机器学习与深度学习面试系列七（集成方法）

什么是集成学习（Ensemble Learning）？有哪些常用的模型?

集成学习(Ensemble Learning)就是通过某种策略将多个模型集成起来， 通过群体决策来提高决策准确率.。集成学习首要的问题是如何集成多个模型，比较常用的集成策略有直接平均、加权平均等。

集成学习的思想可以用一句古老的谚语来描述:“三个臭皮匠赛过诸葛亮”，但是一个有效的集成需要各个基模型的差异尽可能大。为了增加模型之间的差异性，可以采取 Bagging和Boosting 这两类方法.

Bagging类方法。 通过随机构造训练样本、随机选择特征等方法来提高每个基模型的独立性，代表性方法有 Bagging(Bootstrap Aggregating) 和随机森林(Random Forest)等。

Boosting类方法。 按照一定的顺序来先后训练不同的基模型， 每个模型都针对前序模型的错误进行专门训练. 根据前序模型的结果，来调整训练样本的权重，从而增加不同基模型之间的差异性. 只要基模型的准确率比随机猜测高，Boosting类方法就可以通过集成方法来显著地提高集成模型的准确率。例如：AdaBoost、GBDT、XGBoost、LightGBM等。

为什么集成学习需要各个基模型的差异尽可能大?

给定一个学习任务，假设输入 $\boldsymbol{x}$ 和输出 $\boldsymbol{y}$ 的真实关系为 $\boldsymbol{y}=\boldsymbol{h}(\boldsymbol{x})$ 。对于M个不同的模型 $\boldsymbol{f}_1(\boldsymbol{x})$ 、$\boldsymbol{f}_2(\boldsymbol{x})$ ... $\boldsymbol{f}_M(\boldsymbol{x})$ ，每个模型的期望错误为：
$\boldsymbol{R}(\boldsymbol{f}_m)=\boldsymbol{E}_x[(\boldsymbol{f}_m(\boldsymbol{x})-\boldsymbol{h}(\boldsymbol{x}))^2]=\boldsymbol{E}_x[\epsilon_m(\boldsymbol{x})^2]$ ，那么所有的模型的平均错误为：
$\hat{\boldsymbol{R}}(\boldsymbol{f}_m)=\frac{1}{\boldsymbol{M}}\sum_{\boldsymbol{m}=1}^{\boldsymbol{M}}\boldsymbol{E}_x[\epsilon_m(\boldsymbol{x})^2]$ 。

基于简单投票机制的集成模型 $\boldsymbol{F}(\boldsymbol{x})=\frac{1}{\boldsymbol{M}}\sum_{\boldsymbol{m}=1}^{\boldsymbol{M}}\boldsymbol{f}_m(\boldsymbol{x})$ ，$F(x)$ 的期望错误：

$$
\begin{aligned}
R(F) &= E_x[(\frac{1}{M}\sum_{m=1}^{M} f_m(x) - h(x))^2] \\
&= E_x[(\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(x))^2] \\
&= \frac{1}{M^2} E_x[\sum_{m=1}^{M}\sum_{n=1}^{M} \epsilon_m(x)\epsilon_n(x)] \\
&= \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M} E_x[\epsilon_m(x)\epsilon_n(x)]
\end{aligned}
$$

其中 $E_x[\epsilon_m(x)\epsilon_n(x)]$ 为两个不同模型错误的相关性. 如果每个模型的错误不相关，即 $\forall m \neq n$ ， $E_x[\epsilon_m(x)\epsilon_n(x)] = 0$ 。如果每个模型的错误都是相同的，则， $\epsilon_m(x) = \epsilon_n(x)$ 。并且，由于 $\epsilon_m(x) > 0$ ， $\forall m$,可以得到: $\hat{R}(f) \geq R(F) \geq \frac{1}{M}\hat{R}(f)$ 。

所以当模型之间不相关时，集成模型的期望错误为所有模型的平均期望错误的 $\frac{1}{M}$ ，如果 $M \to \infty$ ，则集成模型的期望错误趋近0 。

Bagging类方法是如何降低模型之间相关性的?

Bagging是Bootstrap Aggregating，思想就是从总体样本（N个数据）当中有放回的采样选出N个样本（样本数据点个数仍然不变为N）进行训练，通过多次（M次得出M个决策树模型）这样的结果，进行投票获取平均值作为结果输出，这就极大可能的避免了不好的样本数据，从而提高准确度。M个模型可以并行化训练得到。

Bootstrap Aggregating

随机森林(RF)是怎么做的?

Bagging + 决策树 = 随机森林，所以算法流程与Bagging的算法流程一样，只不过基学习器采用了决策树。此外随机森林在Bagging的基础上再做了一次特征的随机抽样。如果每个样本的特征维度为M，指定一个常数m<<M，随机从M个特征中选取m个特征子集，每次树进行分裂时，从这m个特征中选择最优的，这样可以增大模型之间的不相关性。

随机森林的两个随机性：

- 弱学习器的训练集通过**随机**且有放回采样得到
- 每个弱学习器的特征集通过从原始特征集**随机**选择得到

随机森林有什么优缺点?

优点:

- 模型泛化能力强，很大程度克服了决策树过拟合的缺点（数据扰动+属性扰动）
- 它能够处理很高维度（feature很多）的数据，并且不用做特征选择(因为特征子集是随机选择的)。
- 在训练完后，它能够给出哪些feature比较重要。（被决策树优先选择的feature比较重要）
- 训练速度快，容易做成并行化方法(训练时树与树之间是相互独立的)。
- 在训练过程中，能够检测到feature间的互相影响。
- 对于不平衡的数据集来说，它可以平衡误差。
- 原理和实现简单

缺点:

- 随机森林已经被证明在某些**噪音较大**的分类或回归问题上会过拟合。
- 对于有不同取值的属性的数据，取值划分较多的属性会对随机森林产生更大的影响，所以随机森林在这种数据上产出的属性权值是不可信的。

随机森林特征维度数量m选择有什么影响?

- 森林中任意两棵树的相关性：相关性越大，错误率越大；
- 森林中每棵树的分类能力：每棵树的分类能力越强，整个森林的错误率越低。

减小特征选择个数m，树的相关性和分类能力也会相应的降低；增大m，两者也会随之增大。

如何选择最优的特征选择个数m?

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。

- 对于已经生成的随机森林,用袋外数据测试其性能,假设袋外数据总数为O,用这O个袋外数据作为输入,带进之前已经生成的随机森林分类器,分类器会给出O个数据相应的分类
- 因为这O条数据的类型是已知的,则用正确的分类与随机森林分类器的结果进行比较,统计随机森林分类器分类错误的数目,设为X,则袋外数据误差大小=X/O

Bagging类方法和Boosting类方法有什么区别?

- 训练方式上，Bagging每个弱学习器的数据集通过又放回采样得到，Boosting每个弱学习器的数据集一样，但是样本权重会根据上一次迭代的预测结果进行调整。
- 弱学习器权重，Bagging都一样，Boosting权重根据其预测结果迭代得到。
- 弱学习器训练集中各样本权重，Bagging都一样，Boosting根据错误率不断调整样本权重，错误率越大，样本权重越高。
- 训练方式上，Bagging可以并行训练各个弱学习器，Boosting必须串行训练。
- 方差和偏差，Bagging降低方差，Boosting降低偏差。

AdaBoost的基本流程?

AdaBoost仅用于二分类问题。

用一个二分类器(如LR)去拟合数据集，预测每个样本是属于类别-1还是1，然后计算误分率。分类器会得到一个分类器权重，该权重是误分率的单调函数，该权重随后被用来给样本重新加权。对第一个分类器来说，所有的样本权重都是一样的。

第二个分类器拟合一样的数据集，只不过样本权重被改变了，前一个分类器分错的样本的权重会被加大。

同样地，第三个分类器也按照同样的方式拟合数据集，以此类推。

最后，所有分类器的结果进行加权投票得到每个样本的预测结果。

AdaBoost权重的推导?

AdaBoost算法也可以看作是一种分步(Stage- Wise)优化的加性模型，其损失函数定义为：
$L(F)=e^{-yF(x)}=e^{-y\sum_{m=1}^{M}\alpha_m f_m(x)}$ ，其中 y 和 $f_m(x)\in\{-1,1\}$ 。

经过M-1次迭代，此时 $F(x)=\sum_{m=1}^{M-1}\alpha_m f_m(x)$ ，则第M次迭代，就是寻找一个 $f_M(x)$ 和 α_M 使得 $L(F)=\sum_{n=1}^{N}e^{-y^{(n)}(F_{M-1}(x^{(n)})+\alpha_M f_M(x^{(n)})}$ 最小。

令 $w_m^{(n)}=e^{-y^{(n)}(F_{M-1}(x^{(n)})}$ 则， $L(F)=\sum_{n=1}^{N}w_m^{(n)}e^{-y^{(n)}(\alpha_M f_M(x^{(n)})}$ 。

由于 y 和 $f_m(x)\in\{-1,1\}$ ，故 $yf_m(x)=1-2\mathbb{I}(y\neq f_m(x))$ 。

e^x 在 $x=0$ 处的二阶泰勒展开公式为 $1+x+\frac{x^2}{2!}$ ，所以，

$$L(F)=\sum_{n=1}^{N}w_m^{(n)}(1-y^{(n)}\alpha_M f_M(x^{(n)}+\frac{1}{2}\alpha_M^2)\propto\alpha_M\sum_{n=1}^{N}w_m^{(n)}\mathbb{I}(y^{(n)}\neq f_m(x^{(n)}))$$

从上式可以看出，当 $\alpha_M>0$ 时，最优的分类器 $f_M(x)$ 为使得在样本权重为 $w_m^{(n)}$ 时的加权错误率最小的分类器。在求解出 $f_M(x)$ 之后，

$L(F)=\sum_{y^{(n)}=f_m(x^{(n)})}w_m^{(n)}e^{-\alpha_m}+\sum_{y^{(n)}\neq f_m(x^{(n)})}w_m^{(n)}e^{\alpha_m}\propto(1-\epsilon_m)e^{-\alpha_m}+\epsilon_m e^{\alpha_m}$ ，其中其中 ϵ_m 为分类器 $f_m(x)$ 的加权错误率，另上式导数为0，计算出 $\alpha_m=\frac{1}{2}log\frac{1-\epsilon_m}{\epsilon_m}$ 。

解释一下GBDT算法?

Gradient Boosting是Boosting中的一大类算法，其基本思想是根据当前模型损失函数的负梯度信息来训练新加入的弱分类器，然后将训练好的弱分类器以累加 的形式结合到现有模型中。在每一轮迭代中，首先计算出当前模型在所有样本上的负梯度，然后以该值为目标训练一个新的弱分类器进行拟合并计算出该弱分类器的权重，最终实现对模型的更新。

采用决策树作为弱分类器的Gradient Boosting算法被称为GBDT（梯度提升决策树），有时又被称为MART(Multiple Additive Regression Tree)。GBDT中使用的决策树通常为CART。

GBDT(Gradient Boosting Decision Tree)，全名叫梯度提升决策树。GBDT使用的损失函数为均方差损失函数： $L=\frac{1}{2}\sum_{n=1}^{N}(f(x)-y)^2$ ，则负梯度是： $-\frac{\partial L}{\partial f}=y-f(x)$ 。所以，当损失函数选用均方损失函数是时，每一次拟合的值就是（真实值 - 当前模型预测的值），即残差。GBDT的下一个弱分类器就是去拟合误差函数对预测值的残差。GBDT非常好地体现了“从错误中学习”的理念，基于决策树预测的残差进行迭代的学习。

GBDT例子

GBDT使用梯度提升为什么有效？它与梯度下降的联系是什么?

GBDT本质上和梯度下降一样，都是用迭代的方法去最小化损失函数，若干个决策树可以认为是每一次迭代对损失函数最小值的若干次迭代逼近。第M次迭代时，整个模型的预测结果应该是前M-1个决策树的预测结果之和。

两者都是在每一轮迭代中，利用损失函数相对于模型的负梯度方向的信息来对当前模型进行更新，只不过在梯度下降中，模型是以参数化形式表示，从而模型的更新等价于参 数的更新。而在

梯度提升中，模型并不需要进行参数化表示，而是直接定义在函数空间中，从而大大扩展了可以使用的模型种类。

GBDT的优点和局限性有哪些?

优点

1. 预测阶段的计算速度快，树与树之间可并行化计算。
2. 在分布稠密的数据集上，泛化能力和表达能力都很好，这使得GBDT在Kaggle的众多竞赛中，经常名列榜首。
3. 采用决策树作为弱分类器使得GBDT模型具有较好的解释性和鲁棒性，能够自动发现特征间的高阶关系。

局限性

1. GBDT在高维稀疏的数据集上，表现不如支持向量机或者神经网络。
2. GBDT在处理文本分类特征问题上，相对其他模型的优势不如它在处理数值特征时明显。
3. 训练过程需要串行训练，只能在决策树内部采用一些局部并行的手段提高训练速度。

GBDT为什么比传统的逻辑回归和SVM效果更好?

GBDT是拿Decision Tree作为GBM里的弱分类器，GBDT的优势 首先得益于 Decision Tree 本身的一些良好特性，具体可以列举如下:

1. Decision Tree 可以很好的处理 missing feature，这是他的天然特性，因为决策树的每个节点只依赖一个 feature，如果某个 feature 不存在，这颗树依然可以拿来做决策，只是少一些路径。像逻辑回归，SVM 就没这个好处。
2. Decision Tree 可以很好的处理各种类型的 feature，也是天然特性，很好理解，同样逻辑回归和 SVM 没这样的天然特性。
3. 对特征空间的 outlier 有鲁棒性，因为每个节点都是 $x < T$ 的形式，至于大多少，小多少没有区别，outlier 不会有什么大的影响，同样逻辑回归和 SVM 没有这样的天然特性。
4. 如果有不相关的 feature，没什么干扰，如果数据中有不相关的 feature，顶多这个 feature 不出现在树的节点里。逻辑回归和 SVM 没有这样的天然特性(但是有相应的补救措施，比如逻辑回归里的 L1 正则化)。
5. 数据规模影响不大，因为我们对弱分类器的要求不高，作为弱分类器的决策树的深度一般设的比较小，即使是大数据量，也可以方便处理。像 SVM 这种数据规模大的时候训练会比较麻烦。

RF(随机森林)与GBDT之间的区别与联系?

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。
- RF和GBDT在使用CART树时，可以是分类树或者回归树。

不同点：

- 组成随机森林的树可以并行生成，而GBDT是串行生成
- 随机森林的结果是多数表决表决的，而GBDT则是多棵树累加之和
- 随机森林对异常值不敏感，而GBDT对异常值比较敏感
- 随机森林是减少模型的方差，而GBDT是减少模型的偏差
- 随机森林不需要进行特征归一化，而GBDT则需要进行特征归一化（因为GBDT的树是在上一颗树的基础上通过梯度下降求解最优解，归一化能收敛的更快）。

XGBoost、LightGBM和GBDT的关系是什么?

GBDT是机器学习算法，XGBoost和LightGBM是该算法的工程实现，分别在工程实现上做了大量的优化。

XGBoost做了哪些优化?

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。

LightGBM在哪些地方进行了优化 (区别XGBoost)?

- 基于Histogram的决策树算法
- 带深度限制的Leaf-wise的叶子生长策略
- 直方图做差加速直接

- 支持类别特征(Categorical Feature)
- Cache命中率优化
- 基于直方图的稀疏特征优化多线程优化。