Boosting

Mats Hansson

17 oktober 2016

Boosting

These methods use trees as building blocks to build more complex models. Here we will use the Boston Houseing data to explore boosting. These data are in the `MASS`packages. It gives housing values others statistics in each of 506 suburbs of Boston based on a 1970 census.

Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and a family of machine learning algorithms which convert weak learners to strong ones.

Lets load the packages and create a training data.

```
library(ISLR)
library(gbm)
```

```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.1
```

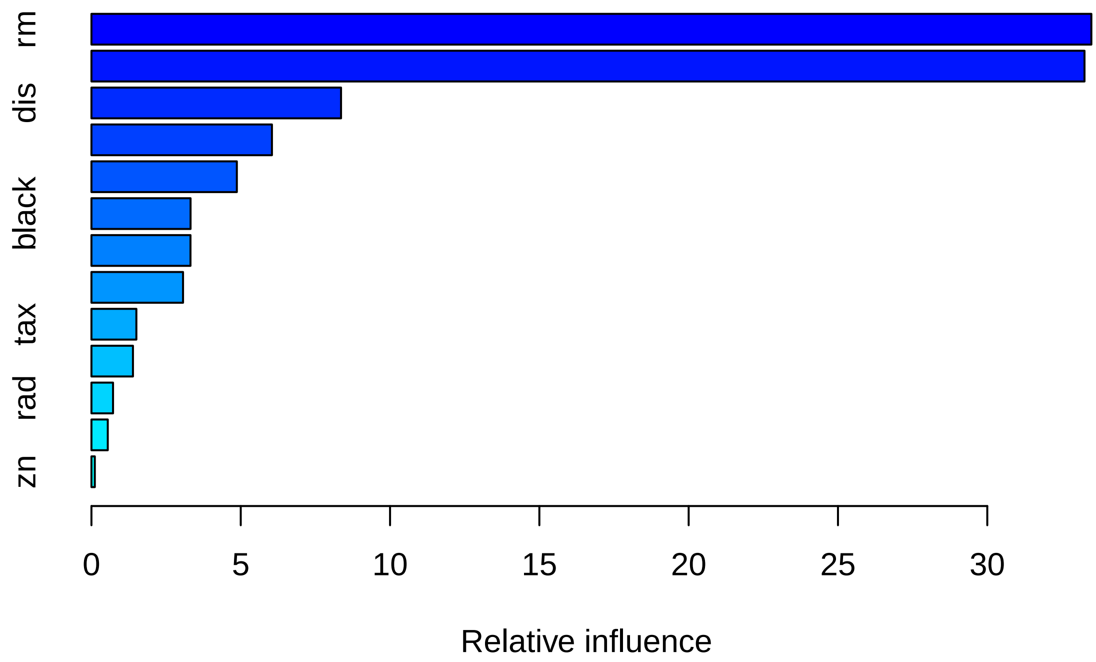
```
library(MASS)

set.seed(101)

train=sample(1:nrow(Boston),300)
```

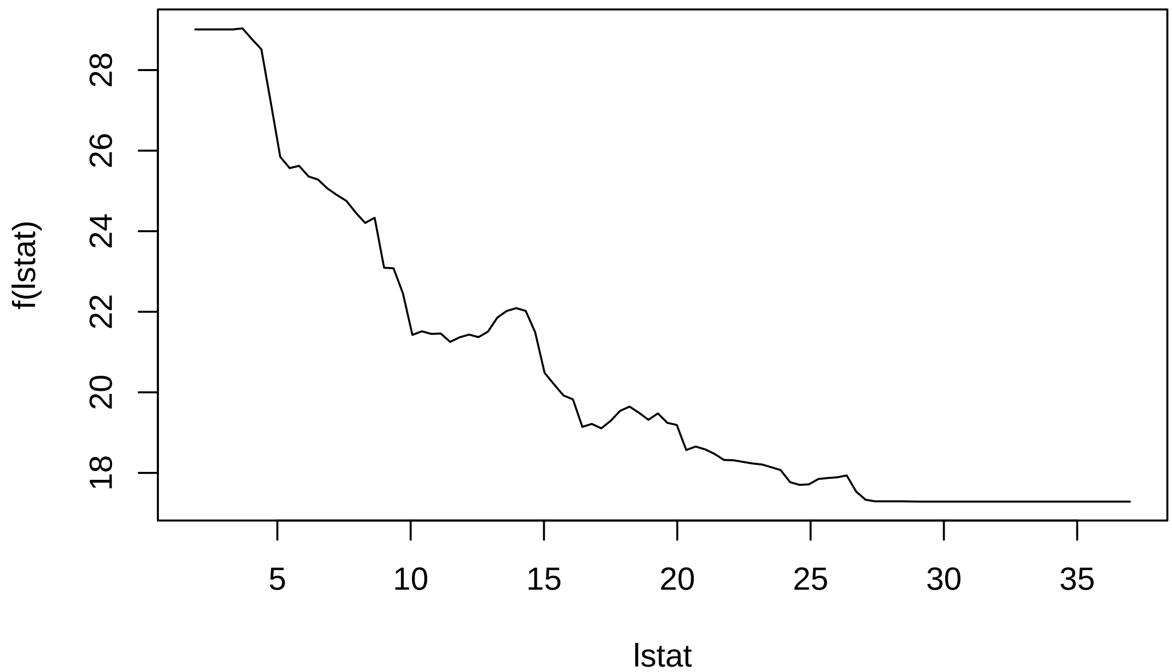
Boosting builds lots of smaller trees. Unlike random forests, each new tree in boosting tries to patch up the deficiencies of the current ensamble.

```
boost.boston=gbm(medv~.,data=Boston[train,], distribution="gaussian", n.trees=10000,
    shrinkage=0.01, interaction.depth=4)
summary(boost.boston)
```

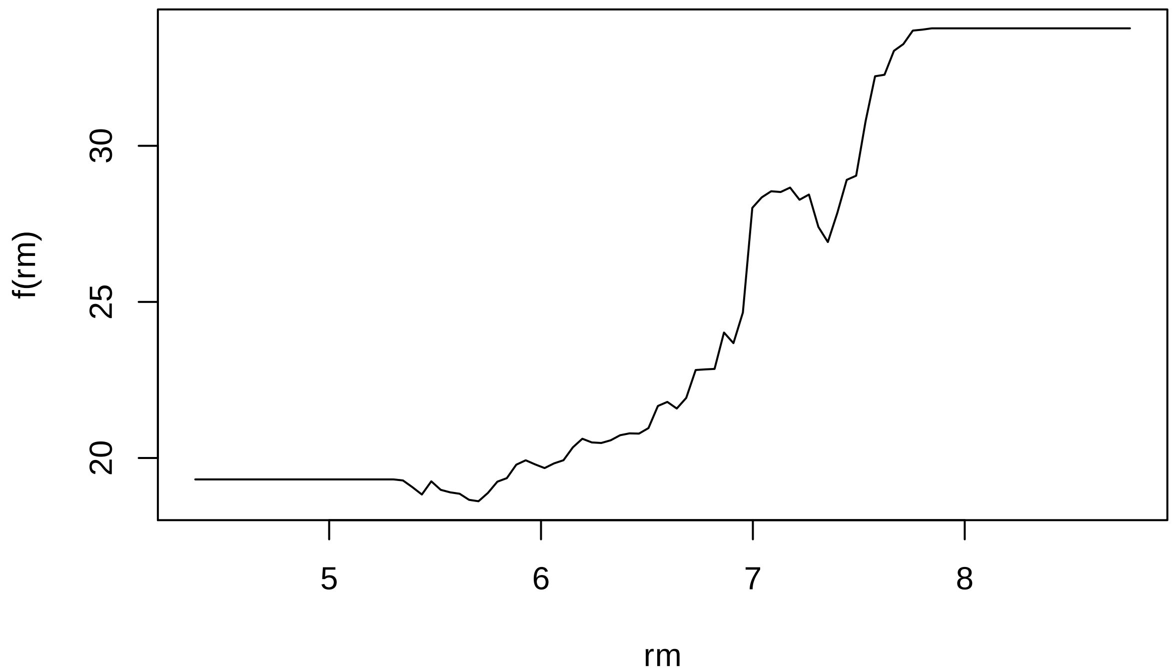



```
##              var    rel.inf
## rm            rm 33.4858251
## lstat      lstat 33.2567329
## dis          dis  8.3591410
## crim        crim  6.0449106
## nox          nox  4.8699097
## black      black  3.3197609
## ptratio ptratio  3.3162724
## age          age  3.0673985
## tax          tax  1.5049077
## chas        chas  1.3903596
## rad          rad  0.7237600
## indus      indus  0.5471690
## zn            zn  0.1138526
```

```r
plot(boost.boston, i="lstat")
```

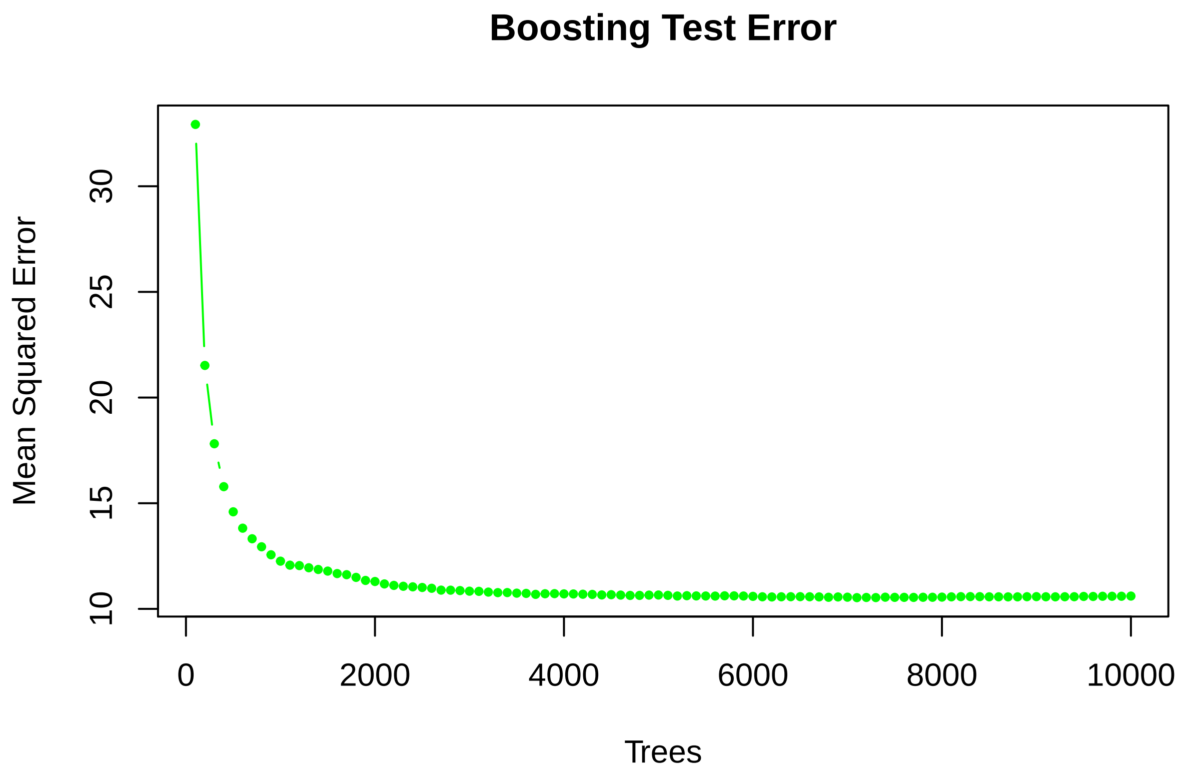



```r
plot(boost.boston, i="rm")
```



Lets make a prediction on the test set. With boosting, the number of trees is a tuning parameter, and if we have too many we can overfit. So we should use cross-validation to select the numver if trees. Instead, we will compute the test error as a function of the numver of trees, and make a plot.

```
n.trees=seq(from=100, to=10000, by=100)
predmat=predict(boost.boston, newdata=Boston[-train,], n.trees=n.trees)
## dim(predmat)
berr=with(Boston[-train,], apply((predmat-medv)^2,2,mean))
plot(n.trees, berr, pch=19, ylab="Mean Squared Error", xlab="Trees",
    main="Boosting Test Error", cex=0.5, type="b", col="green")
```



Comparing different interaction.

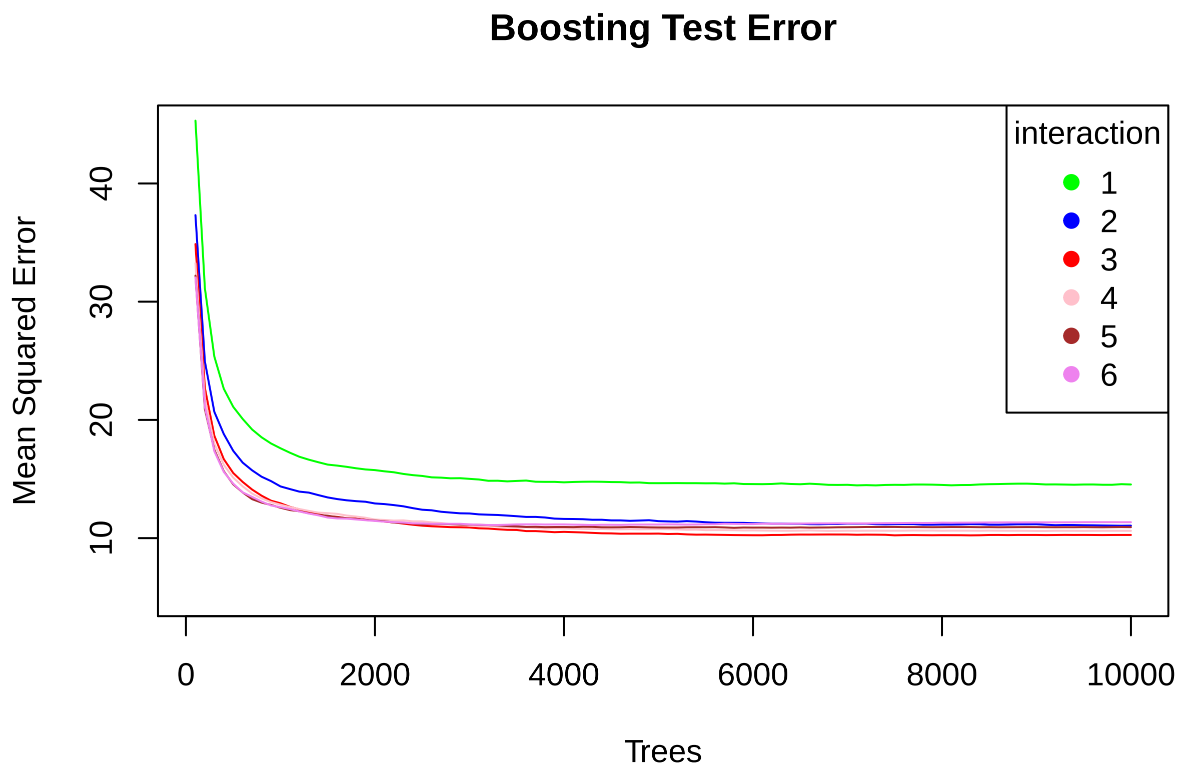
```
berrmat=matrix(NA, length(n.trees), 6)

for (i in 1:6) {
boost.boston=gbm(medv~.,data=Boston[train,], distribution="gaussian", n.trees=10000,
    shrinkage=0.01, interaction.depth=i)
  predmat=predict(boost.boston, newdata=Boston[-train,], n.trees=n.trees)
  berrmat[,i]=with(Boston[-train,], apply((predmat-medv)^2,2,mean))
}

plot(n.trees, berrmat[,1], pch=19, ylab="Mean Squared Error", xlab="Trees",
    main="Boosting Test Error", cex=1, type="l", col="green", ylim=c(5,45))
lines(n.trees, berrmat[,2], cex=1, type="l", col="blue")
lines(n.trees, berrmat[,3], cex=1, type="l", col="red")
lines(n.trees, berrmat[,4], cex=1, type="l", col="pink")
lines(n.trees, berrmat[,5], cex=1, type="l", col="brown")
lines(n.trees, berrmat[,6], cex=1, type="l", col="violet")



legend("topright",title="interaction", legend=c("1", "2","3","4","5","6"), pch=19,
       col=c("green", "blue","red","pink","brown","violet"))
```



as we can see in the plot above the red curve has the lowest mean squared error the mean on the error is.

```
apply(berrmat, 2, mean)
```

```
## [1] 15.77793 12.47834 11.31602 11.54488 11.62591 11.83760
```