452 Project2

Zhaojin Zhu

11/19/2021

```
# big data sets
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.2
```

```
## Loaded gbm 2.1.8
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.6.2
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.6.2
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```
library(permute)
library(pls)
```

```
## Warning: package 'pls' was built under R version 3.6.2
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:caret':
##
##     R2
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
train_data = read.csv("Data2021_final.csv")
test_data = read.csv("Data2021test_final_noY.csv")
```

GAM

```
#First I try to use GAM on all variables
gam.all <- gam(data=train_data,
            formula=Y ~s(X1)+s(X2) + s(X3) + s(X4) + s(X5) + s(X6) + s(X7) + s(X8) + s(X9) + s(X
10) + s(X11) + s(X12) + s(X13) + s(X14) + s(X15))
summary(gam.all)
```

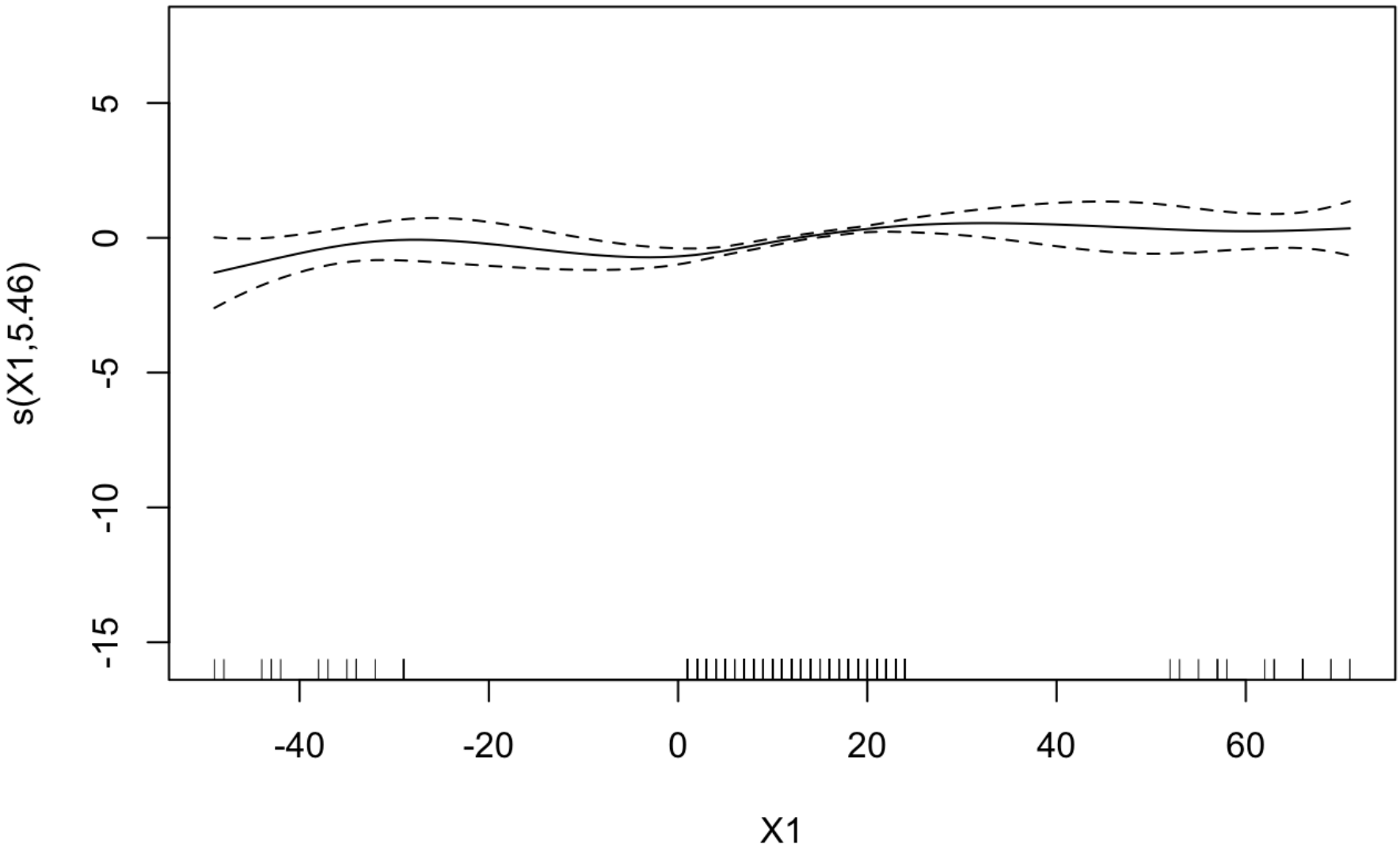
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## Y ~ s(X1) + s(X2) + s(X3) + s(X4) + s(X5) + s(X6) + s(X7) + s(X8) +
##     s(X9) + s(X10) + s(X11) + s(X12) + s(X13) + s(X14) + s(X15)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.92071    0.04659   277.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df     F  p-value
## s(X1)  5.457  6.447 7.940 1.26e-08 ***
## s(X2)  5.826  6.701 4.686 6.21e-05 ***
## s(X3)  1.006  1.013 0.002 0.966578
## s(X4)  8.334  8.809 3.189 0.000671 ***
## s(X5)  1.001  1.001 0.042 0.838291
## s(X6)  1.434  1.723 1.828 0.245688
## s(X7)  2.363  2.622 0.507 0.575202
## s(X8)  1.356  1.634 0.165 0.751404
## s(X9)  1.000  1.000 3.966 0.046814 *
## s(X10) 3.332  3.883 1.564 0.168146
## s(X11) 1.000  1.000 4.078 0.043807 *
## s(X12) 7.594  8.229 4.265 4.88e-05 ***
```

```
## s(X13) 1.869  2.081 2.516 0.074819 .
## s(X14) 1.000  1.000 0.808 0.368879
## s(X15) 2.807  3.306 1.935 0.111758
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.201   Deviance explained = 24.9%
## GCV = 1.7353  Scale est. = 1.628     n = 750
```

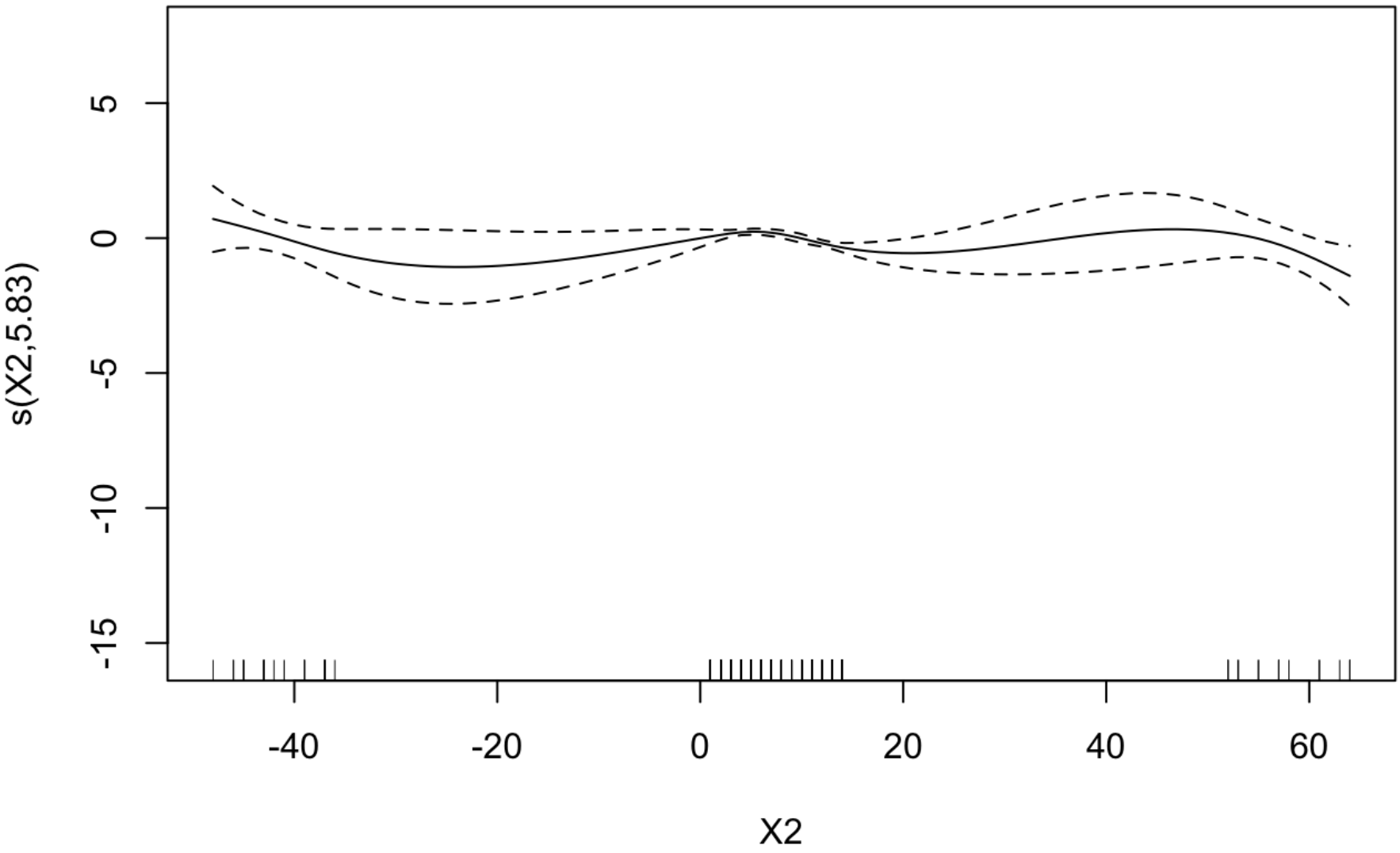
Analysis:

Only the X1, X2, X4, X9, X11 and X12 looks important. The variables X1 appears to have the most non linear relationship with Y, followed by X2, X12, X4, X11 and X9.

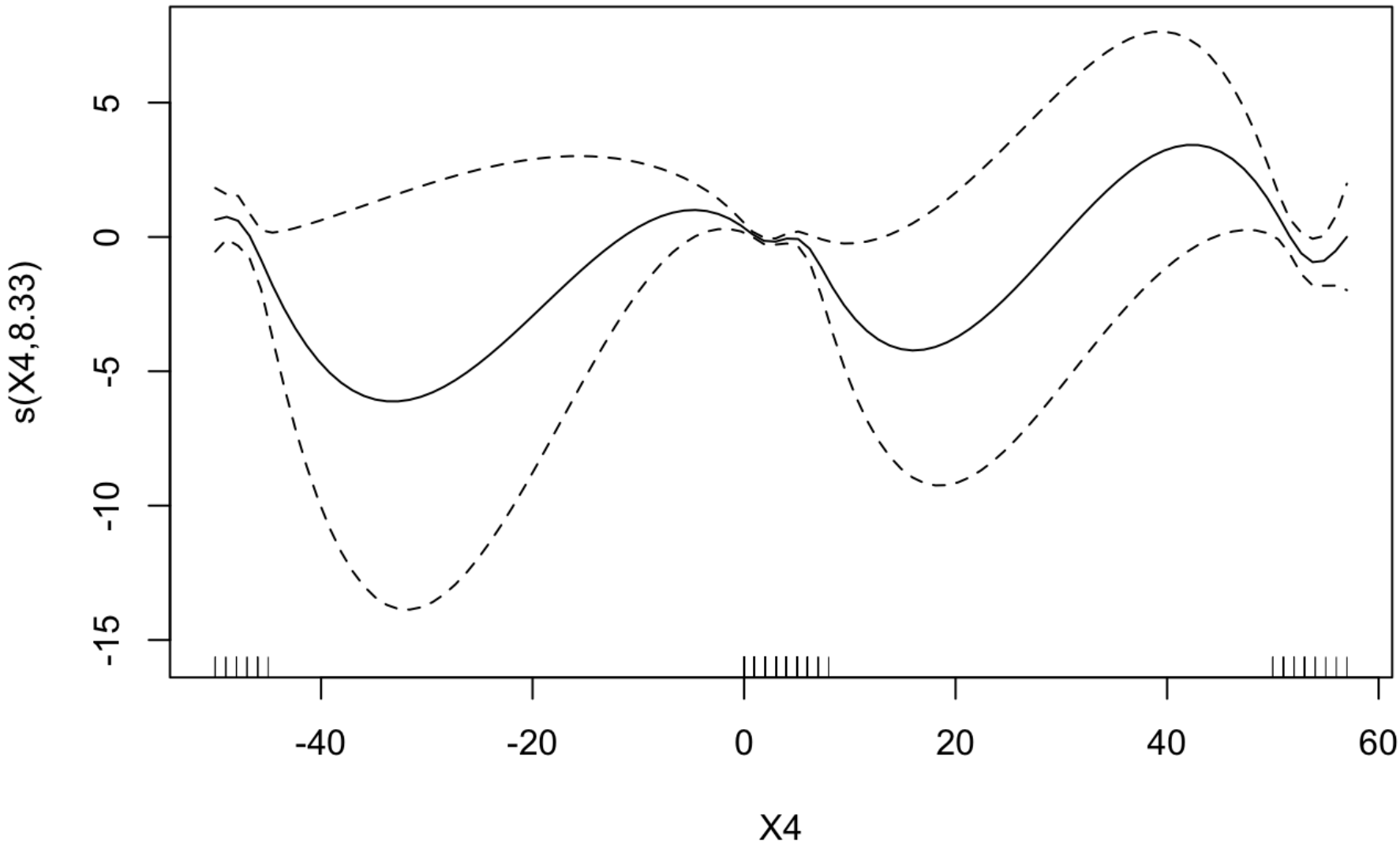
```
plot(gam.all, select = 1) # nonlinear, slightly increasing
```



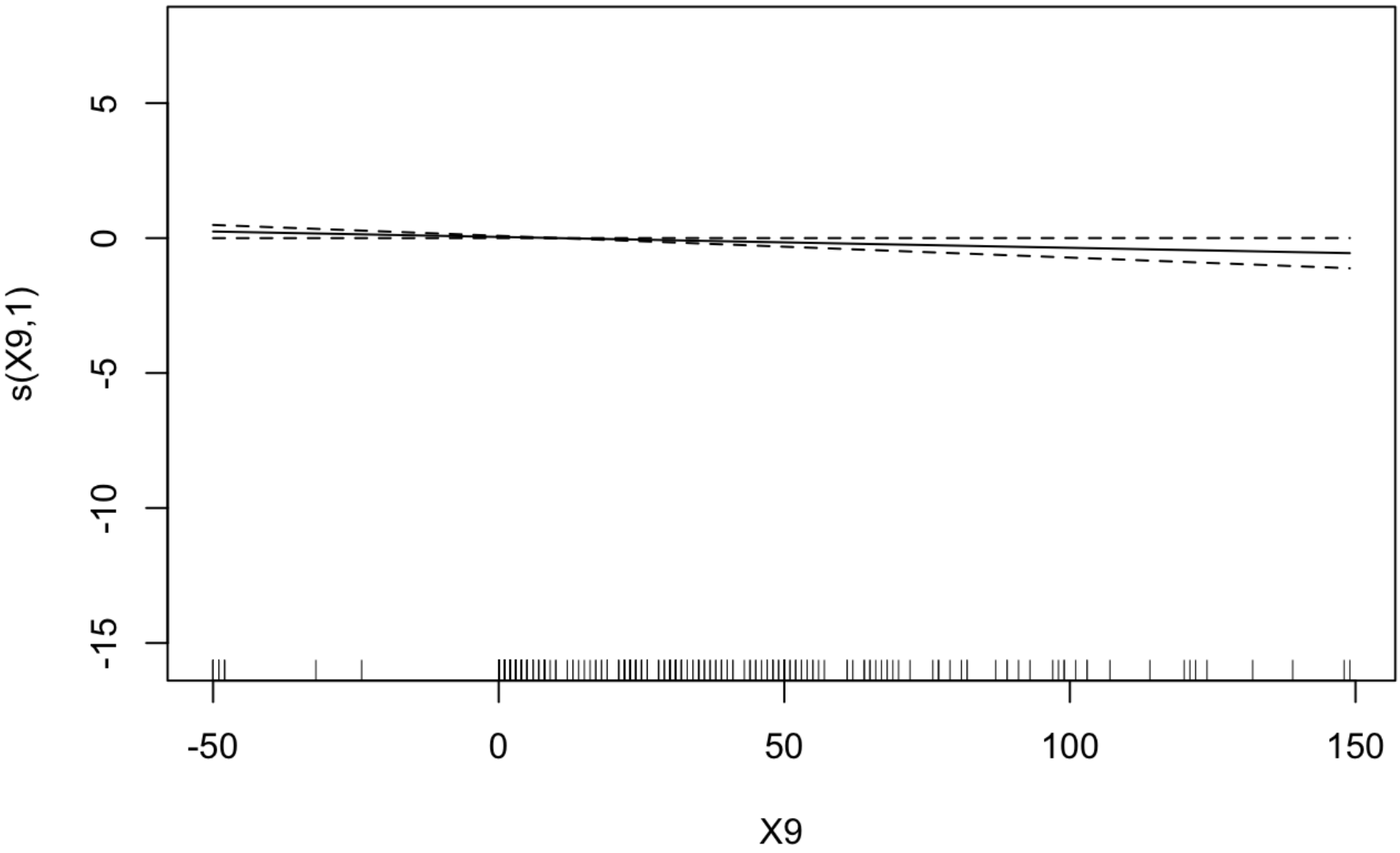

```
plot(gam.all, select = 2) # nonlinear
```



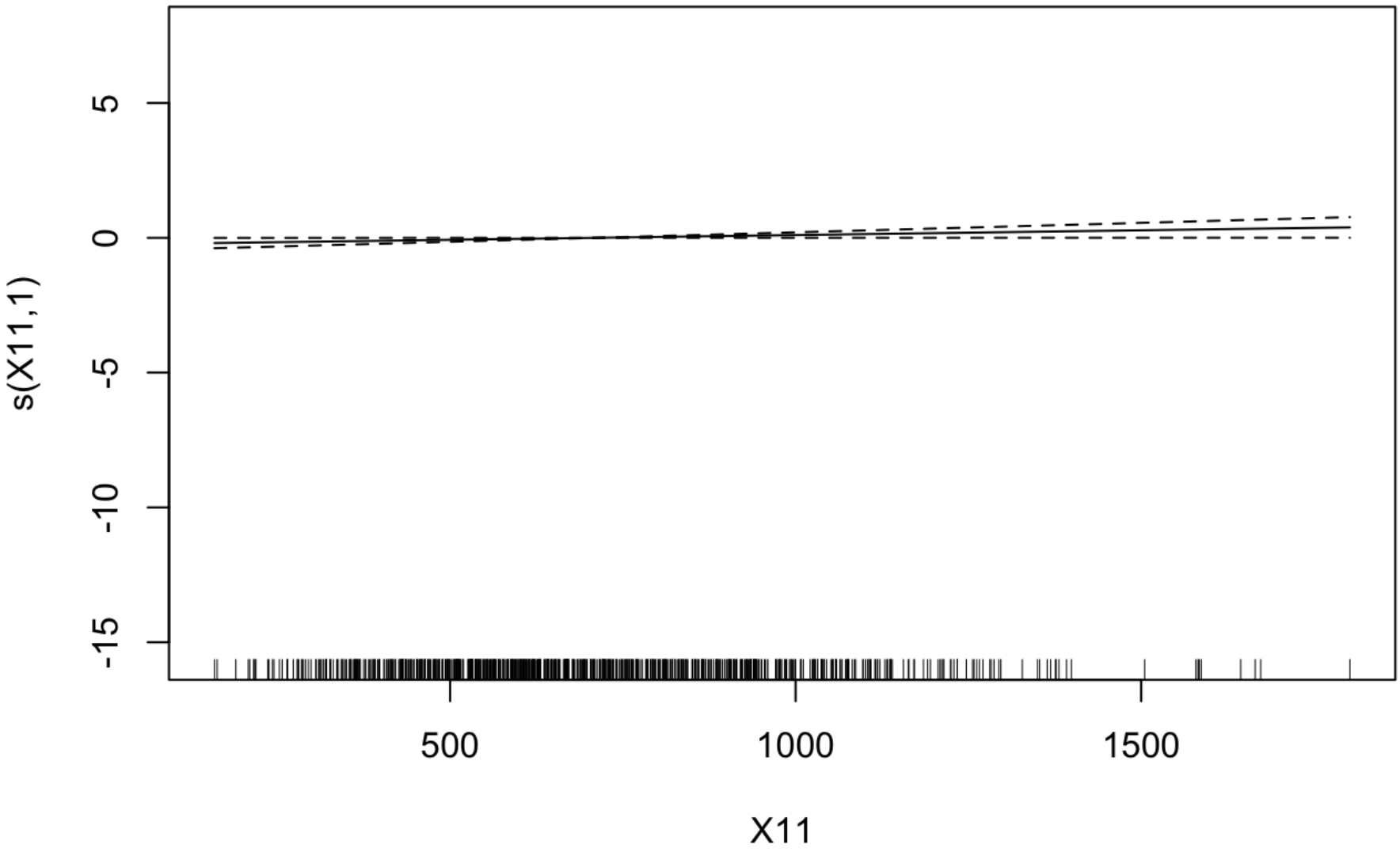

```
plot(gam.all, select = 4) # clearly nonlinear
```



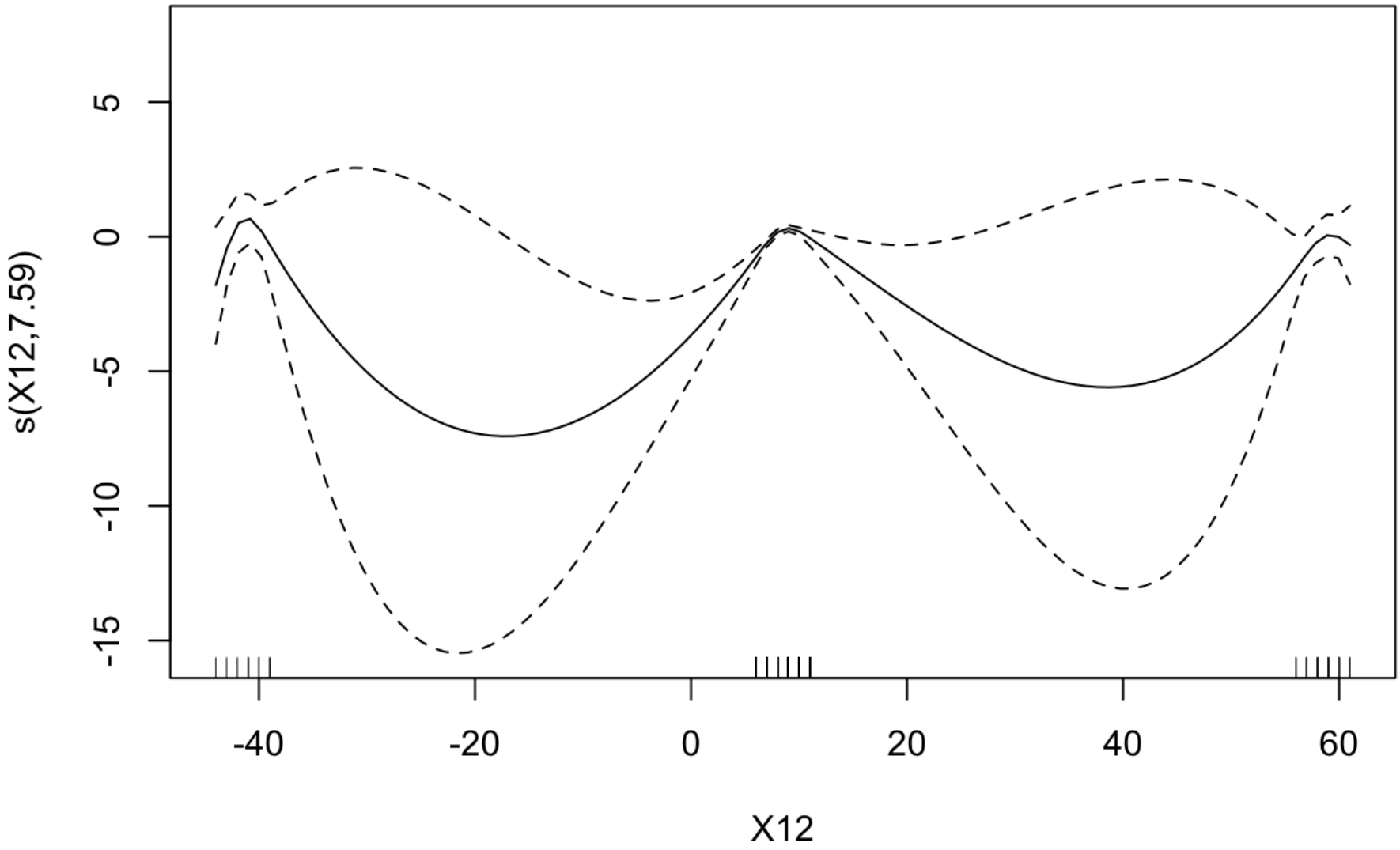

```
plot(gam.all, select = 9) # pretty much linear
```




```
plot(gam.all, select = 11) # pretty much linear
```




```
plot(gam.all, select = 12) # clearly nonlinear
```



Now I want to add GAM on all variables to the 10-fold CV comparison that has beeen used for LASSO, Ridge and other methods.

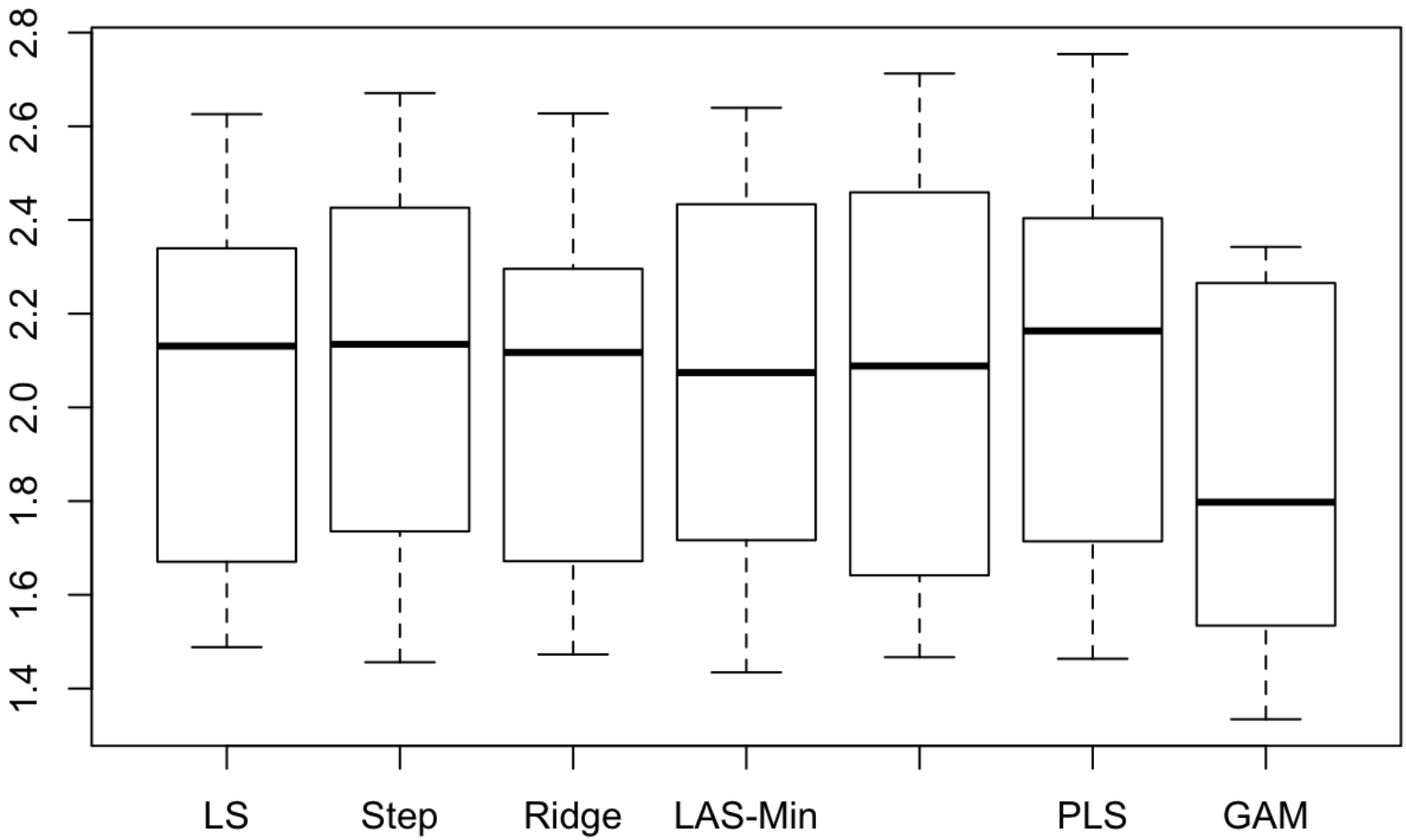
```r
library(MASS)
library(glmnet)
library(pls)
library(mgcv)
#library(kableExtra)

### Create function to compute MSPEs
get.MSPE = function(Y, Y.hat){
  return(mean((Y - Y.hat)^2))
}
### Create function which constructs folds for CV
### n is the number of observations, K is the number of folds
get.folds = function(n, K) {
  ### Get the appropriate number of fold labels
  n.fold = ceiling(n / K) # Number of observations per fold (rounded up)
  fold.ids.raw = rep(1:K, times = n.fold)
  fold.ids = fold.ids.raw[1:n]
  ### Shuffle the fold labels
  folds.rand = fold.ids[sample.int(n)]
  return(folds.rand)
}

K = 10 #number of folds

#Container for CV MSPES
all.models = c("LS", "Step", "Ridge", "LAS-Min", "LAS-1se", "PLS", "GAM")
CV.MSPEs = array(0, dim = c(length(all.models),K))
rownames(CV.MSPEs) = all.models
```

```
colnames(CV.MSPEs) = 1:K

### Construct candidate lambda values (outside loop to save time)
lambda.vals = seq(from = 0, to = 100, by = 0.05)

### Get CV fold labels
n = nrow(train_data)
folds = get.folds(n, K)

### Perform cross-validation
for (i in 1:K){
  ### Get training and validation sets
  # data.train = na.omit(AQ[folds != i, ]) # data.valid = na.omit(AQ[folds == i, ])
  data.train = train_data[folds != i, ]
  data.valid = train_data[folds == i, ]
  Y.train = data.train$Y
  Y.valid = data.valid$Y
  ### We need the data matrix to have an intercept for ridge, and to not have an intercept
  mat.train.int = model.matrix(Y ~ ., data = data.train)
  mat.train = mat.train.int[,-1]
  mat.valid.int = model.matrix(Y ~ ., data = data.valid)
  mat.valid = mat.valid.int[,-1]

  ########## ### LS ### ##########
  fit.ls = lm(Y ~ ., data = data.train)
  pred.ls = predict(fit.ls, data.valid)
  MSPE.ls = get.MSPE(Y.valid, pred.ls)
  CV.MSPEs["LS", i] = MSPE.ls
```

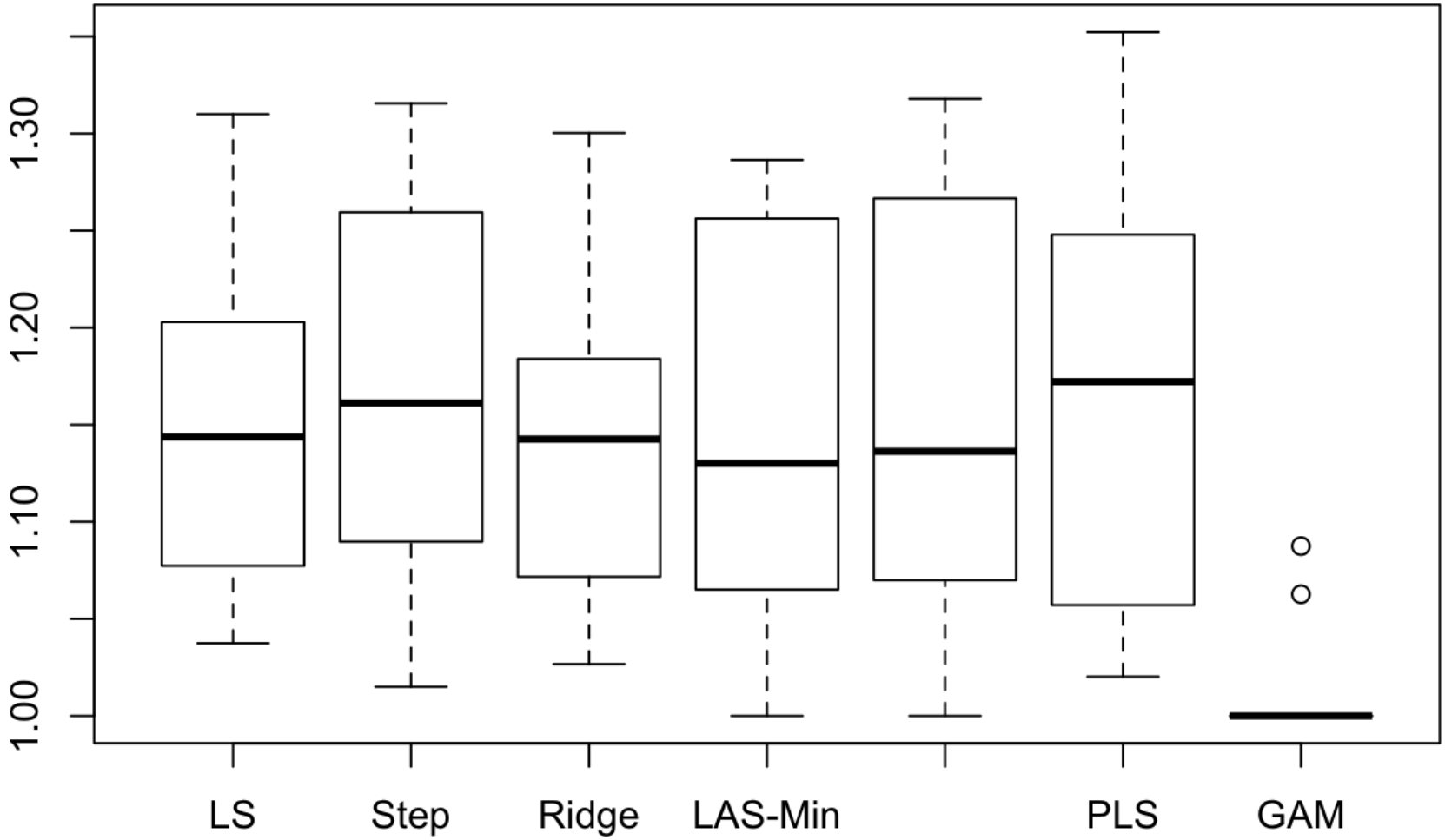
```
############ ### Step ### ############
fit.start = lm(Y ~ 1, data = data.train)
fit.step = step(fit.start, list(upper = fit.ls), trace = 0)
pred.step = predict(fit.step, data.valid)
MSPE.step = get.MSPE(Y.valid, pred.step)
CV.MSPEs["Step", i] = MSPE.step

############ ### Ridge ### ############
### Fit ridge regression
### We already definted lambda.vals. No need to re-invent the wheel
fit.ridge = lm.ridge(Y ~ ., lambda = lambda.vals, data = data.train)
### Get optimal lambda value
ind.min.GCV = which.min(fit.ridge$GCV)
lambda.min = lambda.vals[ind.min.GCV]
### Get coefficients for optimal model
all.coefs.ridge = coef(fit.ridge)
coef.min.ridge = all.coefs.ridge[ind.min.GCV,]
### Get predictions and MSPE on validation set
pred.ridge = mat.valid.int %*% coef.min.ridge
pred.ridge = as.numeric(pred.ridge)
MSPE.ridge = get.MSPE(Y.valid, pred.ridge)
CV.MSPEs["Ridge", i] = MSPE.ridge

############ ### LASSO ### ############
### Fit model
fit.LASSO = cv.glmnet(mat.train, Y.train)
### Get optimal lambda values
lambda.min = fit.LASSO$lambda.min
lambda.1se = fit.LASSO$lambda.1se
```

```
  ### Get predictions
  pred.min = predict(fit.LASSO, mat.valid, lambda.min)
  pred.1se = predict(fit.LASSO, mat.valid, lambda.1se)
  ### Get and store MSPEs
  MSPE.min = get.MSPE(Y.valid, pred.min)
  MSPE.1se = get.MSPE(Y.valid, pred.1se)
  CV.MSPEs["LAS-Min", i] = MSPE.min
  CV.MSPEs["LAS-1se", i] = MSPE.1se

  ### Partial Least Squares ###
  ### Fit PLS
  fit.pls = plsr(Y ~ ., data = data.train, validation = "CV",
    segments = 10)
  ### Get optimal number of folds
  CV.pls = fit.pls$validation
  PRESS.pls = CV.pls$PRESS
  n.comps = which.min(PRESS.pls)
  ### Get predictions and MSPE
  pred.pls = predict(fit.pls, data.valid, ncomp = n.comps)
  MSPE.pls = get.MSPE(Y.valid, pred.pls)
  CV.MSPEs["PLS", i] = MSPE.pls

  ########## ### GAM ### ##########
  ### Fit model
  fit.gam = gam(Y ~ s(X1)+s(X2) + s(X3) + s(X4) + s(X5) + s(X6) + s(X7) + s(X8) + s(X9) + s(X10)
+ s(X11) + s(X12) + s(X13) + s(X14) + s(X15) , data = data.train)
  ### Get predictions and MSPE
  pred.gam = predict(fit.gam, data.valid)
  MSPE.gam = get.MSPE(Y.valid, pred.gam)
```

```
  CV.MSPEs["GAM", i] = MSPE.gam
}

### Store MSPEs
write.csv(CV.MSPEs, "/Users/janezhu/Desktop/STAT452/MSPEs-L11.csv", row.names = T)
### Get full-data MSPEs
full.MSPEs = apply(CV.MSPEs, 1, mean)



### Get full-data MSPEs
full.MSPEs = apply(CV.MSPEs, 1, mean)

### Create table of all MSPEs and round nicely
MSPE.table = cbind(CV.MSPEs, full.MSPEs)
colnames(MSPE.table)[11] = "Full"
MSPE.table = round(MSPE.table)



# MSPE Boxplot
plot.MSPEs = t(CV.MSPEs)
boxplot(plot.MSPEs)
```


1.4
1.6
1.8
2.0
2.2
2.4
2.6
2.8
LS
Step
Ridge
LAS-Min
PLS
GAM

```
### Compute RMSPEs
plot.RMSPEs = apply(CV.MSPEs, 2, function(W){ best = min(W)
return(W/best)
})
plot.RMSPEs = t(plot.RMSPEs)
### RMSPE Boxplot
boxplot(plot.RMSPEs)
```


1.00
1.10
1.20
1.30
LS
Step
Ridge
LAS-Min
PLS
GAM

```
#Boxplots of MSPEs and RMSPEs are given. We can see that GAM performs very well relative to the
 other models, both on MSPE and RMSPE boxplots. The reason that GAM does this well is that it mo
dels nonlinear relationships between variables, and from previous plots we can see that the data
sets clearly does not have a linear relationships.
```

Random forest

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
##### OOB error



fit.rf.1 = randomForest(Y ~ ., data = train_data, importance = T)
### We can get out-of-bag (OOB) error directly from the predict() function.
### Specifically, if we don't include a new dataset, R gives the OOB predictions
### on the training set.
#Predicted value for any x is the average mean from that x's terminal node in each of the trees
 in the forest
OOB.pred.1 = predict(fit.rf.1) #OOB error
(OOB.MSPE.1 = get.MSPE(train_data$Y, OOB.pred.1))
```

```
## [1] 1.740147
```

```
cat("The OOB error is :", OOB.MSPE.1,"\n")
```

```
## The OOB error is : 1.740147
```

Most important variable is X12, then X1 and X15. If we leave variable X13 out of the picture, then the meam square error will only by increase by 8%. Thus, I think only X12, X1 and X10 seems important.

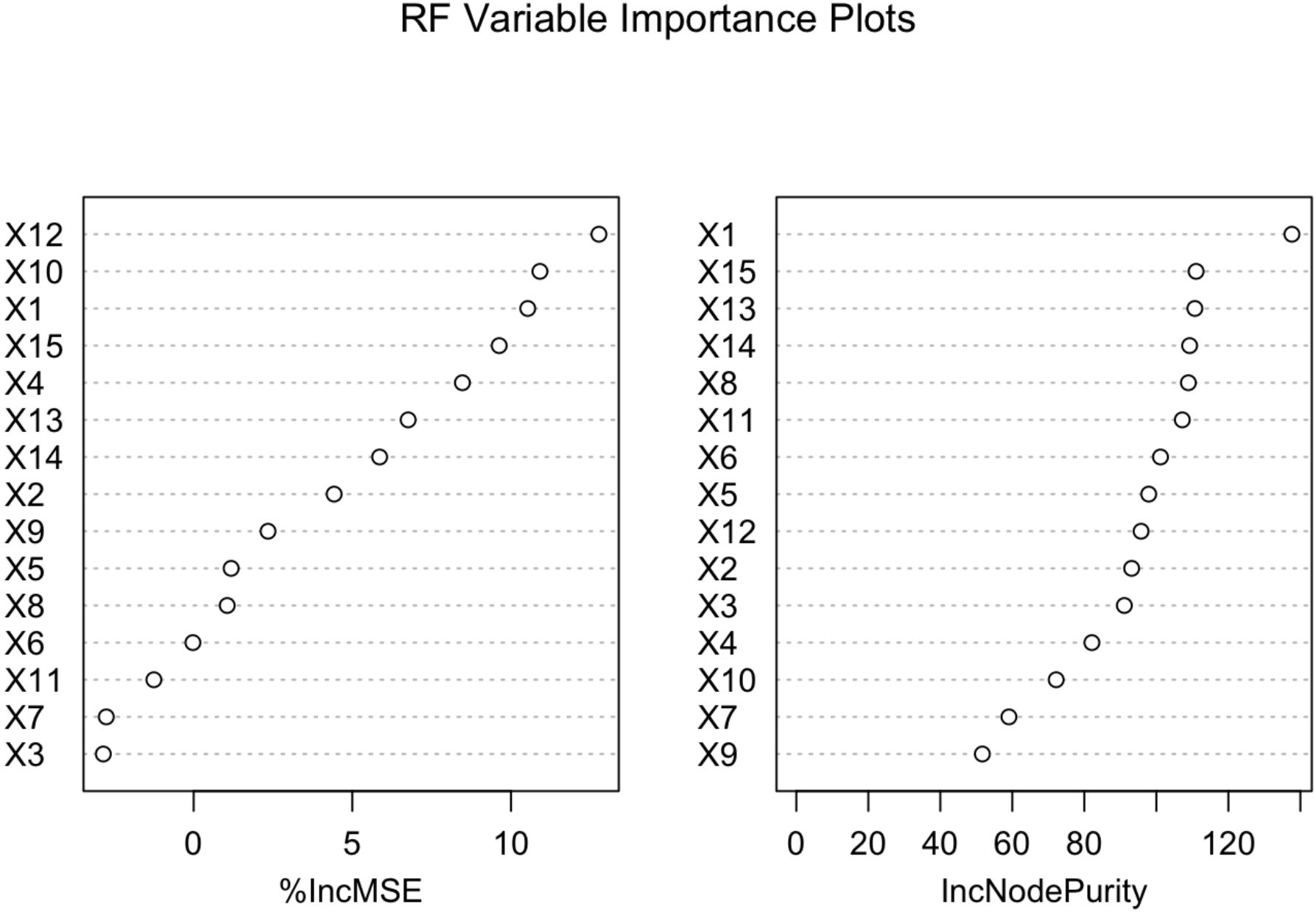
```
### We can get variable importance measures using the importance() function, and
### we can plot them using VarImpPlot()
cat("The importance measures\n")
```

```
## The importance measures
```

```
importance(fit.rf.1) #Print out importance measures
```

```
##         %IncMSE IncNodePurity
## X1  10.52801319     137.64437
## X2   4.42699374      93.13449
## X3  -2.84709365      91.10498
## X4   8.46862298      82.09416
## X5   1.18116133      97.88081
## X6  -0.02392025     101.14389
## X7  -2.75323833      59.05053
## X8   1.05743070     108.89280
## X9   2.34380536      51.67431
## X10 10.91200333      72.17763
## X11 -1.25385865     107.20231
## X12 12.77124496      95.75619
## X13  6.75937822     110.69228
## X14  5.86051941     109.23736
## X15  9.62810659     111.03771
```

```
varImpPlot(fit.rf.1,
           main="RF Variable Importance Plots") #Plot of importance measures
```



OOB error

The OOB error is improved

```
fit.rf.2 = randomForest(Y ~X15 + X1 +X12  , data = train_data, importance = T)
### We can get out-of-bag (OOB) error directly from the predict() function.
### Specifically, if we don't include a new dataset, R gives the OOB predictions
### on the training set.
#Predicted value for any x is the average mean from that x's terminal node in each of the trees
 in the forest
OOB.pred.2 = predict(fit.rf.2) #OOB error
(OOB.MSPE.2 = get.MSPE(train_data$Y, OOB.pred.2))
```

```
## [1] 1.604312
```

```
cat("The OOB error is :", OOB.MSPE.2,"\n")
```

```
## The OOB error is : 1.604312
```

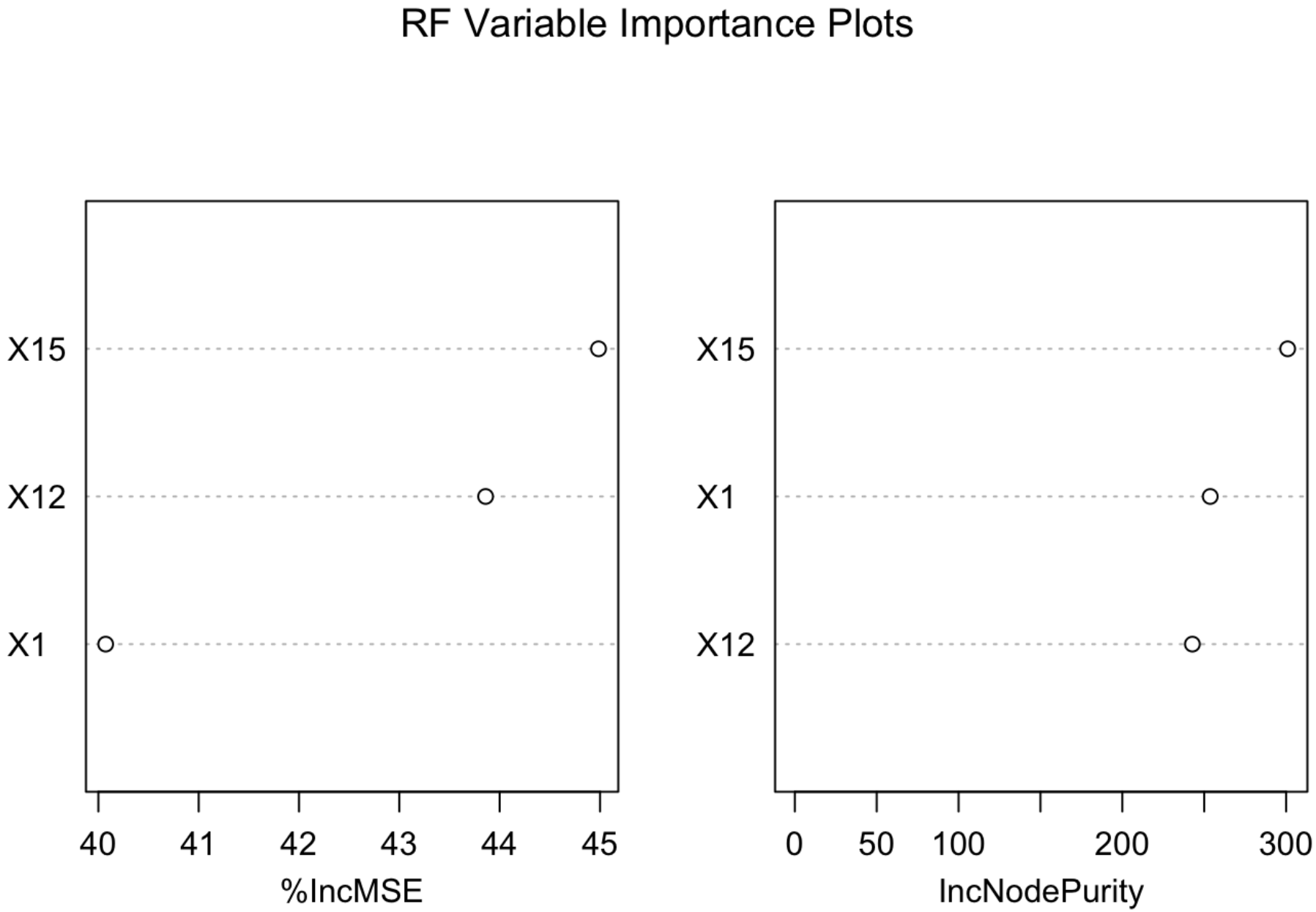
```
### We can get variable importance measures using the importance() function, and
### we can plot them using VarImpPlot()
cat("The importance measures\n")
```

```
## The importance measures
```

```
importance(fit.rf.2) #Print out importance measures
```

```
##      %IncMSE IncNodePurity
## X15 44.98540      300.9275
## X1  40.07305      253.6555
## X12 43.85982      242.7643
```

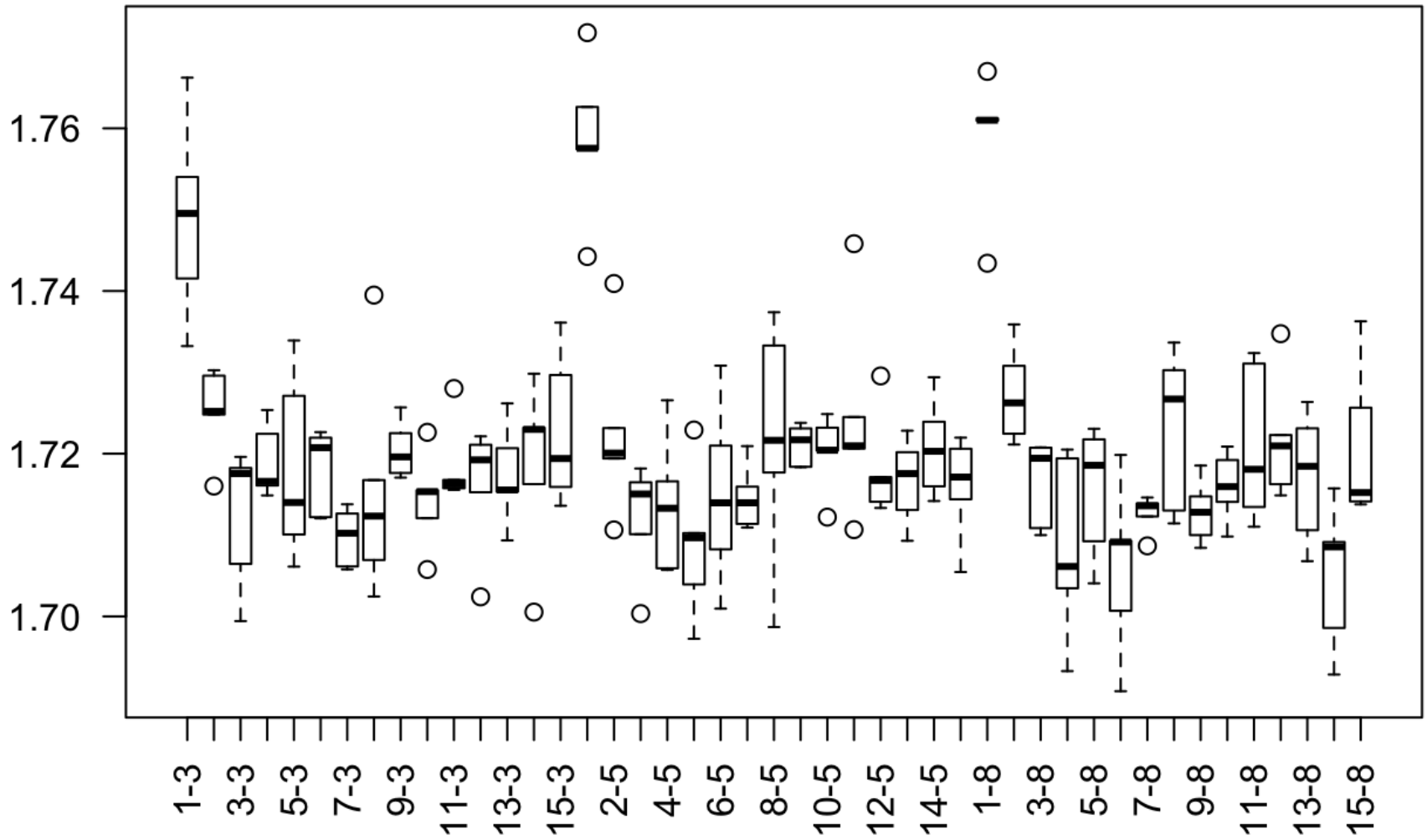
```
varImpPlot(fit.rf.2,
           main="RF Variable Importance Plots") #Plot of importance measures
```



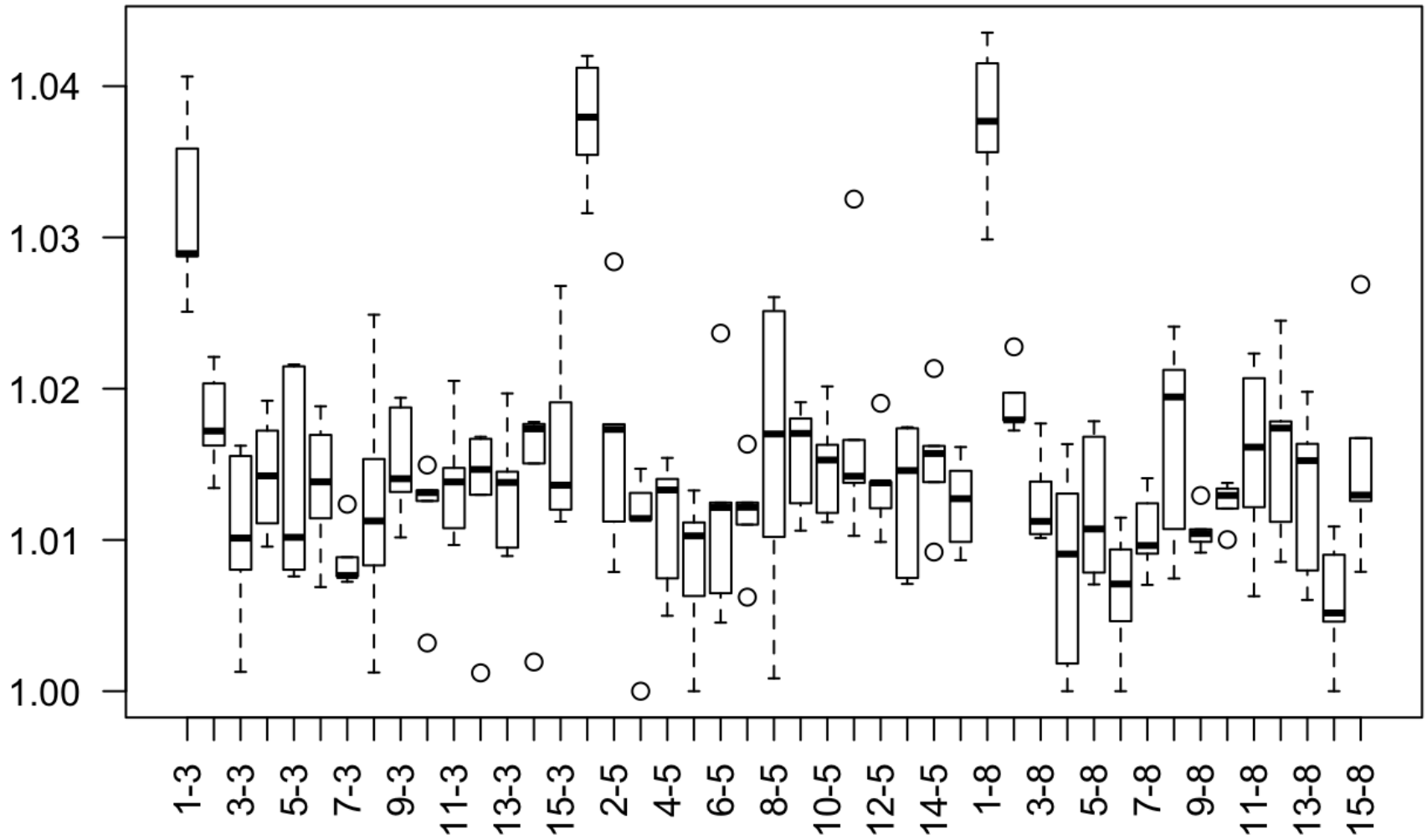
Random forest on replication

```
### Set parameter values
all.mtry = 1:15
all.nodesize = c(3, 5,8)
all.pars = expand.grid(mtry = all.mtry, nodesize = all.nodesize)
n.pars = nrow(all.pars)
### Number of times to replicate process. OOB errors are based on bootstrapping,
### so they are random and we should repeat multiple runs
M = 5

### Create container for OOB MSPEs
OOB.MSPEs = array(0, dim = c(M, n.pars))

for(i in 1:n.pars){
  ### Print progress update
  #print(paste0(i, " of ", n.pars))

  ### Get current parameter values
  this.mtry = all.pars[i,"mtry"]
  this.nodesize = all.pars[i,"nodesize"]
  ### Fit random forest models for each parameter combination
  ### A second for loop will make our life easier here
  for(j in 1:M){
    ### Fit model using current parameter values. We don't need variable
    ### importance measures here and getting them takes time, so set
    ### importance to F
    fit.rf = randomForest(Y ~ ., data = train_data, importance = T,
      mtry = this.mtry, nodesize = this.nodesize)
```

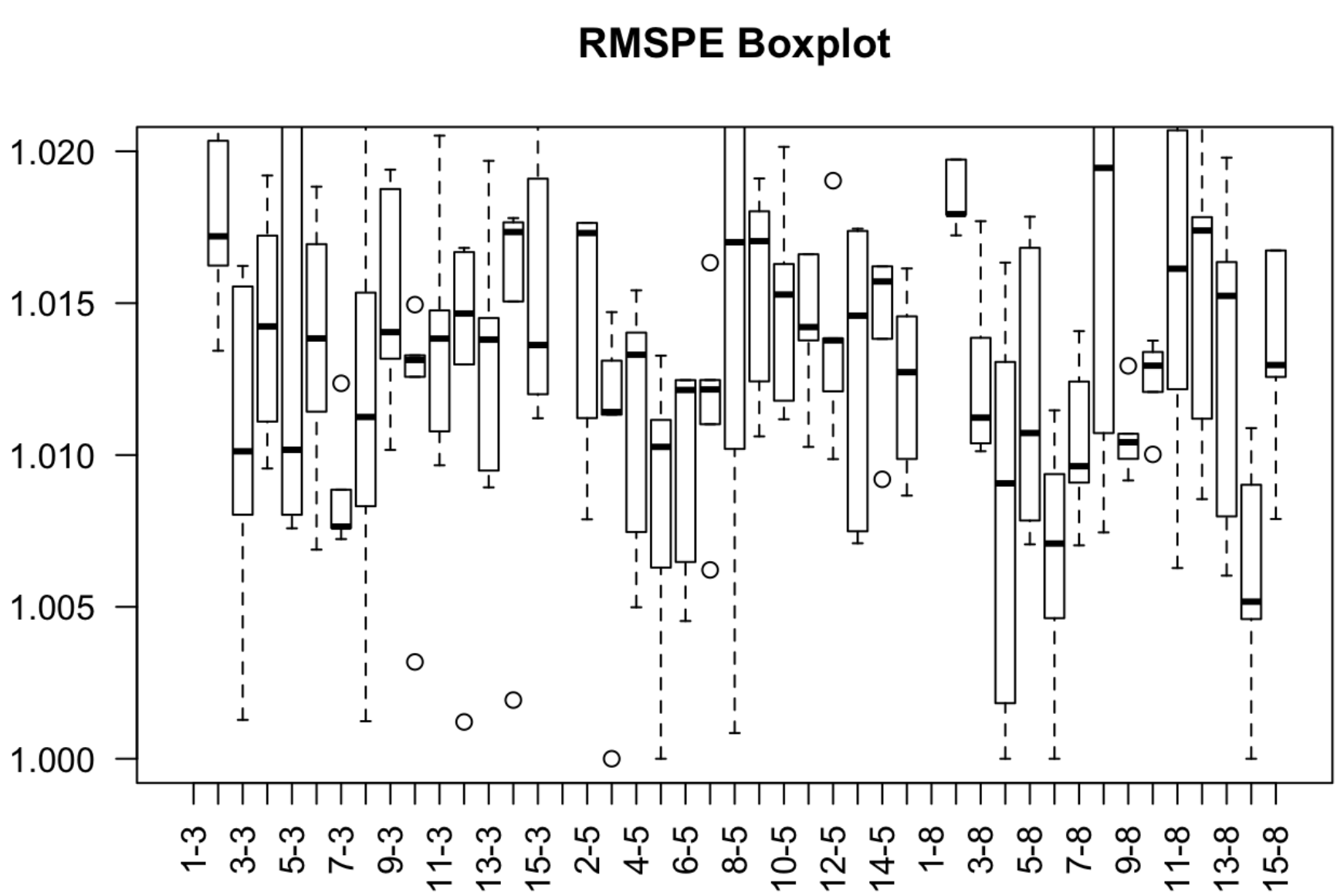
```
    ### Get OOB predictions and MSPE, then store MSPE
    OOB.pred = predict(fit.rf)
    OOB.MSPE = get.MSPE(train_data$Y, OOB.pred)

    OOB.MSPEs[j, i] = OOB.MSPE # Be careful with indices for OOB.MSPEs
  }
}


### We can now make an MSPE boxplot. First, add column names to indicate
### which parameter combination was used. Format is mtry-nodesize
names.pars = paste0(all.pars$mtry,"-",
 all.pars$nodesize)
colnames(OOB.MSPEs) = names.pars

### Make boxplot
boxplot(OOB.MSPEs, las = 2, main = "MSPE Boxplot")
```


MSPE Boxplot
1.76
1.74
1.72
1.70
1-3
3-3
5-3
7-3
9-3
11-3
13-3
15-3
2-5
4-5
6-5
8-5
10-5
12-5
14-5
1-8
3-8
5-8
7-8
9-8
11-8
13-8
15-8

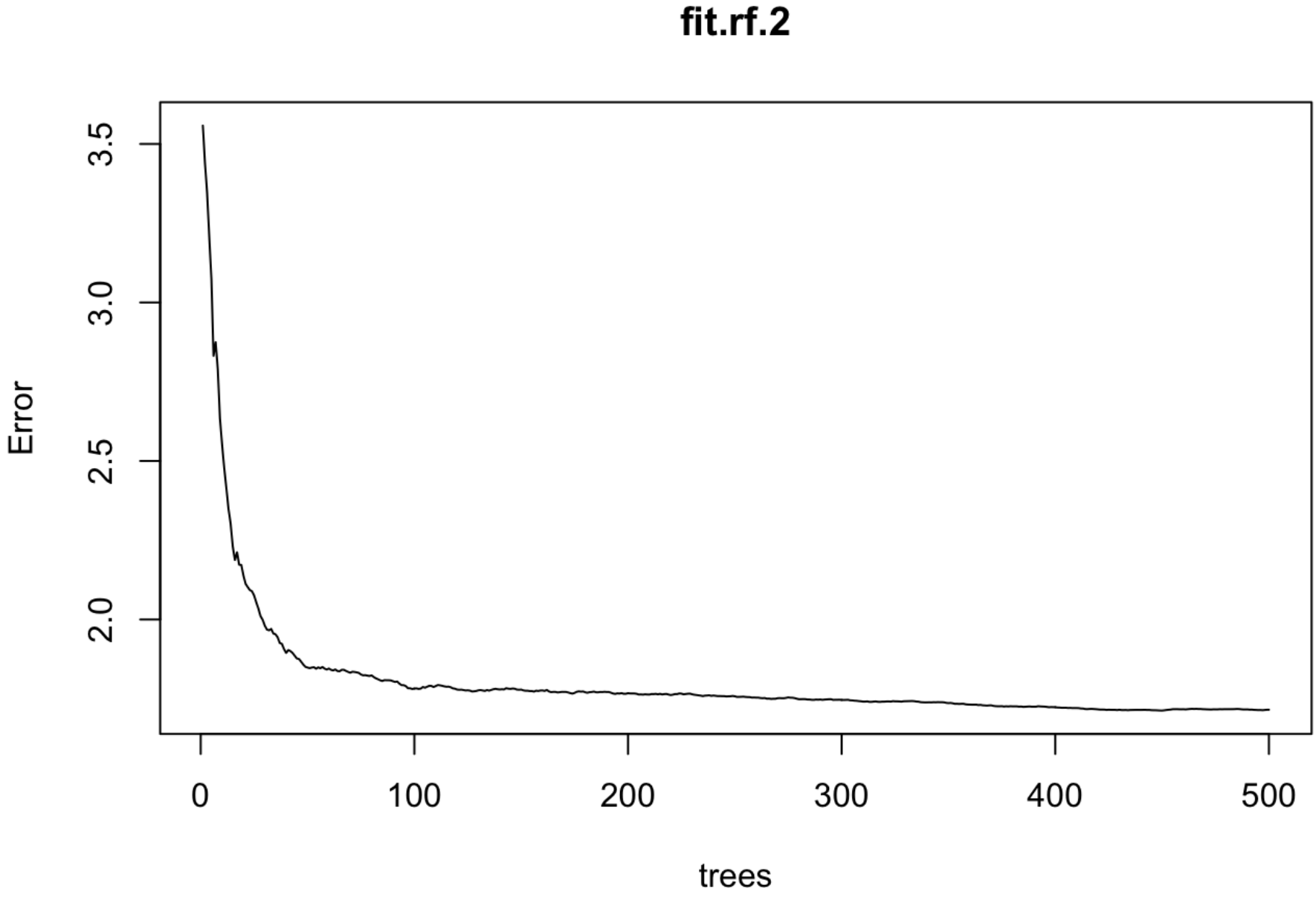
```
### Get relative MSPEs and make boxplot
OOB.RMSPEs = apply(OOB.MSPEs, 1, function(W) W/min(W))
OOB.RMSPEs = t(OOB.RMSPEs)
boxplot(OOB.RMSPEs, las = 2, main = "RMSPE Boxplot")
```


RMSPE Boxplot
1.04
1.03
1.02
1.01
1.00
1-3
3-3
5-3
7-3
9-3
11-3
13-3
15-3
2-5
4-5
6-5
8-5
10-5
12-5
14-5
1-8
3-8
5-8
7-8
9-8
11-8
13-8
15-8

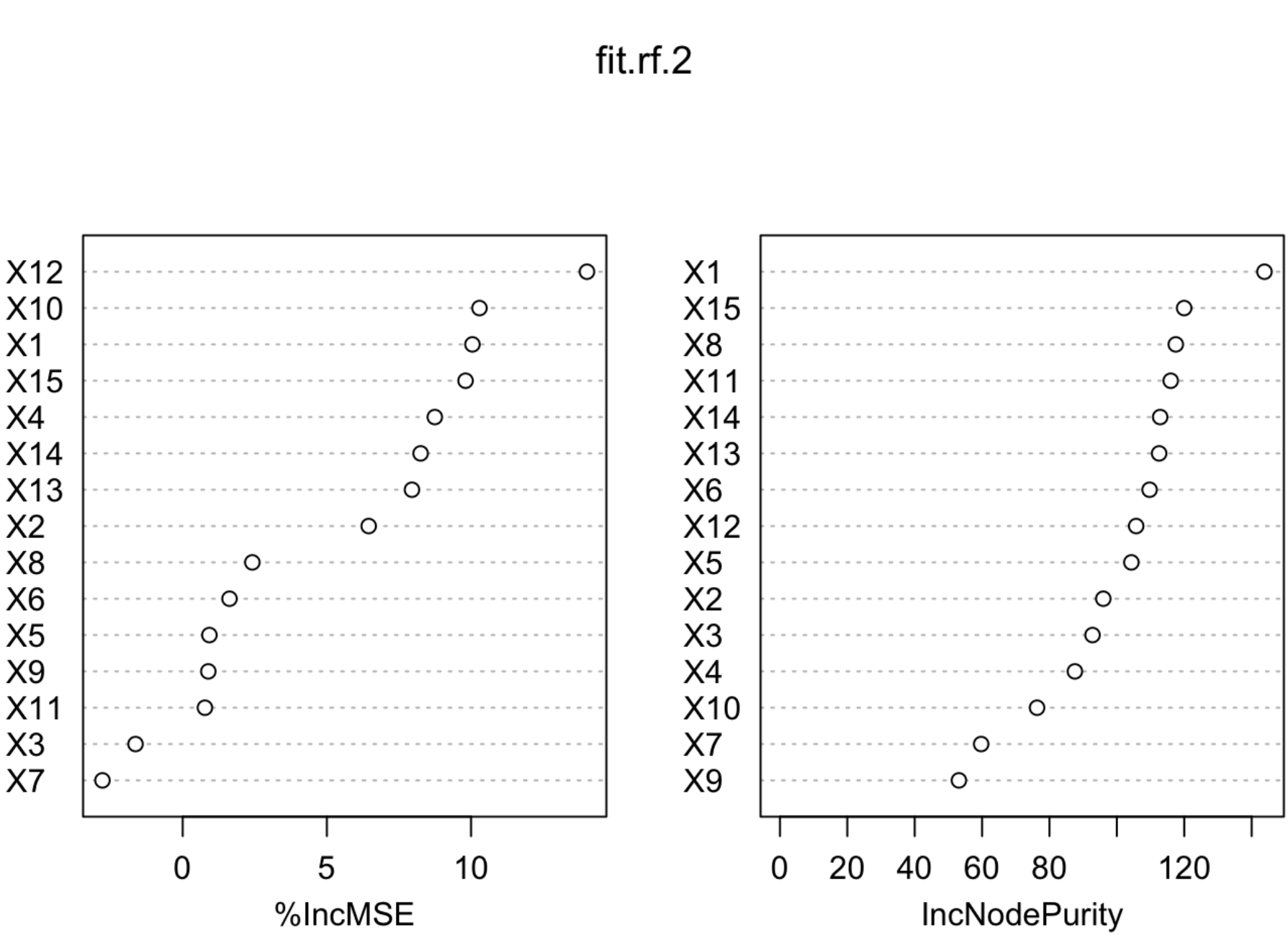
```
### Zoom in on the competitive models
boxplot(OOB.RMSPEs, las = 2, main = "RMSPE Boxplot", ylim = c(1, 1.02))
```


RMSPE Boxplot
1.020
1.015
1.010
1.005
1.000
1-3
3-3
5-3
7-3
9-3
11-3
13-3
15-3
2-5
4-5
6-5
8-5
10-5
12-5
14-5
1-8
3-8
5-8
7-8
9-8
11-8
13-8
15-8

```
### Based on the RMSPE boxplot, the model with mtry=8 and nodesize=3 looks best
### to me. Let's fit this model and see how it compares to the default one from
### above.
fit.rf.2 = randomForest(Y ~ ., data = train_data, importance = T,
  mtry = 8, nodesize = 3)

### Did we use enough trees?
plot(fit.rf.2)
```


fit.rf.2
Error
2.0
2.5
3.0
3.5
0
100
200
300
400
500
trees

```
### How important are the predictors?
varImpPlot(fit.rf.2)
```


fit.rf.2
X12
X10
X1
X15
X4
X14
X13
X2
X8
X6
X5
X9
X11
X3
X7
0
5
10
%IncMSE
X1
X15
X8
X11
X14
X13
X6
X12
X5
X2
X3
X4
X10
X7
X9
0
20
40
60
80
120
IncNodePurity

```
### What is the OOB error?
OOB.pred.2 = predict(fit.rf.2)
(OOB.MSPE.2 = get.MSPE(train_data$Y, OOB.pred.2))
```

```
## [1] 1.715405
```

```
### How about the SMSE
sample.pred.2 = predict(fit.rf.2, train_data)
(SMSE.2 = get.MSPE(train_data$Y, sample.pred.2))
```

```
## [1] 0.2524184
```

Boosting

```
library(gbm)
# Create the folds and save in a matrix
V=5
R=2
n2 = nrow(train_data)

folds = matrix(NA, nrow=n2, ncol=R)
for(r in 1:R){
        folds[,r]=floor((sample.int(n2)-1)*V/n2) + 1
}


trees = 5000
all.shrink = c(0.001, 0.005, 0.025,0.125)
all.depth = c(2,4,6)
all.pars = expand.grid(shrink = all.shrink, depth = all.depth)
n.pars = nrow(all.pars)


NS = length(all.shrink)
ND = length(all.depth)
gb.cv = matrix(NA, nrow=ND*NS, ncol=V*R)
opt.tree = matrix(NA, nrow=ND*NS, ncol=V*R)



qq = 1
for(r in 1:R){
```

```
  for(v in 1:V){
    pro.train = train_data[folds[,r]!=v,]
    pro.test = train_data[folds[,r]==v,]
    counter=1
    for(d in all.depth){
      for(s in all.shrink){
        pro.gbm <- gbm(data=pro.train, Y ~ ., distribution="gaussian",
                       n.trees=trees, interaction.depth=d, shrinkage=s,
                       bag.fraction=0.8)
        treenum = min(trees, 2*gbm.perf(pro.gbm, method="OOB", plot.it=FALSE))
        opt.tree[counter,qq] = treenum
        preds = predict(pro.gbm, newdata=pro.test, n.trees=treenum)
        gb.cv[counter,qq] = mean((preds - pro.test$Y)^2)
        counter=counter+1
      }
    }
    qq = qq+1
  }
}
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
```

```
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
```

```
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
```

```
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
```

```
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
```

```
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
```

```
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
```

```
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
```

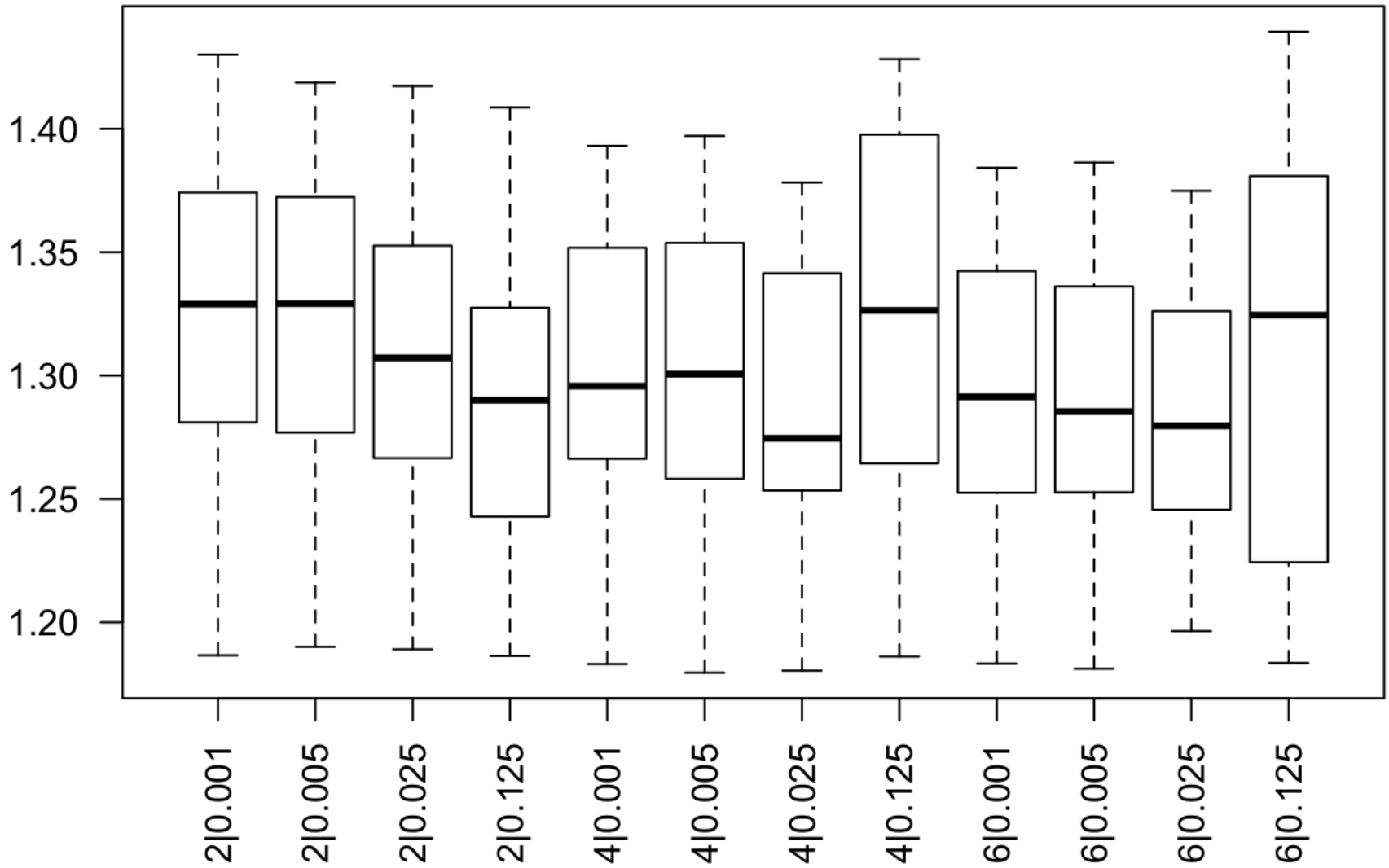
```
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
```

```
parms = expand.grid(all.shrink,all.depth)
row.names(gb.cv) = paste(parms[,2], parms[,1], sep="|")
row.names(opt.tree) = paste(parms[,2], parms[,1], sep="|")

#(mean.tree = apply(opt.tree, 1, mean))

#The mean root-MSPE for each combination of λ and d
(mean.cv = sqrt(apply(gb.cv, 1, mean)))
```

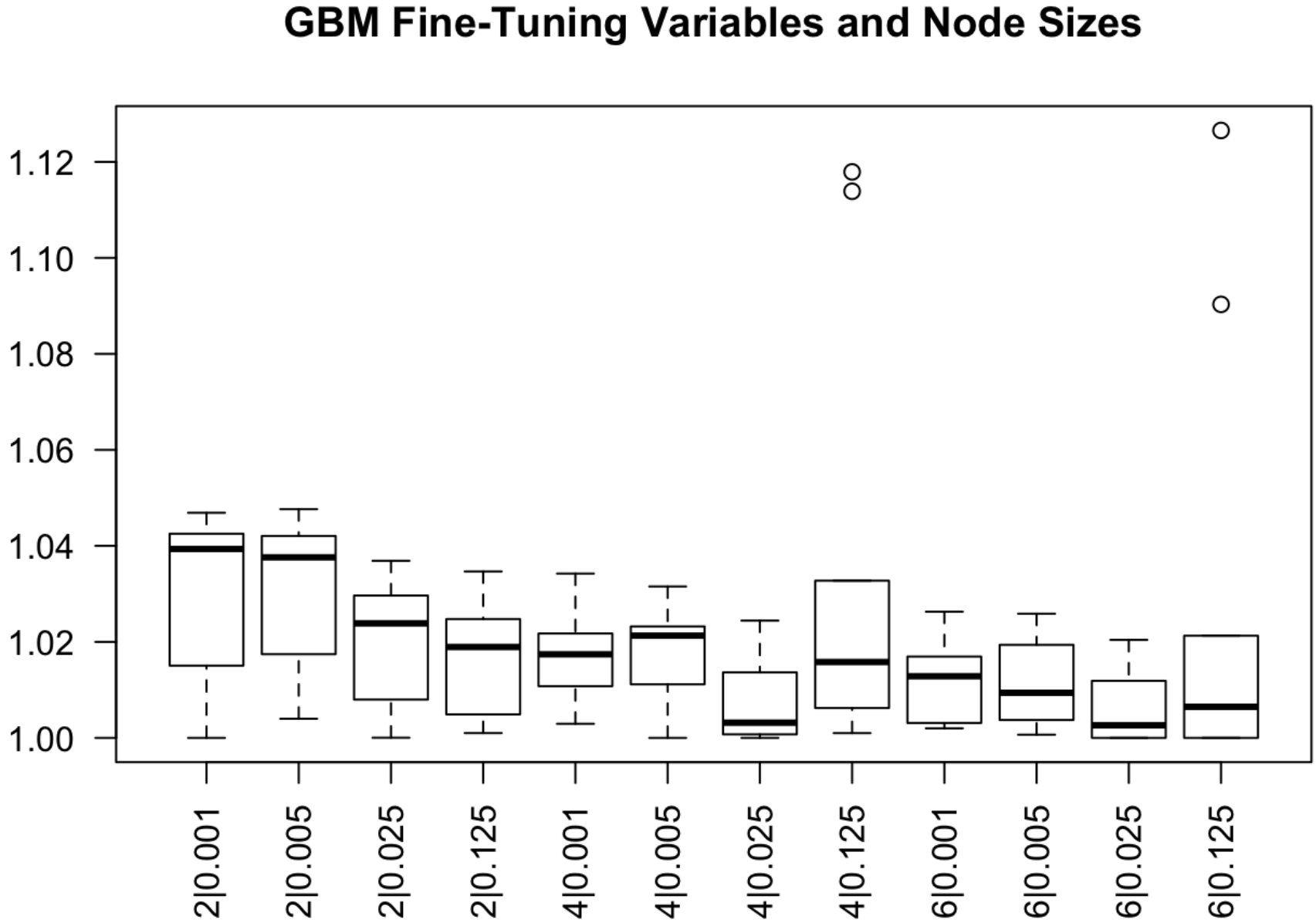
```
##  2|0.001  2|0.005  2|0.025  2|0.125  4|0.001  4|0.005  4|0.025  4|0.125
## 1.313901 1.313526 1.300073 1.293616 1.295452 1.296300 1.282502 1.318520
##  6|0.001  6|0.005  6|0.025  6|0.125
## 1.288761 1.287249 1.280492 1.308071
```

```
min.cv = apply(gb.cv, 2, min)
```

```
boxplot(sqrt(gb.cv), use.cols=FALSE, las=2)
```

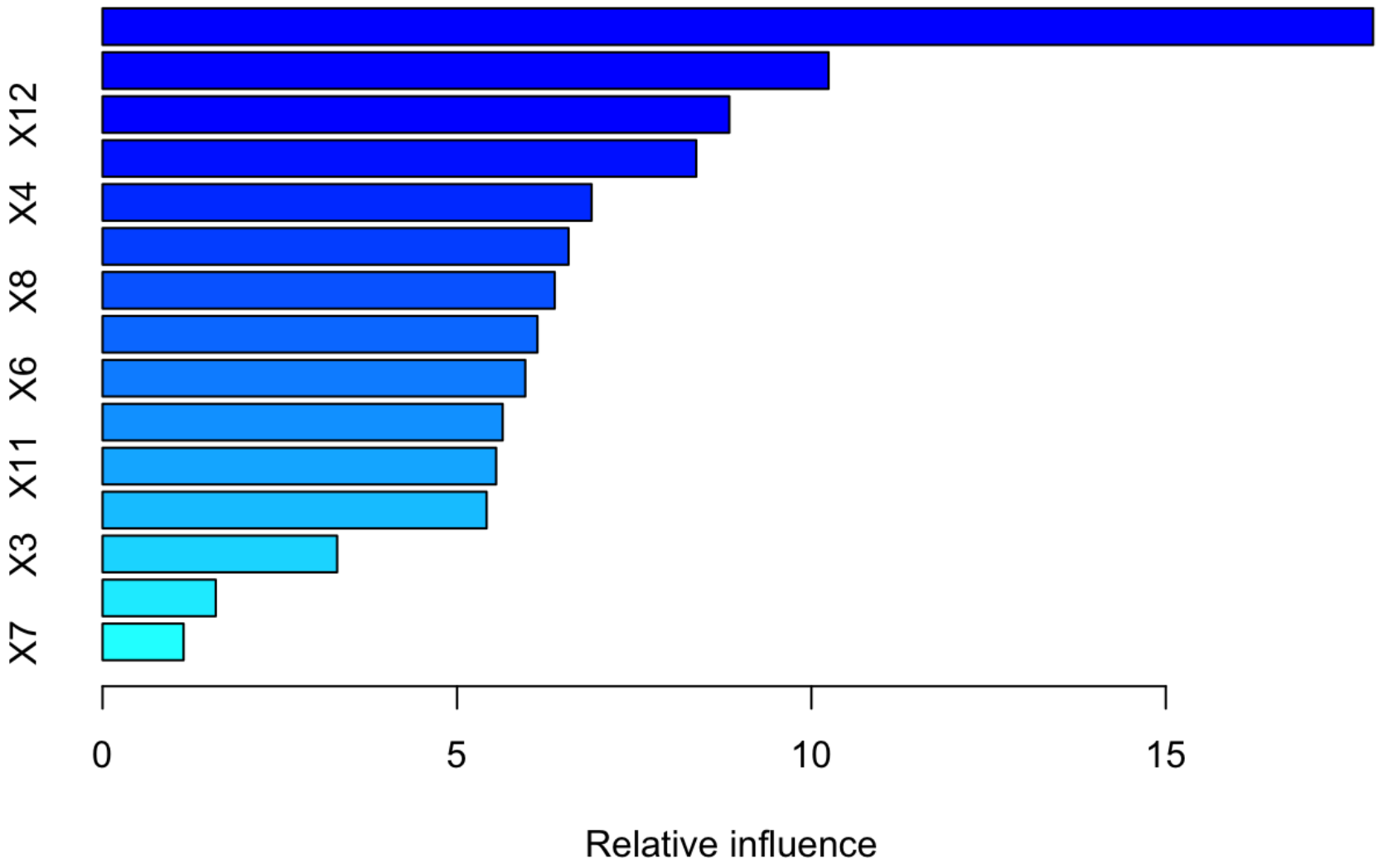

1.40
1.35
1.30
1.25
1.20
2|0.001
2|0.005
2|0.025
2|0.125
4|0.001
4|0.005
4|0.025
4|0.125
6|0.001
6|0.005
6|0.025
6|0.125

```
boxplot(sqrt(t(gb.cv)/min.cv), use.cols=TRUE, las=2,
        main="GBM Fine-Tuning Variables and Node Sizes")
```



The shrinkage 0.125 on all depth are the worst. The combination of depth 4 and shrinkage 0.025 are the best.

```
#############################################################
# Refit Final Model

pro.opt <- gbm(data=train_data, Y~., distribution="gaussian",
               n.trees=500, interaction.depth=4, shrinkage=0.025,
               bag.fraction=0.8)

# Variable Importance

summary(pro.opt)
```


X12
X4
X8
X6
X11
X3
X7
0
5
10
15
Relative influence

```
##     var   rel.inf
## X15 X15 17.924046
## X1   X1 10.241619
## X12 X12  8.841788
## X2   X2  8.375667
## X4   X4  6.898577
## X10 X10  6.574291
## X8   X8  6.379401
## X5   X5  6.135297
## X6   X6  5.963909
## X14 X14  5.645014
## X11 X11  5.552576
## X13 X13  5.417867
## X3   X3  3.309596
## X9   X9  1.596999
## X7   X7  1.143352
```

Final model

```
pro.opt <- gbm(data=train_data, Y~ X1 + X2 + X12 +X15, distribution="gaussian",
               n.trees=500, interaction.depth=4, shrinkage=0.025,
               bag.fraction=0.8)
treenum = min(500, 2*gbm.perf(pro.opt, method="OOB", plot.it=FALSE))
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
is reasonably competitive. Using cv_folds>1 when calling gbm usually results in improved predict
ive performance.
```

```
predictions = predict(pro.opt, newdata=test_data, n.trees=treenum)
max(predictions)
```

```
## [1] 15.01109
```

```
min(predictions)
```

```
## [1] 11.08356
```

```
write.table(predictions, "/Users/janezhu/Desktop/STAT452/theprediction.csv", sep = ",", row.name
s = F, col.names = F)
```