# SEJ Analytix: Machine Learning Methods for Digitally (0/1) Coded COVID19 Policies

## The supervisor id the normalized daily traffic in 2020

EE

#Clear the space

```
 rm(list = ls())
```

#Load packages

```
packs = c('dplyr','ggplot2','AppliedPredictiveModeling', 'caret','corrplot','doParallel',
          'glmnet','earth','kernlab','xgboost','ranger','rpart')
lapply(packs,require,character.only=TRUE)
```

```
## Loading required package: dplyr
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

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

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Loading required package: AppliedPredictiveModeling
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 4.3.3
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

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

```
## Loading required package: corrplot
```

```
## Warning: package 'corrplot' was built under R version 4.3.3
```

```
## corrplot 0.94 loaded
```

```
## Loading required package: doParallel
```

```
## Warning: package 'doParallel' was built under R version 4.3.3
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 4.3.3
```

```
## Loading required package: iterators
```

```
## Warning: package 'iterators' was built under R version 4.3.3
```

```
## Loading required package: parallel
```

```
## Loading required package: glmnet
```

```
## Warning: package 'glmnet' was built under R version 4.3.3
```

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

```
## Loaded glmnet 4.1-8
```

```
## Loading required package: earth
```

```
## Warning: package 'earth' was built under R version 4.3.3
```

```
## Loading required package: Formula
```

```
## Loading required package: plotmo
```

```
## Warning: package 'plotmo' was built under R version 4.3.3
```

```
## Loading required package: plotrix
```

```
## Loading required package: kernlab
```

```
## Warning: package 'kernlab' was built under R version 4.3.3
```

```
##
## Attaching package: 'kernlab'
```

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

```
## Loading required package: xgboost
```

```
## Warning: package 'xgboost' was built under R version 4.3.3
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
## Loading required package: ranger
```

```
## Warning: package 'ranger' was built under R version 4.3.3
```

```
## Loading required package: rpart
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] TRUE
##
## [[5]]
## [1] TRUE
##
## [[6]]
## [1] TRUE
##
## [[7]]
## [1] TRUE
##
## [[8]]
## [1] TRUE
##
## [[9]]
## [1] TRUE
##
## [[10]]
## [1] TRUE
##
## [[11]]
## [1] TRUE
##
## [[12]]
## [1] TRUE
```

```
setwd("C:/Users/Mama/Desktop/Customer_cases/CovidOnTransportation")
```

#Load the data sets

```
covid = read.csv('binaryCodedCovid_and_imputedTrafficDaily.csv')
```

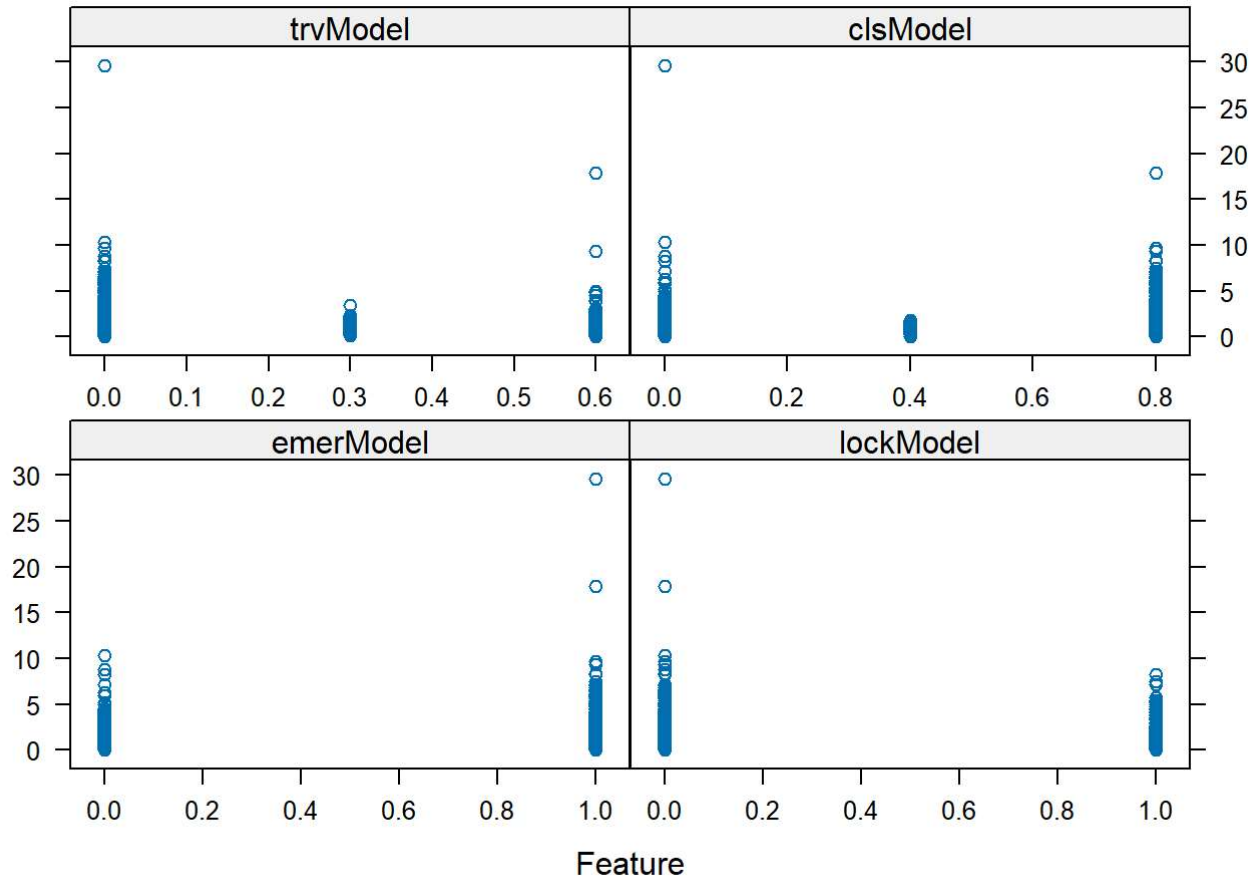
# Data set and training test split

```
concrete<-covid%>%dplyr::select(propTraffic, emerModel, lockModel, trvModel, clsModel)
Xtot = select(concrete,-propTraffic)
Ytot = select(concrete,propTraffic) %>% unlist

set.seed(1)
inTrain   = createDataPartition(concrete$propTraffic, p = 0.8, list = FALSE)
Ytrain    = Ytot[inTrain]
Ytest     = Ytot[-inTrain]
Xtrain    = Xtot[inTrain,]
Xtest     = Xtot[-inTrain,]
training  = concrete[ inTrain,]
testing   = concrete[-inTrain,]
```

# Exploratory data analysis (EDA)

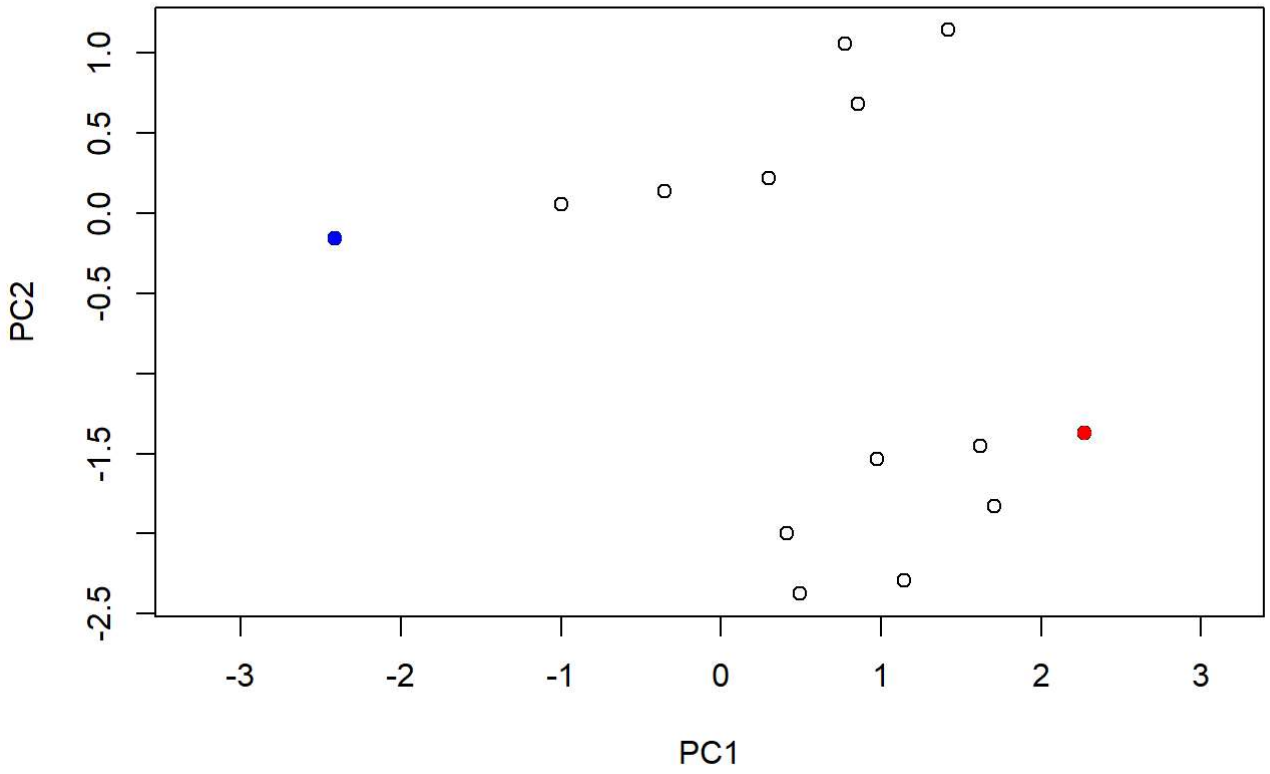
- Find any issues (missing data, extreme observations)
- Gain insights into types of transformations and methods that might be of use
- Discover interesting things about the problem you're ultimately trying to solve

```
featurePlot(Xtrain, Ytrain)
```



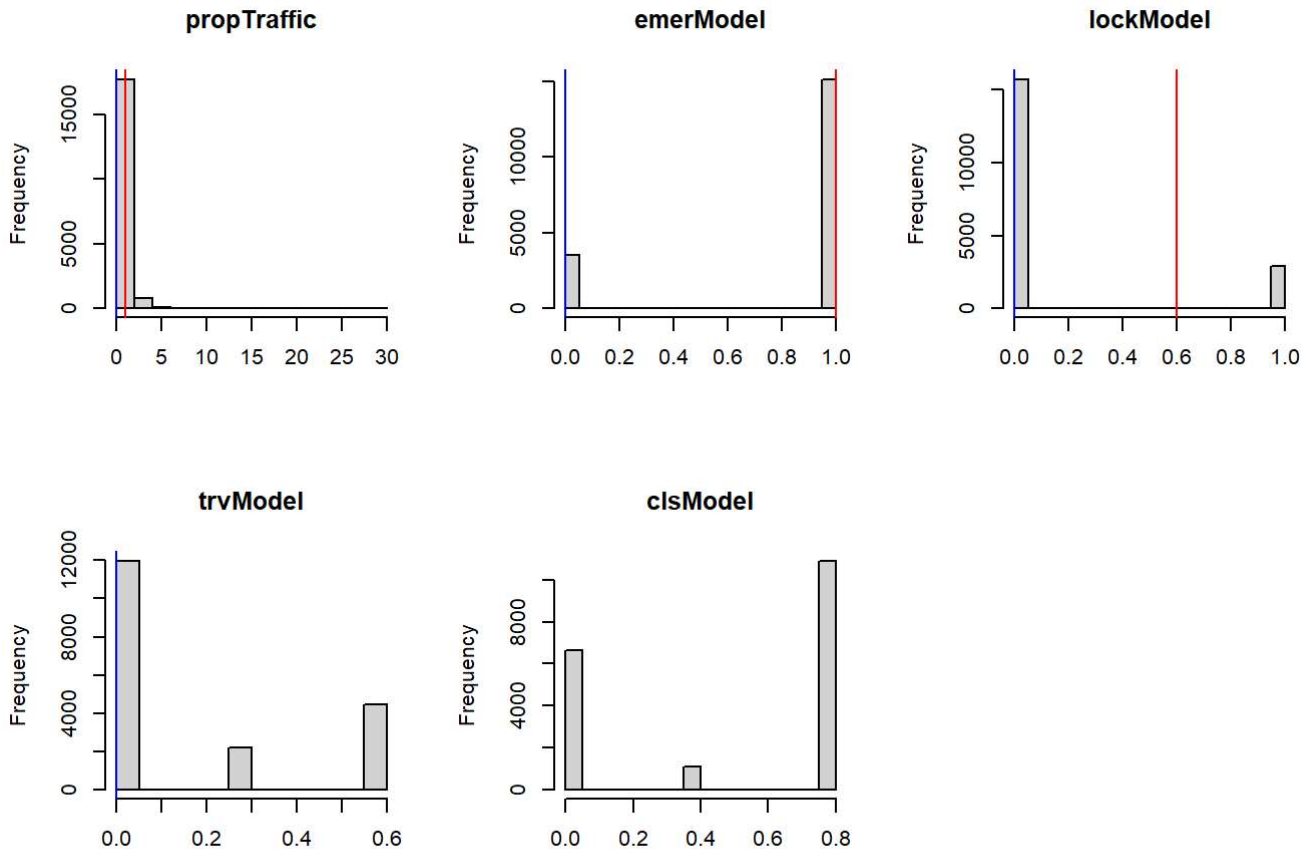

Let's also look at the shape of the distributions of the features. Since I'm only looking at the features, I'm going to go the 'semi supervised' approach and look at all the data's features. Let's look at the PCA:

```
pcaOut = prcomp(Xtot,center=TRUE,scale=TRUE)

extreme1 = c(100,100)
extreme2 = c(22,51)

plot(pcaOut$x[,1:2], asp = 1)
points(pcaOut$x[c(extreme1,extreme2),1:2],
       col = c('red','red','blue','blue'), pch=16)
```




Let's take a look at the distributions via histograms and highlight these observations:

```
par(mfrow = c(2,3))
for(j in 1:5){
  hist(concrete[,j], main = names(concrete)[j],xlab='')
  abline(v = Xtot[extreme1[1],j],col='red')
  abline(v = Xtot[extreme1[2],j],col='red',lty=2)
  abline(v = Xtot[extreme2[1],j],col='blue')
  abline(v = Xtot[extreme2[2],j],col='blue',lty=2)
}
```

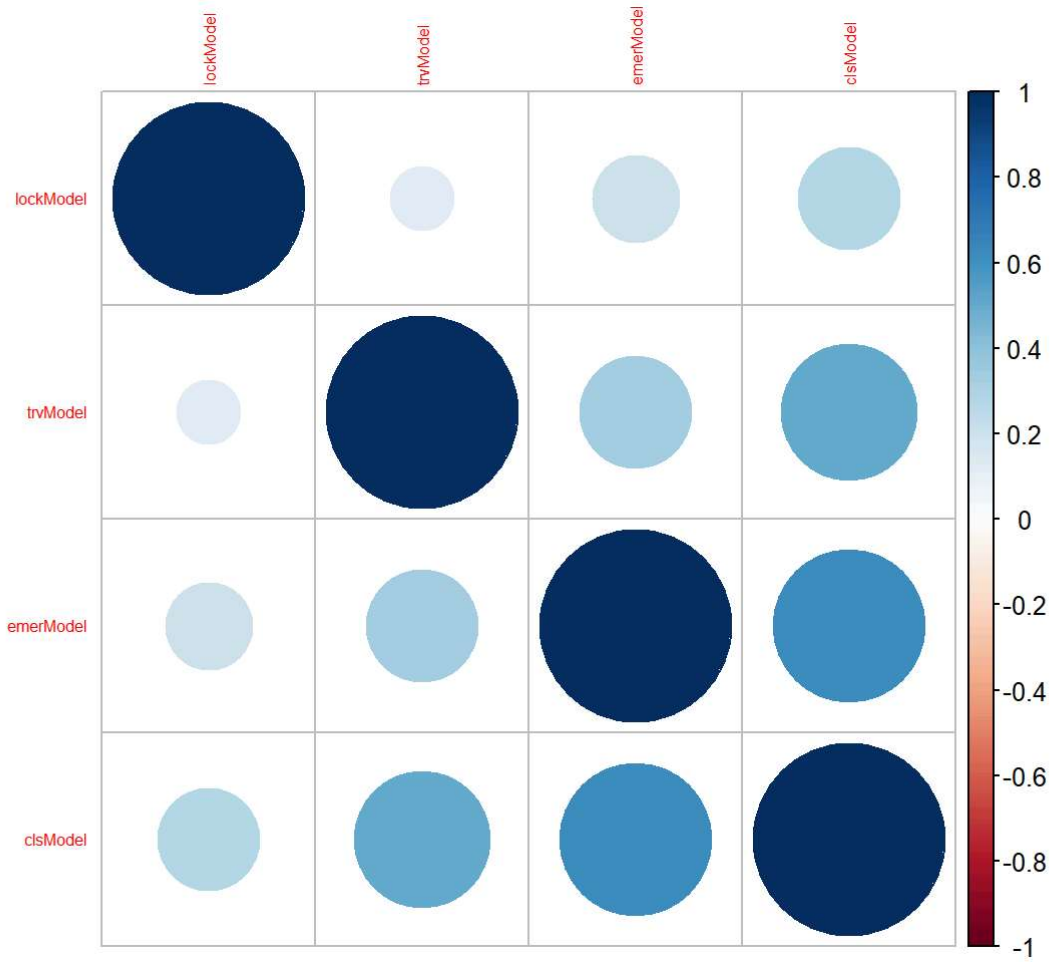



Some comments:

- skew transformations won't be useful
- tree-like methods look attractive due to some features having two 'groups' (MARS, Pruned Trees, boosting, random forest)

Let's examine the correlation structure of the features:

```
corrplot(cor(Xtot), order = "hclust", tl.cex = .5)
```




Emergency Order seems linearly correlated to the Close Non-Essential Stores Order. As Emergency has affaected all states, and we have seen that already from the CC2 results of the time-series modeling, remove Emergency from the data set for ML.

# Repeated cross validation

When computations aren't severely constrained, repeating the cross-validation procedure protects against unfortunate fold selection:

```
trControl = trainControl(method = "repeatedcv", repeats = 2, number = 10)
```

## Parallelism

Caret can use a parallel back end (mainly for doing cross-validation or bootstrap). We need to choose the number of clusters. Be sure to take care of memory usage; you can quickly run out of memory as you add cores

```
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)
print(cl)
```

```
## socket cluster with 11 nodes on host 'localhost'
```

#Make matrices

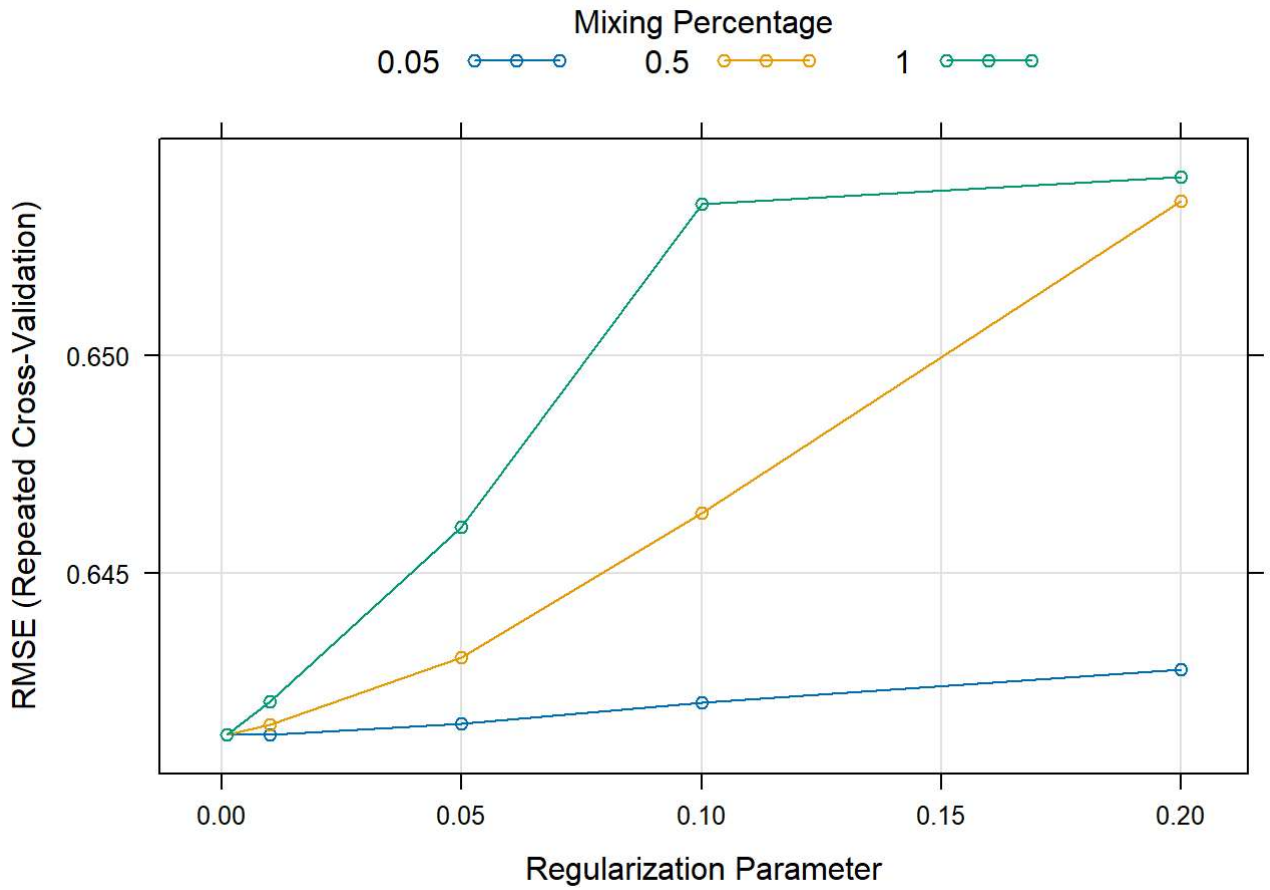
```
XtestMat  = as.matrix(Xtest)
XtrainMat = as.matrix(Xtrain)
```

# Linear models

```
set.seed(1)
lassoGrid = expand.grid(lambda = c(0.001, .001, .01, .05, .1,.2),

alpha = c(0.05, 0.5,1))
elasticOut = train(x = XtrainMat, y = Ytrain,
method = "glmnet",
preProc = c("center", "scale"),
tuneGrid = lassoGrid,
trControl = trControl)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

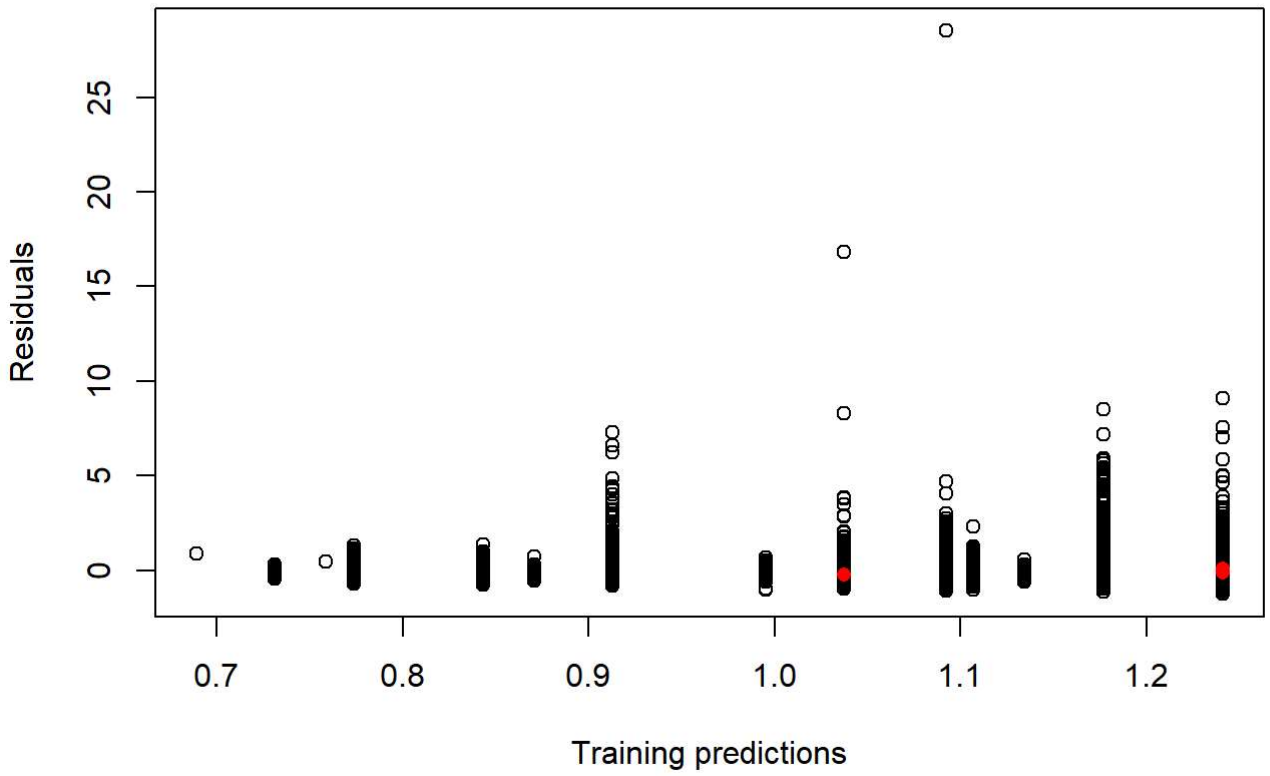
```
plot(elasticOut)
```




```
elasticOut$bestTune
```

```
##   alpha lambda
## 1  0.05  0.001
```

```
glmnetOut = glmnet(x = XtrainMat, y = Ytrain, alpha = elasticOut$bestTune$alpha)
betaHatGlmnet = coef(glmnetOut, s = elasticOut$bestTune$lambda)
YhatTrainGlmnet = predict(glmnetOut, XtrainMat, s = elasticOut$bestTune$lambda)

residuals = Ytrain - YhatTrainGlmnet
plot(YhatTrainGlmnet, residuals,
xlab = 'Training predictions', ylab = 'Residuals')
points(YhatTrainGlmnet[c(extreme1,extreme2)],
residuals[c(extreme1,extreme2)],
col = 'red', pch=16)
```



# Nonlinear methods

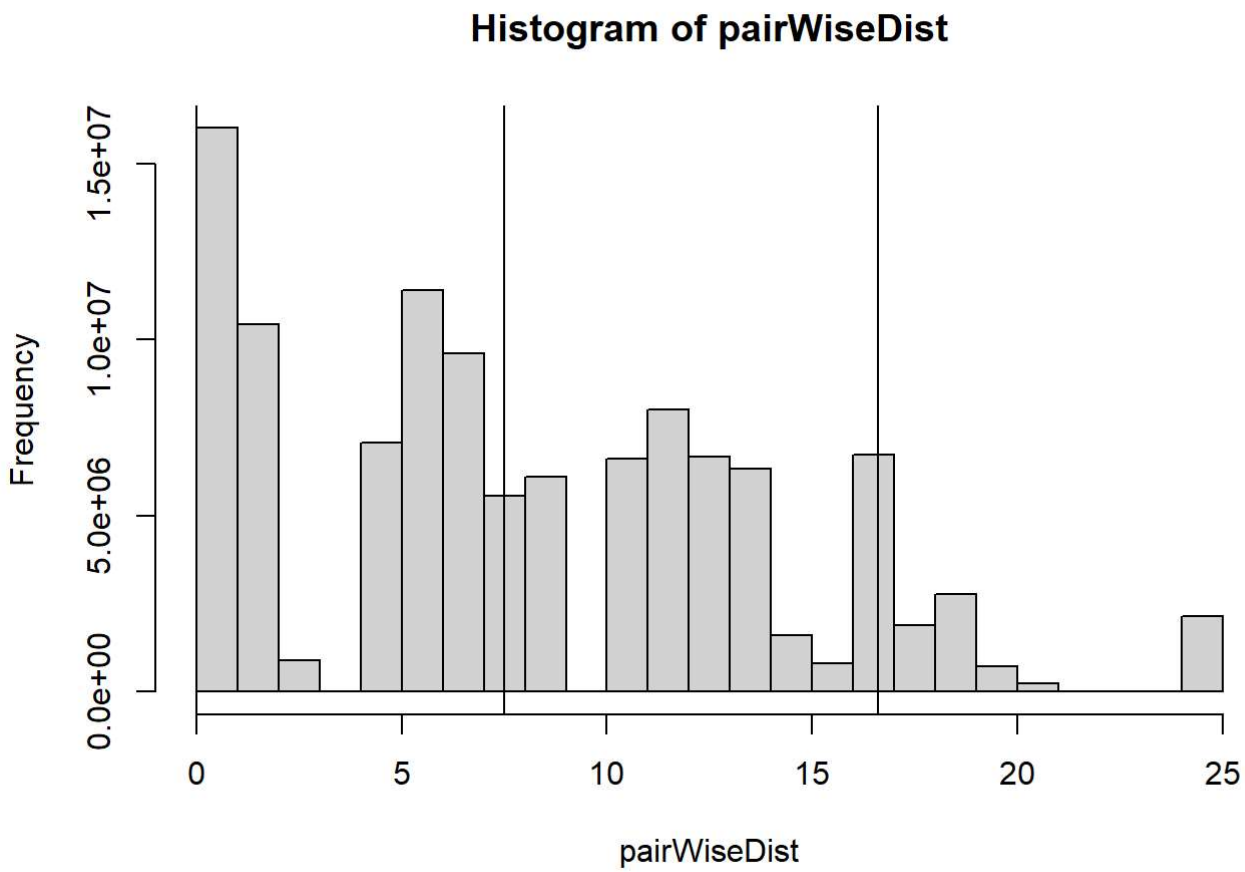
## MARS

```
set.seed(1)
tuneGrid = expand.grid(degree = 1:2,
                       nprune = 10:35)
marsOut = train(x = Xtrain, y = Ytrain,
                method = "earth",
                tuneGrid = tuneGrid,
                trControl = trControl)
```

## SVM

Let's look at SVMs as well. We can use the radial basis function and the distance-based range we discussed earlier.

```
pairWiseDist = dist(scale(Xtrain), method = 'euclidean')**2

sigmaRange = quantile(pairWiseDist, c(0.9,0.5,0.1))
hist(pairWiseDist)
abline(v = sigmaRange[1])
abline(v = sigmaRange[2])
abline(v = sigmaRange[3])
```

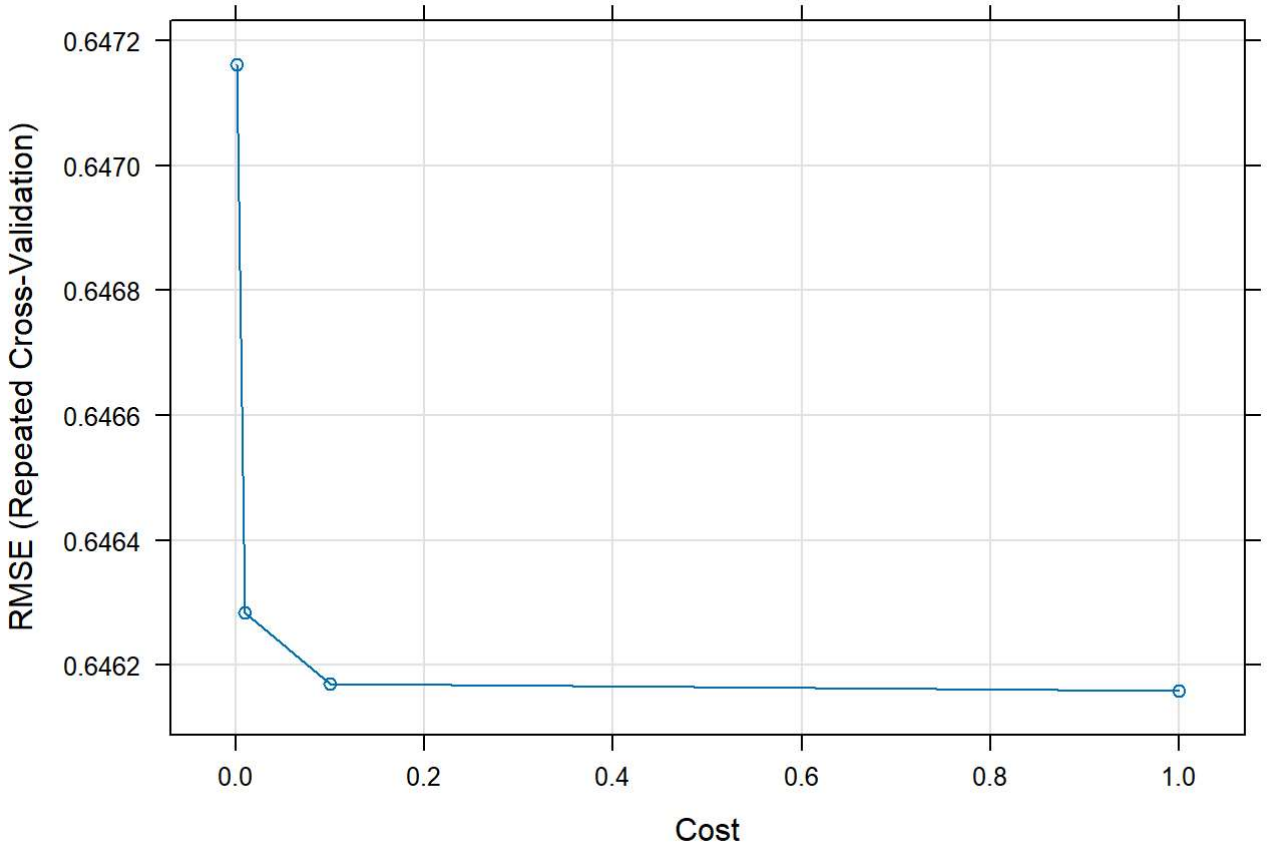



```
1/sigmaRange
```

```
##        90%        50%        10%
## 0.06024714 0.13329340        Inf
```

#Plot SVM Linear

```
set.seed(1)
tuneGrid = expand.grid(C = c(.001,.01,.1,1))
svmOut = train(x = Xtrain, y = Ytrain,
               method = "svmLinear",
               tuneGrid = tuneGrid,
               preProc = c("center", "scale"),
               trControl = trControl)

plot(svmOut)
```



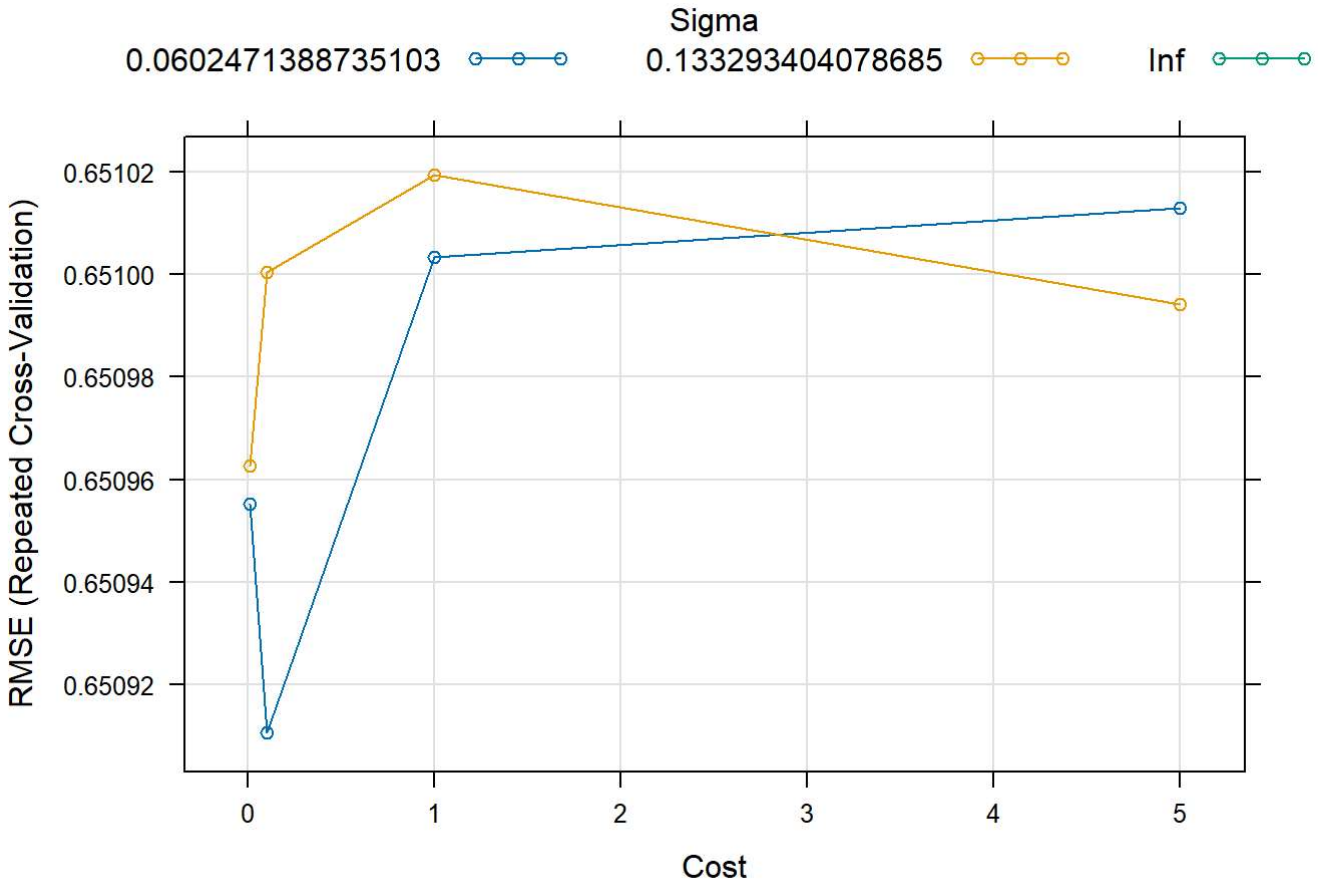

#Plot SVM Radial

```
set.seed(1)
tuneGrid = expand.grid(C = c(.01,.1,1,5),
                       sigma = 1/sigmaRange)
svmRadialOut = train(x = Xtrain, y = Ytrain,
               method = "svmRadial",
               tuneGrid = tuneGrid,
               preProc = c("center", "scale"),
               trControl = trControl)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
## Warning in train.default(x = Xtrain, y = Ytrain, method = "svmRadial", tuneGrid
## = tuneGrid, : missing values found in aggregated results
```

```
plot(svmRadialOut)
```

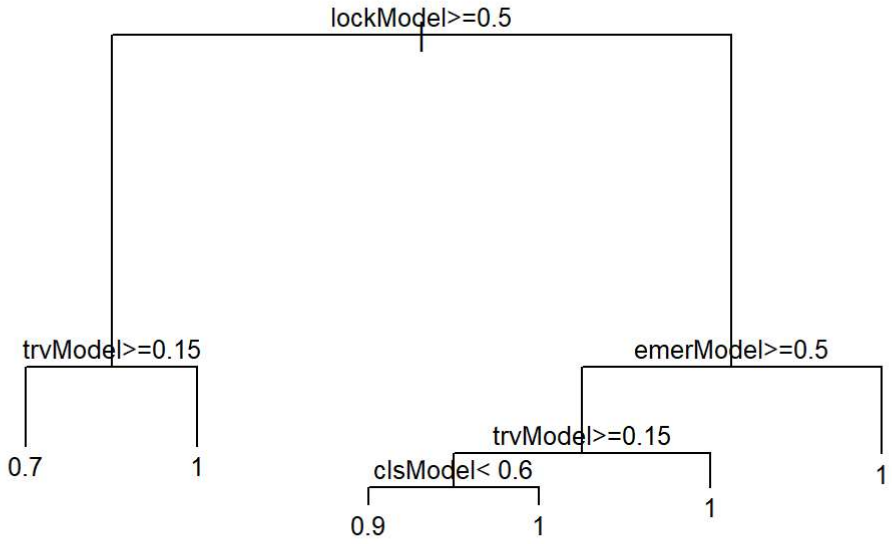



# Pruned Tree

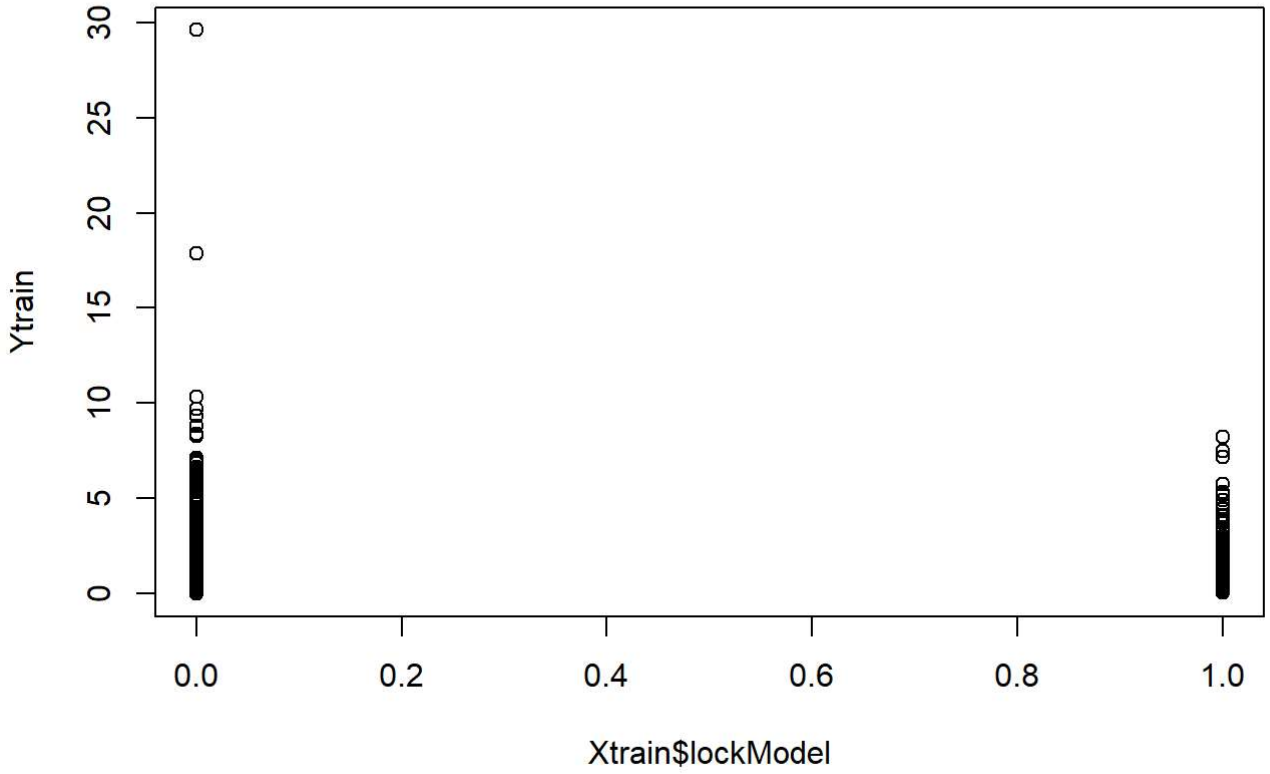
```
tuneGrid = expand.grid(cp = c(0.001, 0.01, 0.1))

rpartOut = train(x = Xtrain, y = Ytrain,
                 method = "rpart",
                 tuneGrid = tuneGrid,
                 trControl = trControl)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
plot(rpartOut$finalModel,margin= rep(.1,4))
text(rpartOut$finalModel, cex = 0.8, digits = 1)
```




```
plot(Xtrain$lockModel, Ytrain)
abline(v = 15, col = 'red')
```




# Random forest

#Timing functions

```
require(microbenchmark)
```

```
## Loading required package: microbenchmark
```

```
## Warning: package 'microbenchmark' was built under R version 4.3.3
```

```
tuneGridRanger = data.frame(splitrule = 'variance',
                            min.node.size = 5,
                            mtry = round(sqrt(ncol(Xtrain))))
tuneGridRf     = data.frame(mtry = round(sqrt(ncol(Xtrain))))

mbm = microbenchmark(
  "ranger" = {
  rangerOut   = train(x = Xtrain, y = Ytrain,
                      method = "ranger",
                      tuneGrid = tuneGridRanger,
                      trControl = trainControl(method = 'none'))
},
  "rf" = {
  rfOut   = train(x = Xtrain, y = Ytrain,
                      method = "rf",
                      tuneGrid = tuneGridRf,
                      trControl = trainControl(method = 'none'))
}, times = 2)
mbm
```

```
## Unit: seconds
##    expr      min       lq     mean   median       uq      max neval
##  ranger 1.036541 1.036541 1.041678 1.041678 1.046814 1.046814     2
##      rf 6.815504 6.815504 6.818400 6.818400 6.821296 6.821296     2
```

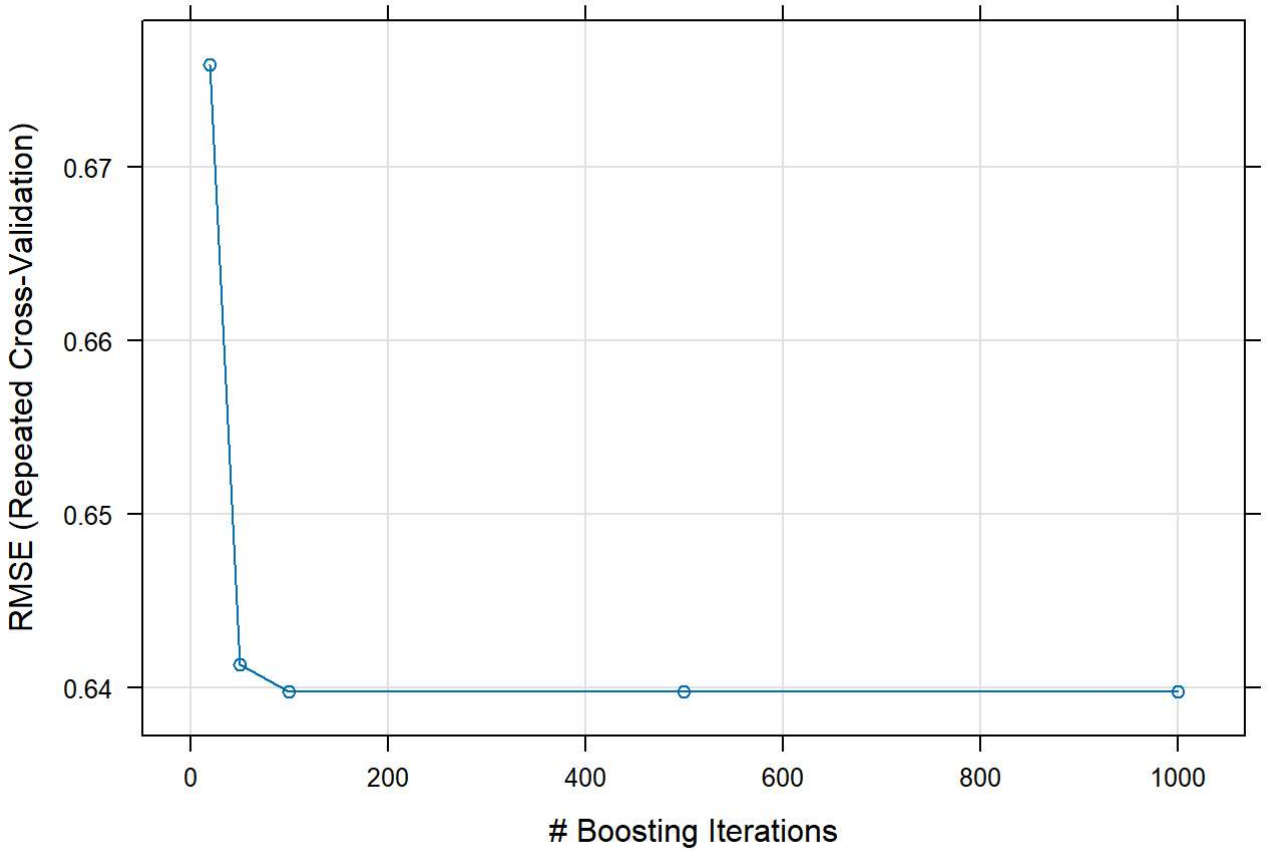
#Ranger method is substantially faster.

```
set.seed(1)
tuneGridRanger = data.frame(splitrule = 'variance',
                            min.node.size = 5,mtry = round(sqrt(ncol(Xtrain))))
rfOut      = train(x = Xtrain, y = Ytrain,
                   method = "ranger",
                   num.trees = 500,
                   tuneGrid = tuneGridRanger,
                   importance = 'permutation',
                   trControl = trControl)
```

# Boosting

There are two main implementations of boosting: Xgboost and gbm (which stands for gradient boosting machine). Xgboost is faster and has some more capabilities (like supporting sparse matrices). gbm has some things that Xgboost doesn't as well: 'partial dependency plots' (PDP).

```
#inbalnce classes -> use kappa instaed of accuracy
set.seed(1)
tuneGrid = data.frame('nrounds'=c(20, 50,100,500,1000),
                      'max_depth' = 6,
                      'eta' = .05,
                      'gamma' = 0,
                      'colsample_bytree' = 1,
                      'min_child_weight' = 0,
                      'subsample' = .5)
boostOut   = train(x = Xtrain, y = Ytrain,
                   method = "xgbTree",
                   tuneGrid = tuneGrid,
                   trControl = trControl)
plot(boostOut)
```




## Boosting importance

Xgboost reports importance slightly differently than the other methods. It reports the importance in decreasing order, instead of in the order of the original features and also doesn't report features that have 0 importance. This is slightly inconvenient for the plot I want to make, so I'm going to reorder it and add the zeros back in:

```
boostImportance0 = xgb.importance(model = boostOut$finalModel)
boostImportance0
```

```
##      Feature      Gain     Cover  Frequency
##       <char>     <num>     <num>      <num>
## 1: lockModel 0.4647403 0.2204673 0.21299385
## 2:  trvModel 0.2309326 0.3105424 0.31799824
## 3: emerModel 0.1552330 0.1592539 0.08621598
## 4:  clsModel 0.1490941 0.3097364 0.38279192
```

```
boostImportance  = c(boostImportance0$Gain[order(match(boostImportance0$Feature,names(Xtrain)))],0,0)
```

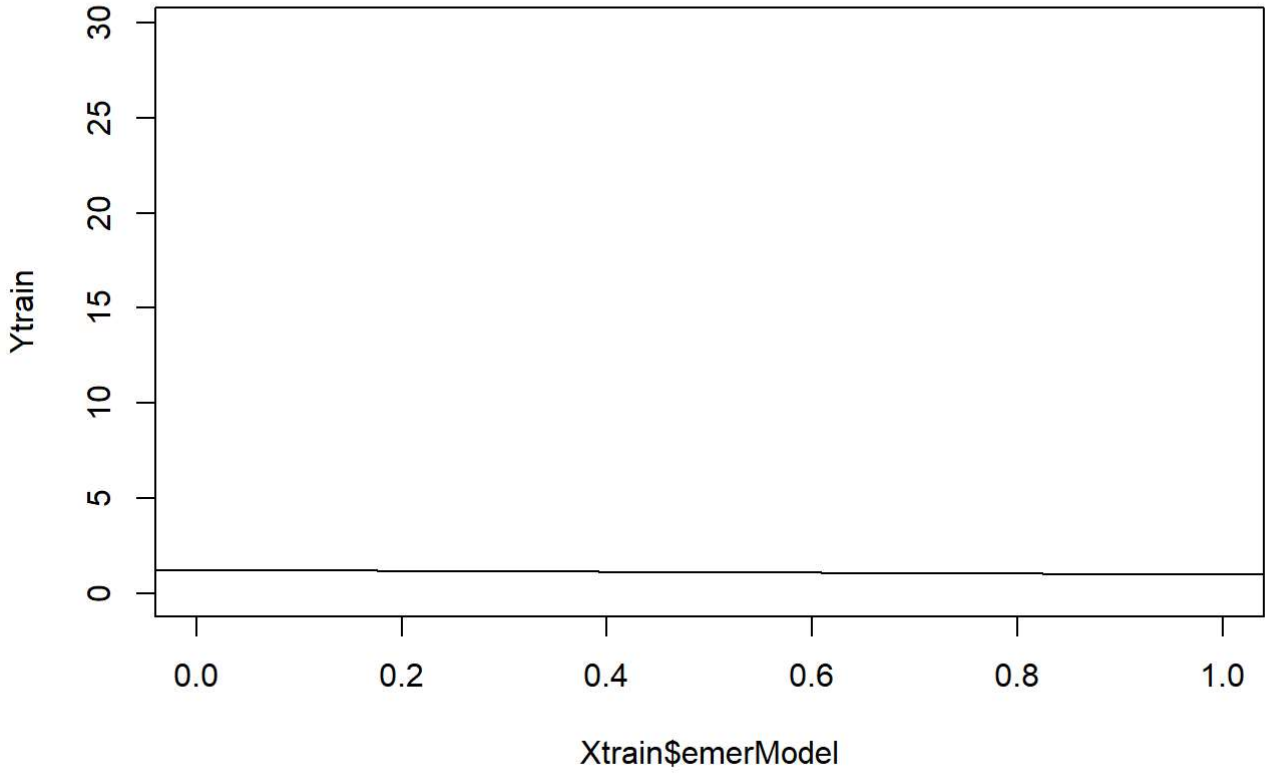
# Partial dependency plots

Let's look at PDP. These are the direct analogy of interpreting a coefficient in multiple linear regression, but with taking into account the non linearity of the method. Xgboost doesn't support PDP, but 'gbm' does.

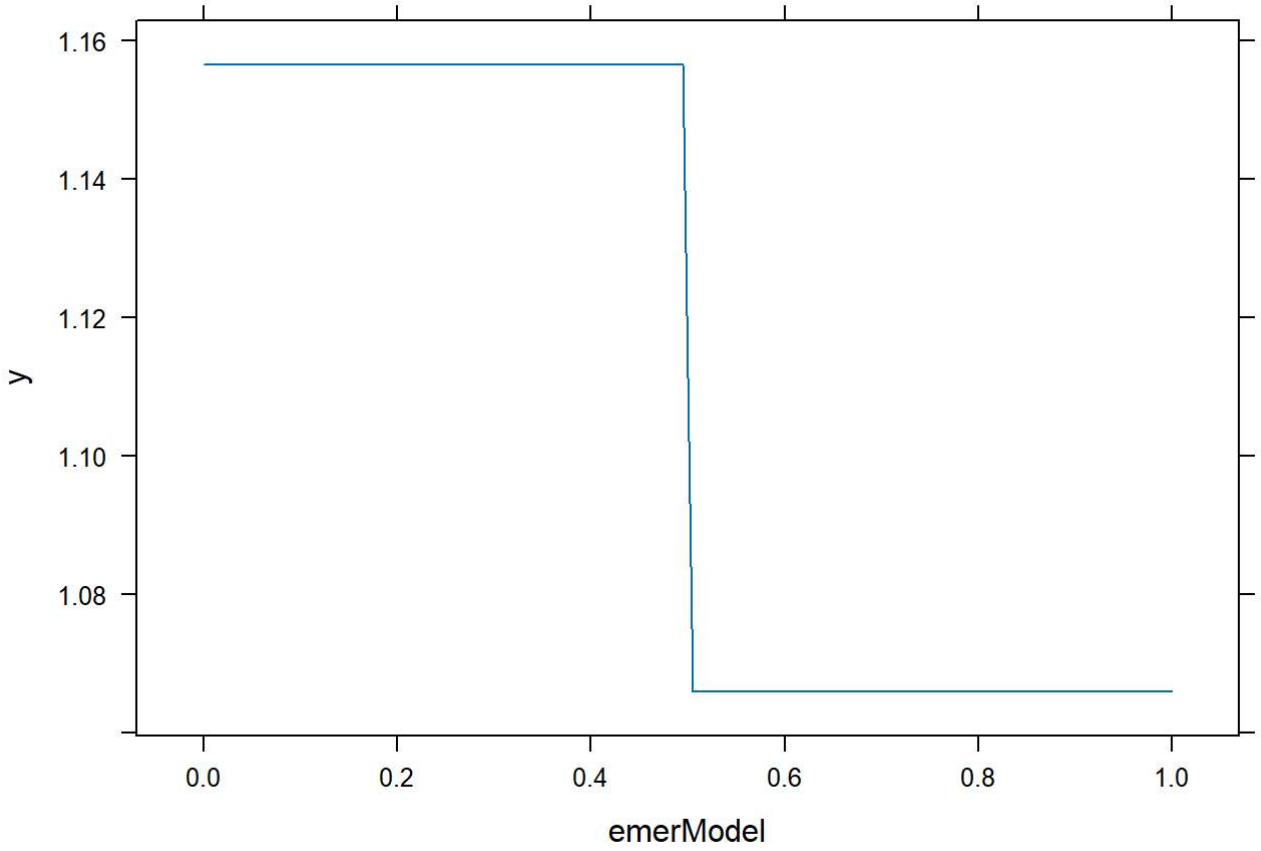
```
set.seed(1)
tuneGrid = expand.grid(interaction.depth = 6,
                       n.trees = c(50,150,500,1000,2000),
                       shrinkage = c(0.01, 0.1),
                       n.minobsinnode = 10)
gbmOut = train(x = Xtrain, y = Ytrain,
               method = "gbm", bag.fraction = 0.5,
               tuneGrid = tuneGrid,
               verbose = FALSE  #gbm produces a lot of output
               )
```

Let's compare PDP to the coefficient estimates from the elastic net model

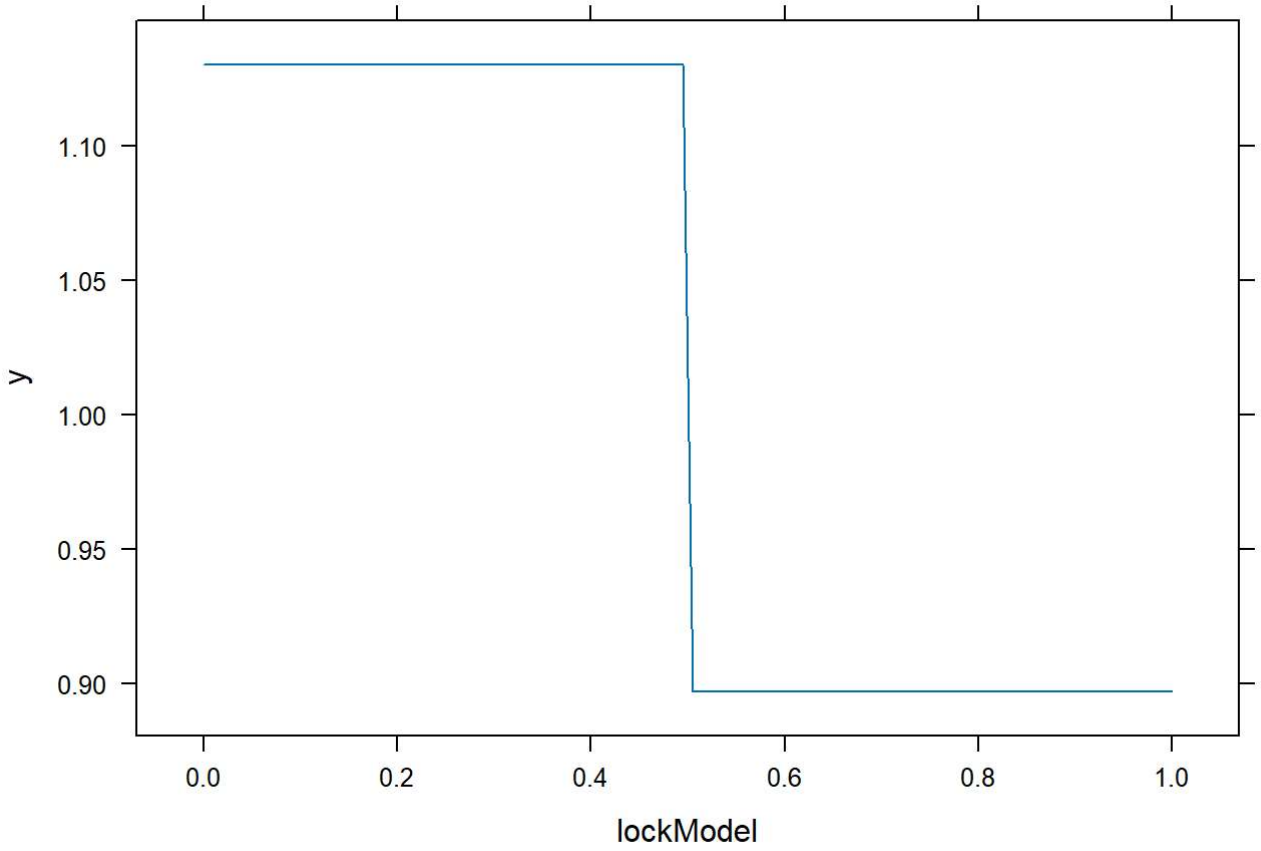
```
require(gbm)
#par(mfrow=c(1,2))
plot(Xtrain$emerModel,Ytrain,type='n')
abline(a = betaHatGlmnet[1], b = betaHatGlmnet[4])
```



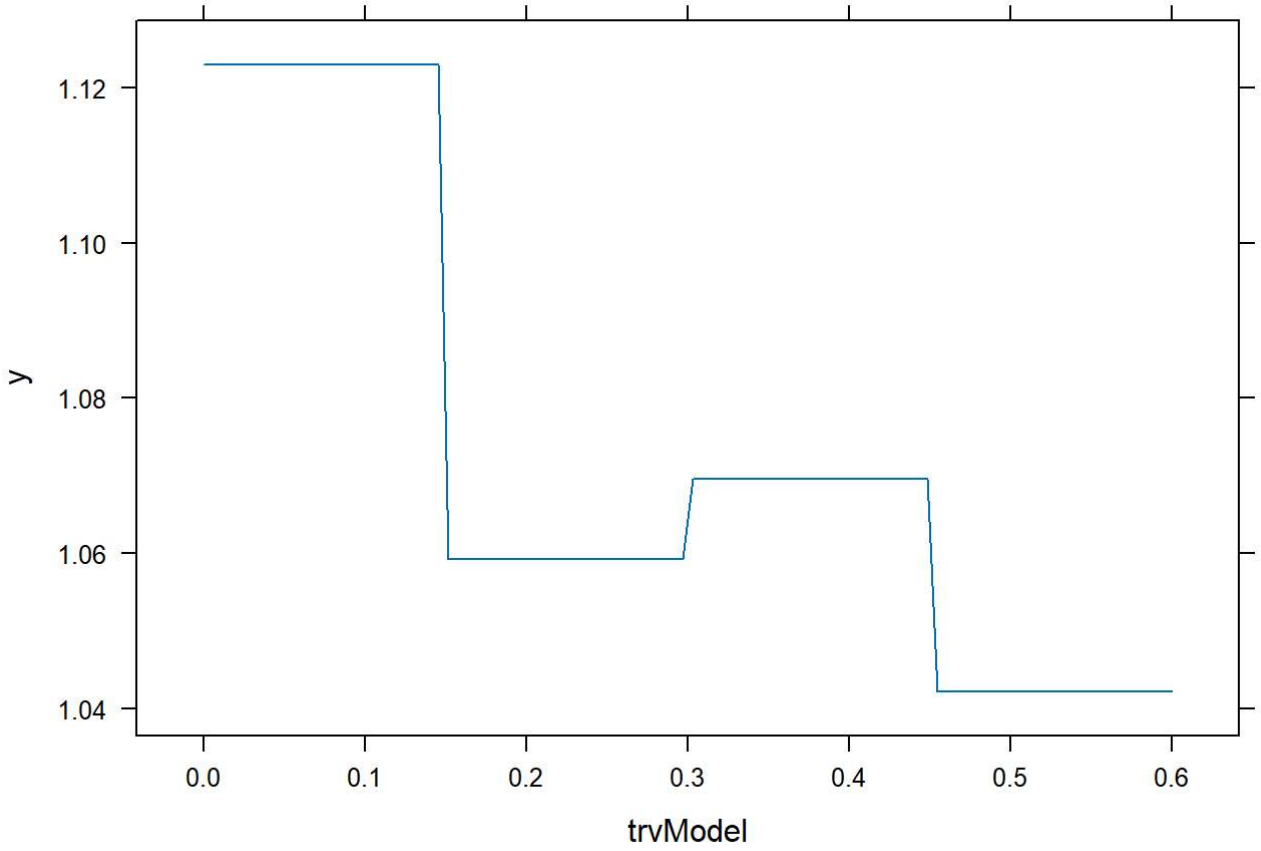

```
plot(gbmOut$finalModel,i.var=1)
```



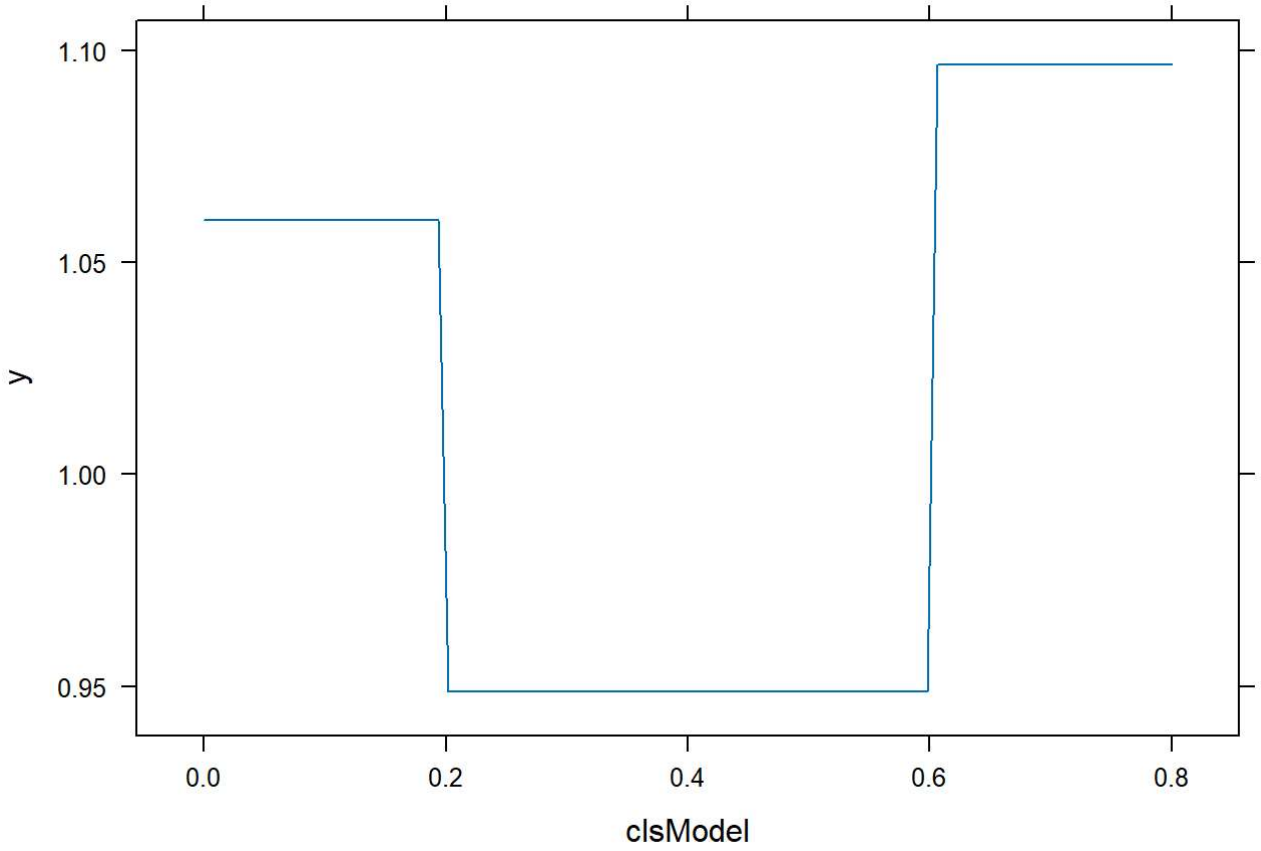

```
plot(gbmOut$finalModel,i.var=2)
```




```
plot(gbmOut$finalModel,i.var=3)
```



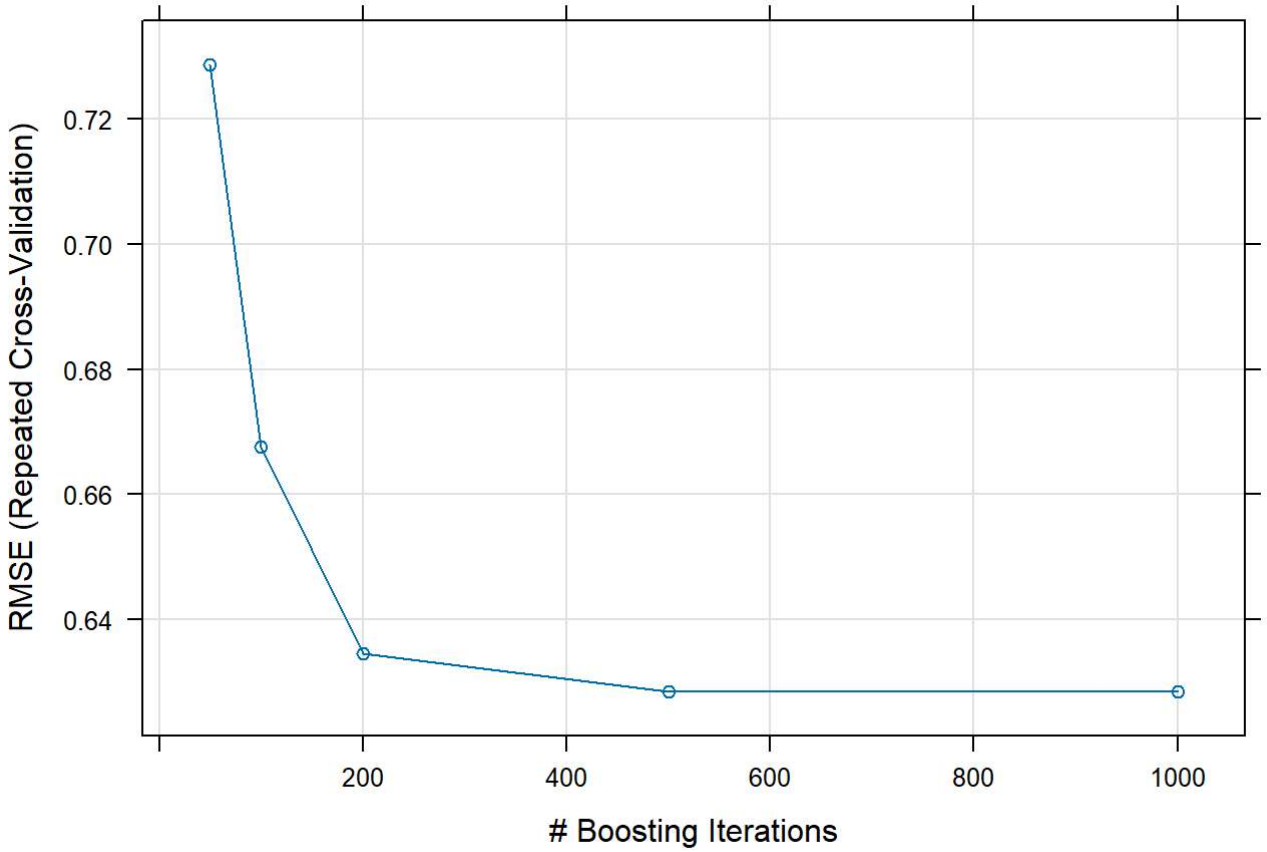

```
plot(gbmOut$finalModel,i.var=4)
```




# What is a good mixture?

Let’s look at the boosting implementation

```
set.seed(1)
tuneGrid = data.frame('nrounds'=c(50, 100,200,500,1000),
                      'max_depth' = 2,
                      'eta' = .01,
                      'gamma' = 0,
                      'colsample_bytree' = 1,
                      'min_child_weight' = 0,
                      'subsample' = .5)
boostOut = train(x = Xtot, y = Ytot,
                 method = "xgbTree",
                 tuneGrid = tuneGrid,
                 trControl = trControl,
                 verbosity = 0)

YhatBoost = predict(boostOut, Xtest)

plot(boostOut)
```




```
YhatBoost  = predict(boostOut, Xtot)
Xtot[which.max(YhatBoost),]
```

```
##   emerModel lockModel trvModel clsModel
## 1         0         0        0        0
```

```
max(YhatBoost)
```

```
## [1] 1.245829
```

# Finding new mixtures

```
objectiveF = function(X){
  X = as.data.frame(matrix(X,nrow = 1))
  names(X) = names(Xtot)
  return(-predict(boostOut,X))
}
```

```
out = optim(Xtot[which.max(YhatBoost),], objectiveF)
out
```

```
## $par
## emerModel lockModel  trvModel  clsModel
##         0         0         0         0
##
## $value
## [1] -1.245829
##
## $counts
## function gradient
##        5       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

#stop parallel computing

```
stopCluster(cl)
registerDoSEQ()
```