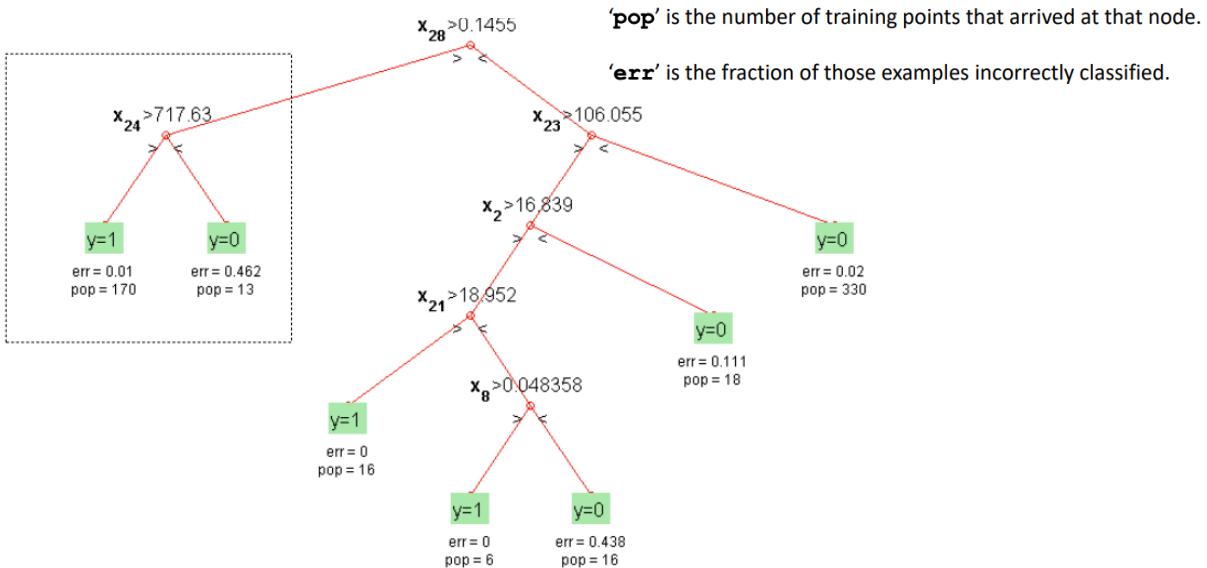


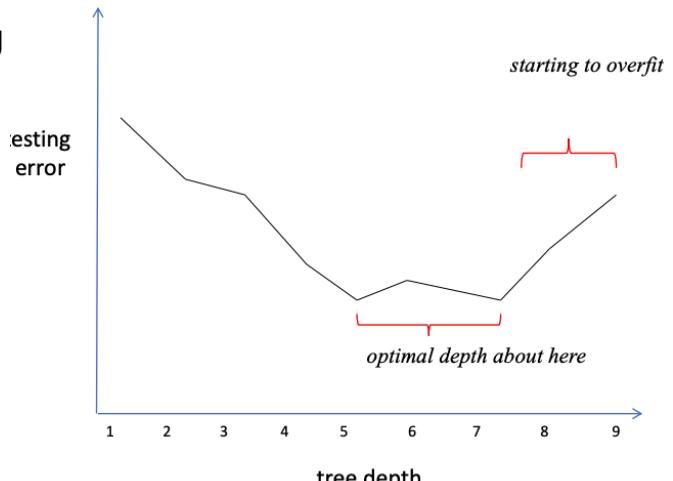
Decision Trees (non-linear model)



Depth越大 --> more complex, overfitting

ID3树中止的条件:

- 1.) it reaches a minimum number of examples - i.e. there are very few examples left in the branch
- 2.) the most recent split has led to perfect classification. In other words, when the labels are equal



Entropy 信息熵

$$H(T) = - \sum_i p(x_i) \log_2 p(x_i)$$

这里的P是根据label来算的

When P = 0.5

$$\begin{aligned} H(X) &= - (p(\text{head}) \log p(\text{head}) + p(\text{tail}) \log p(\text{tail})) \\ &= - (0.5 \log 0.5 + 0.5 \log 0.5) \\ &= - ((-0.5) + (-0.5)) = 1 \end{aligned}$$

$H(T)$ is the entropy before we split

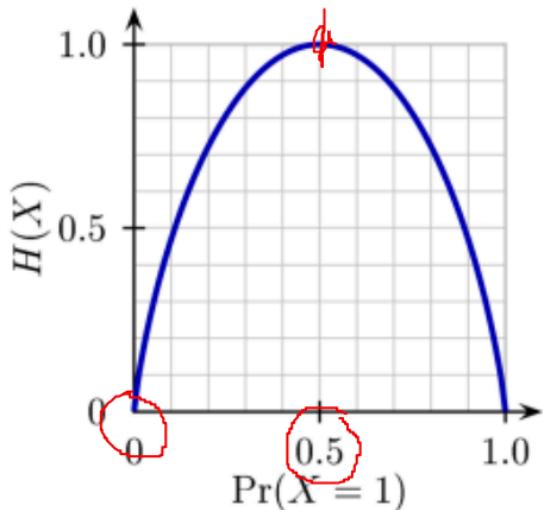
Repeat all features and select feature with minimum entropy

条件熵:

$H(T | W = L)$ is the entropy of the data on left branch

$H(T | W = R)$ is the entropy of the data on right branch

$H(T | W)$ is the weighted average of the two



信息熵越小说明用来筛选越好，越在顶部

若按某特征划分后，熵值减少的最大，则次特征为最优分类特征

分类之后熵值变小

Information Gain 信息增益

每次选择最大信息增益的属性作为当前的划分属性

$$IG(T | W) = H(T) - H(T | W)$$

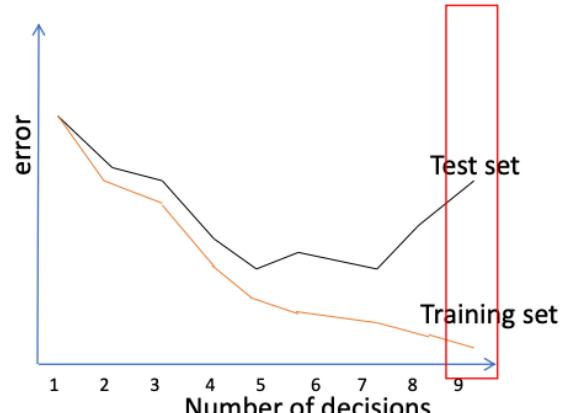
信息增益 = entroy(前) - entroy(后)

衡量一个属性(W)区分样本(T)的能力。当新增一个属性(W)时，信息熵H(T)的变化大小即为信息增益。IG(T|W)越大表示W越重要。影响力

Pruning 剪枝

目的是为了避免决策树模型的过拟合。因为决策树算法在学习的过程中为了尽可能的正确的分类训练样本，不停地对结点进行划分，因此这会导致整棵树的分支过多，也就导致了过拟合

Result: complex tree, with high accuracy on the training set, but low accuracy on a validation set



预剪枝 (pre-pruning)：预剪枝就是在构造决策树的过程中，先对每个结点在划分前进行估计，若果当前结点的划分不能带来决策树模型泛化性能的提升，则不对当前结点进行划分并且将当前结点标记为叶结点。

判断依据：计算Information Gain，如果为0则剪，泛化性能是否有提升

Example:

Pros and Cons

脐部:

D^1 {脐部 = 凹陷}: (1,2,3,14), D^2 {脐部 = 稍凹}: (6,7,15,17), D^3 {脐部 = 平坦}: (10,16) 则：

$$Ent(D^1) = - \sum_{k=1}^2 p_k \log_2 p_k = -\left(\frac{3}{4} \log_2 \frac{3}{4} + \frac{1}{4} \log_2 \frac{1}{4}\right) = 0.811$$

$$Ent(D^2) = - \sum_{k=1}^2 p_k \log_2 p_k = -\left(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}\right) = 1$$

$$Ent(D^3) = - \sum_{k=1}^2 p_k \log_2 p_k = -(0 + 1 \log_2 1) = 0$$

有: $Gain(D, 脐部) = 1 - \left(\frac{4}{10} * 0.811 + \frac{4}{10} * 1 + \frac{2}{10} * 0\right) = 0.276$

触感:

D^1 {触感 = 硬滑}: (1,2,3,16,17), D^2 {触感 = 软粘}: (6,7,10,15) 则：

$$Ent(D^1) = - \sum_{k=1}^2 p_k \log_2 p_k = -\left(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}\right) = 1$$

$$Ent(D^2) = - \sum_{k=1}^2 p_k \log_2 p_k = -\left(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}\right) = 1$$

有: $Gain(D, 触感) = 1 - \left(\frac{6}{10} * 1 + \frac{4}{10} * 1\right) = 0$

Pros

- Does not require validation data
- Fast to train – no unnecessary branches are created

Cons

- No guarantee that we get the best answer
 - Possible for one split to have low information gain, but subsequent splits to have high information gain.

summary:

预剪枝使得决策树的很多分支都没有“展开”，这不仅降低了过拟合的风险，还显著减少了决策树的训练时间开销和测试时间开销。

but, 因为预剪枝是基于“贪心”的，所以，虽然当前划分不能提升泛化性能，但是基于该划分的后续划分却有可能导致性能提升，因此预剪枝决策树有可能带来欠拟合的风险

后剪枝 (post-pruning) : 后剪枝就是先把整颗决策树构造完毕，然后自底向上的对非叶结点进行考察，若将该结点对应的子树换为叶结点能够带来泛化性能的提升，则把孩子树替换为叶结点。

Via cross validation

Pros

- Possible to guarantee the ‘best’ answer (but still very difficult for large trees)

Cons

- Two-stage process, so slower than pre-pruning
- Typically requires validation data (which may mean that it is not possible for smaller datasets)

Summary:

后剪枝决策树通常比预剪枝决策树保留了更多的分支，一般情形下，后剪枝决策树的欠拟合风险小，泛化性能往往也要优于预剪枝决策树。但后剪枝过程是在构建完全决策树之后进行的，并且要自底向上的对树中的所有非叶结点进行逐一考察，因此其训练时间开销要比未剪枝决策树和预剪枝决策树都大得多。

ROC analysis

True Positive (TP): 正类被正确分类为正类

➤ Total positive

False Negative (FN): 正类被错误分类为负类

➤ Total negative

True Negative (TN): 负类被正确分类为负类

False Positive (FP): 负类被错误分类为正类

True&False => 预测是否正确 Positive&Negative => 预测结果

tp rate = TP/P = TP/(TP + FN) 真阳率, **recall, sensitivity**, 正确分类正样本占所有正样本比例

fp rate = FP/N = FP/(FP + TN) 假阳率, 表示错误分类负样本占总负样本比例 假的判成真

specificity = TN/(TN + FP) 真负类率 **TNR** = 1-FPR

precision = TP/(TP + FP) 表示正确分类正样本占分类为正样本的比例

accuracy = (TP + TN)/(P + N) 不管正负, 表示正确分类样本占所有样本比例

F-score = 2/(1/precision + 1/recall) 是recall和precision的调和平均

Example

	Positive	Negative	All
True	TP: 14	TN: 18	32
False	FP: 2	FN: 6	8
All	16	24	40

所以可以进行计算了

$$precision = \frac{TP}{TP + FP} = \frac{14}{14 + 2}$$

$$recall = \frac{TP}{TP + FN} = \frac{14}{14 + 6}$$

ROC曲线

① Discrete Classifier 只输出正或负类

在一份测试集上，Discrete Classifier只得到ROC曲线上一个离散点，如图中的A、B、C等点；

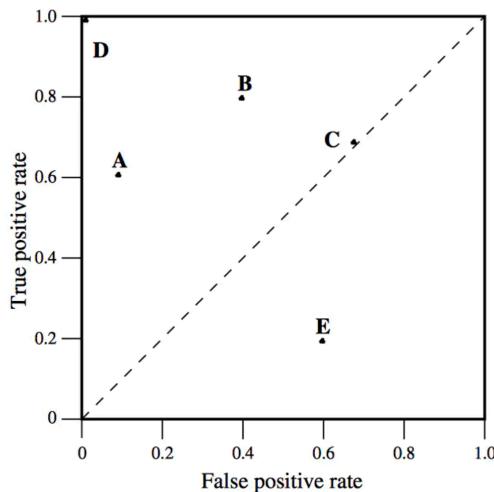
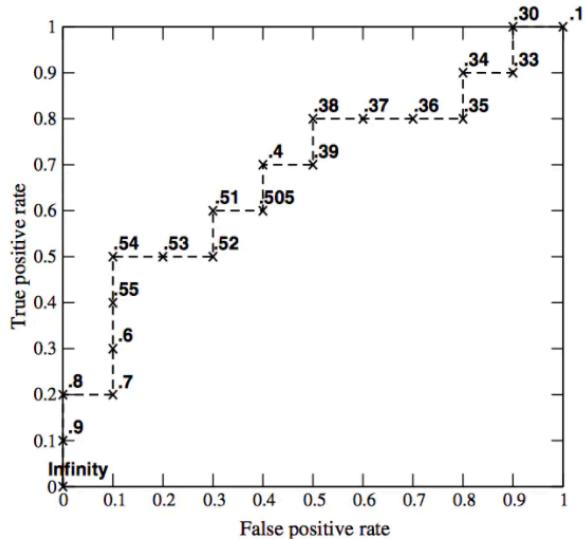


Fig. 2. A basic ROC graph showing five discrete classifiers.

② Probabilistic Classifier 输出为正类的概率未矫正得分

Probabilistic Classifier则不同，通过取不同阈值，可得到ROC曲线上一系列点，形成一个step function。

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



ROC曲线上有四个坐标点比较特殊：

- (0,0) 所有样本预测为负样本；不会有负样本误判为正样本，假阳率为0；
不会有正样本被召回，真阳率为0

- (1,1) 所有样本预测为正样本；所有负样本都误判为正样本，假阳率为1；

所有正样本都被召回，真阳率为1

- (0,1) 完美分类器；没有负样本被误判为正样本，假阳率为0；
所有正样本都被召回，真阳率为1

曲线越高越好，
说明面积越大

- (1,0) 颠倒分类器；所有负样本均误判为正样本，假阳率为1；
所有正样本未被召回，真阳率为0

大体上讲，越接近左上角(0,1)，分类器性越倾向于好。一个非正式解释就是
越靠左，分类器越conservative，证据很强时，才分类为正样本，fp rate较低

越靠右上侧，分类器越liberal，证据不太强时，就分类为正样本，fp rate较高

x1	x2	x3	y	Model output
0.65	0.03	0.99	0	0.1
0.55	0.86	0.04	0	0.7
0.24	0.71	0.44	0	0.25
0.43	0.55	0.33	0	0.8
0.07	0.89	0.78	0	0.12
0.41	0.93	0.72	0	0.44
0.83	0.46	0.75	1	1
0.22	0.83	0.06	1	0.8
0.87	0.79	0.82	1	0.62
0.24	0.93	0.50	1	0.75
0.46	0.82	0.12	1	0.9
0.57	0.60	0.33	1	0.45

{0, 0.2, 0.4, 0.6, 0.8, 1}

4.) At $p(y=1) = 0.2$:

How are the items
classified?

0.2作为boundry小于0.2的
判0，大于0.2判1

- 2 examples classified in class zero (blue dots)
- 2 TN, 4 FP, 6 TP, 0 FN
- Sens = 6/6
- Spec = 2/6

Deal with biggest impact feature

- What can we do about this problem? (covered in THIS week's recorded lectures)
 - Normalisation
 - Scale the data so that each feature has a similar range
 - Either by a zero-mean, unit variance transform (sometimes called standardisation)
 - Or by scaling using the max/min values

standardization 标准化

Preprocessing.scale() 把数据标准化

将每一列特征标准化为标准正太分布，注意，标准化是针对每一列而言的

$$\text{标准差: } \sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

$$\text{标准化: } x_{norm} = \frac{x - \bar{x}}{\sigma} \quad (\text{Standardization})$$

标准化后的数据的均值=0，方差=1

标准化方法： <https://blog.csdn.net/u010758410/article/details/78158781>

Normalization 归一化

正则化是将样本在向量空间模型上的一个转换，经常被使用在分类与聚类中。

将所有特征数据按比例缩放到0-1的这个取值区间。有时也可以是-1到1的区间

函数normalize 提供了一个快速有简单的方式在一个单向量上来实现这正则化的功能。正则化有l1,l2等

因此normalization和standardization 是针对数据而言的，消除一些数值差异带来的特種重要性偏见。经过归一化的数据，能加快训练速度，促进算法的收敛。结果不变

regularization 正则化

在cost function里面加惩罚项，增加建模的模糊性，但是能有效防止过度拟合over-fitting的问题

增加模型准确性。regularization是针对模型而言

一般形式，应该是 $\min \sum_{i=1} J(x_i, y_i) + \lambda R(f)$, R是regularization term。一般方法有

1. L1 regularization: 对整个绝对值只和进行惩罚。

2. L2 regularization: 对系数平方和进行惩罚。

后面一项 $\alpha||w||_1$ 即为L1正则化项。

$$\min_w \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha ||w||_1$$

下图是Python中Ridge回归的损失函数，式中加号后面一项 $\alpha||w||_2^2$ 即为L2正则化项。

$$\min_w ||Xw - y||_2^2 + \alpha ||w||_2^2$$

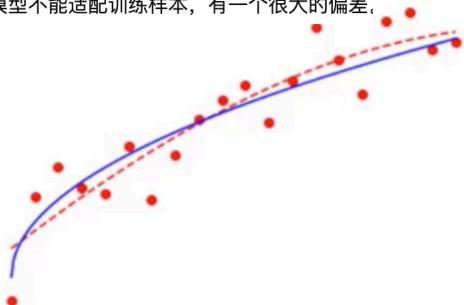
L1正则化可以产生稀疏权值矩阵，即产生一个稀疏模型，可以用于特征选择

L2正则化可以防止模型过拟合 (overfitting)；一定程度上，L1也可以防止过拟合

方差vs偏差

过拟合：模型很好的适配训练样本，但在测试集上表现很糟，有一个很大的方差。

欠拟合：模型不能适配训练样本，有一个很大的偏差。



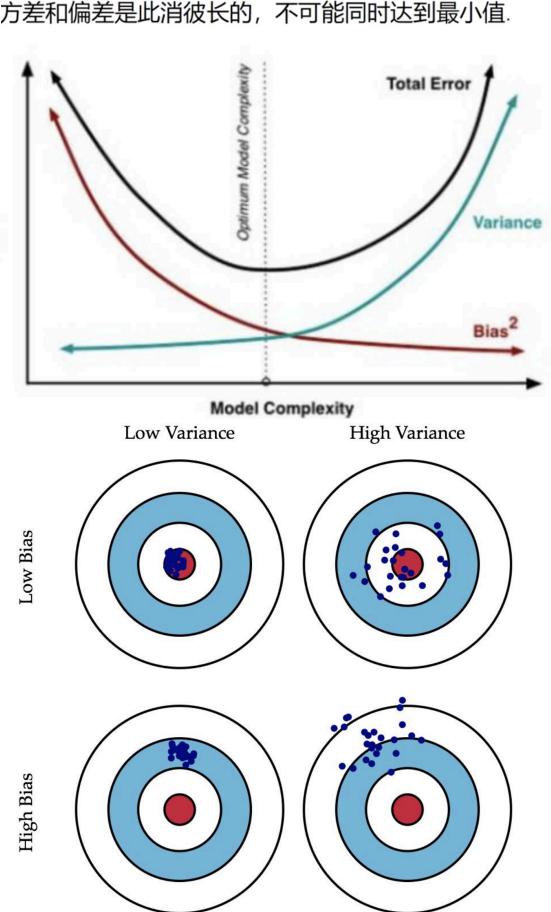
图中每个点就是集成算法中的一个基评估器产生的预测值。红色虚线代表着这些预测值的均值，而蓝色的线代表着数据本来的面貌。偏差：每一个红点到蓝线的距离，方差：即每一个红点到红色虚线的距离

其中偏差衡量模型是否预测得准确，偏差越小，模型越“准”；而方差衡量模型每次预测的结果是否接近，即是说方差越小，模型越“稳”。

	偏差大	偏差小
方差大	模型不适合这个数据 换模型	过拟合 模型很复杂 对某些数据集预测很准确 对某些数据集预测很糟糕
方差小	欠拟合 模型相对简单 预测很稳定 但对所有的数据预测都不太准确	泛化误差小，我们的目标

一个集成模型 (f) 在未知数据集 (D) 上的泛化误差 $E(f; D)$ ，由方差 (var)，偏差 ($bias$) 和噪声 (ϵ) 共同决定

$$E(f, D) = bias^2(x) + var(x) + \epsilon^2$$



针对偏差和方差的思路：

偏差：实际上也可以称为避免欠拟合。

1、寻找更好的特征 -- 具有代表性。

2、用更多的特征 -- 增大输入向量的维度，增加模型复杂度。

方差：避免过拟合。

1、增大数据集合 -- 使用更多的数据，减少数据扰动所造成的影响

2、减少数据特征 -- 减少数据维度，减少模型复杂度

3、正则化方法

4、交叉验证法