```
# First we load the bank marketing dataset
myData = bank.additional.full
summary(myData)

# We need to do some cleanup.

myData$default = NULL
myData$duration = NULL
myData$nr.employed = NULL

# We have a mixture of categorical and numerical
# predictors - good candidate for a tree model
# We'll need a couple of libraries: 'randomForest' and 'gbm'

# We will do a simple CV

train = sample(1:41188, 35000)

# First we build a random forest - attach randomForest package
# This takes a second ...

rf.bank = randomForest(y~., data=myData, subset=train, mtry=4)
rf.pred = predict(rf.bank, myData[-train,],type="class")
table(rf.pred, myData[-train,]$y)
mean(rf.pred == myData[-train,]$y)

# Seems impressive, but go back to the data set - 89% were
# no to begin with!

mean(myData[-train,]$y == "no")

# Next we try boosting. We need the response to be numerical
# for this (adding models together)

myData$target = ifelse(myData$y == "no",0,1)
summary(myData$target)

# Attach the 'gbm' package
?gbm
boost.bank = gbm(target~. -y, data=myData[train,], distribution="bernoulli", n.trees=500, shrinkage=.01)
boost.probs = predict(boost.bank,newdata=myData[-train,],n.trees=500, type="response")
head(boost.probs)
dim(myData[-train,])
boost.pred = rep("no",6188)
boost.pred[boost.probs>.5]="yes"
table(boost.pred,myData[-train,]$y)

# More false negatives, but better accuracy on the 'yes'
# Next we try adaboost

boost.bank = gbm(target~. -y, data=myData[train,], distribution="adaboost", n.trees=500, shrinkage=.01)
boost.probs = predict(boost.bank,newdata=myData[-train,],n.trees=500, type="response")
boost.pred = rep("no",6188)
boost.pred[boost.probs>.5]="yes"
table(boost.pred,myData[-train,]$y)

# Note gbm has some built in CV functions

best.iter=gbm.perf(boost.bank,method = "OOB")
print(best.iter)

# Shows that the error continued to drop as we boosted,
# as the theory predicted!

# Let's try SMOTE. Attach the DMwR package ...
# We need to use the 'y' variable since it is the factor

myData_2 <- SMOTE(y~ . -target, myData[train,], perc.over = 100, perc.under=200)
myData_2$target = ifelse(myData_2$y == "no",0,1)

boost.bank = gbm(target~. -y, data=myData_2, distribution="adaboost", n.trees=500, shrinkage=.01)
boost.probs = predict(boost.bank,newdata=myData[-train,],n.trees=500, type="response")
boost.pred = rep("no",5740)
boost.pred[boost.probs>.5]="yes"
table(boost.pred,myData[-train,]$y)

# Let's try one more model - we will use 04 cars

myData=cars04
```

```
summary(myData)

# Data cleanup - several variables are too rare to be useful

myData$V1 = NULL
myData$V4=NULL
myData$V5=NULL
myData$V6=NULL

myData$V2 = as.factor(myData$V2)
myData$V3 = as.factor(myData$V3)
myData$V7 = as.factor(myData$V7)
myData$V8 = as.factor(myData$V8)

myData$V14 = as.numeric(as.character(myData$V14))
myData$V15 = as.numeric(as.character(myData$V15))
myData$V16 = as.numeric(as.character(myData$V16))
myData$V17 = as.numeric(as.character(myData$V17))
myData$V18 = as.numeric(as.character(myData$V18))
myData$V19 = as.numeric(as.character(myData$V19))

myData = na.omit(myData)
summary(myData)
dim(myData)
boxplot(myData$V9)

# This predictor has outliers - remove them

myData = myData[which(myData$V9 < 65000),]
dim(myData)
summary(myData)

# V9 is retail price (response), and is probably closely
# connected to V10, dealer cost

plot(myData$V9~myData$V10)
myData$V10 = NULL

# We will again do a simple CV - note this is a regression
# model, as we are trying to predict price. (For better
# results, we could have taken the log of the price.)

train = sample(1:362,275)
rf.cars = randomForest(V9~. , data=myData,subset=train,mtry=4)
yhat.rf = predict(rf.cars,newdata = myData[-train,])
mean((yhat.rf - myData[-train,]$V9)^2)
sqrt(mean((yhat.rf - myData[-train,]$V9)^2))/mean(myData[-train,]$V9)

# Boosting - note we use the Gausssian distribution

boost.cars = gbm(V9~. ,data=myData[train,],distribution = "gaussian", n.trees=5000, shrinkage = .001)
yhat.boost = predict(boost.cars,newdata = myData[-train,],n.trees=5000)
sqrt(mean((yhat.boost - myData[-train,]$V9)^2))/mean(myData[-train,]$V9)

# Let's try to increase the shrinkage

boost.cars = gbm(V9~. ,data=myData[train,],distribution = "gaussian", n.trees=5000, shrinkage = .01)
yhat.boost = predict(boost.cars,newdata = myData[-train,],n.trees=5000)
sqrt(mean((yhat.boost - myData[-train,]$V9)^2))/mean(myData[-train,]$V9)

# Now let's try to add some interaction terms

boost.cars = gbm(V9~. ,data=myData[train,],distribution = "gaussian", n.trees=5000, interaction.depth = 4,
shrinkage = .01)
yhat.boost = predict(boost.cars,newdata = myData[-train,],n.trees=5000)
sqrt(mean((yhat.boost - myData[-train,]$V9)^2))/mean(myData[-train,]$V9)

# Let's take a look at the OOB error

best.iter=gbm.perf(boost.cars,method = "OOB")
best.iter

# Don't need all the trees?

yhat.boost = predict(boost.cars,newdata = myData[-train,],best.iter)
sqrt(mean((yhat.boost - myData[-train,]$V9)^2))/mean(myData[-train,]$V9)

# OOB can be deceptive - graph seems to imply that 316 is not
# optimal! Let's try 10-fold CV on the entire dataset
```

```
boost.cars = gbm(V9~. ,data=myData,distribution = "gaussian", n.trees=5000, interaction.depth = 4, shrinkage =
.01, cv.folds = 10)

# We can access the 10-fold CV loss function values in the
# vector boost.cars$cv.error - this gives a cross vlaidated
# estimate of the loss function with each iteration. Let's
# look at the best one ...

sqrt(min(boost.cars$cv.error))/mean(myData[-train,]$V9)


# This code can be used to do GBM directly ...
# Does not seem to work well?

learnRate = .8

reg.mod = lm(V9~.,data=myData, subset=train)
y_hat = predict(reg.mod, myData[-train,])*learnRate



myDataBoost = myData[train,]
myDataBoost$V9 = reg.mod$residuals
for(i in 1:10000)
{
  reg.mod = lm(V9~.,data=myDataBoost)
  y_hat = y_hat+predict(reg.mod, myData[-train,])*learnRate
  myDataBoost$V9 = reg.mod$residuals
}

sqrt(mean((myData[-train,]$V9 - y_hat )^2))/mean(myData[-train,]$V9)

_____________________________________________________________________# bstSparse <- xgboost(data =
as.matrix(myData[train,-5]), label = myData[train,5], max.depth = 2, eta = 1, nthread = 2, nround = 2, objective
= "binary:logistic")
Error in xgb.iter.update(bst$handle, dtrain, iteration - 1, obj) :
  [00:24:01] amalgamation/../src/objective/regression_obj.cc:108: label must be in [0,1] for logistic regression

# First attach the 'xgboost' package. The Mushroom dataset
# is included, already broken into test and train sets ...
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
dim(train$data)
dim(test$data)

# The test and train datasets are stored as lists, with
# a data and label part ...
names(train)

# The data is actually in the form of a dgCMatrix ...
class(train$data)
class(train$label)

# Here we train a model on the train set ...
bstSparse <- xgboost(data = train$data, label = train$label, max.depth = 2, eta = 1, nthread = 2, nround = 2,
objective = "binary:logistic")

# The previous model worked with the sparse matrix form ...
# Here is the more traditional (dense) matrix form ...
bstDense <- xgboost(data = as.matrix(train$data), label = train$label, max.depth = 2, eta = 1, nthread = 2,
nround = 2, objective = "binary:logistic")

# We can also combine the data and label into one dataframe ...

# Now we predict the Test data ...
pred <- predict(bstDense, test$data)
prediction <- as.numeric(pred > 0.5)
print(head(prediction))
err <- mean(as.numeric(pred > 0.5) != test$label)
print(paste("test-error=", err))

table(prediction,test$label)
```