Random_forest

Vinoth Aryan Nagabosshanam

March 25, 2017

Random Forests and Boosting

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

Random Forests

Random forests build lots of bushy trees, and then average them to reduce the variance.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.3.3
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.3.3
```

```
set.seed(101)
dim(Boston)
```

```
## [1] 506  14
```

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

```
## starting httpd help server ...
```

```
##  done
```

Lets fit a random forest and see how well it performs. We will use the response `medv`, the median housing value (in $1K dollars)

```
rf.boston=randomForest(medv~.,data=Boston,subset=train)
rf.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.62686
##                     % Var explained: 84.74
```

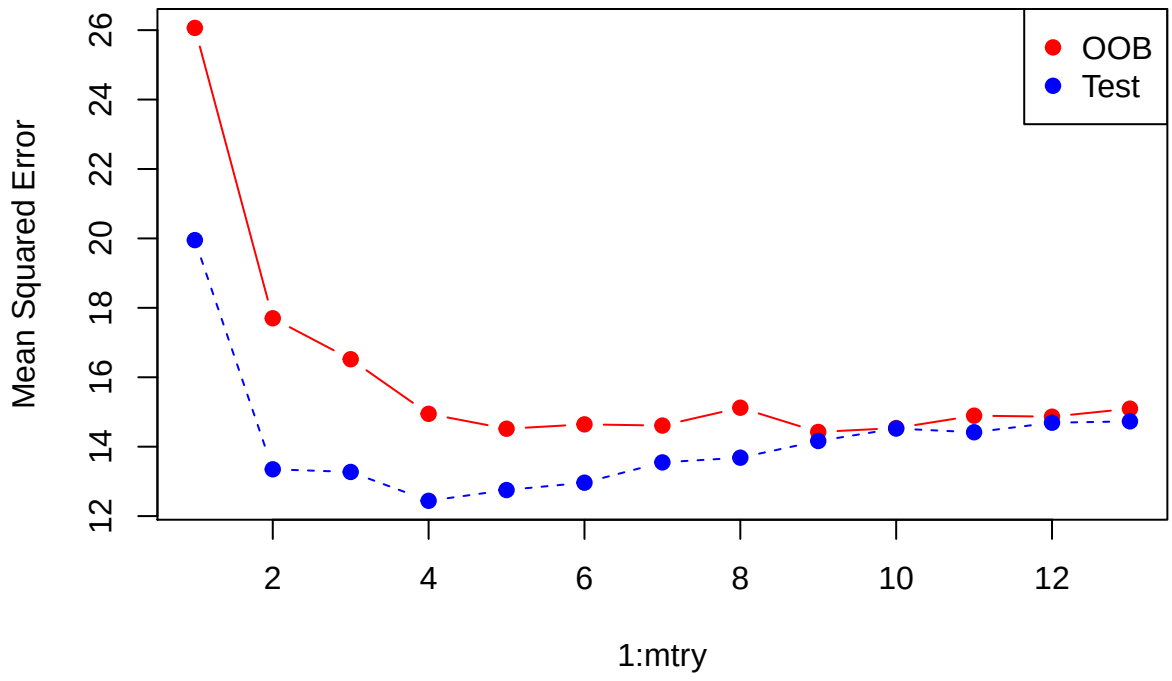
The MSR and % variance explained are based on OOB or *out-of-bag* estimates, a very clever device in random forests to get honest error estimates. The model reports that `mtry=4`, which is the number of variables

randomly chosen at each split. Since $p = 13$ here, we could try all 13 possible values of `mtry`. We will do so, record the results, and make a plot.

```
oob.err=double(13)
test.err=double(13)
for(mtry in 1:13){
  fit=randomForest(medv~.,data=Boston,subset=train,mtry=mtry,ntree=400)
  oob.err[mtry]=fit$mse[400]
  pred=predict(fit,Boston[-train,])
  test.err[mtry]=with(Boston[-train,],mean((medv-pred)^2))
  cat(mtry," ")
}
```

```
## 1  2  3  4  5  6  7  8  9  10  11  12  13
```

```
matplot(1:mtry,cbind(test.err,oob.err),pch=19,col=c("red","blue"),type="b",ylab="Mean Squared Error")
legend("topright",legend=c("OOB","Test"),pch=19,col=c("red","blue"))
```



Not too difficult! Although the test-error curve drops below the OOB curve, these are estimates based on data, and so have their own standard errors (which are typically quite large). Notice that the points at the end with `mtry=13` correspond to bagging.