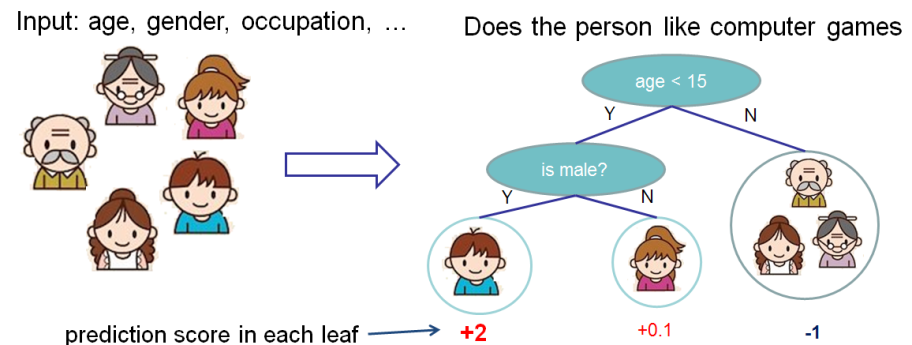
牛刀小试——利用XGBoost预测海藻数据

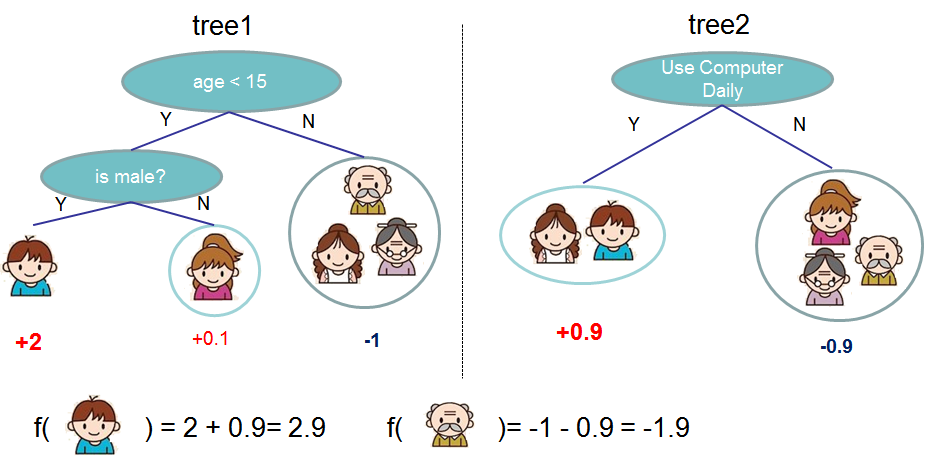
1. 关于XGBoost

XGBoost是一组分类回归树（CART）的集成。[1]

举个例子，在单棵分类回归树上，我们将一个家庭的成员通过一些特征（如年龄）分类为不同的叶子，并在相应的叶子上为他们分配分数，以判断某个成员是否喜欢电脑游戏。



通常，单棵树的强度不足以在实践中使用。实际使用的是集合模型，它将多棵树的预测结合在一起。



这是两棵树的树集合的例子。将每个树的预测分数相加以得到最终分数。利用数学语言描述我们的模型：



这里是树的数量，是函数空间里的一个函数，也是所有可能的CART的集合。我们的目标函数由下式给出：



我们模型的参数是，它控制了树的结构和节点上的得分。训练的难度远远高于一般机器学习中参数的训练，为此，我们采用一种**添加策略**：固定我们已经学习到的模型，再每次向模型里添加一棵树。我们将第次得到的预测记为，于是我们有：



现在的问题是，在每一次，我们应该添加怎样的树。一个很自然的想法是，我们添加的树要使得我们的目标函数最优。



若我们利用MSE作为我们的损失函数，那么我们的目标函数变为：

其中



该定义的一个重要优点是目标函数的值仅取决于 和 。

下面我们定义模型的复杂度，为此我们首先完善的定义：



其中是叶子上的分数矢量，是一个将每个数据点分配给相应叶子的映射，是叶子的数量。在XGBoost中，我们将复杂性定义为



当然，定义复杂性的方法不止一种，但这种方法在实践中效果很好。

在重新制定树模型之后，我们可以写出在加入了第t棵树后的目标函数：



这里



最后的目标函数是关于的二次函数，这样它的极小值点和极小值分别为：



最后一个式子正是衡量树结构好坏的标准，值越小代表结构越好。

有了这个标准后，我们理应尝试所有可能并选择最优，然而太费时。于是我们每次优化树的一层。我们假设一个叶子分裂为两个叶子，则它的得分增加为：



如果增益小于，我们就不将此叶子分裂。

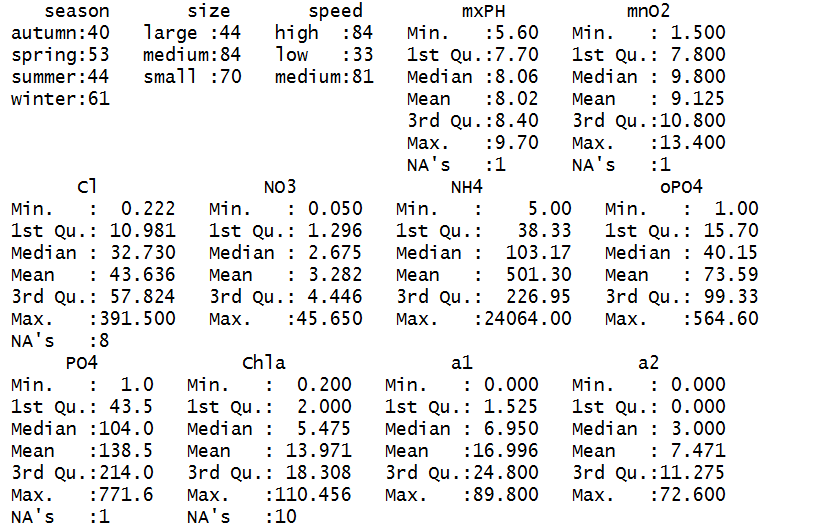
1. 应用到海藻数据集

首先导入数据集：

1. library(DMwR)
2. data(algae)

查看数据集的整体描述：

1. summary(algae)



我们有season等11个自变量，a1等7个因变量，其中有一些缺失值。我们利用内置函数对缺失值进行处理：

1. algae=algae[-manyNAs(algae),]
2. clean.algae=knnImputation(algae,k=10)

我们删去了一条缺失值较多的记录，剩下199条记录。

注意到season等三个变量是因子型变量，我们对它们进行独热编码处理，使之成为数值型变量：

1. seosan\_one <- model.matrix(~season-1,clean.algae)
2. size\_one <- model.matrix(~size-1,clean.algae)
3. speed\_one <- model.matrix(~speed-1,clean.algae)

于是我们得到了最终的数据集clean.algae\_matrix：

1. clean.algae\_num=clean.algae[,4:11]
2. clean.algae\_num=cbind(clean.algae\_num,seosan\_one,size\_one,speed\_one)
3. clean.algae\_matrix=data.matrix(clean.algae\_num)
4. label\_a1=clean.algae$a1

为了能对模型进行测试，我们将数据集划分为训练集和测试集：

1. numberOfTrainingSamples <- round(length(label\_a1) \* .75)
2. train\_data <- clean.algae\_matrix[1:numberOfTrainingSamples,]
3. train\_labels <- label\_a1[1:numberOfTrainingSamples]
4. test\_data <- clean.algae\_matrix[-(1:numberOfTrainingSamples),]
5. test\_labels <- label\_a1[-(1:numberOfTrainingSamples)]

将数据格式转换为DMatrix：

1. dtrain <- xgb.DMatrix(data = train\_data, label= train\_labels)
2. dtest <- xgb.DMatrix(data = test\_data, label= test\_labels)

下面训练XGBoost：

1. model <- xgboost(data = dtrain,
2. nround = 3,
3. max.depth = 2,
4. lambda=0.01,
5. feature\_selector="shuffle",
6. objective = "reg:linear")

nround是训练轮数，也即树的棵树；

max.depth每棵树的最大深度；

lambda是正则化参数；

feature\_selector是变量选择方法，这里我们选择的是shuffle；

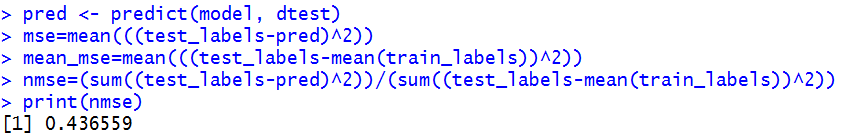
objective 是针对的问题的类型，由于是回归预测问题，我们选择的是reg:linear。

选择nround为3，max.depth为2，这是网格寻优的结果：

1. max.depths = c(1, 9)
2. nround=c(1,100)
4. best\_params = 0
5. best\_score = 0
7. count = 1
8. **for**( depth in max.depths ){
9. **for**( num in nround){
11. bst\_grid = xgb.train(data = dtrain,
12. nround = num,
13. max.depth = depth,
14. lambda=0.01,
15. feature\_selector="shuffle",
16. objective = "reg:linear")
18. **if**(count == 1){
19. best\_params = bst\_grid$params
20. best\_score = bst\_grid$best\_score
21. count = count + 1
22. }
23. **else** **if**( bst\_grid$best\_score < best\_score){
24. best\_params = bst\_grid$params
25. best\_score = bst\_grid$best\_score
26. }
27. }
28. }
30. best\_params
31. best\_score

模型的评价：

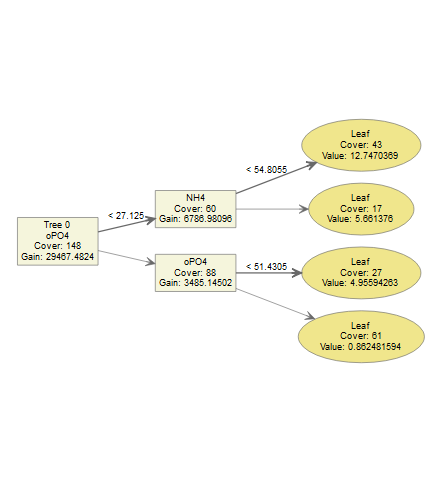
1. pred <- predict(model, dtest)
2. mse=mean(((test\_labels-pred)^2))
3. mean\_mse=mean(((test\_labels-mean(train\_labels))^2))
4. nmse=(sum((test\_labels-pred)^2))/(sum((test\_labels-mean(train\_labels))^2))
5. print(nmse)

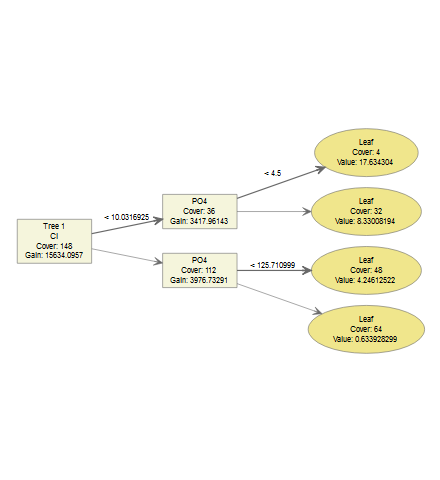


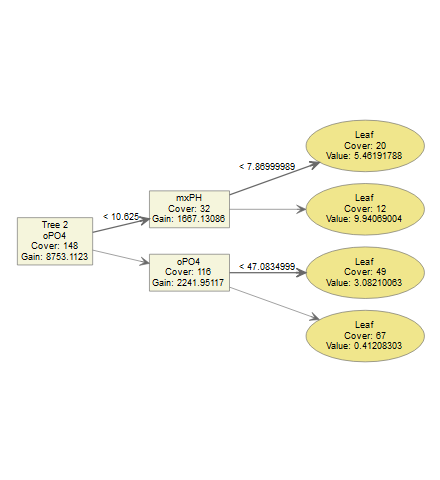
可以看到我们的nmse为0.43，效果略好于随机森林。

结果的可视化：

我们的模型的结果是下面的三棵树的集成：







对于每个记录，依次在三棵树上进行。下一棵树的输入是上一棵树的输出与真实标签的残差，也就是说下一棵树实际上学习的是上一棵树的残差。最终的预测结果是在三棵树上的预测结果之和。

1. 实验结论

我们利用XGBoost预测了海藻数据，在a1上的nmse为0.436，好于随机森林，并给出了形象化的解释。