California Housing: Decision Trees; Bagging and Random Forests; Boosting

Chicago Booth ML Team

Load necessary libraries

```
library(tree)
library(rpart)
library(maps)
library(gbm)
library(randomForest)
```

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

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

Let's try all this stuff on the California Housing data. That is, we'll try trees, Random Forests, and Boosting. How will they do??

We'll do a simple three set approach since we have a fairly large data set. We randomly divide the data into three sets:

- Train: 10,320 observations.
- Validation: 5,160 observations.
- Test: 5,160 observations.

We download and prepare the data.

```
download.file(
    "https://raw.githubusercontent.com/ChicagoBoothML/DATA___CaliforniaHousing/master/CaliforniaHousing.csv",
              "CaliforniaHousing.csv")
ca = read.csv("CaliforniaHousing.csv",header=TRUE)

#train, val, test
set.seed(99)
n=nrow(ca)
n1=floor(n/2)
n2=floor(n/4)
n3=n-n1-n2
ii = sample(1:n,n)
catrain=ca[ii[1:n1],]
caval = ca[ii[n1+1:n2],]
catest = ca[ii[n1+n2+1:n3],]
```

We start by fitting a regression tree to California Housing data: `logMedVal ~ longitude+latitude`.

First, we create a big tree:

```
temp = tree(logMedVal~longitude+latitude,catrain,mindev=.0001)
cat('first big tree size: \n')
```

```
## first big tree size:
```

```
print(length(unique(temp$where)))
```

```
## [1] 504
```

and then we prune it

```
caltrain.tree = prune.tree(temp,best=50)
cat('pruned tree size: \n')
```

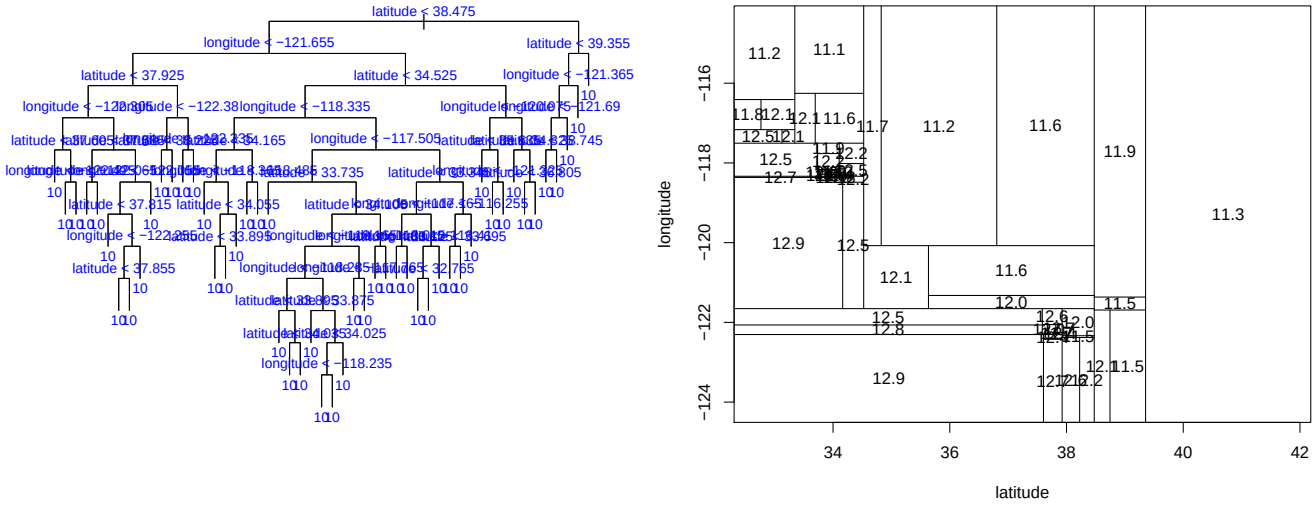
```
## pruned tree size:
```

```
print(length(unique(caltrain.tree$where)))
```

```
## [1] 51
```

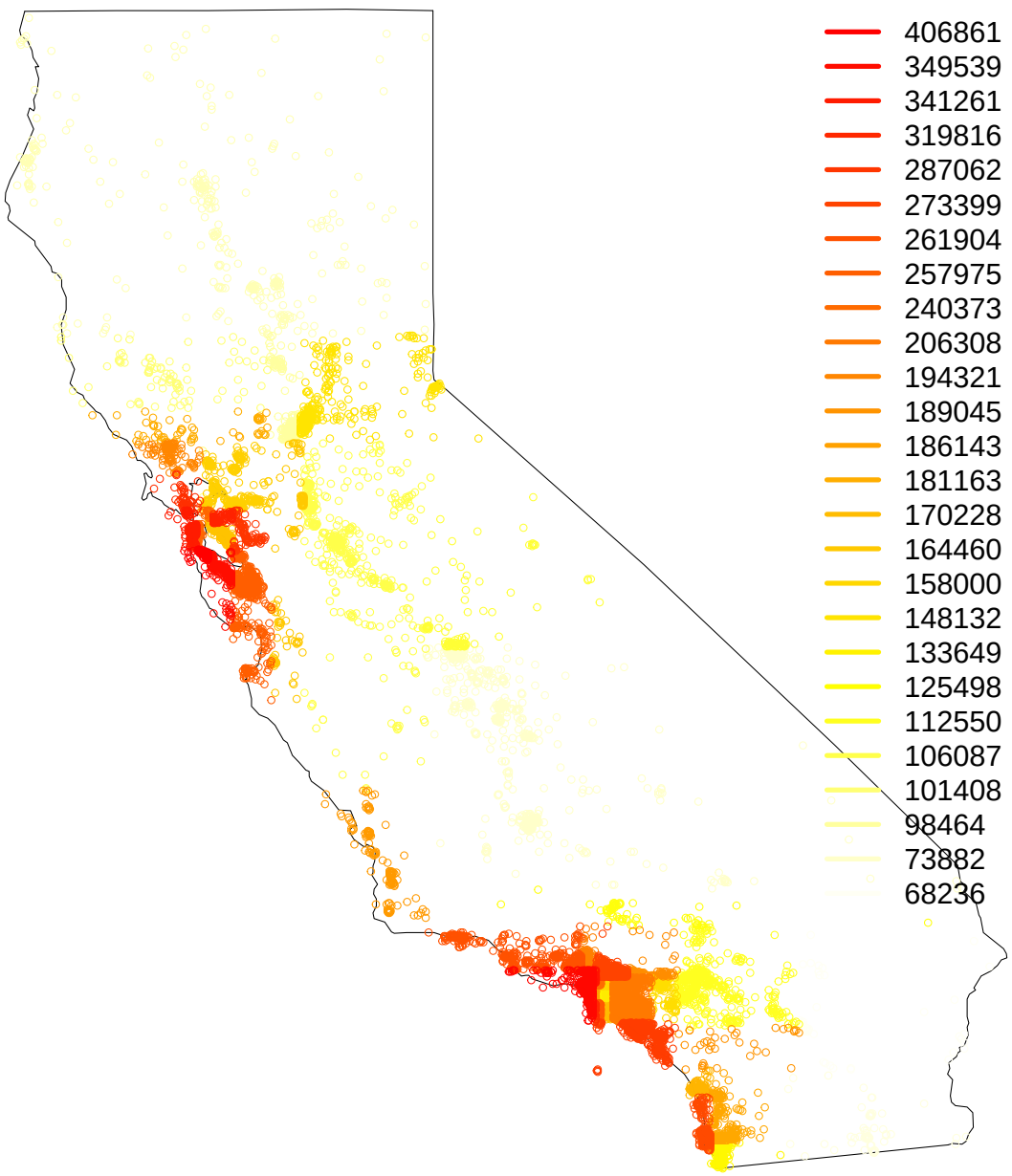
Here is a plot of the tree and the corresponding partition:

```
par(mfrow=c(1,2))
plot(caltrain.tree,type="u")
text(caltrain.tree,col="blue",label=c("yval"),cex=.8)
#plot the partition
partition.tree(caltrain.tree)
```



Finally, here is a map with the fit

```
frm = caltrain.tree$frame
wh = caltrain.tree$where
nrf = nrow(frm)
iil = frm[,"var"]=="<leaf>"
iil = (1:nrf)[iil] #indices of leaves in the frame
oo = order(frm[iil,"yval"]) #sort by yval so iil[oo[i]] give frame row of ith yval leaf

map('state', 'california')
nc=length(iil)
colv = heat.colors(nc)[nc:1]
for(i in 1:length(iil)) {
    iitemp  = (wh == iil[oo[i]]) #where refers to rows of the frame
    points(catrain$longitude[iitemp],catrain$latitude[iitemp],col=colv[i])
}
lglabs=as.character(round(exp(frm[iil[oo],"yval"]),0))
lseq = seq(from=nc,to=1,by=-2)
legend("topright",legend=lglabs[lseq],col=colv[lseq],
        cex=1.8,lty=rep(1,nc),lwd=rep(5,nc),bty="n")
```



Next, we

- Try various approaches using the Training data to fit and see how well we do out-of-sample on the Validation data set.
- After we pick an approach we like, we fit using the combined Train+Validation and then predict on the test to get a final out-of-sample measure of performance

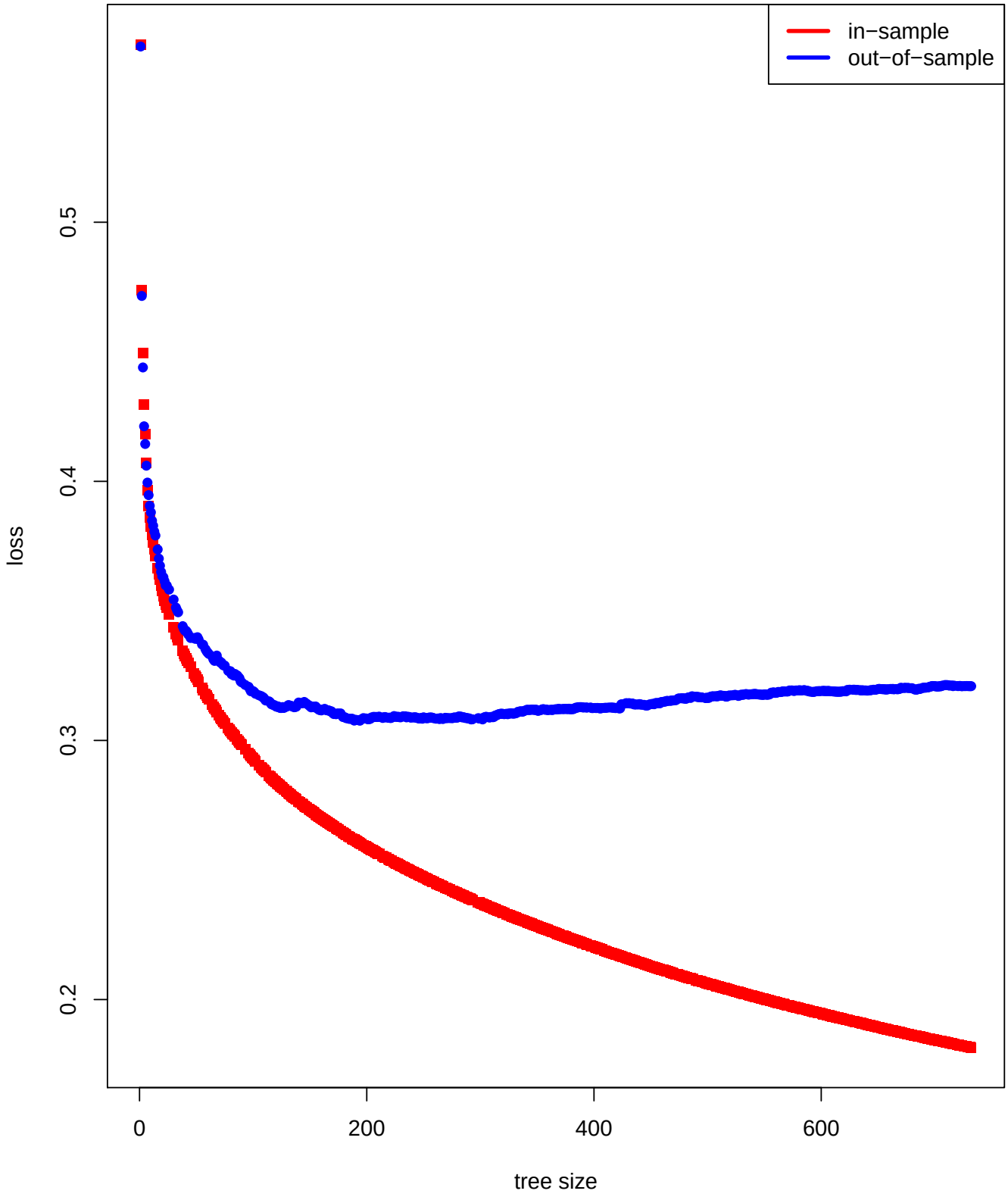
Trees:

- Fit big tree on train.
- For many cp=α, prune tree, giving trees of various sizes.
- Get in-sample loss on train.
- Get out-of-sample loss on validation.

```
#-----------------------------------------------
#get big tree
big.tree = rpart(logMedVal~.,method="anova",data=catrain,
              control=rpart.control(minsplit=5,cp=.0001))
nbig = length(unique(big.tree$where))
cat('size of big tree: ',nbig,'\n')
```

```
## size of big tree:  732
```

```
#-----------------------------------------------
#fit on train, predict on val for vector of cp.
cpvec = big.tree$cptable[,"CP"] #cp values to try
ntree = length(cpvec) #number of cp values = number of trees fit.
iltree = rep(0,ntree) #in-sample loss
oltree = rep(0,ntree) #out-of-sample loss
sztree = rep(0,ntree) #size of each tree
for(i in 1:ntree) {
   temptree = prune(big.tree,cp=cpvec[i])
   sztree[i] = length(unique(temptree$where))
   iltree[i] = sum((catrain$logMedVal-predict(temptree))^2)
   ofit = predict(temptree,caval)
   oltree[i] = sum((caval$logMedVal-ofit)^2)
}
oltree=sqrt(oltree/nrow(caval)); iltree = sqrt(iltree/nrow(catrain))

#-----------------------------------------------
#plot losses
rgl = range(c(iltree,oltree))
plot(range(sztree),rgl,type='n',xlab='tree size',ylab='loss')
points(sztree,iltree,pch=15,col='red')
points(sztree,oltree,pch=16,col='blue')
legend("topright",legend=c('in-sample','out-of-sample'),lwd=3,col=c('red','blue'))
```




```
#write validation predictions
iitree = which.min(oltree)
thetree = prune(big.tree,cp=cpvec[iitree])
```

```
thetreepred = predict(thetree,caval)
```

We get the smallest out-of-sample loss (.307) at a tree size of 194.

Boosting:

Let's try:

- maximum depths of 4 or 10.
- 1,000 or 5,000 trees.
- $\lambda = .2$ or .001.

Here:

- *olb*: out-of-sample loss
- *ilb*: in-sample loss

```
idv = c(4, 10)
ntv = c(1000, 5000)
lamv=c(.001, .2)
parmb = expand.grid(idv,ntv,lamv)
colnames(parmb) = c('tdepth','ntree','lam')
print(parmb)
```

```
##   tdepth ntree   lam
## 1      4  1000 0.001
## 2     10  1000 0.001
## 3      4  5000 0.001
## 4     10  5000 0.001
## 5      4  1000 0.200
## 6     10  1000 0.200
## 7      4  5000 0.200
## 8     10  5000 0.200
```

```
nset = nrow(parmb)
olb = rep(0,nset)
ilb = rep(0,nset)
bfitv = vector('list',nset)
for(i in 1:nset) {
   tempboost = gbm(logMedVal~.,data=catrain,distribution='gaussian',
               interaction.depth=parmb[i,1],n.trees=parmb[i,2],shrinkage=parmb[i,3])
   ifit = predict(tempboost,n.trees=parmb[i,2])
   ofit=predict(tempboost,newdata=caval,n.trees=parmb[i,2])
   olb[i] = sum((caval$logMedVal-ofit)^2)
   ilb[i] = sum((catrain$logMedVal-ifit)^2)
   bfitv[[i]]=tempboost
}
ilb = round(sqrt(ilb/nrow(catrain)),3); olb = round(sqrt(olb/nrow(caval)),3)

print(cbind(parmb,olb,ilb))
```

```
##   tdepth ntree   lam   olb   ilb
## 1      4  1000 0.001 0.414 0.416
## 2     10  1000 0.001 0.378 0.380
## 3      4  5000 0.001 0.279 0.282
## 4     10  5000 0.001 0.252 0.250
## 5      4  1000 0.200 0.231 0.164
## 6     10  1000 0.200 0.230 0.097
## 7      4  5000 0.200 0.232 0.079
## 8     10  5000 0.200 0.235 0.013
```

```
#write validation predictions
iib=which.min(olb)
```

```
theb = bfitv[[iib]]
thebpred = predict(theb,newdata=caval,n.trees=parmb[iib,2])
```

Minimum loss of .231 is quite a bit better than trees!

Random Forests:

Let's try:

- m equal 3 and 9 (Bagging).
- 100 or 500 trees.

Here:

- *olrf*: out-of-sample loss
- *ilrf*: in-sample loss

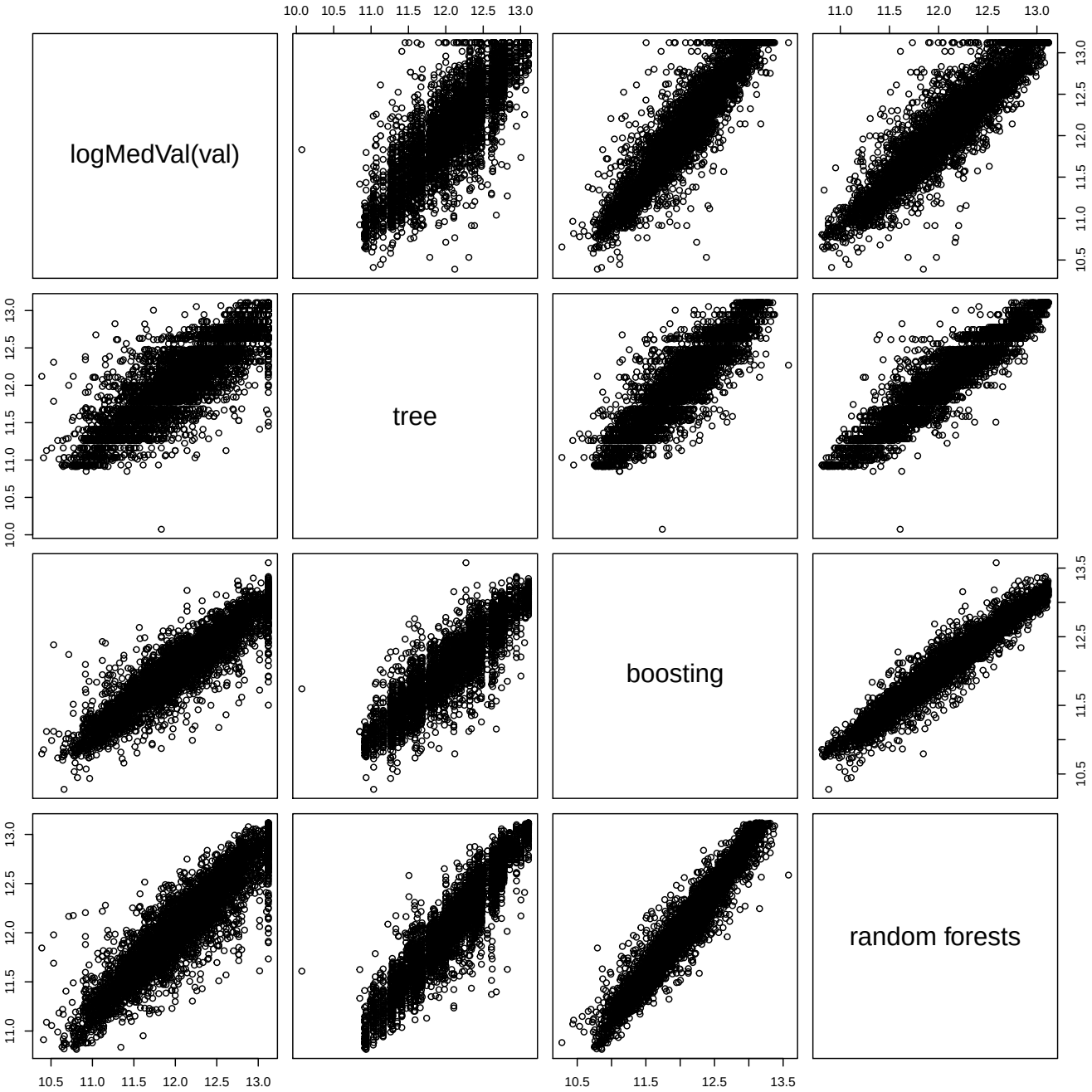
```
p=ncol(catrain)-1
mtryv = c(p,sqrt(p))
ntreev = c(100,500)
parmrf = expand.grid(mtryv,ntreev)
colnames(parmrf)=c('mtry','ntree')
nset = nrow(parmrf)
olrf = rep(0,nset)
ilrf = rep(0,nset)
rffitv = vector('list',nset)
for(i in 1:nset) {
   temprf = randomForest(logMedVal~.,data=catrain,mtry=parmrf[i,1],ntree=parmrf[i,2])
   ifit = predict(temprf)
   ofit=predict(temprf,newdata=caval)
   olrf[i] = sum((caval$logMedVal-ofit)^2)
   ilrf[i] = sum((catrain$logMedVal-ifit)^2)
   rffitv[[i]]=temprf
}
ilrf = round(sqrt(ilrf/nrow(catrain)),3); olrf = round(sqrt(olrf/nrow(caval)),3)
#---------------------------------------
#print losses
print(cbind(parmrf,olrf,ilrf))
```

```
##   mtry ntree  olrf  ilrf
## 1    9   100 0.243 0.258
## 2    3   100 0.235 0.248
## 3    9   500 0.243 0.254
## 4    3   500 0.234 0.246
```

```
#write validation predictions
iirf=which.min(olrf)
therf = rffitv[[iirf]]
therfpred=predict(therf,newdata=caval)
```

Let's compare the predictions on the Validation data with the best performing of each of the three methods.

```
predmat=cbind(caval$logMedVal,thetreepred,thebpred,therfpred)
colnames(predmat) = c('logMedVal(val)','tree','boosting','random forests')
pairs(predmat)
```



It does look like Boosting and Random Forests are a lot better than a single tree.

The fits from Boosting and Random Forests are not too different (this is not always the case).

Test Set Performance, Boosting

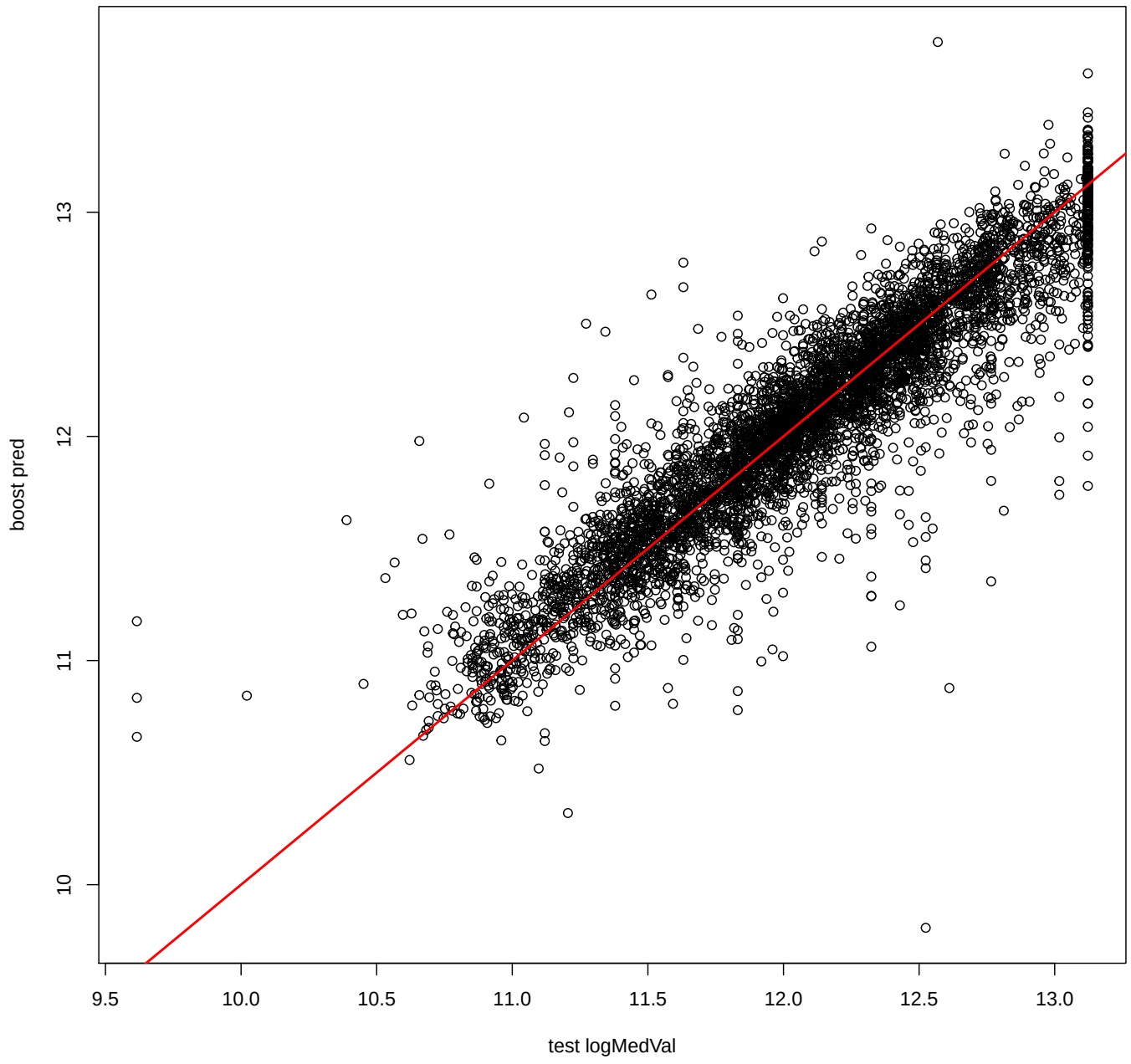
Let's fit Boosting using depth=4, 5,000 trees, and shrinkage = λ=.2 on the combined train and validation data sets.

```
#fit on train+val
set.seed(1)
catrainval = rbind(catrain,caval)
ntrees=5000
finb = gbm(logMedVal~.,data=catrainval,distribution='gaussian',
              interaction.depth=4,n.trees=ntrees,shrinkage=.2)
finbpred=predict(finb,newdata=catest,n.trees=ntrees)
```

```
finbrmse = sqrt(sum((catest$logMedVal-finbpred)^2)/nrow(catest))
cat('finb rmse: ', finbrmse,'\n')
```

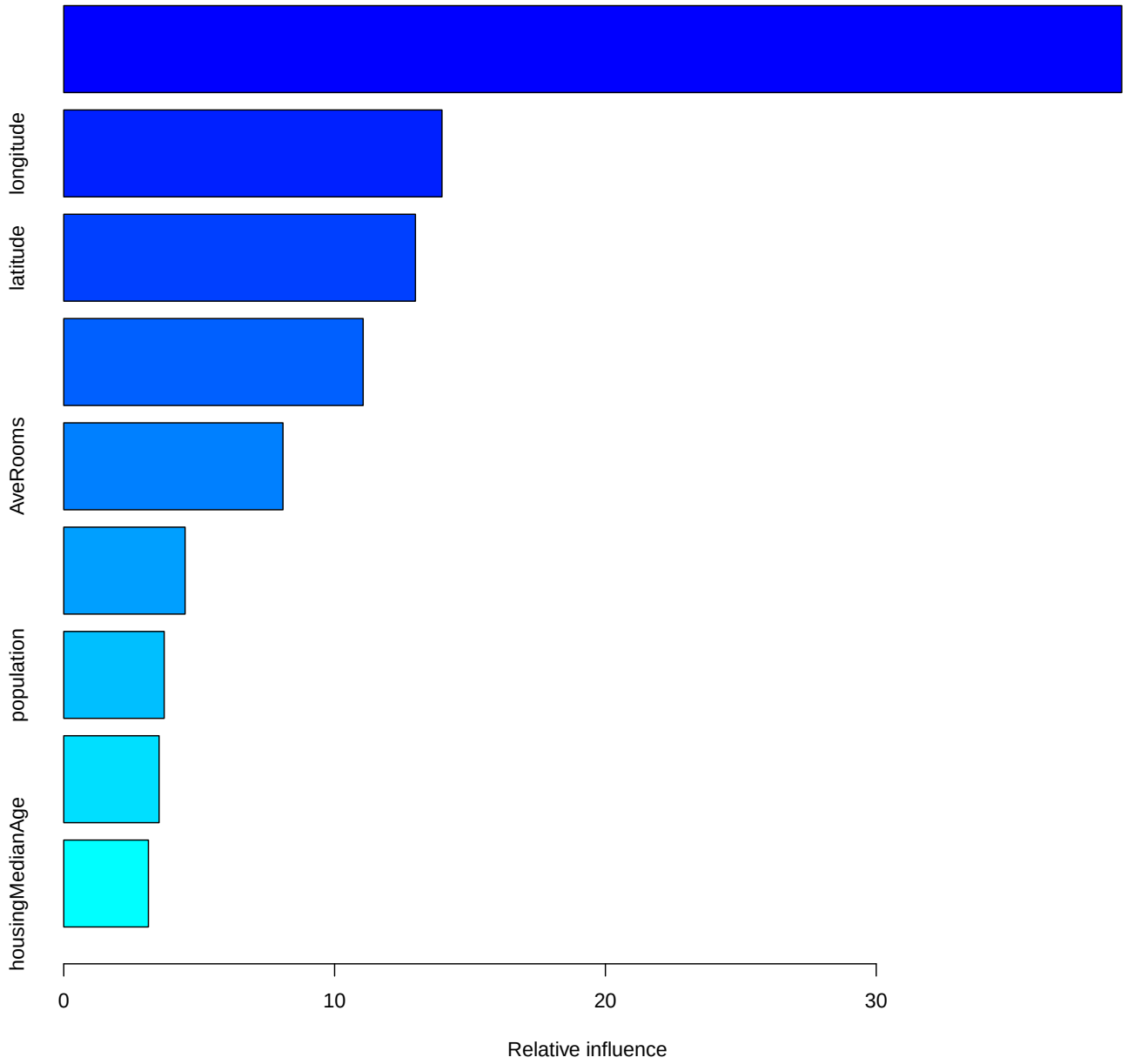
```
## finb rmse:  0.231
```

```
#plot y vs yhat for test data and compute rmse on test.
plot(catest$logMedVal,finbpred,xlab='test logMedVal',ylab='boost pred')
abline(0,1,col='red',lwd=2)
```



Plot variable importance

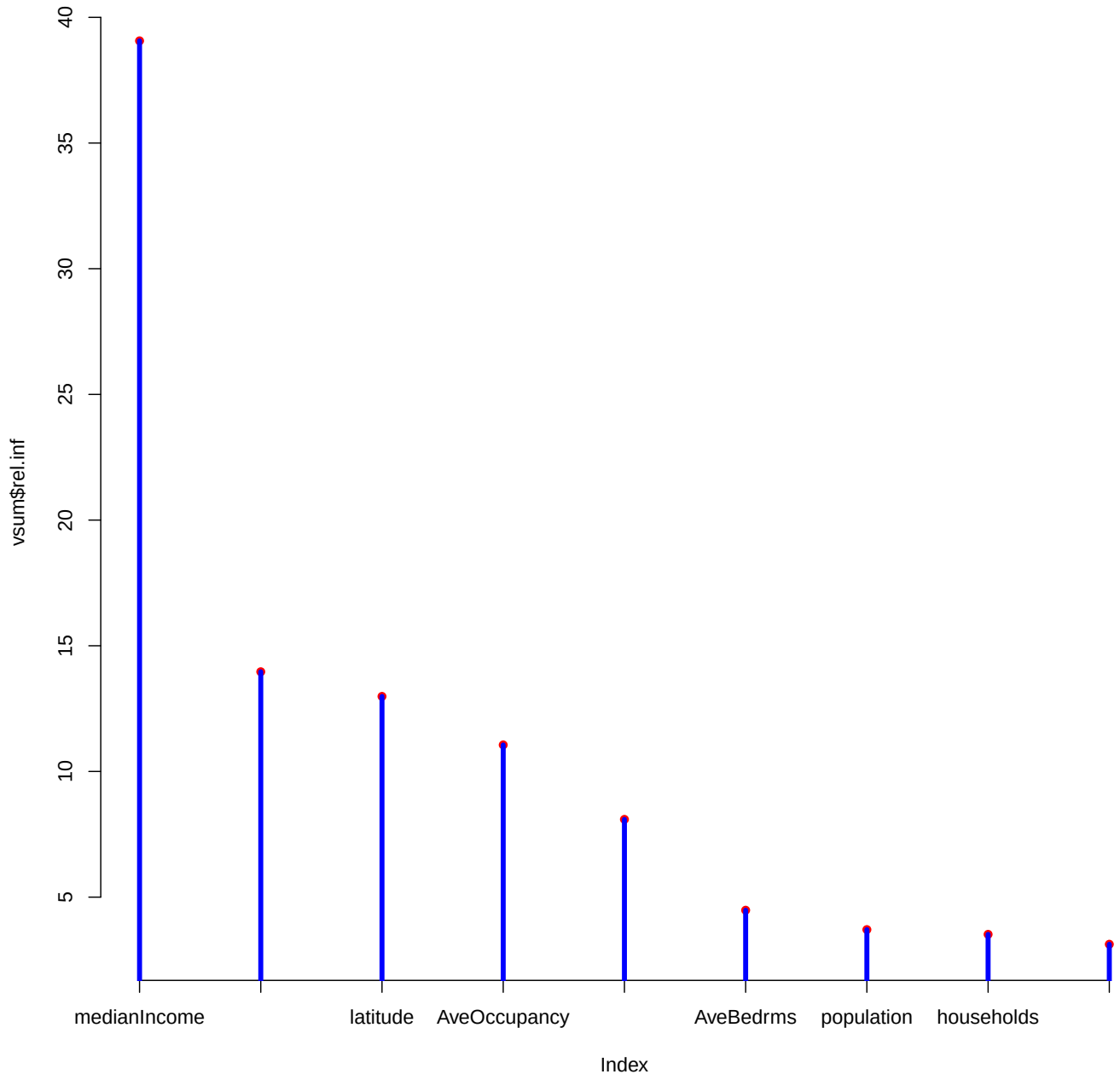
```
p=ncol(catrain)-1 #want number of variables for later

vsum=summary(finb) #this will have the variable importance info
```




```
#write variable importance table
print(vsum)
```

```
##                                 var rel.inf
## medianIncome         medianIncome   39.07
## longitude               longitude   13.97
## latitude                 latitude   12.99
## AveOccupancy         AveOccupancy   11.06
## AveRooms                 AveRooms    8.09
## AveBedrms               AveBedrms    4.48
## population             population    3.71
## households             households    3.52
## housingMedianAge housingMedianAge    3.13
```

```r
#plot variable importance
#the package does this automatically, but I did not like the plot
plot(vsum$rel.inf,axes=F,pch=16,col='red')
axis(1,labels=vsum$var,at=1:p)
axis(2)
for(i in 1:p) lines(c(i,i),c(0,vsum$rel.inf[i]),lwd=4,col='blue')
```



`medianIncome` is by far the most important variable. After that, it is location - makes sense.

Test Set Performance, Random Forests

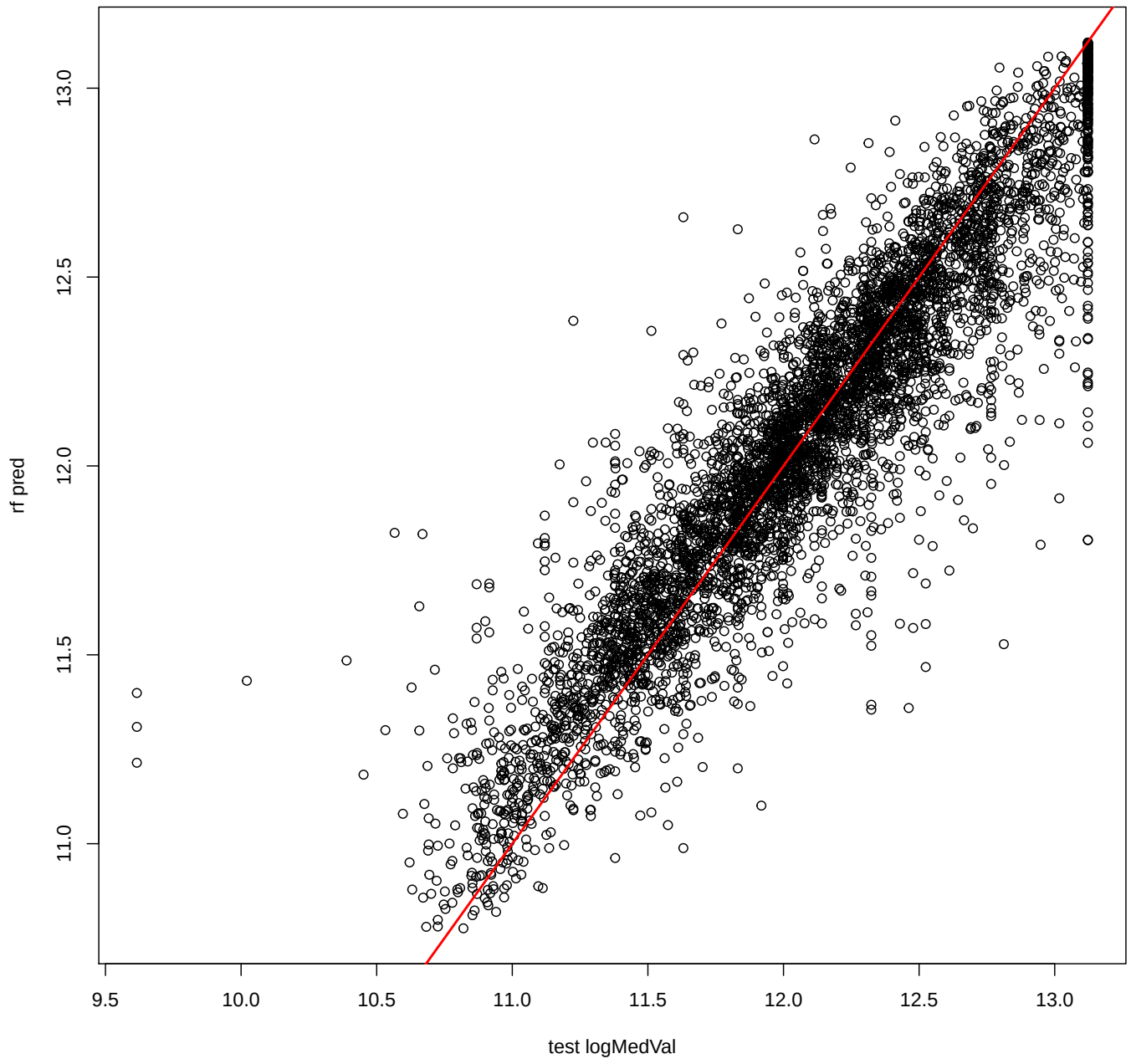
Let's fit Random Forests using m=3 and 500 trees on the combined train and validation data sets.

Let's see how the predictions compare to the test values.

```
set.seed(1)
catrainval = rbind(catrain,caval)
finrf = randomForest(logMedVal~.,data=catrainval,mtry=3,ntree=500)
finrfpred=predict(finrf,newdata=catest)

finrfrmse = sqrt(sum((catest$logMedVal-finrfpred)^2)/nrow(catest))
cat('finrfrmse: ',finrfrmse,'\n')
```

```
## finrfrmse:  0.228
```

```
plot(catest$logMedVal,finrfpred,xlab='test logMedVal',ylab='rf pred')
abline(0,1,col='red',lwd=2)
```



Plot variable importance

```
varImpPlot(finrf)
```