Equity Premium - Prediction

Druce Vertes

July 2017

Initialize

- Libraries to use

```
options(java.parameters='Xmx5g')
library(plyr)
library(reshape2)
library(lattice)
library(ggplot2)
library(MASS)
library(caret)
library(mlbench)
library(rpart)
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
##
##     melanoma
```

Import and clean data

- Import from CSV

```
setwd("C:/Users/druce/R/EquityPremium")
data<-read.csv('PredictorData2015q.csv',na.strings = c("NA","#DIV/0!", "","NaN"))
```

Clean. . . Trim NA valued columns

```
countMissing <- function(mycol) {
  return (sum(is.na(data[, mycol]))/ nrow(data))
}
countNAs <- data.frame(countNA=sapply(colnames(data), countMissing))
subset(countNAs, countNAs$countNA > 0.5)
```

```
##       countNA
## cay 0.5586207
## csp 0.5465517
## ik  0.5241379
## D3  0.8068966
```

```
colsToDeleteNA <- countNAs$countNA > 0.5
data <- data[, !colsToDeleteNA]
```

Clean… Trim NA valued rows

```
rowsToDelete <- data$yyyyq <= 19254
data <- data[!rowsToDelete,]
```

Add EqPrem column, numeric date column for charts

```
data$EqPrem = data$CRSP_SPvw - data$Rfree
data$numdate = as.numeric(substring(data$yyyyq, 1,4))+as.numeric(substring(data$yyyyq, 5,5))/4
```

```
# functions to do leads and lags
mylag <- function(v, n){
  c(rep(NA, n),v[(seq(length(v)-n))])
}

mylead <- function(v, n){
  c(v[-n], rep(NA, n))
}

data$EqPrem = mylead(data$EqPrem,1)
```

Create a big data frame including all predictors, first diffs lagged up to 2 quarters

```
#keep 12 predictors plus EqPrem
# truncate last quarter, no EqPrem to predict
data2 <- data[1:359,c("D12","E12","b.m","tbl","AAA","BAA","lty","ntis","infl","ltr","corpr","svar","EqP

# use trailing 1 year inflation
# should really do cum product of 1+infl , 70s/80s compounding would have made small difference
rsum.cumsum <- function(x, n = 4L) {
  tail(cumsum(x) - cumsum(c(rep(0, n), head(x, -n))), -n + 1)
}

# use real long term yields sted nominal
data2$infl <- tail(c(rep(NA,3), rsum.cumsum(data$infl)), 359)
data2$AAA <- data2$AAA - data2$infl
data2$BAA <- data2$BAA - data2$infl
data2$lty <- data2$lty - data2$infl

# compute first diffs
diffs <- tail(data2, -1) - head(data2, -1)
diffs <- diffs[complete.cases(diffs),]

# truncate oldest 2 qs, no trailing diffs
bigdata <- tail(data2,-2)

# truncate oldest q
diffs <- tail(diffs,-1)
diffs <- diffs[,c("D12","E12","b.m", "tbl","AAA","BAA","lty","ntis","infl","ltr","corpr","svar")]
names(diffs)<-c("D12.diff","E12.diff","b.m.diff","tbl.diff","AAA.diff","BAA.diff","lty.diff","ntis.diff
bigdata=merge(bigdata, diffs,by=0)
```

```r
# add previous quarter's 1st diff for tbl, AAA, BAA, lty, ltr, corpr
# compute first diffs
diffs <- tail(data2, -1) - head(data2, -1)
diffs <- head(diffs, -1)
diffs <- diffs[,c("tbl","AAA","BAA","lty","ltr","corpr")]
names(diffs)<-c("tbl.lagdiff","AAA.lagdiff","BAA.lagdiff","lty.lagdiff","ltr.lagdiff","corpr.lagdiff")
bigdata$rownums=1:nrow(bigdata)
diffs$rownums=1:nrow(diffs)
bigdata=merge(bigdata, diffs,by="rownums")

colsToDelete = names(bigdata) %in% c("Row.names", "rownums")
bigdata <- bigdata[,!colsToDelete]

# truncate oldest 2q, no ntis diff
bigdata <- tail(bigdata,-2)
```

Run models

Run a linear model

```r
fit <- lm(EqPrem~., data=bigdata)
summary(fit) # show results
```

```
##
## Call:
## lm(formula = EqPrem ~ ., data = bigdata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40547 -0.04502  0.00723  0.04945  0.62624
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.004810   0.029676  -0.162  0.87135
## D12             0.003733   0.002989   1.249  0.21270
## E12            -0.001480   0.001228  -1.205  0.22924
## b.m             0.142499   0.037742   3.776  0.00019 ***
## tbl             0.255291   0.628620   0.406  0.68493
## AAA             4.681644   3.706974   1.263  0.20753
## BAA            -3.185591   1.826102  -1.744  0.08203 .
## lty            -2.175096   2.394202  -0.908  0.36430
## ntis           -0.953382   0.325359  -2.930  0.00363 **
## infl           -1.209049   0.734305  -1.647  0.10064
## ltr            -0.166464   0.601717  -0.277  0.78223
## corpr           1.350825   0.565446   2.389  0.01747 *
## svar            0.215472   0.778638   0.277  0.78217
## D12.diff        0.025987   0.035443   0.733  0.46398
## E12.diff        0.007989   0.002710   2.948  0.00344 **
## b.m.diff        0.087582   0.088118   0.994  0.32102
## tbl.diff       -0.955060   0.945247  -1.010  0.31307
## AAA.diff        1.878845   5.088644   0.369  0.71220
```

```
## BAA.diff       6.202014   2.700882   2.296  0.02230 *
## lty.diff      -1.266921   5.399494  -0.235  0.81464
## ntis.diff      0.174851   0.752454   0.232  0.81640
## infl.diff      7.344698   5.090144   1.443  0.15002
## ltr.diff      -0.074361   0.300636  -0.247  0.80480
## corpr.diff    -0.372279   0.335495  -1.110  0.26798
## svar.diff     -0.183968   0.633211  -0.291  0.77160
## tbl.lagdiff   -0.270609   0.798579  -0.339  0.73493
## AAA.lagdiff    8.476419   4.112842   2.061  0.04011 *
## BAA.lagdiff   -5.312101   1.960281  -2.710  0.00709 **
## lty.lagdiff   -3.843309   3.253168  -1.181  0.23832
## ltr.lagdiff   -0.090060   0.213760  -0.421  0.67381
## corpr.lagdiff -0.007068   0.213117  -0.033  0.97356
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1061 on 321 degrees of freedom
## Multiple R-squared:  0.2026, Adjusted R-squared:  0.128
## F-statistic: 2.718 on 30 and 321 DF,  p-value: 7.931e-06
```

```r
#plot(fit)
```

Run a stepwise regression for variable selection

```r
library(MASS)

step <- stepAIC(fit, direction="both")
```

```r
step$anova # display results
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## EqPrem ~ D12 + E12 + b.m + tbl + AAA + BAA + lty + ntis + infl +
##     ltr + corpr + svar + D12.diff + E12.diff + b.m.diff + tbl.diff +
##     AAA.diff + BAA.diff + lty.diff + ntis.diff + infl.diff +
##     ltr.diff + corpr.diff + svar.diff + tbl.lagdiff + AAA.lagdiff +
##     BAA.lagdiff + lty.lagdiff + ltr.lagdiff + corpr.lagdiff
##
## Final Model:
## EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl + corpr +
##     E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
##     BAA.lagdiff
##
##
##              Step Df     Deviance Resid. Df Resid. Dev       AIC
## 1                                       321   3.614009 -1549.742
## 2  - corpr.lagdiff  1 1.238426e-05       322   3.614021 -1551.741
## 3      - ntis.diff  1 6.071146e-04       323   3.614628 -1553.682
## 4       - ltr.diff  1 5.661556e-04       324   3.615195 -1555.627
## 5       - lty.diff  1 5.134941e-04       325   3.615708 -1557.577
## 6      - svar.diff  1 7.594287e-04       326   3.616467 -1559.503
```

```
## 7           - svar  1 5.243009e-04       327   3.616992 -1561.452
## 8    - tbl.lagdiff  1 9.724069e-04       328   3.617964 -1563.357
## 9       - AAA.diff  1 1.499855e-03       329   3.619464 -1565.211
## 10           - tbl  1 1.433615e-03       330   3.620898 -1567.072
## 11           - ltr  1 4.264306e-03       331   3.625162 -1568.658
## 12      - D12.diff  1 3.768510e-03       332   3.628930 -1570.292
## 13      - b.m.diff  1 6.373825e-03       333   3.635304 -1571.674
## 14   - ltr.lagdiff  1 9.155316e-03       334   3.644460 -1572.789
## 15   - lty.lagdiff  1 7.272834e-03       335   3.651732 -1574.087
## 16      - tbl.diff  1 1.452022e-02       336   3.666253 -1574.690
## 17           - lty  1 1.497761e-02       337   3.681230 -1575.255
```

Run a model, with just the useful predictors

Slightly lower R-squared, higher adjusted R-squared

```
fit2<-lm(EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl + corpr +
    E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
    BAA.lagdiff, data=bigdata)
summary(fit2) # show results
```

```
##
## Call:
## lm(formula = EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl +
##     corpr + E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
##     BAA.lagdiff, data = bigdata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.41476 -0.04490  0.00556  0.05016  0.61774
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0129848  0.0265728  -0.489 0.625407
## D12          0.0037127  0.0021676   1.713 0.087669 .
## E12         -0.0010391  0.0007359  -1.412 0.158872
## b.m          0.1499120  0.0348112   4.306 2.18e-05 ***
## AAA          2.0166543  1.2404174   1.626 0.104930
## BAA         -2.4754532  1.1768969  -2.103 0.036174 *
## ntis        -0.7823446  0.2776333  -2.818 0.005119 **
## infl        -1.0142144  0.2374272  -4.272 2.53e-05 ***
## corpr        1.2809540  0.2582765   4.960 1.12e-06 ***
## E12.diff     0.0065916  0.0019365   3.404 0.000745 ***
## BAA.diff     7.3436719  1.5860955   4.630 5.23e-06 ***
## infl.diff    7.8938213  1.6750942   4.712 3.59e-06 ***
## corpr.diff  -0.4210623  0.1393510  -3.022 0.002707 **
## AAA.lagdiff  3.9888395  1.8693423   2.134 0.033579 *
## BAA.lagdiff -4.6794150  1.7882072  -2.617 0.009275 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1045 on 337 degrees of freedom
## Multiple R-squared:  0.1877, Adjusted R-squared:  0.154
```

```
## F-statistic: 5.564 on 14 and 337 DF,  p-value: 1.124e-09
```

```r
#plot(fit2)
```

Run an out of sample test

TODO: don't select the variables using the whole set, which is bad practice/cheating

```r
# test set v. training set
# sample(1000,1)
set.seed(710)

trainindex <- sample(nrow(bigdata), trunc(nrow(bigdata)*.75))
trainingset <- bigdata[trainindex,]
testset <- bigdata[-trainindex, ]

fit3<-lm(EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl + corpr +
    E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
    BAA.lagdiff, data=trainingset)
summary(fit3) # show results
```

```
##
## Call:
## lm(formula = EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl +
##     corpr + E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
##     BAA.lagdiff, data = trainingset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42968 -0.05169  0.00434  0.04671  0.44382
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0022471  0.0281245   0.080 0.936383
## D12          0.0052819  0.0025201   2.096 0.037101 *
## E12         -0.0016198  0.0008519  -1.901 0.058405 .
## b.m          0.1508326  0.0379532   3.974 9.26e-05 ***
## AAA          2.6361540  1.3332986   1.977 0.049126 *
## BAA         -3.2571519  1.2599596  -2.585 0.010304 *
## ntis        -0.9968195  0.2939358  -3.391 0.000809 ***
## infl        -1.1586336  0.2566480  -4.514 9.80e-06 ***
## corpr        1.5687385  0.2801824   5.599 5.68e-08 ***
## E12.diff     0.0149032  0.0028250   5.276 2.88e-07 ***
## BAA.diff    10.4730928  1.6674033   6.281 1.49e-09 ***
## infl.diff   10.0450859  1.7631369   5.697 3.43e-08 ***
## corpr.diff  -0.4499933  0.1559687  -2.885 0.004255 **
## AAA.lagdiff -3.3584476  2.6152019  -1.284 0.200265
## BAA.lagdiff  2.9035611  2.5988682   1.117 0.264968
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09754 on 249 degrees of freedom
## Multiple R-squared:  0.2553, Adjusted R-squared:  0.2135
```

```
## F-statistic: 6.098 on 14 and 249 DF,  p-value: 2.244e-10
```

```r
# R-squared goes up when all we did was reduce the sample size
# suggests overfitting

# in-sample RMSE
mdss <- function (var1, var2) {
  mean((var1 - var2)^2)
}

MSEis <- mdss(predict(fit3), trainingset$EqPrem)
print(sqrt(MSEis))
```

```
## [1] 0.09473303
```

```r
# in-sample population standard dev
print(sqrt(mdss(trainingset$EqPrem, mean(trainingset$EqPrem))))
```

```
## [1] 0.1097792
```

```r
# check vs. sd function
sqrt((sd(trainingset$EqPrem))^2 * (nrow(trainingset)-1) / nrow(trainingset))
```

```
## [1] 0.1097792
```

```r
# bigdata population standard dev
print(sqrt(mdss(bigdata$EqPrem, mean(bigdata$EqPrem))))
```

```
## [1] 0.1134688
```

```r
# if out-of-sample RMSE is better than those we are probably predicting something

# in-sample MSE / Population Variance = R-squared  (as a check)
print(1- MSEis / mdss(trainingset$EqPrem, mean(trainingset$EqPrem)))
```

```
## [1] 0.2553324
```

```r
# out-of-sample RMSE
mypredict <- predict(fit3, newdata = testset)
MSEos <- mdss(mypredict, testset$EqPrem)
print(sqrt(MSEos))
```

```
## [1] 0.1450144
```

```r
# suppose we just used the mean of training set as predictor, RMSE would be
print(sqrt(mdss(testset$EqPrem, mean(trainingset$EqPrem))))
```

```
## [1] 0.1239499
```

```r
# our model predicts worse out-of-sample than just using the training set mean (!)
#print("out-of-sample MSE / Variance") # out-of-sample R-squared maybe different
#print("not sure what is correct out-of-sample R-squared formula but")
#print(1- MSEos / mean((testset$EqPrem - mean(trainingset$EqPrem))^2))
#print(1- MSEos / mean((testset$EqPrem - mean(testset$EqPrem))^2))

# leave one out cross-validation
# glm same as lm but supports cross-validation

glm.fit <- glm(EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl + corpr +
             E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
```

```r
              BAA.lagdiff, data=bigdata)
# same as fit2
summary(glm.fit)
```

```
##
## Call:
## glm(formula = EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl +
##     corpr + E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
##     BAA.lagdiff, data = bigdata)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.41476  -0.04490   0.00556   0.05016   0.61774
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0129848  0.0265728  -0.489 0.625407
## D12          0.0037127  0.0021676   1.713 0.087669 .
## E12         -0.0010391  0.0007359  -1.412 0.158872
## b.m          0.1499120  0.0348112   4.306 2.18e-05 ***
## AAA          2.0166543  1.2404174   1.626 0.104930
## BAA         -2.4754532  1.1768969  -2.103 0.036174 *
## ntis        -0.7823446  0.2776333  -2.818 0.005119 **
## infl        -1.0142144  0.2374272  -4.272 2.53e-05 ***
## corpr        1.2809540  0.2582765   4.960 1.12e-06 ***
## E12.diff     0.0065916  0.0019365   3.404 0.000745 ***
## BAA.diff     7.3436719  1.5860955   4.630 5.23e-06 ***
## infl.diff    7.8938213  1.6750942   4.712 3.59e-06 ***
## corpr.diff  -0.4210623  0.1393510  -3.022 0.002707 **
## AAA.lagdiff  3.9888395  1.8693423   2.134 0.033579 *
## BAA.lagdiff -4.6794150  1.7882072  -2.617 0.009275 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.01092353)
##
##     Null deviance: 4.5321  on 351  degrees of freedom
## Residual deviance: 3.6812  on 337  degrees of freedom
## AIC: -574.32
##
## Number of Fisher Scoring iterations: 2
```

```r
cv.err <- cv.glm(bigdata, glm.fit)

print("Leave one out cross-validation RMSE")
```

```
## [1] "Leave one out cross-validation RMSE"
```

```r
MSEos <- cv.err$delta[1]
print(sqrt(MSEos))
```

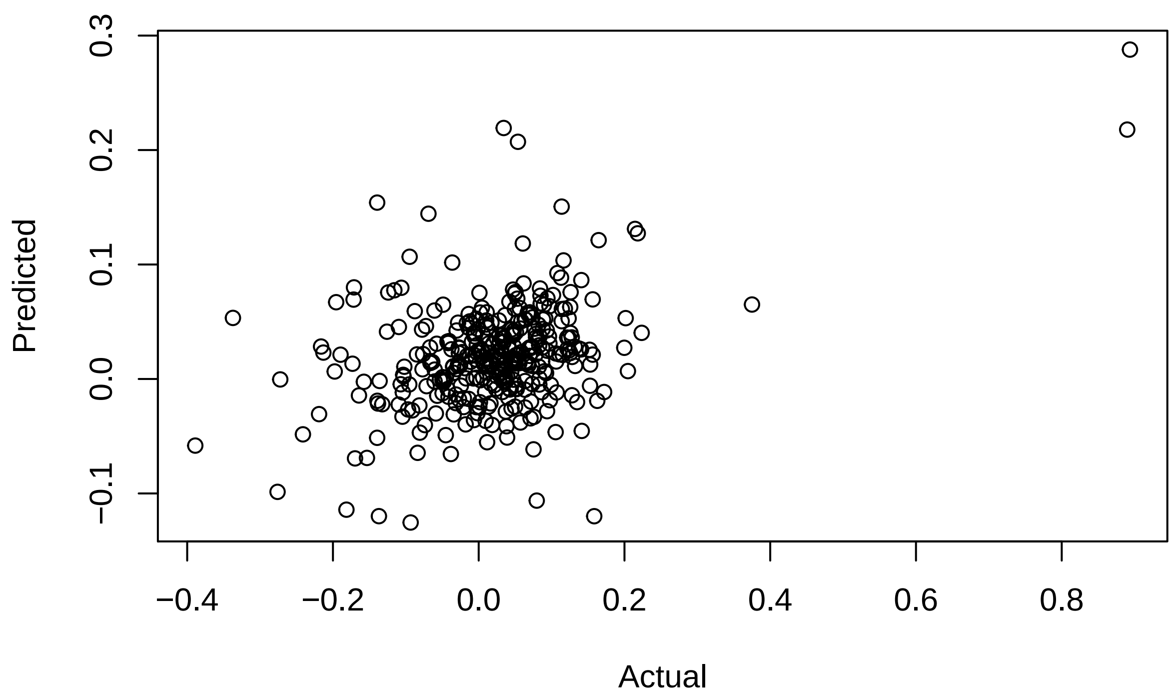
```
## [1] 0.116313
```

```r
# larger than in-sample RMSE which makes sense
# smaller than OOSE RMSE we found with training/test 75%/25% which makes sense
# smaller than RMSE we get just using the mean of the training set
```

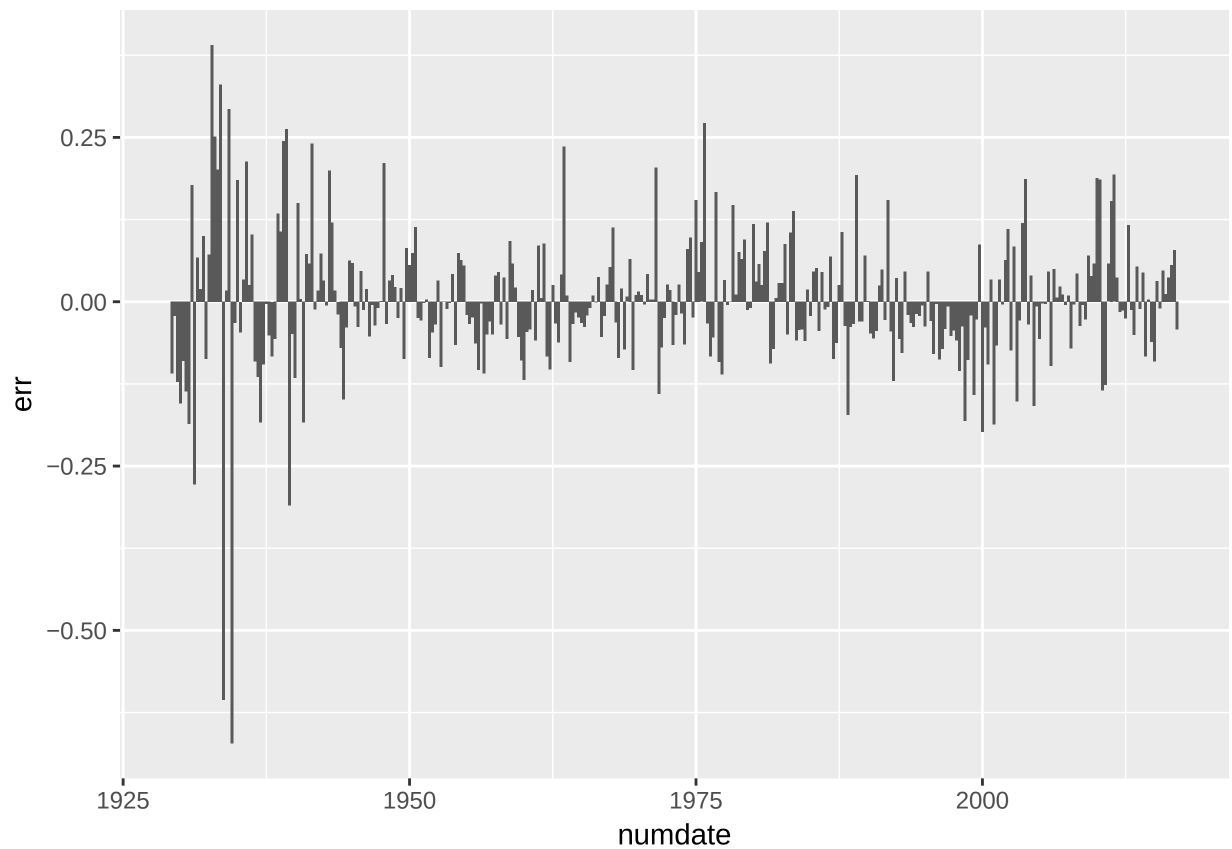
```r
# so, if you leave one out, estimate model on remainder, test on one you left out,
# error is a little smaller than just using a constant
# a wee bit but not much useful prediction going on

#print("LOOCV MSE / Variance") # out-of-sample R-squared maybe different
#print(1- MSEos / mean((bigdata$EqPrem - mean(bigdata$EqPrem))^2))
```

Plot predicted vs. actual

```r
# scatter plot
plotframe=data.frame(bigdata$EqPrem, fitted(fit2))
plot(plotframe, ylab="Predicted", xlab="Actual")
```




```r
## error plot
plotframe$numdate <- tail(data$numdate, 352)
plotframe$err <- plotframe$fitted.fit2. - plotframe$bigdata.EqPrem
ggplot(data=plotframe, aes(x=numdate, y=err)) + geom_bar(stat="identity")
```




```
## bars since 1974
plotframe2 = plotframe[plotframe$numdate > 2000, c("bigdata.EqPrem", "fitted.fit2." ,  "numdate") ]
plotframe3 = melt(plotframe2,id="numdate")
ggplot(plotframe3, aes(x=numdate, y=value, fill=variable)) + geom_bar(stat="identity", position="dodge")
```

Run caret regression models

- Observe OOS RMSE with various nonlinear models v. linear model

```
library(frbs)
library(pls)
```

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

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

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

```
library(monomvn)
```

```
## Loading required package: lars

## Loaded lars 1.2
```

```
library(elasticnet)
library(foba)
library(fastICA)
```

```r
library(kernlab)
```

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

```r
library(KRLS)
```

```
## ## KRLS Package for Kernel-based Regularized Least Squares.
## ## See Hainmueller and Hazlett (2014) for details.
```

```r
library(lars)
library(neuralnet)
library(nnls)
library(leaps)

# use same as before
trainingset <- bigdata[trainindex,]
testset <- bigdata[-trainindex, ]

# these returned valid values at one time, maybe a version hell situation, subsequently loaded package
# "lars" "lasso", "neuralnet", 'rqlasso', , 'superpc', , 'lasso', "krlsRadial", "krlsPoly", , "rlm"", '
# , "lmStepAIC" # this one just generates too much annoying output

regressionMethods <- c("lm", "enet", "leapBackward", "leapForward","leapSeq",
                       "nnls", "pcr", 'rvmLinear', 'rvmRadial', 'ridge'
                       )

regressionModels <- array(1:length(regressionMethods))
regressionTrainPredicts <- data.frame(row.names=row.names(trainingset))
regressionTestPredicts <- data.frame(row.names=row.names(testset))

print("Out of sample RMSE using various methods")
```

```
## [1] "Out of sample RMSE using various methods"
```

```r
# trc_cv = trainControl(method="cv")
i <- 0
for(mx in regressionMethods) {
  i <- i + 1
  print(mx)
  mymodel <- train(EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl + corpr +
    E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
    BAA.lagdiff, data=trainingset, method=mx, preProc = c("center", "scale"), verbose=FALSE)

  mypredict <- predict(mymodel, newdata = testset)
  MSEos <- mdss(mypredict, testset$EqPrem)
  print(sqrt(MSEos))

  regressionModels[i] <- mymodel
  regressionTrainPredicts[, mx] <- predict(mymodel, newdata=trainingset)
  regressionTestPredicts[, mx] <- mypredict
```

```
}
```

```
## [1] "lm"
## [1] 0.1450144
## [1] "enet"
## [1] 0.122276
## [1] "leapBackward"
## [1] 0.1274167
## [1] "leapForward"
## [1] 0.1236317
## [1] "leapSeq"
## [1] 0.1236317
## [1] "nnls"
## [1] 0.1345647
## [1] "pcr"
## [1] 0.1239154
## [1] "rvmLinear"
## [1] 0.1366359
## [1] "rvmRadial"
## [1] 0.1321609
## [1] "ridge"
## [1] 0.1448538
```

the nonlinear methods do better, sometimes significantly better

- note lm model has same OOS RMSE as we found earlier, all the others are smaller

```
mx <- 'leapBackward'
mymodel <- train(EqPrem ~ D12 + E12 + b.m + AAA + BAA + ntis + infl + corpr +
    E12.diff + BAA.diff + infl.diff + corpr.diff + AAA.lagdiff +
    BAA.lagdiff, data=trainingset, method=mx, preProc = c("center", "scale"), verbose=FALSE)
mypredict <- predict(mymodel, newdata = testset)

MSEos <- mdss(mypredict, testset$EqPrem)
print("Out of sample RMSE")
```

```
## [1] "Out of sample RMSE"
```

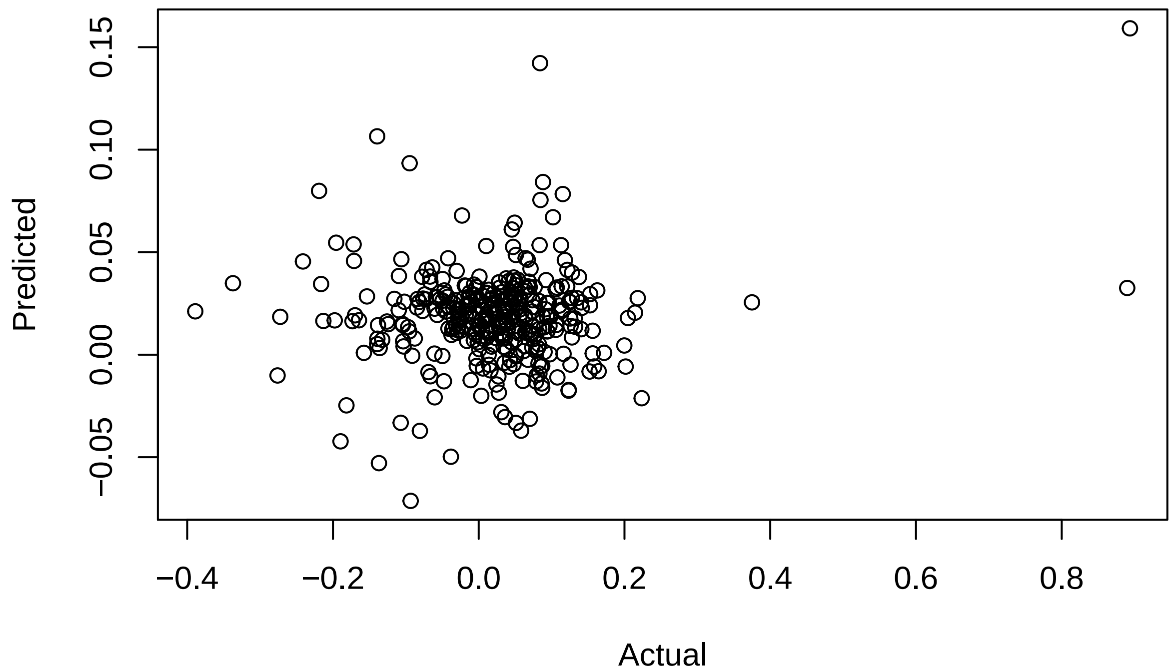
```
print(sqrt(MSEos))
```

```
## [1] 0.1274167
```

```
# suppose we just used the mean of training set as predictor, RMSE would be
print(sqrt(mdss(mean(trainingset$EqPrem), testset$EqPrem)))
```

```
## [1] 0.1239499
```

```
#print(1- MSEos / mean((testset$EqPrem - mean(trainingset$EqPrem))^2))
#print(1- MSEos / mean((testset$EqPrem - mean(testset$EqPrem))^2))

plotframe <- data.frame(bigdata$EqPrem, predict(mymodel, newdata = bigdata))

plot(plotframe, ylab="Predicted", xlab="Actual")
```



not good but at least a little more predictive than using the mean or linear model

```
# try preprocessing with PCA

print("Out of sample RMSE using various methods")
```

```
## [1] "Out of sample RMSE using various methods"
```

```
for(mx in regressionMethods) {

  trc_cv = trainControl(method="cv")

  print(mx)
  mymodel <- train(EqPrem ~ ., data=trainingset, method=mx, preProc = c("center", "scale", "pca"),
                   verbose=FALSE)
  mypredict <- predict(mymodel, newdata = testset)
  MSEos <- mean((mypredict - testset$EqPrem)^2)
  print(sqrt(MSEos))
}
```

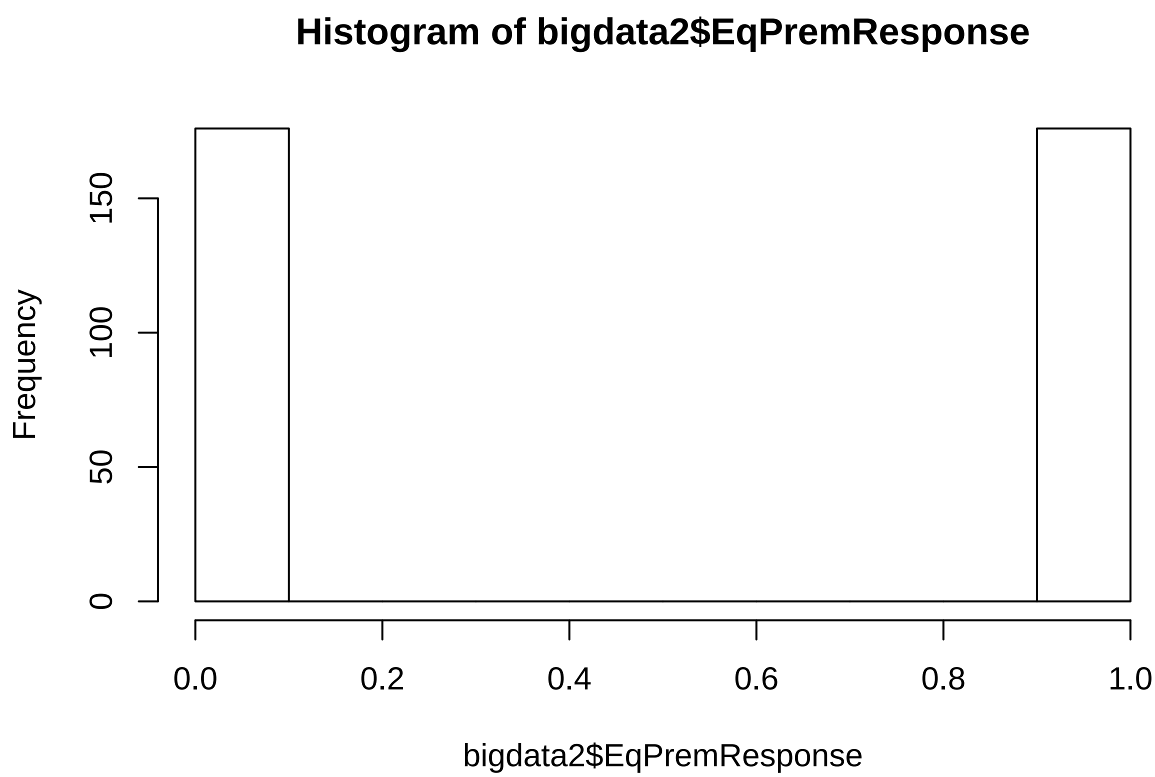
```
## [1] "lm"
## [1] 0.1349884
## [1] "enet"
## [1] 0.1233069
## [1] "leapBackward"
## [1] 0.1265972
```

```
## [1] "leapForward"
## [1] 0.1265972
## [1] "leapSeq"
## [1] 0.1265972
## [1] "nnls"
## [1] 0.1414556
## [1] "pcr"
## [1] 0.1250058
## [1] "rvmLinear"
## [1] 0.1273842
## [1] "rvmRadial"
## [1] 0.1283124
## [1] "ridge"
## [1] 0.1349884
```

no real help

- Run a binary classification model
- Create indicator for classification

```
bigdata2=bigdata
Z <- quantile(bigdata2$EqPrem, probs=c(0,0.5,1)) # really just need 0.5
bigdata2$EqPremResponse=1
bigdata2$EqPremResponse[bigdata$EqPrem < Z[2]] = 0
hist(bigdata2$EqPremResponse)
```



```
# some algos try todo regression instead of classification on numbers, or error
bigdata2$EqPremResponse <- as.factor(bigdata2$EqPremResponse)
bigdata2 = bigdata2[, !(colnames(bigdata2) == "EqPrem")]
```

Create training and test sets

```
# create training and test sets
# use same samples as earlier

trainingset <- bigdata2[trainindex,]
testset <- bigdata2[-trainindex, ]
```

Predict quantiles using a variety of algorithms

```
# "nnet", "pcaNNet", "stepLDA", "stepQDA" don't work great and generate pages of output

myMethods <- c("bartMachine", "deepboost", "gbm", "lda", "lda2", "LogitBoost", "multinom", "nb", "qda",

#myMethods <- c("lda")

trc_cv = trainControl(method="cv")

# center and scale for better performance on some methods
runModel <- function(mxpar) {
    return (train(EqPremResponse ~ ., data=trainingset, method=mxpar,
                  preProc = c("center", "scale"), verbose=FALSE))
}

for(mx in myMethods) {
  print(mx)
  mymodel = runModel(mx)

  print("Training set confusion matrix")
  myPredict <- data.frame(prediction=predict(mymodel, trainingset))
  myPredict$EqPremResponse<-trainingset$EqPremResponse
  print(confusionMatrix(myPredict$prediction, myPredict$EqPremResponse))

  print("Test set confusion matrix")
  myPredict <- data.frame(prediction=predict(mymodel, testset))
  myPredict$EqPremResponse<-testset$EqPremResponse
  print(confusionMatrix(myPredict$prediction, myPredict$EqPremResponse))
}
```

```
## [1] "bartMachine"

## Loading required package: bartMachine

## Loading required package: rJava

## Loading required package: bartMachineJARs

## Loading required package: car

##
## Attaching package: 'car'

## The following object is masked from 'package:boot':
##
```

```
##     logit
## Loading required package: randomForest
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
## Loading required package: missForest
## Loading required package: foreach
## Loading required package: itertools
## Loading required package: iterators
## Welcome to bartMachine v1.2.3! You have 3.82GB memory available.
##
## If you run out of memory, restart R, and use e.g.
## 'options(java.parameters = "-Xmx5g")' for 5GB of RAM before you call
## 'library(bartMachine)'.
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111  23
##          1  23 107
##
##                Accuracy : 0.8258
##                  95% CI : (0.7745, 0.8695)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6514
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8284
##             Specificity : 0.8231
##          Pos Pred Value : 0.8284
##          Neg Pred Value : 0.8231
##              Prevalence : 0.5076
##          Detection Rate : 0.4205
##    Detection Prevalence : 0.5076
##       Balanced Accuracy : 0.8257
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 21 21
##          1 21 25
##
##                Accuracy : 0.5227
##                  95% CI : (0.4135, 0.6304)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.5431
##
##                   Kappa : 0.0435
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.5000
##             Specificity : 0.5435
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.5435
##              Prevalence : 0.4773
##          Detection Rate : 0.2386
##    Detection Prevalence : 0.4773
##       Balanced Accuracy : 0.5217
##
##        'Positive' Class : 0
##
## [1] "deepboost"
```

```
## Loading required package: deepboost
```

```
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 134   1
##          1   0 129
##
##                Accuracy : 0.9962
##                  95% CI : (0.9791, 0.9999)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9924
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9923
##          Pos Pred Value : 0.9926
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5076
##          Detection Rate : 0.5076
##    Detection Prevalence : 0.5114
##       Balanced Accuracy : 0.9962
##
##        'Positive' Class : 0
##
```

```
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23 16
##          1 19 30
##
##                Accuracy : 0.6023
##                  95% CI : (0.4923, 0.7051)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.08227
##
##                   Kappa : 0.2004
##  Mcnemar's Test P-Value : 0.73532
##
##             Sensitivity : 0.5476
##             Specificity : 0.6522
##          Pos Pred Value : 0.5897
##          Neg Pred Value : 0.6122
##              Prevalence : 0.4773
##          Detection Rate : 0.2614
##    Detection Prevalence : 0.4432
##       Balanced Accuracy : 0.5999
##
##        'Positive' Class : 0
##
## [1] "gbm"

## Loading required package: gbm

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:deepboost':
##
##     heart

## The following object is masked from 'package:boot':
##
##     aml

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3

## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 113  19
##          1  21 111
##
##                Accuracy : 0.8485
##                  95% CI : (0.7994, 0.8895)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.697
##  Mcnemar's Test P-Value : 0.8744
##
##             Sensitivity : 0.8433
##             Specificity : 0.8538
##          Pos Pred Value : 0.8561
##          Neg Pred Value : 0.8409
##              Prevalence : 0.5076
##          Detection Rate : 0.4280
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8486
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 28 25
##          1 14 21
##
##                Accuracy : 0.5568
##                  95% CI : (0.447, 0.6627)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.2974
##
##                   Kappa : 0.1218
##  Mcnemar's Test P-Value : 0.1093
##
##             Sensitivity : 0.6667
##             Specificity : 0.4565
##          Pos Pred Value : 0.5283
##          Neg Pred Value : 0.6000
##              Prevalence : 0.4773
##          Detection Rate : 0.3182
##    Detection Prevalence : 0.6023
##       Balanced Accuracy : 0.5616
##
##        'Positive' Class : 0
##
## [1] "lda"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  0  1
##          0 87 47
##          1 47 83
##
##                Accuracy : 0.6439
##                  95% CI : (0.5829, 0.7017)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 5.369e-06
##
##                   Kappa : 0.2877
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.6493
##             Specificity : 0.6385
##          Pos Pred Value : 0.6493
##          Neg Pred Value : 0.6385
##              Prevalence : 0.5076
##          Detection Rate : 0.3295
##    Detection Prevalence : 0.5076
##       Balanced Accuracy : 0.6439
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 29 23
##          1 13 23
##
##                Accuracy : 0.5909
##                  95% CI : (0.4809, 0.6946)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.1200
##
##                   Kappa : 0.1885
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.6905
##             Specificity : 0.5000
##          Pos Pred Value : 0.5577
##          Neg Pred Value : 0.6389
##              Prevalence : 0.4773
##          Detection Rate : 0.3295
##    Detection Prevalence : 0.5909
##       Balanced Accuracy : 0.5952
##
##        'Positive' Class : 0
##
## [1] "lda2"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 87 47
##          1 47 83
##
##                Accuracy : 0.6439
##                  95% CI : (0.5829, 0.7017)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 5.369e-06
##
##                   Kappa : 0.2877
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.6493
##             Specificity : 0.6385
##          Pos Pred Value : 0.6493
##          Neg Pred Value : 0.6385
##              Prevalence : 0.5076
##          Detection Rate : 0.3295
##    Detection Prevalence : 0.5076
##       Balanced Accuracy : 0.6439
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 29 23
##          1 13 23
##
##                Accuracy : 0.5909
##                  95% CI : (0.4809, 0.6946)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.1200
##
##                   Kappa : 0.1885
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.6905
##             Specificity : 0.5000
##          Pos Pred Value : 0.5577
##          Neg Pred Value : 0.6389
##              Prevalence : 0.4773
##          Detection Rate : 0.3295
##    Detection Prevalence : 0.5909
##       Balanced Accuracy : 0.5952
##
##        'Positive' Class : 0
##
## [1] "LogitBoost"

## Loading required package: caTools
```

```
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 118  30
##          1  16 100
##
##                Accuracy : 0.8258
##                  95% CI : (0.7745, 0.8695)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6509
##  Mcnemar's Test P-Value : 0.05527
##
##             Sensitivity : 0.8806
##             Specificity : 0.7692
##          Pos Pred Value : 0.7973
##          Neg Pred Value : 0.8621
##              Prevalence : 0.5076
##          Detection Rate : 0.4470
##    Detection Prevalence : 0.5606
##       Balanced Accuracy : 0.8249
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 25 22
##          1 17 24
##
##                Accuracy : 0.5568
##                  95% CI : (0.447, 0.6627)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.2974
##
##                   Kappa : 0.1164
##  Mcnemar's Test P-Value : 0.5218
##
##             Sensitivity : 0.5952
##             Specificity : 0.5217
##          Pos Pred Value : 0.5319
##          Neg Pred Value : 0.5854
##              Prevalence : 0.4773
##          Detection Rate : 0.2841
##    Detection Prevalence : 0.5341
##       Balanced Accuracy : 0.5585
##
##        'Positive' Class : 0
##
```

```
## [1] "multinom"

## Loading required package: nnet

## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.060664
## iter  20 value 153.846482
## iter  30 value 150.720715
## iter  40 value 150.673709
## final  value 150.673412
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.644997
## iter  20 value 157.101126
## iter  30 value 156.520714
## final  value 156.520662
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.061289
## iter  20 value 153.851826
## iter  30 value 150.734416
## iter  40 value 150.688042
## final  value 150.687754
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 146.493403
## iter  20 value 134.922537
## iter  30 value 131.114321
## iter  40 value 131.082686
## final  value 131.082646
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 146.967044
## iter  20 value 138.832496
## iter  30 value 137.959385
## final  value 137.959375
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 146.493892
## iter  20 value 134.928573
## iter  30 value 131.131597
## iter  40 value 131.100479
## final  value 131.100440
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.428843
## iter  20 value 152.487363
## iter  30 value 148.678403
```

```
## iter  40 value 148.398757
## final  value 148.398688
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.945809
## iter  20 value 155.907756
## iter  30 value 155.333013
## final  value 155.332935
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.429391
## iter  20 value 152.493152
## iter  30 value 148.696368
## iter  40 value 148.421311
## final  value 148.421245
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 147.267547
## iter  20 value 137.121351
## iter  30 value 135.408629
## iter  40 value 135.302072
## final  value 135.302068
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 148.021710
## iter  20 value 140.783204
## iter  30 value 140.260608
## final  value 140.260566
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 147.268350
## iter  20 value 137.126516
## iter  30 value 135.418238
## iter  40 value 135.313125
## final  value 135.313121
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.974997
## iter  20 value 138.600045
## iter  30 value 134.989135
## iter  40 value 134.668608
## final  value 134.668545
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 154.763581
## iter  20 value 145.261503
## iter  30 value 144.754643
```

```
## final  value 144.754554
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.978170
## iter  20 value 138.609908
## iter  30 value 135.011168
## iter  40 value 134.695976
## final  value 134.695915
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.556674
## iter  20 value 144.853510
## iter  30 value 140.161405
## iter  40 value 140.067391
## final  value 140.065881
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.910122
## iter  20 value 147.213222
## iter  30 value 146.722335
## final  value 146.722324
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.557036
## iter  20 value 144.856973
## iter  30 value 140.190720
## iter  40 value 140.098263
## final  value 140.096810
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 154.608909
## iter  20 value 142.865105
## iter  30 value 137.851097
## iter  40 value 137.621880
## final  value 137.620620
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 155.262478
## iter  20 value 147.010812
## iter  30 value 146.165877
## final  value 146.165768
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 154.609598
## iter  20 value 142.872391
## iter  30 value 137.874038
## iter  40 value 137.647936
```

```
## final  value 137.646720
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.046977
## iter  20 value 143.988405
## iter  30 value 140.601247
## iter  40 value 140.440036
## final  value 140.437613
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.747749
## iter  20 value 147.104245
## iter  30 value 146.549033
## final  value 146.548983
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.047743
## iter  20 value 143.992708
## iter  30 value 140.617582
## iter  40 value 140.458598
## final  value 140.456257
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 166.412452
## iter  20 value 157.065714
## iter  30 value 154.678889
## iter  40 value 154.666191
## iter  40 value 154.666190
## iter  40 value 154.666190
## final  value 154.666190
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 166.773596
## iter  20 value 160.135416
## iter  30 value 159.681496
## final  value 159.681448
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 166.412831
## iter  20 value 157.071227
## iter  30 value 154.688427
## iter  40 value 154.675872
## iter  40 value 154.675871
## iter  40 value 154.675871
## final  value 154.675871
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
```

```
## iter  10 value 160.548466
## iter  20 value 151.950491
## iter  30 value 147.895734
## iter  40 value 147.596033
## final  value 147.596020
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.971917
## iter  20 value 155.449929
## iter  30 value 154.956915
## final  value 154.956904
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.548910
## iter  20 value 151.955689
## iter  30 value 147.925832
## iter  40 value 147.632261
## final  value 147.632248
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 140.046920
## iter  20 value 119.675084
## iter  30 value 112.713924
## iter  40 value 111.956495
## final  value 111.953052
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 141.535951
## iter  20 value 129.950728
## iter  30 value 128.705969
## final  value 128.705892
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 140.048537
## iter  20 value 119.698311
## iter  30 value 112.756653
## iter  40 value 112.018379
## final  value 112.015150
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.653486
## iter  20 value 151.193446
## iter  30 value 148.722084
## iter  40 value 148.584141
## iter  40 value 148.584140
## iter  40 value 148.584140
## final  value 148.584140
## converged
```

```
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.219808
## iter  20 value 154.638331
## iter  30 value 154.232527
## final  value 154.232456
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.654037
## iter  20 value 151.198545
## iter  30 value 148.734409
## iter  40 value 148.598344
## iter  40 value 148.598343
## iter  40 value 148.598343
## final  value 148.598343
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 156.972585
## iter  20 value 151.349075
## iter  30 value 149.324965
## iter  40 value 149.116603
## final  value 149.116551
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 157.897042
## iter  20 value 154.148567
## iter  30 value 153.788710
## final  value 153.788614
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 156.973596
## iter  20 value 151.352743
## iter  30 value 149.334729
## iter  40 value 149.129662
## final  value 149.129612
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 159.094311
## iter  20 value 151.731931
## iter  30 value 149.515604
## iter  40 value 149.499672
## final  value 149.499592
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 159.622499
## iter  20 value 154.223091
## iter  30 value 153.750594
## final  value 153.750569
```

```
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 159.094893
## iter  20 value 151.735630
## iter  30 value 149.523803
## iter  40 value 149.508062
## final  value 149.507984
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 162.481018
## iter  20 value 156.952439
## iter  30 value 154.855253
## iter  40 value 154.679543
## iter  40 value 154.679543
## iter  40 value 154.679543
## final  value 154.679543
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 162.773507
## iter  20 value 159.066642
## iter  30 value 158.699089
## final  value 158.699035
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 162.481321
## iter  20 value 156.955656
## iter  30 value 154.864868
## iter  40 value 154.691876
## iter  40 value 154.691875
## iter  40 value 154.691875
## final  value 154.691875
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 155.993662
## iter  20 value 144.074977
## iter  30 value 140.720773
## iter  40 value 140.657236
## final  value 140.657039
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 157.050214
## iter  20 value 149.574975
## iter  30 value 148.974852
## final  value 148.974802
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 155.994895
```

```
## iter  20 value 144.083769
## iter  30 value 140.741277
## iter  40 value 140.678650
## final  value 140.678459
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 148.198908
## iter  20 value 136.905342
## iter  30 value 134.305767
## iter  40 value 134.198390
## final  value 134.197511
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 149.891819
## iter  20 value 143.225974
## iter  30 value 142.758334
## final  value 142.758240
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 148.200779
## iter  20 value 136.914904
## iter  30 value 134.322951
## iter  40 value 134.217285
## final  value 134.216441
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 150.204605
## iter  20 value 141.099078
## iter  30 value 140.210957
## iter  40 value 140.200699
## final  value 140.200661
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 150.724185
## iter  20 value 143.068495
## iter  30 value 142.566641
## final  value 142.566632
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 150.205148
## iter  20 value 141.101447
## iter  30 value 140.214577
## iter  40 value 140.204462
## final  value 140.204425
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 154.867498
```

```
## iter  20 value 146.688881
## iter  30 value 143.862733
## iter  40 value 143.802938
## final  value 143.802554
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 155.220541
## iter  20 value 149.820625
## iter  30 value 149.390951
## final  value 149.390937
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 154.867862
## iter  20 value 146.693646
## iter  30 value 143.875947
## iter  40 value 143.817035
## final  value 143.816664
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 147.248738
## iter  20 value 134.854889
## iter  30 value 132.363687
## iter  40 value 132.239346
## final  value 132.238349
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 148.326143
## iter  20 value 139.562757
## iter  30 value 138.885977
## final  value 138.885909
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 147.249898
## iter  20 value 134.862168
## iter  30 value 132.378162
## iter  40 value 132.255664
## final  value 132.254700
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.110900
## iter  20 value 139.889794
## iter  30 value 137.664216
## iter  40 value 137.589368
## final  value 137.589240
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.615796
```

```
## iter  20 value 143.135086
## iter  30 value 142.587423
## final  value 142.587412
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 151.111424
## iter  20 value 139.894132
## iter  30 value 137.674190
## iter  40 value 137.600440
## final  value 137.600316
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.759844
## iter  20 value 152.695934
## iter  30 value 148.618290
## final  value 148.615896
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 161.694927
## iter  20 value 156.412000
## iter  30 value 155.671326
## final  value 155.671316
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.760857
## iter  20 value 152.701197
## iter  30 value 148.634222
## final  value 148.631857
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.322503
## iter  20 value 144.699584
## iter  30 value 141.362983
## iter  40 value 141.280390
## final  value 141.280374
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.470295
## iter  20 value 150.880912
## iter  30 value 150.130691
## final  value 150.130659
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 160.323861
## iter  20 value 144.710185
## iter  30 value 141.381711
## iter  40 value 141.300329
```

```
## final  value 141.300313
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 157.314761
## iter  20 value 149.636601
## iter  30 value 148.421882
## final  value 148.333528
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 157.966082
## iter  20 value 152.877974
## iter  30 value 152.636529
## final  value 152.636501
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 157.315461
## iter  20 value 149.641032
## iter  30 value 148.430117
## final  value 148.343071
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 158.003734
## iter  20 value 146.311826
## iter  30 value 142.052713
## final  value 142.032545
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 158.709394
## iter  20 value 150.299007
## iter  30 value 149.801280
## final  value 149.801269
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 158.004467
## iter  20 value 146.319590
## iter  30 value 142.072553
## final  value 142.052603
## converged
## # weights:  32 (31 variable)
## initial  value 182.990856
## iter  10 value 167.762966
## iter  20 value 161.741218
## iter  30 value 160.666525
## final  value 160.665334
## converged
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 89 43
##          1 45 87
##
##                Accuracy : 0.6667
##                  95% CI : (0.6063, 0.7233)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.24e-07
##
##                   Kappa : 0.3333
##  Mcnemar's Test P-Value : 0.9151
##
##             Sensitivity : 0.6642
##             Specificity : 0.6692
##          Pos Pred Value : 0.6742
##          Neg Pred Value : 0.6591
##              Prevalence : 0.5076
##          Detection Rate : 0.3371
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.6667
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 29 23
##          1 13 23
##
##                Accuracy : 0.5909
##                  95% CI : (0.4809, 0.6946)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.1200
##
##                   Kappa : 0.1885
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.6905
##             Specificity : 0.5000
##          Pos Pred Value : 0.5577
##          Neg Pred Value : 0.6389
##              Prevalence : 0.4773
##          Detection Rate : 0.3295
##    Detection Prevalence : 0.5909
##       Balanced Accuracy : 0.5952
##
##        'Positive' Class : 0
##
## [1] "nb"
```

```
## Loading required package: klaR
```

```
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  45  27
##          1  89 103
##
##                Accuracy : 0.5606
##                  95% CI : (0.4984, 0.6214)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 0.04812
##
##                   Kappa : 0.1272
##  Mcnemar's Test P-Value : 1.481e-08
##
##             Sensitivity : 0.3358
##             Specificity : 0.7923
##          Pos Pred Value : 0.6250
##          Neg Pred Value : 0.5365
##              Prevalence : 0.5076
##          Detection Rate : 0.1705
##    Detection Prevalence : 0.2727
##       Balanced Accuracy : 0.5641
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 15 19
##          1 27 27
##
##                Accuracy : 0.4773
##                  95% CI : (0.3696, 0.5865)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.8316
##
##                   Kappa : -0.0564
##  Mcnemar's Test P-Value : 0.3020
##
##             Sensitivity : 0.3571
##             Specificity : 0.5870
##          Pos Pred Value : 0.4412
##          Neg Pred Value : 0.5000
##              Prevalence : 0.4773
##          Detection Rate : 0.1705
##    Detection Prevalence : 0.3864
##       Balanced Accuracy : 0.4720
##
##        'Positive' Class : 0
##
```

```
## [1] "qda"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  94  10
##          1  40 120
##
##                Accuracy : 0.8106
##                  95% CI : (0.7581, 0.856)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6224
##  Mcnemar's Test P-Value : 4.11e-05
##
##             Sensitivity : 0.7015
##             Specificity : 0.9231
##          Pos Pred Value : 0.9038
##          Neg Pred Value : 0.7500
##              Prevalence : 0.5076
##          Detection Rate : 0.3561
##    Detection Prevalence : 0.3939
##       Balanced Accuracy : 0.8123
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 16 21
##          1 26 25
##
##                Accuracy : 0.4659
##                  95% CI : (0.3588, 0.5754)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.8797
##
##                   Kappa : -0.076
##  Mcnemar's Test P-Value : 0.5596
##
##             Sensitivity : 0.3810
##             Specificity : 0.5435
##          Pos Pred Value : 0.4324
##          Neg Pred Value : 0.4902
##              Prevalence : 0.4773
##          Detection Rate : 0.1818
##    Detection Prevalence : 0.4205
##       Balanced Accuracy : 0.4622
##
##        'Positive' Class : 0
```

```
##
## [1] "rf"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 134   0
##          1   0 130
##
##                Accuracy : 1
##                  95% CI : (0.9861, 1)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5076
##          Detection Rate : 0.5076
##    Detection Prevalence : 0.5076
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23 21
##          1 19 25
##
##                Accuracy : 0.5455
##                  95% CI : (0.4358, 0.652)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.3751
##
##                   Kappa : 0.0909
##  Mcnemar's Test P-Value : 0.8744
##
##             Sensitivity : 0.5476
##             Specificity : 0.5435
##          Pos Pred Value : 0.5227
##          Neg Pred Value : 0.5682
##              Prevalence : 0.4773
##          Detection Rate : 0.2614
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.5455
##
```

```
##        'Positive' Class : 0
##
## [1] "rocc"
## Loading required package: rocc
## Loading required package: ROCR
## Loading required package: gplots
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess
##
## Attaching package: 'ROCR'
## The following object is masked from 'package:neuralnet':
##
##     prediction
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 78 48
##          1 56 82
##
##                Accuracy : 0.6061
##                  95% CI : (0.5443, 0.6654)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 0.0008138
##
##                   Kappa : 0.2127
##  Mcnemar's Test P-Value : 0.4924568
##
##             Sensitivity : 0.5821
##             Specificity : 0.6308
##          Pos Pred Value : 0.6190
##          Neg Pred Value : 0.5942
##              Prevalence : 0.5076
##          Detection Rate : 0.2955
##    Detection Prevalence : 0.4773
##       Balanced Accuracy : 0.6064
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23 23
##          1 19 23
```

```
##
##                Accuracy : 0.5227
##                  95% CI : (0.4135, 0.6304)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.5431
##
##                   Kappa : 0.0474
##  Mcnemar's Test P-Value : 0.6434
##
##             Sensitivity : 0.5476
##             Specificity : 0.5000
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.5476
##              Prevalence : 0.4773
##          Detection Rate : 0.2614
##    Detection Prevalence : 0.5227
##       Balanced Accuracy : 0.5238
##
##        'Positive' Class : 0
##
## [1] "svmLinear"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 85 39
##          1 49 91
##
##                Accuracy : 0.6667
##                  95% CI : (0.6063, 0.7233)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.24e-07
##
##                   Kappa : 0.3339
##  Mcnemar's Test P-Value : 0.3374
##
##             Sensitivity : 0.6343
##             Specificity : 0.7000
##          Pos Pred Value : 0.6855
##          Neg Pred Value : 0.6500
##              Prevalence : 0.5076
##          Detection Rate : 0.3220
##    Detection Prevalence : 0.4697
##       Balanced Accuracy : 0.6672
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 26 23
```

```
##          1 16 23
##
##                Accuracy : 0.5568
##                  95% CI : (0.447, 0.6627)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.2974
##
##                   Kappa : 0.1182
##  Mcnemar's Test P-Value : 0.3367
##
##             Sensitivity : 0.6190
##             Specificity : 0.5000
##          Pos Pred Value : 0.5306
##          Neg Pred Value : 0.5897
##              Prevalence : 0.4773
##          Detection Rate : 0.2955
##    Detection Prevalence : 0.5568
##       Balanced Accuracy : 0.5595
##
##        'Positive' Class : 0
##
## [1] "svmRadial"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 102  26
##          1  32 104
##
##                Accuracy : 0.7803
##                  95% CI : (0.7255, 0.8287)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5608
##  Mcnemar's Test P-Value : 0.5115
##
##             Sensitivity : 0.7612
##             Specificity : 0.8000
##          Pos Pred Value : 0.7969
##          Neg Pred Value : 0.7647
##              Prevalence : 0.5076
##          Detection Rate : 0.3864
##    Detection Prevalence : 0.4848
##       Balanced Accuracy : 0.7806
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
```

```
##          0 22 26
##          1 20 20
##
##                Accuracy : 0.4773
##                  95% CI : (0.3696, 0.5865)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.8316
##
##                   Kappa : -0.0412
##  Mcnemar's Test P-Value : 0.4610
##
##             Sensitivity : 0.5238
##             Specificity : 0.4348
##          Pos Pred Value : 0.4583
##          Neg Pred Value : 0.5000
##              Prevalence : 0.4773
##          Detection Rate : 0.2500
##    Detection Prevalence : 0.5455
##       Balanced Accuracy : 0.4793
##
##        'Positive' Class : 0
##
## [1] "svmRadialWeights"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 102  22
##          1  32 108
##
##                Accuracy : 0.7955
##                  95% CI : (0.7417, 0.8424)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5913
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.7612
##             Specificity : 0.8308
##          Pos Pred Value : 0.8226
##          Neg Pred Value : 0.7714
##              Prevalence : 0.5076
##          Detection Rate : 0.3864
##    Detection Prevalence : 0.4697
##       Balanced Accuracy : 0.7960
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  0  1
##          0 22 26
##          1 20 20
##
##                Accuracy : 0.4773
##                  95% CI : (0.3696, 0.5865)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.8316
##
##                   Kappa : -0.0412
##  Mcnemar's Test P-Value : 0.4610
##
##             Sensitivity : 0.5238
##             Specificity : 0.4348
##          Pos Pred Value : 0.4583
##          Neg Pred Value : 0.5000
##              Prevalence : 0.4773
##          Detection Rate : 0.2500
##    Detection Prevalence : 0.5455
##       Balanced Accuracy : 0.4793
##
##        'Positive' Class : 0
##
## [1] "treebag"
```

```
## Loading required package: ipred
```

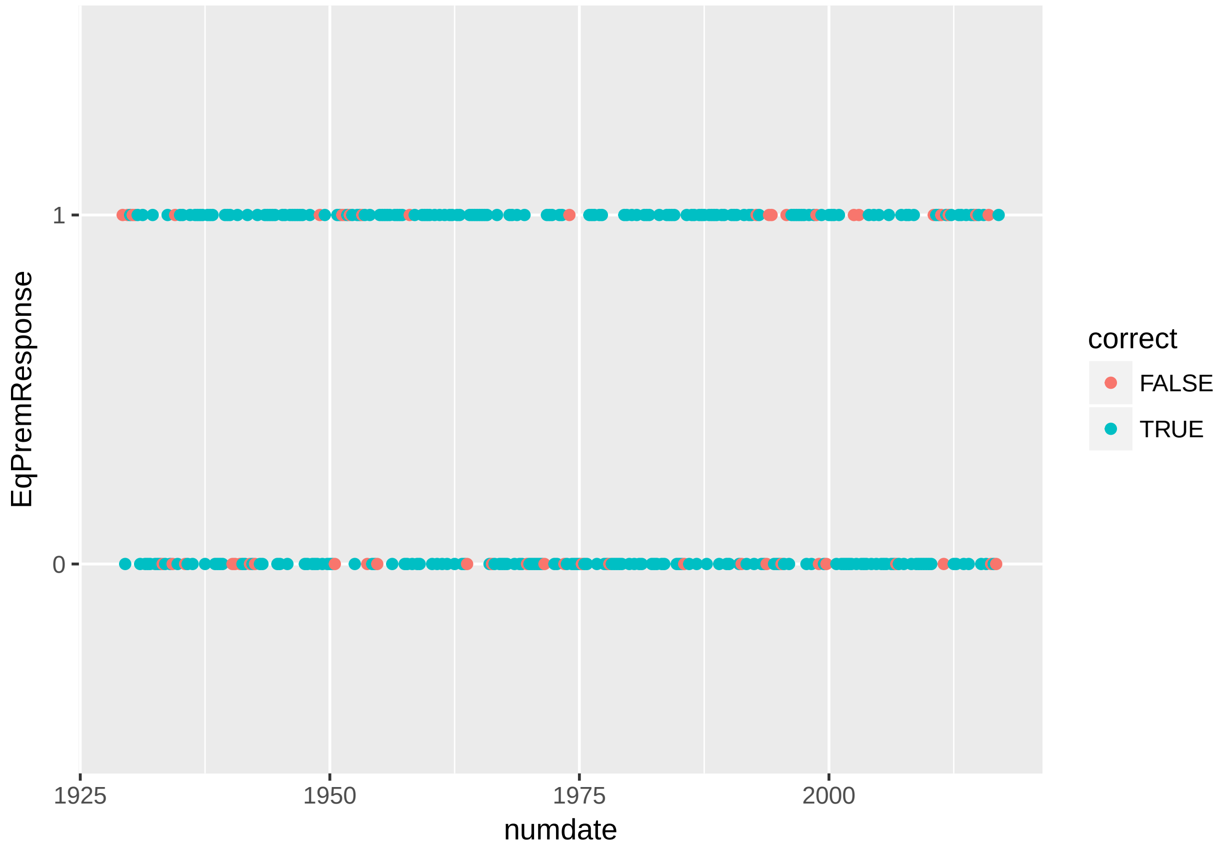
```
## Loading required package: e1071
```

```
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 134   0
##          1   0 130
##
##                Accuracy : 1
##                  95% CI : (0.9861, 1)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5076
##          Detection Rate : 0.5076
##    Detection Prevalence : 0.5076
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
```

```
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 27 17
##          1 15 29
##
##                Accuracy : 0.6364
##                  95% CI : (0.5269, 0.7363)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.02074
##
##                   Kappa : 0.2727
##  Mcnemar's Test P-Value : 0.85968
##
##             Sensitivity : 0.6429
##             Specificity : 0.6304
##          Pos Pred Value : 0.6136
##          Neg Pred Value : 0.6591
##              Prevalence : 0.4773
##          Detection Rate : 0.3068
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.6366
##
##        'Positive' Class : 0
##
```

Chart correct vs. incorrect

```
myPredict <- data.frame(prediction=predict(mymodel, bigdata2))
myPredict$EqPremResponse<-bigdata2$EqPremResponse
myPredict$numdate <- tail(data$numdate, nrow(myPredict))
myPredict$correct <- (myPredict$prediction==myPredict$EqPremResponse)
ggplot(myPredict, aes(x=numdate, y=EqPremResponse, color=correct)) + geom_point()
```



Just for grins, predict on regressionTestPredicts

- kitchen sink ensemble methods FTW

```
regressionTrainPredicts$EqPremResponse <- trainingset$EqPremResponse
runModel <- function(mxpar) {
    return (train(EqPremResponse ~ ., data=regressionTrainPredicts, method=mxpar,
                  preProc = c("center", "scale"), verbose=FALSE))
}

#myMethods <- c("ada", "AdaBag", "adaboost", "bartMachine", "deepboost", "gbm", "lda", "LogitBoost", "m

myMethods <- c("bartMachine", "deepboost", "gbm", "lda", "rf", 'rocc', "svmLinear","svmRadial", "svmRadi

for(mx in myMethods) {
  print(Sys.time())
  print(mx)
  mymodel = runModel(mx)

  print("Training set confusion matrix")
  myPredict <- data.frame(prediction=predict(mymodel, regressionTrainPredicts))
  myPredict$EqPremResponse<-trainingset$EqPremResponse
  print(confusionMatrix(myPredict$prediction, myPredict$EqPremResponse))

  print("Test set confusion matrix")
  myPredict <- data.frame(prediction=predict(mymodel, regressionTestPredicts))
  myPredict$EqPremResponse<-testset$EqPremResponse
  print(confusionMatrix(myPredict$prediction, myPredict$EqPremResponse))
```

```
}
```

```
## [1] "2017-07-06 19:36:15 EDT"
## [1] "bartMachine"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  96  24
##          1  38 106
##
##                Accuracy : 0.7652
##                  95% CI : (0.7093, 0.8149)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5309
##  Mcnemar's Test P-Value : 0.09874
##
##             Sensitivity : 0.7164
##             Specificity : 0.8154
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.7361
##              Prevalence : 0.5076
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.4545
##       Balanced Accuracy : 0.7659
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 25 19
##          1 17 27
##
##                Accuracy : 0.5909
##                  95% CI : (0.4809, 0.6946)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.1200
##
##                   Kappa : 0.1818
##  Mcnemar's Test P-Value : 0.8676
##
##             Sensitivity : 0.5952
##             Specificity : 0.5870
##          Pos Pred Value : 0.5682
##          Neg Pred Value : 0.6136
##              Prevalence : 0.4773
##          Detection Rate : 0.2841
##    Detection Prevalence : 0.5000
```

```
##       Balanced Accuracy : 0.5911
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:53:31 EDT"
## [1] "deepboost"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 86 31
##          1 48 99
##
##                Accuracy : 0.7008
##                  95% CI : (0.6416, 0.7554)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.416e-10
##
##                   Kappa : 0.4025
##  Mcnemar's Test P-Value : 0.07184
##
##             Sensitivity : 0.6418
##             Specificity : 0.7615
##          Pos Pred Value : 0.7350
##          Neg Pred Value : 0.6735
##              Prevalence : 0.5076
##          Detection Rate : 0.3258
##    Detection Prevalence : 0.4432
##       Balanced Accuracy : 0.7017
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23 20
##          1 19 26
##
##                Accuracy : 0.5568
##                  95% CI : (0.447, 0.6627)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.2974
##
##                   Kappa : 0.1127
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.5476
##             Specificity : 0.5652
##          Pos Pred Value : 0.5349
##          Neg Pred Value : 0.5778
##              Prevalence : 0.4773
```

```
##          Detection Rate : 0.2614
##    Detection Prevalence : 0.4886
##       Balanced Accuracy : 0.5564
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:58:52 EDT"
## [1] "gbm"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  87  29
##          1  47 101
##
##                Accuracy : 0.7121
##                  95% CI : (0.6534, 0.766)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.082e-11
##
##                   Kappa : 0.4253
##  Mcnemar's Test P-Value : 0.05117
##
##             Sensitivity : 0.6493
##             Specificity : 0.7769
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.6824
##              Prevalence : 0.5076
##          Detection Rate : 0.3295
##    Detection Prevalence : 0.4394
##       Balanced Accuracy : 0.7131
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 21 19
##          1 21 27
##
##                Accuracy : 0.5455
##                  95% CI : (0.4358, 0.652)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.3751
##
##                   Kappa : 0.0871
##  Mcnemar's Test P-Value : 0.8744
##
##             Sensitivity : 0.5000
##             Specificity : 0.5870
##          Pos Pred Value : 0.5250
```

```
##          Neg Pred Value : 0.5625
##              Prevalence : 0.4773
##          Detection Rate : 0.2386
##    Detection Prevalence : 0.4545
##       Balanced Accuracy : 0.5435
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:58:56 EDT"
## [1] "lda"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 88 43
##          1 46 87
##
##                Accuracy : 0.6629
##                  95% CI : (0.6024, 0.7197)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 2.42e-07
##
##                   Kappa : 0.3258
##  Mcnemar's Test P-Value : 0.8321
##
##             Sensitivity : 0.6567
##             Specificity : 0.6692
##          Pos Pred Value : 0.6718
##          Neg Pred Value : 0.6541
##              Prevalence : 0.5076
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.4962
##       Balanced Accuracy : 0.6630
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 21 17
##          1 21 29
##
##                Accuracy : 0.5682
##                  95% CI : (0.4582, 0.6734)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.2279
##
##                   Kappa : 0.131
##  Mcnemar's Test P-Value : 0.6265
##
##             Sensitivity : 0.5000
```

```
##             Specificity : 0.6304
##          Pos Pred Value : 0.5526
##          Neg Pred Value : 0.5800
##              Prevalence : 0.4773
##          Detection Rate : 0.2386
##    Detection Prevalence : 0.4318
##       Balanced Accuracy : 0.5652
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:58:57 EDT"
## [1] "rf"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 134   0
##          1   0 130
##
##                Accuracy : 1
##                  95% CI : (0.9861, 1)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5076
##          Detection Rate : 0.5076
##    Detection Prevalence : 0.5076
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 25 20
##          1 17 26
##
##                Accuracy : 0.5795
##                  95% CI : (0.4695, 0.684)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.1685
##
##                   Kappa : 0.16
##  Mcnemar's Test P-Value : 0.7423
```

```
##
##             Sensitivity : 0.5952
##             Specificity : 0.5652
##          Pos Pred Value : 0.5556
##          Neg Pred Value : 0.6047
##              Prevalence : 0.4773
##          Detection Rate : 0.2841
##    Detection Prevalence : 0.5114
##       Balanced Accuracy : 0.5802
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:59:11 EDT"
## [1] "rocc"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 89 37
##          1 45 93
##
##                Accuracy : 0.6894
##                  95% CI : (0.6298, 0.7447)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.581e-09
##
##                   Kappa : 0.3792
##  Mcnemar's Test P-Value : 0.4395
##
##             Sensitivity : 0.6642
##             Specificity : 0.7154
##          Pos Pred Value : 0.7063
##          Neg Pred Value : 0.6739
##              Prevalence : 0.5076
##          Detection Rate : 0.3371
##    Detection Prevalence : 0.4773
##       Balanced Accuracy : 0.6898
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 26 18
##          1 16 28
##
##                Accuracy : 0.6136
##                  95% CI : (0.5038, 0.7156)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.05418
##
```

```
##                  Kappa : 0.2273
##  Mcnemar's Test P-Value : 0.86383
##
##            Sensitivity : 0.6190
##            Specificity : 0.6087
##         Pos Pred Value : 0.5909
##         Neg Pred Value : 0.6364
##             Prevalence : 0.4773
##         Detection Rate : 0.2955
##   Detection Prevalence : 0.5000
##      Balanced Accuracy : 0.6139
##
##       'Positive' Class : 0
##
## [1] "2017-07-06 19:59:14 EDT"
## [1] "svmLinear"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 83 32
##          1 51 98
##
##               Accuracy : 0.6856
##                 95% CI : (0.6259, 0.7411)
##    No Information Rate : 0.5076
##    P-Value [Acc > NIR] : 3.412e-09
##
##                  Kappa : 0.3724
##  Mcnemar's Test P-Value : 0.04818
##
##            Sensitivity : 0.6194
##            Specificity : 0.7538
##         Pos Pred Value : 0.7217
##         Neg Pred Value : 0.6577
##             Prevalence : 0.5076
##         Detection Rate : 0.3144
##   Detection Prevalence : 0.4356
##      Balanced Accuracy : 0.6866
##
##       'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 17 17
##          1 25 29
##
##               Accuracy : 0.5227
##                 95% CI : (0.4135, 0.6304)
##    No Information Rate : 0.5227
```

```
##     P-Value [Acc > NIR] : 0.5431
##
##                   Kappa : 0.0355
##  Mcnemar's Test P-Value : 0.2801
##
##             Sensitivity : 0.4048
##             Specificity : 0.6304
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.5370
##              Prevalence : 0.4773
##          Detection Rate : 0.1932
##    Detection Prevalence : 0.3864
##       Balanced Accuracy : 0.5176
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:59:15 EDT"
## [1] "svmRadial"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104  58
##          1  30  72
##
##                Accuracy : 0.6667
##                  95% CI : (0.6063, 0.7233)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.24e-07
##
##                   Kappa : 0.331
##  Mcnemar's Test P-Value : 0.003999
##
##             Sensitivity : 0.7761
##             Specificity : 0.5538
##          Pos Pred Value : 0.6420
##          Neg Pred Value : 0.7059
##              Prevalence : 0.5076
##          Detection Rate : 0.3939
##    Detection Prevalence : 0.6136
##       Balanced Accuracy : 0.6650
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 32 28
##          1 10 18
##
##                Accuracy : 0.5682
```

```
##                  95% CI : (0.4582, 0.6734)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.22786
##
##                   Kappa : 0.1504
##  Mcnemar's Test P-Value : 0.00582
##
##             Sensitivity : 0.7619
##             Specificity : 0.3913
##          Pos Pred Value : 0.5333
##          Neg Pred Value : 0.6429
##              Prevalence : 0.4773
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.6818
##       Balanced Accuracy : 0.5766
##
##        'Positive' Class : 0
##
## [1] "2017-07-06 19:59:18 EDT"
## [1] "svmRadialWeights"
## [1] "Training set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 94 41
##          1 40 89
##
##                Accuracy : 0.6932
##                  95% CI : (0.6337, 0.7483)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 7.197e-10
##
##                   Kappa : 0.3862
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7015
##             Specificity : 0.6846
##          Pos Pred Value : 0.6963
##          Neg Pred Value : 0.6899
##              Prevalence : 0.5076
##          Detection Rate : 0.3561
##    Detection Prevalence : 0.5114
##       Balanced Accuracy : 0.6931
##
##        'Positive' Class : 0
##
## [1] "Test set confusion matrix"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 27 23
##          1 15 23
```

```
##
##                Accuracy : 0.5682
##                  95% CI : (0.4582, 0.6734)
##     No Information Rate : 0.5227
##     P-Value [Acc > NIR] : 0.2279
##
##                   Kappa : 0.1417
##  Mcnemar's Test P-Value : 0.2561
##
##             Sensitivity : 0.6429
##             Specificity : 0.5000
##          Pos Pred Value : 0.5400
##          Neg Pred Value : 0.6053
##              Prevalence : 0.4773
##          Detection Rate : 0.3068
##    Detection Prevalence : 0.5682
##       Balanced Accuracy : 0.5714
##
##        'Positive' Class : 0
##
```