复杂模型的惩罚。 $\lambda \geq 0$ 是惩罚强度系数，用以权衡经验风险和模型复杂度 (如SVM的C参数、 k -NN中的 k 值)。

- **结构风险最小化**，就是求解最优化问题：

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f)$$

结构风险小的模型在训练数据和测试数据上都有较好表现。

(3) 算法：最优模型参数的求取及控制

- 根据学习策略（最优化的目标函数），基于训练数据集，从模型的假设空间中，以有效的计算方法求取并选择**最优模型**
 - 如果最优化问题有**显式的解析式**，求解算法比较简单（数值线性代数方法，或各种梯度求解方法）
 - 通常最优化问题的解析解不存在，需要用**复杂的约束优化方法**
- 机器学习问题最终归结为最优化问题，学习算法实际上就是最优化算法。如何保证找到全局最优解，并使求解的过程高效，就成为机器学习研究中的重要问题；可以利用已有的最优化算法，有时也需要开发针对性最优化算法。
- 机器学习方法之间的不同，主要来自模型类型的设定、优化目标的设定、求解算法的实现等方面的差异。确定了模型、目标、算法，机器学习方法也就确定了。

接下来介绍监督学习中模型评估与选择的几个重要概念。

2. 模型评估方法

可以通过模型在“测试集”上的“测试误差”作为泛化误差的近似来评估模型对新样本的预测能力，即模型的泛化能力。比较不同模型泛化能力的强弱，从而帮助我们进行模型选择。模型选择通常有以下方法：

- 留出法 (Hold-out)
- 交叉验证法 (Cross validation)
- 自助法 (Bootstrapping)

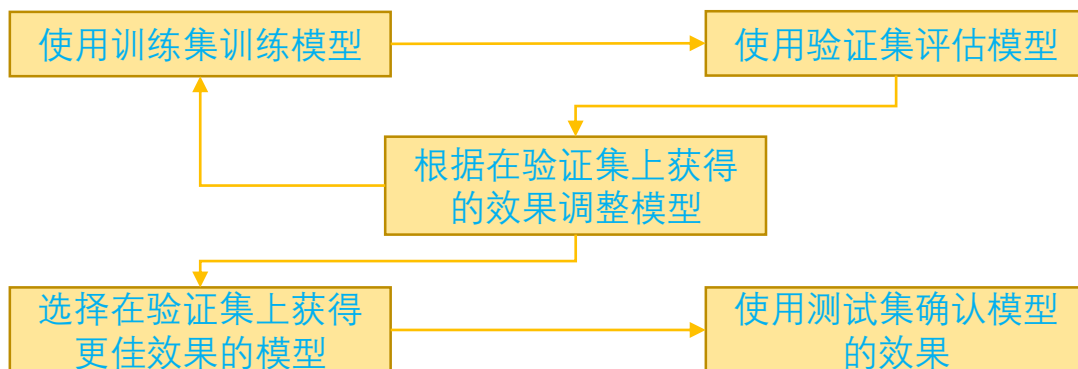
在模型评估方法中，要注意以下几点：

- 训练集 S 和测试集 T 组成数据集 D
- 假设测试样本是从真实分布中采样获得
- 测试集应与训练集互斥

数据集拆分：

在机器学习中，通常将所有的数据集划分为三：训练数据集，验证数据集和测试数据集。它们的功能分别为：

- 训练数据集 (train dataset)：用来构建机器学习模型。
- 验证数据集 (validation dataset)：辅助构建模型，用于构建过程中评估模型，为模型提供无偏估计，进而调整模型超参数。
- 测试数据集 (test dataset)：用来评估训练好的模型的性能。



留出法 (Hold-out)

留出法是一种模型评估方法，其通过将数据集 D 划分为两个互斥的集合，假设其中一个集合为训练集 S ，另一个为测试集 T ，则有：

$$D = S \cup T, S \cap T = \emptyset$$

在 S 上训练出模型后，用 T 来评估测试误差，留出法有以下几个需要注意的地方：

- 训练/测试集的划分要尽可能保持数据分布的一致，避免因数据划分过程中引入的额外偏差对最终结果产生影响（例如：分层采样, Stratified sampling）。
- 由于不同的划分方式会带来不同的训练/测试集，相应的模型评估结果也会存在差别，因此单次使用留出法的估计结果大多不可靠，通常需进行多次随机划分（例如：100次随机采样），重复进行实验评估后取平均值作为其评估结果。
- 留出法带来一个无法避免的矛盾：模型评估的动机是“评估数据集 D 训练出的模型”，但是留出法对数据集需划分为 S 和 T ，这会导致：（1）若 S 较大 T 较小，那么 S 训练出的模型与 D 训练的模型相似，但是 T 太少，评估结果偶然性大、不准确。2. 若 S 较小 T 较大，那么 S 与 D 训练出的模型差异较大， T 的评估失去意义。常用做法是，测试集不能太大、也不能太小（例如：1/5~1/3）。

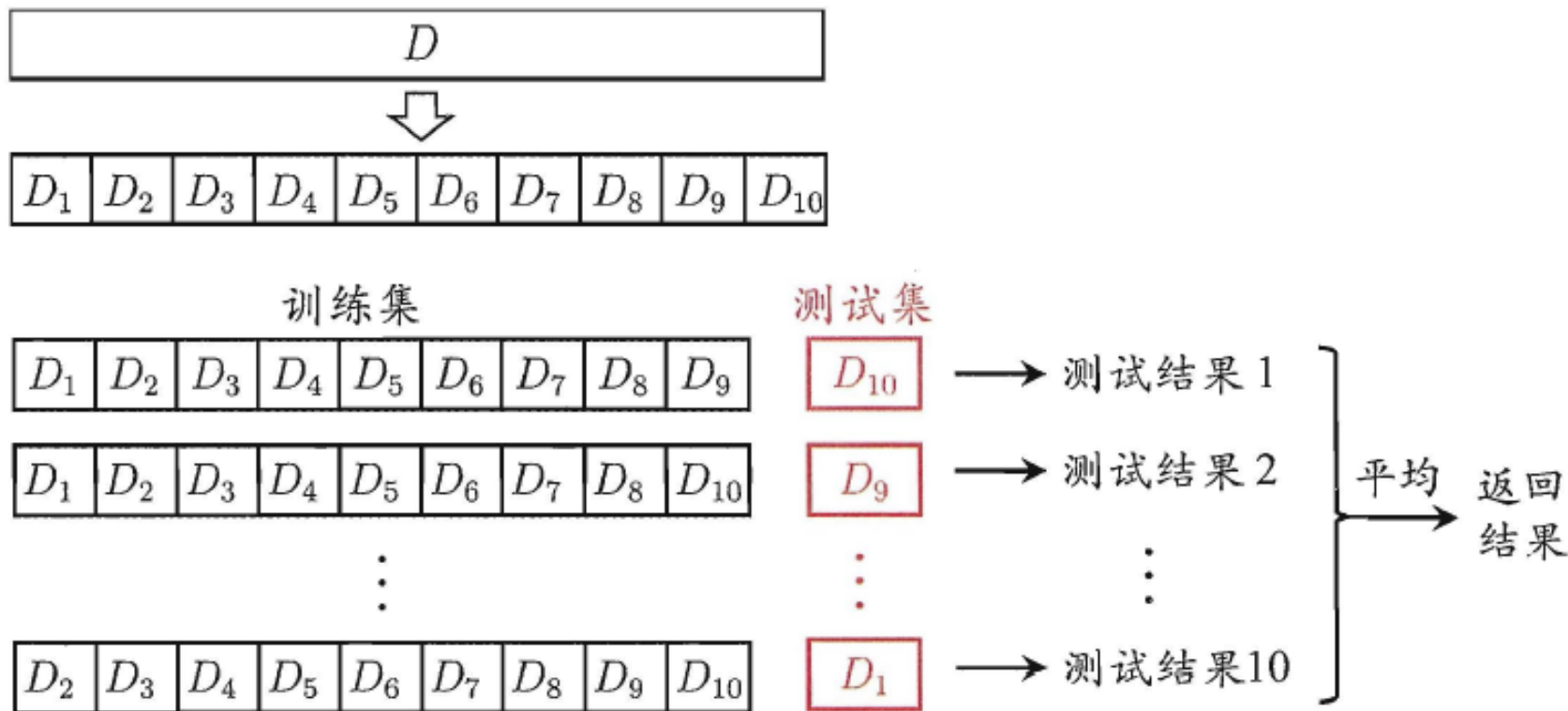
留出法模型评估实例：

- 假定D包含1000个样本，将其划分为S包含700个样本，T包含300个样本，用S进行训练后，如果模型在T上有90个样本分类错误。那么此时可以对模型进行评估，模型错误率为 $(90/300) \times 100\% = 30\%$ ，模型精度为： $1 - 30\% = 70\%$ 。
- 在该实例中进行分层采样：通过对D分层采样使S包含70%样本数，T包含30%的样本数。如果D中包含500个正例，500个反例，则分层采样得到的S应包含350个正例、350个反例；T应包含150个正例、150个反例。就是说，**分层采样是一种保留类别比例的采样方式以确保数据分布比例的平衡**。
- 在该实例中进行随机划分：进行100次样本随机划分，每次产生一个训练/测试集用于实验评估，取这100次评估结果的平均值作为模型评估结果。

交叉验证法 (Cross Validation)

K折交叉验证法 (k-fold cross validation): 将数据集划分为 K 个互不相交、数据分布尽可能一致且大小相同的子集, 利用 $K-1$ 个子集数据训练模型, 利用余下的一个子集测试模型 (一共有 K 种组合方式, 训练得到 K 个模型)。

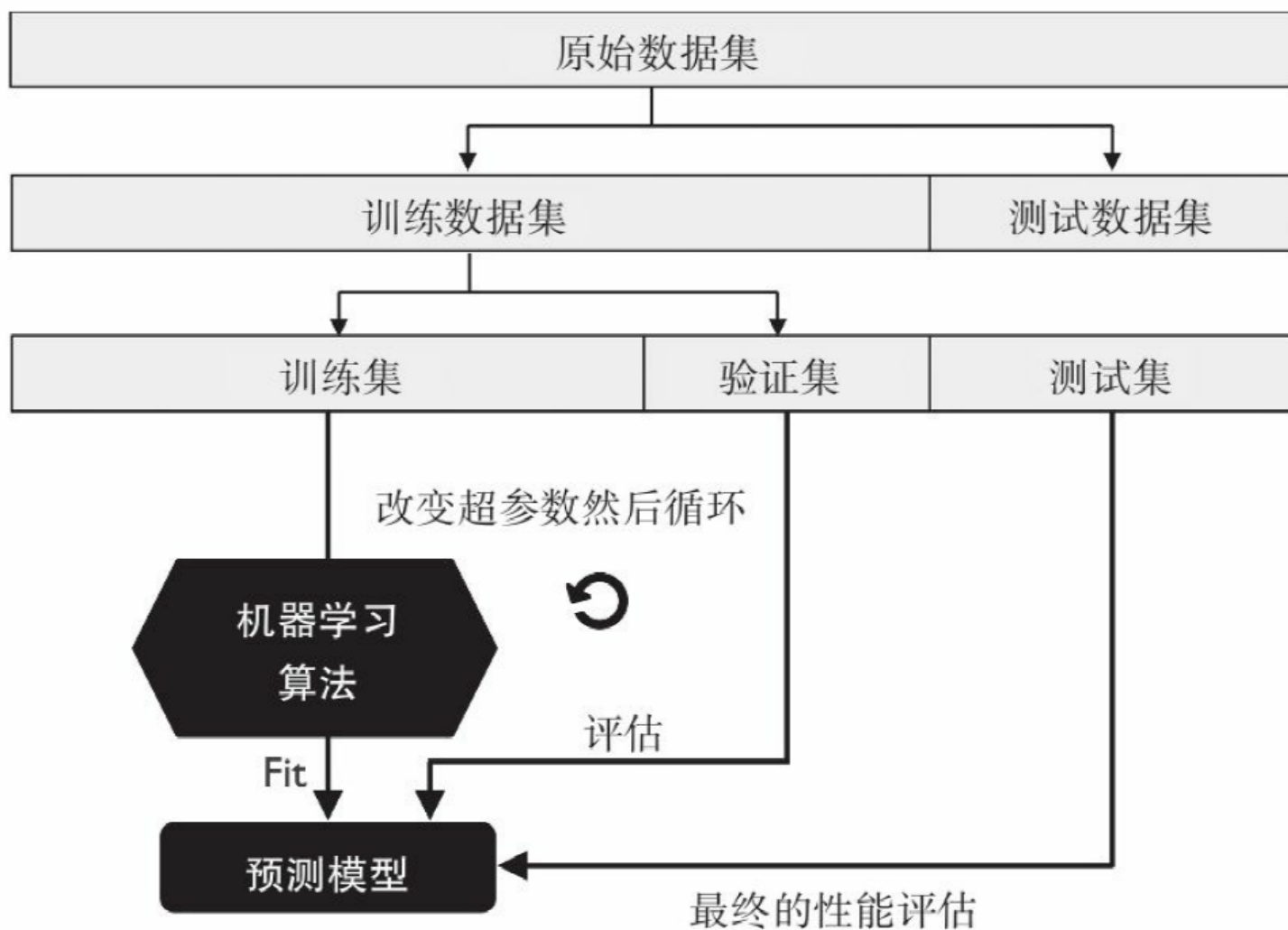
下图是10折交叉验证的一个示例:



10折交叉验证法数据划分示意图

交叉验证法 (Cross Validation)

- K 折交叉验证法将划分好的中的K种组合依次重复进行，获取测试误差的均值，将这个均值作为泛化误差的估计。由于是在K个独立的测试集上获得的模型表现平均情况，因此相比留出法的结果更有代表性。
- 与留出法类似，将数据集D划分为k个子集同样存在多种划分方式。为减小因样本划分不同而引入的差别，k折交叉验证通常要随机使用不同的划分重复p次，最终的评估结果是这p次k折交叉验证结果的均值。
- 假设数据集D中包含m样本，若令k=m，则得到了交叉验证法的一个特例：留一法 (Leave-One-Out CV，简称LOO CV)。该方法适用于数据集很小的情况下的交叉验证。
 - 优点：由于训练集与初始数据集相比仅仅少一个样本，因此留一法的训练数据最多，模型与全量数据得到的模型最接近。
 - 缺点：在数据集较大时，训练K个模型的计算量太大。每个模型只有1条测试数据，无法有效帮助参数调优。



交叉验证法流程示意图

K折交叉验证模型选择Python程序示例:

[Scikit-learn官网](#)提供了Cross-validation的相关文档及使用方法。

```
In [1]: ▶ import numpy as np
        from sklearn.model_selection import KFold
        X = ["a", "b", "c", "d"]
        kf = KFold(n_splits=2)
        for train, test in kf.split(X):
            print("%s %s" % (train, test))
```

```
[2 3] [0 1]
```

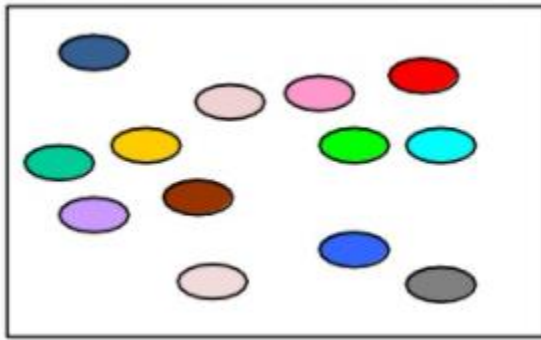
```
[0 1] [2 3]
```

自助法 (Bootstrapping)

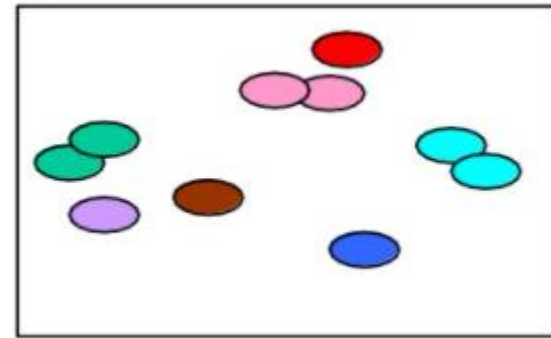
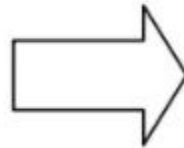
在留出法和 K 折交叉验证法中，由于保留了一部分样本用于测试，因此实际训练模型使用的训练集比初始数据集小 (虽然训练最终模型时会使用所有训练样本)，这必然会引入一些因为训练样本规模不同而导致的估计偏差。留一法受训练样本规模变化的影响较小，但是计算量太大，自助法是一个以自助采样法(Bootstrap sampling)为基础的一种较好解决方案。

自助采样法： 给定包含 N 个样本的数据集 D ，对它进行采样产生数据集 S ：

- 每次随机从 D 中挑选一个样本，将其拷贝放入 S 中，然后再将该样本放回初始数据集 D 中 (该样本下次采样时仍然可以被采到)。
- 重复该过程 N 次，就得到了包含 N 个样本的数据集 S 。



● 有放回采样



● 数据分布有所改变

自助法 (Bootstrapping)

在自助法中，显然D中有些样本会在S中多次出现，有些样本从不出现。D中某个样本始终不被采到的概率为 $(1-1/m)^m$ 。根据极限：

$$\lim_{m \rightarrow \infty} \left(1 - \frac{1}{m}\right)^m \mapsto \frac{1}{e} \approx 0.368$$

可知通过自助采样，初始数据集中约有 36.8% 的样本未出现在采样数据集S中。将S用作训练集，D-S用作测试集T，这样的测试结果称作**包外估计** out-of-bag estimate (**OOB**)。

由于平均有**63.2%**的样本点会被抽到S中，因此平均而言，D中有**36.8%**的样本作为验证集。



自助法在**数据集较小时**很有用。

- 优点：能从初始数据集中产生多个不同的训练集，这对集成学习等方法而言有很大好处。
- 缺点：产生的数据集改变了初始数据集的分布，这会引入估计偏差。因此在**初始数据量足够时**，**留出法和k折交叉验证法更常用**。

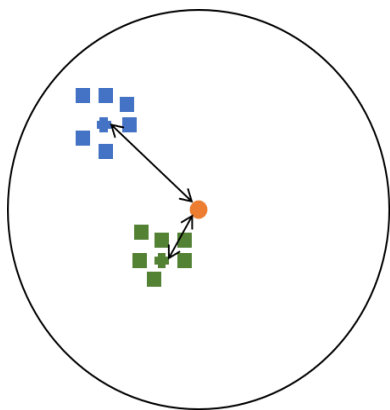
模型评估方法小结

测试方法	数学表达	注意事项	优缺点
留出法 (Hold-out)	$D=S \cup T$ $S \cap T = \emptyset$	分层采样 重复实验取平均评估结果	测试集小，评估结果方差较大 测试集大，评估结果偏差较大
交叉验证法 (Cross validation)	$D=D_1 \cup \dots \cup D_k$ $D_i \cap D_j = \emptyset \ (i \neq j)$	P次k折交叉验证	稳定性和保真性很大程度 取决于k
留一法 (LOO)	$D=D_1 \cup \dots \cup D_k$ $D_i \cap D_j = \emptyset \ (i \neq j)$ $k= D $	每次使用一个样本验证	不受随机样本划分方式影响， 数据量大时计算量大
自助法 (Bootstrap)	$ S = D $ $T=D-S$	可重复采样/有放回采样	数据集较小有用，改变初始数据的分布会引入偏差

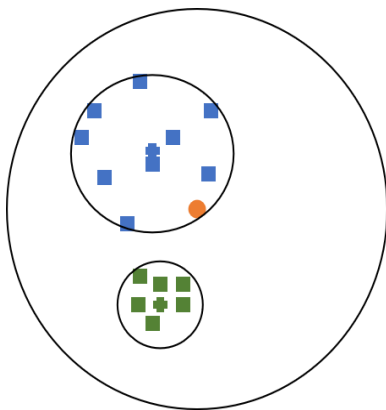
8.3 偏差-方差 (Bias-Variance)

- **偏差**: 表示估计量或预测量的平均值相对于估计对象或预测对象的真值之间的差异。衡量的是估计值或预测值与未知真值间的偏离程度。偏差越大, 越偏离真实数据。
$$B(\hat{\theta}) \equiv E(\hat{\theta}) - \theta \quad B(\hat{y}) \equiv E(\hat{y}) - y$$
- **方差**: 表示估计量或预测量对样本数据随机变异的敏感程度, 衡量的是估计值或预测值围绕自己的平均值的变动程度、发散程度。方差越大, 数据的分布越分散。
$$V(\hat{\theta}) \equiv E[(\hat{\theta} - E(\hat{\theta}))^2] \quad V(\hat{y}) \equiv E[(\hat{y} - E(\hat{y}))^2]$$

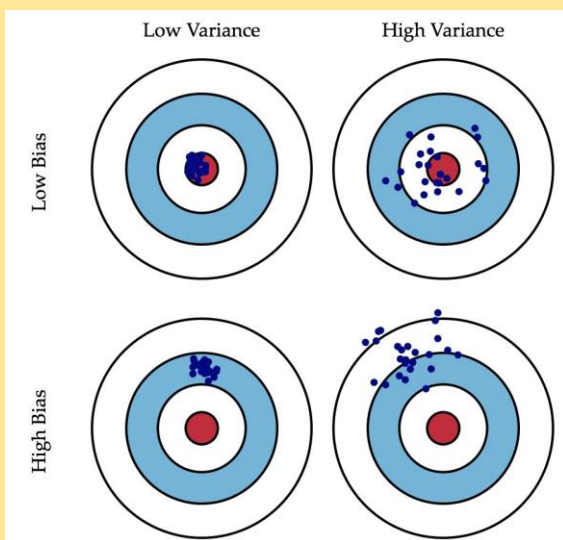
偏差: 跟目标比



方差: 跟自己比



偏差-方差实例: 谁的枪法好? 谁的枪法差?



误差 (Error)

对模型预测能力的评估，可以通过其在样本上的误差来估计。关于误差的几个概念如下：

- 误差(**error**)：学习器的实际预测输出与样本的真实输出之间的差异
- 训练误差(**training error**)：学习器在训练集上的误差，又称为经验误差(**empirical error**)
- 测试误差(**testing error**)：学习器在测试数据集上的平均损失，反应了模型对未知数据的预测能力
- 泛化误差(**generalization**)：学习器在新样本上的误差

我们希望泛化误差小的学习器，通常利用最小化训练误差的原则来训练模型，但真正值得关心的是测试误差，一般情况下我们通过测试误差来近似估计模型的泛化能力。

偏差-方差分解 (Bias-Variance Decomposition)

前面提到，我们希望泛化误差小的学习器，而学习算法的泛化误差可以分解为三个部分：偏差、方差和噪声。这里运用下表所示数学符号代表的含义。

符号	含义
x	测试样本
D	数据集
y_D	x 在数据集 D 中的输出标签
y	x 的真实输出标签
f	学习器在训练集 D 上学习到的模型
$f(x; D)$	由训练集 D 学得模型 f 对 x 的预测输出
$\bar{f}(x)$	模型 f 对 x 的期望预测输出

根据上面符号的定义可以将泛化误差分解为以下三个部分：

$$Err(x) = \underbrace{\mathbb{E}_D \left[(f(x; D) - \bar{f}(x))^2 \right]}_{\text{variance}} + \underbrace{(\bar{f}(x) - y)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_D \left[(y_D - y)^2 \right]}_{\text{noise}}$$

结论：泛化误差可分解为方差、偏差与噪声之和。

偏差和方差分解 (Bias-Variance Decomposition) (*: 选学)

偏差、方差分解的推导过程如下，以回归任务为例，学习算法的期望预测为： $\bar{f}(\mathbf{x}) = \mathbb{E}_D[f(\mathbf{x}; D)]$

使用样本数相同的不同训练集产生的方差为： $var(\mathbf{x}) = \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))^2]$

噪声为： $\epsilon^2 = \mathbb{E}_D [(y_D - y)^2]$

期望输出与真实标记的差别称为偏差(bias)： $bias^2(\mathbf{x}) = (\bar{f}(\mathbf{x}) - y)^2$

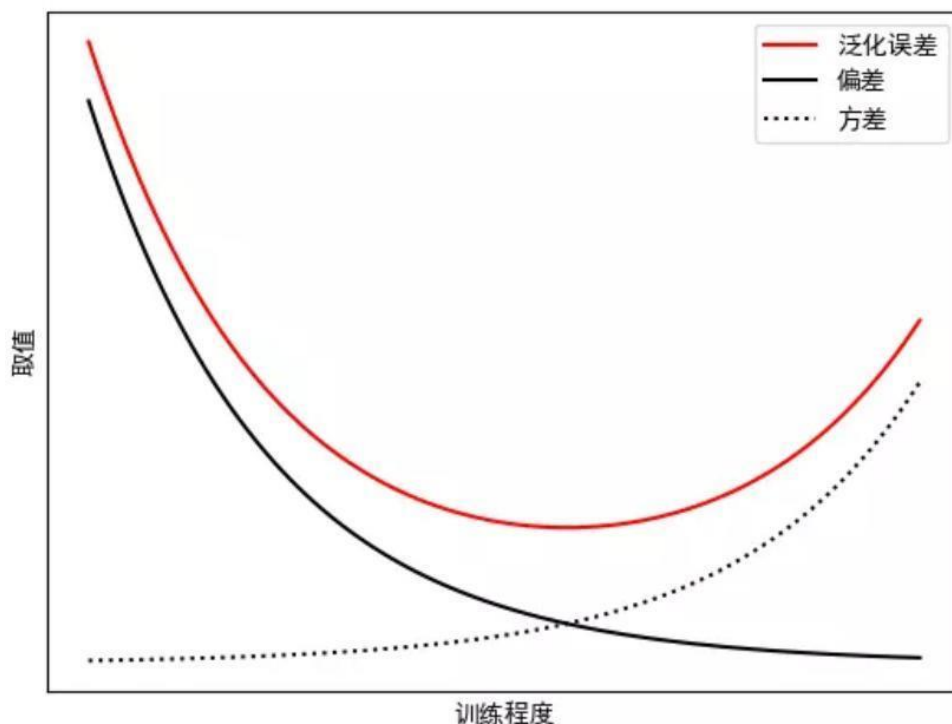
假设噪声的期望为0，即 $\mathbb{E}_D[y_D - y] = 0$ 。通过多项式展开合并可对算法的期望泛化误差进行分解： $E(f; D) = \mathbb{E}_D [(f(\mathbf{x}; D) - y_D)^2]$

$$\begin{aligned} &= \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}) + \bar{f}(\mathbf{x}) - y_D)^2] \\ &= \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_D [(\bar{f}(\mathbf{x}) - y_D)^2] \\ &\quad + \mathbb{E}_D [2(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y_D)] \\ &= \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_D [(\bar{f}(\mathbf{x}) - y_D)^2] \\ &= \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_D [(\bar{f}(\mathbf{x}) - y + y - y_D)^2] \\ &= \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_D [(\bar{f}(\mathbf{x}) - y)^2] + \mathbb{E}_D [(y - y_D)^2] \\ &\quad + 2\mathbb{E}_D [(\bar{f}(\mathbf{x}) - y)(y - y_D)] \\ &= \mathbb{E}_D [(f(\mathbf{x}; D) - \bar{f}(\mathbf{x}))^2] + (\bar{f}(\mathbf{x}) - y)^2 + \mathbb{E}_D [(y_D - y)^2] \end{aligned}$$

于是可得： $E(f; D) = bias^2(\mathbf{x}) + var(\mathbf{x}) + \epsilon^2$

模型拟合的偏差、方差与泛化误差的关系

一般情况，偏差与方差是有冲突的，如下图所示（下方的黑线表示偏差，上方红线表示泛化误差）。当训练程度小时，由于欠拟合，数据预测的偏差较大，此时影响泛化误差的主要因素是偏差。当训练程度逐渐变大时，模型越来越完善，偏差会变小，方差会变大。当训练程度过大时，模型过拟合，虽然偏差非常小，但方差非常大，此时影响泛化误差的主要因素是方差。因此我们无法保证每个指标都达到自身的最佳。



泛化误差与偏差、方差的关系示意图

误差诊断 (Error diagnostics)

■ 偏差-方差可以反映模型的过拟合与欠拟合

- **高偏差对应于模型的欠拟合**：模型过于简单，以至于未能很好的学习训练集，从而使得训练误差过高。例如，线性回归去拟合非线性的数据集。此时模型预测的方差较小，表示预测较稳定。但是模型预测的偏差会较大，表示预测不准确。
- **高方差对应于模型的过拟合**：模型过于复杂，以至于将训练集的细节都学到，将训练集的一些细节当做普遍的规律，从而使得测试集误差与训练集误差相距甚远。例如，不做任何剪枝的决策树，可以在任何训练集上做到极高的准确率。此时模型预测的偏差较小，表示预测较准确。但是模型预测的方差较大，表示预测较不稳定。

过拟合与泛化能力：过拟合 → 泛化能力差

- 复杂的模型，信息吸收能力强，数据拟合能力强，不仅能学到训练数据中存在的真实模式与相关关系，也能学到数据中随机噪声呈现的虚假模式与关系
 - 复杂模型主要体现在参数过多，能记忆更多的数据细节，相对弱化了预测中真正需要的固定模式与相关关系
 - 用过拟合的复杂模型去做预测，容易受新样本中随机噪声的影响。在考察大量新样本的预测结果时，错分率较大，呈现出偏差不低、方差很高的特征——模型泛化能力差的典型特征
 - 使用少量数据进行训练的复杂模型，容易产生对过拟合
 - 过拟合模型，就像一个读死书的书呆子：记得很多，但不能活用
 - **过拟合-泛化能力的关系**，是**偏差-方差权衡**的一个具体实例

If you torture the data long enough, it will confess.

Ronald Coase, 1991 Nobel Laureate economist

如何发现过拟合？如何解决过拟合？

- 如何发现过拟合现象？

- 利用交叉验证

- 如果训练精度、测试精度都低——欠拟合

- 如果训练精度高、测试精度低——过拟合

- 观察学习曲线和验证曲线

- 如何解决过拟合问题？——降低模型复杂度

- 减少模型参数 (人工手动)

- 优化模型特征 (特征工程)

- 约束模型参数 (对复杂性进行惩罚的正则化)

- 增加训练数据 (增加过拟合难度)

- 改善数据质量 (修复数据错误，填补数据缺失，清除异常数据)

8.4 模型性能度量

- **性能度量 (Performance measure)**：衡量模型泛化能力的评价标准

- **回归 (Regression)**: **均方误差 (Mean squared error)**

- 离散数据: $E(f; D) = \frac{1}{m} \sum_{i=1}^m (f(\mathbf{x}_i) - y_i)^2$
- 连续数据: $E(f; \mathcal{D}) = \int_{\mathbf{x} \sim \mathcal{D}} (f(\mathbf{x}) - y)^2 p(\mathbf{x}) d\mathbf{x}$

- **分类 (classification)**: **错误率 (Error rate)** 和 **准确率 (accuracy)**

- 离散数据: $E(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(\mathbf{x}_i) \neq y_i)$
 $\text{acc}(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(\mathbf{x}_i) = y_i) = 1 - E(f; D)$

- 连续数据: $E(f; \mathcal{D}) = \int_{\mathbf{x} \sim \mathcal{D}} \mathbb{I}(f(\mathbf{x}) \neq y) p(\mathbf{x}) d\mathbf{x}$

$$\text{acc}(f; \mathcal{D}) = \int_{\mathbf{x} \sim \mathcal{D}} \mathbb{I}(f(\mathbf{x}) = y) p(\mathbf{x}) d\mathbf{x} = 1 - E(f; \mathcal{D})$$

分类器的混淆矩阵(第1种表达)、查全率R、查准率P

		<u>Hypothesized class</u> (预测类别)			
		Y	N		
<u>True class</u> (真实类别)	p	True Positives	False Negatives	fp rate = $\frac{FP}{N}$	tp rate = $\frac{TP}{P}$
	n	False Positives	True Negatives	precision = $\frac{TP}{TP+FP}$	recall = $\frac{TP}{P}$

accuracy = $\frac{TP+TN}{P+N}$

F-measure = $\frac{2}{1/\text{precision}+1/\text{recall}}$

- TP (True Positive, 真正例/真阳性): 分类器将正例预测为正例的数量
- FN (False Negative, 假反例/假阴性): 分类器将正例预测为反例的数量
- FP (False Positive, 假正例/假阳性): 分类器将反例预测为正例的数量
- TN (True Negative, 真反例/真阴性): 分类器将反例预测为反例的数量

混淆矩阵Python计算示例：

```
In [1]: ▶ from sklearn.metrics import confusion_matrix
y_true = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
y_pred = [0, 1, 0, 0, 0, 0, 0, 1, 1, 1]
confusion_matrix = confusion_matrix(y_true, y_pred)
print(confusion_matrix)

[[4 1]
 [2 3]]
```

```
In [2]: ▶ from sklearn.metrics import confusion_matrix
y_true=[2, 1, 0, 1, 2, 0]
y_pred=[2, 0, 0, 1, 2, 1]
confusion_matrix=confusion_matrix(y_true, y_pred)
print(confusion_matrix)

[[1 1 0]
 [1 1 0]
 [0 0 2]]
```

第1个单元：二分类的混淆矩阵，混淆矩阵的行为真实分类，矩阵的列为实验预测分类；

第2个单元：三分类的混淆矩阵，混淆矩阵的行为真实分类，矩阵的列为实验预测分类。

分类器的混淆矩阵(第1种表达)、查全率R、查准率P

		<u>Hypothesized class</u> (预测类别)			
		Y	N		
<u>True class</u> (真实类别)	p	True Positives	False Negatives	$fp\ rate = \frac{FP}{N}$	$tp\ rate = \frac{TP}{P}$
	n	False Positives	True Negatives	$precision = \frac{TP}{TP+FP}$	$recall = \frac{TP}{P}$
				$accuracy = \frac{TP+TN}{P+N}$	
				$F\text{-measure} = \frac{2}{1/precision+1/recall}$	

- **准确率**: $Accuracy = (TP+TN)/(P+N)$, 预测正确的样本 (TP和TN) 数目占总样本数目的比例。
- **错误率**: $Error\ Rate = (FP+FN)/(P+N)$, 预测错误的样本数目占总样本数目的比例。
- **精确率 (查准率)**: $Precision = TP/(TP+FP)$, 真正例的样本数目占有所有预测类别为正例样本数目的比例。
- **召回率 (查全率)**: $Recall = TP/(TP+FN)$, 真正例的样本数目占有所有真实类别为正例的样本数目的比例。

分类器的混淆矩阵(第2种表达)、查全率R、查准率P

		True class			
		p	n		
Hypothesized class	Y	True Positives	False Positives	fp rate = $\frac{FP}{N}$	tp rate = $\frac{TP}{P}$
	N	False Negatives	True Negatives	precision = $\frac{TP}{TP+FP}$	recall = $\frac{TP}{P}$
Column totals:		P	N	accuracy = $\frac{TP+TN}{P+N}$	
				F-measure = $\frac{2}{1/\text{precision}+1/\text{recall}}$	

- **准确率**: $\text{Accuracy} = (TP+TN)/(P+N)$, 预测正确的样本 (TP和TN) 数目占总样本数目的比例。
- **错误率**: $\text{Error Rate} = (FP+FN)/(P+N)$, 预测错误的样本数目占总样本数目的比例。
- **精确率 (查准率)**: $\text{Precision} = TP/(TP+FP)$, 真正例的样本数目占有所有预测类别为正例样本数目的比例。
- **召回率 (查全率)**: $\text{Recall} = TP/(TP+FN)$, 真正例的样本数目占有所有真实类别为正例的样本数目的比例。

F1度量

- 对于Precision和Recall，虽然从计算公式来看，并没有什么必然的相关性关系。但是在大规模数据集合中，这两个指标往往是相互制约的。理想情况下，两个指标都较高。但一般情况下，如果Precision高，Recall就低；如果Recall高，Precision就低。
- 在实际项目中，常常需要根据具体情况做出取舍，例如一般的搜索情况，在保证召回率的条件下，尽量提升精确率。而像癌症检测、地震检测、金融欺诈等，则在保证精确率的条件下，尽量提升召回率。
- 很多时候我们需要综合权衡这2个指标，这就引出了一个新的指标F-score。这是综合考虑Precision和Recall后的调和值。

F1度量

更常用的 **F1 度量** (查全率与查准率的调和平均):

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{\text{样例总数} + TP - TN}$$

□ **真正例率(TPR)**: $TPR = TP / (TP + FN)$

被正确分类为正例的样本数，占全部实际正样本的比例---查全率

□ **假正例率(FPR)**: $FPR = FP / (FP + TN)$

被错误分类为正例的样本数，占全部实际负样本的比例

ROC(接收者操作特征曲线)、AUC(曲线下面积)

- ROC: 根据分类器的预测结果(预测属于正例的概率)对样本进行排序: 排在最前面的是分类器认为“最可能”是正类的样本,排在最后面的是分类器认为“最不可能”是正类的样本。假设排序后的样本集合为 $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$, 预测为正类的概率依次为 (p_1, p_2, \dots, p_n) 。接下来,从高到低依次将 p_i 作为分类阈值,即:

$$\hat{y}_j = \begin{cases} 1, & \text{if } p_j \geq p_i \\ 0, & \text{else} \end{cases}, j=1, 2, \dots, n$$

当样本属于正例的概率大于等于 p_i 时,我们认为它是正例,否则为负例,这样每次选择一个不同的阈值,计算得到的真正例率记做 TPR_i , 假正例率记做 FPR_i 。以真正例率为纵轴、假正例率为横轴作图,就得到ROC曲线。

- AUC: ROC(Receiver Operating Characteristic)曲线下面积大小记作AUC (Area Under ROC Curve), AUC值是一个概率值,涵义是当你随机挑选一个正样本以及一个负样本,当前的分类算法根据计算得到的预测值将这个正样本排在负样本前面的概率就是AUC值。AUC值越大,当前的分类算法越有可能将正样本排在负样本前面,即能够更好的分类。

ROC（接收者操作特征曲线）

对于右图所示ROC曲线：

- 点 (0,1)：即FPR=0, TPR=1, 意味着FN=0, 并且FP=0。这是一个完美的分类器，它将所有的样本都正确分类。
- 点 (1,0)：即FPR=1, TPR=0, 意味着这是一个最糟糕的分类器，因为它成功避开了所有正确答案。
- 点 (0,0)：即FPR=TPR=0, 即PF=TP=0, 可以发现该分类器预测的所有样本为负样本。
- 点 (1,1)：分类器预测的所有样本均为正样本。

通常ROC曲线越靠近点 (0,1)，分类器的分类性能越好。

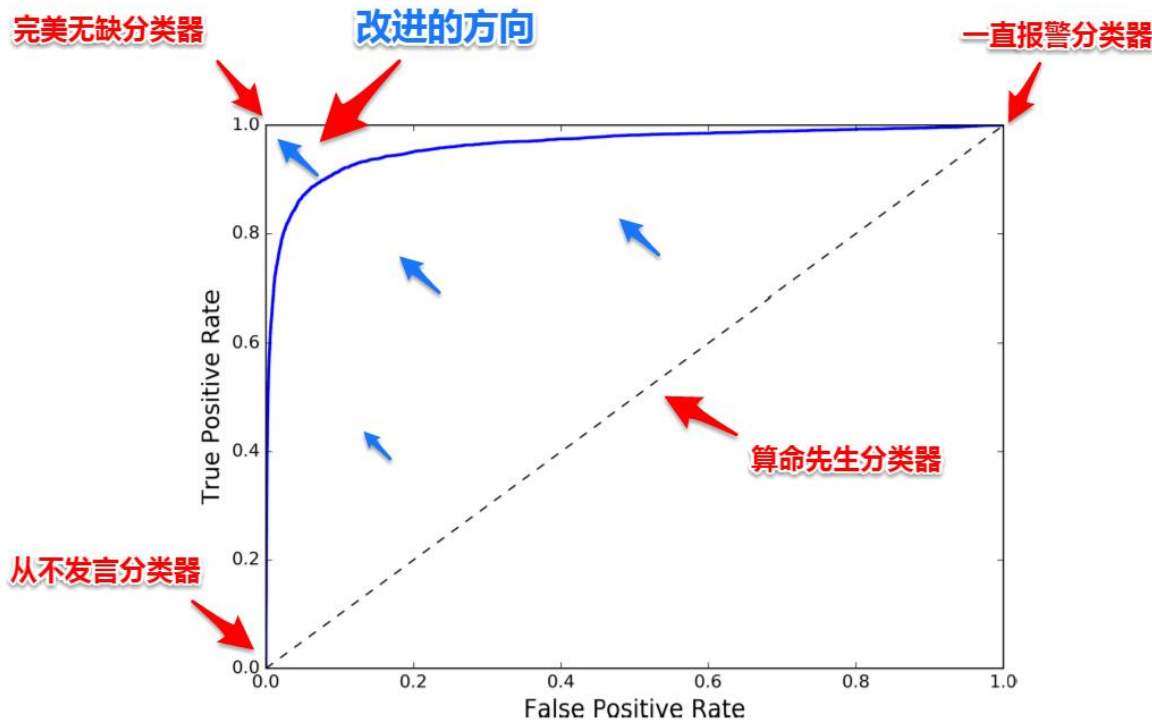
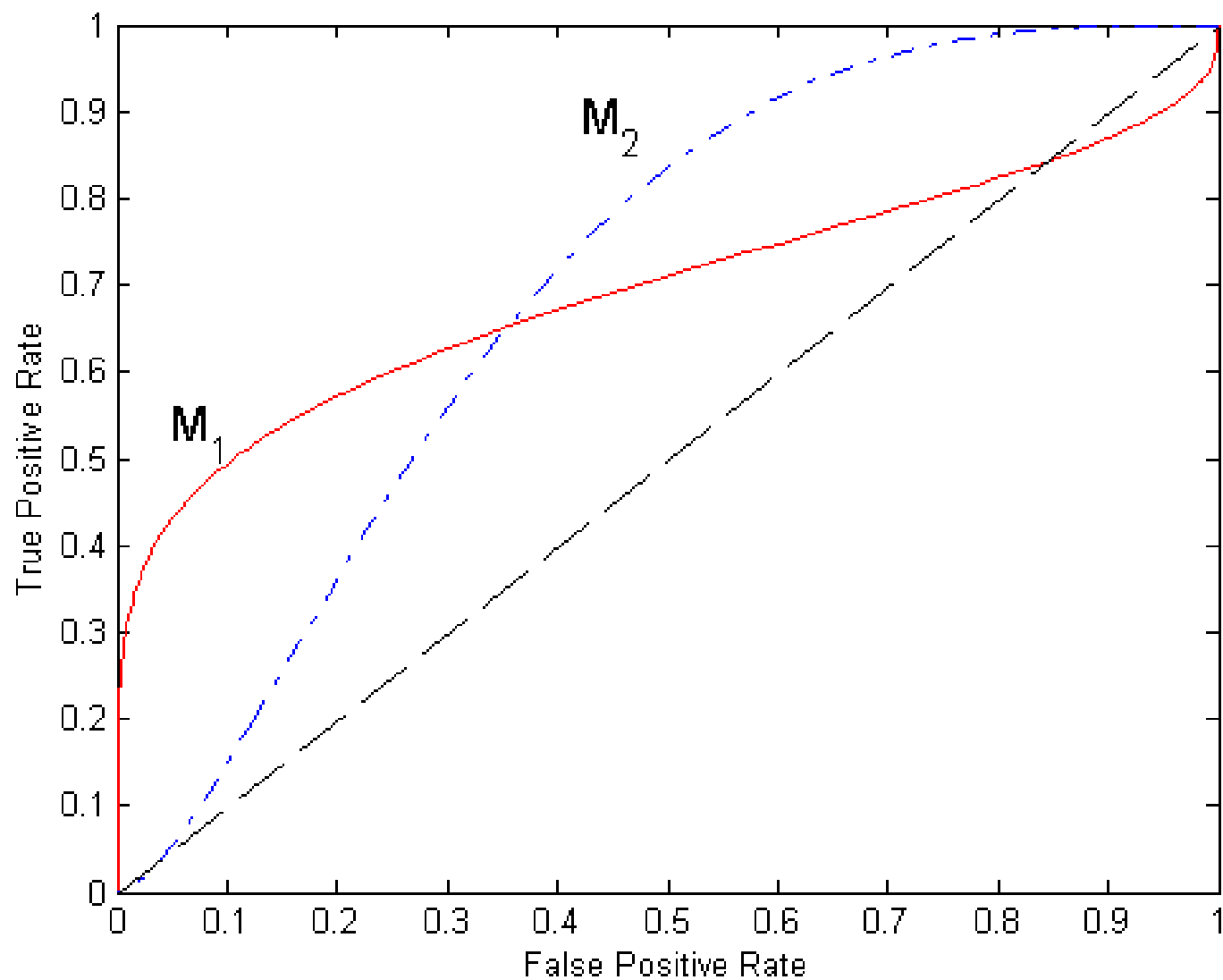


图 ROC曲线示例

$$TPR = TP / (TP + FN)$$

$$FPR = FP / (FP + TN)$$



- 在假正例率小于0.36区域，分类器 M_1 优于 M_2
- 在假正例率大于0.36区域，分类器 M_2 优于 M_1

P-R曲线(*: 选学)

对二类分类问题，根据分类器的预测结果 (预测属于正例的概率) 对样本进行排序：排在最前面的是分类器认为“最可能”是正类的样本，排在最后面的是分类器认为“最不可能”是正类的样本。假设排序后的样本集合为 $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$ ，预测为正类的概率依次为 (p_1, p_2, \dots, p_n) 。接下来，从高到低依次将 p_i 作为分类阈值，即：

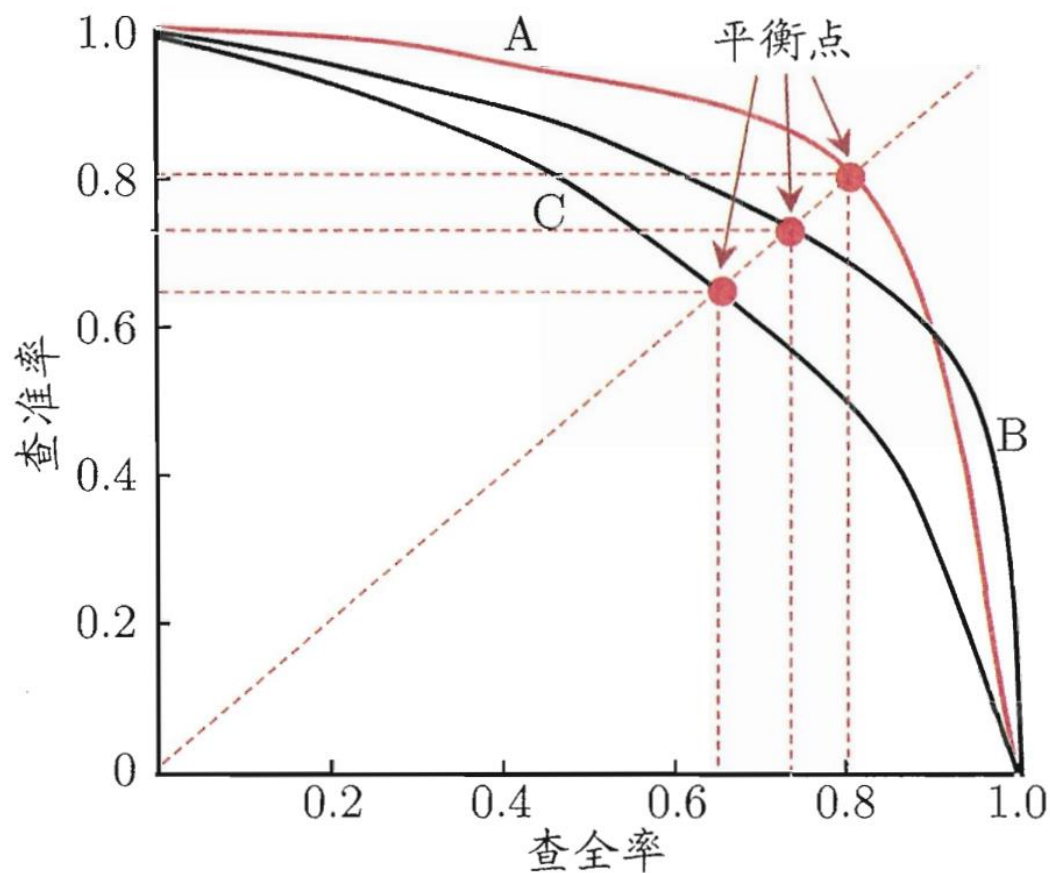
$$\hat{y}_j = \begin{cases} 1, & \text{if } p_j \geq p_i \\ 0, & \text{else} \end{cases}, j=1, 2, \dots, n$$

当样本属于正例的概率大于等于 p_i 时，我们认为它是正例，否则为负例，这样每次选择一个不同的阈值，查准率记做 P_i ，查全率记做 R_i 。以查准率为纵轴、查全率为横轴作图，就得到P-R曲线。

P-R曲线(*: 选学)

P-R 曲线直观显示出分类器在样本总体上的查全率、查准率。因此可以通过两个分类器在同一个测试集上的 P-R 曲线来比较它们的预测能力：

- 如果分类器B的P-R 曲线被分类器A的曲线完全包住，则可断言：A的性能好于B。
- 如果分类器A的 P-R 曲线与分类器B的曲线发生了交叉，则难以一般性的断言两者的优劣，只能在具体的查准率和查全率下进行比较。
- 平衡点是P-R 曲线上查准率等于查全率的点，可以判定：平衡点较远的P-R 曲线较好。



“调参”与最终模型

算法的参数：一般由人工设定，亦称“超参数”

模型的参数：一般由学习确定

调参(parameter tuning)过程相似：先产生若干模型，
然后基于某种评估方法进行选择

参数调得好不好往往对最终性能有关键影响

区别：训练集 vs. 测试集 vs. 验证集 (validation set)

算法参数选定后，要用“训练集+验证集”重新训练最终模型

8.5 超参数的搜索

- 超参数搜索的一般过程：

1. 将数据集分成训练集、验证集、测试集。
2. 在训练集上根据模型的性能指标对模型参数进行优化。
3. 在验证集上根据模型的性能指标对模型的超参数进行搜索。
4. 步骤2和3交替迭代进行，最终确定模型的参数和超参数，并在测试集中评价模型的优劣。

其中，步骤3的搜索过程需要的搜索算法有以下几种：

1. 网格搜索
2. 随机搜索
3. 贝叶斯搜索

.....

网格搜索 (Grid Search)

最传统的超参数优化方法就是网格搜索 (Grid search) ，即在指定范围内的超参数集合进行搜索。网格搜索的做法是：

- 对于每个超参数，选择一个较小的有限值集合去搜索。
- 然后这些超参数笛卡尔乘积得到多组超参数。
- 网格搜索使用每一组超参数训练模型，挑选验证集误差最小的超参数作为最好的超参数。

如何确定搜索集合的范围？

- 如果超参数是数值，则搜索集合的最小、最大元素可以基于先前相似实验的经验保守地挑选出来。
- 如果超参数是离散的，则直接使用离散值。

网格搜索 (Grid Search)

网格搜索是一种全因子实验设计，它给每个超参设置组有限的测试值，然后验证每一种参数组合。网格搜索具有以下特点：

- 效率受超参数数量的影响大,验证次数会随参数数量的增长呈指数增长，出现维度灾难。
- 增加每个超参的搜索点会极大增加验证次数，假设有 n 个超参，每个超参仅有2个值，配置总数达到 2^n ，而当每个超参有3个超参值时，配置总数达到 3^n ，数量大大增加。因此，此方法仅在少量配置上是可行的。
- 配置验证相互不影响，容易并行验证。

随机搜索(Random Search)

随机搜索是一种可以替代网格搜索的方法，它编程简单、使用方便、能更快收敛到超参数的良好取值。

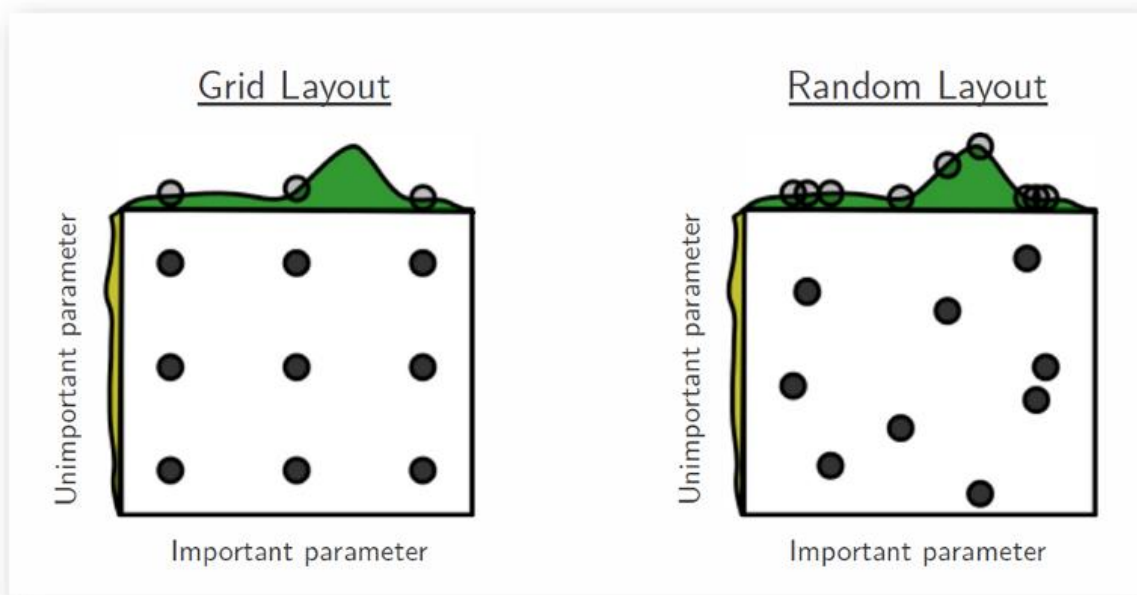
- 首先为每个超参数定义一个边缘分布，如伯努利分布（对应着二元超参数）或者对数尺度上的均匀分布（对应着正实值超参数）。
- 然后假设超参数之间相互独立，从各分布中抽样出一组超参数。
- 使用这组超参数训练模型。
- 经过多次抽样 -> 训练过程，挑选验证集误差最小的超参数作为最好的超参数。

随机搜索的优点：

- 不需要离散化超参数的值，也不需要限定超参数的取值范围。这允许我们在一个更大的集合上进行搜索。
- 当某些超参数对于性能没有显著影响时，随机搜索相比于网格搜索指数级地高效，它能更快的减小验证集误差。

随机搜索(Random Search)

随机搜索预先设置一个搜索的次数，然后随机采样超参组合进行验证。相比网格搜索，随机搜索的优点是对于每个超参能够验证更多的参数点。直观地，给定 N 个超参进行 C 次验证，假如采用网格搜索的方式，每个超参只能验证 $C^{\frac{1}{N}}$ 个点，但是随机搜索可以对每个超参都验证 C 个不同的点。当有部分超参对模型特别重要时，随机搜索的这个性质就呈得尤为重要。如图所示随机搜索可以验证到重要参数更多的点。



随机搜索(Random Search)

随机搜索还有如下特点：

- (1) 算法没有对模型本身做任何假设，而且只要有足够多的资源，随机搜索在预期中能够达到最优解的任意近似。
- (2) 经常会比基于模型的搜索算法消耗更长的时间。例如，当超参空间是 N 个独立的布尔值参数，并且每个参数的取值分别对应一个好的参数和一个差的参数时，随机搜索需要 2^{N-1} 次验证才能找到最优解，而按照每次优化单个超参的思路，只需要验证 $N+1$ 次就可以找到最优解。
- (3) 不依赖其他配置的验证，可以并行验证。

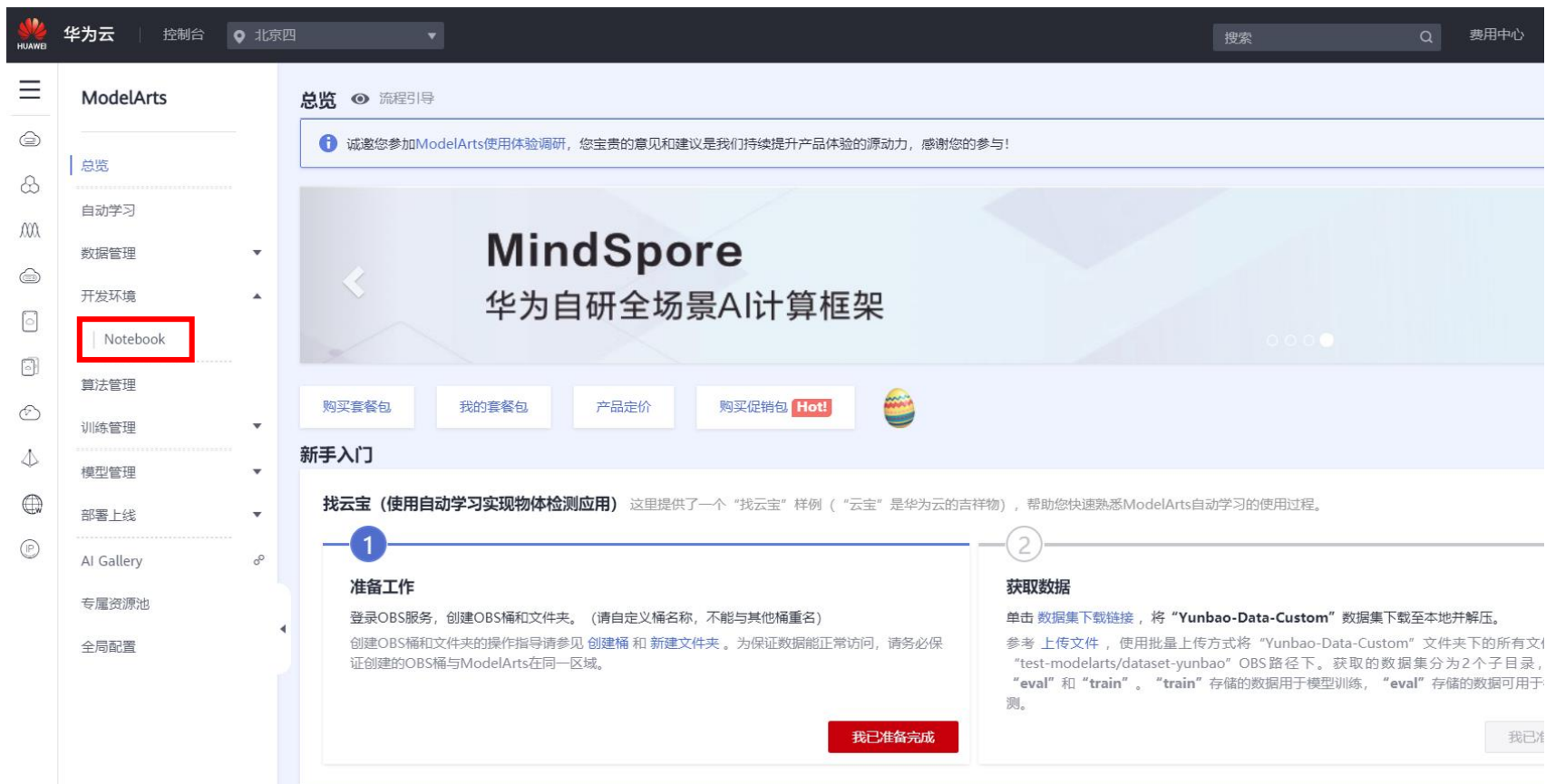
8.5 模型评估与选择实例 (基于华为云ModelArts AI平台)

下面介绍一个模型评估与选择的实例，在该实例中首先利用线性回归、SVM、KNN回归算法训练三个回归模型，对应用市场中的APP进行评分预测，接下来将评分数据离散化，用决策树模型训练分类模型并分别对各模型进行评估和调参。以下是该实例中采用的数据集截图，共包含10240条APP相关数据，每条数据包含41个属性。各模型需要实现对Rating的预测。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Category_ART_AND_DESIGN	Category_AUTO_AND_VEHICLES	Category_	Category_	Category_	Category_	Category_
0	4.1	159	19	10000	1	0	1	3	1	0	0	0	0	0	
1	3.9	967	14	500000	1	0	1	3	1	0	0	0	0	0	
2	4.7	87510	8.7	5000000	1	0	1	3	1	0	0	0	0	0	
3	4.5	215644	25	50000000	1	0	4	3	1	0	0	0	0	0	
4	4.3	967	2.8	100000	1	0	1	3	1	0	0	0	0	0	
5	4.4	167	5.6	50000	1	0	1	3	1	0	0	0	0	0	
6	3.8	178	19	50000	1	0	1	3	1	0	0	0	0	0	
7	4.1	36815	29	1000000	1	0	1	3	1	0	0	0	0	0	
8	4.4	13791	33	1000000	1	0	1	3	1	0	0	0	0	0	
9	4.7	121	3.1	10000	1	0	1	3	1	0	0	0	0	0	
10	4.4	13880	28	1000000	1	0	1	3	1	0	0	0	0	0	
11	4.4	8788	12	1000000	1	0	1	3	1	0	0	0	0	0	
12	4.2	44829	20	10000000	1	0	4	3	1	0	0	0	0	0	
13	4.6	4326	21	100000	1	0	1	3	1	0	0	0	0	0	
14	4.4	1518	37	100000	1	0	1	3	1	0	0	0	0	0	
15	3.2	55	2.7	5000	1	0	1	3	1	0	0	0	0	0	
16	4.7	3632	5.5	500000	1	0	1	3	1	0	0	0	0	0	
17	4.5	27	17	10000	1	0	1	3	1	0	0	0	0	0	
18	4.3	194216	39	5000000	1	0	1	3	1	0	0	0	0	0	
19	4.6	224399	31	10000000	1	0	1	3	1	0	0	0	0	0	
20	4	450	14	100000	1	0	1	3	1	0	0	0	0	0	
21	4.1	654	12	100000	1	0	1	3	1	0	0	0	0	0	
22	4.7	7699	4.2	500000	1	0	2	3	1	0	0	0	0	0	
23	4.3	61	7	100000	1	0	1	3	1	0	0	0	0	0	
24	4.7	118	23	50000	1	0	1	3	1	0	0	0	0	0	
25	4.8	192	6	10000	1	0	1	3	1	0	0	0	0	0	
26	4.7	20260	25	500000	1	0	1	3	1	0	0	0	0	0	
27	4.1	203	6.1	100000	1	0	1	3	1	0	0	0	0	0	
28	3.9	136	4.6	10000	1	0	1	3	1	0	0	0	0	0	
29	4.1	223	4.2	100000	1	0	1	3	1	0	0	0	0	0	
30	4.2	1120	9.2	100000	1	0	1	3	1	0	0	0	0	0	

模型评估与选择实例

首先进入ModelArts平台并点击左侧导航栏当中的开发环境当中的Notebook选项



The screenshot displays the Huawei ModelArts console interface. At the top, the header includes the Huawei logo, the text "华为云", "控制台", a location dropdown set to "北京四", a search bar, and a "费用中心" link. The left sidebar contains a menu with icons and labels: "ModelArts", "总览", "自动学习", "数据管理", "开发环境", "Notebook" (highlighted with a red box), "算法管理", "训练管理", "模型管理", "部署上线", "AI Gallery", "专属资源池", and "全局配置". The main content area features a "MindSpore" banner with the text "华为自研全场景AI计算框架". Below the banner are buttons for "购买套餐包", "我的套餐包", "产品定价", and "购买促销包 Hot!". A "新手入门" section follows, containing two numbered steps: 1. "准备工作" (Preparation) and 2. "获取数据" (Get Data). Step 1 includes instructions on logging into OBS and creating buckets/folders. Step 2 includes instructions on downloading and uploading data. At the bottom right, there is a "我已准备完成" button.

模型评估与选择实例

在Notebook界面点击创建按钮并按提示完成Notebook的创建



创建 Notebook

< 返回 Notebook 列表

1 服务选型 — 2 规格确认 — 3 完成

使用指南 | 建议反馈

* 计费模式: 按需计费

* 名称: notebook-acb7

描述: 0/512

* 工作环境: 公共镜像

名称	描述
<input checked="" type="radio"/> Multi-Engine 1.0 (Python3, Recommended)	MXNet-1.2.1, PySpark-2.3.2, Pytorch-1.0.0, TensorFlow-1.13.1, TensorFlow-1.8, XGBoost-Sklearn

* 资源池: 公共资源池 | 专属资源池

* 类型: CPU | GPU

* 规格: [限时免费]体验规格CPU版 | 2 核 8GiB | 8 核 32GiB

适合场景: 适合在Notebook上的代码编写与体验

1、免费规格用于使用体验，会在1小时后自动停止，72小时内没有再次启动，会释放资源，**请注意文件备份**。每个用户只限创建一个基于此免费规格的实例。

2、ModelArts免费算力不包含对象存储服务（OBS）存储资源费用，对象存储服务（OBS）计费标准详见如下链接：[对象存储服务（OBS）计费详情](#)。

☒ 我已阅读并同意以上内容

存储配置: 云硬盘（EVS） | 对象存储服务（OBS）

Notebook文件管理页面显示/home/ma-user/work目录（云硬盘（EVS）挂载位置），只有该目录下的数据在Notebook停止后不会被清理。

模型评估与选择实例

打开创建好的Notebook之后即可进行代码的编写，首先读取数据并确定标签

```
In [1]: #导入相关库
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

数据集划分

```
In [2]: #读取数据集
data = pd.read_csv('AppDataV2.csv', index_col=0)
```

```
In [3]: #首先确定样本的数据的标签
```

```
X = data.drop(["Rating"], axis='columns')
Y = data["Rating"]
```

去除数据集中每条数据的Rating列
将该列取出作为每条数据的标签

```
In [4]: X.info()
```

模型评估与选择实例

此时数据集X的详情输出如下图所示，共包含10240条数据，每条数据包含40个属性

```
In [53]: X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10240 entries, 0 to 10239
Data columns (total 40 columns):
Reviews                10240 non-null int64
Size                   10240 non-null float64
Installs               10240 non-null float64
Type                   10240 non-null int64
Price                  10240 non-null float64
Content Rating         10240 non-null int64
Genres                 10240 non-null int64
Category_ART_AND_DESIGN 10240 non-null int64
Category_AUTO_AND_VEHICLES 10240 non-null int64
Category_BEAUTY         10240 non-null int64
Category_BOOKS_AND_REFERENCE 10240 non-null int64
Category_BUSINESS       10240 non-null int64
Category_COMICS         10240 non-null int64
Category_COMMUNICATION  10240 non-null int64
Category_DATING         10240 non-null int64
Category_EDUCATION      10240 non-null int64
Category_ENTERTAINMENT  10240 non-null int64
```

模型评估与选择实例

再次回顾我们的APP评分数据集里面的数据，我们首先需要对数据进行归一化预处理，为什么要进行归一化？归一化后加快了梯度下降求最优解的速度；如果机器学习模型使用梯度下降法求最优解时，归一化往往非常有必要，否则很难收敛甚至不能收敛。其次一些分类器需要计算样本之间的距离（如欧氏距离），例如本次实例中采用的KNN，下表中APP安装数量这个特征值域范围非常大，那么距离计算就主要取决于这个特征，从而与实际情况相悖。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Category_ART_AND_DESIGN	Category_AUTO_AND_VEHICLES	Category_	Category_	Category_	Category_	Category_
0	4.1	159	19	10000	1	0	1	3	1	0	0	0	0	0	0
1	3.9	967	14	500000	1	0	1	3	1	0	0	0	0	0	0
2	4.7	87510	8.7	5000000	1	0	1	3	1	0	0	0	0	0	0
3	4.5	215644	25	50000000	1	0	4	3	1	0	0	0	0	0	0
4	4.3	967	2.8	100000	1	0	1	3	1	0	0	0	0	0	0
5	4.4	167	5.6	50000	1	0	1	3	1	0	0	0	0	0	0
6	3.8	178	19	50000	1	0	1	3	1	0	0	0	0	0	0
7	4.1	36815	29	1000000	1	0	1	3	1	0	0	0	0	0	0
8	4.4	13791	33	1000000	1	0	1	3	1	0	0	0	0	0	0
9	4.7	121	3.1	10000	1	0	1	3	1	0	0	0	0	0	0
10	4.4	13880	28	1000000	1	0	1	3	1	0	0	0	0	0	0
11	4.4	8788	12	1000000	1	0	1	3	1	0	0	0	0	0	0
12	4.2	44829	20	10000000	1	0	4	3	1	0	0	0	0	0	0
13	4.6	4326	21	100000	1	0	1	3	1	0	0	0	0	0	0
14	4.4	1518	37	100000	1	0	1	3	1	0	0	0	0	0	0
15	3.2	55	2.7	5000	1	0	1	3	1	0	0	0	0	0	0
16	4.7	3632	5.5	500000	1	0	1	3	1	0	0	0	0	0	0
17	4.5	27	17	10000	1	0	1	3	1	0	0	0	0	0	0
18	4.3	194216	39	5000000	1	0	1	3	1	0	0	0	0	0	0
19	4.6	224399	31	10000000	1	0	1	3	1	0	0	0	0	0	0
20	4	450	14	100000	1	0	1	3	1	0	0	0	0	0	0
21	4.1	654	12	100000	1	0	1	3	1	0	0	0	0	0	0
22	4.7	7699	4.2	500000	1	0	2	3	1	0	0	0	0	0	0
23	4.3	61	7	100000	1	0	1	3	1	0	0	0	0	0	0
24	4.7	118	23	50000	1	0	1	3	1	0	0	0	0	0	0
25	4.8	192	6	10000	1	0	1	3	1	0	0	0	0	0	0
26	4.7	20260	25	500000	1	0	1	3	1	0	0	0	0	0	0
27	4.1	203	6.1	100000	1	0	1	3	1	0	0	0	0	0	0
28	3.9	136	4.6	10000	1	0	1	3	1	0	0	0	0	0	0
29	4.1	223	4.2	100000	1	0	1	3	1	0	0	0	0	0	0
30	4.2	1120	9.2	100000	1	0	1	3	1	0	0	0	0	0	0

模型评估与选择实例

大多数机器学习算法中，会选择StandardScaler来进行特征缩放，因为MinMaxScaler对异常值非常敏感。在PCA，聚类，逻辑回归，支持向量机，神经网络这些算法中，StandardScaler往往是最好的选择。

StandardScaler的作用：去均值和方差归一化。且是针对每一个特征维度来做的，而不是针对样本。标准差标准化（standardScale）使得经过处理的数据符合标准正态分布，即均值为0，标准差为1。函数表达式如下：

$$x^* = \frac{x - \mu}{\sigma}$$

其中 μ 为所有样本数据的均值， σ 为所有样本数据的标准差。新值符合 ---> (旧值-均值) / 标准差。

将数据按期属性（按列进行）删除平均值和缩放到单位方差来标准化特征。得到的结果是，对于每个属性/每列来说所有数据都聚集在0附近，标准差为1，使得新的X数据集方差为1，均值为0。

模型评估与选择实例

选用StandardScaler的代码和输出结果如下图所示：

```
In [5]: x.iloc[30].std()
Out[5]: 15807.776631856452
```

iloc函数：通过行号来取行数据
std函数：得到的数就是这组数的标准差

```
In [6]: from sklearn.preprocessing import StandardScaler
sc_X=StandardScaler(copy=False)
X = pd.DataFrame(sc_X.fit_transform(X))
```

调用StandardScaler函数对数据进行归一化

```
In [7]: x.head()
Out[7]:
```

	0	1	2	3	4	5	6	7	8	9	...	30	31	32	33
0	-0.260256	-0.086188	-0.173110	-0.282134	-0.060853	-0.453022	-1.610166	12.60952	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.149205
1	-0.259277	-0.325793	-0.164460	-0.282134	-0.060853	-0.453022	-1.610166	12.60952	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.149205
2	-0.154461	-0.579775	-0.085016	-0.282134	-0.060853	-0.453022	-1.610166	12.60952	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.149205
3	0.000729	0.201339	0.709424	-0.282134	-0.060853	2.552773	-1.610166	12.60952	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.149205
4	-0.259277	-0.862509	-0.171522	-0.282134	-0.060853	-0.453022	-1.610166	12.60952	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.149205

5 rows x 40 columns

```
In [8]: x.iloc[30].std()
Out[8]: 2.0441345735552545
```

同一样本进行归一化处理后各特征值之间的标准差

通过上面的输出结果可以看出来在用StandardScaler函数进行归一化处理后各特征的数值被明显缩放，而且同一样本各特征的标准差有了显著的下降。

模型评估与选择实例

接下来进行数据集的划分

```
In [9]: #数据集划分
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 10)
```

```
In [10]: #from sklearn.preprocessing import StandardScaler
#sc_X=StandardScaler()
#X_train=sc_X.fit_transform(X_train)
#X_test=sc_X.transform(X_test)
```

```
In [11]: X
```

```
Out[11]:
```

	0	1	2	3	4	5	6	7	8	9	...	30	31	32	
0	-0.260256	-0.086188	-0.173110	-0.282134	-0.060853	-0.453022	-1.610166	12.609520	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
1	-0.259277	-0.325793	-0.164460	-0.282134	-0.060853	-0.453022	-1.610166	12.609520	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
2	-0.154461	-0.579775	-0.085016	-0.282134	-0.060853	-0.453022	-1.610166	12.609520	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
3	0.000729	0.201339	0.709424	-0.282134	-0.060853	2.552773	-1.610166	12.609520	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
4	-0.259277	-0.862509	-0.171522	-0.282134	-0.060853	-0.453022	-1.610166	12.609520	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
...
10235	-0.260403	1.543129	-0.173199	-0.282134	-0.060853	-0.453022	-0.699162	-0.079305	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
10236	-0.260444	-0.824172	-0.173285	-0.282134	-0.060853	-0.453022	-0.699162	-0.079305	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
10237	-0.260445	-0.541438	-0.173269	-0.282134	-0.060853	-0.453022	0.135924	-0.079305	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
10238	-0.260311	-0.364642	-0.173269	-0.282134	-0.060853	1.550842	-1.306498	-0.079305	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492
10239	0.221961	-0.086188	0.003255	-0.282134	-0.060853	-0.453022	-0.015910	-0.079305	-0.091489	-0.07213	...	-0.198185	-0.179025	-0.203188	-0.1492

模型评估与选择实例

线性回归模型的建立以及MSE的计算

回归算法

```
In [12]: from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, accuracy_score, r2_score
```

线性回归

`sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True):`

`fit_intercept`: 默认True, 是否计算模型的截距, 为False时, 则数据中心化处理。

`normalize`: 默认False, 是否中心化, 或者使用`sklearn.preprocessing.StandardScaler()`。

`copy_X`: 默认True, 否则x会被改写。

```
In [13]: #初始化线性回归模型
linreg = LinearRegression()
#训练模型
linreg.fit(X_train,y_train)
#训练集上的MSE
linreg_pred_train = linreg.predict(X_train)
linreg_mse_train = mean_squared_error(linreg_pred_train,y_train)
#输出测试集上的测试结果
linreg_pred_test=linreg.predict(X_test)
linreg_mse_test = mean_squared_error(linreg_pred_test,y_test)
```

```
In [14]: print("训练集MSE: ", linreg_mse_train)
print("测试集MSE: ", linreg_mse_test)
```

训练集MSE: 0.23003183400203064

测试集MSE: 0.22310830550022365

模型评估与选择实例

SVM回归模型的建立以及MSE的计算

SVM回归

`sklearn.svm.SVR` (kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, tol=0.001, C=1.0, verbose=False, max_iter=-1)

kernel: 指定要在算法中使用的内核类型。可以是'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'或者callable之一, 默认为rbf。

degree: 多项式poly函数的维度, 默认是3, 选择其他核函数时会被忽略。

gamma: 'rbf', 'poly' 和 'sigmoid' 的核函数参数。默认是'auto', 则会选择 $1/n_features$

coef0: 核函数的常数项。对于'poly'和'sigmoid'有用, 默认值= 0.0。

tol: 停止训练的误差值大小, 默认值= $1e-3$ 。

C: 惩罚参数, 默认= 1.0。

C越大, 相当于惩罚松弛变量, 希望松弛变量接近0, 即对误分类的惩罚增大, 趋向于对训练集全分对的情况, 这样对训练集测试时准确率很高, 但泛化能力弱。C值小, 对误分类的惩罚减小, 允许容错, 将他们当成噪声点, 泛化能力较强。

verbose: 日志。

max_iter: 最大迭代次数。-1为无限制。

```
In [15]: from sklearn.svm import SVR
from sklearn.metrics import accuracy_score
#初始化SVM模型
svr=SVR(kernel='rbf',C=1)
#训练
svr.fit(X_train,y_train)
#训练集上的MSE
svr_pred_train = svr.predict(X_train)
svr_mse_train = mean_squared_error(svr_pred_train,y_train)
#输出测试集上的测试结果
svr_pred_test=svr.predict(X_test)
svr_mse_test = mean_squared_error(svr_pred_test,y_test)
svr_pred_train
```

```
Out[15]: array([4.30201276, 4.19101256, 4.27980086, ..., 4.21456242, 4.28946345,
4.21034414])
```

```
In [16]: print("训练集MSE: ", svr_mse_train)
print("测试集MSE: ", svr_mse_test)
```

```
训练集MSE: 0.23104604651305077
测试集MSE: 0.22375583086876552
```

模型评估与选择实例

KNN回归模型的建立以及MSE的计算

KNN回归

`sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30):`

`n_neighbors`: knn算法中指定以最近的几个最近邻样本具有投票权，默认参数为5

`algorithm`: 即内部采用什么算法实现。有以下几种选择参数:

'ball_tree': 球树、

'kd_tree': kd树、

'brute': 暴力搜索、

'auto': 自动根据数据的类型和结构选择合适的算法。默认情况下是'auto'。

暴力搜索就不用说了大家都知道。具体前两种树型数据结构哪种好视情况而定。KD树是对依次对k维坐标轴，以中值切分构造的树，每一个节点是一个超矩形，在维数小于20时效率最高ball tree 是为了克服KD树高维失效而发明的，其构造过程是以质心c和半径r分割样本空间，每一个节点是一个超球体。一般低维数据用kd_tree速度快，用ball_tree相对较慢。超过20维之后的高维数据用kd_tree效果反而不佳，而ball_tree效果要好，具体构造过程及优劣势的理论大家有兴趣可以去具体学习。

`leaf_size`: 这个值控制了使用KD树或者球树时，停止建子树的叶子节点数量的阈值。这个值越小，则生成的KD树或者球树就越大，层数越深，建树时间越长，反之，则生成的KD树或者球树会小，层数较浅，建树时间较短。默认是30。

请大家根据线性回归的实现和KNN的参数说明，训练一个KNN模型。代码填写:

```
In [17]: #初始化knn模型
knn_model = KNeighborsRegressor(n_neighbors=50)
#训练
knn_model.fit(X_train,y_train)
#训练集上的MSE
knn_pred_train = knn_model.predict(X_train)
knn_mse_train = mean_squared_error(knn_pred_train,y_train)
#输出测试集上的测试结果
knn_pred_test=knn_model.predict(X_test)
knn_mse_test = mean_squared_error(knn_pred_test,y_test)
```

```
In [18]: print("训练集MSE: ", knn_mse_train)
print("测试集MSE: ", knn_mse_test)
```

```
训练集MSE: 0.2211942919921875
测试集MSE: 0.22388142578124998
```

模型评估与选择实例

输出表格和条形图对三个模型的MSE进行对比

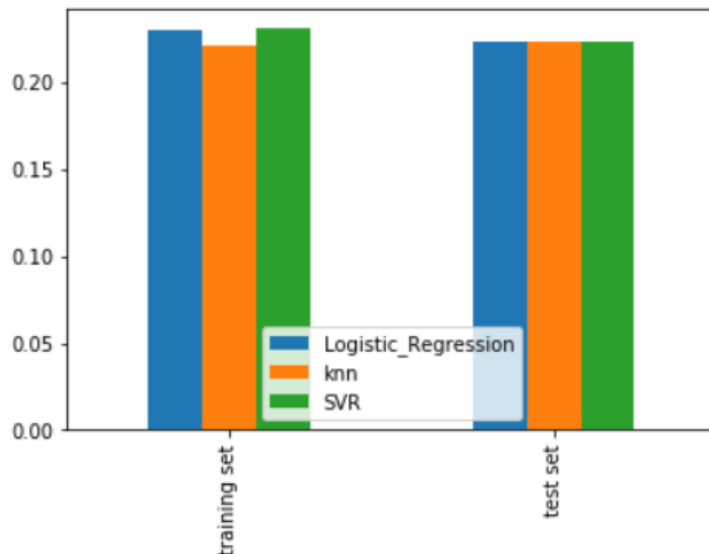
```
In [20]: model_mse = pd.DataFrame(data=[[linreg_mse_train,knn_mse_train,svr_mse_train],  
                                         [linreg_mse_test,knn_mse_test,svr_mse_test]],  
                                   columns=['Logistic_Regression','knn','SVR'],index=["training set","test set"])  
model_mse
```

Out[20]:

	Logistic_Regression	knn	SVR
training set	0.230032	0.221194	0.231046
test set	0.223108	0.223881	0.223756

```
In [21]: plt.figure(figsize=(20, 10))  
model_mse.plot(kind = 'bar')
```

<Figure size 1440x720 with 0 Axes>



模型评估与选择实例

下面我们将数据的标签变为整型，由于APP分数为1-5分，因此按照1、2、3、4、5这5个分数等级对APP评分进行分类预测，训练一个决策树分类模型。首先利用`astype(int)`函数将APP评分中带有小数的部分去除，得到整数评分，例如4.3分在经过函数处理后将变为4分，下面是对训练集和测试集的评分标签进行取整处理的代码以及处理后训练集前5条数据的标签的输出结果：

下面我们将数据的标签变为整型，训练一个决策树分类模型。

```
In [22]: y_train_int = y_train.astype(int)
         y_test_int = y_test.astype(int)
         y_train_int.head()
```

```
Out[22]: 2077    4
         4387    3
         8974    4
         8189    4
         6541    4
         Name: Rating, dtype: int32
```

模型评估与选择实例

决策树分类模型的模型参数说明如下：

决策树分类

`DecisionTreeClassifier(criterion="mse", splitter="best", max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0., max_features=None, random_state=None, max_leaf_nodes=None)`

`criterion`: 切分质量的评价准则。默认为'mse' (mean squared error)。

`splitter`: 指定了在每个节点切分的策略。有两种切分策略：

- (1). `splitter='best'`: 表示选择最优的切分特征和切分点。
- (2). `splitter='random'`: 表示随机切分。

`max_depth`: 指定树的最大深度。如果为None, 则表示树的深度不限, 直到每个叶子都是纯净的。

`min_samples_split`: 默认为2。它指定了分裂一个内部节点(非叶子节点)需要的最小样本数。如果为浮点数(0到1之间), 最少样本分割数为 $\text{ceil}(\text{min_samples_split} * \text{n_samples})$

`min_samples_leaf`: 指定了每个叶子节点包含的最少样本数。如果为浮点数(0到1之间), 每个叶子节点包含的最少样本数为 $\text{ceil}(\text{min_samples_leaf} * \text{n_samples})$

`min_weight_fraction_leaf`: 指定了叶子节点中样本的最小权重系数。默认情况下样本有相同的权重。

`max_feature`:

- (1). 如果是整数, 则每次节点分裂只考虑`max_feature`个特征。
- (2). 如果是浮点数(0到1之间), 则每次分裂节点的时候只考虑 $\text{int}(\text{max_features} * \text{n_features})$ 个特征。
- (3). 如果是字符串'auto', `max_features=n_features`。
- (4). 如果是字符串'sqrt', `max_features=sqrt(n_features)`。
- (5). 如果是字符串'log2', `max_features=log2(n_features)`。
- (6). 如果是None, `max_feature=n_feature`。

`random_state`: 随机数生成器

`max_leaf_nodes`:

- (1). 如果为None, 则叶子节点数量不限。
- (2). 如果不为None, 则`max_depth`被忽略。

模型评估与选择实例

决策树分类模型的建立以及模型准确率的计算:

```
In [23]: from sklearn.tree import DecisionTreeClassifier
```

```
In [24]: #初始化决策树模型
dtree=DecisionTreeClassifier(max_depth=8, min_samples_leaf=5, random_state=42)
#训练
dtree.fit(X_train,y_train_int)
#训练集上的准确率
dtree_pred_train = dtree.predict(X_train)
dtree_mse_train = dtree.score(X_train,y_train_int)
#输出测试集上的测试结果
dtree_pred_test=dtree.predict(X_test)
dtree_mse_test =dtree.score(X_test,y_test_int)
dtree_pred_train[0:50]
```

```
Out[24]: array([4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4,  
                4, 4, 4, 5, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,  
                4, 4, 4, 4, 4, 4])
```

```
In [25]: print("训练集accuracy_score: ", dtree_mse_train)
print("测试集accuracy_score: ", dtree_mse_test)
```

训练集accuracy_score: 0.793701171875
测试集accuracy_score: 0.7900390625

模型评估与选择实例

决策树分类模型的混淆矩阵绘制

```
from sklearn.metrics import confusion_matrix #导入混淆矩阵函数

cm = confusion_matrix(y_test_int, dtree_pred_test) #混淆矩阵

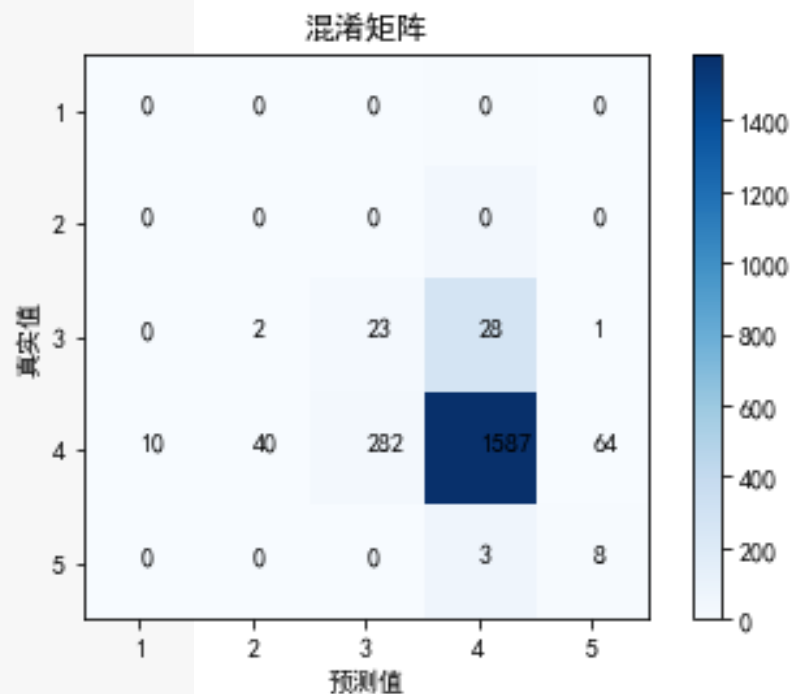
import matplotlib.pyplot as plt #导入作图库
# 热度图, 后面是指定的颜色块, 可设置其他的不同颜色
plt.imshow(cm, cmap=plt.cm.Blues)
# ticks 坐标轴的坐标点
# label 坐标轴标签说明
indices = range(len(cm))
# 第一个是迭代对象, 表示坐标的显示顺序, 第二个参数是坐标轴显示列表
# plt.xticks(indices, [0, 1, 2])
# plt.yticks(indices, [0, 1, 2])
plt.xticks(indices, ['1', '2', '3', '4', '5'])
plt.yticks(indices, ['1', '2', '3', '4', '5'])

plt.colorbar()

plt.xlabel('预测值')
plt.ylabel('真实值')
plt.title('混淆矩阵')

# plt.rcParams两行是用于解决标签不能显示汉字的问题
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 显示数据
for first_index in range(len(cm)): #第几行
    for second_index in range(len(cm[first_index])): #第几列
        plt.text(first_index, second_index, cm[first_index][second_index])
# 在matlab里面可以对矩阵直接imagesc(confusion)
# 显示
plt.show()
```

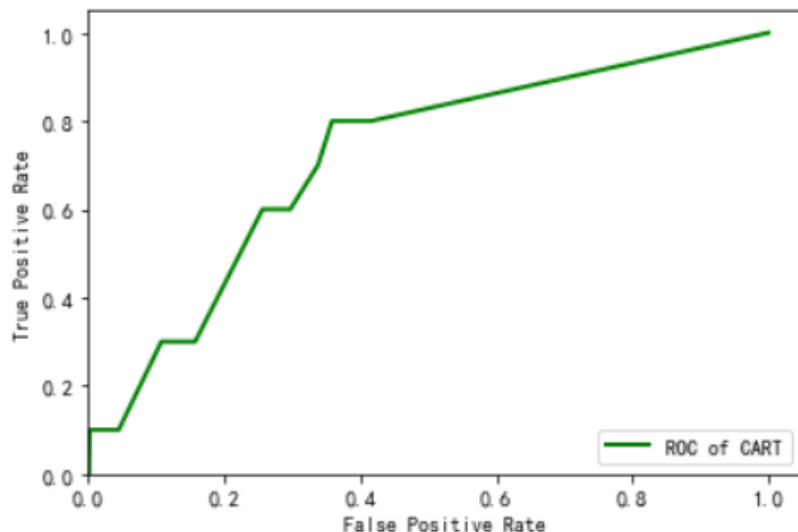


模型评估与选择实例

由于实例中的决策树分类模型是多分类模型，这里假设评分标签为1的样本为正样本，其余分数的为反样本，进行ROC曲线绘制的演示过程如下：

```
from sklearn.metrics import roc_curve #导入ROC曲线函数
import matplotlib.pyplot as plt #导入作图库
fpr, tpr, thresholds = roc_curve(y_test_int, dtree.predict_proba(X_test)[: ,0], pos_label=1)
plt.plot(fpr, tpr, linewidth=2, label = 'ROC of CART', color = 'green') #作出ROC曲线
plt.xlabel('False Positive Rate') #坐标轴标签
plt.ylabel('True Positive Rate') #坐标轴标签
plt.ylim(0,1.05) #边界范围
plt.xlim(0,1.05) #边界范围
plt.legend(loc=4) #图例
plt.show() #显示作图结果
```

predict_proba返回的是一个 n 行 k 列的数组，第 i 行第 j 列上的数值是模型预测 第 i 个预测样本为某个标签的概率，并且每一行的概率和为1。



pos_label: 正样本的标签

模型评估与选择实例

下面用是采用交叉验证来搜索KNN模型中的n_neighbors的最佳参数值的代码实例：

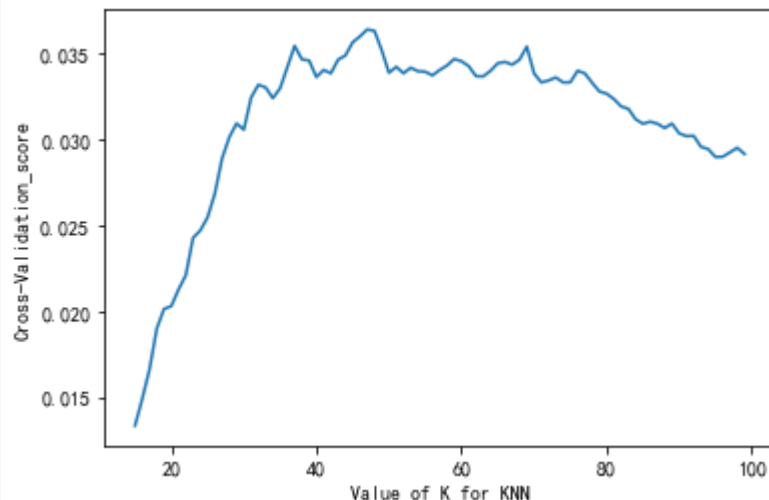
```
from sklearn.model_selection import cross_val_score # K折交叉验证模块

#建立测试参数集
k_range = range(15, 100)

k_scores = []

#藉由迭代的方式来计算不同参数对模型的影响，并返回交叉验证后的平均准确率
for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10)#r2 score
    k_scores.append(scores.mean())

#可视化数据
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validation_score')
plt.show()
```



cross_val_score这个方法是对数据进行多次分割，然后训练多个模型并评分，每次分割不一样。之后我们用评分的均值来代表这个模型的得分。方法重要参数是：cv代表计算多少次，分割次数；scoring代表方法。

输出结果如右上图所示，可知n_neighbors的数值在47左右最佳，大于此值后模型的表现没有明显提升。

模型评估与选择实例

下面用是采用网格搜索，对决策树分类器的超参数进行搜索的实例：

```
###1、决策树分类器
from sklearn.model_selection import GridSearchCV

params = [{'criterion':['gini'],'max_depth':[30,50,60,100],'min_samples_leaf':[2,3,5,10],'min_impurity_decrease':[0.1,0.2,0.5]},
          {'criterion':['gini','entropy']},
          {'max_depth': [30,60,100], 'min_impurity_decrease':[0.1,0.2,0.5]}]

best_model = GridSearchCV(dtree, param_grid=params,cv = 5,scoring ="accuracy")
best_model.fit(X_train,y_train_int)
print('最优分类器:',best_model.best_params_,'最优分数:', best_model.best_score_) # 得到最优的参数和分值

最优分类器: {'criterion': 'gini', 'max_depth': 30, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2} 最优分数: 0.7819824470831181
```

param_grid参数是需要最优化的参数的取值，值为字典或者列表，例如：param_grid = param_test1, param_test1 = {'n_estimators':range(10,71,10)}

cv：交叉验证参数，默认None，使用三折交叉验证。指定fold数量，默认为3，也可以是yield训练/测试数据的生成器。

scoring代表模型评价标准，默认None,这时需要使用score函数；或者如scoring='roc_auc'，根据所选模型不同，评价准则不同。

模型评估与选择实例

下面用是随机搜索来搜索SVM回归模型中的参数值的代码实例：

```
from sklearn.model_selection import RandomizedSearchCV

params_svr = {'kernel': ['rbf'], 'C': np.logspace(-3, 2, 6), 'gamma': np.arange(0,10,2)}
best_svr_model = RandomizedSearchCV(svr, param_distributions=params_svr, cv =
3, scoring = "neg_mean_squared_error")
best_svr_model.fit(X, Y)
print('最优分类器:', best_svr_model.best_params_, '最优分数:',
best_svr_model.best_score_) # 得到最优的参数和分值
```

输出结果如下：

最优分类器: {'kernel': 'rbf', 'gamma': 4, 'C': 0.01} 最优分数: -0.24271347471983348

本章小结:

1. 统计学习方法三要素：模型，策略，算法。
2. 在机器学习中，对模型进行选择或者说提高学习泛化能力是一个重要问题。如果只考虑减少训练误差，就可以产生过拟合现象。模型选择的常见方法有正则化、交叉验证、自助法。
3. 偏差-方差：偏差越大，越偏离真实数据；方差越大，数据的分布越分散。
4. 模型性能度量：分类器的混淆矩阵、查全率 R 、查准率 P ；ROC,AUC。
5. 超参数的搜索：网络搜索，随机搜索。

本章主要参考文献:

[01] 周志华. 机器学习: 第2章, 清华版, 2016

[02] 李航. 统计学习方法(第2版): 第1章, 清华版, 2019

End of this lecture.
Thanks!