Final Project Draft - Part 2 Prediction

Jacob Walsh

December 6, 2016

library(knitr)  
opts\_chunk$set(tidy.opts=list(width.cutoff=65),tidy=TRUE, cache=TRUE)

# Part 2 Prediction

charity <- read.csv("C:/Penn State MAS/Stat 897D/charity.csv")  
# transform data  
t.charity = charity  
t.charity$avhv = log(charity$avhv)  
t.charity$incm = log(charity$incm)  
t.charity$inca = log(charity$inca)  
t.charity$plow = charity$plow^(1/3)  
t.charity$tgif = log(charity$tgif)  
t.charity$lgif = log(charity$lgif)  
t.charity$rgif = log(charity$rgif)  
t.charity$tdon = log(charity$tdon)  
t.charity$tlag = log(charity$tlag)  
t.charity$agif = log(charity$agif)  
  
  
# partition data  
data.train <- t.charity[t.charity$part == "train", ]  
x.train <- data.train[, 2:21]  
c.train <- data.train[, 22] # donr  
n.train.c <- length(c.train) # 3984  
y.train <- data.train[c.train == 1, 23] # damt for observations with donr=1  
n.train.y <- length(y.train)  
  
  
data.valid <- t.charity[t.charity$part == "valid", ]  
x.valid <- data.valid[, 2:21]  
c.valid <- data.valid[, 22] # donr  
n.valid.c <- length(c.valid) # 2018  
y.valid <- data.valid[c.valid == 1, 23] # damt for observations with donr=1  
n.valid.y <- length(y.valid)  
  
  
data.test <- t.charity[t.charity$part == "test", ]  
n.test <- dim(data.test)[1] # 2007  
x.test <- data.test[, 2:21]  
  
x.train.mean <- apply(x.train, 2, mean)  
x.train.sd <- apply(x.train, 2, sd)  
x.train.std <- t((t(x.train) - x.train.mean)/x.train.sd) # standardize to have zero mean and unit sd  
apply(x.train.std, 2, mean) # check zero mean  
apply(x.train.std, 2, sd) # check unit sd  
data.train.std.c <- data.frame(x.train.std, donr = c.train) # to classify donr  
data.train.std.y <- data.frame(x.train.std[c.train == 1, ], damt = y.train) # to predict damt when donr=1  
  
x.valid.std <- t((t(x.valid) - x.train.mean)/x.train.sd) # standardize using training mean and sd  
data.valid.std.c <- data.frame(x.valid.std, donr = c.valid) # to classify donr  
data.valid.std.y <- data.frame(x.valid.std[c.valid == 1, ], damt = y.valid) # to predict damt when donr=1  
  
x.test.std <- t((t(x.test) - x.train.mean)/x.train.sd) # standardize using training mean and sd  
data.test.std <- data.frame(x.test.std)

## Model Comparison

### Linear Regression

library(leaps)  
model.ls1 <- lm(damt ~ reg1 + reg2 + reg3 + reg4 + home + chld + hinc +   
 genf + wrat + avhv + incm + inca + plow + npro + tgif + lgif +   
 rgif + tdon + tlag + agif, data.train.std.y)  
  
pred.valid.ls1 <- predict(model.ls1, newdata = data.valid.std.y) # validation predictions  
lm.mse = mean((y.valid - pred.valid.ls1)^2) # mean prediction error  
sd((y.valid - pred.valid.ls1)^2)/sqrt(n.valid.y) # std error  
  
# best subsets  
model.bestsub = regsubsets(damt ~ ., data = data.train.std.y, nvmax = 20)  
test.mse = model.matrix(damt ~ ., data = data.valid.std.y)  
val.errors = rep(NA, 20)  
for (i in 1:20) {  
 coefi = coef(model.bestsub, id = i)  
 pred = test.mse[, names(coefi)] %\*% coefi  
 val.errors[i] = mean((data.valid.std.y$damt - pred)^2)  
}  
val.errors[which.min(val.errors)]  
which.min(val.errors)  
coefficients(model.bestsub, which.min(val.errors)) #Coefficients for the 'best' model. There is no reduction in the amount of predictors to minimize the mean prediction error, all 20 predictors are included.  
  
plot(model.bestsub, scale = "bic")  
  
model.bestfwd = regsubsets(damt ~ ., data = data.train.std.y, method = "forward")  
plot(model.bestfwd, scale = "bic")  
  
model.bestbwd = regsubsets(damt ~ ., data = data.train.std.y, nvmax = 20,   
 method = "backward")  
plot(model.bestbwd, scale = "bic")  
  
# bic for best subsets and backward are the same 10 predictors  
# reg3+reg4+home+chld+hinc+incm+tgif+lgif+rgif+agif  
lm8.mse = val.errors[8]  
# bic for the best foward regressions is minimized at 8 predictors  
lm10.mse = val.errors[10]  
# mSE isn't any lower

### Ridge and Lasso

library(glmnet)  
grid = 10^seq(10, -2, length = 100)  
xtrain = model.matrix(damt ~ ., data = data.train.std.y)  
ytrain = data.train.std.y$damt  
xtest = model.matrix(damt ~ ., data.valid.std.y)  
ytest = data.valid.std.y$damt  
  
model.ridge = glmnet(xtrain, ytrain, alpha = 0, lambda = grid)  
cv.ridge = cv.glmnet(xtrain, ytrain, alpha = 0)  
pred.ridge = predict(model.ridge, s = cv.ridge$lambda.min, newx = xtest)  
ridge.mse = mean((pred.ridge - ytest)^2)  
  
model.lasso = glmnet(xtrain, ytrain, alpha = 1, lambda = grid)  
cv.lasso = cv.glmnet(xtrain, ytrain, alpha = 1)  
pred.lasso = predict(model.lasso, s = cv.lasso$lambda.min, newx = xtest)  
lasso.mse = mean((pred.lasso - ytest)^2)  
  
# ridge and lasso aren't performing any better than plain old  
# least squares.

### PCR and PLS

# pcr  
library(pls)  
set.seed(1)  
pcr.fit = pcr(damt ~ ., data = data.train.std.y, scale = TRUE, validation = "CV")  
validationplot(pcr.fit, val.type = "MSEP")  
  
# not much gained after 5, next slight dip is at 13.  
pcr.pred = predict(pcr.fit, data.valid.std.y, ncomp = 5)  
pcr5.mse = mean((pcr.pred - y.valid)^2)  
pcr.pred = predict(pcr.fit, data.valid.std.y, ncomp = 13)  
pcr13.mse = mean((pcr.pred - y.valid)^2)  
  
pls.fit = plsr(damt ~ ., data = data.train.std.y, scale = TRUE, validation = "CV")  
validationplot(pls.fit, val.type = "MSEP")  
  
pls.pred = predict(pls.fit, data.valid.std.y, ncomp = 2)  
pls.mse = mean((pls.pred - y.valid)^2) #slight decrease in mse, with only 2 components..

### SVM

library(e1071)  
svmfit1 = svm(damt ~ ., data = data.train.std.y, cost = 1, gamma = 0.01,   
 kernel = "radial")  
svmfit2 = svm(damt ~ ., data = data.train.std.y, cost = 1, gamma = 0.01,   
 epsilon = 0.2, kernel = "radial")  
  
svmpred1 = predict(svmfit1, newdata = data.valid.std.y)  
svmpred2 = predict(svmfit2, newdata = data.valid.std.y)  
  
svm1.mse = mean((svmpred1 - y.valid)^2)  
svm2.mse = mean((svmpred2 - y.valid)^2) #best so far

### Random Forests

library(randomForest)  
set.seed(1)  
bag.charity = randomForest(damt ~ ., data = data.train.std.y, mtry = 20,   
 importance = TRUE)  
pred.valid.bag = predict(bag.charity, newdata = data.valid.std.y)  
bag.mse = mean((pred.valid.bag - y.valid)^2)  
  
set.seed(1)  
rf.charity = randomForest(damt ~ ., data = data.train.std.y, mtry = 10,   
 ntree = 25, importance = TRUE)  
pred.valid.rf = predict(rf.charity, newdata = data.valid.std.y)  
rf.mse = mean((pred.valid.rf - y.valid)^2)

### GBM

I started out by manually changing the parameters using the example in ISLR. After some research, I found a package that does this for you. It's called caret and it did better than any other model by far.

library(gbm)  
boost.mod = gbm(damt ~ ., data = data.train.std.y, distribution = "gaussian",   
 n.trees = 3000, interaction.depth = 4)  
pred.valid.gbm = predict(boost.mod, newdata = data.valid.std.y, n.trees = 1000)  
gbm1.mse = mean((pred.valid.gbm - y.valid)^2)  
  
boost.mod = gbm(damt ~ ., data = data.train.std.y, distribution = "gaussian",   
 n.trees = 5000, interaction.depth = 6)  
pred.valid.gbm = predict(boost.mod, newdata = data.valid.std.y, n.trees = 5000)  
gbm2.mse = mean((pred.valid.gbm - y.valid)^2)

library(caret)  
set.seed(1)  
gbmfit <- train(damt ~ ., data = data.train.std.y, method = "gbm")

#### Optimal GBM

Using the package caret I tuned the parameters to the following

gbmfit  
set.seed(1)  
boost.mod = gbm(damt ~ ., data = data.train.std.y, distribution = "gaussian",   
 n.trees = 150, interaction.depth = 3, shrinkage = 0.1, n.minobsinnode = 10)  
pred.valid.gbm = predict(boost.mod, newdata = data.valid.std.y, n.trees = 150)  
gbmoptimal.mse = mean((pred.valid.gbm - y.valid)^2)

## Results

lm.mse  
lm8.mse  
lm10.mse  
ridge.mse  
lasso.mse  
pcr5.mse  
pcr13.mse  
pls.mse  
svm1.mse  
svm2.mse  
bag.mse  
rf.mse  
gbmoptimal.mse