CEU Data Science 2: Ensemble Trees, Variable Importance, & Model Stacking

Bruno Helmeczy

21/03/2021

This report summarizes my homework prepared for the Data Science 2: Machine Learning Tools class at the Central European University's MSc Business Analytics program. The 1st Chapter predicts a binary customer choice between orange juices, using a CART-, a Random Forest-, & a stochastic- & an extreme Gradient boosting model. The 2nd Chapter investigates Variable importance plots between 2 Random Forests & 2 Gradient Boosting Machines, in the 1st case comparing variables' importance with respect to the number of variables sampled at each node (2 vs 10), in the latter with respect to the sampling rate (10% vs 100%).

The 3rd chapter looks to predict whether patients are going to show up to their medical appointments. After setting a penalized linear model as benchmark, I fit 1-1 Random Forest, Stochastic Gradient Boosting, eXtreme Gradient Boosting, K-Nearest Neighbor & Support Vector Machine model. Finally, I use these models & the caretEnsemble package to fit a linear & tree-based stacked ensemble model, looking to improve predictive power.

```
library(tidyverse)
library(data.table)
library(caret)
library(gbm)
library(rpart.plot)
library(MLeval)
library(pROC)
library(caTools)
library(ggplot2)
library(viridis)
library(ggthemes)
library(dplyr)
library(kableExtra)
library(huxtable)
library(gridExtra)
library(caretEnsemble)

options(scipen=999)
theme_set(theme_tufte() + theme(
   plot.title = element_text(face = "bold.italic",size = 10)
  ,plot.title.position = "plot"
  ,axis.title = element_text(face = "bold", size = 9)))

# For continuity Models were uploaded to github as RDS objects & are loaded from there
GithubRDS_Folder <- "https://raw.githubusercontent.com/BrunoHelmeczy/Machine_Learning_Courses_DS1-3/mai
  # The raw script generates the models but with 3-4 hour total run time
    # All models saved as RDS & uploaded to github
      # In markdown, they're downloaded & loaded into the script
```

1) TREE ENSEMBLE MODELS

In this problem I'm using the OJ dataset from the ISLR package. This dataset records purchases of 2 orange juices and presents customer- and product characteristics. The goal is to predict which juice is chosen in a given purchase situation. See ?ISLR::OJ for a description of the variables.

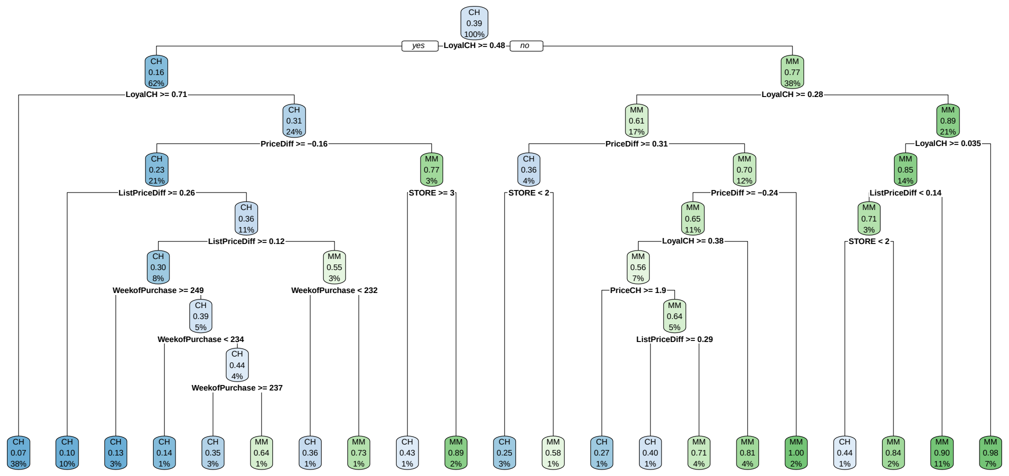
1 a) Create a training data of 75% and keep 25% of the data as a test set. I work with the caret package throughout the report, as such use createDataPartition() to randomly split my dataset into train & test sets. I also already set my trainControl object, using 5-fold cross-validation throughout this exercise, asking for 2-class summaries, predicting class probabilities, & only saving final predictions i.e. once the model has been parameter-tuned.

```r
# 1) TREE ENSEMBLE MODELS - 7points ----
df <- as_tibble(ISLR::OJ)
# Target var = Purchase

# 1 a) Create a training data of 75% and keep 25% of the data as a test set.
set.seed(1)
train_indices <- as.integer(createDataPartition(df$Purchase, p = 0.75, list = FALSE))
data_train <- df[train_indices, ]
data_holdout <- df[-train_indices, ]

# train control is 5 fold cross validation
train_control <- trainControl(method = "cv",
                              number = 5,
                              verboseIter = FALSE,
                              summaryFunction = twoClassSummary,
                              classProbs = T,
                              savePredictions = "final")
```

Train a decision tree as a benchmark model. I use the 'rpart' method, which requires minimum improvement in the chosen metric, in order to make a further split. Throughout all classification models I'm setting "ROC" as the metric to optimize for. Also, for continuity after training my models, I saved them all as RDS objects & stored it on github, downloading them for evaluation throughout the report. Having trained the CART model, I use rpart.plot() from the package of the same name to plot the decision-tree. Please refer to section **1 c)** for cross-validated 'Area under the Curve', Sensitivity & Specifity values. The AUC is 86.2%, i.e. there is an 86.2% probability that the model will successfully distinguish classes, in this case orange juice purchases.

```r
#### CART ####
set.seed(1234)
cart_model <- train(
  Purchase ~ .,
  data = data_train,
  method = "rpart",
  metric = "ROC",
  trControl = train_control,
  tuneGrid= expand.grid(cp = 0.0005))
saveRDS(cart_model,"cart_model.rds")
```

```r
download.file(paste0(GithubRDS_Folder,"Ex_1b/cart_model.rds"),
              "cart_model.rds", method = "curl")
cart_model <- readRDS("cart_model.rds")

Tree_Plot <- rpart.plot(cart_model$finalModel)
```



```r
print(Tree_Plot)
```

1.b) Investigate Tree Ensemble Models: RF, GBM, XGBoost Below I specify a Random Forest, a Stochastic Gradient Boosting-, & an eXtreme Gradient Boosting model, using caret's train() function. I still use "ROC" as the metric to optimize over, while seperately specifying a tuning grid for each model with expand.grid().

For Rand Forest, I compare sampling of 2 to 10 variables at each decision point, & minimum number of required observations at a node, in order for a split to even be considered. The resulting tuned hyperparameters were 6 variables to sample at each node, & minimum 40 observations remaining at a node for a split to be considered.

For Stochastic Gradiant Boosting, I keep growing the same number of trees (500), & test the allowed depths of a tree, the required number of minimum observations, & the various minimal learning / shrinkage rates to staz safe from overfitting. The resulting hyperparameters were an interaction / tree depth of 5, the highest tested learning rate, & minimum 5 required observation at a node for a split to be considered.

Finally, for the XGBoost model I also tested higher number of trees, as earlier research showed it to stabilize after higher number of trees vs other tree-based models. I further test various values for maximum tree depth, 'eta', variable sampling & minimum weights given to child-trees. The bestTuned model returned 500 trees with maximum tree depth of 5, an eta of 0.03, variable sampling of 75% & minimum 0.2 child-tree weights. Please refer to sections **1c)** & **1d)** for cross-validated AUC metrics

```r
#### RAND FOREST ####
rf_tune_grid <- expand.grid(
  .mtry = 2:10,
  .splitrule = "gini",
  .min.node.size = seq(5,45,by = 5) )
```

```r
set.seed(1234)
rf_model_1b <- train(
  Purchase ~ .,
  data = data_train,
  method = "ranger",
  metric = "ROC",
  trControl = train_control,
  tuneGrid= rf_tune_grid,
  importance = "impurity")
saveRDS(rf_model_1b,"rf_model_1b.rds")

#### GBM ####
gbm_grid <-  expand.grid(
  interaction.depth = c(1:5)*2-1, # complexity of the tree
  n.trees = 500, # number of iterations, i.e. trees
  shrinkage = c(1,5,10)*0.0001, # learning rate: how quickly the algorithm adapts
  n.minobsinnode = seq(5,25,by = 5)) # the minimum number of training set samples in a node to commence

set.seed(1234)
gbm_model_1b <- train(
  Purchase ~ .,
  data = data_train,
  method = "gbm",
  metric = "ROC",
  trControl = train_control,
  tuneGrid= gbm_grid)
saveRDS(gbm_model_1b,"gbm_model_1b.rds")

 #### XGBoost ####
 xgb_grid <-  expand.grid(
   nrounds=c(500,750),
   max_depth = (2:4)*2+1,
   eta = c(0.03,0.05, 0.06),
   gamma = c(0.01),
   colsample_bytree = seq(75,95,by =10 )/100,
   min_child_weight = (1:5)/10,
   subsample = c(0.75))

set.seed(1234)
 xgb_model_1b <- train(
   Purchase ~ .,
   data = data_train,
   method = "xgbTree", # xgbDART / xgbLinear / xgbTree
   metric = "ROC",
   trControl = train_control,
   tuneGrid= xgb_grid)
 saveRDS(xgb_model_1b,"xgb_model_1b.rds")
```

```r
download.file(paste0(GithubRDS_Folder,"Ex_1b/rf_model_1b.rds"),
              "rf_model_1b.rds", method = "curl")
rf_model_1b <- readRDS("rf_model_1b.rds")

download.file(paste0(GithubRDS_Folder,"Ex_1b/gbm_model_1b.rds"),
              "gbm_model_1b.rds", method = "curl")
gbm_model_1b <- readRDS("gbm_model_1b.rds")

download.file(paste0(GithubRDS_Folder,"Ex_1b/xgb_model_1b.rds"),
              "xgb_model_1b.rds", method = "curl")
xgb_model_1b <- readRDS("xgb_model_1b.rds")
```

1 c) Compare models' performance Please see 5-fold cross-validated Areas under the Curve, Sensitivity & Specificity values for the 4 models presented above. All 3 tree-ensemble models improved AUC vs the benchmark CART model. Random_Forest performed best, reaching highest AUC value with 89.4% i.e. the model has 89.4% probability to correct distinguish the possible outcomes of the Orange Juice purchase decision. Though GBM has more then 6% higher true positivity rate (Sensitivity) & is a close 2nd in terms of AUC, it is computationally much more intensive. Thus the Random Forest model is my selected best model.

```
 final_models <-
  list("CART" = cart_model,
       "Random_Forest" = rf_model_1b,
       "GBM" = gbm_model_1b,
       "XGBoost" = xgb_model_1b)

results <- resamples(final_models) %>% summary()
CV_SummTable <- sapply(c('ROC','Sens','Spec'), function(x) {
  tl <- list()
  tl <- results$statistics[[x]][,c("Mean")] %>% round(4)
  return(tl)
}) %>% as.data.frame()
colnames(CV_SummTable) <- c("AUC","Sensitivity","Specificity")

CV_SummTable %>% as_hux(add_rownames = T)
```

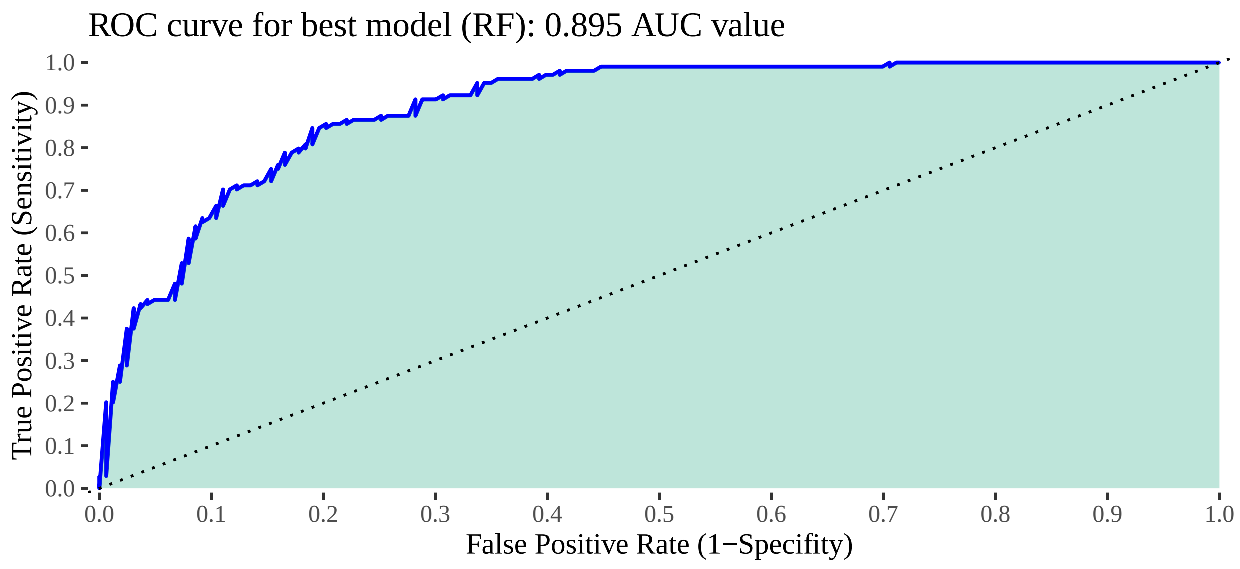
rownames	AUC	Sensitivity	Specificity
CART	0.862	0.847	0.703
Random_Forest	0.894	0.861	0.764
GBM	0.892	0.924	0.652
XGBoost	0.887	0.849	0.735

1 d) Choose best model & Plot ROC curve for best model based on test set. Below I define a function to re-calculate AUC values for the models above, & use for future models in the report. One can see the calculations' validity given the resamples-based approach & the triple-loop below qield the same result. I also re-use a function to visualize the best models' AUC from Gabor Bekes' & Gabor Kezdi's Data Analysis books. As the Random Forest model boasts highest AUC, it is my selected best model. Please see the Random Forest models' visualized ROC curve & AUC in the graph below. Note the AUC value associated with ex-sample prediction in the plot title. Thus, the random forest model kept a consistent AUC value also for out-of-sample prediction, i.e. we managed to avoid over-fitting the final model.

```
GetCV_AUCs <- function(named_model_list,control_value) {
  CV_AUC_folds <- list()

  for (model_name in names(named_model_list)) {
    auc <- list()
    model <- named_model_list[[model_name]]

    for (fold in c("Fold1", "Fold2", "Fold3", "Fold4", "Fold5")) {
      cv_fold <- model$pred %>% filter(Resample == fold)
      TuneParams <- model$bestTune %>% colnames()

      for (param in TuneParams) {
        cv_fold <- cv_fold[cv_fold[,param] == model$bestTune[,param],]
      }
      roc_obj <- roc(cv_fold$obs, cv_fold[,c(control_value)])
      auc[[fold]] <- as.numeric(roc_obj$auc)
    }
    CV_AUC_folds[[model_name]] <- data.frame("Resample" = names(auc),
                                             "AUC" = unlist(auc))
  }

  CV_AUC <- list()
  for (model_name in names(named_model_list)) {
    CV_AUC[[model_name]] <- mean(CV_AUC_folds[[model_name]]$AUC) %>% round(4)
  }
  AUCs <- CV_AUC %>% cbind() %>% as.data.frame()
  colnames(AUCs) <- "Model_CV_AUC"
  return(AUCs)
}
AUC_1b <- GetCV_AUCs(final_models,"CH")
AUC_1b %>% as_hux(add_rownames = T)
```

rownames	Model_CV_AUC
CART	0.8615
Random_Forest	0.8937
GBM	0.8925
XGBoost	0.8874

```
rf_pred <- predict(rf_model_1b, data_holdout, type="prob")
data_holdout[,"best_model_pred"] <- rf_pred[,"CH"]
roc_obj_holdout <- roc(data_holdout$Purchase, data_holdout$best_model_pred)

createRocPlot <- function(r, plot_name) {
  all_coords <- coords(r, x="all", ret="all", transpose = FALSE)

  roc_plot <- ggplot(data = all_coords, aes(x = fpr, y = tpr)) +
    geom_line(color='blue', size = 0.7) +
    geom_area(aes(fill = 'red', alpha=0.4), alpha = 0.3, position = 'identity', color = 'blue') +
    scale_fill_viridis(discrete = TRUE, begin=0.6, alpha=0.5, guide = FALSE) +
    xlab("False Positive Rate (1-Specifity)") +
    ylab("True Positive Rate (Sensitivity)") +
    geom_abline(intercept = 0, slope = 1,  linetype = "dotted", col = "black") +
    scale_y_continuous(limits = c(0, 1), breaks = seq(0, 1, .1), expand = c(0, 0.01)) +
    scale_x_continuous(limits = c(0, 1), breaks = seq(0, 1, .1), expand = c(0.01, 0)) +
    theme_tufte() +
    labs(x = "False Positive Rate (1-Specifity)",
         y = "True Positive Rate (Sensitivity)",
         title = paste0(plot_name,": ",round(r$auc,3)," AUC value"))

  roc_plot
}
FinModelROC <- createRocPlot(roc_obj_holdout, "ROC curve for best model (RF)")

grid.arrange(FinModelROC, ncol = 1)
```



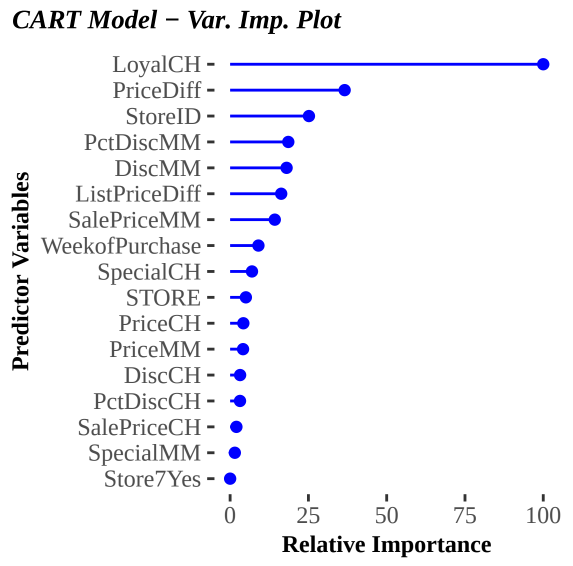
1 e) Inspect variable importance plots for the 3 models Similar variables found most important?

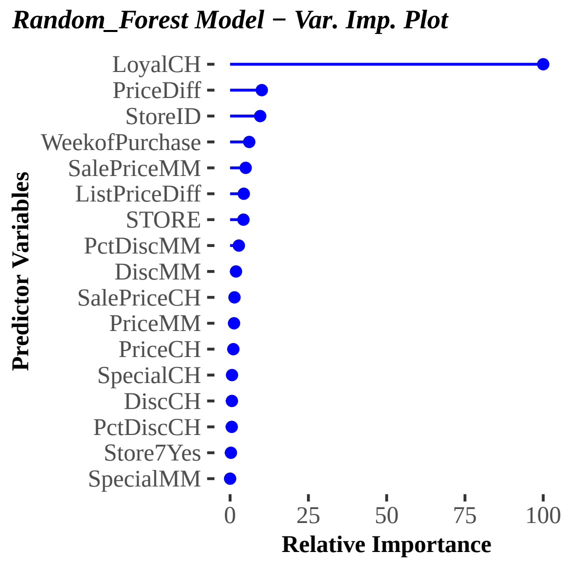
Please see the 4 models' variable importance plots below (using a function with a named list as input, allowing future use in the report). One can see the relative importance of variables per model differs significantly. E.g. the 5th most important variable for the CART model has ca. 25 relative importance score, while for the other 3 models these are close to zero. This is because the CART model splits nodes even at a very insignificant improvement & does not have a limit on the minimum number of observations at a node to be able to split. Thus all variables possible contributions are exhausted to the fullest, resulting in higher variable importance values. This does not seem to be the case for other models: only the 3rd most importance variable for GBM has a relative importance arbitrarily close to zero.

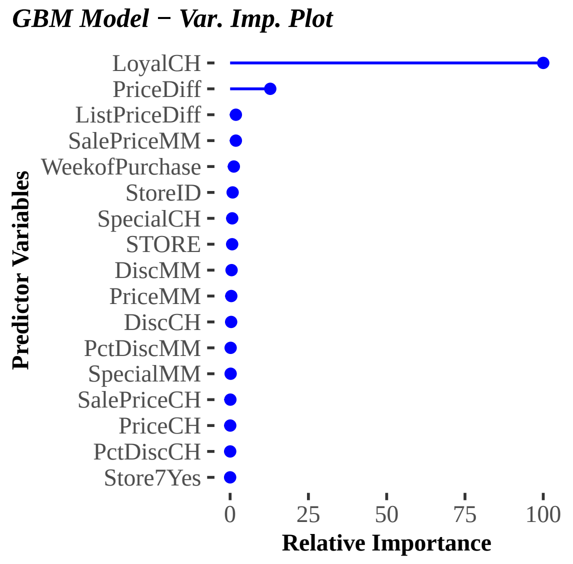
On another note observe the most important variables themselves. Whether the customer is loyal OJ consumer is by far the most important irrespective of the model, while relative price is either the 2nd or 3rd most important variable. By consensus, Store ID, i.e. location rounds out the top 3, being 3rd most important per the CART & RF model, 4th per XGBoost & 6th according to GBM.

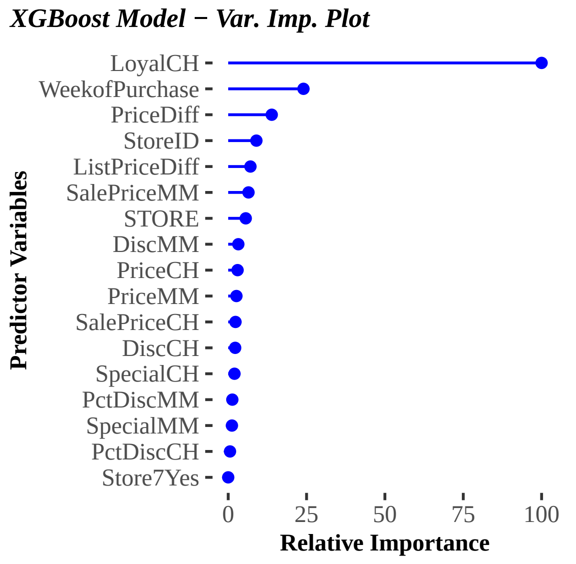
```
VarImpPlots <- function(named_model_list) {
  lapply(names(named_model_list), function(x) {
```

```r
    VarImp_Plots <- list()
    df <- varImp(named_model_list[[x]])$importance
    df$names <- df %>% rownames()

    VarImp_Plots[[x]] <- df %>%
      ggplot(aes(reorder(names,Overall), Overall) ) +
      geom_point(color = "blue") +
      geom_segment(aes(x=names,xend=names,y=0,yend=Overall), color = "blue") +
      coord_flip() +
      labs(x = "Predictor Variables", y = "Relative Importance",
           title = paste0(x, " Model - Var. Imp. Plot"))
    return(VarImp_Plots)
  })
}
VarPlot1e <- VarImpPlots(final_models)
print(VarPlot1e)
```









2. VARIABLE IMPORTANCE PROFILES.

```
df <- as_tibble(ISLR::Hitters) %>% drop_na(Salary) %>%
  mutate(log_salary = log(Salary), Salary = NULL)

# 2 a) Train 2 RFs - 1 sampling 2 vars randomly / split -> 1 sampling 10 ----
      # Use whole data & no cross-validation
      # Inspect variable importance profiles
    # What do you see -
      # how important the first few variables are relative to each other?
train_control <- trainControl(method = "none", savePredictions = T)
```

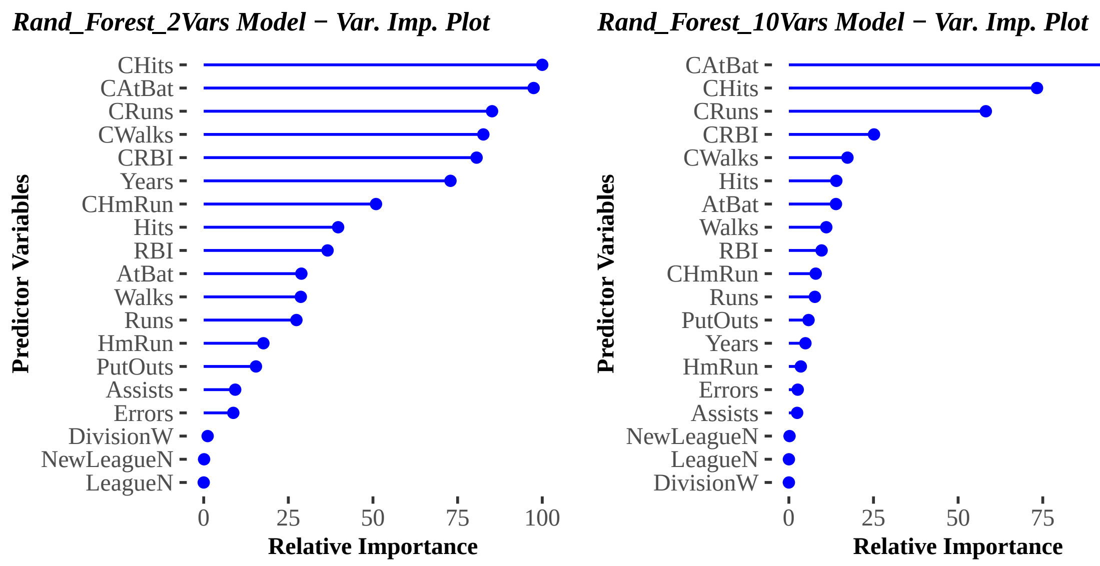
2. a) Train two random forest models: one with sampling 2 variables randomly for each split and one with 10 (use the whole dataset and don't do cross-validation). Inspect variable importance profiles. What do you see in terms of how important the first few variables are relative to each other?

Please see the train() specifications & variable importance plots for the 2 random forest models below. Note how gradual the decline is in relative importance for the model with 2 mtry-s vs the one with 10 which has a sharp, steep decline in relative importance.

```
# No Cross-Validation - No need for seeding

rf_model_2a_2 <- train(
  log_salary ~ .,
  data = df,
  method = "rf",
  trControl = train_control,
  tuneGrid = expand.grid(
    .mtry = 2)
  )
saveRDS(rf_model_2a_2,"rf_model_2a_2.rds")

rf_model_2a_10 <- train(
  log_salary ~ .,
  data = df,
  method = "rf",
  trControl = train_control,
  tuneGrid = expand.grid(
    .mtry = 10)
)
saveRDS(rf_model_2a_10,"rf_model_2a_10.rds")
```

```
download.file(paste0(GithubRDS_Folder,"Ex_2/rf_model_2a_2.rds"),
              "rf_model_2a_2.rds", method = "curl")
rf_model_2a_2 <- readRDS("rf_model_2a_2.rds")

download.file(paste0(GithubRDS_Folder,"Ex_2/rf_model_2a_10.rds"),
              "rf_model_2a_10.rds", method = "curl")
rf_model_2a_10 <- readRDS("rf_model_2a_10.rds")

RFs_2a <- list("Rand_Forest_2Vars" = rf_model_2a_2,
               "Rand_Forest_10Vars" = rf_model_2a_10)
```

```r
VarPlots_2a <- VarImpPlots(RFs_2a)
print(VarPlots_2a)
```

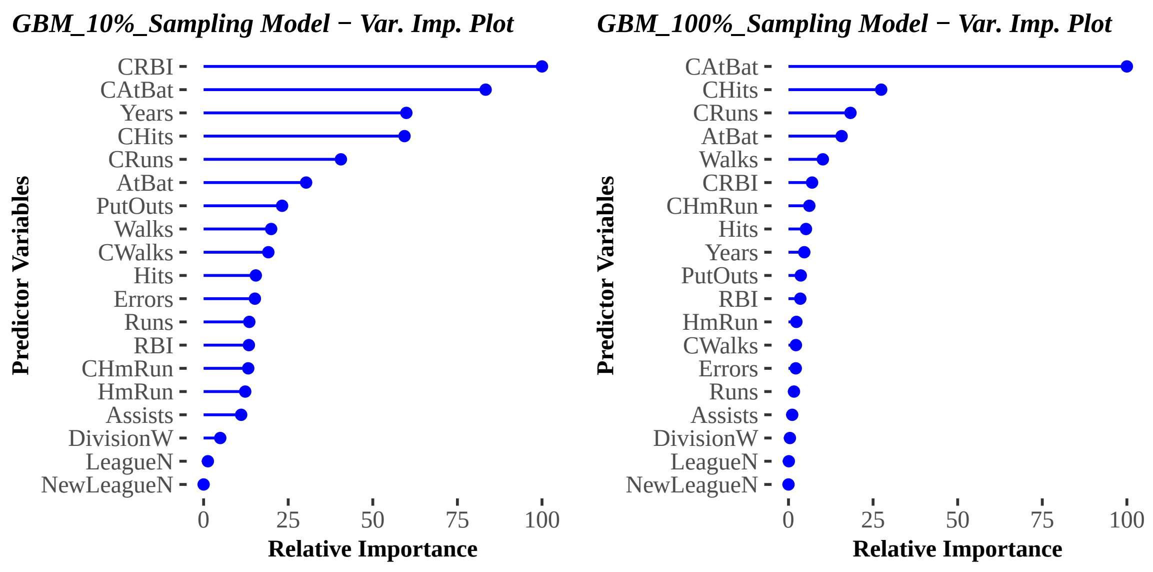


2. b) Either model is more extreme fr 2a) Give Intuition how mtry relates Now why is this? Well, mtry refers to how many predictors are randomly selected at each node, from which the most suitable is chosen to splite the given node. If 10 variables are to be chosen, any variable has larger chance to chosen (19 choose 10 vs 19 choose 2), & given the chosen set of variables, the best variable is going to determine the node split. Resultingly, with a smaller number of variables to be sampled at each node, the chance for weaker predictors to determine a split & thus gain some relative importance is larger.

2. c) Same as 2a), estimate 2 GBMs - vary sampling of trees Please see variable importance plots for the 2 Gradient Boosting models with different column sampling parameters. My personal intuition behind is the same as for Random forests. The column sample rate denotes the % of variables to segment at each node for splitting. If 1 randomly samples a smaller % of all variables, weaker predictor variables have larger chance to be chosen to split nodes by. Also, this time I use the 'gbm__h2o' method, requiring an h2o instance to run, hence I initialize it then stop it after running these models. The 'Y' input answers to whether I'm sure about shutting down the instance.

```r
# 2 c) Same as 2a), estimate 2 GBMs - vary sampling of trees ----
        # 0.1 & 1 for bag.fraction / sample_rate parameter
          # Hold all other params fixed
            # grow 500 trees -> max depth = 5 -> learning rate = 0.1
      # Compare variable importance plots for 2 models
        # Intuition on why 1 var importance profile more extreme then the other?
h2o.init()
gbm_h2o_2c_0.1 <- train(
  log_salary ~ .,
  data = df,
  method = "gbm_h2o",
  trControl = train_control,
  tuneGrid= expand.grid(
    ntrees = 500,
    max_depth = 5,
    min_rows = 5,
    learn_rate = 0.1,
    col_sample_rate = 0.1))
saveRDS(gbm_h2o_2c_0.1,"gbm_h2o_2c_0.1.rds")

gbm_h2o_2c_1 <- train(
  log_salary ~ .,
  data = df,
  method = "gbm_h2o",
  trControl = train_control,
  tuneGrid= expand.grid(
    ntrees = 500,
    max_depth = 5,
    min_rows = 5,
    learn_rate = 0.1,
    col_sample_rate = 1))
saveRDS(gbm_h2o_2c_1,"gbm_h2o_2c_1.rds")

h2o.shutdown()
Y
```

```r
download.file(paste0(GithubRDS_Folder,"Ex_2/gbm_h2o_2c_0.1.rds"),
              "gbm_h2o_2c_0.1.rds", method = "curl")
gbm_h2o_2c_0.1 <- readRDS("gbm_h2o_2c_0.1.rds")

download.file(paste0(GithubRDS_Folder,"Ex_2/gbm_h2o_2c_1.rds"),
              "gbm_h2o_2c_1.rds", method = "curl")
gbm_h2o_2c_1 <- readRDS("gbm_h2o_2c_1.rds")

GBMs_2c <- list("GBM_10%_Sampling" = gbm_h2o_2c_0.1,
               "GBM_100%_Sampling" = gbm_h2o_2c_1)

VarPlots_2c <- VarImpPlots(GBMs_2c)
print(VarPlots_2c)
```



3. STACKING

```
# 3) STACKING - 10 points ----data <- read_csv("KaggleV2-May-2016.csv") ----
myurl <- "https://raw.githubusercontent.com/BrunoHelmeczy/Machine_Learning_Courses_DS1-3/main/DS2/Kaggl
data <- read_csv(myurl)

# some data cleaning
data <- select(data,-one_of(c("PatientId", "AppointmentID", "Neighbourhood"))) %>%
  janitor::clean_names()

# for binary prediction, the target variable must be a factor + generate new variables
data <- mutate(data,
  no_show = factor(no_show, levels = c("Yes", "No")),
  handcap = ifelse(handcap > 0, 1, 0),
  across(c(gender, scholarship, hipertension, alcoholism, handcap), factor),
  hours_since_scheduled = as.numeric(appointment_day - scheduled_day))

# clean up a little bit
data <- filter(data, between(age, 0, 95), hours_since_scheduled >= 0) %>%
  select(-one_of(c("scheduled_day", "appointment_day", "sms_received")))
```

3. a) Train / Validation / Test Sets - 5% / 45 / 50

To predict whether patients are going to show up for their appointments, I again use createDataPartition() to split my data 1st to testing & modeling, then the 2nd time further splitting modeling data to training & testing data. Also, In preparation for making ensemble models I specify row indexes for cross-validation folds, using createFolds() & specifying 5 folds.

```
# 3 a) Train / Validation / Test Sets - 5% / 45 / 50 -----

# CreateDataPartition
set.seed(1)
test_indices <- as.integer(createDataPartition(data$no_show, p = 0.5, list = FALSE))
data_test <- data[test_indices, ]
data_model <- data[-test_indices, ]

set.seed(1)
train_indices <- as.integer(createDataPartition(data_model$no_show, p = 0.1,list = FALSE))
data_train <- data_model[train_indices,]
data_validate <- data_model[-train_indices,]
data_model <- NULL

# 5 fold cross validation - using index in trainControl for caretStack()
MyFolds <- createFolds(data_train$no_show,k = 5)

# 3c: Keep cross-validated predictions only
train_control <- trainControl(method = "cv",
                              number = 5,
                              index = MyFolds,
                              verboseIter = T,
                              summaryFunction = twoClassSummary,
                              classProbs = T,
                              savePredictions = "final")

Formula <- paste0("no_show ~ ", data %>%
                    select(-no_show) %>% colnames() %>% paste(collapse = " + "))
```

3. b) Train benchmark model of choice - RF / GBM / Glm

I chose an Elastic Net model as my benchmark, tuning both Alpha & Lambda values, after normalizing the data via centering & scaling. For evaluating any future models, I wrote a function to create prediction probabilities, given a named list of models & test data set. Passing the Benchmark GLM model & the validation data set returns my benchmark Ex-Sample AUC value, 57.6%, or 7.6% better then a coin toss. The best tuned model boasts alpha = 0, & lambda = 37.65, i.e. the best penalized model turns out to be a RIDGE Regression model.

```
# 3 b) Train benchmark model of choice - rf / gbm / glm ----

# My E-Net Grid & Model ----
ENet_tunegrid <- expand.grid(
  "alpha"  = seq(0, 1, by = 0.1),
  "lambda" = 10^seq(2,-5,length=100))

Benchmark_GLM <- train(
  formula(Formula),
  data = data_train,
  method = "glmnet",
  family = "binomial",
  preProcess = c("center","scale"),
  trControl = train_control,
  tuneGrid = ENet_tunegrid,
  na.action=na.exclude)
```

```
download.file(paste0(GithubRDS_Folder,"Ex_3c/Benchmark_GLM.rds"),
              "Benchmark_GLM.rds", method = "curl")
Benchmark_GLM <- readRDS("Benchmark_GLM.rds")

# Function 2 get Ex-Sample Model performances
Get_ExSample_AUCs <- function(named_model_list, test_df) {
  ExSample_AUCs <- list()

  for (model in names(named_model_list)) {
    Pred <- predict(named_model_list[[model]],newdata = test_df, type = "prob")
    test_df[,model] <- as.data.frame(Pred)[,1]
    test_df$pred <- as.data.frame(Pred)[,1]
    ExSample_AUCs[[model]] <- roc(test_df$no_show,test_df$pred)$auc %>% round(4)
  }

  findf <- ExSample_AUCs %>% cbind() %>% as.data.frame()
  colnames(findf) <- "Ex_Sample_AUC"
  test_df$pred <- NULL
  return(findf)
}
```

```
ExSamp_AUC_3b <- Get_ExSample_AUCs(list("Enet_Benchmark" = Benchmark_GLM), data_validate)
ExSamp_AUC_3b %>% as_hux(add_rownames = T)
```

rownames	Ex_Sample_AUC
Enet_Benchmark	0.5761

3. c) Build min 3 models of different model families with cross-validation

I below specified 5 models: A Random Forest, a stochastic Gradient Boosting Machine, an eXtreme Gradient Booster, a K-Nearest Neighbor & a Support Vector Machine with Radial basis functions. I used identical tuning grids to the models specified in chapter 1, & added K-Nearest Neighbors & SVM for my personal curiosity.

The Best-tuned Random Forest sample 2 variables for node-splits & requires 45 observations in a node to consider splitting it. The best-tuned GBM boasts interaction depth of 1, the highest available adaptation speed, & the minimum number of observations required per node. The best-tuned XGBoost model also boasts 500 trees, the smallest available tree depth, eta, colsample & highest child weight. The best KNN model considers the 41 closest observations, while the best-tuned SVM has sigma = 0.3 & the Cost function = 512.

```
#### 1) Random Forest ####
rf_tune_grid <- expand.grid(
  .mtry = 2:5,
  .splitrule = "gini",
  .min.node.size = seq(5,45,by = 5))

set.seed(1234)
RF_model_3c <- train(
  formula(Formula),
  data = data_train,
  method = "ranger",
  metric = "ROC",
  trControl = train_control,
  tuneGrid = rf_tune_grid,
  na.action=na.exclude)
saveRDS(RF_model_3c,"RF_model_3c.rds")

#### 2) GBM ####
gbm_grid <-  expand.grid(
  interaction.depth = c(1:5)*2-1, # complexity of the tree
  n.trees = 500, # number of iterations, i.e. trees
  shrinkage = c(1,5,10)*0.0001, # learning rate: how quickly the algorithm adapts
  n.minobsinnode = seq(5,25,by = 5)) # the minimum number of training set samples in a node to commence

set.seed(1234)
GBM_model_3c <- train(
  formula(Formula),
  data = data_train,
  method = "gbm",
  metric = "ROC",
```

```r
  trControl = train_control,
  tuneGrid= gbm_grid)
saveRDS(GBM_model_3c,"GBM_model_3c.rds")

#### 3) XGBoost ####
xgb_grid <-  expand.grid(
  nrounds=c(500,750),
  max_depth = (2:4)*2+1,
  eta = c(0.03,0.05, 0.06),
  gamma = c(0.01),
  colsample_bytree = seq(75,95,by =10 )/100,
  min_child_weight = (1:5)/10,
  subsample = c(0.75))

set.seed(1234)
XGB_model_3c <- train(
  formula(Formula),
  data = data_train,
  method = "xgbTree", # xgbDART / xgbLinear / xgbTree
  metric = "ROC",
  trControl = train_control,
  tuneGrid= xgb_grid)
saveRDS(XGB_model_3c,"XGB_model_3c.rds")

#### 4) KNN ####
set.seed(1234)
KNN_model_3c <- train(
  formula(Formula),
  data = data_train,
  method = "knn",
  metric = "ROC",
  preProcess = c("center","scale"),
  trControl = train_control,
  tuneGrid= data.frame(k = c((1:21)*2-1)))
saveRDS(KNN_model_3c,"KNN_model_3c.rds")

#### 5) Radial SVM ####
set.seed(1234)
SVM_Radial_3c <- train(
  formula(Formula),
  data = data_train,
  method = "svmRadial",
  metric = "ROC",
  trControl=train_control,
  tuneGrid = expand.grid(
    "sigma" = c(0:10)/10,
    "C" = 2^c(2:11)),
  tuneLength = 10)
saveRDS(SVM_Radial_3c,"SVM_Radial_3c.rds")
```

3. d) Evaluate Validation set performance of each model

Please see below both the Cross-Validated & Ex-Sample AUC values of each model. Interestingly, the benchmark model performs best during cross-validation with AUC of 58.2%, while the stochastic Gradient Boosting Machine reaches 59.4% AUC, with the Random Forest boasting the 2nd highest- & the benchmark model the 3rd highest ex-sample AUC value. Also note, the SVM model barely outperforms a simple coinflip, with both CV- & Ex-sample AUCs of 52.2%.

```r
download.file(paste0(GithubRDS_Folder,"Ex_3c/RF_model_3c.rds"),
              "RF_model_3c.rds", method = "curl")
RF_model_3c <- readRDS("RF_model_3c.rds")

download.file(paste0(GithubRDS_Folder,"Ex_3c/GBM_model_3c.rds"),
              "GBM_model_3c.rds", method = "curl")
GBM_model_3c <- readRDS("GBM_model_3c.rds")

download.file(paste0(GithubRDS_Folder,"Ex_3c/XGB_model_3c.rds"),
              "XGB_model_3c.rds", method = "curl")
XGB_model_3c <- readRDS("XGB_model_3c.rds")

download.file(paste0(GithubRDS_Folder,"Ex_3c/KNN_model_3c.rds"),
              "KNN_model_3c.rds", method = "curl")
KNN_model_3c <- readRDS("KNN_model_3c.rds")

download.file(paste0(GithubRDS_Folder,"Ex_3c/SVM_Radial_3c.rds"),
              "SVM_Radial_3c.rds", method = "curl")
SVM_Radial_3c <- readRDS("SVM_Radial_3c.rds")


# 3 d) Evaluate Validation set performance of each model ----
final_models_3c <- list(
  "Benchmark_Enet" = Benchmark_GLM,
  "Random_Forest"  = RF_model_3c,
  "Grad_Boost"     = GBM_model_3c,
  "XGBoost"        = XGB_model_3c,
  "KNN"            = KNN_model_3c,
  "SVM_Non_Lin"    = SVM_Radial_3c
)

AUCs_3d <- GetCV_AUCs(final_models_3c,"No")
ExSample_AUCs <- Get_ExSample_AUCs(final_models_3c, data_validate)
cbind(AUCs_3d,ExSample_AUCs) %>% as_hux(add_rownames = T)
```

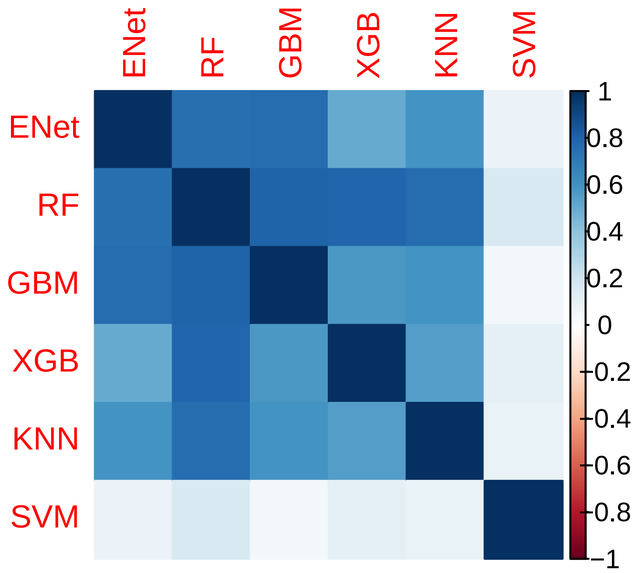
3. e) How large are correlations of base learner predicted scores on the validation set

Please see above models' correlogram heatmap & correlation matrix. As can be seen, the support vector machine distinctively differs in its predictions from the other models. On the other hand the Random Forest's absolute correlation value are between 0.76-0.81, while the KNN & E-Net models correlating between 0.5-0.6 with the XGBoost, KNN, & SVM models.

```r
# 3 e) How large are correlations of predicted scores on the validation set ----
    # produced by the base learners
```

rownames	Model_CV_AUC	Ex_Sample_AUC
Benchmark_Enet	0.5822	0.5761
Random_Forest	0.5686	0.5897
Grad_Boost	0.5777	0.594
XGBoost	0.5406	0.5715
KNN	0.5581	0.5661
SVM_Non_Lin	0.5218	0.5216

```r
Get_ExSample_Preds <- function(named_model_list, test_df) {
  ExSample_Preds <- list()

  for (model in names(named_model_list)) {
    Pred <- predict(named_model_list[[model]],newdata = test_df, type = "prob")
    ExSample_Preds[[model]] <- as.data.frame(Pred)[,1]
  }
  ExSample_Preds <- ExSample_Preds %>% as.data.frame()

  return(ExSample_Preds)
}

BaseLearnPreds<- Get_ExSample_Preds(final_models_3c,data_validate)
colnames(BaseLearnPreds) <- c("ENet","RF","GBM","XGB","KNN","SVM")
CorrPlot <- BaseLearnPreds %>% cor() %>% abs() %>% corrplot::corrplot(method = 'color')
```



```r
CorrPlot %>% round(3) %>% as_hux(add_rownames = T, add_colnames = T)
```

rownames	ENet	RF	GBM	XGB	KNN	SVM
ENet	1	0.759	0.761	0.508	0.596	0.089
RF	0.759	1	0.808	0.791	0.769	0.161
GBM	0.761	0.808	1	0.572	0.592	0.056
XGB	0.508	0.791	0.572	1	0.559	0.114
KNN	0.596	0.769	0.592	0.559	1	0.094
SVM	0.089	0.161	0.056	0.114	0.094	1

3 f) Create Stacked ensemble model from the base learners.

To create Stacked ensemble models, I use the caretEnsemble package, which allows calling a train() function on a list of caret train objects. To create stacked models, I decided to exclude the SVM model, given it is barely outperforming a random guess. Out of curiosity I trained 3 stacked models below: An Elastic Net, A Random Forest, & a Gradient Boosting Machine. The best-tuned ensemble elastic net boasts an alpha of 0.06 & lambda = 0.1, the best ensemble random forest sample 2 base learner models & requires at least 45 observation in a split-node & the GBM ensemble uses an interaction depth of 9 with atleast 10 observations per split-node & shrinkage of 0.0005.

```
# 3 f) Create Stacked ensemble model from the base learners. ----
library(caretEnsemble)

# E-Net ----
glmEnsemble <- caretStack(
  as.caretList(final_models_3c[1:5]),
  method = "glmnet",
  metric = "ROC",
  family = 'binomial',
  trControl = trainControl(
    method = "cv",
    number = 5,
    savePredictions = "final",
    classProbs = T,
    verboseIter = T,
    summaryFunction = twoClassSummary),
  tuneGrid = expand.grid(
    'alpha' = 10^seq(2,-5,length = 100),
    'lambda' = c(0:10)/10))
saveRDS(glmEnsemble,"glmEnsemble.rds")

# RF -----
RF_Ensemble <- caretStack(
  as.caretList(final_models_3c[1:5]),
  method = "ranger",
  metric = "ROC",
  tuneGrid = expand.grid(
    .mtry = 2:5,
    .splitrule = "gini",
    .min.node.size = seq(5,45,by = 5)),
```

```r
  trControl = trainControl(
    method = "cv",
    number = 5,
    savePredictions = "final",
    classProbs = T,
    verboseIter = T,
    summaryFunction = twoClassSummary))
saveRDS(RF_Ensemble,"RF_Ensemble.rds")

# GBM ----
GBM_Ensemble <- caretStack(
  as.caretList(final_models_3c[1:5]),
  method = "gbm",
  metric = "ROC",
  tuneGrid = gbm_grid,
  trControl = trainControl(
    method = "cv",
    number = 5,
    savePredictions = "final",
    classProbs = T,
    verboseIter = T,
    summaryFunction = twoClassSummary))
saveRDS(GBM_Ensemble,"GBM_Ensemble.rds")
```

```r
download.file(paste0(GithubRDS_Folder,"Ensambles/glmEnsemble.rds"),
              "glmEnsemble.rds", method = "curl")
glmEnsemble <- readRDS("glmEnsemble.rds")

download.file(paste0(GithubRDS_Folder,"Ensambles/RF_Ensemble.rds"),
              "RF_Ensemble.rds", method = "curl")
RF_Ensemble <- readRDS("RF_Ensemble.rds")

download.file(paste0(GithubRDS_Folder,"Ensambles/GBM_Ensemble.rds"),
              "GBM_Ensemble.rds", method = "curl")
GBM_Ensemble <- readRDS("GBM_Ensemble.rds")
```

3 g) Evaluate ensembles on validation set - Did prediction improve ?

Please see below all 9 models of this chapter - 6 base learners, & 3 ensembles. Based on validation set performance, neither ensemble model beat the best-performing base learner in terms of AUC, the Gradient Boosting Machine. However, both the Elastic Net & Gradient Booster stacked Ensemble model came pretty close. Thus the 3 best models all boast ex-sample AUC values between 59.1% - 59.4%. Due to the Gradient Booster base learner having highest AUC value, while also being significantly less computationally taxing, it is my final chosen model to test on the test set. Out of sheer curiosity however, I also included the Elastic Net & GBM stacked ensembles as well in the final exercise.

```r
# 3 g) Evaluate ensembles on validation set - Did prediction improve ? ----
final_models_3f <- list(
  "Benchmark_Enet" = Benchmark_GLM,
  "Random_Forest"  = RF_model_3c,
  "Grad_Boost"     = GBM_model_3c,
  "XGBoost"        = XGB_model_3c,
  "KNN"            = KNN_model_3c,
  "SVM_Non_Lin"    = SVM_Radial_3c,
  'Lin_Ensemble' = glmEnsemble$ens_model,
  'RF_Ensemble' = RF_Ensemble$ens_model,
  'GBM_Ensemble' = GBM_Ensemble$ens_model)

AUCs_3f <- GetCV_AUCs(final_models_3f,"No")
ExSample_AUCs_3f <- Get_ExSample_AUCs(final_models_3f, data_validate)
FinSumm_3f <- cbind(AUCs_3f,ExSample_AUCs_3f) %>% as.data.frame()

FinSumm_3f %>% as_hux(add_rownames = T)
```

rownames	Model__CV__AUC	Ex__Sample__AUC
Benchmark__Enet	0.5822	0.5761
Random__Forest	0.5686	0.5897
Grad__Boost	0.5777	0.594
XGBoost	0.5406	0.5715
KNN	0.5581	0.5661
SVM__Non__Lin	0.5218	0.5216
Lin__Ensemble	0.5786	0.5927
RF__Ensemble	0.554	0.5398
GBM__Ensemble	0.5806	0.5907

3 h) Evaluate best performing model on test set.

Please see aforementioned models' Cross-Validated, Ex-Sample, & Test set AUC values. The base-learner Gradient booster indeed is the best model, outperforming both stacked models. In all 3 cases one can also observe a slight decrease in AUC values, roughly by 0.6-0.8 %. Overall, one can expect the Gradient Booster base learned to correctly distinguish classes in 58.66% of cases, classes in this context being whether patients show up to their appointments, or not.

```
# 3 h) Evaluate best performing model on test set.
    # How does performance compare to validation set?

FinSumm_3h <- Get_ExSample_AUCs(final_models_3f, data_test)
FinSumm_3h <- cbind(FinSumm_3f,FinSumm_3h)
colnames(FinSumm_3h) <- c("CV_Model_AUC","Validation_Set_AUC","Testing_AUC")
FinSumm_3h[c(3,7,9),] %>% as_hux(add_rownames = T)
```

rownames	CV_Model_AUC	Validation_Set_AUC	Testing_AUC
Grad_Boost	0.5777	0.594	0.5866
Lin_Ensemble	0.5786	0.5927	0.5855
GBM_Ensemble	0.5806	0.5907	0.5847