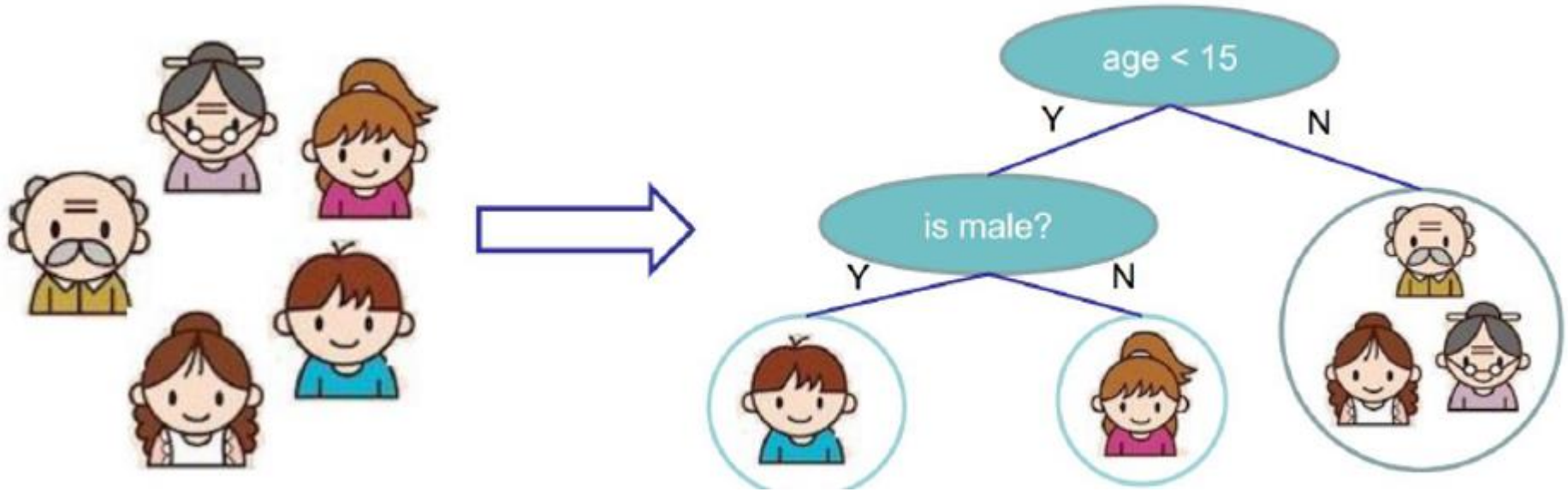
决策树

- 树模型

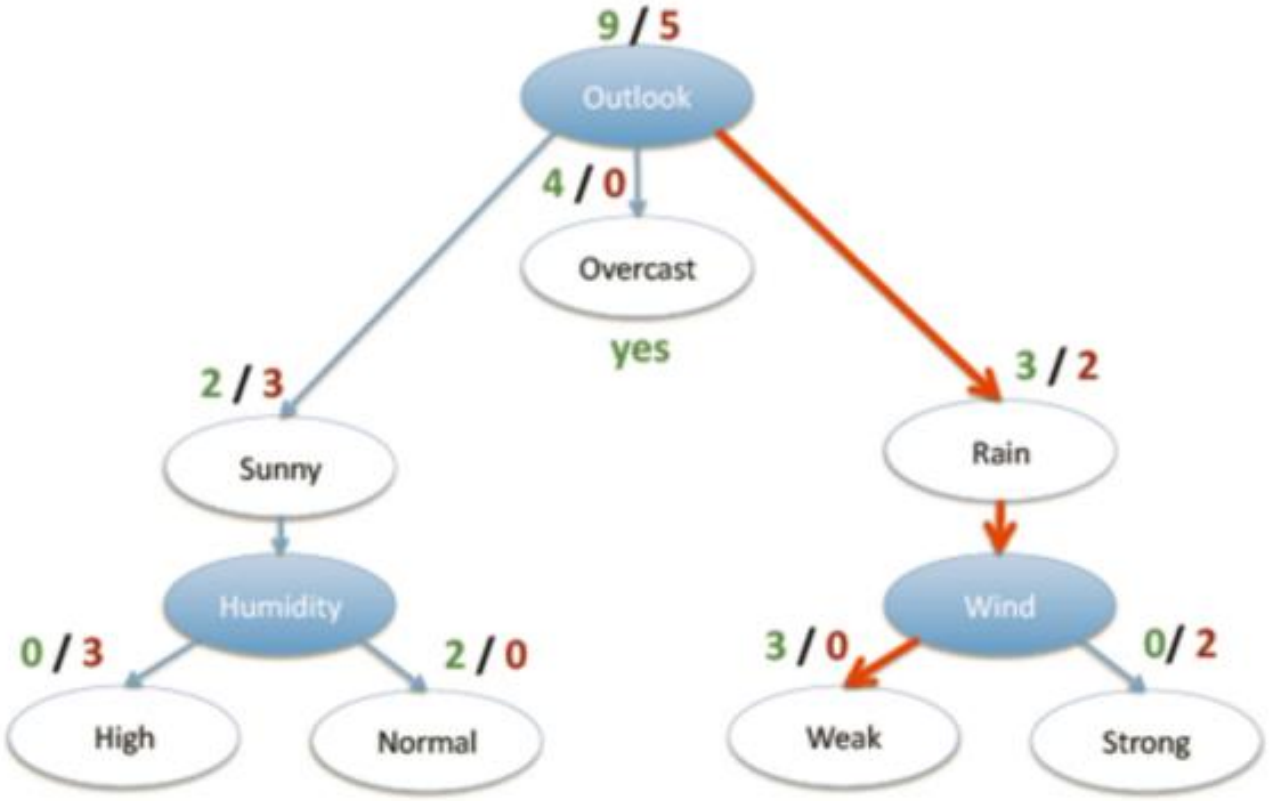
  - 决策树：从根节点开始一步步走到叶子节点（决策）

  - 所有的数据最终都会落到叶子节点，既可以做分类也可以做回归



决策树

- 树的组成
  - 根节点：第一个选择点
  - 非叶子节点与分支：中间过程
  - 叶子节点：最终的决策结果



决策树

- 决策树的训练与测试

  - 训练阶段：从给定的训练集构造出来一棵树（从跟节点开始选择特征，如何进行特征切分）

  - 测试阶段：根据构造出来的树模型从上到下去走一遍就好了

  - 一旦构造好了决策树，那么分类或者预测任务就很简单了，只需要走一遍就可以了，那么难点就在于如何构造出来一颗树，这就没那么容易了，需要考虑的问题还有很多的！

决策树

- 如何切分特征（选择节点）

  - 问题：根节点的选择该用哪个特征呢？接下来呢？如何切分呢？

  - 想象一下：我们的目标应该是根节点就像一个老大似的能更好的切分数据（分类的效果更好），根节点下面的节点自然就是二当家了。

  - 目标：通过一种衡量标准，来计算通过不同特征进行分支选择后的分类情况，找出来最好的那个当成根节点，以此类推。

决策树

- 衡量标准-熵

  - 熵：熵是表示随机变量不确定性的度量
    （解释：说白了就是物体内部的混乱程度，比如杂货市场里面什么都有那肯定混乱呀，专卖店里面只卖一个牌子的那就稳定多啦）

  - 公式：$H(X)=-\sum pi * logpi, i=1,2,\dots,n$

  - 一个栗子：　　A集合[1,1,1,1,1,1,1,1,2,2]
    B集合[1,2,3,4,5,6,7,8,9,1]

    显然A集合的熵值要低，因为A里面只有两种类别，相对稳定一些
    而B中类别太多了，熵值就会大很多。（在分类任务中我们希望通过节点分支后数据类别的熵值大还是小呢？）

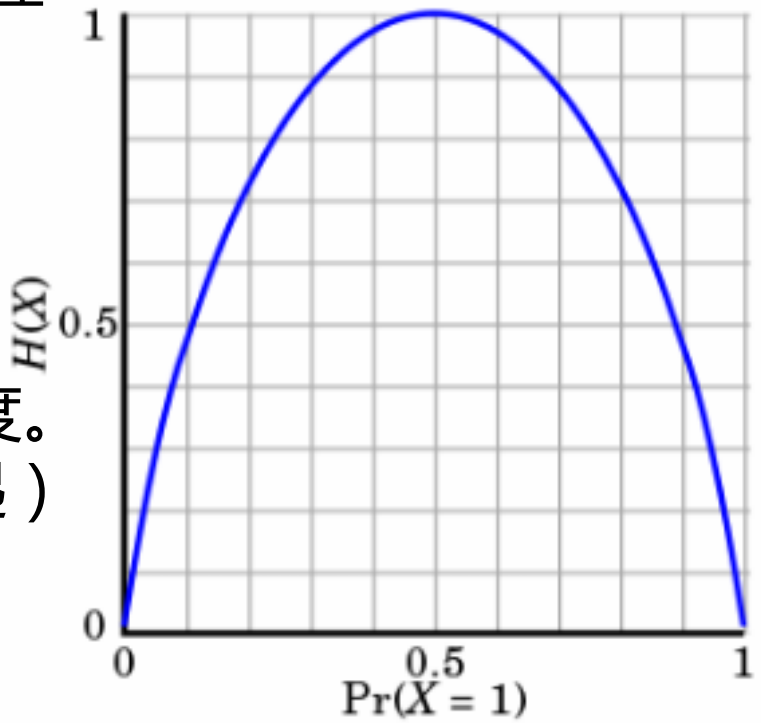
决策树

- 衡量标准-熵

  - 熵：不确定性越大，得到的熵值也就越大
    当p=0或p=1时，H(p)=0,随机变量完全没有不确定性
    当p=0.5时，H(p)=1,此时随机变量的不确定性最大

  - 如何决策一个节点的选择呢？

  - 信息增益：表示特征X使得类Y的不确定性减少的程度。
    （分类后的专一性，希望分类后的结果是同类在一起）



决策树

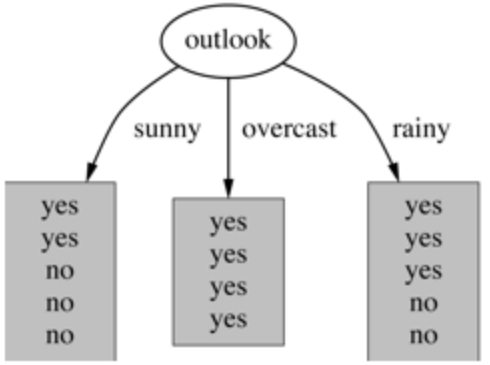
- 决策树构造实例
  - 数据：14天打球情况
  - 特征：4种环境变化
  - 目标：构造决策树

outlook	temperature	humidity	windy	play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

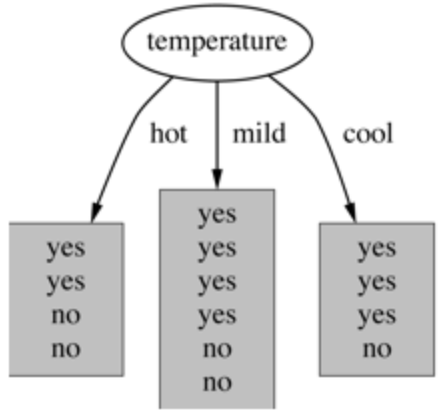
决策树

- 决策树构造实例
- 划分方式：4种
- 问题：谁当根节点呢？
- 依据：信息增益

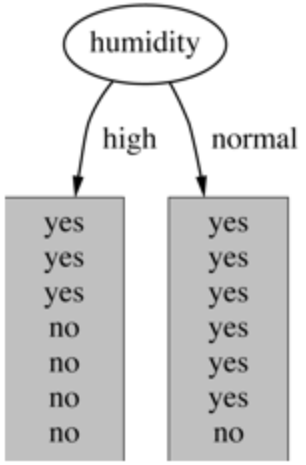
1.基于天气的划分



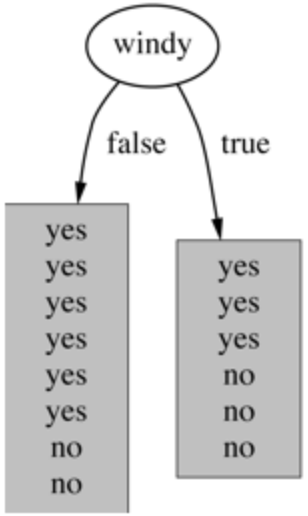
2.基于温度的划分



3.基于湿度的划分



4.基于有风的划分



决策树

- 决策树构造实例

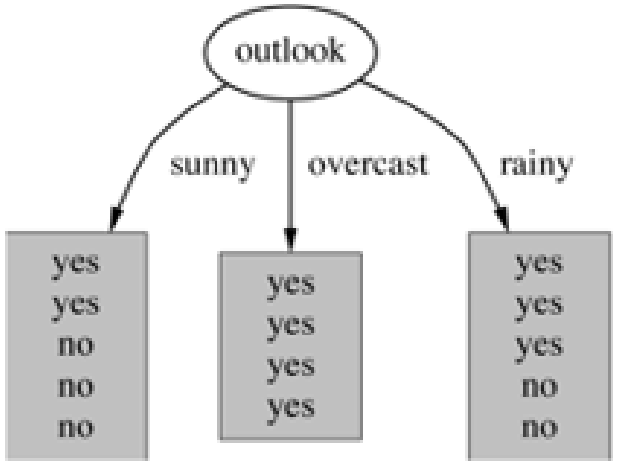
在历史数据中（14天）有9天打球，5天不打球，所以此时的熵应为：

$$-\frac{9}{14}\log_2\frac{9}{14}-\frac{5}{14}\log_2\frac{5}{14}=0.940$$

4个特征逐一分析，先从outlook特征开始：

Outlook = sunny时，熵值为0.971
Outlook = overcast时，熵值为0
Outlook = rainy时，熵值为0.971

1.基于天气的划分



决策树

决策树构造实例

- 根据数据统计，outlook取值分别为sunny,overcast,rainy的概率分别为：5/14, 4/14, 5/14
- 熵值计算：5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971 = 0.693

  （gain(temperature)=0.029 gain(humidity)=0.152 gain(windy)=0.048）
- 信息增益：系统的熵值从原始的0.940下降到了0.693，增益为0.247
- 同样的方式可以计算出其他特征的信息增益，那么我们选择最大的那个就可以啦，相当于是遍历了一遍特征，找出来了大当家，然后再其余的中继续通过信息增益找二当家！

决策树

- 决策树算法
  - ID3：信息增益（有什么问题呢？）
  - C4.5：信息增益率（解决ID3问题，考虑自身熵）
  - CART：使用GINI系数来当做衡量标准
  - GINI系数：$Gini(p)=\sum_{k=1}^{K} p_k(1-p_k)=1-\sum_{k=1}^{K} p_k^2$

    （和熵的衡量标准类似，计算方式不相同）

决策树

- 连续值怎么办？

选取(连续值的)哪个分界点？

- 贪婪算法！

I. 排序

60 70 75 85 90 95 100 120 125 220

若进行“二分”，则可能有9个分界点。

例子：

60 70 75 85 90 95 100 120 125 220

分割成TaxIn<=80 和 TaxIn>80

60 70 75 85 90 95 100 120 125 220

分割成TaxIn<=97.5 和 TaxIn>97.5

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- 实际上，这就是“离散化”过程

决策树

- 决策树剪枝策略

  - 为什么要剪枝：决策树过拟合风险很大，理论上可以完全分得开数据（想象一下，如果树足够庞大，每个叶子节点不就一个数据了嘛）
  - 剪枝策略：预剪枝，后剪枝
  - 预剪枝：边建立决策树边进行剪枝的操作（更实用）
  - 后剪枝：当建立完决策树后来进行剪枝操作

决策树

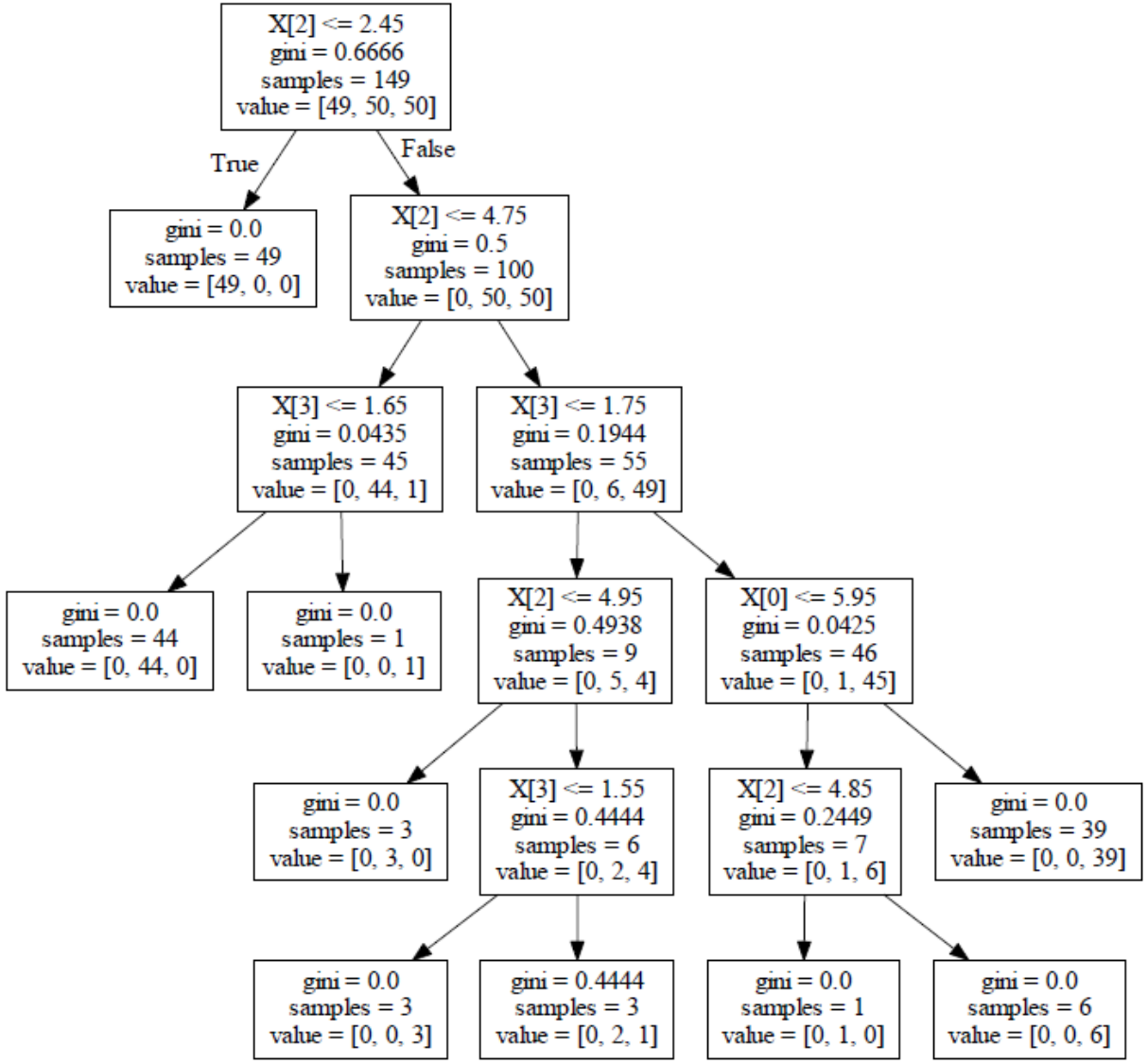
- 决策树剪枝策略

  - 预剪枝：限制深度，叶子节点个数
    叶子节点样本数，信息增益量等

  - 后剪枝：通过一定的衡量标准

$$C_\alpha(T) = C(T) + \alpha \cdot |T_{leaf}|$$

（叶子节点越多，损失越大）



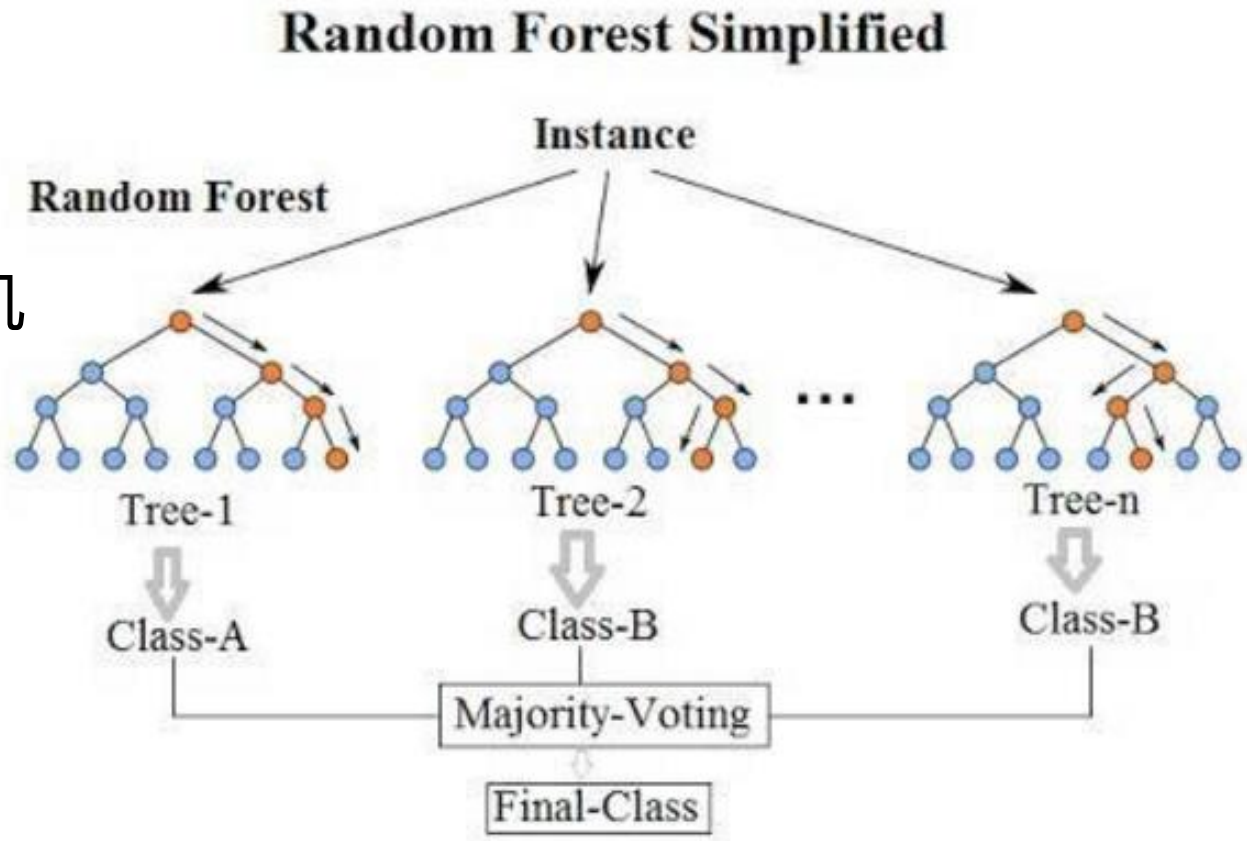
集成算法

- Ensemble learning
  - 目的：让机器学习效果更好，单个不行，群殴走起
  - Bagging：训练多个分类器取平均 $f(x) = 1/M \sum_{m=1}^{M} f_m(x)$
  - Boosting：从弱学习器开始加强，通过加权来进行训练

    $F_m(x) = F_{m-1}(x) + argmin_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$ （加入一棵树，要比原来强）
  - Stacking：聚合多个分类或回归模型（可以分阶段来做）

集成算法

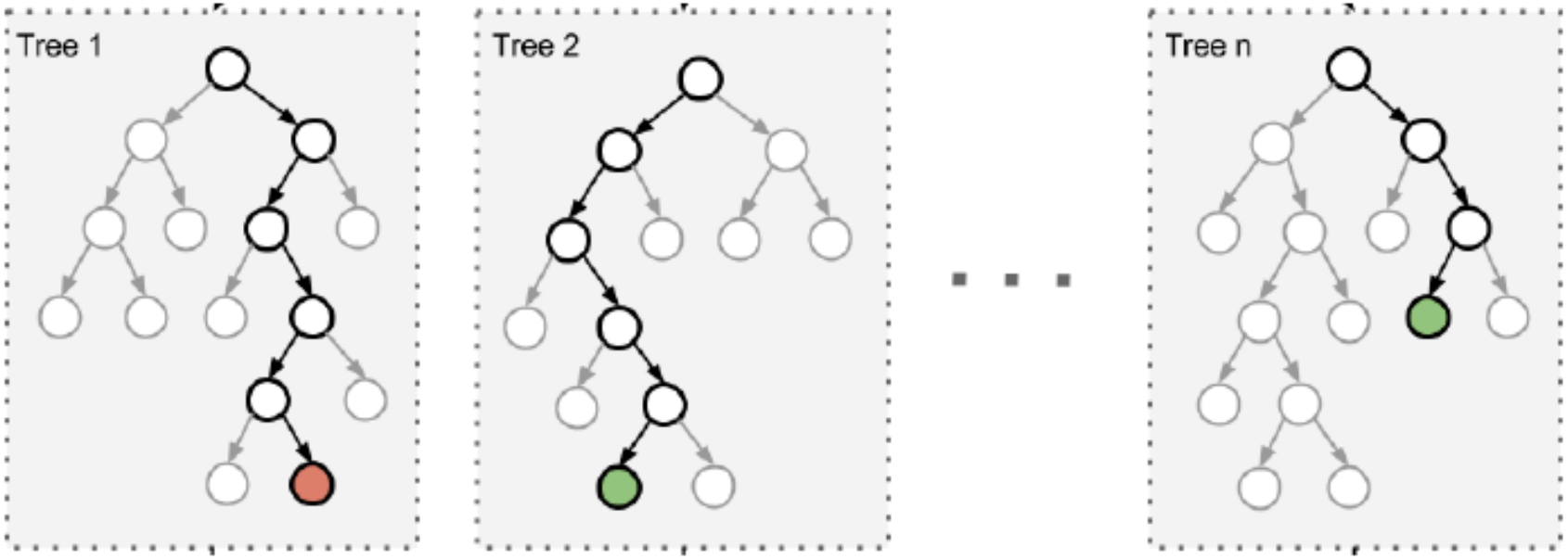
- Bagging模型
  - 全称： bootstrap aggregation（说白了就是并行训练一堆分类器）
  - 最典型的代表就是随机森林啦
  - 随机：数据采样随机，特征选择随机
  - 森林：很多个决策树并行放在一起



集成算法

- 随机森林

- 构造树模型：

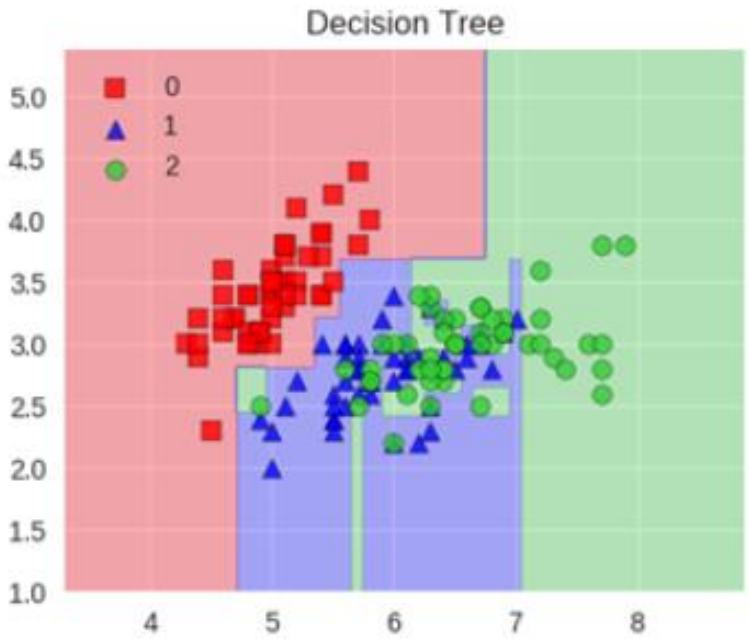


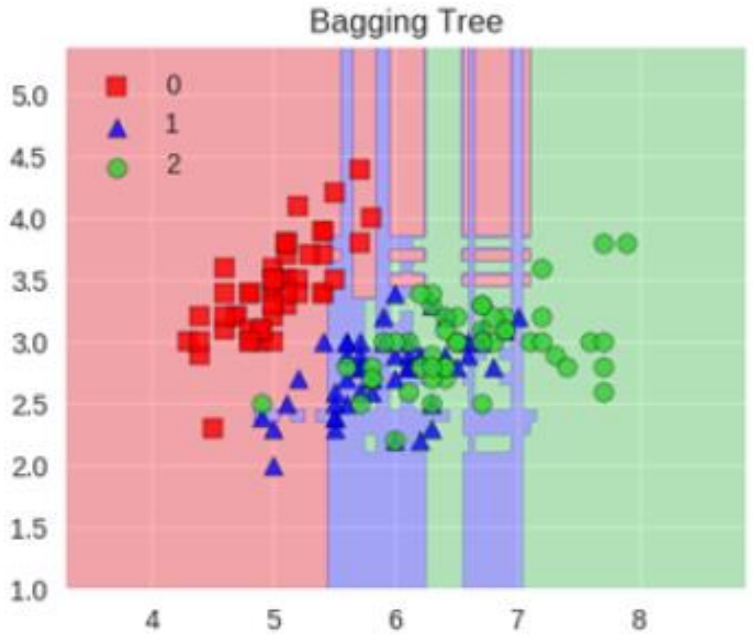
- 由于二重随机性，使得每个树基本上都不会一样，最终的结果也会不一样

集成算法

- Bagging模型

  - 树模型：

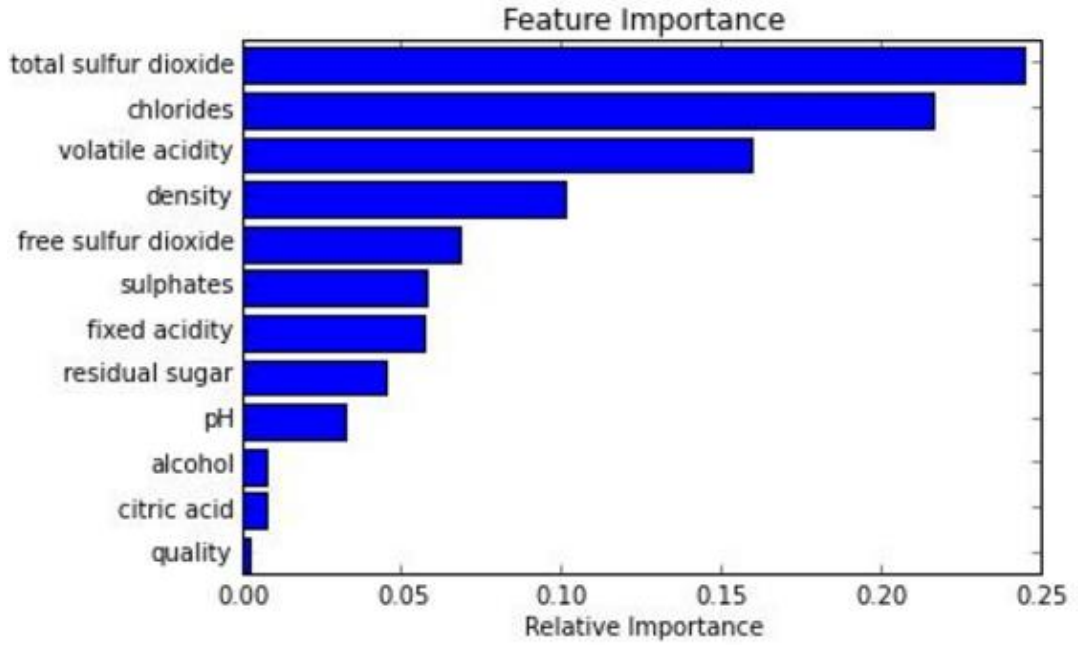




  - 之所以要进行随机，是要保证泛化能力，如果树都一样，那就没意义了！

集成算法

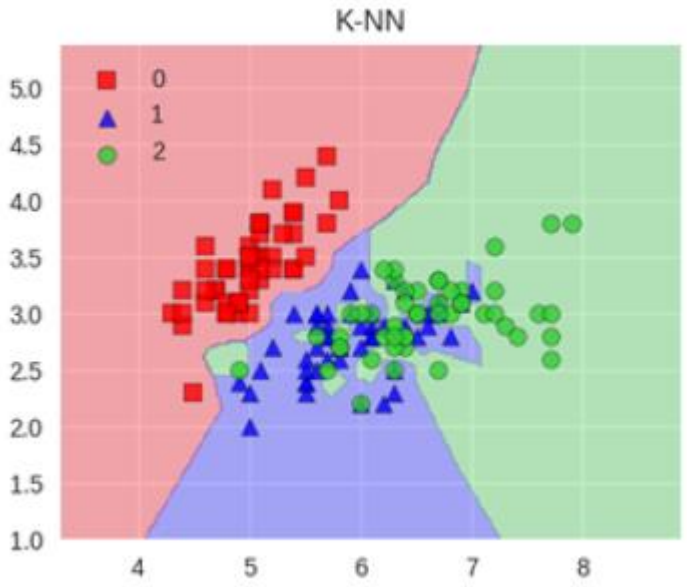
- 随机森林优势
  - 它能够处理很高维度（feature很多）的数据，并且不用做特征选择
  - 在训练完后，它能够给出哪些feature比较重要
  - 容易做成并行化方法，速度比较快
  - 可以进行可视化展示，便于分析

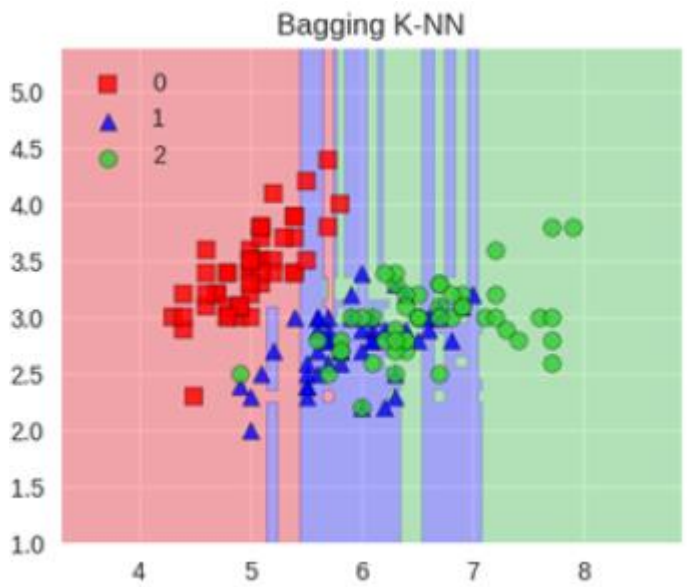


集成算法

- Bagging模型

- KNN模型：

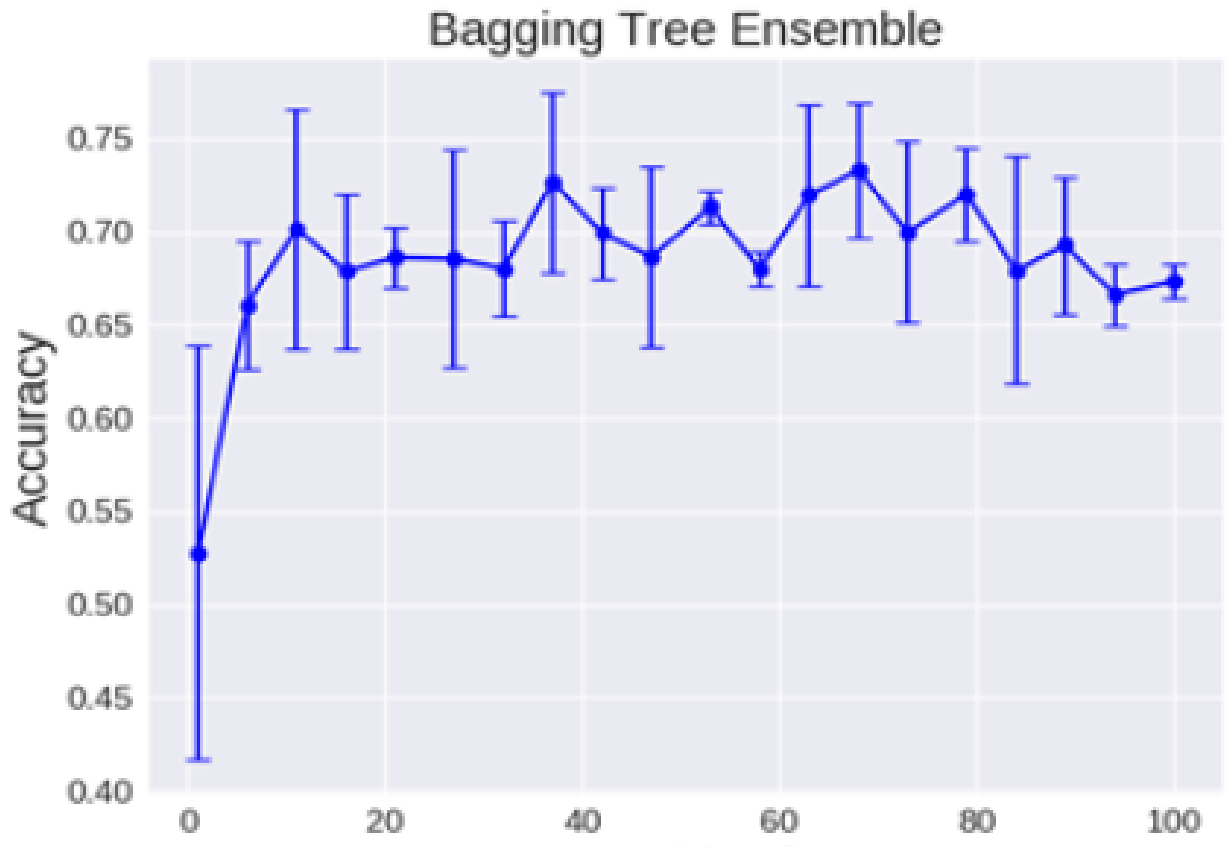




- KNN就不太适合，因为很难去随机让泛化能力变强！

集成算法

- Bagging模型
  - 树模型：



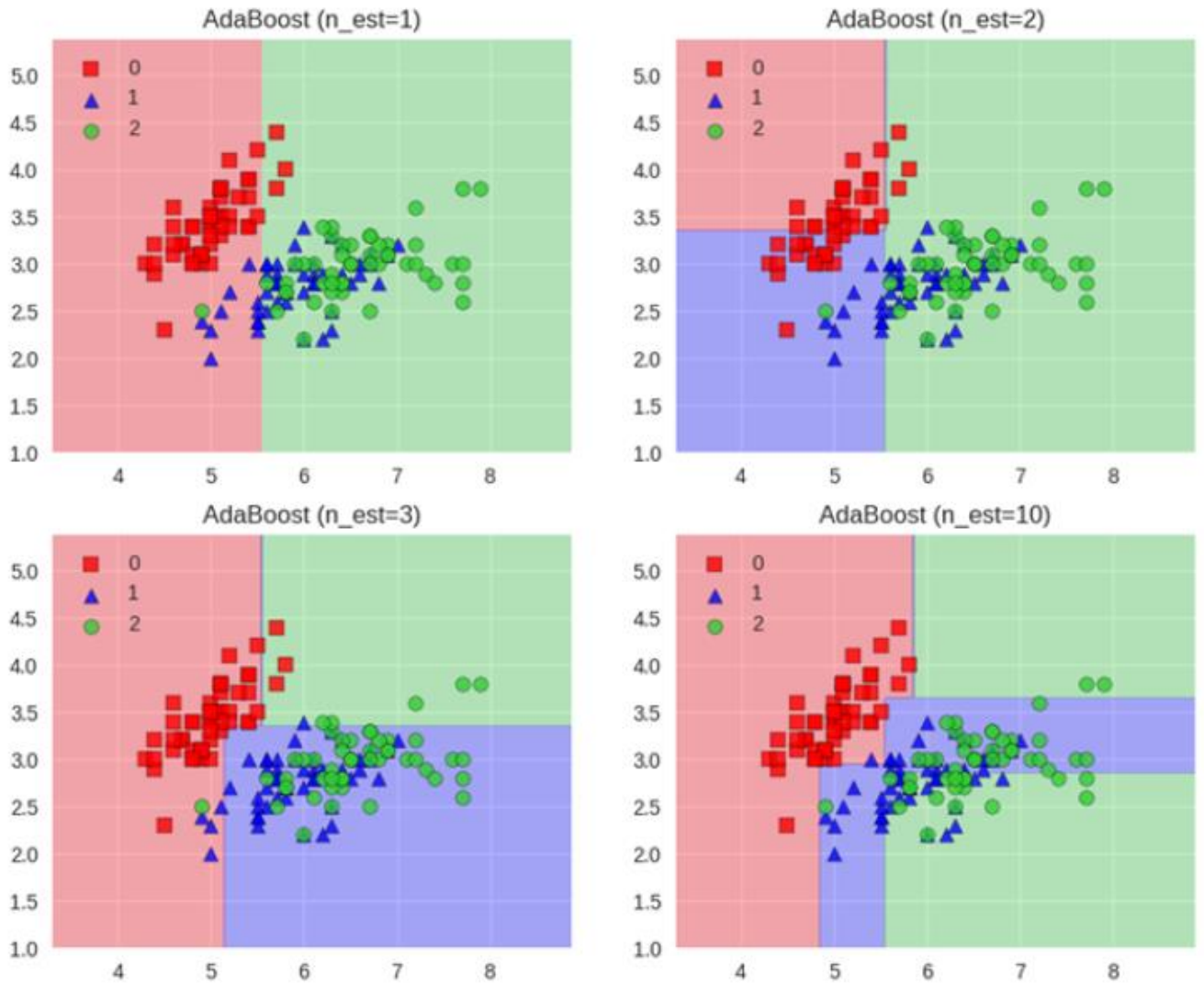
  - 理论上越多的树效果会越好，但实际上基本超过一定数量就差不多上下浮动了

集成算法

- Boosting模型
  - 典型代表：AdaBoost， Xgboost
  - Adaboost会根据前一次的分类效果调整数据权重
  - 解释：如果某一个数据在这次分错了，那么在下一次我就会给它更大的权重
  - 最终的结果：每个分类器根据自身的准确性来确定各自的权重，再合体

集成算法

- Adaboost工作流程
  - 每一次切一刀！
  - 最终合在一起
  - 弱分类器这就升级了！

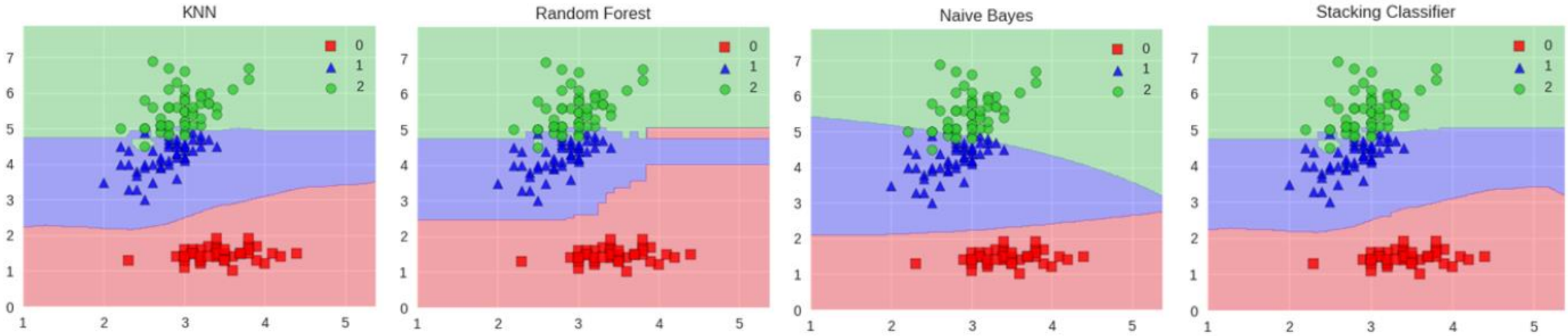


集成算法

- Stacking模型
  - 堆叠：很暴力，拿来一堆直接上（各种分类器都来了）
  - 可以堆叠各种各样的分类器（KNN,SVM,RF等等）
  - 分阶段：第一阶段得出各自结果，第二阶段再用前一阶段结果训练
  - 为了刷结果，不择手段！

集成算法

- Stacking模型



- 堆叠在一起确实能使得准确率提升，但是速度是个问题
  集成算法是竞赛与论文神器，当我们更关注于结果时不妨来试试！