Stats 412 Homework 3

Chris Meade

5/23/2018

Music Genre

This data set was published as a contest data set on the TunedIT web site (http://tunedit.org/challenge/music-retrieval/genres). In this competition, the objective was to develop a predictive model for classifying music into six categories. In total, there were 12,495 music samples for which 191 characteristics were determined. All predictors were continuous; many were highly correlated and the predictors spanned different scales of measurement. This data collection was created using 60 performers from which 15–20 pieces of music were selected for each performer. Then 20 segments of each piece were parameterized in order to create the final data set. Hence, the samples are inherently not independent of each other.

```
library(readr)
genres <- read_csv("https://raw.githubusercontent.com/natelangholz/stat412-advancedregression/master/we
```

1

Random Forest: Fit a random forest model using both CART trees and conditional inference trees to the music genre predictors. What are the differences in the models? Do you have any difficulty on the full data set?

```
# Random Forest with CART
library(randomForest) #for quicker training time, multithreading
# Parallel Processing
library(doMC)
```

```
## Warning: package 'doMC' was built under R version 3.4.3
```

```
registerDoMC(cores = detectCores())

# Subset the data
sampleIndices <- sample(1:nrow(genres), 2000)
train <- genres[sampleIndices,]

# Train CART RF
RFcart <- randomForest(as.factor(GENRE) ~., data = train)
```

Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance.

```
library(party)
RFcond <- cforest(as.factor(GENRE) ~., data = train)
importance <- varimp(RFcond)
head(sort(importance, decreasing = T)) # variable importance
```

```
## PAR_SFMV24  PAR_SFM24  PAR_SFM22  PAR_MFCC1 PAR_MFCCV1  PAR_ASE32
## 0.03019293 0.02529076 0.01684239 0.01399457 0.01309511 0.01142391
```

Both of these models utilize bagging to create an ensemble of weak learner. However, the random forest function in the `ranger` package utilizes traditional CARTs and random variable selection for each tree to

build this ensemble, whereas the `cforest` function utilizes conditional inference trees, which selects tree splits based on predictors with a strong association with the response variable.

Both Random Forest models had prohibitively long training times on the full dataset. To combat this, I train the models with a subset of the data.

We assess the 'best' model using a simple measure of prediction accuracy.

```
predictCART <- predict(RFcart, genres)
predictCOND <- predict(RFcond, train, OOB=TRUE, type = "response")
cat("RFcart Accuracy = ", mean(genres$GENRE == predictCART),
    "\nRFcond Accuracy = ", mean(train$GENRE == predictCOND))
```

```
## RFcart Accuracy =  0.9156463
## RFcond Accuracy =  0.8525
```

Clearly the CART based model performs better.

2

Data Splitting: What data splitting method(s) would you use for these data? Explain.

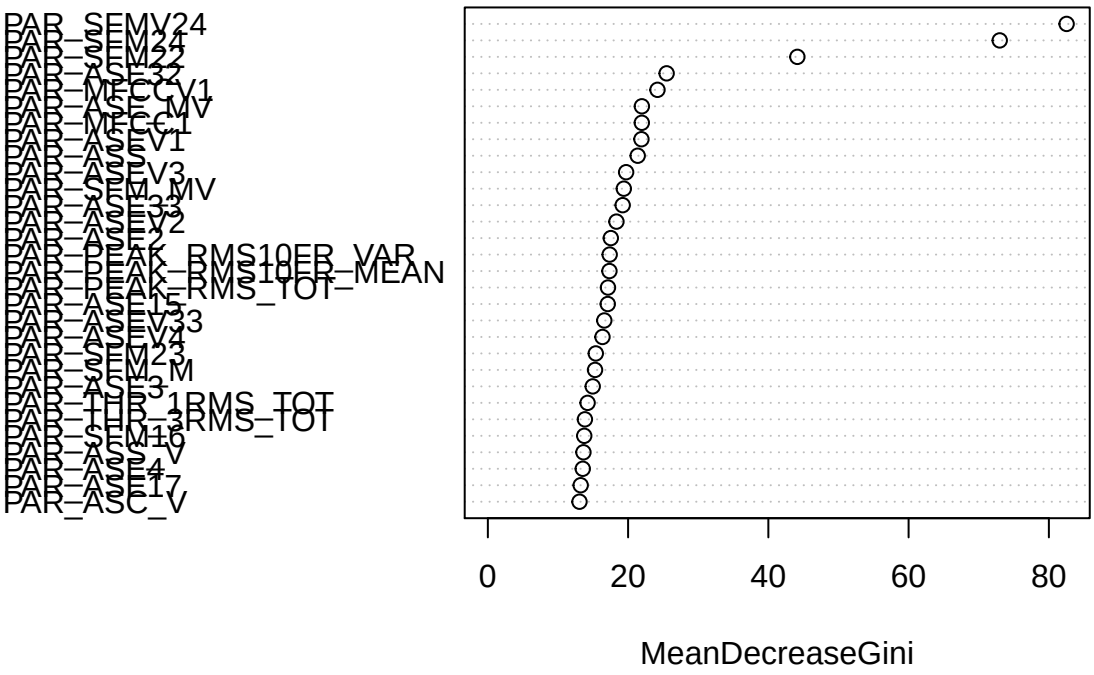
Since this dataset is reasonably large, I would first split it into 80-20 training and test sets. Then I would perform 5-fold cross validation for each model on the training set to select the best hyperparemters for each. I choose 5-fold since these models have a relatively long training time, so any more folds or ever LOOCV would be too computationally prohibitive. Finally, I would compare the performance of both models on the test set to choose the best.

3

Variable Importance: Create a variable importance plot from you best model. What features are most important?

Below we plot the variable importance plot of the Random Forest built with an ensemble of classical classifiction trees.

```
varImpPlot(RFcart)
```



We note that the top 4 most important variables are the same between both models.

Simulated data

Friedman introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:

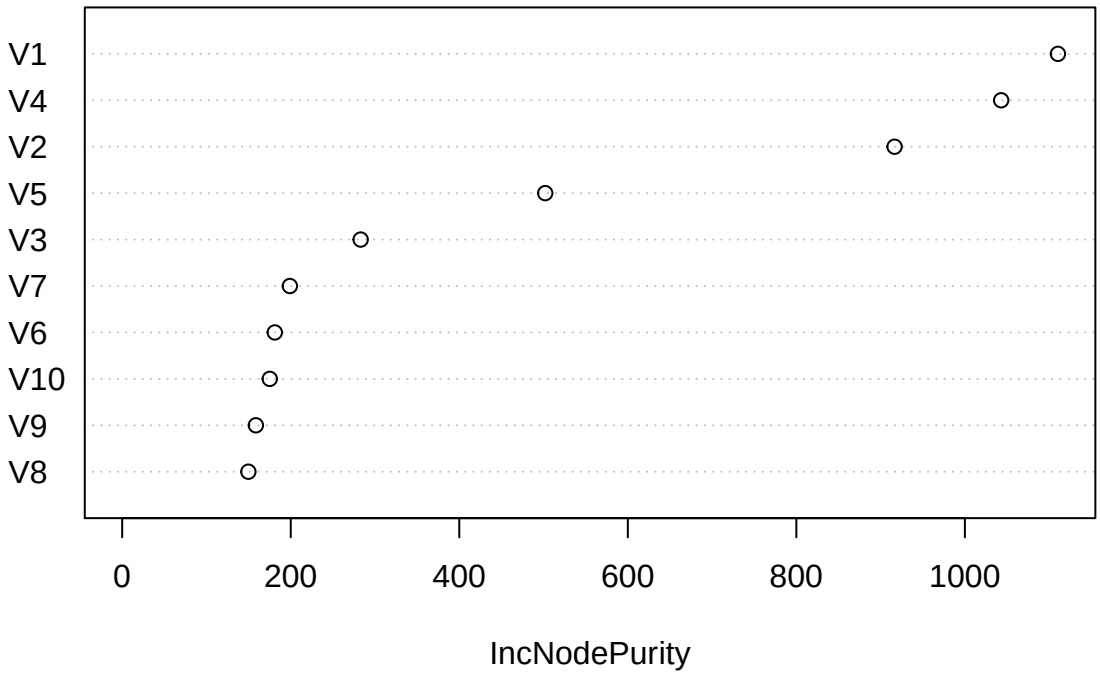
where the x values are random variables uniformly distributed between [0, 1] (there are also 5 other non-informative variables also created in the simulation). The package mlbench contains a function called `mlbench.friedman1` that simulates these data

```
library(mlbench)
set.seed(200)
simulated <- mlbench.friedman1(200, sd = 1)
simulated <- cbind(simulated$x, simulated$y)
simulated <- as.data.frame(simulated)
colnames(simulated)[ncol(simulated)] <- "y"
```

4

Random Forest and Variable Importance: Fit a random forest model to all of the predictors, then estimate the variable importance scores. Did the random forest model significantly use the uninformative predictors (V6 – V10)?

```
simFit <- randomForest(y~., data = simulated, ntree = 2000)
varImpPlot(simFit)
```



The random forest did not significantly use variables V6-V10, as we would hope

5

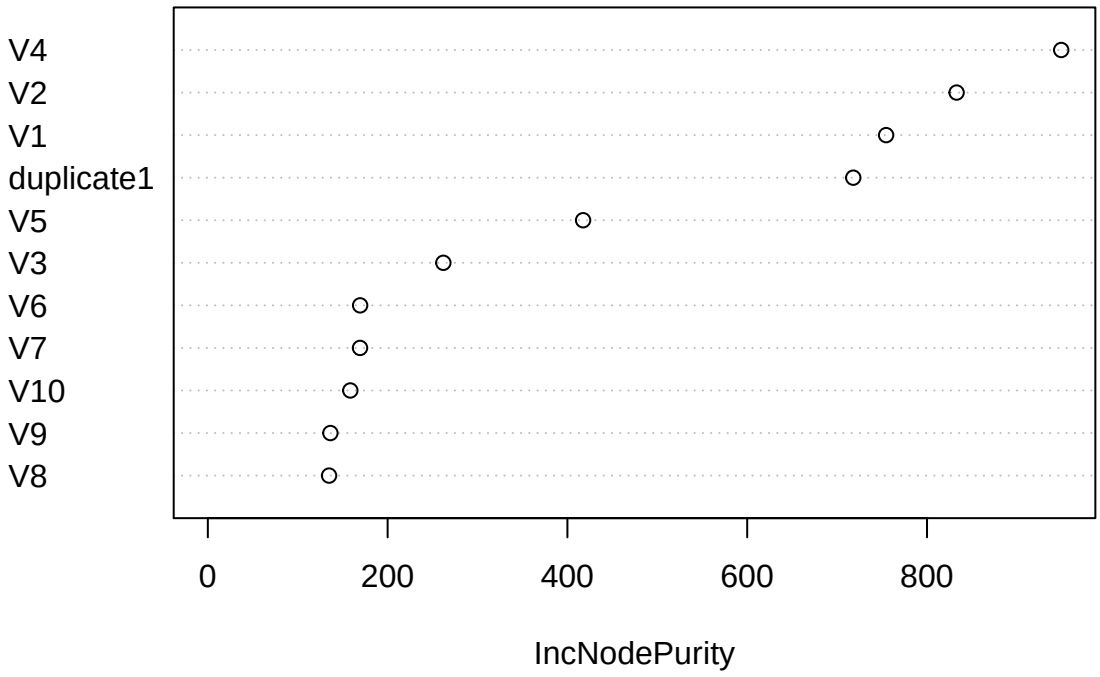
Correlated Predictor:Now add an additional predictor that is highly correlated with one of the informative predictors. For example:

```
simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
cor(simulated$duplicate1, simulated$V1)
```

```
## [1] 0.9395354
```

Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

```
simFit2 <- randomForest(y~., data = simulated, ntree = 2000)
varImpPlot(simFit2)
```

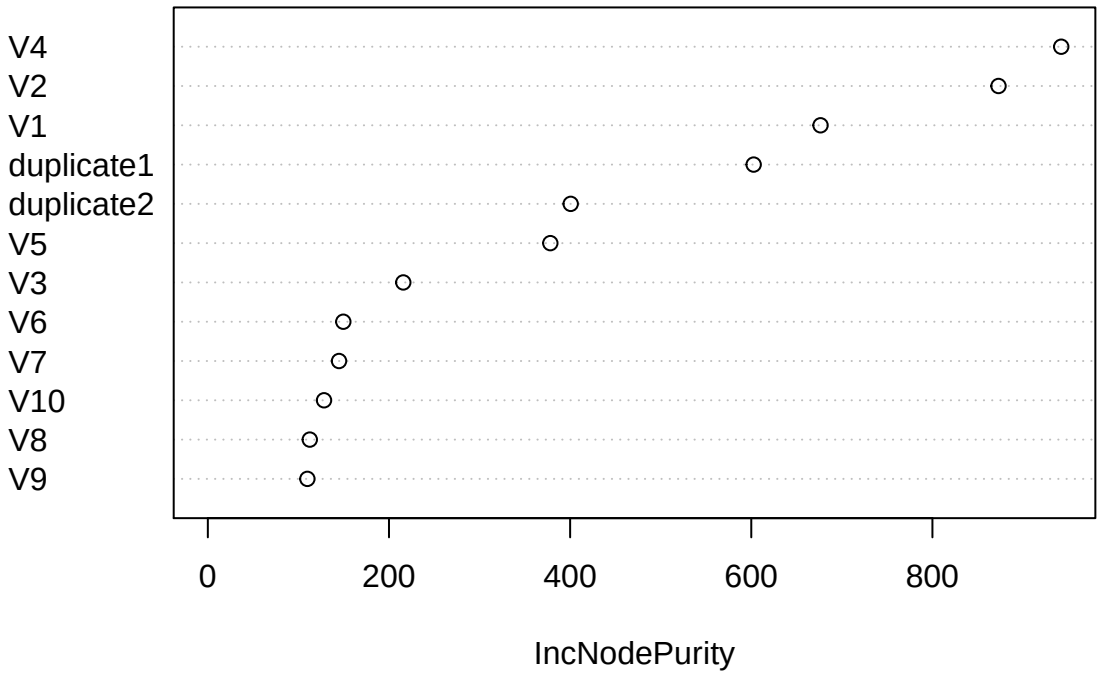


When this highly correlated variable is added, the importance of V1 decreases sharply. We now add a second variable with high correlation to V1, then fit another model

```
simulated$duplicate2 <- simulated$V1 + rnorm(200) * .1
cor(simulated$duplicate2, simulated$V1)
```

```
## [1] 0.9472513
```

```
simFit3 <- randomForest(y~., data = simulated, ntree = 2000)
varImpPlot(simFit3)
```



Again, we see that the importance of V1 decreases. Because of the high correlation between these three predictors, the model does not rely on V1 as often in making splits, thereby decreasing its importance.

6

Gradient Boosted Machine: Repeat the process in 5 and 6 with different tree models, such as boosted trees. Does the same pattern occur?

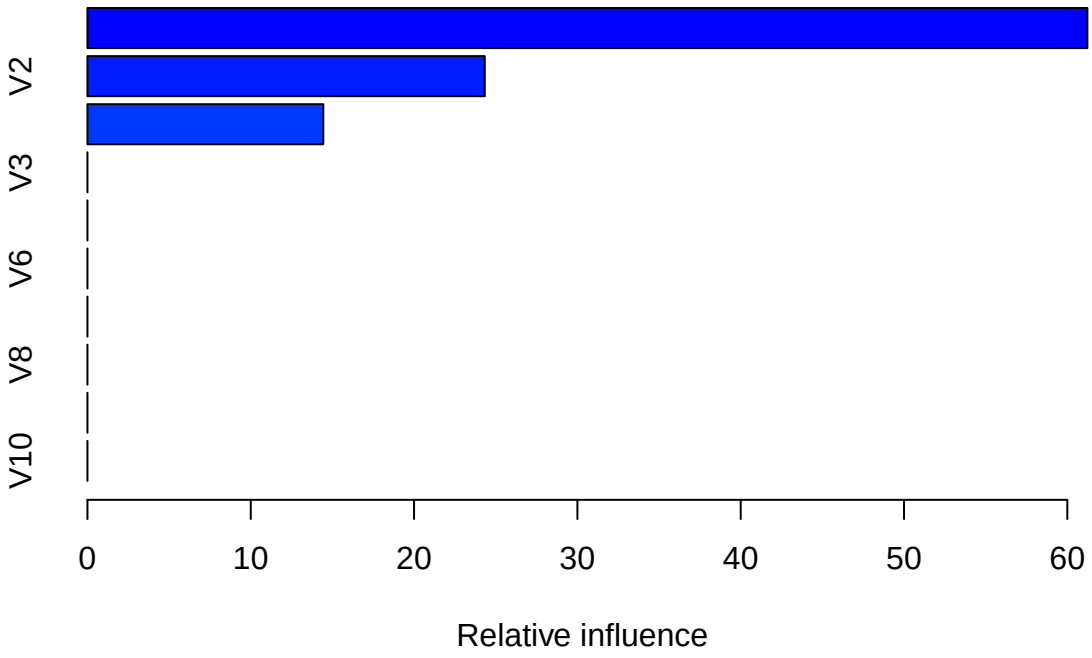
```
library(gbm)

set.seed(200)
simulated2 <- mlbench.friedman1(200, sd = 1)
simulated2 <- cbind(simulated2$x, simulated2$y)
simulated2 <- as.data.frame(simulated2)
colnames(simulated2)[ncol(simulated2)] <- "y"

GBMfit1 <- gbm(y~., data = simulated2)
```

```
## Distribution not specified, assuming gaussian ...
```

```
summary(GBMfit1)
```




```
##     var  rel.inf
## V1   V1 61.22786
## V2   V2 24.32941
## V4   V4 14.44273
## V3   V3  0.00000
## V5   V5  0.00000
## V6   V6  0.00000
## V7   V7  0.00000
## V8   V8  0.00000
## V9   V9  0.00000
## V10 V10  0.00000
```

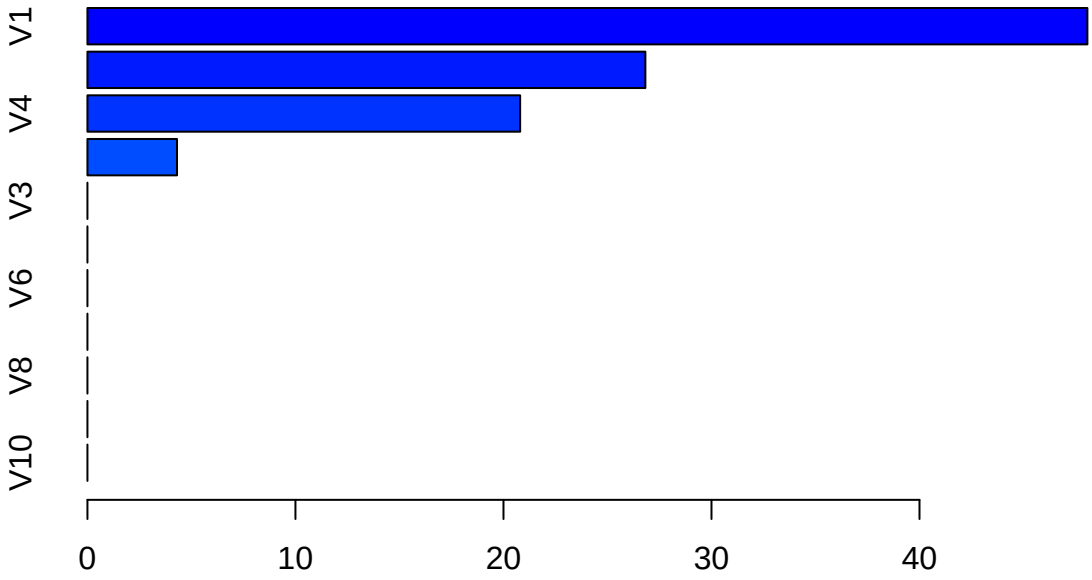
```
simulated2$duplicate1 <- simulated2$V1 + rnorm(200) * .1
cor(simulated2$duplicate1, simulated2$V1)
```

```
## [1] 0.9425726
```

```
GBMfit2 <- gbm(y~., data = simulated2)
```

```
## Distribution not specified, assuming gaussian ...
```

```
summary(GBMfit2)
```




```
##                   var   rel.inf
## V1                 V1 48.070939
## V2                 V2 26.821641
## V4                 V4 20.801028
## duplicate1 duplicate1  4.306392
## V3                 V3  0.000000
## V5                 V5  0.000000
## V6                 V6  0.000000
## V7                 V7  0.000000
## V8                 V8  0.000000
## V9                 V9  0.000000
## V10               V10  0.000000
```

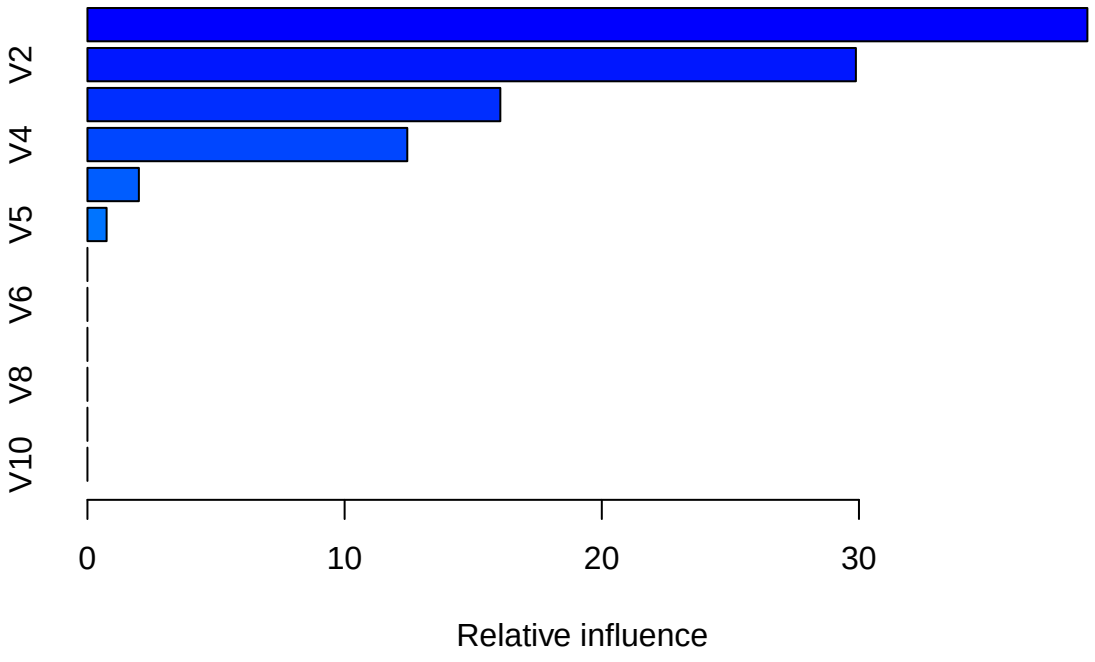
```
simulated2$duplicate2 <- simulated2$V1 + rnorm(200) * .1
cor(simulated2$duplicate2, simulated2$V1)
```

```
## [1] 0.9484525
```

```
GBMfit3 <- gbm(y~., data = simulated2)
```

```
## Distribution not specified, assuming gaussian ...
```

```
summary(GBMfit3)
```




```
##                    var    rel.inf
## V1                  V1 38.8819646
## V2                  V2 29.8800013
## duplicate2 duplicate2 16.0551465
## V4                  V4 12.4374939
## duplicate1 duplicate1  2.0001146
## V5                  V5  0.7452791
## V3                  V3  0.0000000
## V6                  V6  0.0000000
## V7                  V7  0.0000000
## V8                  V8  0.0000000
## V9                  V9  0.0000000
## V10                V10  0.0000000
```

With the gradient boosted machine models, we see the same pattern as in the random forests. As more correlated predictors are added, the relative importance of V1 decreases.

Pulling Punches Part 2

The two `.Rdata` files under week 7 come as an abbreviated version of punch profiles from a boxing system to measure acceleration of boxers during live fights. The `gridded` list from the second file below has each individual punch profile which has a list item being a 3 column data frame of time (in ms around the middle of the punch event), acceleration in x (forward-back in g's), and acceleration in y (side-to-side in g's). Also attached are some other fields which are of less importance and contribute to this being a somewhat messy data set.

```
library(curl)
```

```
## Warning: package 'curl' was built under R version 3.4.4
```

```
##
## Attaching package: 'curl'
```

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

```
##
##     parse_date
```

```
load(file = curl('https://github.com/natelangholz/stat412-advancedregression/raw/master/week-7/problem-s
load(file = curl('https://github.com/natelangholz/stat412-advancedregression/raw/master/week-7/problem-s
```

There are 1000 punch profiles each with an associated force (in Newtons) and boxer who threw the punch. Use the `ff` data frame as ground truth for punch force (variable =`force`) in addition to the rest of the boxer information. Other boxer information included variables around their size to be a proxy for 'effective mass'.

7

Estimations: Use features (and/or new features) created from your problem set 2 to estimate punch force using a MARS model. Use RMSE as your error estimate

```
# Use features from last time
profiles <- gridded
datalist = list()

for(i in 1:nrow(ff)){
  row <- c()
  row[1] <- max(profiles[[i]]$profile[,3]) #Max Y
  row[2] <- min(profiles[[i]]$profile[,3]) #Min Y
  row[3] <- (row[1]-row[2])/(which.max( profiles[[i]]$profile[,3])-which.min( profiles[[i]]$profile[,3])
  row[4] <- max(profiles[[i]]$profile[,2]) #Max X
  row[5] <- min(profiles[[i]]$profile[,2]) #Min X
  row[6] <- (row[4]-row[5])/(which.max( profiles[[i]]$profile[,2])-which.min( profiles[[i]]$profile[,2])
  row[7] <- IQR(profiles[[i]]$profile[,3]) #IQR(Y)
  row[8] <- IQR(profiles[[i]]$profile[,2]) #IQR(X)
  row[9] <- median(profiles[[i]]$profile[,3]) #Median(Y)
  row[10] <- median(profiles[[i]]$profile[,2]) #Median(X)
  row[11] <- ff$hands[i] #left or right handed
  row[12] <- ff$stance[i] #stance
  row[13] <- ff$wingspan[i] #wingspan
  row[14] <- ff$punch.type[i] #pt
  row[15] <- ff$boxer[i] #boxer
  row[16] <- ff$wingspan[i] #wingspan
  row[17] <- ff$force[i]#force
  datalist[[i]] <- row
}

profileDF <- as.data.frame(do.call("rbind", datalist))
head(profileDF)
```

```
##          V1        V2         V3       V4        V5        V6       V7
## 1 14.928572 -30.98748 -11.479013 30.21429 -12.11756 0.8819134 1.478627
## 2 11.642858 -43.07143 -18.238096 30.21429 -10.63530 0.8880345 1.675506
## 3 12.000000 -31.73585 -21.867926 30.21429 -11.33941 0.8480346 1.760299
## 4  6.305743 -42.66853   1.579815 30.21429 -12.34882 0.8512621 1.728412
## 5 11.642858 -23.89056  -5.076203 30.21429 -11.02476 0.8965010 1.424584
## 6  7.131197 -30.28571  -0.656437 30.21429 -11.95431 0.7956339 1.422034
##          V8         V9        V10 V11 V12 V13 V14 V15 V16      V17
## 1 0.5741310 -0.2141512 -0.6951871   1   0  66   1   2  66 3237.317
## 2 0.5230862 -0.2798638 -0.7519638   1   0  66   1   2  66 4014.273
## 3 0.4491911 -0.1548346 -0.7360643   1   0  66   1   2  66 3690.541
```

```
## 4 0.5730037 -0.1480877 -0.8753393   1   0  66   1   2  66 3981.900
## 5 0.6279214 -0.2476954 -0.7111719   1   0  66   1   2  66 3917.154
## 6 0.6908476 -0.1540652 -0.7274457   1   0  66   1   2  66 4046.646
```

We first build a MARS model:

```
library(earth)
```

```
## Warning: package 'earth' was built under R version 3.4.4
```

```
library(caret)
MARSfit <- earth(V17~., data = profileDF)
MARSpredict <- predict(MARSfit, profileDF)
RMSE(MARSpredict, profileDF$V17)
```

```
## [1] 709.7185
```

8

Estimations improved Now try a few different (gbm, randomForest) models through the `caret` package and different data splitting techniques to compare. Comparing RMSE which model performs the best?

```
# Train Test Split
library(caret)
set.seed(3456)
trainIndex <- createDataPartition(profileDF$V17, p = .8,
                                  list = FALSE,
                                  times = 1)

profileTrain <- profileDF[trainIndex,]
profileTest <- profileDF[-trainIndex,]

tc <- trainControl(method = "repeatedcv",
                   number = 10,
                   repeats = 5)

library(earth)
fitMARS <- earth(V17~., data = profileTrain)
predictMARS <- predict(fitMARS, profileTest)



fitRF <- train(V17~.,
               method = "rf",
               data = profileTrain,
               metric = "RMSE",
               trControl = tc,
               preProcess = c("center", "scale", "YeoJohnson"))
predictRF <- predict(fitRF, profileTest)


fitGBM <- train(V17~.,
               method = "gbm",
               data = profileTrain,
               metric = "RMSE",
               trControl = tc,
```

```
                preProcess = c("center", "scale", "YeoJohnson"))
predictGBM <- predict(fitGBM, profileTest)

fitXGB <- train(V17~.,
                method = "xgbTree",
                data = profileTrain,
                metric = "RMSE",
                trControl = tc,
                preProcess = c("center", "scale", "YeoJohnson"))
predictXGB <- predict(fitXGB, profileTest)

tc2 <- trainControl(method = "boot632",
                    number = 20)

fitRF2 <- train(V17~.,
                method = "rf",
                data = profileTrain,
                metric = "RMSE",
                trControl = tc2,
                preProcess = c("center", "scale", "YeoJohnson"))
predictRF2 <- predict(fitRF2, profileTest)

fitGBM2 <- train(V17~.,
                method = "gbm",
                data = profileTrain,
                metric = "RMSE",
                trControl = tc2,
                preProcess = c("center", "scale", "YeoJohnson"))
predictGBM2 <- predict(fitGBM2, profileTest)

fitXGB2 <- train(V17~.,
                method = "xgbTree",
                data = profileTrain,
                metric = "RMSE",
                trControl = tc2,
                preProcess = c("center", "scale", "YeoJohnson"))
predictXGB2 <- predict(fitXGB2, profileTest)
```

MARS RMSE = 810.054 RF RMSE w/ CV = 486.4612 RF RMSE w/ boot632 = 484.0815 GBM RMSE w/ CV = 608.4163 GBM RMSE w/ boot632 = 625.8724 XGBoost RMSE w/ CV = 573.7047 XGBoost RMSE w/ boot632 = 568.7018