ML Final Project

Atsumi Kainosho & Haley Townsend

May 8, 2019

```r
# First, load libraries necessary for the coding part of the project.

library(Hmisc)
```

```
## Warning: package 'Hmisc' was built under R version 3.5.3
## Loading required package: lattice
## Loading required package: survival
## Warning: package 'survival' was built under R version 3.5.3
## Loading required package: Formula
## Loading required package: ggplot2
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```r
library(tidyr)
library(plyr)
```

```
##
## Attaching package: 'plyr'
## The following objects are masked from 'package:Hmisc':
##
##     is.discrete, summarize
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
## The following objects are masked from 'package:Hmisc':
##
##     src, summarize
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(readr)
library(haven)
```

```
## Warning: package 'haven' was built under R version 3.5.3
```

```r
library(foreign)
library(RNHANES)
```

```
## Warning: package 'RNHANES' was built under R version 3.5.3
```

```r
library(purrr)
```

```
##
## Attaching package: 'purrr'

## The following object is masked from 'package:plyr':
##
##     compact
```

```r
library(magrittr)
```

```
##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following object is masked from 'package:tidyr':
##
##     expand

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loaded glmnet 2.0-16
```

```r
library(keras)
```

```
## Warning: package 'keras' was built under R version 3.5.3
```

```r
library(stringr)
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.5.3
```

```
##
## Attaching package: 'data.table'

## The following object is masked from 'package:purrr':
##
##     transpose

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year

## The following object is masked from 'package:plyr':
##
##     here

## The following object is masked from 'package:base':
##
##     date
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.3

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

## The following object is masked from 'package:survival':
##
##     cluster
```

```r
library(ROCR)
```

```
## Loading required package: gplots

##
## Attaching package: 'gplots'

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

```r
library(rpart)
```

```
##
## Attaching package: 'rpart'

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

```
##     solder
```

```
library(rpart.plot)
library(glmnet)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ada)
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
# Import data set from National Health and Nutrition Examination Survey (NHANES) web site.
# We used National  Youth  Fitness  Survey  (NNYFS) for 2012, specifically, Demographic Variables & Sam
# Since the data format was SAS, we extracted data into CSV file first, and then read the file to use t

# NHANES NNYFS  Demographic Variables & Sample Weights (Y_DEMO) data
demo = read_xpt("C:\\Users\\atsum\\Documents\\GitHub\\ML-Final-Project\\Y_DEMO.xpt")
write.csv(demo, file = 'C:\\Users\\atsum\\Documents\\GitHub\\ML-Final-Project\\attempt.csv')
demodf = read_csv("C:\\Users\\atsum\\Documents\\GitHub\\ML-Final-Project\\attempt.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]

## Parsed with column specification:
## cols(
##   .default = col_double()
## )

## See spec(...) for full column specifications.
```

```
# Body Measures (Y_BMX) data that includes BMI information
body = read_xpt('C:\\Users\\atsum\\Documents\\GitHub\\ML-Final-Project\\Y_BMX.xpt')
write.csv(body, file = 'body.csv')
bodydf=read_csv('body.csv')
```

```
## Warning: Missing column names filled in: 'X1' [1]

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   BMIARML = col_logical()
## )
## See spec(...) for full column specifications.
```

```
# After reading datasets, we implemented data cleaning.

# First, we remove objects created to import SAS files in order to avoid confusions.
```

```r
rm(demo)
rm(body)

# Remove the X1 columns from both dfs that manage row numbers
demodf = demodf %>% select(-X1)
bodydf = bodydf %>% select(-X1)

# Join the demographic data to the body data using SEQN that represents Respondent sequence number
df = left_join(bodydf, demodf, by = 'SEQN')

# Look at where there is missing data
colSums(is.na(df))
```

```
##     SEQN BMDSTATS    BMXWT    BMIWT    BMXHT    BMIHT   BMXBMI  BMDBMIC
##        0        0        5     1554        5     1574        5        5
##  BMXARML  BMIARML  BMXARMC  BMIARMC BMXWAIST BMIWAIST  BMXCALF  BMICALF
##        8     1576        7     1575        8     1575       10     1574
## BMXCALFF BMICALFF   BMXTRI   BMITRI   BMXSUB   BMISUB RIDSTATR RIAGENDR
##      134     1451       23     1561       69     1513        0        0
## RIDAGEYR RIDRETH1 RIDEXMON RIDEXAGY DMDBORN4 DMDEDUC3  SIALANG    WTINT
##        0        0        0        0        0      353        0        0
##    WTMEC  SDMVPSU SDMVSTRA INDHHIN2 INDFMIN2 INDFMPIR DMDHHSIZ DMDFMSIZ
##        0        0        0        6        0      100        0        0
## DMDHHSZA DMDHHSZB DMDHHSZF DMDHRGND DMDHRAGE DMDHRBR4 DMDHREDU DMDHRMAR
##        0        0        0        0        0       13       13        6
```

```r
# Include only the demographic columns of interest and the Y variable (BMDBMIC)
df = df %>% select(RIAGENDR, RIDEXAGY,RIDRETH1, DMDBORN4, INDHHIN2, DMDHHSZA, BMDBMIC, SEQN)

# Rename the columns, so they are easier to understand
names(df)[1]<-"gender"
names(df)[2]<-"age"
names(df)[3]<-"race"
names(df)[4]<-"birth_country"
names(df)[5]<-"annual_household_income"
names(df)[6] <-"num_children_5yrs_younger"
names(df)[7] <-"Y_BMI_category"

# Drop rows containing NA for the outcome (BMI) immediately
df = df %>% filter(!is.na(Y_BMI_category))

# Clean the outcome variable so it is a biclassification problem instead of multi-class
# Relevel the outcome or BMI (Y) to have just two levels: Y is obese (obese:4) and
#N is not obese (which includes underweight:1, normal:2, overweight:3)
df = df %>% mutate(Y = case_when(
  Y_BMI_category == 2 ~ 'N',
  Y_BMI_category == 1 ~ 'N',
  Y_BMI_category == 3 ~ 'N',
  Y_BMI_category == 4 ~ 'Y'
)) %>% select(-Y_BMI_category)

# Change levels of factors for predictors: gender, race, birth_country, and the outcome
cols <- c('gender', 'race', 'birth_country', 'Y')
```

```r
df[cols] <- lapply(df[cols], factor)

# Gender, 1 is male and 2 is female
df$gender <- ordered(df$gender,
                     levels = c(1,2),
                     labels = c("male", "female"))

# Race, 1 is Mexican American, 2 is Other Hispanic, 3 Non-Hispanic White, 4 Non-Hispanic Black, 5 Other
df$race <- ordered(df$race,
                   levels = c(1,2,3,4,5),
                   labels = c("Mexican American", "Other Hispanic","White",
                              "Black", "Other"))

# birth_country, 1 is US, 2 is Other
df$birth_country <- ordered(df$birth_country,
                            levels = c(1,2),
                            labels = c("USA", "Other"))

# Modify the annual household income variable to enable us to use numeric income information instead of
# 77 represents "Refused" and 99 respresents "Don't Know", which equals to no income value information.
#We convert them as NA first, which gives us 45 new NAs.
df$annual_household_income = na_if(df$annual_household_income, 99)
df$annual_household_income = na_if(df$annual_household_income, 77)
sum(is.na(df$annual_household_income))
```

```
## [1] 45
```

```r
# Modify annual_household_income into numeric using the middle value in each group
df = df %>% mutate(annual_household_income_num = case_when(
  annual_household_income == 1 ~ 2500,
  annual_household_income == 2 ~ 7500,
  annual_household_income == 3 ~ 12500,
  annual_household_income == 4 ~ 17500,
  annual_household_income == 5 ~ 22500,
  annual_household_income == 6 ~ 30000,
  annual_household_income == 7 ~ 40000,
  annual_household_income == 8 ~ 50000,
  annual_household_income == 9 ~ 60000,
  annual_household_income == 10 ~ 70000,
  annual_household_income == 12 ~ 30000,
  annual_household_income == 13 ~ 10000,
  annual_household_income == 14 ~ 87500,
  annual_household_income == 15 ~ 100000
  )) %>% select(-annual_household_income)

# Remove unique ID from main dataframe
df = df %>% select(-SEQN)

# Drop rows incluging NAs
df = df %>% na.omit()

# Show summary of created dataset, which equals to Table 1 in the report.
df %>% summary()
```

```
##     gender          age                          race      birth_country
##  male  :768   Min.   : 3.00   Mexican American:228   USA   :1439
##  female:758   1st Qu.: 6.00   Other Hispanic  :225   Other:   87
##               Median : 9.00   White           :610
##               Mean   : 9.09   Black           :338
##               3rd Qu.:12.00   Other           :125
##               Max.   :16.00
##  num_children_5yrs_younger Y        annual_household_income_num
##  Min.   :0.0000            N:1227   Min.   :  2500
##  1st Qu.:0.0000            Y: 299   1st Qu.: 22500
##  Median :0.0000                     Median : 40000
##  Mean   :0.6494                     Mean   : 51837
##  3rd Qu.:1.0000                     3rd Qu.: 87500
##  Max.   :3.0000                     Max.   :100000
```

```
## Building a Base Model: Logistic Regression using base features: gender, age, race, and birth country

# Create a new dataset df_1 ontaining base features and the outcome
df_1 = df %>% select(gender, age, race, birth_country, Y)

# Set seed first to create training and test datasets
set.seed(518) # our graduation date!

# Shuffle sampes for randamization
shuffled_df_1 = df_1[sample(1:nrow(df_1)),]

# Assign 75% of the total samples into training set and 25% into test set.
# Traning set contains 1144 samples and test set has 382 samples.
n = round((0.75 * nrow(shuffled_df_1)),0)
train_1 = df_1[1:n,]
test_1 = df_1[-(1:n),]

# Confirm that train and test are balanced with the outcome of interest (Y)
train_1 %>% select(Y) %>% table() %>% prop.table()
```

```
## .
##         N         Y
## 0.7998252 0.2001748
```

```
test_1 %>% select(Y) %>% table() %>% prop.table()
```

```
## .
##         N         Y
## 0.8167539 0.1832461
```

```
# Run a base model, Logistic Regression
lr_1 = with(train_1, glm(Y=="Y" ~.,
                     family = binomial("logit"),
                     data = train_1))
lr_1 %>% summary()
```

```
##
## Call:
## glm(formula = Y == "Y" ~ ., family = binomial("logit"), data = train_1)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -0.9418  -0.7186  -0.6152  -0.4570   2.3472
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -2.49524    0.33856  -7.370  1.7e-13 ***
## gender.L        -0.21215    0.10634  -1.995  0.04604 *
## age              0.04619    0.02076   2.225  0.02607 *
## race.L          -0.34766    0.23492  -1.480  0.13889
## race.Q          -0.11374    0.20950  -0.543  0.58721
## race.C          -0.34812    0.19048  -1.828  0.06761 .
## race^4          -0.42927    0.15378  -2.791  0.00525 **
## birth_country.L -1.02291    0.37488  -2.729  0.00636 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1145.5  on 1143  degrees of freedom
## Residual deviance: 1115.1  on 1136  degrees of freedom
## AIC: 1131.1
##
## Number of Fisher Scoring iterations: 5
```

```
## Building Additional Models and Comparing Performance

# Under the estimation that ensemble methods can improve the model performance
# since they aggregate multiple weak learners produced by small number of samples,
# we implemented Random Forest, AdaBoosting, and Gradient Boosted Forest.

# Implement Random Forest
forest_1 = randomForest(formula = Y ~ .,
                      data=train_1, ntrees=10)

# Implement AdaBoosting
aforest_1 = ada(formula = Y=="Y" ~ .,
              data=train_1,
              iter=10)

# Implement Gradient Boosted Forest
gforest_1 = gbm(formula = Y=="Y" ~ .,
              data=train_1 %>%
                mutate_if(is.logical, as.factor),
              interaction.depth=10,
              cv.folds = 2)
```

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

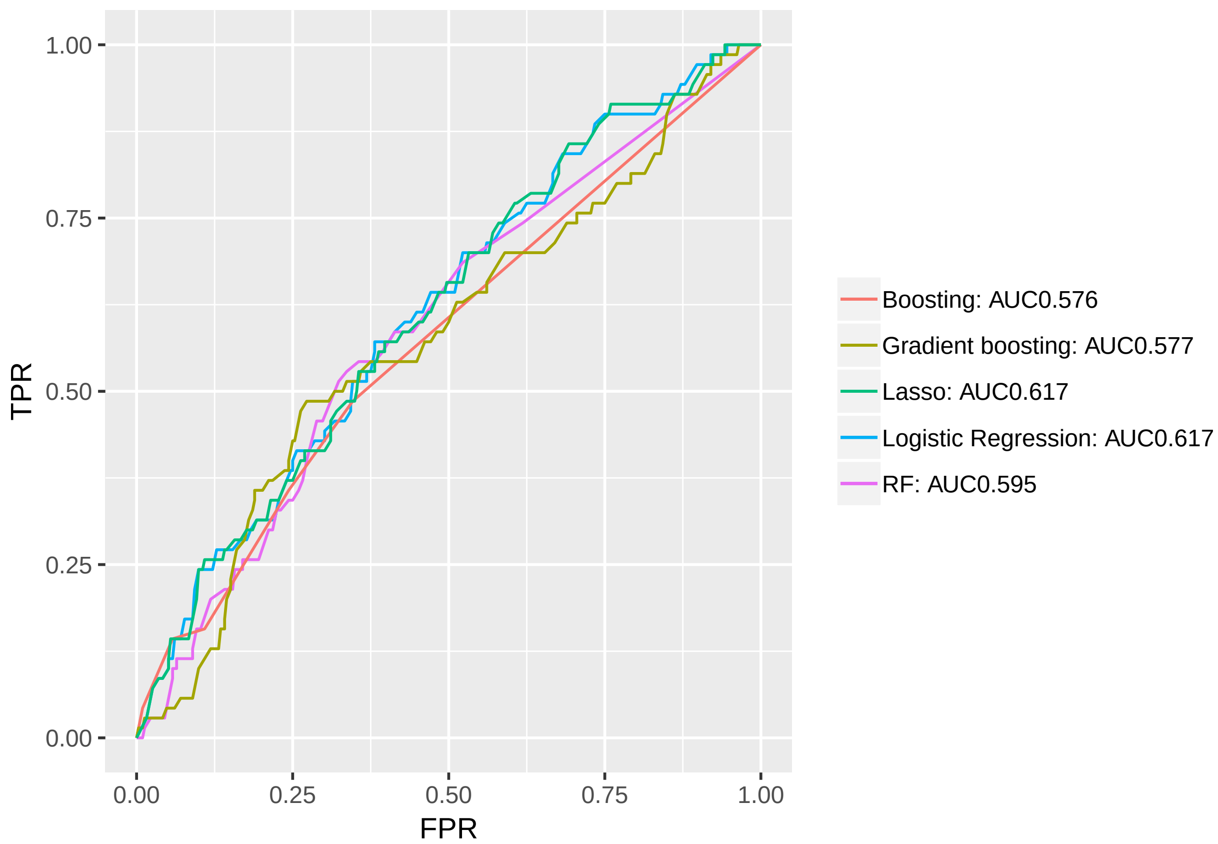
```
# Since the sample size is small, to avoid overfitting, we see whether there are variables that are res

# First, we numerically normalize datasets to do Lasso-Regularized Logistic Regression
m_1 = model.matrix(Y~., df_1)
mtrain_1 = m_1[1:n,]
mtest_1 = m_1[-(1:n),]
```

```r
# Perform cross validation to find the best lambda
lasso_cv_1 = cv.glmnet(x= mtrain_1, y=train_1$Y, family="binomial")

# Store the minimum lambda as bestlam_1
bestlam_1 = lasso_cv_1$lambda.min

# Build a Lasso-Regularized Logistic Regression model
lasso_1 = glmnet(x = mtrain_1, y=train_1$Y, alpha = 1, family="binomial")

# Look at coefficients with the best lambda
lasso_1.coef = predict(lasso_1, type="coefficients", s= bestlam_1)
lasso_1.coef
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)     -2.306871223
## (Intercept)      .
## gender.L        -0.183423489
## age              0.039557973
## race.L          -0.222563335
## race.Q          -0.007087217
## race.C          -0.284803294
## race^4          -0.358488655
## birth_country.L -0.870085711
```

```r
## To see performance of the new model, we calculated AUC and draw ROC curves
# Calculate AUC and prepare for ROC curve for Logistic Regression
logit_prediction_1 = predict(lr_1,test_1)
logit_rocdata_1 = prediction(predictions=logit_prediction_1,
                             labels=test_1$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_lg_1 = prediction(predictions=logit_prediction_1,
                      labels=test_1$Y) %>% performance("auc")
auc_lg_1 = round(auc_lg_1@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for Random Forest
test_1$Y <- as.factor(test_1$Y)
rf_prediction_1 = predict(forest_1, test_1, type = "prob")
rf_rocdata_1 = prediction(predictions=rf_prediction_1[,2],
                          labels=test_1$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_rf_1 = prediction(predictions=rf_prediction_1[,2],
                      labels=test_1$Y) %>% performance("auc")
auc_rf_1 = round(auc_rf_1@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for Boosting
boosting_prediction_1 = predict(aforest_1, test_1, type = "prob")
boosting_rocdata_1 = prediction(predictions=boosting_prediction_1[,2],
                                labels=test_1$Y) %>%
  performance("tpr", "fpr") %>%
```

```r
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_boosting_1 = prediction(predictions=boosting_prediction_1[,2],
                            labels=test_1$Y) %>% performance("auc")
auc_boosting_1 = round(auc_boosting_1@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for Gradient Boosted Forest
gradient_prediction_1 = predict(gforest_1, test_1,
                                n.trees=gforest_1$n.trees,type = "response")
gradient_rocdata_1 = prediction(predictions=gradient_prediction_1,
                                labels=test_1$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_gradient_1 = prediction(predictions=gradient_prediction_1,
                            labels=test_1$Y) %>% performance("auc")
auc_gradient_1 = round(auc_gradient_1@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for Lasso-Regularized Logistic Regression
lasso_prediction_1 = predict(lasso_1, mtest_1, s= bestlam_1, type='response')
lasso_rocdata_1 = prediction(predictions=lasso_prediction_1,
                             labels=test_1$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_lasso_1 =  prediction(predictions=lasso_prediction_1,
                          labels=test_1$Y) %>% performance("auc")
auc_lasso_1 = round(auc_lasso_1@y.values[[1]],3)

# Plot ROC curve
ggplot(data = logit_rocdata_1, aes(x=FPR,y=TPR, col = paste0("Logistic Regression: AUC", auc_lg_1))) + g
  geom_line(data = rf_rocdata_1, aes(x=FPR,y=TPR, col = paste0("RF: AUC", auc_rf_1))) +
  geom_line(data = boosting_rocdata_1, aes(x=FPR,y=TPR, col = paste0("Boosting: AUC", auc_boosting_1)))
  geom_line(data = gradient_rocdata_1, aes(x=FPR,y=TPR, col = paste0("Gradient boosting: AUC", auc_gradi
  geom_line(data = lasso_rocdata_1, aes(x=FPR,y=TPR, col = paste0("Lasso: AUC", auc_lasso_1))) +
  theme(legend.title=element_blank())
```




```
## Since we could not see the improvement of performance with ensemble methods, we add features to impro
# Annual household income and nunmber of children younger than 5 years old are added.

# Create a new dataset
df_2 = df %>% select(gender, age, race, birth_country, num_children_5yrs_younger, annual_household_incom

# Set seed first to create training and test datasets
set.seed(518) # our graduation date!

# Shuffle sampes for randamization
shuffled_df_2 = df_2[sample(1:nrow(df_2)),]

# Assign 75% of the total samples into training set and 25% into test set.
# Traning set contains 1144 samples and test set has 382 samples.
n = round((0.75 * nrow(shuffled_df_2)),0)
train_2 = df_2[1:n,]
test_2 = df_2[-(1:n),]

# Confirm that train and test are balanced with the outcome of interest (Y)
train_2 %>% select(Y) %>% table() %>% prop.table()
```

```
## .
##         N         Y
## 0.7998252 0.2001748
```

```r
test_2 %>% select(Y) %>% table() %>% prop.table()
```

```
## .
##         N         Y
## 0.8167539 0.1832461
```

```r
# Run a base model, Logistic Regression, with new features
lr_2 = with(train_2, glm(Y=="Y" ~ .,
                         family = binomial("logit"),
                         data = train_2))
lr_2 %>% summary()
```

```
##
## Call:
## glm(formula = Y == "Y" ~ ., family = binomial("logit"), data = train_2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0173  -0.7221  -0.6017  -0.4195   2.4235
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -1.859e+00  4.091e-01  -4.544 5.51e-06 ***
## gender.L                    -2.292e-01  1.071e-01  -2.140  0.03235 *
## age                          2.628e-02  2.476e-02   1.062  0.28843
## race.L                      -2.227e-01  2.431e-01  -0.916  0.35944
## race.Q                      -1.422e-01  2.106e-01  -0.675  0.49942
## race.C                      -3.208e-01  1.920e-01  -1.671  0.09472 .
## race^4                      -3.026e-01  1.618e-01  -1.870  0.06147 .
## birth_country.L             -1.031e+00  3.765e-01  -2.740  0.00615 **
## num_children_5yrs_younger   -2.141e-01  1.173e-01  -1.826  0.06790 .
## annual_household_income_num -6.927e-06  2.570e-06  -2.696  0.00703 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1145.5  on 1143  degrees of freedom
## Residual deviance: 1105.3  on 1134  degrees of freedom
## AIC: 1125.3
##
## Number of Fisher Scoring iterations: 5
```

```r
# Then, we also run ensemble metehods with new features
# Run Random Forest
forest_2 = randomForest(formula = Y ~ .,
                        data=train_2, ntrees=10)
# Run Boosting
aforest_2 = ada(formula = Y=="Y" ~ .,
                data=train_2,
                iter=10)

# Run Gradient Boosted Forest
gforest_2 = gbm(formula = Y=="Y" ~ .,
                data=train_2 %>%
```

```r
                  mutate_if(is.logical, as.factor),
                interaction.depth=10,
                cv.folds = 2)
```

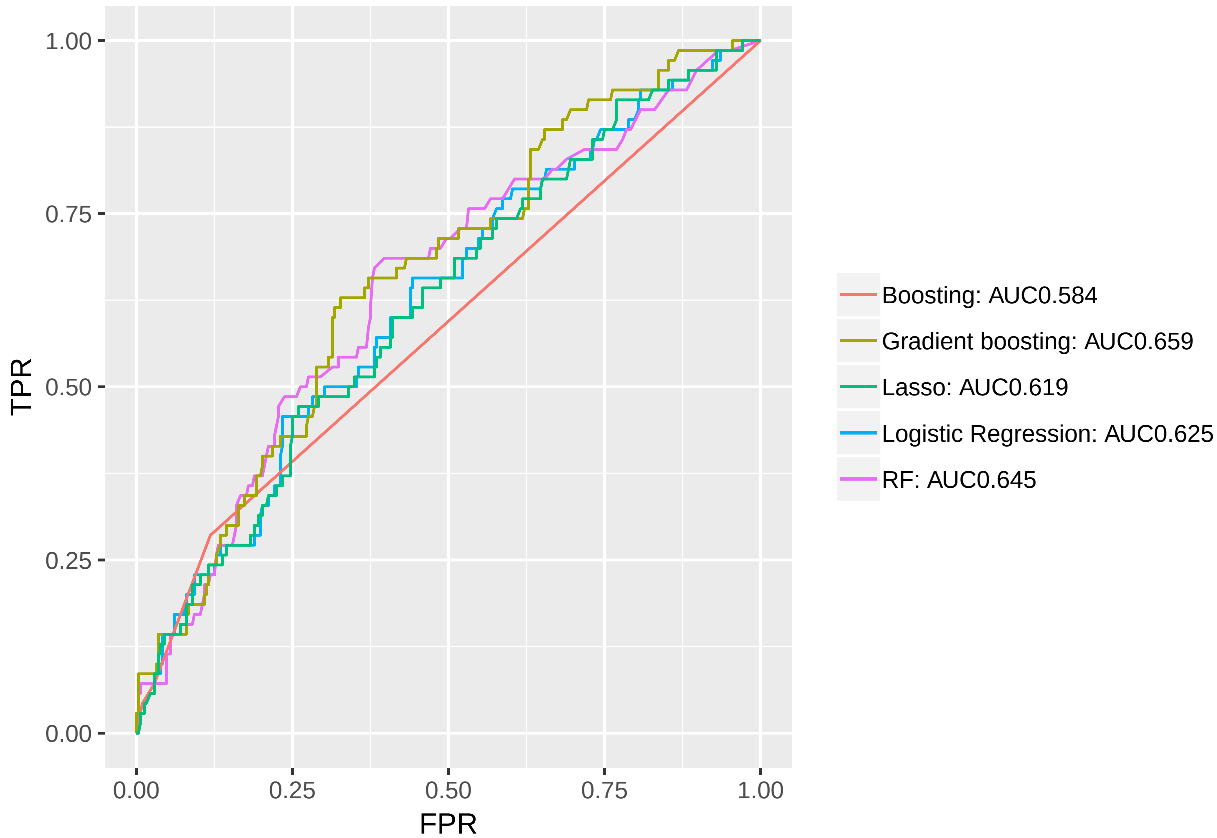
```
## Distribution not specified, assuming bernoulli ...
```

```r
# Since we added new features, we also want to see the performance of Lasso-Regularized Logistic Regres

# Numerically normalize datasets to do Lasso-Regularized Logistic Regression
m_2 = model.matrix(Y~., df_2)
mtrain_2 = m_2[1:n,]
mtest_2 = m_2[-(1:n),]

# Perform cross validation to find the best lambda
lasso_cv_2 = cv.glmnet(x= mtrain_2, y=train_2$Y, family="binomial")

# Store the minimum lambda as bestlalm_2
bestlam_2 = lasso_cv_2$lambda.min

# Build a Lasso-Regularized Logistic Regression model
lasso_2 = glmnet(x = mtrain_2, y=train_2$Y, alpha = 1, family="binomial")

# Look at coefficients with the best lambda
lasso_2.coef = predict(lasso_2, type="coefficients", s= bestlam_2)
lasso_2.coef
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                         1
## (Intercept)                 -1.662897e+00
## (Intercept)                  .
## gender.L                    -1.767818e-01
## age                          1.934213e-02
## race.L                      -4.576263e-02
## race.Q                       .
## race.C                      -2.244633e-01
## race^4                      -2.066425e-01
## birth_country.L             -7.860323e-01
## num_children_5yrs_younger   -1.695561e-01
## annual_household_income_num -6.074823e-06
```

```r
## To see performance of the new model, we calculated AUC and draw ROC curves
# Calculate AUC and prepare for ROC curve for Logistic Regression
logit_prediction_2 = predict(lr_2,test_2)
logit_rocdata_2 = prediction(predictions=logit_prediction_2,
                              labels=test_2$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_lg_2 =  prediction(predictions=logit_prediction_2,
                       labels=test_2$Y) %>% performance("auc")
auc_lg_2 = round(auc_lg_2@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for Random Forest
test_2$Y <- as.factor(test_2$Y)
rf_prediction_2 = predict(forest_2, test_2, type = "prob")
```

```r
rf_rocdata_2 = prediction(predictions=rf_prediction_2[,2],
                          labels=test_2$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_rf_2 = prediction(predictions=rf_prediction_2[,2],
                      labels=test_2$Y) %>% performance("auc")
auc_rf_2 = round(auc_rf_2@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for Boosting
boosting_prediction_2 = predict(aforest_2, test_2, type = "prob")
boosting_rocdata_2 = prediction(predictions=boosting_prediction_2[,2],
                                labels=test_2$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_boosting_2 = prediction(predictions=boosting_prediction_2[,2],
                            labels=test_2$Y) %>% performance("auc")
auc_boosting_2 = round(auc_boosting_2@y.values[[1]],3)

# Calculate AUC and prepare for ROC curve for gradient boosting
gradient_prediction_2 = predict(gforest_2, test_2,
                                n.trees=gforest_2$n.trees,type = "response")
gradient_rocdata_2 = prediction(predictions=gradient_prediction_2,
                                labels=test_2$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_gradient_2 = prediction(predictions=gradient_prediction_2,
                            labels=test_2$Y) %>% performance("auc")
auc_gradient_2 = round(auc_gradient_2@y.values[[1]],3)


# Calculate AUC and prepare for ROC curve for logistic regularized regression
lasso_prediction_2 = predict(lasso_2, mtest_2, s= bestlam_2, type='response')
lasso_rocdata_2 = prediction(predictions=lasso_prediction_2,
                             labels=test_2$Y) %>%
  performance("tpr", "fpr") %>%
  (function(.) data.frame(FPR=.@x.values[[1]], TPR=.@y.values[[1]]) %>% as_tibble())(.)

auc_lasso_2 =  prediction(predictions=lasso_prediction_2,
                          labels=test_2$Y) %>% performance("auc")
auc_lasso_2 = round(auc_lasso_2@y.values[[1]],3)


# Plot ROC curve
ggplot(data = logit_rocdata_2, aes(x=FPR,y=TPR, col = paste0("Logistic Regression: AUC", auc_lg_2))) + g
  geom_line(data = rf_rocdata_2, aes(x=FPR,y=TPR, col = paste0("RF: AUC", auc_rf_2))) +
  geom_line(data = boosting_rocdata_2, aes(x=FPR,y=TPR, col = paste0("Boosting: AUC", auc_boosting_2)))
  geom_line(data = gradient_rocdata_2, aes(x=FPR,y=TPR, col = paste0("Gradient boosting: AUC", auc_gradi
  geom_line(data = lasso_rocdata_2, aes(x=FPR,y=TPR, col = paste0("Lasso: AUC", auc_lasso_2))) +
  theme(legend.title=element_blank())
```




```
## Appendix C.
# Since we could not see considerable improvement of our model even when adding 2 more features, we exp

# First, create input variables data frame and output variable data frame to create dummy variables.
xtrain_2 = train_2 %>% select(-Y)
xtest_2 = test_2 %>% select(-Y)

# Also, modify the outcome (Y) into numeric
ytrain_2 = ordered(train_2$Y,
                   levels = c("N","Y"),
                   labels = c(0, 1)) %>% as.matrix()

ytest_2 = ordered(test_2$Y,
                  levels = c("N","Y"),
                  labels = c(0, 1)) %>% as.matrix()

# Then, create dummy variables for input variables.
dmy_train_2 = dummyVars(~., data = xtrain_2)
trsf_train_2 <- as.data.frame(predict(dmy_train_2,xtrain_2)) %>% as.matrix()

dmy_test_2 = dummyVars(~ ., data = xtest_2)
trsf_test_2 <- as.data.frame(predict(dmy_test_2,xtest_2)) %>% as.matrix()

# Report demension of the training set
dim(trsf_train_2)
```

```
## [1] 1144    9
```

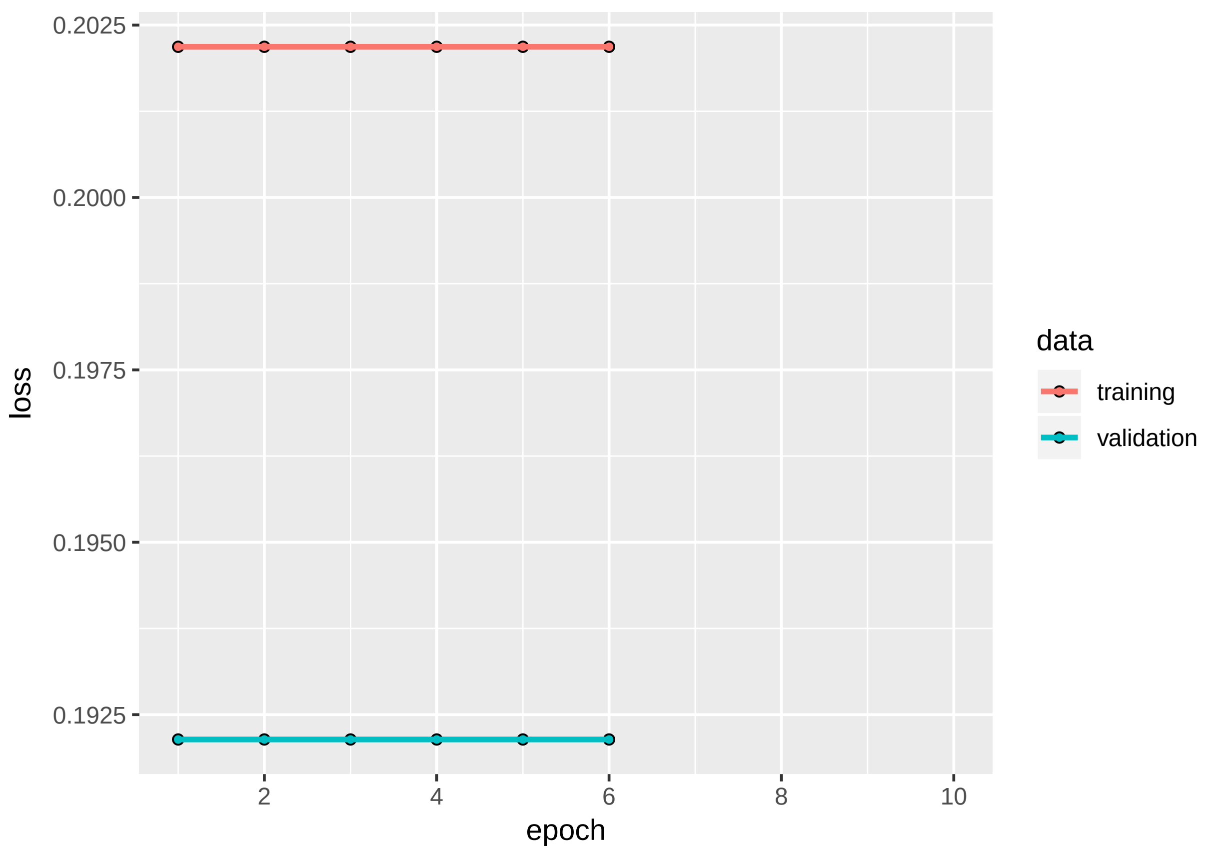
```r
# Specify a model architecture
model = keras_model_sequential()

# Add layers
# Based on the previous sections, adding regularization shows the similar performance as the one withou
# Therefore, we do not use regularization in our network.
# As an activation function for hidden layers, we use ReLU, which is the most common one.
# Since our purpose is classification, we use sigmoid output unit.
model %>%
  layer_dense(units = 32,
              activation = 'relu',
              input_shape = c(ncol(trsf_train_2))) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')

summary(model)
```

```
## ____________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ============================================================================
## dense_1 (Dense)                  (None, 32)                    320
## ____________________________________________________________________________
## dense_2 (Dense)                  (None, 32)                    1056
## ____________________________________________________________________________
## dense_3 (Dense)                  (None, 32)                    1056
## ____________________________________________________________________________
## dense_4 (Dense)                  (None, 1)                     33
## ============================================================================
## Total params: 2,465
## Trainable params: 2,465
## Non-trainable params: 0
## ____________________________________________________________________________
```

```r
# Specify loss and optimization method
# We use mean squared error to optimize the model
model %>% compile(
  loss = c('mse'),
  optimizer = optimizer_nadam(clipnorm = 10),
  metrics = c('mse')
)

# Train model with training dataset
inner_epochs = 10
early_stopping = callback_early_stopping(monitor = "val_loss",
                                         patience = inner_epochs/2)
bestLoss = 1e10
for(i in 1:20) {
  history = model %>% fit(trsf_train_2 , ytrain_2,
                          epochs = inner_epochs,
                          callbacks = c(early_stopping),
                          batch_size = 16,
                          validation_split = 0.2, shuffle=T)
```

```
  loss = history$metrics$val_loss[length(history$metrics$val_loss)]
  if(loss < bestLoss) {
    bestLoss = loss
    model %>% save_model_weights_hdf5("my_model_weights.h5")
  }
  if(length(history$metrics$val_loss) < inner_epochs)
    break
}

### Plot performance
plot(history, metrics = "loss")  # only plots the last part of training
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in sqrt(sum.squares/one.delta): NaNs produced

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in sqrt(sum.squares/one.delta): NaNs produced
```



```r
### Load the early-stopping model
bestModel = model %>% load_model_weights_hdf5('my_model_weights.h5')
bestModel %>% compile(
  loss = 'mse',
  optimizer = optimizer_nadam(),
  metrics = c('mse')
)

### Make predictions
bestModel %>% evaluate(trsf_test_2, ytest_2)
```

```
## $loss
## [1] 0.1832461
##
## $mean_squared_error
## [1] 0.1832461
```

```r
predition_2 = bestModel %>% predict_on_batch(trsf_test_2)

# Show predictions and true classification
rownum = 1:nrow(ytest_2)
ytest = cbind(ytest_2,rownum)
summarytable = cbind(ytest_2, predition_2[,1]) %>% as.data.frame()
```