机器学习面试干货精讲

2018-01-23 沧潇雨浪 来源 阅 53 分享： 微信 转藏到我的图书馆

来源： de,light-GitChat技术杂谈 链接：

http://mp.weixin.qq.com/s/ixt_ceh6RJuSzL0Uwcwsqg

序言

本文尽可能的不涉及到繁杂的数学公式，把面试中常问的模型核心点，用比较通俗易懂但又不是专业性的语言进行描述。

希望可以帮助大家在找工作时提纲挈领的复习最核心的内容，或是在准备的过程中抓住每个模型的重点。

实战环境说明：

Python 2.7；

Sklearn 0.19.0；

graphviz 0.8.1 决策树可视化。

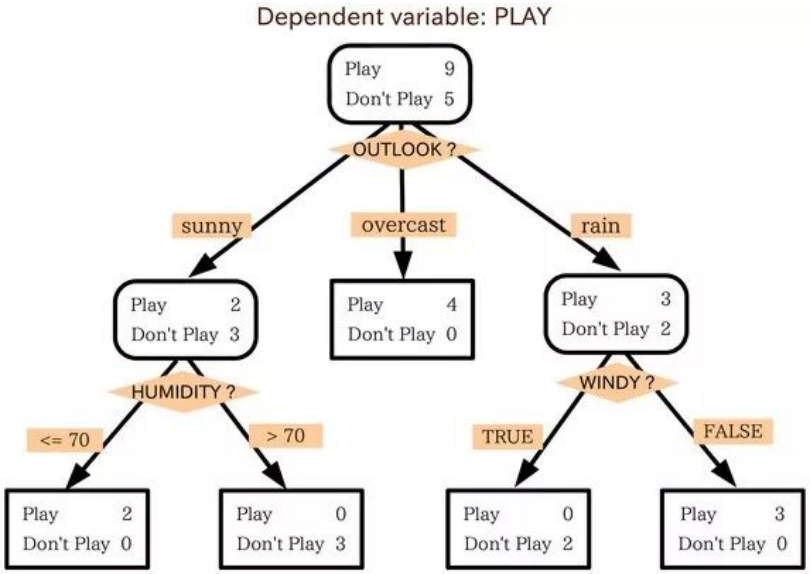
一、决策树

1.1 原理

顾名思义，决策树就是用一棵树来表示我们的整个决策过程。这棵树可以是二叉树（比如 CART 只能是二叉树），也可以是多叉树（比如 ID3、C4.5 可以是多叉树或二叉树）。

根节点包含整个样本集，每个叶节点都对应一个决策结果（注意，不同的叶节点可能对应同一个决策结果），每一个内部节点都对应一次决策过程或者说是一次属性测试。从根节点到每个叶节点的路径对应一个判定测试序列。

举个例子：



那么我们该如何选择特征对训练集进行划分呢？连续型特征（比如温度）划分的阈值又是如何确定的？

决策树的生成就是不断的选择最优的特征对训练集进行划分，是一个递归的过程。递归返回的条件有三种：

（1）当前节点包含的样本属于同一类别，无需划分；

（2）当前属性集为空，或所有样本在属性集上取值相同，无法划分；

（3）当前节点包含样本集合为空，无法划分。

1.2 ID3、C4.5、CART

这三个是非常著名的决策树算法。简单粗暴来说，ID3 使用信息增益作为选择特征的准则；C4.5 使用信息增益比作为选择特征的准则；CART 使用 Gini 指数作为选择特征的准则。

ID3

熵表示的是数据中包含的信息量大小。熵越小，数据的纯度越高，也就是说数据越趋于一致，这是我们希望的划分之后每个子节点的样子。

信息增益 = 划分前熵 - 划分后熵。信息增益越大，则意味着使用属性 a 来进行划分所获得的 “纯度提升” 越大 **。也就是说，用属性 a 来划分训练集，得到的结果中纯度比较高。

ID3 仅仅适用于二分类问题。ID3 仅仅能够处理离散属性。

C4.5

C4.5 克服了 ID3 仅仅能够处理离散属性的问题，以及信息增益偏向选择取值较多特征的问题，使用信息增益比来选择特征。信息增益比 = 信息增益 / 划分前熵 选择信息增益比最大的作为最优特征。

C4.5 处理连续特征是先将特征取值排序，以连续两个值中间值作为划分标准。尝试每一种划分，并计算修正后的信息增益，选择信息增益最大的分裂点作为该属性的分裂点。

CART

CART 与 ID3，C4.5 不同之处在于 CART 生成的树必须是二叉树。也就是说，无论是回归还是分类问题，无论特征是离散的还是连续的，无论属性取值有多个还是两个，内部节点只能根据属性值进行二分。

CART 的全称是分类与回归树。从这个名字中就应该知道，CART 既可以用于分类问题，也可以用于回归问题。

回归树中，使用平方误差最小化准则来选择特征并进行划分。每一个叶子节点给出的预测值，是划分到该叶子节点的所有样本目标值的均值，这样只是在给定划分的情况下最小化了平方误差。

要确定最优化分，还需要遍历所有属性，以及其所有的取值来分别尝试划分并计算在此种划分情况下的最小平方误差，选取最小的作为此次划分的依据。由于回归树生成使用平方误差最小化准则，所以又叫做最小二乘回归树。

分类树种，使用 Gini 指数最小化准则来选择特征并进行划分；

Gini 指数表示集合的不确定性，或者是不纯度。基尼指数越大，集合不确定性越高，不纯度也越大。这一点和熵类似。另一种理解基尼指数的思路是，基尼指数是为了最小化误分类的概率。

1.3 信息增益 vs 信息增益比

之所以引入了信息增益比，是由于信息增益的一个缺点。那就是：信息增益总是偏向于选择取值较多的属性。信息增益比在此基础上增加了一个罚项，解决了这个问题。

1.4 Gini 指数 vs 熵

既然这两个都可以表示数据的不确定性，不纯度。那么这两个有什么区别那？

1.5 剪枝

决策树算法很容易过拟合（overfitting），剪枝算法就是用来防止决策树过拟合，提高泛华性能的方法。

剪枝分为预剪枝与后剪枝。

预剪枝是指在决策树的生成过程中，对每个节点在划分前先进行评估，若当前的划分不能带来泛化性能的提升，则停止划分，并将当前节点标记为叶节点。

后剪枝是指先从训练集生成一颗完整的决策树，然后自底向上对非叶节点进行考察，若将该节点对应的子树替换为叶节点，能带来泛化性能的提升，则将该子树替换为叶节点。

那么怎么来判断是否带来泛化性能的提升那？最简单的就是留出法，即预留一部分数据作为验证集来进行性能评估。

1.6 总结

决策树算法主要包括三个部分：特征选择、树的生成、树的剪枝。常用算法有 ID3、C4.5、CART。

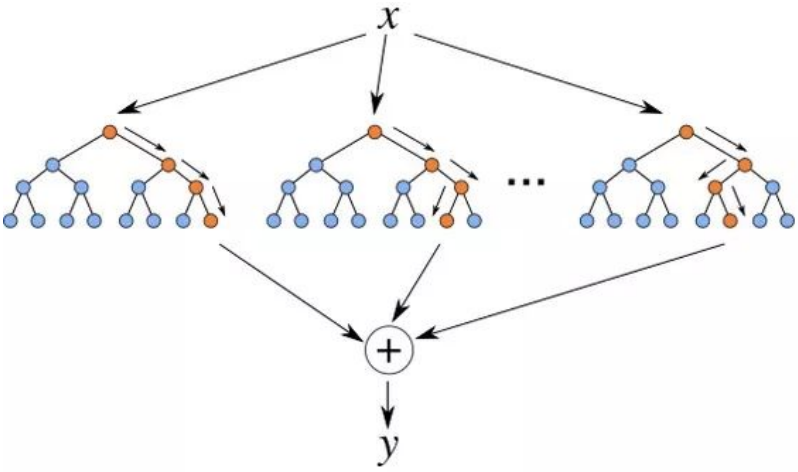
特征选择。特征选择的目的是选取能够对训练集分类的特征。特征选择的关键是准则：信息增益、信息增益比、Gini 指数；

决策树的生成。通常是利用信息增益最大、信息增益比最大、Gini 指数最小作为特征选择的准则。从根节点开始，递归的生成决策树。相当于是不断选取局部最优特征，或将训练集分割为基本能够正确分类的子集；

决策树的剪枝。决策树的剪枝是为了防止树的过拟合，增强其泛化能力。包括预剪枝和后剪枝。

二、随机森林（Random Forest）

要说随机森林就要先说 Bagging，要说 Bagging 就要先说集成学习。



2.1 集成学习方法

集成学习（ensemble learning）通过构建并组合多个学习器来完成学习任务。集成学习通过将多个学习器进行结合，常获得比单一学习器显著优越的泛化性能。

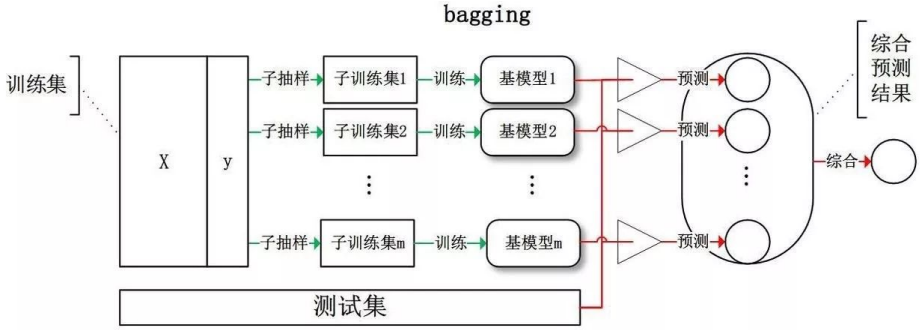
根据个体学习器是否是同类型的学习器（由同一个算法生成，比如 C4.5,BP 等），分为同质和异质。同质的个体学习器又叫做基学习器，而异质的个体学习器则直接成为个体学习器。

原则：要获得比单一学习器更好的性能，个体学习器应该好而不同。即个体学习器应该具有一定的准确性，不能差于弱学习器，并且具有多样性，即学习器之间有差异。

根据个体学习器的生成方式，目前集成学习分为两大类：

个体学习器之间存在强依赖关系、必须串行生成的序列化方法。代表是 Boosting；

2.2 Bagging



前面提到，想要集成算法获得性能的提升，个体学习器应该具有独立性。虽然 “独立” 在现实生活中往往无法做到，但是可以设法让基学习器尽可能的有较大的差异。

Bagging 给出的做法就是对训练集进行采样，产生出若干个不同的子集，再从每个训练子集中训练一个基学习器。由于训练数据不同，我们的基学习器可望具有较大的差异。

Bagging 是并行式集成学习方法的代表，采样方法是自助采样法，用的是有放回的采样。初始训练集中大约有 63.2% 的数据出现在采样集中。

Bagging 在预测输出进行结合时，对于分类问题，采用简单投票法；对于回归问题，采用简单平均法。

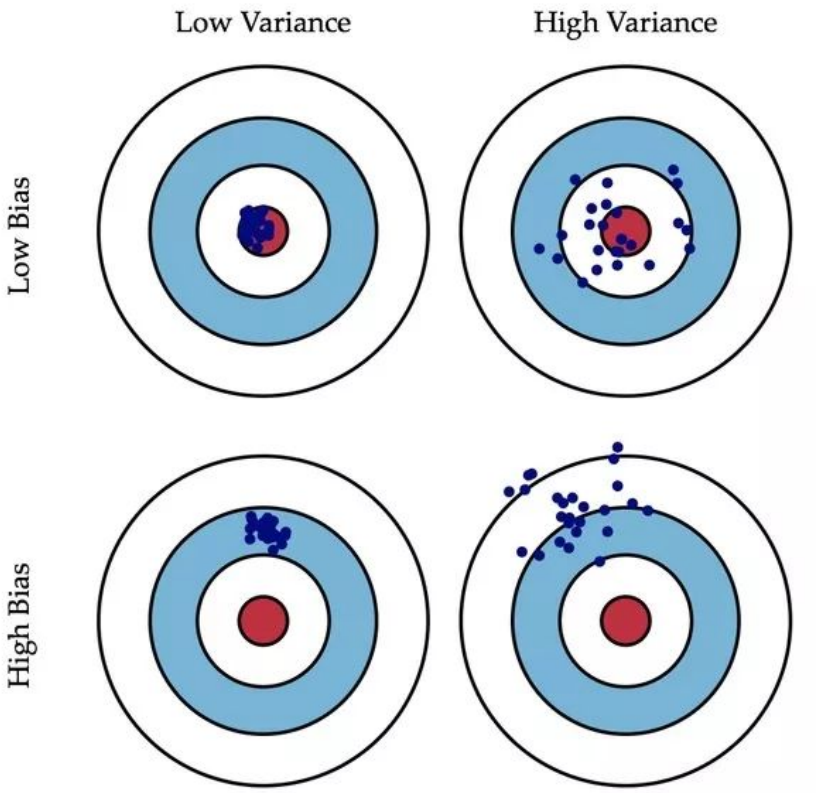
Bagging 优点：

高效。Bagging 集成与直接训练基学习器的复杂度同阶；

Bagging 能不经修改的适用于多分类、回归任务；

包外估计。使用剩下的样本作为验证集进行包外估计（out-of-bag estimate）。

Bagging 主要关注降低方差。（low variance）



2.3 随机森林（Random Forest）

2.3.1 原理

原来决策树从所有属性中，选择最优属性。Random Forest 的每一颗决策树中的每一个节点，先从该节点的属性集中随机选择 K 个属性的子集，然后从这个属性子集中选择最优属性进行划分。

K 控制了随机性的引入程度，是一个重要的超参数。

预测 ：

分类：简单投票法；

回归：简单平均法。

2.3.2 优缺点

优点：

由于每次不再考虑全部的属性，而是一个属性子集，所以相比于 Bagging 计算开销更小，训练效率更高；

由于增加了属性的扰动，随机森林中基学习器的性能降低，使得在随机森林在起始时候性能较差，但是随着基学习器的增多，随机森林通常会收敛于更低的泛化误差，相比于 Bagging；

两个随机性的引入，使得随机森林不容易陷入过拟合，具有很好的抗噪声能力；

对数据的适应能力强，可以处理离散和连续的，无需要规范化；

可以得到变量的重要性， 基于 oob 误分类率和基于 Gini 系数的变化。

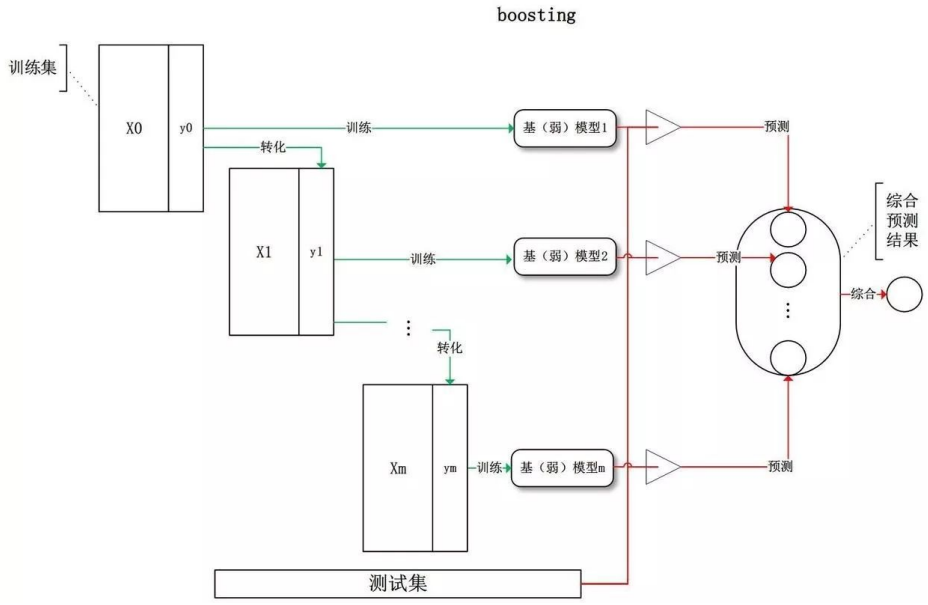
缺点：

在噪声较大的时候容易过拟合。

三、AdaBoost

AdaBoost 是 Boosting 的代表，前面我们提到 Boosting 是集成学习中非常重要的一类串行化学习方法。

3.1 Boosting



Boosting 是指个体学习器之间存在强依赖关系，必须串行序列化生成的集成学习方法。他的思想来源是三个臭皮匠顶个诸葛亮。Boosting 意为提升，意思是希望将每个弱学习器提升为强学习器。

工作机制如下：

先从初始训练集中学习一个基学习器；

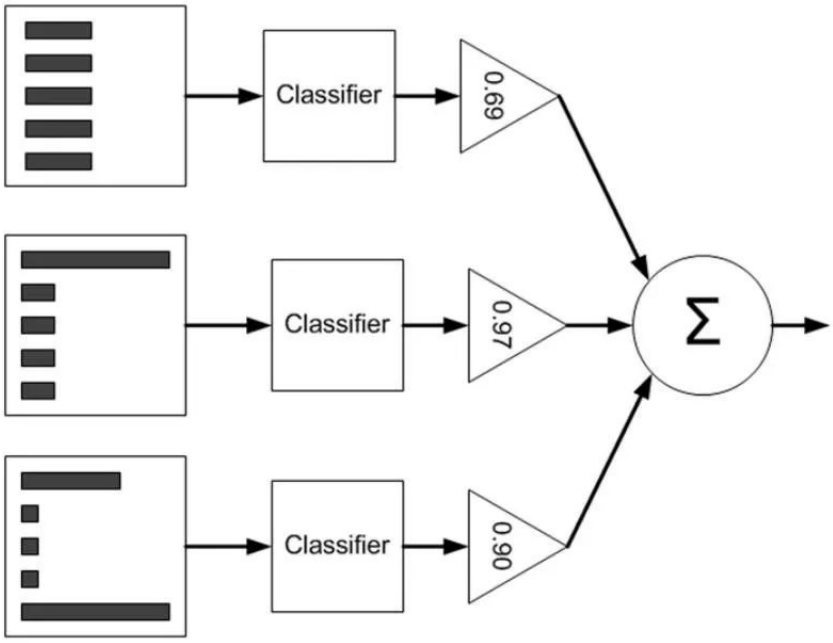
基于调整后的样本分布来训练下一个基学习器；

如此反复，直到基学习器数目达到 T，最终将这 T 个基学习器进行加权结合。

对训练样本分布调整，主要是通过增加误分类样本的权重，降低正确分类样本的权重。

Boosting 关注的主要是降低偏差。（ low bias ）

3.2 AdaBoost 原理



面临两个问题：

（ 1 ）在每一轮，如何改变训练数据的概率分布或者权值分布。（ 也可以重采样，但是 AdaBoost 没这么做 ）；

（ 2 ）如何将弱分类器组合成强分类器。

AdaBoost 的做法：

（ 1 ）提高那些被前一轮弱分类器错误分类样本的权值，降低那些被正确分类的样本的权值。这样一来，那些没有得到正确分类的数据，由于其权值的加大而受到后一轮弱分类器的关注；

（ 2 ）采用加权多数表决。具体的，加大分类错误率低的分类器的权值，使其在表决中起较大作用，减少分类误差率大的弱分类器的权值，使其在表决中起较小作用。

弱分类器被线性组合成为一个强分类器。

训练目标：

最小化指数损失函数。

三部分组成：

（ 1 ）分类器权重更新公式；

（ 2 ）样本分布（ 也就是样本权重 ）更新公式；

（ 3 ）加性模型。 最小化指数损失函数。

3.3 AdaBoost 优缺点

优点：

利用基本分类器的加权线性组合构建最终分类器，是 AdaBoost 的另一个特点；

AdaBoost 被实践证明是一种很好的防止过拟合的方法，但至今为什么至今没从理论上证明。

缺点：

AdaBoost 只适用于二分类问题。

四、GBDT

GBDT（Gradient Boosting Decision Tree）又叫 MART（Multiple Additive Regression Tree）。是一种迭代的决策树算法。

本文从以下几个方面进行阐述：

Regression Decision Tree(DT)；

Gradient Boosting(GB)；

Shrinkage(算法的一个重要演进分支，目前大部分源码都是基于该版本实现)；

GBDT 适用范围；

与随机森林的对比。

4.1 DT：回归树 Regression Decision Tree

GDBT 中的树全部都是回归树，核心就是累加所有树的结果作为最终结果。只有回归树的结果累加起来才是有意义的，分类的结果加是没有意义的。

GDBT 调整之后可以用于分类问题，但是内部还是回归树。

这部分和决策树中的是一样的，无非就是特征选择。回归树用的是最小化均方误差，分类树是用的是最小化基尼指数（CART）

4.2 GB：梯度迭代 Gradient Boosting

首先 Boosting 是一种集成方法。通过对弱分类器的组合得到强分类器，他是串行的，几个弱分类器之间是依次训练的。GBDT 的核心就在于，每一颗树学习的是之前所有树结论和的残差。

Gradient 体现在：无论前面一颗树的 cost function 是什么，是均方差还是均差，只要它以误差作为衡量标准，那么残差向量都是它的全局最优方向，这就是 Gradient。

4.3 Shrinkage

Shrinkage（缩减）是 GBDT 算法的一个重要演进分支，目前大部分的源码都是基于这个版本的。

核心思想在于：Shrinkage 认为每次走一小步来逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易防止过拟合。

也就是说，它不信任每次学习到的残差，它认为每棵树只学习到了真理的一小部分，累加的时候只累加一小部分，通过多学习几棵树来弥补不足。

具体的做法就是：仍然以残差作为学习目标，但是对于残差学习出来的结果，只累加一小部分（step* 残差）逐步逼近目标，step 一般都比较小 0.01-0.001, 导致各个树的残差是渐变而不是陡变的。

本质上，Shrinkage 为每一颗树设置了一个 weight，累加时要乘以这个 weight，但和 Gradient 没有关系。

这个 weight 就是 step。跟 AdaBoost 一样，Shrinkage 能减少过拟合也是经验证明的，目前还没有理论证明。

4.4 GBDT 适用范围

GBDT 也可用于二分类问题（设定阈值，大于为正，否则为负）和多分类问题。

4.5 GBDT 和随机森林

GBDT 和随机森林的相同点：

都是由多棵树组成；

最终的结果都由多棵树共同决定。

GBDT 和随机森林的不同点：

组成随机森林的可以是分类树、回归树；组成 GBDT 只能是回归树；

组成随机森林的树可以并行生成（Bagging）；GBDT 只能串行生成（Boosting）；

对于最终的输出结果而言，随机森林使用多数投票或者简单平均；而 GBDT 则是将所有结果累加起来，或者加权累加起来；

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权值一样，GBDT 是基于权值的弱分类器的集成；

随机森林通过减小模型的方差提高性能，GBDT 通过减少模型偏差提高性能。

TIP

1. GBDT 相比于决策树有什么优点

泛化性能更好！GBDT 的最大好处在于，每一步的残差计算其实变相的增大了分错样本的权重，而已经分对的样本则都趋向于 0。这样后面就更加专注于那些分错的样本。

2. Gradient 体现在哪里？

可以理解为残差是全局最优的绝对方向，类似于求梯度。

3. re-sample

GBDT 也可以在使用残差的同时引入 Bootstrap re-sampling，GBDT 多数实现版本中引入了这个选项，但是是否一定使用有不同的看法。

原因在于 re-sample 导致的随机性，使得模型不可复现，对于评估提出一定的挑战，比如很难确定性能的提升是由于 feature 的原因还是 sample 的随机因素。

五、Logistic 回归

LR 原理；

参数估计；

LR 的正则化；

为什么 LR 能比线性回归好？

LR 与 MaxEnt 的关系。

5.1 LR 模型原理

首先必须给出 Logistic 分布：

u 是位置参数，r 是形状参数。对称点是 (u,1/2)，r 越小，函数在 u 附近越陡峭。

$$1 + e^{-(x-\mu)/\gamma}$$

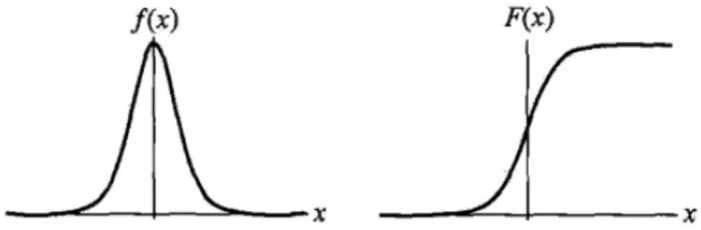


图 6.1 逻辑斯谛分布的密度函数与分布函数

然后，二分类 LR 模型，是参数化的 logistic 分布，使用条件概率来表示：

$$P(Y=1|x) == \frac{e^{w\cdot x}}{1+e^{w\cdot x}}$$

$$P(Y=0|x) == \frac{1}{1+e^{w\cdot x}}$$

然后，一个事件的几率（odds）：指该事件发生与不发生的概率比值：

$$odds = \frac{p}{1-p}$$

对数几率：

$$logit(p) = log\ \frac{p}{1-p}$$

那么对于逻辑回归而言，Y=1 的对数几率就是：

$$log\ \frac{P(Y=1|x)}{1-P(Y=1|x)} = w\cdot x$$

最终，输出 Y=1 的对数几率是输入 x 的线性函数表示的模型，这就是逻辑回归模型。

5.2 参数估计

在统计学中，常常使用极大似然估计法来估计参数。即找到一组参数，使得在这组参数下，我们数据的似然度（概率）最大。

似然函数：

$$L(w) = \prod [\pi(x_i)]^{y_i}[1-\pi(x_i)]^{1-y_i}$$

$$= \sum[y_i ln \frac{\pi(x_i)}{1-\pi(x_i)} + ln(1-\pi(x_i))]$$

$$= \sum[y_i(w \cdot x_i) - ln(1+e^{w \cdot x_i})]$$

对应的损失函数：

$$J(w) = -\frac{1}{N} lnL(w)$$

5.3 最优化方法

逻辑回归模型的参数估计中，最后就是对 J(W) 求最小值。这种不带约束条件的最优化问题，常用梯度下降或牛顿法来解决。

使用梯度下降法求解逻辑回归参数估计

求 J(W) 梯度：g(w):

$$J(w_k) = -\frac{1}{N} lnL(w_k) \Rightarrow -lnL(w_k)$$

$$= \sum[y_i(w_k \cdot x_i) - ln(1+e^{w_k \cdot x_i})]$$

$$g(w_k) = \sum[x_i \cdot y_i - \frac{x_i \cdot e^{w_k \cdot x_i}}{1+e^{w_k \cdot x_i}}]$$

$$= \sum[x_i \cdot y_i - \pi(x_i)]$$

更新 Wk：

$$ W_{k 1} = W_k - \lambda*g(W_k) $$

5.4 正则化

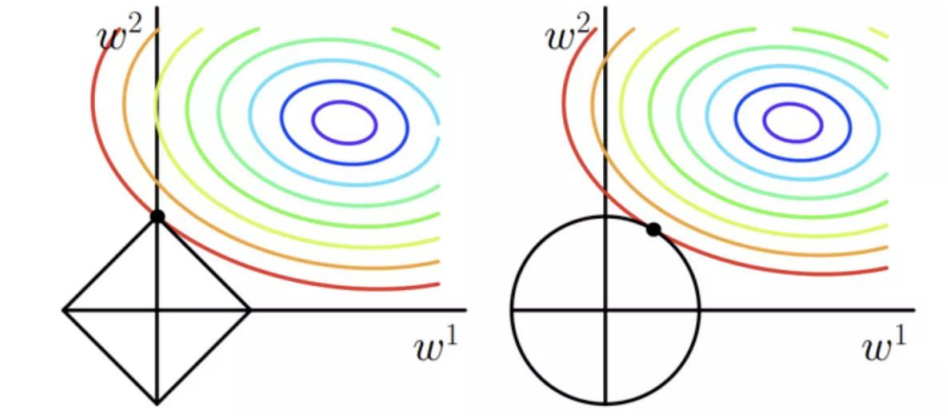
正则化为了解决过拟合问题。分为 L1 和 L2 正则化。主要通过修正损失函数，加入模型复杂性评估；

正则化是符合奥卡姆剃刀原理：在所有可能的模型中，能够很好的解释已知数据并且十分简单的才是最好的模型。

$$J(w) => J(w) + \lambda \|w\|_p$$

p 表示范数，p=1 和 p=2 分别应用 L1 和 L2 正则。

L2 正则化。使得每个元素都尽可能的小，但是都不为零。在回归里面，有人把他的回归叫做岭回归（Ridge Regression），也有人叫他 “权值衰减”（weight decay）。



一句话总结就是：L1 会趋向于产生少量的特征，而其他的特征都是 0，而 L2 会选择更多的特征，这些特征都会接近于 0.

5.5 逻辑回归 vs 线性回归

首先，逻辑回归比线性回归要好。

两者都属于广义线性模型。

线性回归优化目标函数用的最小二乘法，而逻辑回归用的是最大似然估计。逻辑回归只是在线性回归的基础上，将加权和通过 sigmoid 函数，映射到 0-1 范围内空间。

线性回归在整个实数范围内进行预测，敏感度一致，而分类范围，需要在 [0,1]。而逻辑回归就是一种减小预测范围，将预测值限定为 [0,1] 间的一种回归模型。

逻辑曲线在 z=0 时，十分敏感，在 z>>0 或 z<<0 处，都不敏感，将预测值限定为 (0,1)。逻辑回归的鲁棒性比线性回归要好。

5.6 逻辑回归模型 vs 最大熵模型 MaxEnt

简单粗暴的说：逻辑回归跟最大熵模型没有本质区别。逻辑回归是最大熵对应为二类时的特殊情况，也就是说，当逻辑回归扩展为多类别的时候，就是最大熵模型。

最大熵原理：学习概率模型的时候，在所有可能的概率模型（分布）中，熵最大的模型是最好的模型。

六、SVM 支持向量机

虽然咱们的目标是尽可能的不涉及到公式，但是提到 SVM 就没有办法不涉及到公式推导，因为面试中只要问到 SVM，最基本也是最难的问题就是：SVM 的对偶问题数学公式推导。

所以想学好机器学习，还是要塌下心来，不仅仅要把原理的核心思想掌握了，公式推导也要好好学习才行。

6.1 SVM 原理

简单粗暴的说：SVM 的思路就是找到一个超平面将数据集进行正确的分类。对于在现有维度不可分的数据，利用核函数映射到高纬空间使其线性可分。

支持向量机 SVM 是一种二分类模型。它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机。SVM 的学习策略是间隔最大化，可形式化为求解凸二次规划问题。

SVM 分为：

线性可分支持向量机。当训练数据线性可分时，通过硬间隔最大化，学习到的一个线性分类器；

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

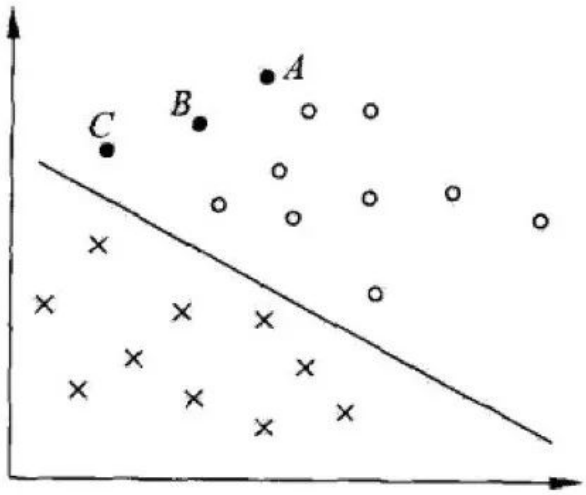


图 7.1 二类分类问题

上图中，X 表示负例，O 表示正例。此时的训练数据可分，线性可分支持向量机对应着将两类数据正确划分并且间隔最大的直线。

6.1.1 支持向量与间隔

支持向量：在线性可分的情况下，训练数据样本集中的样本点中与分离超平面距离最近的样本点的实例称为支持向量（support vector）。

函数间隔定义如下：

定义 7.2（函数间隔） 对于给定的训练数据集 T 和超平面 (w,b)，定义超平面 (w,b) 关于样本点 (x_i,y_i) 的函数间隔为

$$\hat{\gamma}_i = y_i(w \cdot x_i + b) \tag{7.3}$$

定义超平面 (w,b) 关于训练数据集 T 的函数间隔为超平面 (w,b) 关于 T 中所有样本点 (x_i,y_i) 的函数间隔之最小值，即

$$\hat{\gamma} = \min_{i=1,\cdots,N} \hat{\gamma}_i \tag{7.4}$$

yi 表示目标值，取值为 1、-1. 函数间隔虽然可以表示分类预测的准确性以及确信度。但是有个不好的性质：只要成倍的改变 W 和 B，虽然此时的超平面并没有改变，但是函数间隔会变大。

所以我们想到了对超平面的法向量 W 施加一些约束，比如规范化，使得间隔确定，这就引出了几何间隔：

定义 7.3（几何间隔） 对于给定的训练数据集 T 和超平面 (w,b)，定义超平面 (w,b) 关于样本点 (x_i,y_i) 的几何间隔为

$$\gamma_i = y_i\left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right) \tag{7.5}$$

定义超平面 (w,b) 关于训练数据集 T 的几何间隔为超平面 (w,b) 关于 T 中所有样本点 (x_i,y_i) 的几何间隔之最小值，即

$$\gamma = \min_{i=1,\cdots,N} \gamma_i \tag{7.6}$$

支持向量学习的基本思想就是求解能够正确划分训练数据集并且几何间隔最大的分类超平面。

6.1.2 对偶问题

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2$$

$$\text{s.t.} \quad y_i(w \cdot x_i + b) - 1 \geqslant 0, \quad i = 1,2,\cdots,N$$

将它作为原始最优化问题，应用拉格朗日对偶性，通过求解对偶问题得到原始问题的最优解，这就是线性可分支持向量机的对偶算法。

本来的算法也可以求解 SVM，但是之所以要用对偶问题来求解，优点是：

一是对偶问题往往更容易求解；

二是自然引入核函数，进而推广到非线性分类问题。

说点题外话，这也是面试中会被问到的一个问题：原始问题既然可以求解，为什么还要转换为对偶问题。

答案就是上述这两点。由于篇幅的愿意，此处就不在展开数学公式的推导了，大家可以参考《机器学习》与《统计学习方法》。

6.1.3 核函数

对于在原始空间中不可分的问题，在高维空间中是线性可分的。

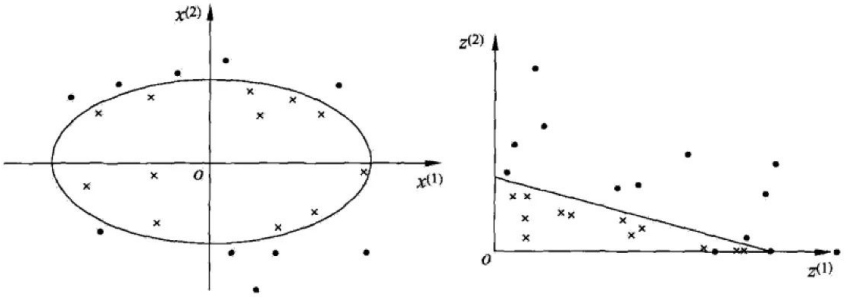


图 7.7 非线性分类问题与核技巧示例

对于线性不可分的问题，使用核函数可以从原始空间映射到高纬空间，使得问题变得线性可分。核函数还可以使得在高维空间计算的内积在低维空间中通过一个函数来完成。

常用的核函数有：高斯核、线性核、多项式核。

6.1.4 软间隔

线性可分问题的支持向量机学习方法，对现行不可分训练数据是不适用的。所以讲间隔函数修改为软间隔，对于函数间隔，在其上加上一个松弛变量，使其满足大于等于 1。约束条件变为：

$$y_i(w \cdot x_i + b) \geqslant 1 - \xi_i$$

6.2 优缺点

缺点：

时空开销比较大，训练时间长；

核函数的选取比较难，主要靠经验。

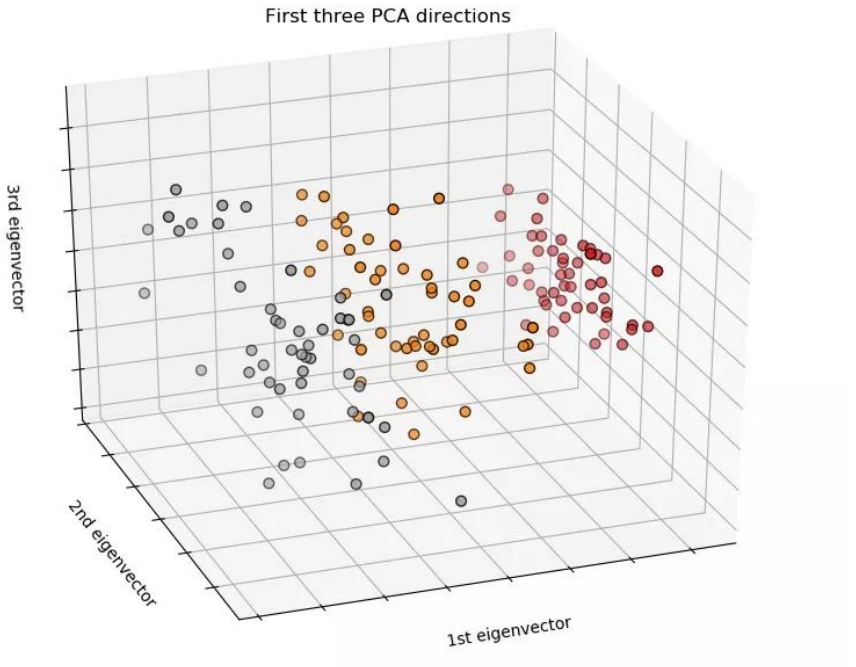
优点：

在小训练集上往往得到比较好的结果；

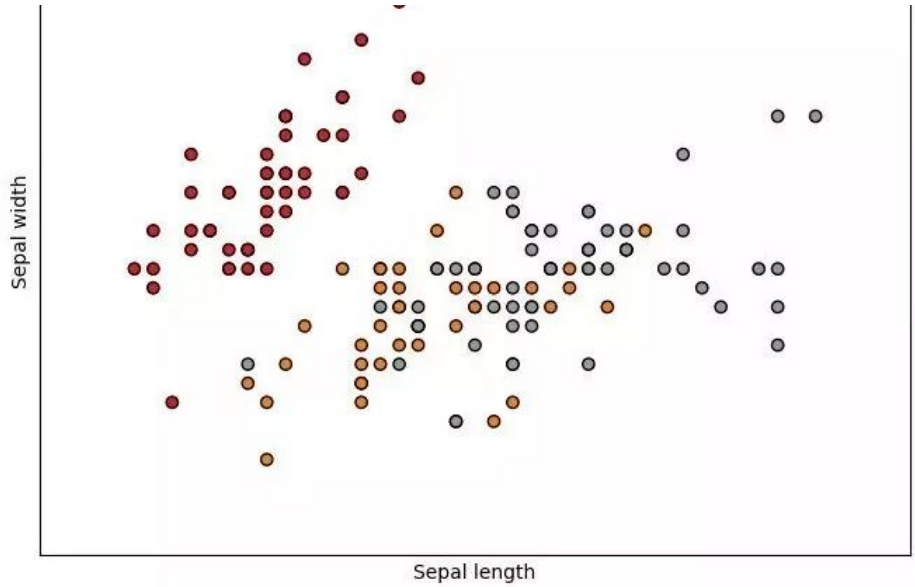
使用核函数避开了高纬空间的复杂性；

使用 sklearn 用决策树来进行莺尾花数据集的划分问题。代码上没有固定随机种子，所以每次运行的结果会稍有不同。

数据集三维可视化：



在 Sepal length 和 Sepal width 二维上的可视化：



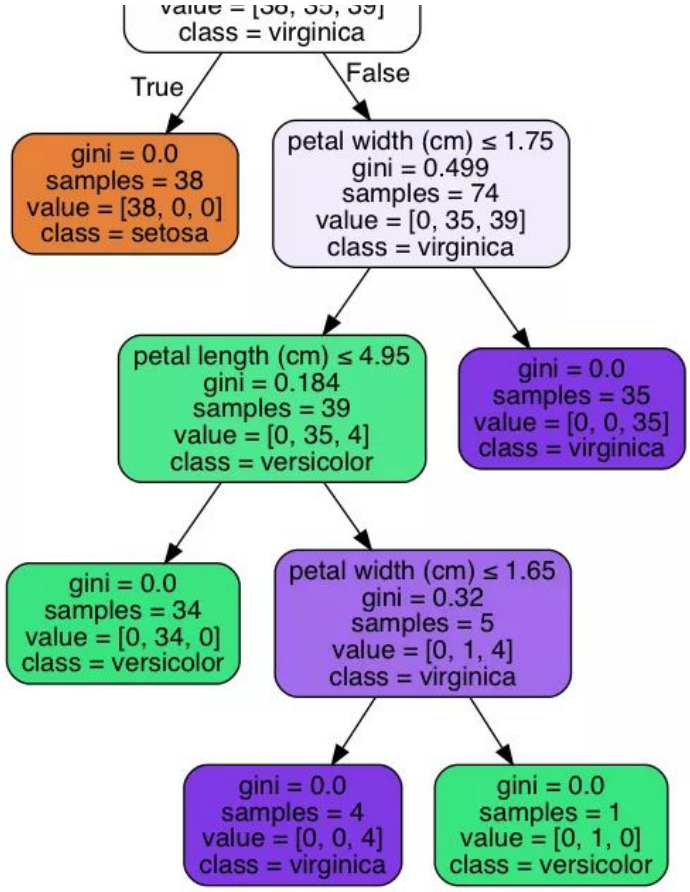

```
from sklearn import tree
from sklearn.svm import SVC
from subprocess import check_call
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import GradientBoostingClassifier

# load data
iris = load_iris()

# split valid dataset
X_train, X_val, y_train, y_val = train_test_split(iris.data, iris.target,
test_size=0.25)

# decision tree
dt = tree.DecisionTreeClassifier(criterion='gini')
dt2 = tree.DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train, y_train)
dt2.fit(X_train,y_train)

# accuracy
print "Decision Tree (Gini) Accuracy: ", accuracy_score(y_val, dt.predict(X_val))
print "Decision Tree (Entropy) Accuracy: ", accuracy_score(y_val,
dt2.predict(X_val))
```

```
                    filled=True,
                    rounded=True,
                    special_characters=True)
check_call(['dot','-Tpng','tree.dot','-o','tree.png'])

# Random forest
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
print "Random Forest Accuracy: ", accuracy_score(y_val, rfc.predict(X_val))

# AdaBoost
adaboost = AdaBoostClassifier(n_estimators=300)
adaboost.fit(X_train, y_train)
print "AdaBoost Accuracy: ", accuracy_score(y_val, adaboost.predict(X_val))

# GBDT
gbdt = GradientBoostingClassifier()
gbdt.fit(X_train, y_train)
print "GBDT Accuracy: ",accuracy_score(y_val, gbdt.predict(X_val))

# logistic regression
# for multiclass use One-vs-Rest(ovr) model
lr = LogisticRegression(multi_class='ovr')
lr.fit(X_train, y_train)
print "Logistic Regression Accuracy: ", accuracy_score(y_val, lr.predict(X_val))

# SVM
svm = SVC()
svm.fit(X_train, y_train)
print "SVM Accuracy: ", accuracy_score(y_val, svm.predict(X_val))
```

运行结果：

```
Decision Tree (Gini) Accuracy:  0.894736842105
Decision Tree (Entropy) Accuracy:  0.894736842105
Random Forest Accuracy:  0.894736842105
AdaBoost Accuracy:  0.894736842105
GBDT Accuracy:  0.894736842105
Logistic Regression Accuracy:  0.894736842105
SVM Accuracy:  0.947368421053
```

Decision Tree 可视化，也就是生成的决策树：



参考书籍

《机器学习》周志华

《统计学习方法》

（完）