机器学习基础

特征工程

ONE-HOT 的作用是什么，为什么不直接使用数字作为表示

注意：理解 **One-hot** 编码和数字编码的特点

One-hot 主要用来编码类别特征，即采用哑变量(dummy variables) 对类别进行编

码。它的作用是避免因将类别用数字作为表示而给函数带来抖动。直接使用数字会给将人工误差而导致的假设引入到类别特征中，比如类别之间的大小关系，以及差异关系等等。

什么是数据不平衡，如何解决

数据不平衡主要指的是在有监督机器学习任务中，样本标签值的分布不均匀。这将使得模型更倾向于将结果预测为样本标签分布较多的值，从而使得少数样本的预测性能下降。绝大多数常见的机器学习算法对于不平衡数据集都不能很好地工作。

解决方法：

重新采样训练集

欠采样 –通过减少丰富类的大小来平衡数据集

过采样 – 增加稀有样本，通过使用重复，自举或合成少数类

设计使用不平衡数据集的模型

在代价函数增大对稀有类别分类错误的惩罚权重。

对于树形结构为什么不需要归一化

决策树的学习过程本质上是选择合适的特征，分裂并构建树节点的过程；而分裂节点的标准是由树构建前后的目标增益（比如信息增益和信息增益率）决定的。这些指标与特征值之间的数值范围差异并无关系。

模型评估

请比较欧式距离与曼哈顿距离

欧式距离，即欧几里得距离，表示两个空间点之间的直线距离。

曼哈顿距离，即所有维度距离绝对值之和。

在基于地图，导航等应用中，欧式距离表现得理想化和现实上的距离相差较大；而曼哈顿距离就较为合适；另外欧式距离根据各个维度上的距离自动地给每个维度计算了一个“贡献权重”，这个权重会因为各个维度上距离的变化而动态的发生变化；而曼哈顿距离的每个维度对最终的距离都有同样的贡献权重。

降低过拟合和欠拟合的方法

降低过拟合的方法：

特征 - 减少不必要的特征

根据特征的重要性，直接删除稀疏特征；

通过收集更多的数据，或者用数据增广的方法，产生更多的训练数据；从而阻止模型学习不相关的特征。

模型复杂度 - 降低模型复杂度

神经网络，减少网络层数和神经元个数

决策树模型中降低树的深度，进行剪枝

正则化 - 加入正则化项并提高正则化项的系数

对复杂模型和系数比较大的模型进行惩罚，使得算法倾向于训练简单的模型。

多模型决策

采用 Bagging 或者 Stacking 的集成方法；将多个模型融合起来共同决策；以减少模型预测的 variance.

模型训练

训练模型时采用早停策略或采用知识蒸馏方法进行训练

数据目标 - 平滑目标

比如用于分类任务的标签平滑方法，即在 One-hot 表示的 ground true 标签里面，将值为 1 那一位上的一小部分值减掉，均分到其他值为 0 的位值上。

降低欠拟合的方法：

特征 - 添加新特征

比如上下文特征,ID 类特征, 组合特征等等

模型复杂度 - 增加模型复杂度

比如在线性模型中添加高次项;

在神经网络模型中增加网络层数或者神经元个数。

正则化 - 减少正则化项的系数

机器学习模型

线性回归与逻辑回归

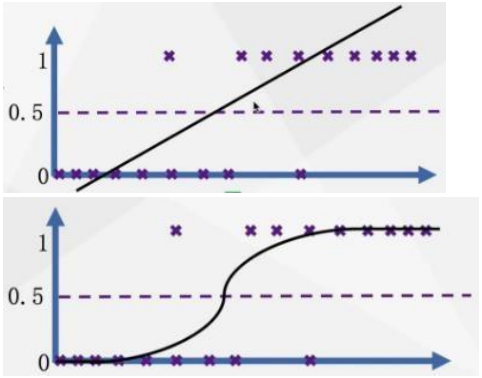
逻辑回归相比线性回归，有何异同

逻辑回归和线性回归之间既有区别又有联系。逻辑回归和线性回归最大的不同点是逻辑回归解决的是分类问题而线性回归则解决的是回归问题。逻辑回归又可以认为是广义线性回归的一种特殊形式，其特殊之处在于其目标(label/target)的取值服从二元分布。

所谓逻辑回归是一种特殊的广义线性回归，我们可以通过狭义线性回归到逻辑回归的转化过程来理解。狭义线性回归的表达式可表示为：

$$y = w * x + b$$

如果我们希望这个模型可以对二分类任务做出预测，即目标满足 0,1 分布。那么希望预测出来的值经过某种转换之后，大部分可以分布在 0,1 两个值附近。



我们知道逻辑回归的求值计算其实就是在线性回归的基础上，再做一个sigmoid计算。所以它们都可以用相同的方法比如梯度下降来求解参数。

常见分类问题的度量指标

分类问题度量指标的基础是混淆矩阵。

		预测类别		
		Yes	No	总计
实际类别	Yes	TP	FN	P（实际为Yes）
	No	FP	TN	N（实际为No）
	总计	P'（被分为Yes）	N'（被分为No）	P+N

准确率 - 所有预测正确的样本（正样本预测为正，负样本预测为负）与所有样本的比值。

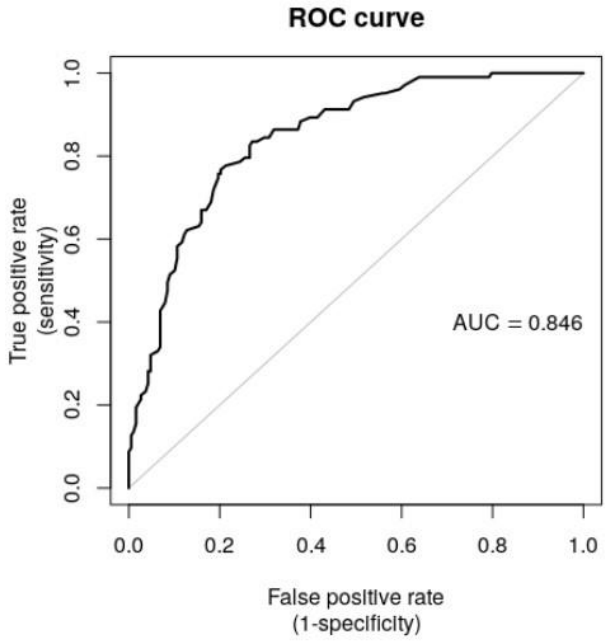
精确率 - 精确率是针对我们预测结果而言的，它表示的是预测为正的样本中有多少是真正的正样本。那么预测为正就有两种可能了，一种就是把正类预测为正类(TP)，另一种就是把负类预测为正类(FP)，

召回率-召回率是针对我们原来的样本而言的，它表示的是样本中的正例有多少被预测正确了。那也有两种可能，一种是把原来的正类预测成正类(TP)，另一种就是把原来的正类预测为负类(FN)。

F1 值-F1 值是精确率和召回率的调和值

AUC (Area Under Curve)，曲线下的面积。这里的 Curve 指的的是 ROC (receiver operating characteristic curve) 接收者操作特征曲线，是反映敏感性 (sensitivity)和特异性(1-specificity)连续变量的综合指标，ROC 曲线上每个点反映着对同一信号刺激的感受性。ROC 曲线是通过取不同的阈值来分别计算在每个阈值

下，伪正类率 FPR（False Positive Rate）和真正类率 TPR （True Positive Rate）的值来绘制的。



朴素贝叶斯

朴素贝叶斯有没有超参数可以调

基础朴素贝叶斯模型的训练过程，本质上是通过数学统计方法从训练数据中统计先验概率$P(c)$和后验概率$P(x_i|c)$；而这个过程是不需要超参数调节的。所以朴素贝叶斯模型没有可调节的超参数。虽然在实际应用中朴素贝叶斯会与拉普拉斯平滑修正

（Laplacian Smoothing Correction）一起使用，而拉普拉斯平滑修正方法中有平滑系数这一超参数，但是这并不属于朴素贝叶斯模型本身的范畴。

朴素贝叶斯对异常值敏不敏感

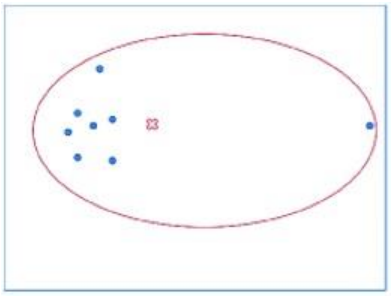
基础的朴素贝叶斯模型的训练过程，本质上是通过数学统计方法从训练数据中统计先验概率$P(c)$和后验概率$P(x_i|c)$；少数的异常值，不会对统计结果造成比较大的影响。

所以朴素贝叶斯模型对异常值不敏感。

K-MEANS

KMEANS 对异常值是否敏感,为何

Kmeans 对异常值较为敏感，因为一个集合内元素的均值易受到一个极大值的影响。如图中展示的结果，因为有了一个异常值，这个元素集合的中心点严重偏离了大多数点所在的位置区域；因此在有异常值影响的情况下，均值所计算出来的中心位置不能够反映真实的类别中心。



如何评估聚类效果

聚类往往不像分类一样有一个最优化目标和学习过程，而是一个统计方法，将相似的数据和不相似的数据分开。所以，评估聚类效果可以从以下维度入手：聚类趋势(对数据进行评估)

霍普金斯统计量（Hopkins Statistic）评估给定数据集是否存在有意义的可聚类的非随机结构。如果一个数据集是由随机、均匀的点生成的，虽然也可以产

生聚类结果，但该结果没有意义。聚类的前提需要数据是非均匀分布的。该值在区间[0, 1]之间， [0.01, 0.3]表示数据接近随机分布，该值为 0.5 时数据是均匀分布的，[0.7, 0.99] 表示聚类趋势很强。

超参数 K 如何选择

根据业务，比如业务需要把用户分成高中低三种，则 k 选择为 3；

观察法，对于低纬度数据适用；对数据进行可视化或者 PCA 降维可视化之后，可大致观测出数据分布自然区分的类别数；

手肘法

所有样本点到它所存在的聚类中心点的距离之和，作为模型好坏的衡量标准。

具体步骤为：

计算$D_k = \sum_{i=1}^{k} \sum_{X \in Ci} ||X - M_i||^2$（$k$越大$D_k$会越小）； ○ 绘制$D_k$关于$k$的函数图像；

观察是否存在某个值，当k大于这个值之后，D_k的减小速度明显变缓或者基本不变，则这个值所在的点即为拐点，拐点处的 k 值就是要选择的最佳分类个数。

Gap Statistic

Gap Statistic 计算公式为：

$$Gap(K) = E(logD'_k) - logD_k$$

采用均匀分布模型抽样产生和原始样本一样多的随机样本，对随机样本做

Kmeans 得到D'_k，重复多次可以获得$E(logD'_k)$，Gap statistic 取最大值所对应的 k 就是最佳的 k。Gap Statistic 方法借鉴了蒙特卡洛算法的思想。

KMEANS 算法的优缺点

优点

容易理解，聚类效果不错，虽然是局部最优， 但往往局部最优就能够给出一个不错的结果；

处理大数据集的时候，该算法可以保证较好的伸缩性 - 即稍作改进即可处理大数据集；

当数据簇近似高斯分布的时候，效果非常不错；

算法复杂度低 – 时间复杂度为$O(n*k*t)$ 其中n为样本数，k为类别数，t为迭代次数；而且通常来说k和t 相对于n来说很小。

缺点

k 值需要人为设定，不同 k 值得到的结果不一样；

对初始的类别中心敏感，不同选取方式会得到不同的结果；

对异常值敏感；

样本只能归为一类，不适合多类别多标签分类任务；

不适合太离散的分类、样本类别不平衡的分类以及同类样本分布呈非凸形状的分类。

SVM

请简述 SVM 原理

SVM 是一种二类分类模型。它的基本模型是在特征空间中寻找间隔最大化的分离超平面的线性分类器。

当训练样本线性可分时，通过硬间隔最大化，学习一个线性分类器，即线性可分支持向量机；

当训练数据近似线性可分时，引入松弛变量，通过软间隔最大化，学习一个线性分类器，即线性支持向量机；

当训练数据线性不可分时，通过使用核技巧及软间隔最大化，学习非线性支持向量机。

SVM 为什么采用间隔最大化

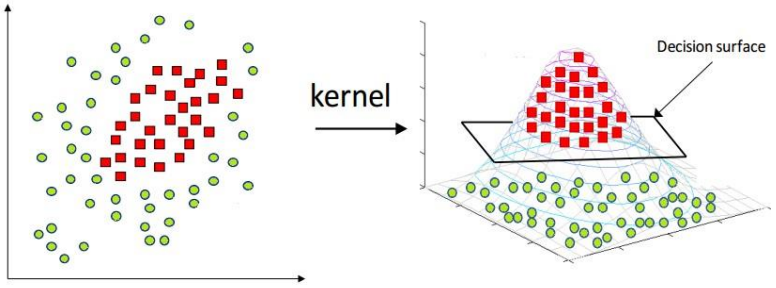
当训练数据线性可分时，存在无穷多个分离超平面可以将两类数据正确分开。感知机利用误分类最小策略，求得分离超平面，不过此时的解有无穷多个。线性可分支持向量机利用间隔最大化求得最优分离超平面，这时，解是唯一的。另一方面，此时的分隔超平面所产生的分类结果是鲁棒的，对未知实例的泛化能力最强。

SVM 为什么要引入核函数

当样本在原始空间线性不可分时，可将样本从原始空间映射到一个更高维的特征空

间，使得样本在这个特征空间内线性可分。而引入这样的映射后，在所要求解的问题中，无需求解真正的映射函数，而只需要知道其核函数。核函数的定义为：$K(x,y)=<\phi(x),\phi(y)>$，即在特征空间的内积等于它们在原始样本空间中通过核函数 K 计算的结果。

SVM引入核函数之后，一方面数据变成了高维空间中线性可分的数据；另一方面不需要求解具体的映射函数，只需要给定具体的核函数即可，这样使得求解的难度大大降低。



SVM 核函数之间的区别

SVM 常用的核函数有线性核函数，多项式核函数，高斯核（RBF），拉普拉斯核和

Sigmoid 核函数。其中多项式核函数，高斯核（RBF），拉普拉斯核和 Sigmoid 核函数通常用来处理线性不可分的情形。而一般选择核函数时通常考虑线性核和高斯核，也就是线性核与 RBF 核。 线性核：主要用于线性可分的情形，参数少，速度快，对于一般数据，分类效果已经很理想了。 RBF 核：主要用于线性不可分的情形，参数多，分类结果非常依赖于参数。有很多人是通过训练数据的交叉验证来寻找合适的参数，不过这个过程比较耗时。

为什么 SVM 对缺失数据和噪声数据敏感

这里说的缺失数据是指缺失某些特征数据，向量数据不完整。

SVM 没有处理缺失值的策略。而 SVM 希望样本在特征空间中线性可分，所以特征空间的好坏对 SVM 的性能很重要。缺失特征数据将影响训练结果的好坏。

另外，SVM 对噪声数据也较为敏感；原因是 SVM 的决策只基于少量的支持向量，若噪音样本出现在支持向量中，容易对决策造成影响，即影响目标函数中损失项的收敛，所以 SVM 对噪音敏感。

SVM 算法的优缺点

优点：

可以有效解决高维特征的分类和回归问题；

无需依赖全体样本，只依赖支持向量；

有大量的核技巧可以使用，从而可以应对线性不可分问题；

对于样本量中等偏小的情况，仍然有较好的效果。

缺点：

如果特征维度远大于样本个数，SVM 表现一般；

SVM 在样本巨大且使用核函数时，计算量很大；

在应对非线性可分数据时，核函数的选择依旧没有统一的标准；

SVM 对缺失和噪声数据敏感；

特征的多样性限制了 SVM 的使用，因为 SVM 本质上是属于一个几何模型，这个模型需要用 Kernel 定义样本之间的相似度（线性 SVM 中的内积），而我们无法预先为所有特征设定一个统一的相似度计算方法。这样的数学模型使得 SVM 更适合去处理“同性质”的特征。

SVM 的超参数 C 如何调节

在使用 SVM 库时候，通常有两个需要手工调节的关键参数

参数 C

核函数(Kernel)

由于 C 可以看做与正则化参数 λ 作用相反 （C 作用于损失项上，而 λ 作用于正则化项上，见简述 SVM 软间隔推导过程 ），则对于 C 的调节：

低偏差，高方差，即遇到了过拟合时；减少 C 值。

高偏差，低方差，即遇到了欠拟合时：增大 C 值。

SVM 的核函数如何选择

当特征维度n较高，而样本规模m较小时，不宜使用核函数或者选用线性核的 SVM，否则容易引起过拟合。

当特征维度n较低，而样本规模m足够大时，考虑使用高斯核函数。不过在使用高斯核函数前，需要进行特征缩放(feature scaling)。

树模型

决策树的优缺点

优点：

缺失值不敏感，对特征的宽容程度高 – 可缺失可连续可离散；

可以计算特征重要性，且可解释性强；

算法对数据没有强假设；

可以解决线性及非线性问题；

有特征选择等辅助功能。

缺点：

处理关联性数据比较薄弱 – 重要性强且关联的特征都得到了重视；

正负量级有偏样本的样本效果较差；

单棵树的拟合效果欠佳，容易过拟合。

决策树如何防止过拟合,说说具体方法

我们在讨论防止机器学习过拟合的时候，通过分类的方法，大致确立了这么几个改进方向，1）数据 2）模型 3）正则化 4）训练。我们重点讨论如何通过改进决策树模型来防止过拟合，当然其他几个方向对防止决策树过拟合同样适用。

通过改进模型来防止过拟合的主要思路是简化模型，使得模型能够学习样本中的共同特征（即主要特征），而摈弃个性化的特征（即次要特征）。而对树模

型进行简化的方法又可以分为预剪枝（在训练过程中进行剪枝），和后剪枝（在决策树构建完成之后进行剪枝）

预剪枝的主要方法有：

限制树的深度 - 当树到达一定深度的时候，停止树的生长

限制叶子节点的数量

规定叶子区域内最少的样本数，未达到最少样本数的叶子区域不做分裂

计算每次分裂对测试集的准确度提升

后剪枝的核心思想是让算法生成一颗完全生长的决策树，然后最底层向上计算是否剪枝。剪枝过程将子树删除，用一个叶子节点替代，该节点的类别同样按照多数投票的原则进行判断。同样地，后剪枝也可以通过在测试集上的准确率进行判断，如果剪枝过后准确率有所提升，则进行剪枝。相比于预剪枝，后剪枝方法通常可以得到泛化能力更强的决策树，但时间开销会更大。

集成学习

BAGGING & BOOSTING

什么是集成学习算法

集成学习(Ensemble Learning)就是将多个机器学习模型组合起来，共同工作以达到优化算法的目的。集成学习的一般步骤是：1.生产一组 “个体学习器”(Individual leaner);2.用某种策略将它们结合起来。

个体学习器通常由一个学习算法通过训练数据训练产生。在同质集成（系统中个体学习器的类型相同）中，个体学习器又被称为“基学习器”而在异质集成（系统中个体学习的类型不同）中，个体学习器又被称为“组件学习器”(component learner)。

集成学习的思想类似我们俗话常说的“三个臭皮匠胜过一个诸葛亮”。

集成学习主要有哪几种框架,分别简述这几种集成学习框架的工作过程

集成学习主要的集成框架有，Bagging，Boosting 和 Stacking。其中 Bagging 和

Boosting 为同质集成，而 Stacking 为异质集成。

Bagging （Bootstrap Aggregating）；Bagging 的核心思想为并行地训练一系列各自独立的同类模型，然后再将各个模型的输出结果按照某种策略进行聚合（例如分类中可采用投票策略，回归中可采用平均策略）。Bagging 方法主要分为两个阶段：

Bootstrap 阶段，即采用有放回的抽样方法，将训练集分为 n 个子样本集；并用基学习器对每组样本分别进行训练，得到 n 个基模型。

Aggregating 阶段，将上一阶段训练得到的 n 个基模型组合起来，共同做决策。在分类任务中，可采用投票法。比如相对多数投票法，即将结果预测为得票最多的类别。而在回归任务中可采用平均法，即将每个基

模型预测得到的结果进行简单平均或者加权平均来获得最终的预测结果。

Stacking；Stacking 的核心思想为并行地训练一系列各自独立的不同类模型，然后通过训练一个元模型（meta-model）来将各个模型的输出结果进行结合。也可以用两个阶段来描述 Stacking 算法：

第一阶段，分别采用全部训练样本训练 n 个组件模型，要求这些个体学习器必须是异构的，也就是说采用的学习方法不同；比如可以分别是线性学习器，SVM，决策树模型和深度模型。

第二阶段，训练一个元模型（meta-model）来将各个组件模型的输出结果进行结合。具体过程是，将各个学习器在训练集上得到的预测结果作为训练特征和训练集的真实结果组成新的训练集；用这个新组成的训练集来训练一个元模型。这个元模型可以是线性模型或者树模型。

Boosting：Boosting 的核心思想为串行地训练一系列前后依赖的同类模型，即后一个模型用来对前一个模型的输出结果进行纠正。Boosting 算法是可将弱学习器提升为强学习的算法。学习过程是：先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本进行调整，使得先前基学习器做错的训练样本在后续训练中受到更多关注，然后将当前基学习器集成到集成学习器中并基于调整后的样本分布来训练下一个基学习器；如此重复进行，直至基学习器数目达到事先指定的值 T。

BOOSTING 算法有哪两类，它们之间的区别是什么

Boosting 算法主要有 AdaBoost（Adaptive Boost）自适应提升算法和 Gradient

Boosting 梯度提升算法。最主要的区别在于两者如何识别和解决模型的问题。

AdaBoost 用分错的数据样本来识别问题，通过调整分错的数据样本的权重来改进模型。

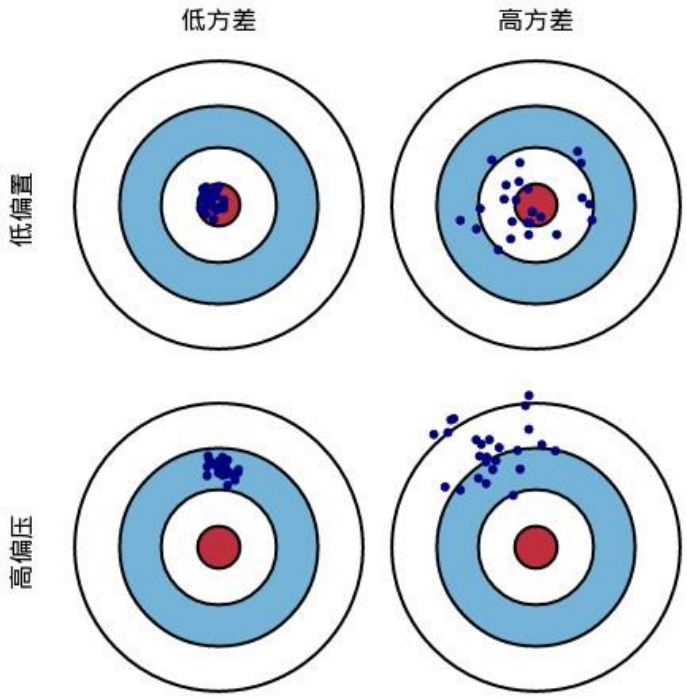
Gradient Boosting 通过负梯度来识别问题，通过计算负梯度来改进模型。

什么是偏差和方差

偏差指的是预测值的期望与真实值之间的差距，偏差越大，预测值越偏离真实数据的标签。方差描述的是预测值的变化范围，离散程度，也就是离预测值期望的距离，方差越大，数据的分布越分散。

可通过打靶射击的例子来做类比理解，我们假设一次射击就是一个机器学习模型对一个样本进行预测，射中红色靶心位置代表预测准确，偏离靶心越远代表预测误差越大。偏差则是衡量射击的蓝点离红圈的远近，射击位置即蓝点离红色靶心越近则偏差越小，蓝点离红色靶心越远则偏差越大；方差衡量的是射击

是否稳定，即射击的位置蓝点是否聚集，蓝点越集中则方差越小，蓝点越分散则方差越大。



随机森林

简述一下随机森林算法的原理

随机森林算法是 Bagging 集成框架下的一种算法，它同时对训练数据和特征采用随机抽样的方法来构建更加多样化的基模型。随机森林具体的算法步骤如下：

假设有N个样本，则有放回的随机选择N个样本(每次随机选择一个样本，然后将该样本放回并继续选择)。采用选择好的N个样本用来训练一个决策树，作为决策树根节点处的样本。

假设每个样本有M个属性，在决策树做节点分裂时，随机从这M个属性中选取m个属性，满足条件$m \ll M$。然后采用某种策略（比如信息增益最大化）从m个属性中选择一个最优属性作为该节点的分裂属性。

决策树形成过程中重复步骤 2 来计算和分裂节点。一直到节点不能够再分裂，或者达到设置好的阈值（比如树的深度，叶子节点的数量等）为止。注意整个决策树形成过程中没有进行剪枝。

重复步骤 1~3 建立大量的决策树，这样就构成了随机森林。

随机森林的随机性体现在哪里

随机森林的随机性体现在每颗树的训练样本是随机的，树中每个节点的分裂属性集合也是随机选择确定的。

随机采样：随机森林在计算每棵树时，从全部训练样本（样本数为N）中选取一个可能有重复的、大小同样为N的数据集进行训练（即Boostrap采样）。 特征选取的随机性：在节点分裂计算时，随机地选取所有特征的一个子集，用来计算最佳分割方式。

随机森林算法的优缺点

优点特征和数据的随机抽样

它可以处理很高维度（特征很多）的数据，并且不用降维，无需做特征选择；

如果有很大一部分的特征遗失，仍可以维持准确度；

不容易过拟合；

对于不平衡的数据集来说，它可以平衡误差；

可以判断出不同特征之间的相互影响（类似于控制变量法）；树模型的特性

较好的解释性和鲁棒性；

能够自动发现特征间的高阶关系；

不需要对数据进行特殊的预处理如归一化；算法结构

训练速度比较快，容易做成并行方法；

实现起来比较简单。

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。（决策树的学习本质上进行的是决策节点的分裂，依赖于训练数据的空间分布）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。

随机森林为什么不能用全样本去训练 m 棵决策树

随机森林的基学习器是同构的，都是决策树，如果用全样本去训练 m 棵决策树的话；基模型之间的多样性减少，互相相关的程度增加，不能够有效起到减少方差的作用；对于模型的泛化能力是有害的。

随机森林和 GBDT 的区别

随机森林采用的 Bagging 思想，而 GBDT 采用的 Boosting 思想。

组成随机森林的树可以并行生成；而 GBDT 只能是串行生成。

随机森林对异常值不敏感；GBDT 对异常值非常敏感。

随机森林对训练集一视同仁；GBDT 对训练集中预测错误的样本给予了更多关注。

随机森林是通过减少模型方差提高性能；GBDT 是通过减少模型偏差提高性能。

对于最终的输出结果而言，随机森林采用多数投票等方法；而 GBDT 则是将所有结果累加起来，或者加权累加起来。

组成随机森林的树可以是分类树，也可以是回归树；而 GBDT 只能由回归树组成。

GBDT

简述 GBDT 原理

梯度提升树的训练过程大致是这样的：

根据训练集训练一颗初始决策树；

计算之前所有树在此数据集上预测结果之和与真实结果的差值，又叫做残差。

把残差作为当前树拟合的目标在训练集上训练。

重复 2,3 步骤，直到达到设置的阈值（树的个数，早停策略等）采用伪代码表示如下：

输入：训练数据集 $D=\{(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)\}$，其中$x_i \in x \subseteq R^n, y_i \in y \subseteq R$;
输出：提升树$f_M(x)$.

初始化$f_0(x)=0$

对$m=1,2,\dots,M$

计算残差

$r_{mi}=y_i-f_{m-1}(x_i), \quad i=1,2,\dots,N$

拟合残差r_{mi}学习一个回归树，得到$T(x,\theta_m)$

更新 $f_m(x)=f_{m-1}(x)+T(x,\theta_m)$

得到回归问题的提升树

为什么 GBDT 不适合使用高维稀疏特征

高维稀疏特征会使树模型的训练变得极为低效，且容易过拟合。以 ID 类型特征为例进行说明：

树模型训练过程是一个贪婪选择特征的算法，要从候选特征集合中选择一个使分裂后收益函数增益最大的特征来分裂。按照高维的 ID 特征做分裂时，子树数量非常多，计算量会非常大，训练会非常慢。

同时，按 ID 分裂得到的子树的泛化性也比较弱，由于只包含了对应 ID 值的样本，样本稀疏时也很容易过拟合。

GBDT 算法的优缺点

优点：

预测阶段的计算速度快，树与树之间可并行化计算（注意预测时候可并行）。

在分布稠密的数据集上，泛化能力和表达能力都很好。

采用决策树作为弱分类器使得 GBDT 模型具有 1）较好的解释性和鲁棒性，2）能够自动发现特征间的高阶关系，并且也 3）不需要对数据进行特殊的预处理如归一化等。

缺点:

GDBT 在高维稀疏的数据集上，表现不佳。

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。

XGBOOST

简述 XGBOOST

XGBoost 是陈天奇新开发的 Boosting 库。它是一个大规模、分布式的通用 Gradient Boosting 库，它在 Gradient Boosting 框架下实现了 GBDT 和一些广义的线性机器学习算法。

XGBOOST 和 GBDT 有什么不同

GBDT 是机器学习算法，XGBoost 是该算法的工程实现；

在使用 CART 作为基础分类器时，XGBoost 显示地加入了正则项来控制模型的复杂度，有利于防止过拟合，从而提高模型的泛化能力；

GBDT 在模型训练时只使用了损失函数的一阶导数信息，XGBoost 对代价函数进行二阶泰克展开，可以同时使用一阶和二阶导数；

传统的 GBDT 采用 CART 作为基础分类器，XGBoost 支持多种类型的基础分类器，比如线性分类器；

传统的 GBDT 在每轮迭代时使用全部的数据，XGBoost 则支持对数据进行采样；

传统的 GBDT 没有涉及对缺失值进行处理，XGBoost 能够自动学习出缺失值的处理策略；

XGBoost 还支持并行计算， XGBoost 的并行是基于特征计算的并行，将特征列排序后以 block 的形式存储在内存中，在后面的迭代中重复使用这个结构。

XGBOOST 为什么可以并行训练

注意 XGBoost 的并行不是树粒度的并行，XGBoost 也是一次迭代完才能进行下一次迭代的（第 t 次迭代的代价函数里包含了前面 t-1 次迭代的预测值）。

XGBoost的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost在训练之前，预先对数据进行了排序，然后保存为 block 结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个 block 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以在多线程进行。

树节点在进行分裂时，我们需要计算每个特征的每个分割节点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以 XGBoost 还提出了一种可并行的直方图算法，用于高效地生成候选的分割点。

XGBoost 防止过拟合的方法

数据

样本采样 - 在生成每颗树的时候可以对数据进行随机采样。

特征采样 - 还可以在生成每棵树的时候，每棵树生成每一层子节点的时候，以及在每次做节点分裂的时候，选择是否对特征进行采样。

模型

限制树的深度，树的深度越深模型越复杂。

设置叶子节点上样本的最小数量，这个值越小则树的枝叶越多模型越复杂；相反如果这个值越大，则树的枝叶越少，模型越简单。这个值太小会导致过拟合，太大会导致欠拟合。

正则化

L1 和 L2 正则化损失项的权重

叶子节点数量惩罚项的权重值

XGBoost 为什么这么快

同时采用了损失函数的一阶导数和二阶导数，使得目标函数收敛更快。

在进行节点分类时采用的贪心算法和直方图算法，大大加速了节点分裂的计算过程。

工程优化，使得模型训练时可进行并行计算。

深度学习基础

非线性

为什么必须在神经网络中引入非线性

如果神经网络中没有引入非线性层，那么神经网络就变成了线性层的堆叠。而多层线性网络的堆叠本质上还是一个线性层。我们知道线性函数的表现力是有限的，它只能表示特征与目标值之间比较简单的关系，相反带有非线性层的神经网络被证明可以表示任何函数。所以为了使得网络设计发挥作用，并且提高网络的表现力，必须要在神经网络中引入非线性。

ReLU 的优缺点

优点：

使用 ReLU 的 SGD 算法的收敛速度比 sigmoid 和 tanh 快；

在 x>0 上，不会出现梯度饱和，梯度消失的问题。

计算复杂度低，不需要进行指数运算，只要一个阈值（0）就可以得到激活值。

缺点：

ReLU 的输出不是 0 均值的，它将小于 0 的值都置为 0; 使得所有参数的更新方向都相同，导致了 ZigZag 现象。

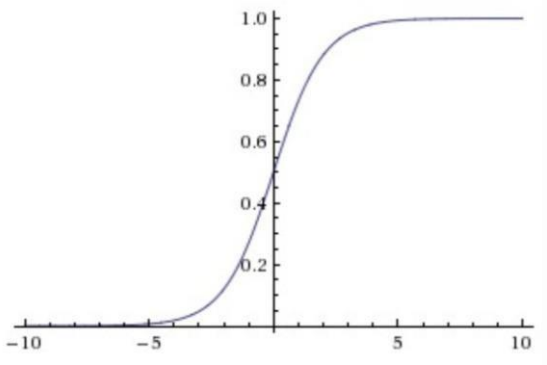
Dead ReLU Problem (ReLU 神经元坏死现象)：某些神经元可能永远不被激活，导致相应参数永远不会被更新(在负数部分，梯度为 0) ReLU 不会对数据做幅度压缩，所以数据的幅度会随着模型层数的增加不断扩张。 在同一个线性层中所有参数的导数符号都相同，而且只取决于 Loss 函数对当前层函数导数的符号。

激活函数有什么作用，常用的激活函数

如果不用激活函数（其实相当于激活函数是 f(x) = x），在这种情况下你每一层节点的输入都是上层输出的线性函数，所以无论神经网络有多少层，输出都是输入的线性组合。这种情况就是最原始的感知机了，那么网络的表达能力就相当有限。正因为上面的原因，引入非线性函数作为激活函数之后，深层神经网络表达能力就很强大了（不再是输入的线性组合，而是几乎可以表示任意函数）。

常用的激活函数有：Sigmoid, Tanh, ReLU, Leaky ReLU,

Sigmoid 激活函数：



Sigmoid 将回归问题的结果转换到(0,1)的范围，图像成中间窄两边宽的结构；即大部分的时候 Sigmoid 的值集中的在两边。Sigmoid 函数提供了一个很好的适用于二分类的问题的输出结果，以及非线性表达。但是它的缺点也是非常明显的：

前向传播和反向传播都要做指数计算，计算耗费资源较多。

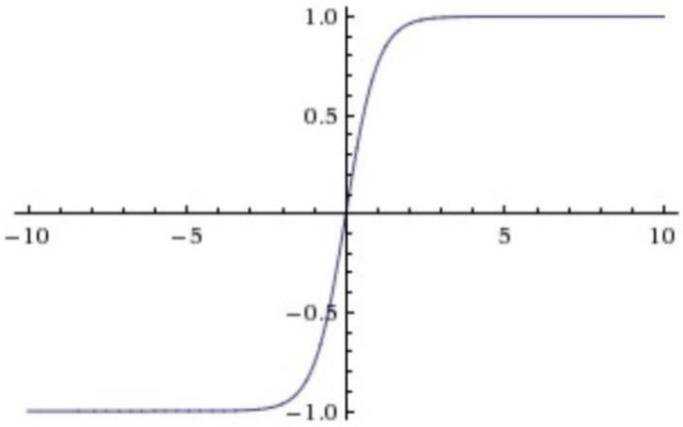
Sigmoid 函数梯度饱和导致神经网络反向传播时出现梯度消失的现象。当神经元的激活在接近 0 或 1 处时会饱和，在这些区域梯度几乎为 0，这就会导致梯度消失，几乎就有没有信号通过传回上一层。

Sigmoid 函数的输出不是零中心的。因为如果输入神经元的数据总是正数，那

么关于 w 的梯度在反向传播的过程中，将会要么全部是正数，要么全部是负数，这将会导致梯度下降权重更新时出现 Z 字型的下降（ZigZag 现象）。

Tanh 函数的数学公式是：

$\tanh(x) = 2sigmoid(2x) - 1$ 其图像为：



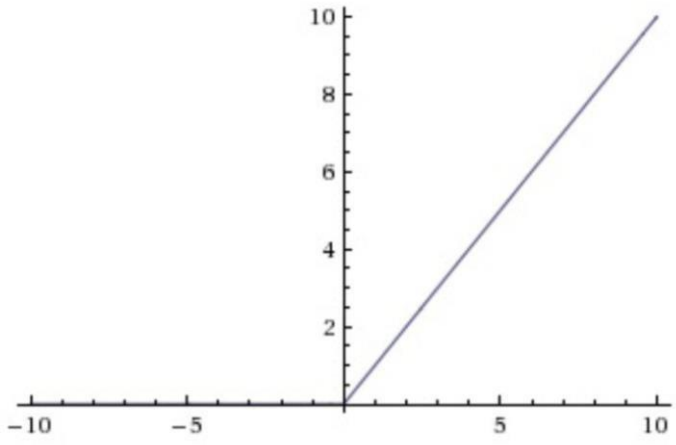
Tanh 激活函数的提出虽然解决了 sigmoid 函数的输出不是 0 中心的问题。但是它仍然存在两个主要问题：

函数梯度饱和的问题；

计算复杂的问题。

ReLU 函数的数学公式是：

$f(x) = \max(0, x)$ 其图像为：



ReLU 激活函数相较于 Sigmoid 和 Tanh 函数来说，对于随机梯度下降收敛有巨大的作用，很好地解决了梯度消失的问题；同时函数的前向和反向传播计算计算效率也高。

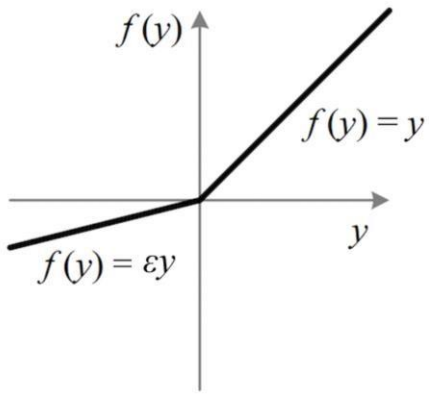
但是 ReLU 存在以下缺点：

ReLU 的输出不是零中心的，所以梯度下降时存在 ZigZag 现象。

ReLU 函数存在小于等于 0 一侧，神经元坏死的现象。

Leaky ReLU 函数的公式是：

$f(y) = \max(y, y)$ 函数的图像为：



其中 通常取很小的值比如 0.01，这样可以确保负数信息没有全部丢失。解决了 ReLU 神经元坏死的问题；并且一定程度上缓解了数据均值不为 0 导致的 ZigZag 现象。

梯度训练

怎么解决梯度消失问题

在深度网络中，网络参数的学习是通过反向传播的链式求导法则来求 Loss 对某个参数的偏导数，然后进行参数更新的。因此造成梯度消失的原因主要有两个：1.当网络层数很深，而当前的参数所在层又靠近网络的输入时，求导链就会非常长；2.如果其中的某些中间结果的值很小，并经过链式的累成作用，最终求得的梯度值就会接近于零，而导致参数得不到更新。 可通过以下方法解决梯度消失的问题：

选用合适的激活函数。比如 ReLU 或者 Leaky ReLU。因为像 Sigmoid 和 Tanh 这样的激活函数，会出现比较大的梯度饱和区域，使得梯度的取值接近于 0。采用 Batch Normalization 层，对网络中计算得到中间值进行归一化，使得中间计算结果的取值在均值为 0，方差为 1 这样的分布内。那么此时，在 sigmoid 和 tanh 中，函数取值处于中间变化较大的部分，梯度取值也相对较大，从而可以防止过拟合。 使用残差结构，残差结构相当于给靠近输入端的网络层提供了一个与靠近输出端层的直连操作。在反向传播计算时，减少了梯度传播的路径长度，以缓解梯度消失的问题。

在 RNN 网络中，可以通过使用 LSTM（long-short term memory networks）长短时记忆网络，来解决信息遗忘和梯度传播的问题。

计算机视觉

CNN 经典模型

AlexNet, VGG，GoogleNet，ResNet 等网络之间的区别是什么

AlexNet 相比传统的 CNN，主要改动包括 Data Augmentation（数据增强）、Dropout 方法、激活函数用 ReLU 代替了传统的 Tanh 或者 Sigmoid、采用 Local Response Normalization（LRN，实际就是利用临近的像素数据做归一化）、Overlapping Pooling（有重叠，即 Pooling 的步长比 Pooling Kernel 的对应边要小）、多 GPU 并行。

VGG 很好地继承了 AlexNet 的特点，采用了更小的卷积核堆叠来代替大的卷积核，并且网络更深。

GoogLeNet，网络更深，但主要的创新在于他的 Inception，这是一种网中网（Network In Network）的结构，即原来的节点也是一个网络。相比于前述几个网络，用了 Inception 之后整个网络结构的宽度和深度都可扩大，能够带来 2-3 倍的性能提升。

ResNet 在网络深度上有了进一步探索。但主要的创新在残差网络，网络的提出本质上还是要解决层次比较深的时候无法训练的问题。这种借鉴了 Highway Network 思想的网络相当于旁边专门开个通道使得输入可以直达输出，而优化的目标由原来的拟合输出 $H(x)$变成输出和输入的差$H(x)-x$，其中$H(x)$是某一层原始的期望映射输出，x是输入。

Pooling 层做的是什么

池化层（Pooling Layer）也叫子采样层（Subsampling Layer）它的一个很重要的作用就是做特征选择，减少特征数量，从而减少网络的参数数量。采用了 Pooling Layer 之后，在不增加参数的情况下，可以增大网络对图像的感受野。虽然可以用带步长的卷积（步长大于 1）代替 Pooling layer 也可以实现下采样和增大感受野的作用，但是 Pooling Layer 的计算简单，同时也没有可学习的参数，在很多需要做特征图拼接的网络，比如 ShuffleNet 中是非常适用的。

Dropout 是否用在测试集上

在训练的时候 Dropout 打开，功能类似于每次将网络进行采样，只训练一部分网络。而训练结束，在测试集上做测试的时候，就要将这些每次训练好的“子网络”集成起来，一起做决策。所以要关闭 Dropout,即关闭网络采样功能。

INCEPTION MODULE 的优点是什么

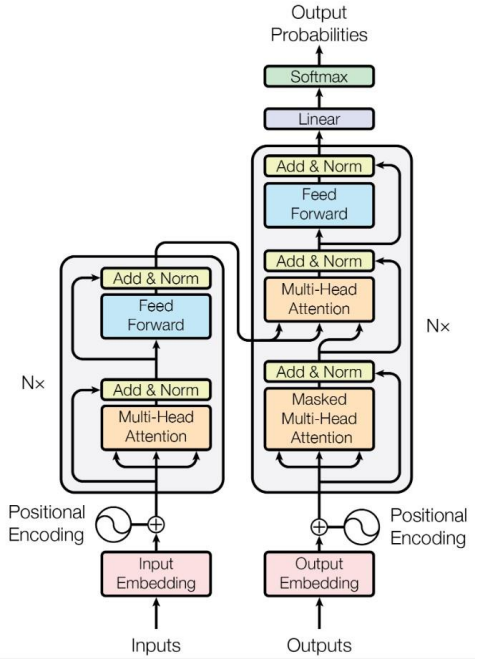
Inception Module 基本组成结构有四个成分。1*1 卷积，3*3 卷积，5*5 卷积，3*3 最大池化。最后对四个成分运算结果进行通道上组合。这就是 Inception Module 的核心思想。通过多个卷积核提取图像不同尺度的信息，最后进行融合，可以得到图像更好的表征。

自然语言处理

TRANSFORMER

写出 SELF-ATTENTION 的公式，SELF-ATTENTION 机制里面的 Q,K,V 分别代表什么

Self-Attention 机制中，Q,K,V 分别指的是 Query, Key 和 Value；每个输入向量的 Q,K,V 向量是由该输入向量分别与权重矩阵W_q, W_k, W_v做相乘计算得到。通过当前位置的 Q 与其他位置的 K 做相乘计算，可得到当前位置对目标位置的 attention 权重；再将这个 attention 权重（经过 Softmax 后）与目标位置的 Value 相乘就得到了当前位置对目标位置的 attention 编码。将当前位置对所有位置（包括它自己）attention 编码进行相加就得到了当前位置的输出编码。



SELF-ATTENTION 中的计算为什么要使用根号 d_k缩放

假设两个 d_k 维向量每个分量都是一个相互独立的服从标准正态分布的随机变量，那么他们的点乘结果会变得很大，并且服从均值为 0，方差为 d_k 的分布，

而很大的点乘会让 Softmax 函数处在梯度很小的区域（我们在讲 Softmax 和 Sigmoid 关系的时

候讲过，面对二分类情况的时候 Softmax 就退化成了 Sigmoid 函数；我们知道 Sigmoid 函数在输入数值较大的区域存在梯度饱和的现象），对每一个分量除以 $sqrt(d_k)$ 可以让点乘的方差变成 1。