机器学习篇

# 基础问题

## 1．什么是机器学习过拟合？

过拟合一般代表模型在训练集数据上学习的很好，但是测试集上表现却有所下降，有点矫枉过正的意思，就是模型泛化能力差

## 2．如何避免过拟合问题？

1. **重采样（增加训练数据，这样可以让模型学到更多的有效特征，从而降低数据噪声对模型的影响，如果没有足够的训练数据一般通过重采样进行样本扩充，但是在我们实际工作中一般不采用这种方法）**
2. **L1，L2正则化（降低模型复杂度）**
3. **决策树剪枝**
4. **交叉验证**
5. **集成学习（是把多个模型集成到一起来作为共同的模型，可以降低单一模型的过拟合风险。如bagging方法）。**
6. **在个人工作中使用XGB遇到的过拟合问题，采用了正则化和交叉验证，依然没有效果，解决思路是分析特征的稳定性以及特征的缺失率，最终发现一些排名靠前的特征中有一些特征在训练集和测试集的缺失水平不同，导致过拟合，其实就是训练集测试集的样本分布式有偏的，最终的解决办法是选用命中率较为稳定的数据作为训练和测试集。**

## 3.什么是机器学习的欠拟合？

所谓欠拟合就是模型复杂度低或者数据集太小,对模型数据的拟合程度不高,因此模型在训练集上的效果就不好.

## 4. 如何避免欠拟合问题？

1.增加样本的数量

2.增加样本特征的个数

3 增加模型复杂度；

模型简单时其表达能力较差，容易导致欠拟合，因此可以适当地增加模型复杂度，使模型拥有更强的拟合能力。如线性模型中添加高次项，神经网络中增加网络层数或神经元个数。

4. 减小正则化系数。

正则化是用于防止过拟合的，但是当出现欠拟合时，就有必要针对性地减小正则化系数。

## 3．交叉验证 ？

#### 什么是交叉验证法？

它的基本思想就是将原始数据（dataset）进行分组，一部分做为训练集来训练模型，另一部分做为测试集来评价模型。

#### 为什么用交叉验证法？

1. 交叉验证用于评估模型的预测性能，尤其是训练好的模型在新数据上的表现，可以在一定程度上减小过拟合。

2. 还可以从有限的数据中获取尽可能多的有效信息。

#### 主要有哪些方法？

1. 留出法 （holdout cross validation）

在机器学习任务中，拿到数据后，我们首先会将原始数据集分为三部分：训练集、验证集和测试集。训练集用于训练模型，验证集用于模型的参数选择配置，测试集对于模型来说是未知数据，用于评估模型的泛化能力。

这个方法操作简单，只需随机把原始数据分为三组即可。不过如果只做一次分割，它对训练集、验证集和测试集的样本数比例，还有分割后数据的分布是否和原始数据集的分布相同等因素比较敏感，不同的划分会得到不同的最优模型，而且分成三个集合后，用于训练的数据更少了

2. k 折交叉验证（k-fold cross validation）加以改进

k 折交叉验证通过对 k 个不同分组训练的结果进行平均来减少方差，因此模型的性能对数据的划分就不那么敏感。

• 第一步，不重复抽样将原始数据随机分为 k 份。

• 第二步，每一次挑选其中 1 份作为测试集，剩余 k-1 份作为训练集用于模型训练。

• 第三步，重复第二步 k 次，这样每个子集都有一次机会作为测试集，其余机会作为训练集。

• 在每个训练集上训练后得到一个模型，

• 用这个模型在相应的测试集上测试，计算并保存模型的评估指标，

• 第四步，计算 k 组测试结果的平均值作为模型精度的估计，并作为当前 k 折交叉验证下模型的性能指标。

k 一般取 10，数据量小的时候，k 可以设大一点，这样训练集占整体比例就比较大，不过同时训练的模型个数也增多。数据量大的时候，k 可以设小一点。

3. 留一法（Leave one out cross validation），每次的测试集都只有一个样本，要进行 m 次训练和预测。Bootstrapping： 通过自助采样法，即在含有 m 个样本的数据集中，每次随机挑选一个样本，再放回到数据集中，再随机挑选一个样本，这样有放回地进行抽样 m 次，组成了新的数据集作为训练集。这个方法用于训练的数据只比整体数据集少了一个样本，因此最接近原始样本的分布。但是训练复杂度增加了，因为模型的数量与原始数据样本数量相同。一般在数据缺乏时使用。

4 Bootstrapping： 通过自助采样法，即在含有 m 个样本的数据集中，每次随机挑选一个样本，再放回到数据集中，再随机挑选一个样本，这样有放回地进行抽样 m 次，组成了新的数据集作为训练集。这里会有重复多次的样本，也会有一次都没有出现的样本，原数据集中大概有 36.8% 的样本不会出现在新组数据集中。优点是训练集的样本总数和原数据集一样都是 m，并且仍有约 1/3 的数据不被训练而可以作为测试集。缺点是这样产生的训练集的数据分布和原数据集的不一样了，会引入估计偏差。此种方法不是很常用，除非数据量真的很少。

## 4. 偏差方差均衡 ？

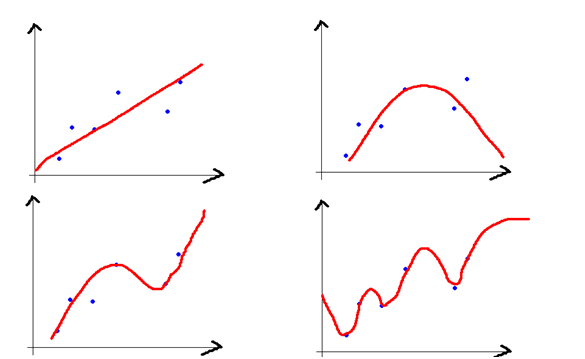
偏差（bias）是指预测结果与真实值之间的差异，排除噪声的影响，偏差更多的是针对某个模型输出的样本误差，偏差是模型无法准确表达数据关系导致，比如模型过于简单，非线性的数据关系采用线性模型建模，偏差较大的模型是错的模型

模型方差（variance）不是针对某一个模型输出样本进行判定，而是指多个(次)模型预测的结果之间的离散差异，注意这里写的是多个模型或者多次模型，即不同模型或同一模型不同时间的输出结果方差较大，方差是由训练集的数据不够导致，一方面量 (数据量) 不够，有限的数据集过度训练导致模型复杂，另一方面质(样本质量)不行，测试集中的数据分布未在训练集中，导致每次抽样训练模型时，每次模型参数不同，输出的结果都无法准确的预测出正确结果。

我们利用线性回归讨论一下这个问题

 回归最简单的定义是，给出一个点集D，用一个函数去拟合这个点集，并且使得点集与拟合函数间的误差最小。

最小二乘法是线性回归中一个最简单的方法，它的推导有一个假设，就是**回归函数的估计值与真实值间的误差假设是一个高斯分布**。这个用公式来表示是下面的样子，y(x,w)就是给定了w系数向量下的回归函数的估计值，而t就是真实值了，ε表示误差，可以回过头去看看，由于最小二乘法有这样一个假设，则会导致，如果我们给出的估计函数y(x,w)与真实值t不是高斯分布的，甚至是一个差距很大的分布，那么算出来的模型一定是不正确的，当给定一个新的点x’想要求出一个估计值y’，与真实值t’可能就非常的远了。 概率分布是一个可爱又可恨的东西，当我们能够准确的预知某些数据的分布时，那我们可以做出一个非常精确的模型去预测它，但是在大多数真实的应用场景中，数据的分布是不可知的，我们也很难去用一个分布、甚至多个分布的混合去表示数据的真实分布，比如说给定了1亿篇网页，希望用一个现有的分布（比如说混合高斯分布）去匹配里面词频的分布，是不可能的。在这种情况下，我们只能得到词的出现概率，比如p(的)的概率是0.5，也就是一个网页有1/2的概率出现“的”。如果一个算法，是对里面的分布进行了某些假设，那么可能这个算法在真实的应用中就会表现欠佳。**最小二乘法对于类似的一个复杂问题，就很无力了。**



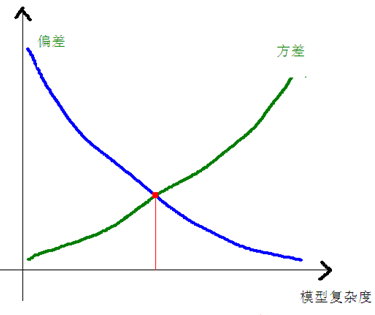
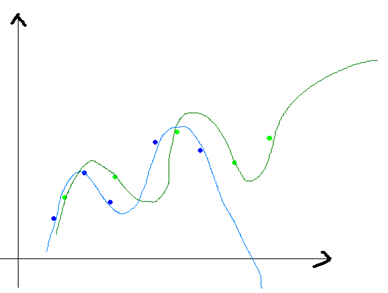
上图所示，给出一个点集(x,y), 需要用一个函数去拟合这个点集，蓝色的点是点集中的点，而红色的曲线是函数的曲线，第一张图是一个最简单的模型，对应的函数为y = f(x) = ax + b，这个就是一个线性函数，

    第二张图是二次曲线，对应的函数是y = f(x) = ax^2 + b。

    第三张图我也不知道是什么函数，瞎画的。

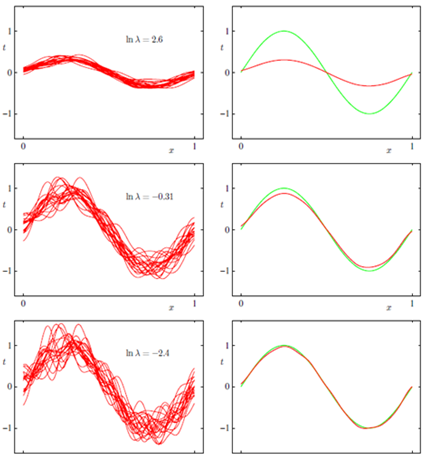
    第四张图可以认为是一个N次曲线，N = M - 1，M是点集中点的个数，有一个定理是，对于给定的M个点，我们可以用一个M - 1次的函数去完美的经过这个点集。

对于图一来说，训练的模型非常简单，导致他的表达也很简单，最终的预测结果与真实数据之间的误差也就是所谓的偏差较大，但是反观图4，从模型的预测结果看拟合的非常好，模型很复杂，偏差很小，但是这是明显的过拟合，这时候用同一批训练数据训练的多个模型的预测结果之间的方差却很大，而对于图一中训练的多个简单模型给出的预测结果之间的方差就很小了。方差和偏差的变化一般是和模型的复杂程度成正比的，当我们一味的追求模型精确匹配，则可能会导致同一组数据训练出不同的模型，它们之间的差异非常大。这就叫做方差，不过他们的偏差就很小了，如下图所示



上图的蓝色和绿色的点是表示一个数据集中采样得到的不同的子数据集，我们有两个N次的曲线去拟合这些点集，则可以得到两条曲线（蓝色和深绿色），它们的差异就很大，但是他们本是由同一个数据集生成的，这个就是模型复杂造成的方差大。模型越复杂，偏差就越小，而模型越简单，偏差就越大，方差和偏差是按下面的方式进行变化的。

 当方差和偏差加起来最优的点，就是我们最佳的模型复杂度。



这是一个曲线拟合的问题，对同分布的不同的数据集进行了多次的曲线拟合，左边表示方差，右边表示偏差，绿色是真实值函数。ln lambda表示模型的复杂程度，这个值越小，表示模型的复杂程度越高，在第一行，大家的复杂度都很低的时候，方差是很小的，但是偏差很大，但是到了最后一幅图，我们可以得到，每个人的复杂程度都很高的情况下，不同的函数就有着天壤之别了，但是偏差就很小了。

## 5．机器学习中模型参数和超参的区别？超参给定的方式有那些？

模型参数是模型内部的配置变量，可以用数据估计模型参数的值；

模型超参是模型外部的配置，必须手动设置参数的值。

模型超参可以通过网格交叉验证来取值,也可以通过经验选定.

# 距离相关

## 1常见的距离度量公式有那些？

1. Minkowski(闵可夫斯基距离)

一般而言，定义一个距离函数 d(x,y), 需要满足下面几个准则：

1. d(x,x) = 0 // 到自己的距离为0
2. d(x,y) >= 0 // 距离非负
3. d(x,y) = d(y,x) // 对称性: 如果 A 到 B 距离是 a，那么 B 到 A 的距离也应该是 a
4. d(x,k) d(k,y) >= d(x,y) // 三角形法则: (两边之和大于第三边)

## 闵可夫斯基距离（Minkowski distance）

是衡量数值点之间距离的一种非常常见的方法，假设数值点 P 和 Q 坐标如下：  
那么，闵可夫斯基距离定义为



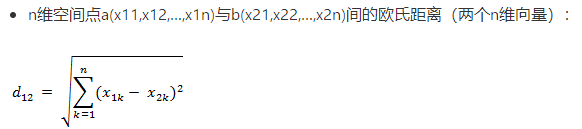
--- P=1:曼哈顿距离(城市距离)

---P=2:欧式距离

---P无穷:切比雪夫距离

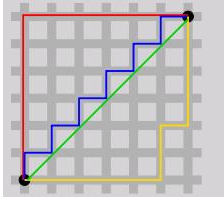
## 欧氏距离

是最容易直观理解的距离度量方法，我们小学、初中和高中接触到的两个点在空间中的距离一般都是指欧氏距离

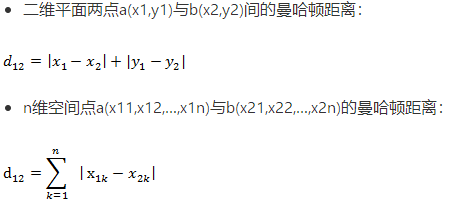


## 曼哈顿距离(Manhattan Distance)

假设在曼哈顿街区乘坐出租车从 P 点到 Q 点，白色表示高楼大厦，灰色表示街道：

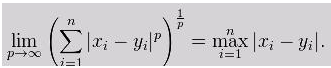


绿色的斜线表示欧几里得距离，在现实中是不可能的。其他三条折线表示了曼哈顿距离，这三条折线的长度是相等的。



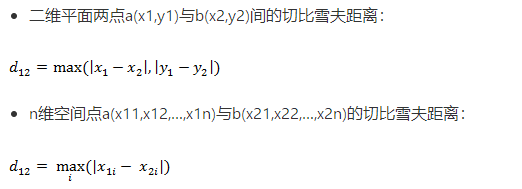
## 切比雪夫距离（Chebyshev distance

当 p 趋近于无穷大时，闵可夫斯基距离转化成切比雪夫距离（Chebyshev distance）：



我们知道平面上到原点欧几里得距离（p = 2）为 1 的点所组成的形状是一个圆，当 p 取其他数值的时候呢？



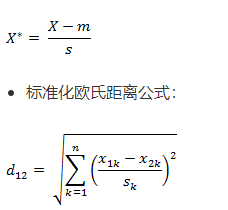


闵可夫斯基距离比较直观，但是它与数据的分布无关，具有一定的局限性，如果 x 方向的幅值远远大于 y 方向的值，这个距离公式就会过度放大 x 维度的作用。

* 闵氏距离，包括曼哈顿距离、欧氏距离和切比雪夫距离都存在明显的缺点。
* e.g. 二维样本(身高[单位:cm],体重[单位:kg]),现有三个样本：a(180,50)，b(190,50)，c(180,60)。那么a与b的闵氏距离（无论是曼哈顿距离、欧氏距离或切比雪夫距离）等于a与c的闵氏距离。但实际上身高的10cm并不能和体重的10kg划等号。
* 闵氏距离的缺点：
* (1)将各个分量的量纲(scale)，也就是“单位”相同的看待了;
* (2)未考虑各个分量的分布（期望，方差等）可能是不同的。

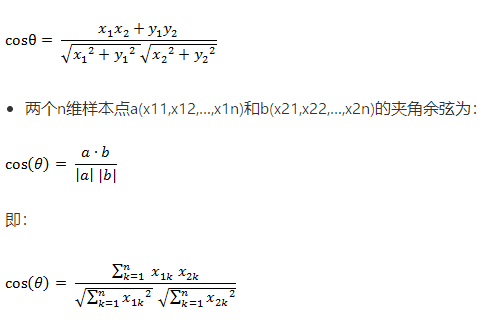
## 标准化欧氏距离 (Standardized Euclidean Distance)

定义： 标准化欧氏距离是针对欧氏距离的缺点而作的一种改进。标准欧氏距离的思路：既然数据各维分量的分布不一样，那先将各个分量都“标准化”到均值、方差相等。假设样本集X的均值(mean)为m，标准差(standard deviation)为s，X的“标准化变量”表示为：



## 余弦距离(Cosine Distance)

几何中，夹角余弦可用来衡量两个向量方向的差异；机器学习中，借用这一概念来衡量样本向量之间的差异。二维空间中向量A(x1,y1)与向量B(x2,y2)的夹角余弦公式：



夹角余弦取值范围为[-1,1]。余弦越大表示两个向量的夹角越小，余弦越小表示两向量的夹角越大。当两个向量的方向重合时余弦取最大值1，当两个向量的方向完全相反余弦取最小值-1。余弦相似度，就是计算两个向量间的夹角的余弦值。具体的余弦距离。余弦距离就是用1减去这个获得的余弦相似度。由上面的余弦距离可以知道，余弦距离的取值范围为[0,2] ,这就满足了非负性的性质。

## 什么时候用余弦距离什么时候用欧式距离呢？

总体来说，欧氏距离体现数值上的绝对差异，而余弦距离体现方向上的相对差异。

1）例如，统计两部剧的用户观看行为，用户A的观看向量为(0,1)，用户B为(1,0)；此时二者的余弦距很大，而欧氏距离很小；我们分析两个用户对于不同视频的偏好，更关注相对差异，显然应当使用余弦距离。

2）而当我们分析用户活跃度，以登陆次数(单位：次)和平均观看时长(单：分钟)作为特征时，余弦距离会认为(1,10)、(10,100)两个用户距离很近；但显然这两个用户活跃度是有着极大差异的，此时我们更关注数值绝对差异，应当使用欧氏距离。

    3) KL距离(相对熵)

    4) 杰卡德相似度系数(**集合**)

1. Pearson相关系数ρ(距离=  1**-ρ**)

# 优化算法

## 1．常见的优化算法有那些，以及这些优化算法有那些优缺点?

优化是应用数学的一个分支，也是机器学习的核心组成部分。有文章[1] 认为，

机器学习算法 =模型表征+ 模型评估+ 优化算法

其中优化算法所做的事情就是在模型假设空间中找到模型评估指标最好的模型。不同的模型假设、不同的评估指标，对应的优化算法都不尽相同，例如经典的SVM（线性分类模型+ 最大间隔）、逻辑回归（线性分类模型+ 交叉熵）、CART（决策树模型+ 基尼纯度）等

1)梯度下降法(Gradient Descent)

---BGD--->每一步迭代都需要遍历所有的样本数据,消耗时间长,但是一定能得到最优解.

---SGD--->迭代速度快,得到局部最优解(凸函数时得到全局最优解)

---MBGD--->小批量梯度下降法

2)牛顿法和拟牛顿法

    牛顿法--->牛顿法是**二阶收敛**,**收敛速度快**; 牛顿法是一种迭代算法，每一步都需要求解目标函数的Hessian(海森)矩阵的逆矩阵，**计算比较复杂**。

拟牛顿法--->改善牛顿法每次需要求解复杂的Hessian矩阵的逆矩阵的缺陷，它**使用正定矩阵来近似Hessian矩阵的逆，从而简化了运算的复杂度**.

3) 启发式的方法

启发式优化方法种类繁多，包括经典的**模拟退火方法**、**遗传算法**、**蚁群算法**以及**粒子群算法**等。

# 文本处理

## 1．文本数据处理的方式有哪些，以及这些处理方式有什么区别

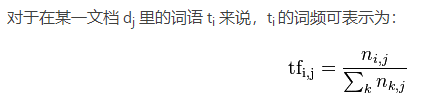
1) **词袋法**(BOW/TF)\**词集法**(SOW)

不考虑文本的语法和语序,只考虑单词存在的次数(BOW/TF)或者是否存在(SOW)

2) **TF-IDF**

既考虑文本的词频,也考虑文件的**逆文档频率(基本思想**是:单词的重要性**与单词在文档中出现的次数成正比**,**与单词在语料库中出现的次数成反比)**

TF（Term Frequency，词频）表示一个给定词语t在一篇给定文档d中出现的频率。TF越高，则词语t对文档d来说越重要，TF越低，则词语t对文档d来说越不重要。那是否可以以TF作为文本相似度评价标准呢？答案是不行的，举个例子，常用的中文词语如“我”，“了”，“是”等，在给定的一篇中文文档中出现的频率是很高的，但这些中文词几乎在每篇文档中都具有非常高的词频，如果以TF作为文本相似度评价标准，那么几乎每篇文档都能被命中。



IDF（Inverse Document Frequency，逆向文件频率）

的主要思想是：如果某个词语t的在整个文档库中出现次数越少，则IDF越大，说明词语t在整个文档集层面上具有很好的类别区分能力。IDF说明了什么问题呢？还是举个例子，常用的中文词语如“我”，“了”，“是”等在每篇文档中几乎具有非常高的词频，那么对于整个文档集而言，这些词都是不重要的。对于整个文档集而言的，评价词语重要性的标准就是IDF。

不足之处，TFIDF算法是建立在这样一个假设之上的：对区别文档最有意义的词语应该是那些在文档中出现频率高，而在整个文档集合的其他文档中出现频率少的词语。所以如果特征空间坐标系取TF词频作为测度，就可以体现同类文本的特点。另外考虑到单词区别不同类别的能力，TF-IDF法认为一个单词出现的文本频数（即包含某个单词的文本数）越小，它区别不同类别文本的能力就越大。因此引入了逆文本频度IDF的概念，以TF和IDF的乘积作为特征空间坐标系的取值测度，并用它完成对权值TF的调整，调整权值的目的在于突出重要单词，抑制次要单词。但是在本质上IDF是一种试图抑制噪声的加权，并且单纯地认为文本频率小的单词就越重要，文本频率大的单词就越无用，显然这并不是完全正确的。IDF的简单结构并不能有效地反映单词的重要程度和特征词的分布情况，使其无法很好地完成对权值调整的功能，所以TF-IDF法的精度并不是很高

**Word2Vec (通过低维稠密的向量形式表达词信息，可以学习到词的上下文信息)**

# 特征工程

## 在数据处理过程中，对于缺失特征的样本如何进行处理？

根据样本缺失的实际情况,我们一般运用:

        1)均值,中值,最大最小值等来填充数据

        2)根据经验值补全数据（实际不可操作）

        3)通过相关计算得到缺失值

        4)样本数量足够,则可以直接删除有缺失值的样本

如果某些特征有缺失，一般的处理方法是，如果缺失率较高但是比较有效的特征予以保留，其他的予以删除。

## 2. 续性数据分区转换为离散数据有什么优点？

1)离散特征的增加和减少都很容易,**易于模型的快速迭代**

2)离散化后的特征对异常数据具有**很强的鲁棒性**

3)离散化后可以进行特征交叉,相当于**引入非线性,提升模型的表达能力**

**缺点也是存在的，有可能损失有效的样本信息，导致特征表达能力下降。**

## 数据标准化、数据归一化、区间缩放法有什么作用以及各个方法之间有什么区别？

数据标准化是将数据集转化为服从标准正态分布的数据.(常用于离散程度较高的数据)

数据归一化是将每个样本的向量转化为单位向量(矩阵的行操作)

区间缩放法是对于分布较大的数据集,通过等比缩放的方法缩放到feature\_range=(0,1).

# 决策树算法

## 决策树的常用算法有那些，这些算法有什么区别？

常用的有:ID3,C4.5,CART三种算法

区别:

1) 纯度量化指标不同.ID3-->**信息增益**,C4.5-->**信息增益率**,CART-->**基尼系数**

2) 数据处理能力不同. ID3-->**离散数据**,C4.5,CART--->**连续数据离散化,剪枝操作**

3)要求的树的类型不同.**ID3和C4.5可以是多叉树,CART只能是二叉树.**

## 2 .决策树算法中如何避免过拟合和欠拟合？

过拟合:选择能够反映业务逻辑的训练集去产生决策树;剪枝操作(预剪枝和后剪枝); K折交叉验证

欠拟合:增加树的深度,bagging，boosting，stacking

## 3 回归决策树算法和分类决策树算法有什么区别与联系？

回归决策树解决的是回归的问题.分类决策树解决的是分类的问题.他们在决策树的构建上是一样的（最优划分）.回归决策树是计算预测点所在的叶子节点的样本的**均值**或者**加权平均值**得到回归结果.分类是通过**多数投票**或者**加权的多数投票**得到分类结果.

### 10. RF和GBDT的区别

**相同点：**

* 都是由多棵树组成，最终的结果都是由多棵树一起决定。

**不同点：**

* **集成学习**：RF属于bagging思想，而GBDT是boosting思想
* **偏差-方差权衡**：RF不断的降低模型的方差，而GBDT不断的降低模型的偏差
* **训练样本**：RF每次迭代的样本是从全部训练集中有放回抽样形成的，而GBDT每次使用全部样本
* **并行性**：RF的树可以并行生成，而GBDT只能顺序生成(需要等上一棵树完全生成)
* **最终结果**：RF最终是多棵树进行多数表决（回归问题是取平均），而GBDT是加权融合
* **数据敏感性**：RF对异常值不敏感，而GBDT对异常值比较敏感
* **泛化能力**：RF不易过拟合，而GBDT容易过拟合

# XGB算法相关

#### 1 简单的介绍xgb

首先需要说一说GBDT，它是一种基于boosting增强策略的加法模型，训练的时候采用前向分步算法进行贪婪的学习，每次迭代都学习一棵CART树来拟合之前 t-1 棵树的预测结果与训练样本真实值的残差。

XGBoost对GBDT进行了一系列优化，比如损失函数进行了二阶泰勒展开、目标函数加入正则项、支持并行和默认缺失值处理等，在可扩展性和训练速度上有了巨大的提升，但其核心思想没有大的变化。

### 2. XGBoost与GBDT有什么不同

* **基分类器**：XGBoost的基分类器不仅支持CART决策树，还支持线性分类器，此时XGBoost相当于带L1和L2正则化项的Logistic回归（分类问题）或者线性回归（回归问题）。
* **导数信息**：XGBoost对损失函数做了二阶泰勒展开，GBDT只用了一阶导数信息，并且XGBoost还支持自定义损失函数，只要损失函数一阶、二阶可导。
* **正则项**：XGBoost的目标函数加了正则项， 相当于预剪枝，使得学习出来的模型更加不容易过拟合。
* **列抽样**：XGBoost支持列采样，与随机森林类似，用于防止过拟合。
* **缺失值处理**：对树中的每个非叶子结点，XGBoost可以自动学习出它的默认分裂方向。如果某个样本该特征值缺失，会将其划入默认分支。
* **并行化**：注意不是tree维度的并行，而是特征维度的并行。XGBoost预先将每个特征按特征值排好序，存储为块结构，分裂结点时可以采用多线程并行查找每个特征的最佳分割点，极大提升训练速度。

### 3. XGBoost为什么使用泰勒二阶展开

* **精准性**：相对于GBDT的一阶泰勒展开，XGBoost采用二阶泰勒展开，可以更为精准的逼近真实的损失函数
* **可扩展性**：损失函数支持自定义，只需要新的损失函数二阶可导。

### 4. XGBoost为什么可以并行训练

* XGBoost的并行，并不是说每棵树可以并行训练，XGB本质上仍然采用boosting思想，每棵树训练前需要等前面的树训练完成才能开始训练。
* XGBoost的并行，指的是特征维度的并行：在训练之前，每个特征按特征值对样本进行预排序，并存储为Block结构，在后面查找特征分割点时可以重复使用，而且特征已经被存储为一个个block结构，那么在寻找每个特征的最佳分割点时，可以利用多线程对每个block并行计算，即对每个特征寻找切割点是可以并行计算的，每个特征采用常数个分位点作为候选分割点。

### 5. XGBoost为什么快

* **分块并行**：训练前每个特征按特征值进行排序并存储为Block结构，后面查找特征分割点时重复使用，并且支持并行查找每个特征的分割点
* **候选分位点**：每个特征采用常数个分位点作为候选分割点
* **CPU cache 命中优化**： 使用缓存预取的方法，对每个线程分配一个连续的buffer，读取每个block中样本的梯度信息并存入连续的Buffer中。
* **Block 处理优化**：Block预先放入内存；Block按列进行解压缩；将Block划分到不同硬盘来提高吞吐

### 6. XGBoost防止过拟合的方法

XGBoost在设计时，为了防止过拟合做了很多优化，具体如下：

* **目标函数添加正则项**：叶子节点个数+叶子节点权重的L2正则化
* **列抽样**：训练的时候只用一部分特征（不考虑剩余的block块即可）
* **子采样**：每轮计算可以不使用全部样本，使算法更加保守
* **shrinkage**: 可以叫学习率或步长，为了给后面的训练留出更多的学习空间

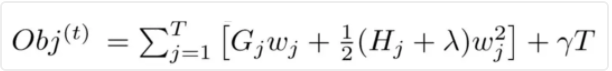
### 7. XGBoost如何处理缺失值

XGBoost模型的一个优点就是允许特征存在缺失值。对缺失值的处理方式如下：

* 在特征k上寻找最佳 split point 时，不会对该列特征 missing 的样本进行遍历，而只对该列特征值为 non-missing 的样本上对应的特征值进行遍历，通过这个技巧来减少了为稀疏离散特征寻找 split point 的时间开销。
* 在逻辑实现上，为了保证完备性，会将该特征值missing的样本分别分配到左叶子结点和右叶子结点，两种情形都计算一遍后，选择分裂后增益最大的那个方向（左分支或是右分支），作为预测时特征值缺失样本的默认分支方向。
* 如果在训练中没有缺失值而在预测中出现缺失，那么会自动将缺失值的划分方向放到右子结点。

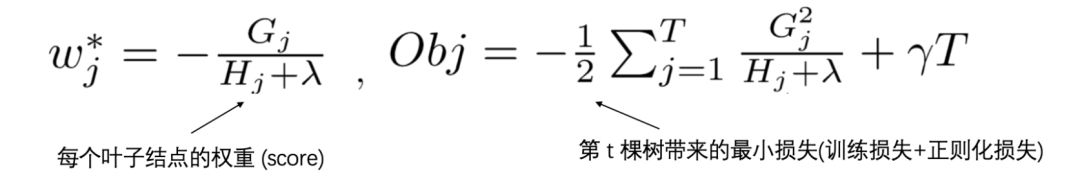
### 8. XGBoost中叶子结点的权重如何计算出来

XGBoost目标函数最终推导形式如下：



利用一元二次函数求最值的知识，当目标函数达到最小值Obj\*时，每个叶子结点的权重为wj\*。

具体公式如下：



### 9. XGBoost中的一棵树的停止生长条件

* 当新引入的一次分裂所带来的增益Gain<0时，放弃当前的分裂。这是训练损失和模型结构复杂度的博弈过程。
* 当树达到最大深度时，停止建树，因为树的深度太深容易出现过拟合，这里需要设置一个超参数max\_depth。
* 当引入一次分裂后，重新计算新生成的左、右两个叶子结点的样本权重和。如果任一个叶子结点的样本权重低于某一个阈值，也会放弃此次分裂。这涉及到一个超参数:最小样本权重和min\_child\_weight，
* 是指如果一个叶子节点包含的样本数量太少也会放弃分裂，防止树分的太细。

### 12. 比较LR和GBDT，说说什么情景下GBDT不如LR

先说说LR和GBDT的区别：

* LR是线性模型，可解释性强，很容易并行化，但学习能力有限，需要大量的人工特征工程
* GBDT是非线性模型，具有天然的特征组合优势，特征表达能力强，但是树与树之间无法并行训练，而且树模型很容易过拟合；
* 当在高维稀疏特征的场景下，LR的效果一般会比GBDT好。原因如下：
* 先看一个例子：
* 假设一个二分类问题，label为0和1，特征有100维，如果有1w个样本，但其中只要10个正样本1，而这些样本的特征 f1的值为全为1，而其余9990条样本的f1特征都为0(在高维稀疏的情况下这种情况很常见)。
* 我们都知道在这种情况下，树模型很容易优化出一个使用f1特征作为重要分裂节点的树，因为这个结点直接能够将训练数据划分的很好，但是当测试的时候，却会发现效果很差，因为这个特征f1只是刚好偶然间跟y拟合到了这个规律，这也是我们常说的过拟合。
* 那么这种情况下，如果采用LR的话，应该也会出现类似过拟合的情况呀：y = W1\*f1 + Wi\*fi+….，其中 W1特别大以拟合这10个样本。为什么此时树模型就过拟合的更严重呢？
* 仔细想想发现，因为现在的模型普遍都会带着正则项，而 LR 等线性模型的正则项是对权重的惩罚，也就是 W1一旦过大，惩罚就会很大，进一步压缩 W1的值，使他不至于过大。但是，树模型则不一样，树模型的惩罚项通常为叶子节点数和深度等，而我们都知道，对于上面这种 case，树只需要一个节点就可以完美分割9990和10个样本，一个结点，最终产生的惩罚项极其之小。
* 这也就是为什么在高维稀疏特征的时候，线性模型会比非线性模型好的原因了：**带正则化的线性模型比较不容易对稀疏特征过拟合。**

### 13. XGBoost中如何对树进行剪枝

* 在目标函数中增加了正则项：使用叶子结点的数目和叶子结点权重的L2模的平方，控制树的复杂度。
* 在结点分裂时，定义了一个阈值，如果分裂后目标函数的增益小于该阈值，则不分裂。
* 当引入一次分裂后，重新计算新生成的左、右两个叶子结点的样本权重和。如果任一个叶子结点的样本权重低于某一个阈值（最小样本权重和），也会放弃此次分裂。
* XGBoost 先从顶到底建立树直到最大深度，再从底到顶反向检查是否有不满足分裂条件的结点，进行剪枝。

### 14. XGBoost如何选择最佳分裂点？

XGBoost在训练前预先将特征按照特征值进行了排序，并存储为block结构，以后在结点分裂时可以重复使用该结构。  
因此，可以采用特征并行的方法利用多个线程分别计算每个特征的最佳分割点，根据每次分裂后产生的增益，最终选择增益最大的那个特征的特征值作为最佳分裂点。如果在计算每个特征的最佳分割点时，对每个样本都进行遍历，计算复杂度会很大，这种全局扫描的方法并不适用大数据的场景。XGBoost还提供了一种直方图近似算法，对特征排序后仅选择常数个候选分裂位置作为候选分裂点，极大提升了结点分裂时的计算效率。

### 15. XGBoost的Scalable性如何体现

* **基分类器的scalability**：弱分类器可以支持CART决策树，也可以支持LR和Linear。
* **目标函数的scalability**：支持自定义loss function，只需要其一阶、二阶可导。有这个特性是因为泰勒二阶展开，得到通用的目标函数形式。
* **学习方法的scalability**：Block结构支持并行化，支持 Out-of-core计算。

### 16. XGBoost如何评价特征的重要性

我们采用三种方法来评判XGBoost模型中特征的重要程度：

官方文档：

（1）weight - the number of times a feature is used to split the data across all trees.

（2）gain - the average gain of the feature when it is used in trees.

（3）cover - the average coverage of the feature when it is used in trees.

* **weight** ：该特征在所有树中被用作分割样本的特征的总次数。
* **gain** ：该特征在其出现过的所有树中产生的平均增益。
* **cover** ：该特征在其出现过的所有树中的平均覆盖范围。

注意：覆盖范围这里指的是一个特征用作分割点后，其影响的样本数量，即有多少样本经过该特征分割到两个子节点。

### 17. XGBooost参数调优的一般步骤

首先需要初始化一些基本变量，例如：

* max\_depth = 5
* min\_child\_weight = 1
* gamma = 0
* subsample, colsample\_bytree = 0.8
* scale\_pos\_weight = 1

**(1) 确定learning rate和estimator的数量**learning rate可以先用0.1，用cv来寻找最优的estimators**(2) max\_depth和 min\_child\_weight**

我们调整这两个参数是因为，这两个参数对输出结果的影响很大。我们首先将这两个参数设置为较大的数，然后通过迭代的方式不断修正，缩小范围。

max\_depth，每棵子树的最大深度，check from range(3,10,2)。

min\_child\_weight，子节点的权重阈值，check from range(1,6,2)。

如果一个结点分裂后，它的所有子节点的权重之和都大于该阈值，该叶子节点才可以划分。

**(3) gamma**也称作最小划分损失min\_split\_loss，check from 0.1 to 0.5，指的是，对于一个叶子节点，当对它采取划分之后，损失函数的降低值的阈值。

* 如果大于该阈值，则该叶子节点值得继续划分
* 如果小于该阈值，则该叶子节点不值得继续划分

**(4) subsample, colsample\_bytree**

subsample是对训练的采样比例

colsample\_bytree是对特征的采样比例

both check from 0.6 to 0.9

**(5) 正则化参数**

alpha 是L1正则化系数，try 1e-5, 1e-2, 0.1, 1, 100

lambda 是L2正则化系数

**(6) 降低学习率**降低学习率的同时增加树的数量，通常最后设置学习率为0.01~0.1

### 18. XGBoost模型如果过拟合了怎么解决

当出现过拟合时，有两类参数可以缓解：  
第一类参数：用于直接控制模型的复杂度。包括max\_depth,min\_child\_weight,gamma 等参数第二类参数：用于增加随机性，从而使得模型在训练时对于噪音不敏感。包括subsample,colsample\_bytree还有就是直接减小learning rate，但需要同时增加estimator 参数。

# 降维

## 常见的降维方式有那些？

L1惩罚模型,PCA(主成分分析法)和LDA(线性判别分析法)

PCA-->无监督的降维(无类别信息)-->选择方差大的方向投影,方差越大所含的信息量越大,信息损失越少.可用于特征提取和特征选择。

LDA-->有监督的降维(有类别信息)-->选择投影后的类别内的方差小,类别间的方差较大.

TSNE—>

## PCA降维的原理是什么？

## LDA线性判别分析降维的原理是什么？

# 回归

## Logistic回归和Linear线性回归有什么区别和联系？

# 算法评价指标

## 如何评价算法模型的效果？

分类算法评价:

准确率(Accuracy)

精确率(Precision)：TP/TP+FP, 精确率是针对我们预测结果而言的，它表示的是预测为正的样本中有多少是真正的正样本 更加关注 模型对于真正的正样本的预测能力

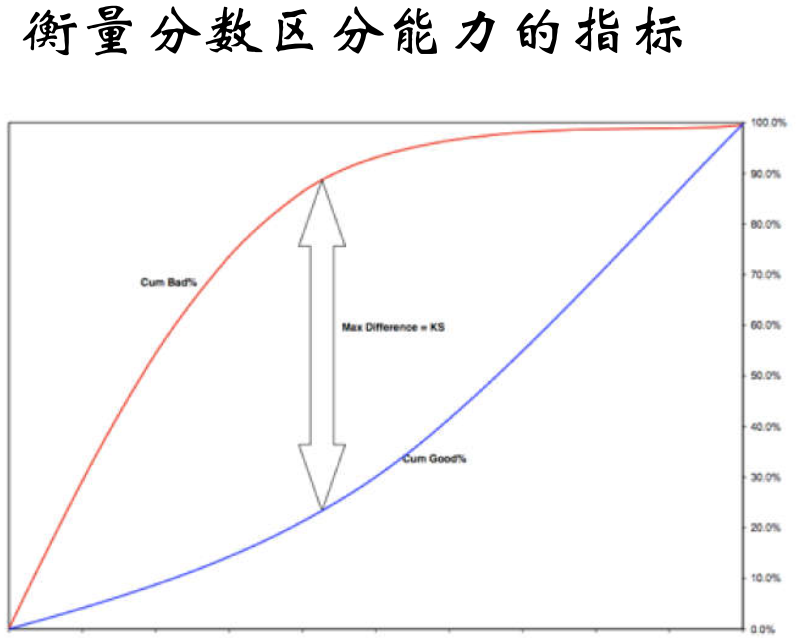
召回率(Recall) : TP/TP+FN 召回率其实是看在所有模型预测的正样本中 准确的是多少，召回了多少

F1值: 2TP / (2TP + FP + FN)混合的度量，对不平衡类别非常有效

是综合考虑了模型查准率和查全率的计算结果，取值更偏向于取值较小的那个指标。

为什么要出现这么多评估指标呢？实际上，不同的分类任务适合使用不同的指标来衡量。例如，推荐系统中，如果希望更精准的了解客户需求，避免推送用户不感兴趣的内容，因为推送的内容都是模型预测的正样本，就是TP+FP,故precision 就更加重要；在疾病检测的时候，我们不希望查漏任何一项疾病，这时 recall（TPR） 就更重要。

KS值：max(good/good sum - bad/bad sum) 模型区分度



这里关键是要知道X轴的定义，把样本按分数从低到高排序，就是把模型预测样本的概率值从高到低排序，得到有序的预测结果序列，X轴表征样本累计占比。试想有一批样本好坏比2:8 当样本累计预测到了50%，这50%都是概率值比较高的样本，然后我们去看Y轴 得到了这50%的样本中goodrate和badrate。一个区分能力较弱的模型，虽然都是概率值比较高的样本 但是预测的结果中，goodrate，badrate很接近，很明显的区分能力弱，如果区分能力强的话，这50%的样本中bad/bad sum 应该非常高，而good/good sum 又很低 因为这些样本均概率值是很高的，那么bad/bad sum good/good sum 离得越远，区分能力当然就越好。随着样本累计100%，bad/bad sum good/good sum 都接近1了，KS曲线可以很好的表征模型区分能力。

ROC曲线(AUC值)

回归算法评价: MAE,MSE,可解释方差的回归评分函数(explained\_variance\_score)

# 非监督学习算法

## 以聚类算法为例，假设没有外部标签数据，如何区分两个无监督学习（聚类）算法性的优劣呢？

数据的聚类依赖于需求的定义，同时也依赖于分类数据的特征度量以及数据相似性的方法。相比于监督式学习，非监督学习通常没有正确答案，算法模型的设计直接影响最终的输出和性能，需要通过多次迭代的方法寻找模型的最优的参数。因此了解常见数据簇的特点和常见聚类算法的特点，对寻求评估不同聚类算法性能的方法有很大的帮助。

## **常见数据簇的特点**：

以中心定义的数据簇：这类数据集合倾向于球形分布，通常中心被定义为质心，即此数据簇中所有点的平均值。集合中的数据到中心的距离相比到其它簇中心的距离更近，更加接近高斯分布；

以密度定义的数据簇：这类数据集合呈现和周围数据簇明显不同的密度，或稠密或稀疏。当数据簇不规则或互相盘绕，并且有噪声和离群点时，常常使用基于密度的簇定义；

以连通定义的数据簇：这类数据集合中的数据点和数据点之间有连接关系，整个数据簇表现为图结构，该定义对不规则形状或者缠绕的的数据簇有效；

以概念定义的数据簇：这类数据集合中的所有数据点具有某种共同性质。

## 常见的聚类算法的特点：

划分聚类：将数据对象划分成互不重叠的数据簇，其中每个数据点恰在一个数据簇中；

层次聚类：数据簇可以具有子簇，具有多个（嵌套）子簇的数据簇可以表示为树状结构；

模糊聚类：每个数据点均以0~1的隶属权值属于某个数据簇；

完全/不完全聚类：是否对所有数据点都指派一个数据簇。

由于数据以及需求的多样性，没有一种算法能够适应所有的数据类型、簇和应用，似乎每种情况都可能需要一种不同的评估度量。例如，K均值聚类通常需要用SSE (Sum of Square Error) 来评估，但是基于密度的数据簇可以不必是球形，SSE则完全失效。在许多情况下，判断聚类算法结果的好坏最终强烈依赖主观解释。尽管如此，聚类算法的评估还是必须的，它是聚类分析中重要部分之一 。

对聚类算法优劣的评估通常可以总结为对以下五个方面的分析：

辨识数据中是否存在非随机簇结构的能力；

辨识数据中正确数据簇的能力；

评估数据被正确聚类的能力；

辨识两个数据簇之间优劣的能力；

评估与客观数据集之间的差异；

# 朴素贝叶斯

## 为什么朴素贝叶斯如此“朴素”

因为它假定所有的特征在数据集中的作用是同样重要和独立的。正如我们所知，这个假设在现实世界中是很不真实的，因此，说朴素贝叶斯真的很“朴素”。

## 什么是朴素贝叶斯？

P(A|B) = P(B)\*P(A,B)

P(B|A) = P(A)\*P(A,B)

P(A|B) = P(B)\*P(B|A) / P(A) 全概率公式

因此可得

P(y|x) = P(x)\*P(x|y) / P(y)

P(y|x)所谓后验概率