Lab_08R

36-290 – Statistical Research Methodology

Week 8 Thursday – Fall 2021

Preliminaries

Goal

Today we will apply both regression-tree-style and classification-tree-style boosting to the datasets we began working with last week. (However, instead of using the `gbm` package of ISLR, we will use the newer, fancier `xgboost` package.)

Questions

Data, Part I

Below we read in the same data that we used during Week 7. On Tuesday we did not downsample, because tree computations go quickly. On Thursday we did downsample, because random forest calculations go relatively slowly. Today we are back in a situation where downsampling is not necessary, because `xgboost` is fast.

```
rm(list=ls())
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/DM_GALAXY/Massive_Black_II.
Rdata"
load(url(file.path))
rm(file.path)

resp.train = resp.train.df$prop.sfr
w = which(resp.train>0)
pred.train = pred.train[w,]
resp.train = log10(resp.train[w])

resp.test  = resp.test.df$prop.sfr
w = which(resp.test>0)
pred.test = pred.test[w,]
resp.test = log10(resp.test[w])

cat("Sample sizes: train = ",length(resp.train)," test = ",length(resp.test),"\n")
```

```
## Sample sizes: train =  61255  test =  30143
```

Question 1

We will install and use the `xgboost` package. `xgboost` is a newer, pricklier package that doesn't always do things "the R way."

First, you will want to use the `xgb.DMatrix()` function to create train and test matrices; the arguments to `xgb.DMatrix()` should be, e.g., `data=as.matrix(pred.train)` and `label=resp.train`. (Note the `as.matrix()` call: `xgboost` does not work with categorical predictor variables!)

Second, you will want to set a random number generator seed and call the function `xgb.cv`, passing in your training data, a number of folds (`nfold`), a maximum number of trees to try (`nrounds`), and a list of parameters (e.g., `params=list(objective="reg:squarederror")`) … all this means evaluate the test-set MSE instead of some other cost function).

Third, you will want to call `xgboost`, passing in your training data and the optimal number of trees. As it is not clear how to get this information: assume your output variable for `xgb.cv()` is simply called `out`. Then the optimal number of trees is given by `which.min(out$evaluation_log$test_rmse_mean)`. Simple, huh? Also input the same `params` argument as above.

Fourth, you want to call `predict()`, with your output from `xgboost` and with `newdata=test`.

Finally: compute the test-set MSE and use `ggplot()` to plot the typical regression diagnostic plot that we use in this class. You should find that the test-set MSE is roughly the same as that for random forest, indicating that boosting doesn't necessarily buy you better predictions (but: it does make predictions faster, which means you can process more data, so there's consequently less uncertainty on the predictions and the test-set MSE…and you can more easily perform cross-validation).

Note: if you want to turn off the output from `xgb.cv` and `xgboost`, pass the argument `verbose=0`.

```r
library(xgboost)
library(ggplot2)

#First, you will want to use the `xgb.DMatrix()` function to create train and test matrices

train = xgb.DMatrix(data=as.matrix(pred.train),label=resp.train)
test = xgb.DMatrix(data=as.matrix(pred.test),label=resp.test)

#Second, set a random number generator seed and call the function `xgb.cv`
#passing in your training data, a number of folds (`nfold`), a maximum number of trees to try (`nrounds`), and a li
st of parameters
#(e.g., `params=list(objective="reg:squarederror"))`...all this means evaluate the test-set MSE instead of some oth
er cost function).

set.seed(101)
xgb.cv.out = xgb.cv(params=list(objective="reg:squarederror"),train,nrounds=30,nfold=5,verbose=0)
cat("The optimal number of trees is", which.min(xgb.cv.out$evaluation_log$test_rmse_mean))
```

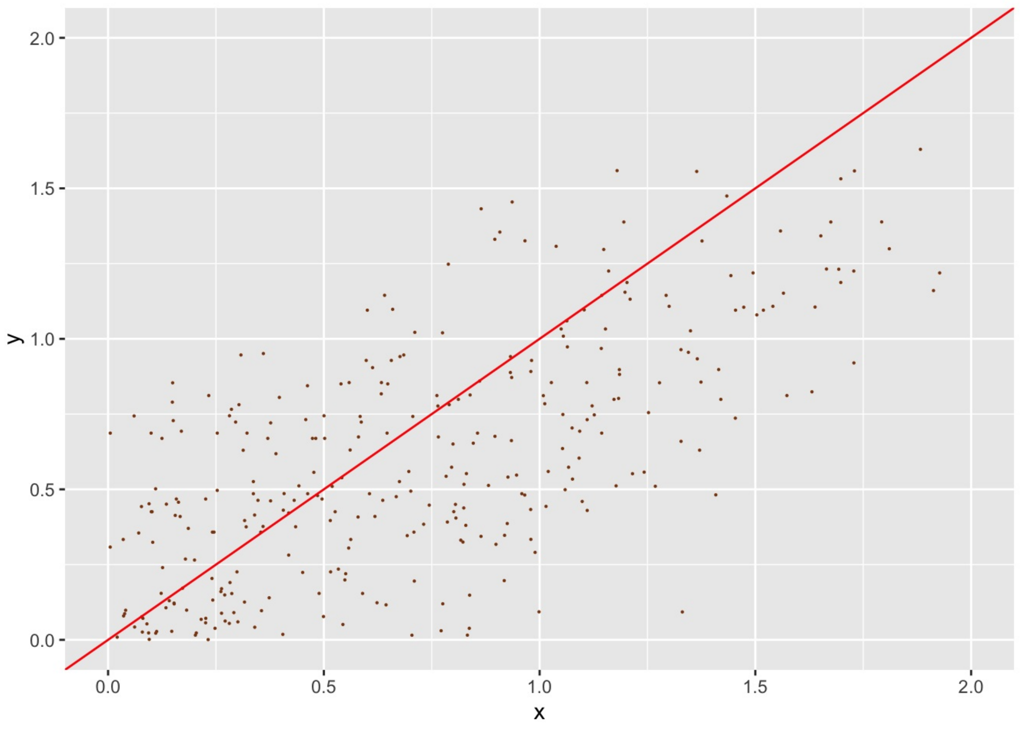
```
## The optimal number of trees is 17
```

```r
#Third, you will want to call `xgboost`, passing in your training data and the optimal number of trees. As it is no
t clear how to get this information: assume your output variable for `xgb.cv()` is simply called `out`. Then the op
timal number of trees is given by `which.min(out$evaluation_log$test_rmse_mean)`. Simple, huh? Also input the same
`params` argument as above.


xgb.out = xgboost(train,nrounds=which.min(xgb.cv.out$evaluation_log$test_rmse_mean),params=list(objective="reg:squa
rederror"),verbose=0)


#Fourth, you want to call `predict()`, with your output from `xgboost` and with `newdata=test`.

resp.pred = predict(xgb.out,newdata=test)

#Finally: compute the test-set MSE and use `ggplot()` to plot the typical regression diagnostic plot that we use in
this class. You should find that the test-set MSE is roughly the same as that for random forest, indicating that bo
osting doesn't necessarily buy you better predictions (but: it does make predictions faster, which means you can pr
ocess more data, so there's consequently less uncertainty on the predictions and the test-set MSE...and you can mor
e easily perform cross-validation).

MSE = round(mean((resp.pred-resp.test)^2),3)
MSE
```

```
## [1] 0.222
```

```r
suppressMessages(library(tidyverse))

ggplot(data=data.frame("x"=resp.test,"y"=resp.pred),mapping=aes(x=x,y=y)) +
  geom_point(size=0.1,color="saddlebrown") + xlim(0,2) + ylim(0,2) +
  geom_abline(intercept=0,slope=1,color="red")
```

```
## Warning: Removed 29832 rows containing missing values (geom_point).
```



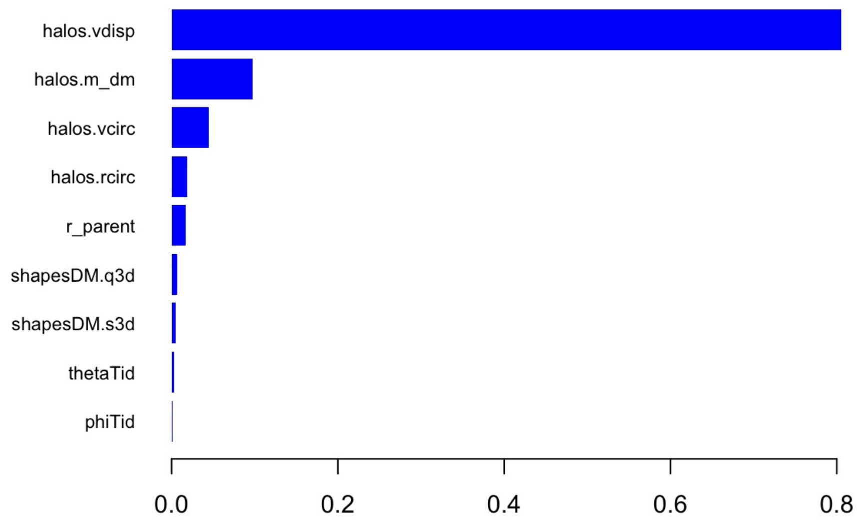
Question 2

Generate an importance plot given the output from training your `xgboost` model. This involves two steps: a call to `xgb.importance()` and a call to `xgb.plot.importance()`. (Display the output from `xgb.importance()`. Note that the importances match what is in the Gain column of the output.) Check the documentation for each. You should discover that `halos.vdisp` is by far the most important variable.

```
imp.out = xgb.importance(model=xgb.out)
imp.out
```

```
##          Feature        Gain      Cover  Frequency
## 1:  halos.vdisp 0.805607303 0.27798641 0.15221774
## 2:   halos.m_dm 0.097272529 0.22140914 0.15221774
## 3:  halos.vcirc 0.045098424 0.10346381 0.11794355
## 4:  halos.rcirc 0.018519067 0.10970326 0.15322581
## 5:     r_parent 0.017182904 0.12599981 0.14112903
## 6: shapesDM.q3d 0.006787005 0.04955371 0.08870968
## 7: shapesDM.s3d 0.005203967 0.06897758 0.07862903
## 8:     thetaTid 0.002702107 0.02750172 0.06955645
## 9:       phiTid 0.001626695 0.01540456 0.04637097
```

```
xgb.plot.importance(importance_matrix=imp.out,col="blue")
```



Now we turn our attention to classification.

Data, Part II

We will now load the second dataset from last week: note that while we will not cut down the sample size, we will still balance the classes. Note that `xgboost` wants integer class labels, so I map "CB" to 0 and "NON-CB" to 1.

```
rm(list=ls())
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/TD_CLASS/css_data.Rdata"
load(url(file.path))
rm(file.path)

# Eliminate the max.slope column (the 11th column), which has infinities.
predictors = predictors[,-11]

set.seed(404)
w.cb = which(response==1)
w.noncb = which(response!=1)
s = sample(length(w.cb),length(w.noncb))
predictors.cb = predictors[w.cb[s],]
response.cb   = response[w.cb[s]]
predictors.noncb = predictors[w.noncb,]
response.noncb   = response[w.noncb]
predictors = rbind(predictors.cb,predictors.noncb)
response   = c(response.cb,response.noncb)

response.new = rep(0,length(response))
w = which(response!=1)
response.new[w] = 1
response = response.new
cat("Sample size: ",length(response),"\n")
```

```
## Sample size:  32452
```

Question 3

You know the drill: split the data, then learn an `xgboost` model and output the test-set MCR value and the confusion matrix. Also create a ROC curve, and determine the AUC. The number should be close to what you observed for random forest, meaning it should be better than what you observed for logistic regression. Finally, plot the importances.

A major difference between this question and Q1: here, the objective is "binary:logistic" as opposed to "reg:squarederror". Also, instead of "test_rmse_mean" as the metric output by `xgb.cv.out`, the metric is now "test_error_mean", and you need to add the argument `eval_metric="error"` to both the calls to `xgb.cv()` and `xgboost()`.

```
library(xgboost)
library(ggplot2)
suppressMessages(library(tidyverse))

set.seed(100)
fraction=.7
sp = sample(nrow(predictors), round(fraction*nrow(predictors)))
pred.train = predictors[sp ,]
pred.test = predictors[-sp ,]

resp = data.frame(response)

s = sample(nrow(resp), round(fraction*nrow(resp)))
resp.train = resp[sp ,]
resp.test = resp[-sp ,]

train = xgb.DMatrix(data=as.matrix(pred.train),label=resp.train)
test = xgb.DMatrix(data=as.matrix(pred.test),label=resp.test)


set.seed(101)
xgb.cv.out = xgb.cv(params=list(objective="binary:logistic"),train,nrounds=30,nfold=5,verbose=0,eval_metric="error"
)
cat("The optimal number of trees is", which.min(xgb.cv.out$evaluation_log$test_error_mean))
```

```
## The optimal number of trees is 29
```

```
xgb.out = xgboost(train,nrounds=which.min(xgb.cv.out$evaluation_log$test_error_mean),params=list(objective="reg:squ
arederror"),verbose=0,eval_metric="error")

resp.pred = predict(xgb.out,newdata=test, type="response")

#and output the test-set MCR value and the confusion matrix. Also create a ROC curve, and determine the AUC. The nu
mber should be close to what you observed for random forest, meaning it should be better than what you observed for
logistic regression. Finally, plot the importances.

#"CB" mapped to 0 and "NON-CB" to 1.


predicted = predict(xgb.out,newdata=test, type="response")

xgb.pred=rep(0, length(response))
xgb.pred[predicted >.5]=1
tab = table(xgb.pred,response)
tab
```

```
##         response
## xgb.pred    0    1
##        0 9755 8647
##        1 6471 7579
```

```
MCRxgb= (9755+7579)/(9755+7579+6471+8647)
MCRxgb
```

```
## [1] 0.5341427
```

```
mean(xgb.pred == response)
```

```
## [1] 0.5341427
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

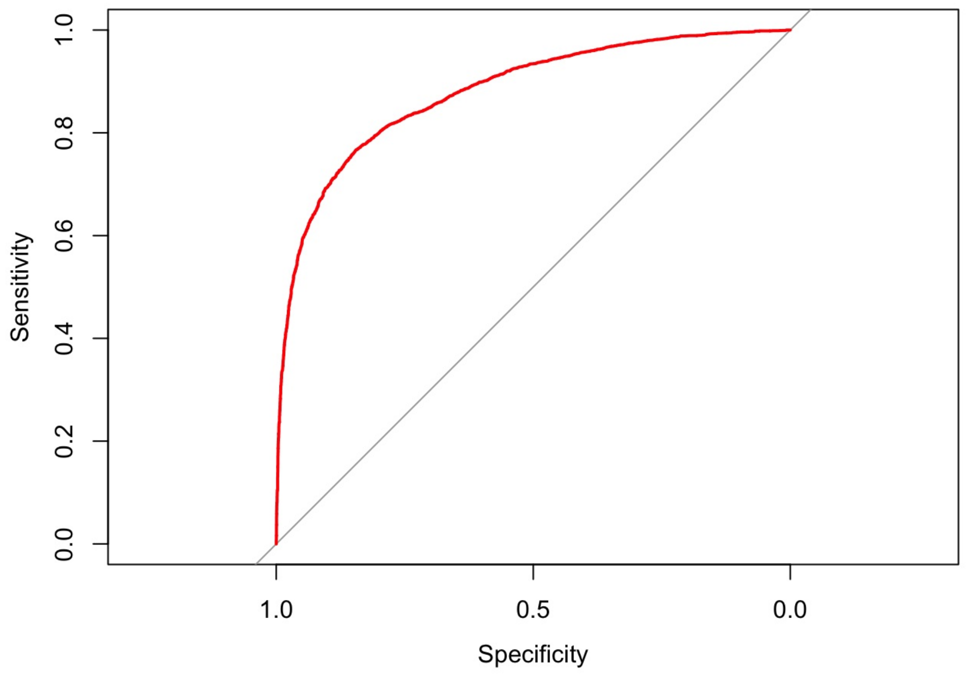
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
roc.xgb = roc(resp.test, predicted)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc.xgb, col="red")
```



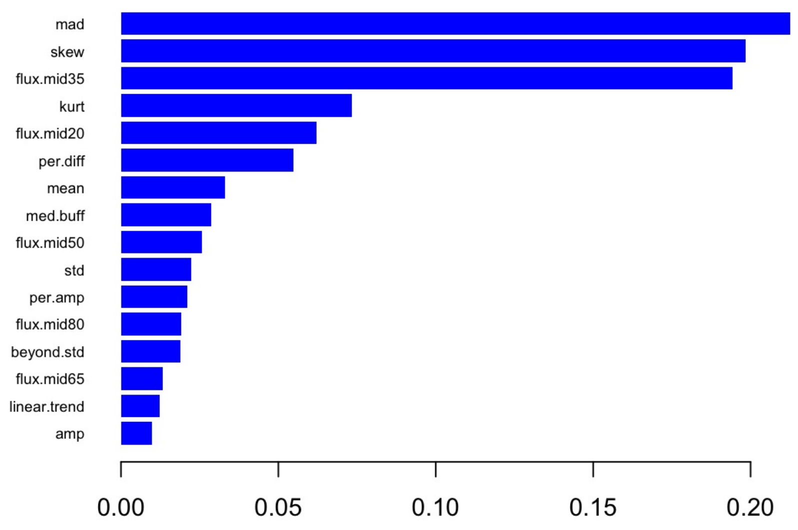
```
roc.xgb$auc
```

```
## Area under the curve: 0.8798
```

```
#AUC = .8798

imp.out = xgb.importance(model=xgb.out)
imp.out
```

```
##          Feature        Gain      Cover  Frequency
##  1:          mad 0.212676263 0.06301717 0.07788595
##  2:         skew 0.198362931 0.15035042 0.09805285
##  3:   flux.mid35 0.194153877 0.05470700 0.05702364
##  4:         kurt 0.073343775 0.08243339 0.07649513
##  5:   flux.mid20 0.062188925 0.07788352 0.07788595
##  6:     per.diff 0.054820229 0.05305701 0.06050070
##  7:         mean 0.032941217 0.07969496 0.08066759
##  8:     med.buff 0.028737729 0.05720779 0.05910987
##  9:   flux.mid50 0.025710722 0.02295813 0.03407510
## 10:          std 0.022299229 0.06538688 0.05424200
## 11:      per.amp 0.021192347 0.07097432 0.03616134
## 12:   flux.mid80 0.019190311 0.05071514 0.05841446
## 13:   beyond.std 0.018904837 0.03309088 0.06536857
## 14:   flux.mid65 0.013294285 0.05621654 0.04311544
## 15: linear.trend 0.012385595 0.06404487 0.04589708
## 16:          amp 0.009797729 0.01826198 0.07510431
```

```
xgb.plot.importance(importance_matrix=imp.out,col="blue")
```


mad
skew
flux.mid35
kurt
flux.mid20
per.diff
mean
med.buff
flux.mid50
std
per.amp
flux.mid80
beyond.std
flux.mid65
linear.trend
amp
0.00
0.05
0.10
0.15
0.20