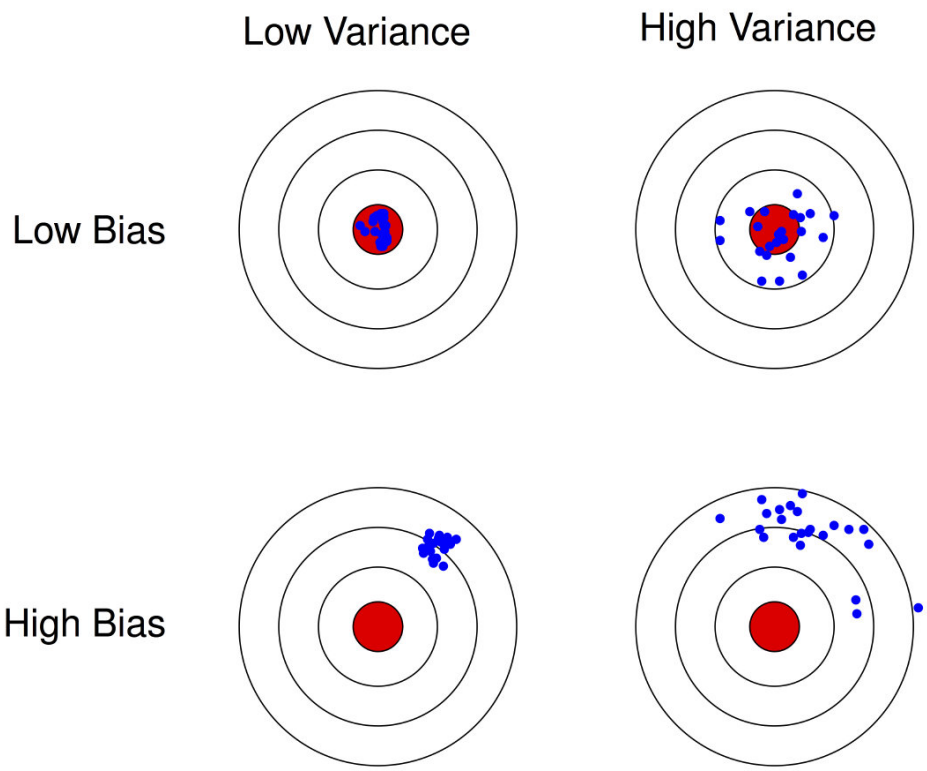
题目1： 说说你对偏差与方差、过拟合与欠拟合的理解

关于偏差、方差可以通过如下图进行对比：



1 / 1

1 方差、偏差和噪声之间区别

- 方差: 预测值的平均值
  - 方差度量了同样大小的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**
- 偏差: 真实值和预测值之间的差别
  - 偏差度量了学习算法的期望预测与真实结果的偏离程序, 即 **刻画了学习算法本身的拟合能力** .
- 噪声:是真实标记与数据集中的实际标记之间的偏差
  - 噪声表达了在当前任务上任何学习算法所能达到的期望泛化误差的下界, 即 **刻画了学习问题本身的难度** . 巧妇难为无米之炊, 给一堆很差的食材, 要想做出一顿美味, 肯定是很有难度的.

2 四个变量之间的关系

- 方差大的时候,过拟合,和平均值之间求得的值比较大
- 欠拟合的时候,偏差大,即预测值和真实值之间的比较

题目2：常见的数据分割有哪些方法，请简要说明

常见的数据分割方法有：

- 留出法
- 交叉验证法
- 自助法

1 留出法

“留出法”(hold-out)直接将数据集D划分为两个互斥的集合，其中一个集合作为训练集S，另一个作为测试集T，即 $D = SUT, S \cap T = \Phi$。在S上训练出模型后，用T来评估其测试误差，作为对泛化误差的估计。

大家在使用的过程中，**需注意的是，训练/测试集的划分要尽可能保持数据分布的一致性，避免因数据划分过程引入额外的偏差而对最终结果产生影响**，例如在分类任务中至少要保持样本的类别比例相似。

如果从采样(sampling)的角度来看待数据集的划分过程，则保留类别比例的采样方式通常称为**“分层采样”(stratified sampling)。**

> 例如通过对D进行分层样而获得含70%样本的训练集S和含30%样本的测试集T，
>
> 若D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，而T则包含150个正例和150个反例；
>
> 若S、T中样本类别比例差别很大，则误差估计将由于训练/测试数据分布的差异而产生偏差。

另一个需注意的问题是，即便在给定训练测试集的样本比例后，仍存在多种划分方式对初始数据集D进行分割。

例如在上面的例子中，可以把D中的样本排序，然后把前350个正例放到训练集中，也可以把最后350个正例放到训练集中，这些不同的划分将导致不同的训练/测试集，相应的，模型评估的结果也会有差别。

因此，单次使用留出法得到的估计结果往往不够稳定可靠，**在使用留出法时，一般要采用若干次随机划分、重复进行实验评估后取平均值作为留出法的评估结果。**

> 例如进行100次随机划分，每次产生一个训练/测试集用于实验评估，100次后就得到100个结果，而留出法返回的则是这100个结果的平均。

此外，我们希望评估的是用D训练出的模型的性能，但留出法需划分训练/测试集，这就会导致一个窘境:

- 若令训练集S包含绝大多数样本，则训练出的模型可能更接近于用D训练出的模型，但由于T比较小，评估结果可能不够稳定准确；

- 若令测试集T多包含一些样本，则训练集S与D差别更大了，被评估的模型与用D训练出的模型相比可能有较大差别，从而降低了评估结果的保真性(fidelity)。

这个问题没有完美的解决方案，常见做法是将大约2/3~4/5的样本用于训练，剩余样本用于测试。

在留出法中，有一个特例，叫：**留一法(Leave-One-Out，简称LOO）**，即每次抽取一个样本做为测试集。

显然，留一法不受随机样本划分方式的影响，因为m个样本只有唯一的方式划分为m个子集一每个子集包含个样本；

留一法优缺点：

优点：

- 留一法使用的训练集与初始数据集相比只少了一个样本，这就使得在绝大多数情况下，留一法中被实际评估的模型与期望评估的用D训练出的模型很相似。因此，**留一法的评估结果往往被认为比较准确**。

缺点：

- 留一法也有其缺陷:在数据集比较大时，训练m个模型的计算开销可能是难以忍受的(例如数据集包含1百万个样本，则需训练1百万个模型，而这还是在未考虑算法调参的情况下。

2 交叉验证法

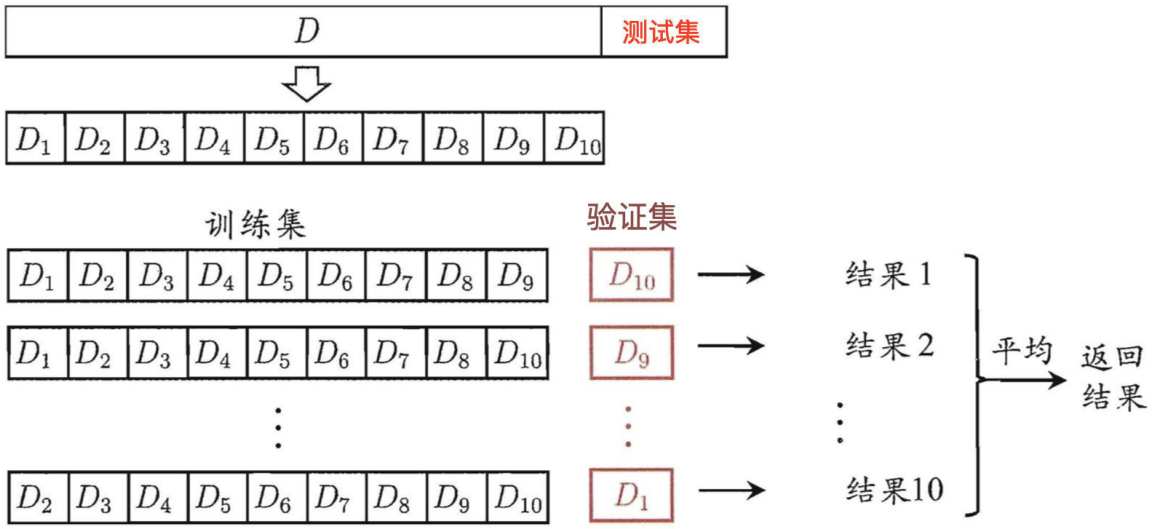
“交叉验证法”(cross validation)先将数据集D划分为k个大小相似的互斥子集，即

$$D = D_1 U D_2 U \ldots U D_k \ , D_i \cap D_j = \Phi (i \neq j)$$

。每个子集D_i*D**i*都尽可能保持数据分布的一致性，即从D中通过分层抽样得到。

然后，每次用k-1个子集的并集作为训练集，余下的那个子集作为测试集；这样就可获得k组训练/测试集，从而可进行k次训练和测试，最终返回的是这k个测试结果的均值。

显然，交叉验证法评估结果的稳定性和保真性在很大程度上取决于k的取值，为强调这一点，通常把交叉验证法称为“k折交叉验证”(k- fold cross validation)。k最常用的取值是10，此时称为10折交叉验证；其他常用的k值有5、20等。下图给出了10折交叉验证的示意图。



与留出法相似，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，例如常见的有 “10次10折交叉验证”。

3 自助法

我们希望评估的是用D训练出的模型。但在留出法和交叉验证法中，由于保留了一部分样本用于测试，因此**实际评估的模型所使用的训练集比D小，**这必然会引入一些因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，但计算复杂度又太高了。

有没有什么办法可以减少训练样本规模不同造成的影响，同时还能比较高效地进行实验估计呢?

“自助法”(bootstrapping)是一个比较好的解决方案，它**直接以自助采样法(bootstrap sampling)为基础**。给定包含m个样本的数据集D，我们对它进行采样产生数据集D:

- 每次随机从D中挑选一个样本，将其拷贝放入D，然后再将该样本放回初始数据集D中，使得该样本在下次采样时仍有可能被到；
- 这个过程重复执行m次后，我们就得到了包含m个样本的数据集D'，这就是自助采样的结果。

显然，D中有一部分样本会在D'中多次出现，而另一部分样本不出现。可以做一个简单的估计，样本在m次采样中始终不被采到的概率是(1-\frac{1}{m})^m(1−*m*1)*m*，取极限得到

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

即通过自助采样，初始数据集D中约有36.8%的样本未出现在采样数据集D'中。

于是我们可将D'用作训练集，D\D'用作测试集；这样，实际评估的模型与期望评估的模型都使用m个训练样本，而我们仍有数据总量约1/3的、没在训练集中出现的样本用于测试。

这样的测试结果，亦称**“包外估计”(out- of-bagestimate）**

自助法优缺点：

- 优点：
  - 自助法在数据集较小、难以有效划分训练/测试集时很有用；
  - 此外，自助法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有很大的好处。
- 缺点：
  - 自助法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；留出法和交叉验证法更常用一些。

题目3： 谈谈你对判别式模型和生成式模型的理解

- 判别方法：
  - 由数据直接学习决策函数 Y = f（X），或者由条件分布概率 P（Y|X）作为预测模型，即判别模型。
- 生成方法：
  - 由数据学习联合概率密度分布函数 P（X,Y）,然后求出条件概率分布P(Y|X)作为预测的模型，即生成模型。 由生成模型可以得到判别模型，但由判别模型得不到生成模型。

- 常见的判别模型有：
  - K近邻、SVM、决策树、感知机、线性判别分析（LDA）、线性回归、传统的神经网络、逻辑斯蒂回归、boosting
- 常见的生成模型有：
  - 朴素贝叶斯、隐马尔可夫模型、文档主题生成模型（LDA）

题目4： 你在使用逻辑回归的时候有哪些感受。觉得它有哪些优缺点。

在这里我们总结了逻辑回归应用到工业界当中一些优点：

- 形式简单，模型的可解释性非常好。从特征的权重可以看到不同的特征对最后结果的影响，某个特征的权重值比较高，那么这个特征最后对结果的影响会比较大。
- 模型效果不错。在工程上是可以接受的（作为baseline)，如果特征工程做的好，效果不会太差，并且特征工程可以大家并行开发，大大加快开发的速度。
- 训练速度较快。分类的时候，计算量仅仅只和特征的数目相关。并且逻辑回归的分布式优化sgd发展比较成熟，训练的速度可以通过堆机器进一步提高，这样我们可以在短时间内迭代好几个版本的模型。
- 资源占用小,尤其是内存。因为只需要存储各个维度的特征值。
- 方便输出结果调整。逻辑回归可以很方便的得到最后的分类结果，因为输出的是每个样本的概率分数，我们可以很容易的对这些概率分数进行cutoff，也就是划分阈值(大于某个阈值的是一类，小于某个阈值的是一类)。

但是逻辑回归本身也有许多的缺点:

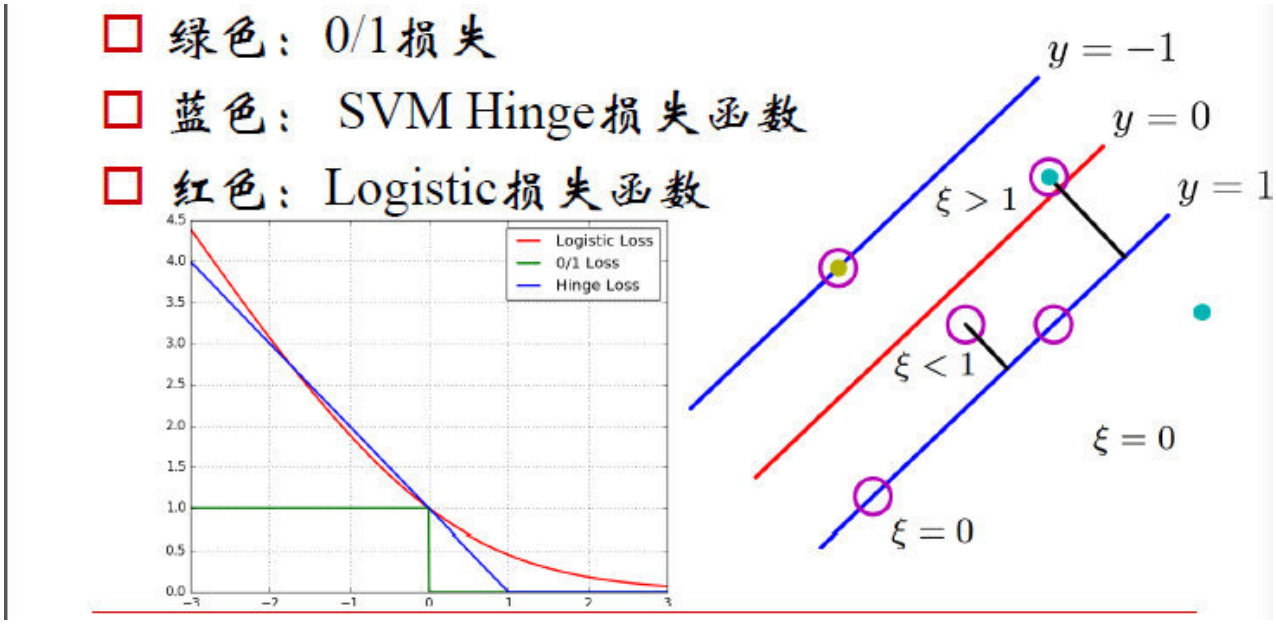
- 准确率并不是很高。因为形式非常的简单(非常类似线性模型)，很难去拟合数据的真实分布。
- 很难处理数据不平衡的问题。举个例子：如果我们对于一个正负样本非常不平衡的问题比如正负样本比 10000:1.我们把所有样本都预测为正也能使损失函数的值比较小。但是作为一个分类器，它对正负样本的区分能力不会很好。
- 处理非线性数据较麻烦。逻辑回归在不引入其他方法的情况下，只能处理线性可分的数据，或者进一步说，处理二分类的问题 。
- 逻辑回归本身无法筛选特征。有时候，我们会用gbdt来筛选特征，然后再上逻辑回归。

题目5： 为什么属性独立性假设在实际情况中很难成立，但朴素贝叶斯仍能取得较好的效果?

- 人们在使用分类器之前，首先做的第一步（也是最重要的一步）往往是**特征选择**，这个过程的目的就是为了**排除特征之间的共线性、选择相对较为独立的特征**；
- 对于分类任务来说，**只要各类别的条件概率排序正确，无需精准概率值就可以得出正确分类**；
- 如果**属性间依赖对所有类别影响相同，或依赖关系的影响能相互抵消，**则属性条件独立性假设在降低计算复杂度的同时不会对性能产生负面影响。

题目6： 请对比一下0/1损失，SVM Hinge损失和Logistic损失函数

三种损失函数，如下图显示：



- **绿色：0/1损失**
  - 当正例的点落在y=0这个超平面的下边，说明是分类正确，无论距离超平面所远多近，误差都是0.
  - 当这个正例的样本点落在y=0的上方的时候，说明分类错误，无论距离多远多近，误差都为1.
  - 图像就是上图绿色线。
- **蓝色：SVM Hinge损失函数**
  - 当一个正例的点落在y=1的直线上，距离超平面长度1，那么1-ξ=1，ξ=0，也就是说误差为0；
  - 当它落在距离超平面0.5的地方，1-ξ=0.5，ξ=0.5，也就是说误差为0.5；
  - 当它落在y=0上的时候，距离为0，1-ξ=0，ξ=1，误差为1；
  - 当这个点落在了y=0的上方，被误分到了负例中，距离算出来应该是负的，比如-0.5，那么1-ξ=-0.5，ξ=-1.5.误差为1.5.
  - 以此类推，画在二维坐标上就是上图中蓝色那根线了。
- **红色：Logistic损失函数**
  - 损失函数的公式为：$ln(1+e^{-y_i})$

- 当$y_i = 0$时，损失等于ln2,这样真丑，所以我们给这个损失函数除以ln2.
- 这样到$y_i = 0$时，损失为1，即损失函数过（0，1）点
- 即上图中的红色线。

题目7： 简述SVM、LR、决策树的对比。

- 模型复杂度：
  - SVM支持核函数，可处理线性非线性问题;
  - LR模型简单，训练速度快，适合处理线性问题;
  - 决策树容易过拟合，需要进行剪枝
- 数据敏感度：
  - SVM添加容忍度对outlier不敏感，只关心支持向量，且需要先做归一化;
  - LR对远点敏感
- 数据量：
  - 数据量大就用LR,
  - 数据量小且特征少就用SVM非线性核

题目8： 朴素贝叶斯与LR的区别?

1 简单来说：

- 区别一：
  - 朴素贝叶斯是生成模型，
    - 根据已有样本进行贝叶斯估计学习出先验概率P(Y)和条件概率P(X|Y),
    - 进而求出联合分布概率P(XY),
    - 最后利用贝叶斯定理求解P(Y|X),
  - 而LR是判别模型，
    - 根据极大化对数似然函数直接求出条件概率P(Y|X)；
  - > 从概率框架的角度来理解机器学习；主要有两种策略：
    >
    > 第一种：给定 x， 可通过直接建模 P(c |x) 来预测 c，这样得到的是"判别式模型" (discriminative models)；
    >
    > 第二种：也可先对联合概率分布 P(x,c) 建模，然后再由此获得 P(c |x)， 这样得到的是"生成式模型" (generative models) ;
    >
    > 显然，前面介绍的逻辑回归、决策树、都可归入判别式模型的范畴，还有后面学到的BP神经网络
    >
    > 支持向量机等；
    >
    > 对生成式模型来说，必然需要考虑

> $$P(c \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x}, c)}{P(\boldsymbol{x})} \ .$$

- 区别二：
  - 朴素贝叶斯是基于很强的条件独立假设（在已知分类Y的条件下，各个特征变量取值是相互独立的），
  - 而LR则对此没有要求；
- 区别三：
  - 朴素贝叶斯适用于数据集少的情景，
  - 而LR适用于大规模数据集。

2 进一步说明：

前者是生成式模型，后者是判别式模型，二者的区别就是生成式模型与判别式模型的区别。

- 首先，Navie Bayes通过已知样本求得先验概率P(Y), 及条件概率P(X|Y), 对于给定的实例，计算联合概率，进而求出后验概率。也就是说，它尝试去找到底这个数据是怎么生成的（产生的），然后再进行分类。哪个类别最有可能产生这个信号，就属于那个类别。
  - 优点： 样本容量增加时，收敛更快；隐变量存在时也可适用。
  - 缺点：时间长；需要样本多；浪费计算资源
- 相比之下，Logistic回归不关心样本中类别的比例及类别下出现特征的概率，它直接给出预测模型的式子。设每个特征都有一个权重，训练样本数据更新权重w，得出最终表达式。
  - 优点：
    - 直接预测往往准确率更高；
    - 简化问题；
    - 可以反应数据的分布情况，类别的差异特征；
    - 适用于较多类别的识别。
  - 缺点
    - 收敛慢；
    - 不适用于有隐变量的情况。

题目9： KMeans初始类簇中心点的选取。

k-means++算法选择初始seeds的基本思想就是：初始的聚类中心之间的相互距离要尽可能的远。

1. 从输入的数据点集合中随机选择一个点作为第一个聚类中心
2. 对于数据集中的每一个点x，计算它与最近聚类中心(指已选择的聚类中心)的距离D(x)
3. 选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大
4. 重复2和3直到k个聚类中心被选出来
5. 利用这k个初始的聚类中心来运行标准的k-means算法

题目10： 说说你对xgboost的理解

XGBoost（Extreme Gradient Boosting）全名叫极端梯度提升树，XGBoost是集成学习方法的王牌，在Kaggle数据挖掘比赛中，大部分获胜者用了XGBoost。

XGBoost在绝大多数的回归和分类问题上表现的十分顶尖，本节将较详细的介绍XGBoost的算法原理。

1 最优模型的构建方法

我们在前面已经知道，构建最优模型的一般方法是**最小化训练数据的损失函数**。

我们用字母 L表示损失，如下式：

$$\min_{f \in F} \quad \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) \qquad (1.1)$$

> 其中，F是假设空间
>
> 假设空间是在已知属性和属性可能取值的情况下，对所有可能满足目标的情况的一种毫无遗漏的假设集合。

式（1.1）称为**经验风险最小化**，训练得到的模型复杂度较高。当训练数据较小时，模型很容易出现过拟合问题。

因此，为了降低模型的复杂度，常采用下式：

$$\min_{f \in F} \quad \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda J(f) \qquad (2.1)$$

> 其中$J(f)$为模型的复杂度，

式（2.1）称为**结构风险最小化**，结构风险最小化的模型往往对训练数据以及未知的测试数据都有较好的预测 。

应用：

- 决策树的生成和剪枝分别对应了经验风险最小化和结构风险最小化，
- XGBoost的决策树生成是结构风险最小化的结果，后续会详细介绍。

2 XGBoost的目标函数推导

2.1 目标函数确定

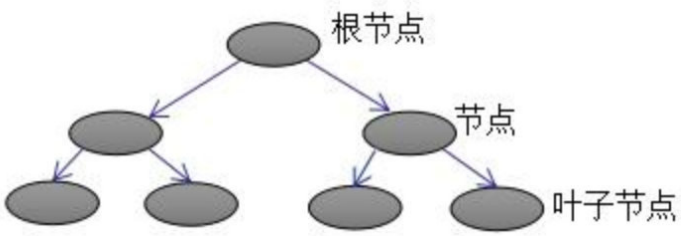
目标函数，即损失函数，通过最小化损失函数来构建最优模型。

由前面可知， 损失函数应加上表示模型复杂度的正则项，且XGBoost对应的模型包含了多个CART树，因此，模型的目标函数为：

$$obj(\theta)=\sum_{i}^{n}L(y_i,\hat{y_i})+\sum_{k=1}^{K}\Omega(f_k) \qquad (3.1)$$

> （3.1）式是正则化的损失函数；
>
> 其中y_i是模型的实际输出结果，$\overline{y_i}$是模型的输出结果；
>
> 等式右边第一部分是模型的训练误差，第二部分是正则化项，这里的正则化项是K棵树的正则化项相加而来的。

2.2 CART树的介绍



上图为第K棵CART树，确定一棵CART树需要确定两部分，

第一部分就是**树的结构**，这个结构将输入样本映射到一个确定的叶子节点上，记为$f_k(x)$;

第二部分就是**各个叶子节点的值**，$q(x)$表示输出的叶子节点序号，$w_q(x)$表示对应叶子节点序号的值。

由定义得：

$$f_k(x)=w_{q(x)} \qquad (3.2)$$

2.3 树的复杂度定义

2.3.1 定义每课树的复杂度

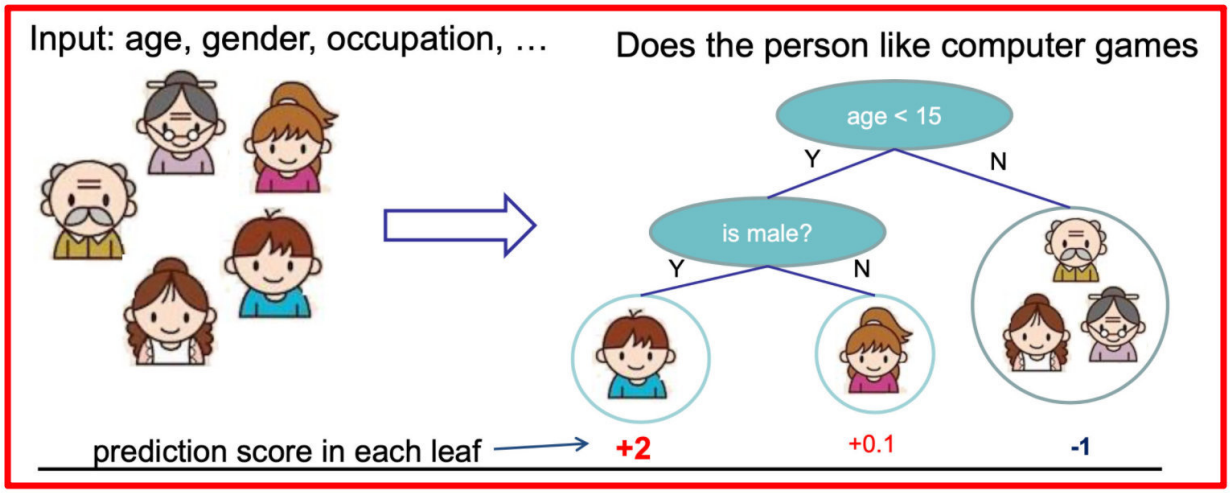
XGBoost法对应的模型包含了多棵cart树，定义每棵树的复杂度：

$$\Omega(f)=\gamma T+\frac{1}{2}\lambda\|w\|^2 \qquad (3.4)$$

> 其中T为叶子节点的个数，||w||为叶子节点向量的模 。γ表示节点切分的难度，λ表示L2正则化系数。

2.3.2 树的复杂度举例

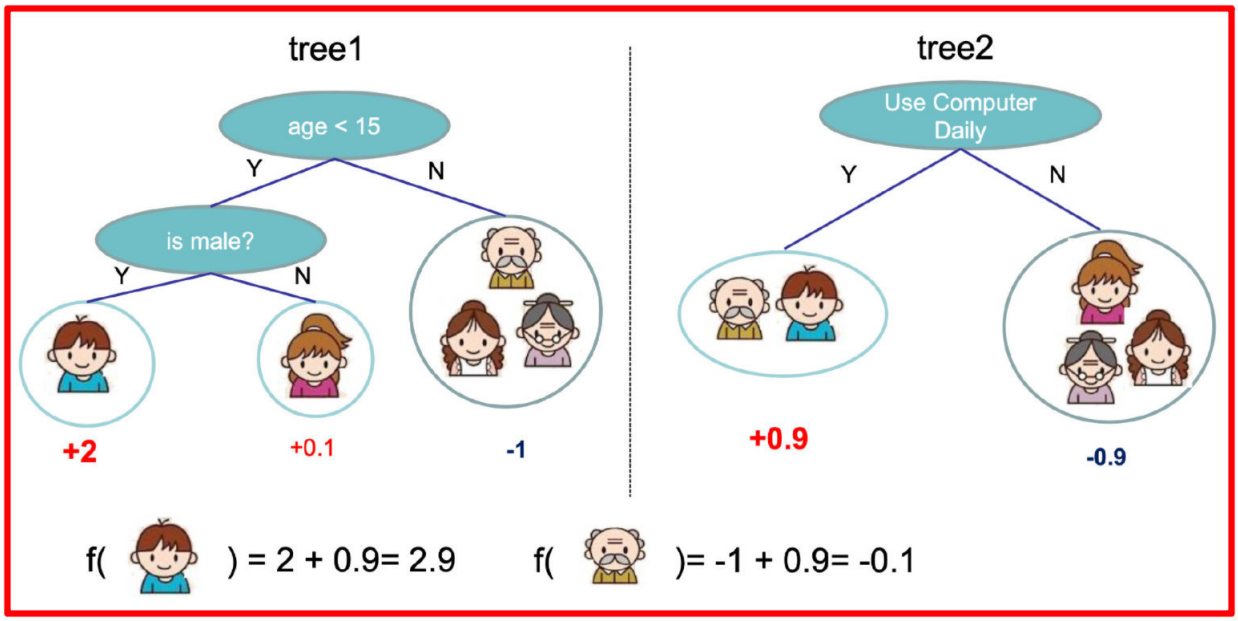
假设我们要预测一家人对电子游戏的喜好程度，考虑到年轻和年老相比，年轻更可能喜欢电子游戏，以及男性和女性相比，男性更喜欢电子游戏，故先根据年龄大小区分小孩和大人，然后再通过性别区分开是男是女，逐一给各人在电子游戏喜好程度上打分，如下图所示：



就这样，训练出了2棵树tree1和tree2，类似之前gbdt的原理，两棵树的结论累加起来便是最终的结论，所以：

- 小男孩的预测分数就是两棵树中小孩所落到的结点的分数相加：2 + 0.9 = 2.9。
- 爷爷的预测分数同理：-1 + （-0.9）= -1.9。

具体如下图所示：

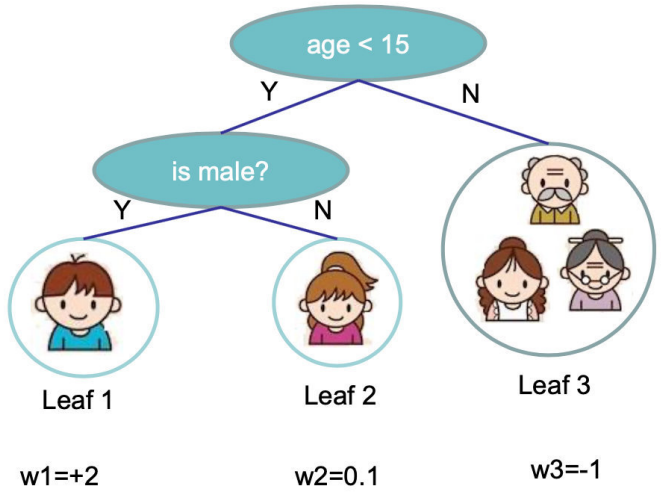


如下例树的复杂度表示：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

叶子的个数

w的L2模平方



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

2.4 目标函数推导

根据（3.1）式，共进行t次迭代的学习模型的目标函数为：

$$obj^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k)$$

$$= \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{k=1}^{t-1} \Omega(f_k) + \Omega(f_t)$$

由前向分布算法可知，前t-1棵树的结构为常数

$$obj^{(t)} == \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + Cons\tan t \qquad (3.5)$$

我们知道，泰勒公式的二阶导近似表示：

$$f(x_0 + \Delta x) \approx f(x_0) + f'(x_0)\cdot(\Delta x) + \frac{f''(x_0)}{2}\cdot(\Delta x)^2 \qquad (3.6)$$

令$f_t(x_i)$为Δx, 则（3.5）式的二阶近似展开：

$$obj^{(t)} \approx \sum_{i=1}^{n}[L(y_i, \hat{y}_i^{(t-1)}) + g_i \cdot f_t(x_i) + \frac{1}{2} h_i \cdot f_t(x_i)^2] + \Omega(f_t) + Cons\tan t \qquad (3.7)$$

$\because \quad L(y_i, \hat{y}_i^{(t-1)})$为常数

$$\therefore \quad obj^{(t)} = \sum_{i=1}^{n}[g_i \cdot f_t(x_i) + \frac{1}{2} h_i \cdot f_t(x_i)^2] + \Omega(f_t) + Cons\tan t \qquad (3.8)$$

其中：

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

> g_i和h_i分别表示预测误差对当前模型的一阶导和二阶导;
>
> $l(y_i, \hat{y}^{(t-1)})$ 表示前t-1棵树组成的学习模型的预测误差。

当前模型往预测误差减小的方向进行迭代。

忽略（3.8）式常数项，并结合（3.4）式，得：

$$obj^{(t)} = \sum_{i=1}^{n}[g_i \cdot f_t(x_i) + \frac{1}{2} h_i \cdot f_t(x_i)^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w^2{}_j \qquad (3.9)$$

通过（3.2）式简化（3.9）式：

$$obj^{(t)} = \sum_{i=1}^{n}[g_i \cdot w_{q(x_i)} + \frac{1}{2} h_i \cdot w^2{}_{q(x_i)}] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w^2{}_j \qquad (3.10)$$

> （3.10）式第一部分是对所有训练样本集进行累加，
>
> 此时，所有样本都是映射为树的叶子节点，

所以，我们换种思维，从叶子节点出发，对所有的叶子节点进行累加，得：

$$obj^{(t)} = \sum_{j=1}^{T}[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \qquad (3.11)$$

令

$$\sum_{i \in I_j} g_i = G_j \,, \quad \sum_{i \in I_j} h_i = H_j$$

G_j表示映射为叶子节点 j 的所有输入样本的一阶导之和，同理，H_j表示二阶导之和。

得：

$$obj^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \qquad (3.12)$$

对于第 t 棵CART树的某一个确定结构（可用q(x)表示），其叶子节点是相互独立的，

G_j和H_j是确定量，因此，（3.12）可以看成是关于叶子节点w的一元二次函数 。

最小化（3.12）式，得：

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad (3.13)$$

把（3.13）带入到（3.12），得到最终的目标函数：

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \qquad (3.14)$$

（3.14）也称为**打分函数(scoring function)**，它是衡量树结构好坏的标准，

- 值越小，代表这样的结构越好 。
- 我们用打分函数选择最佳切分点，从而构建CART树。

3 XGBoost的回归树构建方法

3.1 计算分裂节点

在实际训练过程中，当建立第 t 棵树时，XGBoost采用贪心法进行树结点的分裂：

从树深为0时开始：

- 对树中的每个叶子结点尝试进行分裂；
- 每次分裂后，原来的一个叶子结点继续分裂为左右两个子叶子结点，原叶子结点中的样本集将根据该结点的判断规则分散到左右两个叶子结点中；
- 新分裂一个结点后，我们需要检测这次分裂是否会给损失函数带来增益，增益的定义如下：

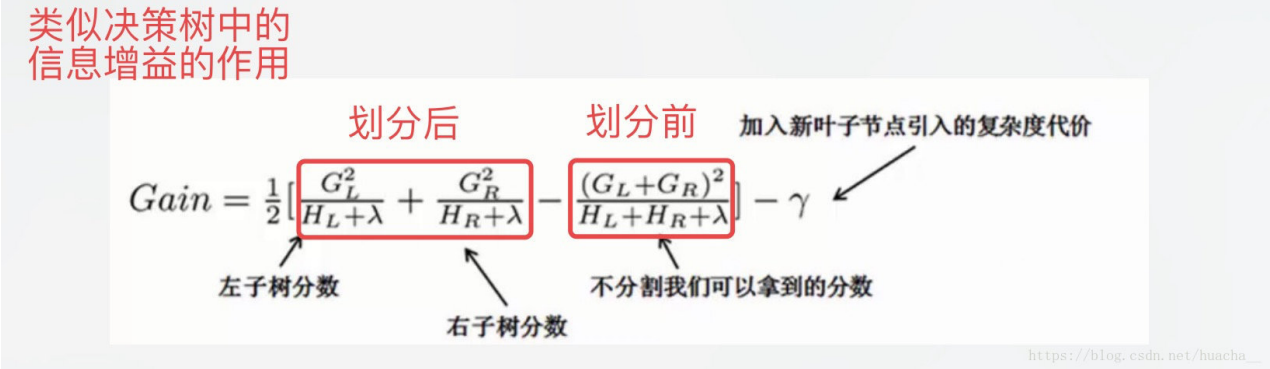
$$\begin{aligned} Gain &= Obj_{L+R} - (Obj_L + Obj_R) \\ &= [-\frac{1}{2}\frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma] - [-\frac{1}{2}(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda}) + 2\gamma] \\ &= \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \end{aligned}$$

如果增益Gain>0，即分裂为两个叶子节点后，目标函数下降了，那么我们会考虑此次分裂的结果。

那么一直这样分裂，什么时候才会停止呢?

3.2 停止分裂条件判断

情况一：上节推导得到的打分函数是衡量树结构好坏的标准，因此，可用打分函数来选择最佳切分点。首先确定样本特征的所有切分点，对每一个确定的切分点进行切分，切分好坏的标准如下：



- Gain表示单节点obj*与切分后的两个节点的树obj之差*，
- 遍历所有特征的切分点，找到最大Gain的切分点即是最佳分裂点，根据这种方法继续切分节点，得到CART树。
- 若 γ 值设置的过大，则Gain为负，表示不切分该节点，因为切分后的树结构变差了。
    - γ 值越大，表示对切分后obj下降幅度要求越严，这个值可以在XGBoost中设定。

情况二：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。

情况三：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，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，这也是过拟合的一种措施。

题目11： 说说你对lightGBM的理解

1 什么是lightGBM

lightGBM是2017年1月，微软在GItHub上开源的一个新的梯度提升框架。

github介绍链接

在开源之后，就被别人冠以“速度惊人”、“支持分布式”、“代码清晰易懂”、“占用内存小”等属性。

LightGBM主打的高效并行训练让其性能超越现有其他boosting工具。在Higgs数据集上的试验表明，LightGBM比XGBoost快将近10倍，内存占用率大约为XGBoost的1/6。

> higgs数据集介绍：这是一个分类问题，用于区分产生希格斯玻色子的信号过程和不产生希格斯玻色子的信号过程。

> 数据链接

2 lightGBM原理

lightGBM 主要基于以下方面优化，提升整体特特性：

1. 基于Histogram（直方图）的决策树算法
2. Lightgbm 的Histogram（直方图）做差加速
3. 带深度限制的Leaf-wise的叶子生长策略
4. 直接支持类别特征
5. 直接支持高效并行

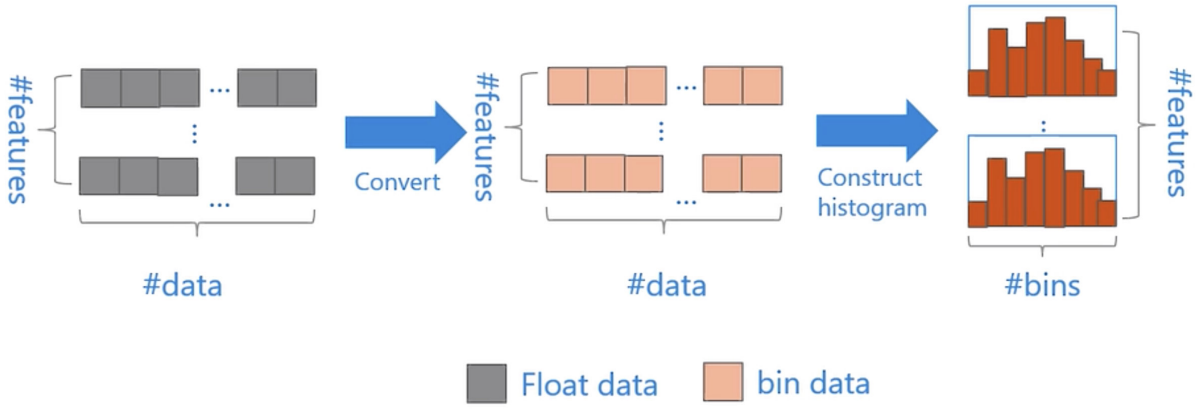
具体解释见下，分节介绍。

2.1 基于Histogram（直方图）的决策树算法

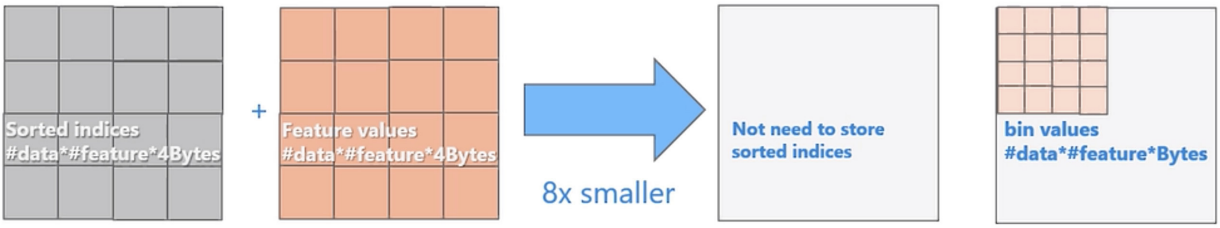
直方图算法的基本思想是

- 先把连续的浮点特征值离散化成k个整数，同时构造一个宽度为k的直方图。
- 在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。

> Eg:
>
> [0, 0.1) --> 0;
>
> [0.1,0.3) --> 1;
>
> ...



使用直方图算法有很多优点。首先，**最明显就是内存消耗的降低，**直方图算法不仅不需要额外存储预排序的结果，而且可以只保存特征离散化后的值，而这个值一般用8位整型存储就足够了，内存消耗可以降低为原来的1/8。



然后**在计算上的代价也大幅降低**，预排序算法每遍历一个特征值就需要计算一次分裂的增益，而直方图算法只需要计算k次（k可以认为是常数），时间复杂度从O(#data*#feature)优化到O(k*#features)。

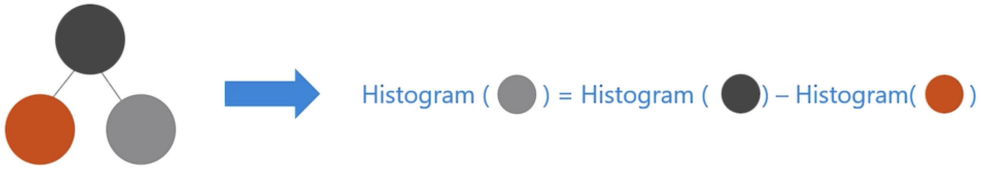
当然，Histogram算法并不是完美的。由于特征被离散化后，**找到的并不是很精确的分割点，所以会对结果产生影响。**但在不同的数据集上的结果表明，离散化的分割点对最终的精度影响并不是很大，甚至有时候会更好一点。原因是决策树本来就是弱模型，分割点是不是精确并不是太重要；较粗的分割点也有正则化的效果，可以有效地防止过拟合；即使单棵树的训练误差比精确分割的算法稍大，但在梯度提升（Gradient Boosting）的框架下没有太大的影响。

2.2 Lightgbm 的Histogram（直方图）做差加速

一个叶子的直方图可以由它的父亲节点的直方图与它兄弟的直方图做差得到。

通常构造直方图，需要遍历该叶子上的所有数据，但直方图做差仅需遍历直方图的k个桶。

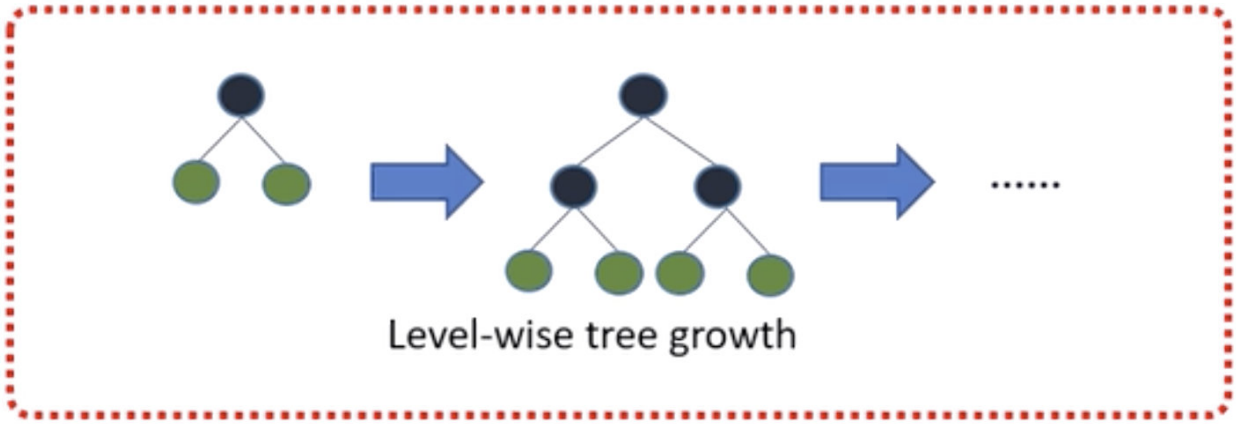
利用这个方法，LightGBM可以在构造一个叶子的直方图后，可以用非常微小的代价得到它兄弟叶子的直方图，在速度上可以提升一倍。



2.3 带深度限制的Leaf-wise的叶子生长策略

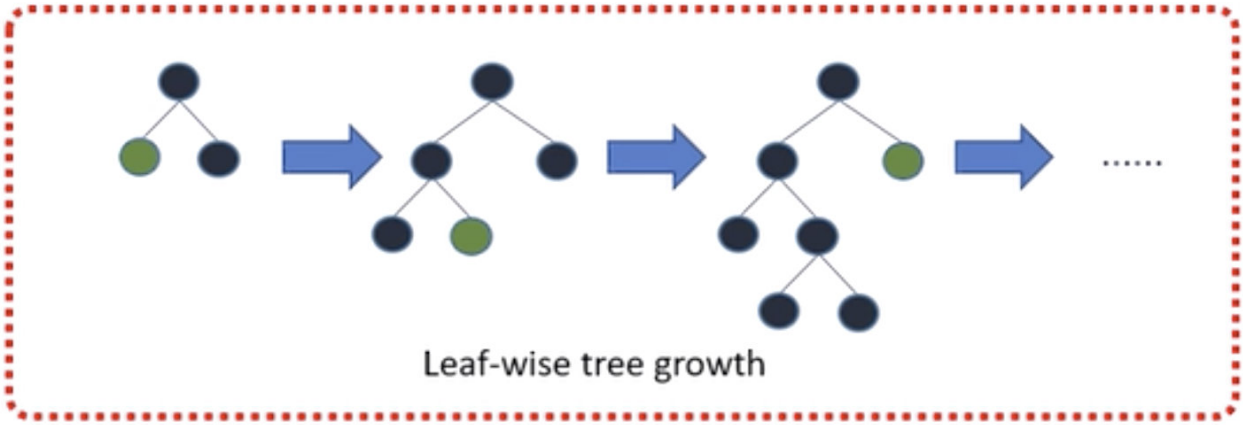
Level-wise便利一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度，不容易过拟合。

- 但实际上Level-wise是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销，因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。



Leaf-wise则是一种更为高效的策略，每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。

- 因此同Level-wise相比，在分裂次数相同的情况下，Leaf-wise可以降低更多的误差，得到更好的精度。
- Leaf-wise的缺点是可能会长出比较深的决策树，产生过拟合。因此LightGBM在Leaf-wise之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。



2.4 直接支持类别特征

实际上大多数机器学习工具都无法直接支持类别特征，一般需要把类别特征，转化到多维的0/1特征，降低了空间和时间的效率。

而类别特征的使用是在实践中很常用的。基于这个考虑，LightGBM优化了对类别特征的支持，可以直接输入类别特征，不需要额外的0/1展开。并在决策树算法上增加了类别特征的决策规则。

在Expo数据集上的实验，相比0/1展开的方法，训练速度可以加速8倍，并且精度一致。目前来看，LightGBM是第一个直接支持类别特征的GBDT工具。

> Expo数据集介绍：数据包含1987年10月至2008年4月美国境内所有商业航班的航班到达和离开的详细信息。这是一个庞大的数据集：总共有近1.2亿条记录。主要用于预测航班是否准时。
>
> 数据链接

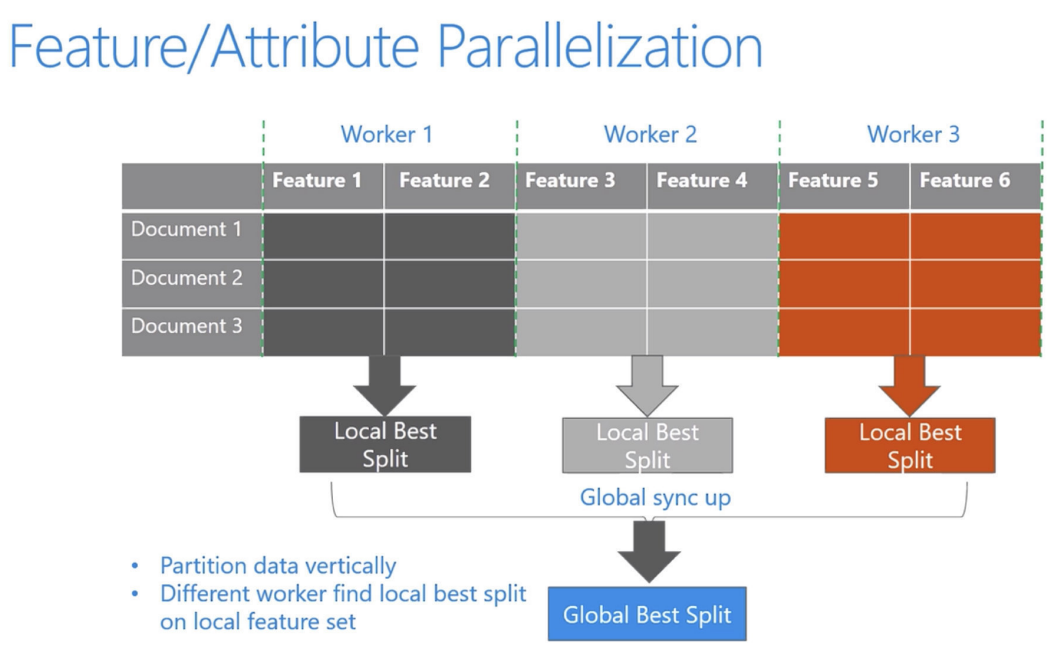
2.5 直接支持高效并行

LightGBM还具有支持高效并行的优点。LightGBM原生支持并行学习，目前支持特征并行和数据并行的两种。

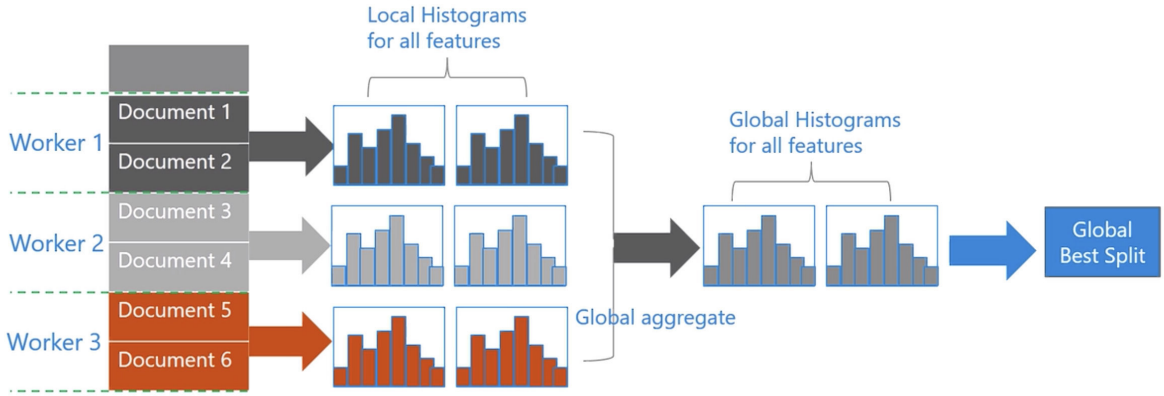
- 特征并行的主要思想是在不同机器在不同的特征集合上分别寻找最优的分割点，然后在机器间同步最优的分割点。
- 数据并行则是让不同的机器先在本地构造直方图，然后进行全局的合并，最后在合并的直方图上面寻找最优分割点。

LightGBM针对这两种并行方法都做了优化:

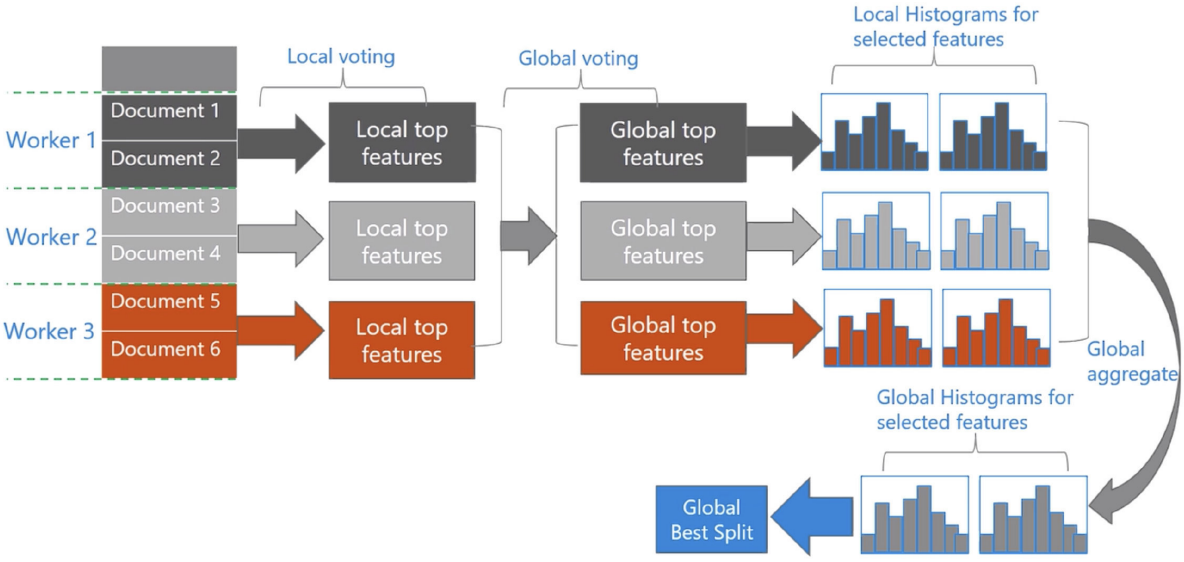
- 在**特征并行**算法中，通过在本地保存全部数据避免对数据切分结果的通信；



- 在**数据并行**中使用分散规约 (Reduce scatter) 把直方图合并的任务分摊到不同的机器，降低通信和计算，并利用直方图做差，进一步减少了一半的通信量。



- **基于投票的数据并行(Voting Parallelization)**则进一步优化数据并行中的通信代价，使通信代价变成常数级别。在数据量很大的时候，使用投票并行可以得到非常好的加速效果。



题目12： 简要说一下Lightgbm相对于xgboost的优缺点

- 优点：
  - 直方图算法—更高（效率）更快（速度）更低（内存占用）更泛化（分箱与之后的不精确分割也起到了一定防止过拟合的作用）；
- 缺点：
  - 直方图较为粗糙，会损失一定精度，但是在gbm的框架下，基学习器的精度损失可以通过引入更多的tree来弥补。

题目13： 为什么xgboost不用后剪枝?

后剪枝计算代价太高了，合并一次叶节点就要计算一次测试集的表现，数据量大的情况下非常消耗时间，而且也并不是特别必要，因为这样很容易过拟合测试集。

题目14： 什么是OOB？随机森林中OOB是如何计算的，它有什么用途?

1 什么是OOB

bagging方法中Bootstrap每次约有1/3的样本不会出现在Bootstrap所采集的样本集合中，当然也就没有参加决策树的建立，把这1/3的数据称为袋外数据oob（out of bag）,它可以用于取代测试集误差估计方法。

2 OOB是如何计算的

随机森林的 Bagging 过程，对于每一颗训练出的决策树 g_t ，与数据集 D 有如下关系：

	g_1	g_2	g_3	.....	g_T
(x_1, y_1)	D_1	*	D_3		D_T
(x_2, y_2)	*	*	D_3		D_T
(x_3, y_3)	*	D_2	*		D_T
.....					
(x_N, y_N)	D_1	D_2	*		*

对于星号的部分，即是没有选择到的数据，称之为 Out-of-bag(OOB)数据，当数据足够多，对于任意一组数据 (x_n, y_n) 是包外数据的概率为：

$$\left(1-\frac{1}{N}\right)^N = \frac{1}{\left(\frac{N}{N-1}\right)^N} = \frac{1}{\left(1+\frac{1}{N-1}\right)^N} \approx \frac{1}{e}$$

由于基分类器是构建在训练样本的自助抽样集上的，只有约 63.2% 原样本集出现在中，而剩余的 36.8% 的数据作为包外数据，可以用于基分类器的验证集。

经验证，包外估计是对集成分类器泛化误差的**无偏估计.**

在随机森林算法中数据集属性的重要性、分类器集强度和分类器间相关性计算都依赖于袋外数据。

3 包外估计的用途

- 当基学习器是决策树时，可使用包外样本来辅助剪枝 ，或用于估计决策树中各结点的后验概率以辅助对零训练样本结点的处理；
- 当基学习器是神经网络时，可使用包外样本来辅助早期停止以减小过拟合 。

题目15：请说说GBDT算法以及优缺点

GBDT和AdaBosst很类似，但是又有所不同。

- GBDT和其它Boosting算法一样，通过**将表现一般的几个模型（通常是深度固定的决策树）组合在一起来集成一个表现较好的模型。**
- AdaBoost是**通过提升错分数据点的权重来定位模型的不足**， Gradient Boosting通过负梯度来识别问题，通过计算负梯度来改进模型，即通过反复地选择一个指向负梯度方向的函数，该算法可被看做在函数空间里对目标函数进行优化。

因此可以说 。

- $GradientBoosting = GradientDescent + Boosting$

缺点：

GBDT ->预排序方法(pre-sorted)

- (1) **空间消耗大。**
  - 这样的算法需要保存数据的特征值，还保存了特征排序的结果（例如排序后的索引，为了后续快速的计算分割点），这里**需要消耗训练数据两倍的内存。**
- (2) **时间上也有较大的开销。**
  - 在遍历每一个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。
- (3) **对内存(cache)优化不友好。**
  - 在预排序后，特征对梯度的访问是一种随机访问，并且不同的特征访问的顺序不一样，无法对

cache进行优化。
    - 同时，在每一层长树的时候，需要随机访问一个行索引到叶子索引的数组，并且不同特征访问的顺序也不一样，也会造成较大的cache miss。

题目16：XGBoost与GDBT的区别

- 区别一：
    - XGBoost生成CART树考虑了树的复杂度，
    - GDBT未考虑，GDBT在树的剪枝步骤中考虑了树的复杂度。
- 区别二：
    - XGBoost是拟合上一轮损失函数的二阶导展开，GDBT是拟合上一轮损失函数的一阶导展开，因此，XGBoost的准确性更高，且满足相同的训练效果，需要的迭代次数更少。
- 区别三：
    - XGBoost与GDBT都是逐次迭代来提高模型性能，但是XGBoost在选取最佳切分点时可以开启多线程进行，大大提高了运行速度。