Weekly Summary 6

Seline Yang

3/26/2018

1. Regression Trees

Purpose: To try and make predictions, as well as understand relationship among predictors and response variables for numerical data, by using clustering.

Summary: There are two steps to build a regression tree: divide the predictor space, the set of possible values for $X_1, X_2, ..., X_p$ into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$; for every observation that falls into the region R_j, we make the same prediction.

Pros: Regression trees implicitly perform variable screening or feature selection. It has a nice graphical representation. Nonlinear relationships between parameters do not affect tree performance. It is easy to interpret and explain. And we don't need to warry about normalizing the data.

Cons: Preparing regression trees, especially large ones with many branches are complex and time comsuming. A small change in the input data can cause large changes in the tree. The decisions contained in the decision tree are based on expectations, and irrational expectations can lead to flaws and errors in the decision tree.

R Commands:

```
library(tree)
library(rpart)

# Grow the tree
tree.hitters <- rpart(log(Salary) ~ Years + Hits,
   method="anova", data=Hitters)
#Visualizing the tree
plot(tree.hitters, uniform=TRUE,
   main="Regression Tree for log(Salary)", margin = .2)
text(tree.hitters, use.n=TRUE, all=TRUE, cex=.8)

set.seed(1)
n <- nrow(Hitters)
train <- sample( 1:n, .6*n)
test <- c(1:n)[-train]

yhat <- predict(tree.hitters, newdata = Hitters[test,])
mean(log(Hitters$Salary)[test] - yhat)^2
summary(tree.hitters)
```

2. Recursive Binary Splitting

Purpose: It is not feasible to consider every possible partition in the tree. We use recursive binary splitting to help us create regions.

Summary: It is a top-down, greedy approach. We start at the top of the tree, with every observation in one node. Then we split into two spaces, so that out training observations fall into one of the boxes, and we make choice of the first split based on minimized SSE. We do not consider what might help us make better choices later on. Then we repeat the process to split smaller regions until we reach some stopping criterion.

Pros: Giving a better looking partitioning. Allows varing prioritizing of misclassifications in order to create a decition rule that has more sesitivity or specificity.

Cons: We can only use exactly one predictor variable to defined the split. The original input variables are projected into reduced set of components, so their individual effect no longer identifiables. May overfir data.

R Commands: This part is very long, see github reference: https://github.com/jspmccain/binary-splitting---regression-tree/blob/master/recursive%20binary%20splitting.R

3. Pruning

Purpose: Reduce the number of predictor in the model.

Summary: We choose among all the possible sub-tree from our original tree to prune. We choose the tree with the lowerst test MSE. We use cross-validation in the prunning process. Cost complexity pruning is used.

Pros: Reduced the complexity of overfitting. Smaller tree with fewer splits might lead to lower variance and better interpretation.

Cons: May leads to bias.

R Commands:

```
prune.tree<- prune(tree.hitters, cp=0.01690198)
```

4. Cost complexity pruning

Purpose: To select smaller set of subtrees for prunning consideration.

Summary: It is also known as weakest link prinning. Rather than considering a sequence os trees indexed by a nonnegative tuning parameter α. The choice of α imposes a penaltyon sub-trees based on the number of leaves in the tree. We use CV to help us choose α. In cost complexity pruning, we minimize

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y_i})^2 + \alpha|T|.$$

Pros: Add penalty to the number of leaves, so we will have smaller sub-tree.

Cons: May leads to bias. Time consuming with big data set

R Commands:

```
tree.hitters <- rpart(log(Salary) ~ Years + Hits,
   method="anova", data=Hitters)
tree.hitters$cptable # display the results of CV
tree.hitters$cptable[which.min(tree.hitters$cptable[,"xerror"]),"CP"]
plotcp(tree.hitters) # visualize cross-validation results
prune.tree<- prune(tree.hitters, cp=0.01690198)
```