Final Project

Shovan Biswas

2020/12/04

Libraries

```
library(tidyverse)
library(caret)
library(ROCR)
library(e1071)
library(pROC)
library(class)
library(knitr)
library(randomForest)
```

PART A

STEP#0: Pick any two classifiers of (SVM,Logistic,DecisionTree,NaiveBayes). Pick heart or ecoli dataset. Heart is simpler and ecoli compounds the problem as it is NOT a balanced dataset. From a grading perspective both carry the same weight.

STEP#1 For each classifier, Set a seed (43)

STEP#2 Do a 80/20 split and determine the Accuracy, AUC and as many metrics as returned by the Caret package (confusionMatrix) Call this the base_metric. Note down as best as you can development (engineering) cost as well as computing cost(elapsed time).

Start with the original dataset and set a seed (43). Then run a cross validation of 5 and 10 of the model on the training set. Determine the same set of metrics and compare the cv_metrics with the base_metric. Note down as best as you can development (engineering) cost as well as computing cost(elapsed time).

Start with the original dataset and set a seed (43) Then run a bootstrap of 200 resamples and compute the same set of metrics and for each of the two classifiers build a three column table for each experiment (base, bootstrap, cross-validated). Note down as best as you can development (engineering) cost as well as computing cost(elapsed time).

Answer:

Before writing anything, I'll write all the necessary functions here. Since I have to repeat the same or similar logic in many of the models, I coded common functions at the top.

I tailored the logic of the RMD programs, given in class to suite the current requirements.

Functions block

```r
# Split dataset

split_dataset <- function(seed_value, dataset, prct) {

  set.seed(seed_value)

  split_index <- sample(1:nrow(dataset), prct * nrow(dataset), replace = F)
  train <- dataset[-split_index,]
  test <- dataset[split_index,]

  return(list(train, test))
}
```

```r
model_build <- function(model_name, train, type) {

  if (model_name == 'glm') {
    model <- glm(target~., data = train, family = 'binomial')
  } else if (model_name == 'nb') {
    model <- naiveBayes(target~., data = train)
  } else if (model_name == 'rf') {
    model <- randomForest(target~., data = train, ntree = type)
  }

  return(model)
}
```

```r
model_predict <- function(model_name, model, test, type) {

  if (model_name == 'glm') {

    model_prediction <- predict(model, newdata = test[,-c(14)], type = type)
    model_prediction_class <- ifelse(model_prediction < 0.5, 0, 1)     # Crux of Logistic Regression ac
    model_caret_results <- caret::confusionMatrix(table(test[[14]], model_prediction_class))

  } else if (model_name == 'nb') {

    model_prediction <- predict(model, newdata = test[,-c(14)], type = type)
    model_prediction_class <- unlist(apply(round(model_prediction), 1, which.max)) - 1
    model_caret_results <- caret::confusionMatrix(table(test[[14]], model_prediction_class))

  } else if (model_name == 'rf') {

    model_prediction <- predict(model, newdata = test[,-c(14)])
    model_prediction_class <- table(test$target, model_prediction)
    model_caret_results <- caret::confusionMatrix(model_prediction_class)

  }

  return(list(model_caret_results, model_prediction))
}
```

```
cv_folds_create <- function(train, NF) {

  # Splitting numbers from 1 to N (N is number of rows in file) into folds of size NF (5, 10 etc). At t

  N <- nrow(train)
  NF = NF
  folds <- split(1:N, cut(1:N, quantile(1:N, probs = seq(0, 1, by = 1/NF))))  # Generates NF folds or b

  return(folds)
}
```

```
create_sample <- function(data_set, tf_value) {

  # The function **Sample(1:n, s, replace = F)** randomly selects s numbers from the vector 1:n. Follow
  # from the vector 1:nrow(train). Note that the numbers are the same. So, it effectively randomizes th
  # a new vector ridx. So, far we have not touched the actual data in train.

  ridx <- sample(1:nrow(data_set), nrow(data_set), replace = tf_value)   # Randomize the data

  return(ridx)
}
```

```
cv_model_build <- function(model_name, folds, train, type, ridx) {

  cv_df <- do.call('rbind', lapply(folds, FUN = function(idx, data = train[ridx,]) {

    if (model_name == 'glm') {
      m <- glm(target~., data = data[-idx,], family = 'binomial')
      p <- predict(m, data[idx, -c(14)], type = type)
      pc <- ifelse(p < 0.5, 0, 1)
    } else if (model_name == 'nb') {
      m <- naiveBayes(target~., data = data[-idx,])
      p <- predict(m, data[idx, -c(14)], type = type)
      pc <- unlist(apply(round(p), 1, which.max)) - 1
    }

    pred_tbl <- table(data[idx, c(14)], pc)
    pred_cfm <- caret::confusionMatrix(pred_tbl)
    list(fold = idx, m = m, cfm = pred_cfm)
  }
  ))

  return(cv_df)
}
```

```
cv_model_predict <- function(model_name, cv_df, test, type) {

  tstcv_preds <- lapply(cv_df, FUN = function(M, D = test[,-c(14)]) predict(M, D, type = type))
  tstcv_cfm <- lapply(tstcv_preds, FUN = function(P, A = test[[14]])  {

    if (model_name == 'glm') {
      pred_class <- ifelse(P < 0.5, 0, 1)
    } else if (model_name == 'nb') {
```

```r
      pred_class <- unlist(apply(round(P), 1, which.max)) - 1
    }

    pred_tbl <- table(pred_class, A)
    pred_cfm <- caret::confusionMatrix(pred_tbl)
  }
  )

  return(list(tstcv_cfm, tstcv_preds))
}
```

```r
cv_compute_param_average <- function(tstcv_cfm) {

  tstcv_perf <- as.data.frame(do.call('rbind', lapply(tstcv_cfm, FUN = function(cfm) c(cfm$overall, cfm$
  cv_tst_perf <- apply(tstcv_perf[tstcv_perf$AccuracyPValue < 0.01, -c(6:7)], 2, mean)   # Compute Aver
  cv_tst_perf_df <- data.frame(cv_tst_perf)

  # cv_tst_perf_var <- apply(tstcv_perf[tstcv_perf$AccuracyPValue < 0.01, -c(6:7)], 2, sd)

  return(cv_tst_perf_df)
}
```

```r
cv_compute_confusionmatrix_average <- function(tstcv_cfm) {

  tstcv_perf <- as.data.frame(do.call('rbind', lapply(tstcv_cfm, FUN = function(cfm) c(cfm$overall, cfm$
  cv_tst_perf <- apply(tstcv_perf[tstcv_perf$AccuracyPValue < 0.01, -c(6:7)], 2, mean)
  cv_confusion_matrix <- matrix(c(cv_tst_perf[6], cv_tst_perf[7], cv_tst_perf[8], cv_tst_perf[9]), nrow

  return(cv_confusion_matrix)
}
```

```r
cv_compute_AUC_average <- function(model_name, tstcv_preds, NF) {

  if (model_name == 'glm') {
    tstcv_preds_df <- data.frame(tstcv_preds)
    sum <- rep(0, nrow(tstcv_preds_df))   # There are 60 items in each prediction
    for(i in 1:NF) {
      sum <- sum + tstcv_preds_df[i]
    }

  } else if (model_name == 'nb') {
    sum <- rep(0, nrow(data.frame(tstcv_preds)))   # There are 60 items in each prediction
    for(i in 1:NF) {
      sum <- sum + tstcv_preds[[i]][,2]
    }
  }

  cv_prediction <- sum / NF
  cv_auc <- performance(prediction(cv_prediction, data_test_redo$target), 'auc')@y.values[[1]]

  return(cv_auc)
}
```

Terminology

Accuracy or **Balanced Accuracy** = $\frac{TP+TN}{(TP+FP+FN+TN)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)
Sensitivity or **Recall** or $TPR = $\frac{TP}{(TP+FN)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)
Specificity or $TNR = \frac{TN}{(TN+FP)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)
Pos Pred Value or **Precision** or $PPV = \frac{TP}{(TP+FP)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)
Neg Pred Value or $NPV = \frac{TN}{(TN+FN)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)
Prevalence = $\frac{TP+FN}{(TP+FP+FN+TN)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)
Detection Rate = $\frac{TP}{(TP+FP+FN+TN)}$ (https://stats.stackexchange.com/questions/316641/what-is-the-usefulness-of-detection-rate-in-a-confusion-matrix)
Detection Prevalence = $\frac{TP+FP}{(TP+FP+FN+TN)}$ (https://yardstick.tidymodels.org/reference/detection_prevalence.html)
F1 = $\frac{2TP}{(2TP+FP+FN)}$ (https://en.wikipedia.org/wiki/Confusion_matrix)

Processing

From the four given models, I am choosing **Logistic Regression** and **Naive Bayes**.

Loading Data

Since heart.csv was more often used and referred, I am using heart.csv. Furthermore, let me state that I'll mostly reuse the R code given in class.

```r
heart <- read.csv("./heart.csv", header = T, sep = ",", stringsAsFactors = F)
names(heart)
```

```
##  [1] "ï..age"   "sex"      "cp"       "trestbps" "chol"     "fbs"
##  [7] "restecg"  "thalach"  "exang"    "oldpeak"  "slope"    "ca"
## [13] "thal"     "target"
```

As we see, the name of column age is displayed as **i..age**. I verified with UNIX command **cat -tve heart.csv** that this is due to the presence of control characters **'/M-oM-;M-?'** at the begining of heart.csv.

I will remove control characters with UNIX command in the below code chunk. Note {bash} as opposed to {r} in below code chunk, which enables me to execute Unix commands from Cygwin installed on my computer. I set up my Windows and R to execute Unix shell scripts from RMD.

In Professor Raman's RMD it was handled a bit differently, by changing the column name age, with **names(heart)**[[**1**]] **<- 'age'**, but I experimented a different approach here, because I don't like to keep control characters in a dataset.

```bash
cat -tve heart.csv | sed 's/M-oM-;M-?//g' | sed 's/...$//' > heart2.csv
```

The real processing of basic Logistic Regression and Naive Bayes begins here. So, I'll start counting time from this point. But, some tasks are common, so I'll compute it separately.

```r
ptm <- proc.time()  # timing the common parts, start
```

```r
heart <- read.csv("./heart2.csv", header = T, sep = ",", stringsAsFactors = F)
names(heart)
```

```
##  [1] "age"      "sex"      "cp"       "trestbps" "chol"     "fbs"
##  [7] "restecg"  "thalach"  "exang"    "oldpeak"  "slope"    "ca"
## [13] "thal"     "target"
```

```r
names(heart)[[1]] <- 'age'
heart$target <- as.factor(heart$target)
```

```r
dim(heart)
```

```
## [1] 303  14
```

Checking for constants in all rows of each column.

```r
isConstant <- function(x) length(names(table(x))) < 2
apply(heart, 2, isConstant)
```

```
##      age      sex       cp trestbps     chol      fbs  restecg  thalach
##    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
##    exang  oldpeak    slope       ca     thal   target
##    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
```

```r
classLabels <- table(heart$target)
print(classLabels)
```

```
##
##   0   1
## 138 165
```

```r
ifelse(length(names(classLabels)) == 2, "binary classification", "multi-class classification")
```

```
## [1] "binary classification"
```

So, we see that there are two values, 0 and 1 in heart$target. So, it's a case binary classification.

Splitting data

Now, we'll split the data into train and test.

```r
split_data <- split_dataset(43, heart, 0.20)

data_train <- as.data.frame(split_data[1])
data_test <- as.data.frame(split_data[2])
```

```r
common_execution_tm <- proc.time() - ptm  # timing the common parts, end
```

At this point, the common part is over.

Running Logistic Regression Model

```r
ptm <- proc.time()  # timing the Logistic Regression parts, start

model_name <- 'glm'
type_name <- 'response'
```

```r
model <- model_build(model_name, data_train, type_name)

model_prediction <- model_predict(model_name, model, data_test, type_name)
model_confusionmatrix <- model_prediction[1][[1]][2]
model_accuracy <- data.frame(model_prediction[1][[1]][3])[1, 1]

model_caret_results_df <- as.data.frame(model_prediction[1][[1]][4])

model_auc <- performance(prediction(model_prediction[2][1], data_test$target), 'auc')@y.values[[1]]
```

Summary of Logistic Regression Model.

```r
summary(model)
```

```
##
## Call:
## glm(formula = target ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5131  -0.3438   0.1307   0.5932   2.7301
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.261356   3.079157   0.410 0.682068
## age          0.025608   0.027562   0.929 0.352832
## sex         -1.968137   0.566090  -3.477 0.000508 ***
## cp           0.885909   0.218406   4.056 4.99e-05 ***
## trestbps    -0.019602   0.012838  -1.527 0.126799
## chol        -0.007745   0.004236  -1.828 0.067483 .
## fbs         -0.194758   0.585447  -0.333 0.739386
## restecg      0.421134   0.401325   1.049 0.294013
## thalach      0.034922   0.012408   2.814 0.004887 **
## exang       -0.838499   0.482003  -1.740 0.081927 .
## oldpeak     -0.734070   0.253577  -2.895 0.003793 **
## slope        0.486916   0.391286   1.244 0.213353
## ca          -0.890561   0.221277  -4.025 5.71e-05 ***
## thal        -0.929230   0.331178  -2.806 0.005019 **
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 335.05  on 242  degrees of freedom
## Residual deviance: 164.15  on 229  degrees of freedom
## AIC: 192.15
##
## Number of Fisher Scoring iterations: 6
```

Confusion Matrix.

```
model_confusionmatrix
```

```
## $table
##    model_prediction_class
##      0  1
##   0 22  5
##   1  6 27
```

Accuracy.

```
model_accuracy
```

```
## [1] 0.8166667
```

AUC.

```
model_auc
```

```
## [1] 0.9068462
```

Below figure displays the terms **Balanced Accuracy**, **Precision**, **Recall**, which are the same as **Accuracy**, **Pos Pred Value**, **Sensitivity** respectively. So, in order not to count twice, I'll exclude the terms **Precision**, **Recall**, **Balanced Accuracy** from the basic metric. There is an additional term F1, which I'll include in the basic metric. I explained the terms in Terminology section above.

```
model_caret_results_df
```

```
##                        byClass
## Sensitivity          0.7857143
## Specificity          0.8437500
## Pos Pred Value       0.8148148
## Neg Pred Value       0.8181818
## Precision            0.8148148
## Recall               0.7857143
## F1                   0.8000000
## Prevalence           0.4666667
## Detection Rate       0.3666667
## Detection Prevalence 0.4500000
## Balanced Accuracy    0.8147321
```

Execution time.

```
lr_execution_tm <- proc.time() - ptm  # timing the Logistic Regression parts, end

tot_tm <- lr_execution_tm + common_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.12    0.00    0.12
```

I'll collect **AUC**, **Accuracy**, **Sensitivity**, **Specificity**, **Pos Pred Value**, **Neg Pred Value**, **Prevalence**, **Detection Rate**, **Detection Prevalence** and an additional term **F1** to build basic metric vector for Logistic Regression. Additionally, I'll include the computation time for this model.

```
lr_basic_metric <- c('Logistic Regression', model_auc, model_accuracy, model_caret_results_df[1, 1], moc
```

Running Naive Bayes Model

```
ptm <- proc.time()  # timing the Logistic Naive Bayes, start

model_name <- 'nb'
type_name <- 'raw'
```

```
model <- model_build(model_name, data_train, type_name)

model_prediction <- model_predict(model_name, model, data_test, type_name)
model_confusionmatrix <- model_prediction[1][[1]][2]
model_accuracy <- data.frame(model_prediction[1][[1]][3])[1, 1]

model_caret_results_df <- as.data.frame(model_prediction[1][[1]][4])

model_auc <- performance(prediction(model_prediction[[2]][,2], data_test$target), 'auc')@y.values[[1]]
# model_auc <- performance(prediction(data.frame(model_prediction[[2]][,2]), data_test$target), 'auc')@
```

Summary of Naive Bayes Model.

```
summary(model)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    13     -none- list
## levels     2     -none- character
## isnumeric 13     -none- logical
## call       4     -none- call
```

Confusion Matrix.

```
model_confusionmatrix
```

```
## $table
##    model_prediction_class
##      0  1
##   0 23  4
##   1  7 26
```

Accuracy.

```
model_accuracy
```

```
## [1] 0.8166667
```

AUC

```
model_auc
```

```
## [1] 0.8978676
```

All key parameters.

```
model_caret_results_df
```

```
##                        byClass
## Sensitivity          0.7666667
## Specificity          0.8666667
## Pos Pred Value       0.8518519
## Neg Pred Value       0.7878788
## Precision            0.8518519
## Recall               0.7666667
## F1                   0.8070175
## Prevalence           0.5000000
## Detection Rate       0.3833333
## Detection Prevalence 0.4500000
## Balanced Accuracy    0.8166667
```

Execution time.

```
nb_execution_tm <- proc.time() - ptm  # timing the Logistic Naive Bayes, end

tot_tm <- nb_execution_tm + common_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.13    0.00    0.13
```

Now, I'll build the basic metric vector for Naive Bayes.

```
nb_basic_metric <- c('Naive Bayes', model_auc, model_accuracy, model_caret_results_df[1, 1], model_caret
```

Summary table for Logistic Regression and Naive Bayes.

```
metric_table <- data.frame(matrix(ncol = 12, nrow = 0))

metric_table <- rbind(metric_table, lr_basic_metric, nb_basic_metric)

colnames(metric_table) <- c("Model", "AUC", "Accuracy", "Sensitivity", "Specificity", "Pos Pred Value",


metric_table %>% kable()
```

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Logistic Regression	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Naive Bayes	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.3833333	[illegible]	[illegible]

Cross validation

Splitting data.

In Cross Validation, begins here. The common portions of CV are being recorded first.

```
ptm <- proc.time()  # timing the CV common part, start
```

```
split_data <- split_dataset(43, heart, 0.20)

data_train_redo <- as.data.frame(split_data[1])
data_test_redo <- as.data.frame(split_data[2])
```

```
cv_common_execution_tm <- proc.time() - ptm  # timing the CV common part, end
```

Cross validation with Logistic Regression Model with folds = 10.

Please be informed that I used code from lecture M09 and modified it wherever required.

```
ptm <- proc.time()  # timing the CV Logistic Regression with 10 folds part, start

model_name <- 'glm'
type_name <- 'response'
```

I'll split the numbers from 1 to N (N is number of rows in the file) into folds or buckets of size NF = 10 and build glm model.

```
NF = 10
folds <- cv_folds_create(data_train_redo, NF)

ridx <- create_sample(data_train_redo, FALSE)
cv_df <- data.frame(cv_model_build(model_name, folds, data_train_redo, type_name, ridx))   # Build glm 



cv_model_prediction <- cv_model_predict(model_name, cv_df$m, data_test_redo, type_name)
tstcv_cfm <- cv_model_prediction[1][[1]]
tstcv_preds <- cv_model_prediction[2][[1]]


# Average of all key parameters (Accuracy etc) over 10 folds is being computed below.
cv_tst_perf_df <- cv_compute_param_average(tstcv_cfm)                   # Average of all key paramete
cv_confusion_matrix <- cv_compute_confusionmatrix_average(tstcv_cfm)      # Average confusion matrix.
cv_auc <- cv_compute_AUC_average(model_name, tstcv_preds, NF)             # Average AUC.
```

Confusion Matrix.

```
cv_confusion_matrix
```

```
##      [,1] [,2]
## [1,] 22.5  5.8
## [2,]  4.5 27.2
```

Observe fractional values at the cells of confusion matrix, due to averaging over the folders.

AUC

```
cv_auc
```

```
## [1] 0.9046016
```

Average of all key parameters.

```
cv_tst_perf_df
```

```
##                      cv_tst_perf
## Accuracy               0.8283333
## Kappa                  0.6547540
## AccuracyLower          0.7091456
## AccuracyUpper          0.9132489
## AccuracyNull           0.5500000
## Sensitivity            0.8333333
## Specificity            0.8242424
## Pos Pred Value         0.7965761
## Neg Pred Value         0.8590574
## Precision              0.7965761
```

```
## Recall                  0.8333333
## F1                      0.8137043
## Prevalence              0.4500000
## Detection Rate          0.3750000
## Detection Prevalence    0.4716667
## Balanced Accuracy       0.8287879
```

Execution time.

```
cv_lr_10_execution_tm <- proc.time() - ptm  # timing the CV Logistic Regression with 10 folds part, end

tot_tm <- cv_common_execution_tm + cv_lr_10_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.17    0.00    0.17
```

Now, I'll build the metric vector for this Cross Validation.

```
cv_lr_10_metric <- c('Cross Validation of LR with 10 folds', cv_auc, cv_tst_perf_df[1, 1], cv_tst_perf_c
```

Cross validation with Logistic Regression Model with folds = 5.

Please be informed that I used code from lecture M09 and modified it wherever required.

```
ptm <- proc.time()     # timing the CV Logistic Regression with 5 folds part, start
```

I'll split the numbers from 1 to N (N is number of rows in the file) into folds or buckets of size NF = 5 and build glm model.

```
NF = 5
folds <- cv_folds_create(data_train_redo, NF)

ridx <- create_sample(data_train_redo, FALSE)
cv_df <- as.data.frame(cv_model_build(model_name, folds, data_train_redo, type_name, ridx))   # Build g


cv_model_prediction <- cv_model_predict(model_name, cv_df$m, data_test_redo, type_name)
tstcv_cfm <- cv_model_prediction[1][[1]]
tstcv_preds <- cv_model_prediction[2][[1]]


# Average of all key parameters (Accuracy etc) over 5 folds is being computed below.
cv_tst_perf_df <- cv_compute_param_average(tstcv_cfm)                    # Average of all key paramete
cv_confusion_matrix <- cv_compute_confusionmatrix_average(tstcv_cfm)     # Average confusion matrix.
cv_auc <- cv_compute_AUC_average(model_name, tstcv_preds, NF)            # Average AUC.
```

Confusion Matrix.

```r
cv_confusion_matrix
```

```
##      [,1] [,2]
## [1,] 22.2  5.8
## [2,]  4.8 27.2
```

Observe fractional values at the cells of confusion matrix, due to averaging over the folders.

AUC

```r
cv_auc
```

```
## [1] 0.9046016
```

Average of all key parameters.

```r
cv_tst_perf_df
```

```
##                      cv_tst_perf
## Accuracy               0.8233333
## Kappa                  0.6441971
## AccuracyLower          0.7035178
## AccuracyUpper          0.9094174
## AccuracyNull           0.5500000
## Sensitivity            0.8222222
## Specificity            0.8242424
## Pos Pred Value         0.7930874
## Neg Pred Value         0.8510707
## Precision              0.7930874
## Recall                 0.8222222
## F1                     0.8068782
## Prevalence             0.4500000
## Detection Rate         0.3700000
## Detection Prevalence   0.4666667
## Balanced Accuracy      0.8232323
```

Execution time.

```r
cv_lr_5_execution_tm <- proc.time() - ptm     # timing the CV Logistic Regression with 5 folds part, en

tot_tm <- cv_common_execution_tm + cv_lr_5_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.10    0.03    0.12
```

Now, I'll build the metric vector for this Cross Validation.

```
cv_lr_5_metric <- c('Cross Validation of LR with 5 folds', cv_auc, cv_tst_perf_df[1, 1], cv_tst_perf_df
```

Cross validation with Naive Bayes Model with folds = 10.

```
ptm <- proc.time()     # timing the CV Naive Bayes with 10 folds part, start

model_name <- 'nb'
type_name <- 'raw'
```

I'll split the numbers from 1 to N (N is number of rows in the file) into folds or buckets of size NF = 10 and build the NB model.

```
NF = 10
folds <- cv_folds_create(data_train_redo, NF)

ridx <- create_sample(data_train_redo, FALSE)
cv_df <- as.data.frame(cv_model_build(model_name, folds, data_train_redo, type_name, ridx))   # Build g


cv_model_prediction <- cv_model_predict(model_name, cv_df$m, data_test_redo, type_name)
tstcv_cfm <- cv_model_prediction[1][[1]]
tstcv_preds <- cv_model_prediction[2][[1]]


# Average of all key parameters (Accuracy etc) over 10 folds is being computed below.
cv_tst_perf_df <- cv_compute_param_average(tstcv_cfm)                   # Average of all key paramete
cv_confusion_matrix <- cv_compute_confusionmatrix_average(tstcv_cfm)      # Average confusion matrix.
cv_auc <- cv_compute_AUC_average(model_name, tstcv_preds, NF)           # Average AUC.
```

Confusion Matrix.

```
cv_confusion_matrix
```

```
##      [,1] [,2]
## [1,] 22.9  8.3
## [2,]  4.1 24.7
```

Observe fractional values at the cells of confusion matrix, due to averaging over the folders.

AUC

```
cv_auc
```

```
## [1] 0.9023569
```

Average of all key parameters.

```
cv_tst_perf_df
```

```
##                         cv_tst_perf
## Accuracy                  0.7933333
## Kappa                     0.5884475
## AccuracyLower             0.6693770
## AccuracyUpper             0.8868583
## AccuracyNull              0.5500000
## Sensitivity               0.8481481
## Specificity               0.7484848
## Pos Pred Value            0.7351807
## Neg Pred Value            0.8578478
## Precision                 0.7351807
## Recall                    0.8481481
## F1                        0.7871755
## Prevalence                0.4500000
## Detection Rate            0.3816667
## Detection Prevalence      0.5200000
## Balanced Accuracy         0.7983165
```

Execution time.

```
cv_nb_10_execution_tm <- proc.time() - ptm     # timing the CV Naive Bayes with 10 folds part, end

tot_tm <- cv_common_execution_tm + cv_nb_10_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.31    0.00    0.31
```

Now, I'll build the metric vector for this Cross Validation.

```
cv_nb_10_metric <- c('Cross Validation of NB with 10 folds', cv_auc, cv_tst_perf_df[1, 1], cv_tst_perf_
```

Cross validation with Naive Bayes Model with folds = 5.

```
ptm <- proc.time()     # timing the CV Naive Bayes with 5 folds part, start
```

I'll split the numbers from 1 to N (N is number of rows in the file) into folds or buckets of size NF = 5 and build NB model.

```
NF = 5
folds <- cv_folds_create(data_train_redo, NF)

ridx <- create_sample(data_train_redo, FALSE)
cv_df <- as.data.frame(cv_model_build(model_name, folds, data_train_redo, type_name, ridx))   # Build g


cv_model_prediction <- cv_model_predict(model_name, cv_df$m, data_test_redo, type_name)
tstcv_cfm <- cv_model_prediction[1][[1]]
tstcv_preds <- cv_model_prediction[2][[1]]
```

```
# Average of all key parameters (Accuracy etc) over 5 folds is being computed below.
cv_tst_perf_df <- cv_compute_param_average(tstcv_cfm)                    # Average of all key paramete
cv_confusion_matrix <- cv_compute_confusionmatrix_average(tstcv_cfm)      # Average confusion matrix.
cv_auc <- cv_compute_AUC_average(model_name, tstcv_preds, NF)             # Average AUC.
```

Confusion Matrix.

```
cv_confusion_matrix
```

```
##      [,1] [,2]
## [1,] 23.2    8
## [2,]  3.8   25
```

Observe fractional values at the cells of confusion matrix, due to averaging over the folders.

AUC

```
cv_auc
```

```
## [1] 0.9012346
```

Average of all key parameters.

```
cv_tst_perf_df
```

```
##                      cv_tst_perf
## Accuracy               0.8033333
## Kappa                  0.6082071
## AccuracyLower          0.6806344
## AccuracyUpper          0.8945124
## AccuracyNull           0.5500000
## Sensitivity            0.8592593
## Specificity            0.7575758
## Pos Pred Value         0.7440726
## Neg Pred Value         0.8691176
## Precision              0.7440726
## Recall                 0.8592593
## F1                     0.7970774
## Prevalence             0.4500000
## Detection Rate         0.3866667
## Detection Prevalence   0.5200000
## Balanced Accuracy      0.8084175
```

Execution time.

```
cv_nb_5_execution_tm <- proc.time() - ptm     # timing the CV Naive Bayes with 5 folds part, end

tot_tm <- cv_common_execution_tm + cv_nb_5_execution_tm

tot_tm
```

```
##    user  system elapsed 
##    0.21    0.00    0.20
```

Now, I'll build the basic metric vector for Cross Validation.

```r
cv_nb_5_metric <- c('Cross Validation of NB with 5 folds', cv_auc, cv_tst_perf_df[1, 1], cv_tst_perf_df
```

Summary table for Cross Validation.

```r
metric_table <- data.frame(matrix(ncol = 12, nrow = 0))

metric_table <- rbind(metric_table, cv_lr_10_metric, cv_lr_5_metric, cv_nb_10_metric, cv_nb_5_metric)

colnames(metric_table) <- c("Model", "AUC", "Accuracy", "Sensitivity", "Specificity", "Pos Pred Value",


metric_table %>% kable()
```

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Cross Validation of LR with 10 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Cross Validation of LR with 5 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Cross Validation of NB with 10 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.31
Cross Validation of NB with 5 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.2

Bootstrapping.

```r
ptm <- proc.time()  # timing the BS part, start
```

```r
split_data <- split_dataset(43, heart, 0.20)

data_train_redo <- as.data.frame(split_data[1])
data_test_redo <- as.data.frame(split_data[2])
```

```r
bs_common_execution_tm <- proc.time() - ptm  # timing the BS part, end
```

Bootstrapping with Logistic Regression Model.

```r
ptm <- proc.time()  # timing the BS with Logistic Regression, start

model_name <- 'glm'
type_name <- 'response'
```

```r
ridx <- create_sample(data_train_redo, TRUE)

model <- model_build(model_name, data_train_redo[ridx,], type_name)
runModel <- function(data_train_redo) { model }
lapplyrunmodel <- function(x)runModel(data_train_redo)

NF = 200
model <- lapply(1:NF,lapplyrunmodel)
```

```r
cv_model_prediction <- cv_model_predict(model_name, model, data_test_redo, type_name)
tstcv_cfm <- cv_model_prediction[1][[1]]
tstcv_preds <- cv_model_prediction[2][[1]]


# Average of all key parameters (Accuracy etc) over 10 folds is being computed below.
cv_tst_perf_df <- cv_compute_param_average(tstcv_cfm)                    # Average of all key paramete
cv_confusion_matrix <- cv_compute_confusionmatrix_average(tstcv_cfm)      # Average confusion matrix.
cv_auc <- cv_compute_AUC_average(model_name, tstcv_preds, NF)             # Average AUC.
```

Confusion Matrix.

```r
cv_confusion_matrix
```

```
##      [,1] [,2]
## [1,]   22    7
## [2,]    5   26
```

AUC

```r
cv_auc
```

```
## [1] 0.8866442
```

Average of all key parameters.

```r
cv_tst_perf_df
```

```
##                      cv_tst_perf
## Accuracy               0.8000000
## Kappa                  0.5986622
## AccuracyLower          0.6766996
## AccuracyUpper          0.8921589
## AccuracyNull           0.5500000
## Sensitivity            0.8148148
## Specificity            0.7878788
## Pos Pred Value         0.7586207
## Neg Pred Value         0.8387097
## Precision              0.7586207
## Recall                 0.8148148
## F1                     0.7857143
## Prevalence             0.4500000
## Detection Rate         0.3666667
## Detection Prevalence   0.4833333
## Balanced Accuracy      0.8013468
```

Execution time.

```
bs_lr_execution_tm <- proc.time() - ptm    # timing the BS with Logistic Regression, end

tot_tm <- bs_common_execution_tm + bs_lr_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.56    0.00    0.57
```

Now, I'll build the metric vector for this Bootstrapping.

```
bs_lr_metric <- c('Bootstrapping with LR', cv_auc, cv_tst_perf_df[1, 1], cv_tst_perf_df[6, 1], cv_tst_pe

bs_lr_metric
```

```
##  [1] "Bootstrapping with LR" "0.886644219977553"     "0.8"
##  [4] "0.814814814814815"     "0.787878787878788"     "0.758620689655172"
##  [7] "0.838709677419355"     "0.785714285714286"     "0.45"
## [10] "0.366666666666667"     "0.483333333333333"     "0.57"
```

Bootstrapping with Naive Bayes Model.

```
ptm <- proc.time()  # timing the BS with Naive Bayes, start

model_name <- 'nb'
type_name <- 'raw'
```

```
ridx <- create_sample(data_train_redo, TRUE)

model <- model_build(model_name, data_train_redo[ridx,], type_name)
```

```r
runModel <- function(data_train_redo) { model }
lapplyrunmodel <- function(x)runModel(data_train_redo)

NF = 200
model <- lapply(1:NF,lapplyrunmodel)
```

```r
cv_model_prediction <- cv_model_predict(model_name, model, data_test_redo, type_name)
tstcv_cfm <- cv_model_prediction[1][[1]]
tstcv_preds <- cv_model_prediction[2][[1]]


# Average of all key parameters (Accuracy etc) over 10 folds is being computed below.
cv_tst_perf_df <- cv_compute_param_average(tstcv_cfm)                  # Average of all key paramete
cv_confusion_matrix <- cv_compute_confusionmatrix_average(tstcv_cfm)      # Average confusion matrix.
cv_auc <- cv_compute_AUC_average(model_name, tstcv_preds, NF)             # Average AUC.
```

Confusion Matrix.

```r
cv_confusion_matrix
```

```
##      [,1] [,2]
## [1,]   22    7
## [2,]    5   26
```

AUC

```r
cv_auc
```

```
## [1] 0.8832772
```

Average of all key parameters.

```r
cv_tst_perf_df
```

```
##                      cv_tst_perf
## Accuracy               0.8000000
## Kappa                  0.5986622
## AccuracyLower          0.6766996
## AccuracyUpper          0.8921589
## AccuracyNull           0.5500000
## Sensitivity            0.8148148
## Specificity            0.7878788
## Pos Pred Value         0.7586207
## Neg Pred Value         0.8387097
## Precision              0.7586207
## Recall                 0.8148148
## F1                     0.7857143
## Prevalence             0.4500000
## Detection Rate         0.3666667
## Detection Prevalence   0.4833333
## Balanced Accuracy      0.8013468
```

Now, I'll build the metric vector for this Bootstrapping.

Execution time.

```
bs_nb_execution_tm <- proc.time() - ptm    # timing the BS with Logistic Regression, end

tot_tm <- bs_common_execution_tm + bs_nb_execution_tm

tot_tm
```

```
##    user  system elapsed
##    3.08    0.00    3.07
```

```
bs_nb_metric <- c('Bootstrapping with NB', cv_auc, cv_tst_perf_df[1, 1], cv_tst_perf_df[6, 1], cv_tst_pe

bs_nb_metric
```

```
##  [1] "Bootstrapping with NB" "0.88327721661055"      "0.8"
##  [4] "0.814814814814815"     "0.787878787878788"     "0.758620689655172"
##  [7] "0.838709677419355"     "0.785714285714286"     "0.45"
## [10] "0.366666666666667"     "0.483333333333333"     "3.07"
```

Summary table for Bootstrapping.

```
metric_table <- data.frame(matrix(ncol = 12, nrow = 0))

metric_table <- rbind(metric_table, bs_lr_metric, bs_nb_metric)

colnames(metric_table) <- c("Model", "AUC", "Accuracy", "Sensitivity", "Specificity", "Pos Pred Value",


metric_table %>% kable()
```

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Bootstrapping with LR	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Bootstrapping with NB	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Summary table for Part A.

```
metric_table <- data.frame(matrix(ncol = 12, nrow = 0))

metric_table <- rbind(metric_table, lr_basic_metric, nb_basic_metric, cv_lr_10_metric, cv_lr_5_metric, c
```

```r
colnames(metric_table) <- c("Model", "AUC", "Accuracy", "Sensitivity", "Specificity", "Pos Pred Value",

metric_table %>% kable()
```

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Logistic Regression	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.12
Naive Bayes	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.13
Cross Validation of LR with 10 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.375	[illegible]	[illegible]
Cross Validation of LR with 5 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.37	[illegible]	[illegible]
Cross Validation of NB with 10 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.31
Cross Validation of NB with 5 folds	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.2
Bootstrapping with LR	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Bootstrapping with NB	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

PART B

For the same dataset, set seed (43) split 80/20.

Using randomForest grow three different forests varying the number of trees atleast three times. Start with seeding and fresh split for each forest. Note down as best as you can development (engineering) cost as well as computing cost(elapsed time) for each run. And compare these results with the experiment in Part A. Submit a pdf and executable script in python or R.

Answer:

First Random Forest execution.

I'll run first RF with 40.

```
ptm <- proc.time()  # timing the RF with 40, start
model_name  <- 'rf'
```

```
split_data <- split_dataset(43, heart, 0.20)

data_train_redo <- as.data.frame(split_data[1])
data_test_redo <- as.data.frame(split_data[2])
```

```
model <- model_build(model_name, data_train_redo, 40)

model_prediction <- model_predict(model_name, model, data_test_redo, 'NO')
model_confusionmatrix <- model_prediction[1][[1]][2]
model_accuracy <- data.frame(model_prediction[1][[1]][3])[1, 1]
model_caret_results_df <- as.data.frame(model_prediction[1][[1]][4])

# Note: My usual method for computing auc will not work in this case, because the the variable model_pr
auc <- roc(as.numeric(data_test_redo$target), as.numeric(as.matrix((predict(model, data_test_redo, type
```

```
## Setting levels: control = 1, case = 2

## Setting direction: controls < cases
```

Summary of Logistic Regression Model.

```
summary(model)
```

```
##                 Length Class  Mode
## call              4    -none- call
## type              1    -none- character
## predicted       243    factor numeric
## err.rate        120    -none- numeric
## confusion         6    -none- numeric
## votes           486    matrix numeric
## oob.times       243    -none- numeric
## classes           2    -none- character
## importance       13    -none- numeric
## importanceSD      0    -none- NULL
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           14    -none- list
## y               243    factor numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

Confusion Matrix.

```
model_confusionmatrix
```

```
## $table
##    model_prediction
##      0  1
##   0 23  4
##   1  6 27
```

Accuracy.

```
model_accuracy
```

```
## [1] 0.8333333
```

AUC.

```
model_auc
```

```
## [1] 0.8978676
```

Average of all key parameters.

```
model_caret_results_df
```

```
##                        byClass
## Sensitivity          0.7931034
## Specificity          0.8709677
## Pos Pred Value       0.8518519
## Neg Pred Value       0.8181818
## Precision            0.8518519
## Recall               0.7931034
## F1                   0.8214286
## Prevalence           0.4833333
## Detection Rate       0.3833333
## Detection Prevalence 0.4500000
## Balanced Accuracy    0.8320356
```

Execution time.

```
rf_40_execution_tm <- proc.time() - ptm  # timing the RF with 40, end

tot_tm <- rf_40_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.06    0.01    0.07
```

Now, I'll build the metric vector for this Random Forest with 40.

```
rf_40_metric <- c('Random Forest with 40', model_auc, model_accuracy, model_caret_results_df[1, 1], mode
```

Second Random Forest execution.

I'll grow by RF by 60.

```
ptm <- proc.time()  # timing the RF with 40, start
```

```
split_data <- split_dataset(43, heart, 0.20)

data_train_redo <- as.data.frame(split_data[1])
data_test_redo <- as.data.frame(split_data[2])
```

WHen I used model returned by the first run in grow(model, 60), I got the following error:
"Error in terms.formula(formula, data = data) : 'data' argument is of the wrong type"

However, on directly calling randomForest, without calling via my function model_build, I did not get the same error. It's a bit weird because the the model returned by first run was used for predicting, and I also checked the class of model with model(class) and got the right results "[1]"randomForest.formula" "randomForest"".

So, now that I am out of time and also lost quite a bit in other trouble shootings, I am directly calling randomForest, as a quick and easy fix.

```
# class(model)
model <- randomForest(target~., data = data_train_redo, ntree = 40)
```

```
model <- grow(model, 60)
model_prediction <- model_predict(model_name, model, data_test_redo, 'NO')
model_confusionmatrix <- model_prediction[1][[1]][2]
model_accuracy <- data.frame(model_prediction[1][[1]][3])[1, 1]
model_caret_results_df <- as.data.frame(model_prediction[1][[1]][4])

# Note: My usual method for computing auc will not work in this case, because the the variable model_pr
auc <- roc(as.numeric(data_test_redo$target), as.numeric(as.matrix((predict(model, data_test_redo, type
```

```
## Setting levels: control = 1, case = 2
```

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

Summary of Logistic Regression Model.

```
summary(model)
```

```
##                 Length Class  Mode
## call              4    -none- call
## type              1    -none- character
## predicted       243    factor numeric
## votes           486    -none- numeric
## oob.times       243    -none- numeric
```

```
## classes           2    -none- character
## importance       13    -none- numeric
## importanceSD      0    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           14    -none- list
## y               243    factor numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

Confusion Matrix.

```
model_confusionmatrix
```

```
## $table
##    model_prediction
##      0  1
##   0 22  5
##   1  4 29
```

Accuracy.

```
model_accuracy
```

```
## [1] 0.85
```

AUC.

```
model_auc
```

```
## [1] 0.8978676
```

Average of all key parameters.

```
model_caret_results_df
```

```
##                        byClass
## Sensitivity          0.8461538
## Specificity          0.8529412
## Pos Pred Value       0.8148148
## Neg Pred Value       0.8787879
## Precision            0.8148148
## Recall               0.8461538
## F1                   0.8301887
## Prevalence           0.4333333
## Detection Rate       0.3666667
## Detection Prevalence 0.4500000
## Balanced Accuracy    0.8495475
```

Execution time.

```
rf_60_execution_tm <- proc.time() - ptm  # timing the RF with 40, end

tot_tm <- rf_60_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.08    0.00    0.08
```

Now, I'll build the metric vector for this Random Forest with 100.

```
rf_60_metric <- c('Random Forest with 100', model_auc, model_accuracy, model_caret_results_df[1, 1], mod
```

Third Random Forest execution.

I'll grow by RF by 100.

```
ptm <- proc.time()  # timing the RF with 40, start
```

```
split_data <- split_dataset(43, heart, 0.20)

data_train_redo <- as.data.frame(split_data[1])
data_test_redo <- as.data.frame(split_data[2])
```

```
model <- grow(model, 100)
model_prediction <- model_predict(model_name, model, data_test_redo, 'NO')
model_confusionmatrix <- model_prediction[1][[1]][2]
model_accuracy <- data.frame(model_prediction[1][[1]][3])[1, 1]
model_caret_results_df <- as.data.frame(model_prediction[1][[1]][4])

# Note: My usual method for computing auc will not work in this case, because the the variable model_pr
auc <- roc(as.numeric(data_test_redo$target), as.numeric(as.matrix((predict(model, data_test_redo, type
```

```
## Setting levels: control = 1, case = 2
```

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

Summary of Logistic Regression Model.

```
summary(model)
```

```
##                 Length Class  Mode
## call              4    -none- call
## type              1    -none- character
## predicted       243    factor numeric
## votes           486    -none- numeric
## oob.times       243    -none- numeric
## classes           2    -none- character
## importance       13    -none- numeric
```

```
## importanceSD      0    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           14    -none- list
## y               243    factor numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

Confusion Matrix.

```
model_confusionmatrix
```

```
## $table
##    model_prediction
##      0  1
##   0 22  5
##   1  5 28
```

Accuracy.

```
model_accuracy
```

```
## [1] 0.8333333
```

AUC.

```
model_auc
```

```
## [1] 0.8978676
```

Average of all key parameters.

```
model_caret_results_df
```

```
##                        byClass
## Sensitivity          0.8148148
## Specificity          0.8484848
## Pos Pred Value       0.8148148
## Neg Pred Value       0.8484848
## Precision            0.8148148
## Recall               0.8148148
## F1                   0.8148148
## Prevalence           0.4500000
## Detection Rate       0.3666667
## Detection Prevalence 0.4500000
## Balanced Accuracy    0.8316498
```

Execution time.

```r
rf_100_execution_tm <- proc.time() - ptm  # timing the RF with 40, end

tot_tm <- rf_100_execution_tm

tot_tm
```

```
##    user  system elapsed
##    0.08    0.00    0.08
```

Now, I'll build the metric vector for this Random Forest with 160.

```r
rf_100_metric <- c('Random Forest with 160', model_auc, model_accuracy, model_caret_results_df[1, 1], mo
```

Summary table for Part B.

```r
metric_table <- data.frame(matrix(ncol = 12, nrow = 0))

metric_table <- rbind(metric_table, rf_40_metric, rf_60_metric, rf_100_metric)

colnames(metric_table) <- c("Model", "AUC", "Accuracy", "Sensitivity", "Specificity", "Pos Pred Value",


metric_table %>% kable()
```

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Random Forest with 40	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Random Forest with 100	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Random Forest with 160	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Part C

Include a summary of your findings. Which of the two methods bootstrap vs cv do you recommend to your customer? And why? Be elaborate. Including computing costs, engineering costs and model performance. Did you incorporate Pareto's maxim or the Razor and how did these two heuristics influence your decision?

Answer:

Summary table for Part A, B (all models).

```
metric_table <- data.frame(matrix(ncol = 12, nrow = 0))

metric_table <- rbind(metric_table, lr_basic_metric, nb_basic_metric, cv_lr_10_metric, cv_lr_5_metric, c

colnames(metric_table) <- c("Model", "AUC", "Accuracy", "Sensitivity", "Specificity", "Pos Pred Value",


metric_table %>% kable()
```

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Logistic Regression	0.906846[illegible]	[illegible]	[illegible]	[illegible]	0.8148148	[illegible]	[illegible]	0.4666666	[illegible]	[illegible]	0.12
Naive Bayes	0.897867[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.3833333	[illegible]	[illegible]	0.13
Cross Validation of LR with 10 folds	0.904601[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.375		0.4716666	[illegible]
Cross Validation of LR with 5 folds	0.904601[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.37		0.4666666	[illegible]
Cross Validation of NB with 10 folds	0.902356[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.3816666	[illegible]	[illegible]	0.31
Cross Validation of NB with 5 folds	0.901234[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.3866666	[illegible]	[illegible]	0.2
Bootstrapping with LR	0.886644[illegible]	[illegible]	0.8148148	[illegible]	[illegible]	[illegible]	[illegible]	0.3666666	[illegible]	[illegible]	[illegible]
Bootstrapping with NB	0.883277[illegible]	[illegible]	0.8148148	[illegible]	[illegible]	[illegible]	[illegible]	0.3666666	[illegible]	[illegible]	[illegible]
Random Forest with 40	0.897867[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0700000000000003
Random Forest with 100	0.897867[illegible]	[illegible]	0.8461538	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.0800000000000001

Model	AUC	Accuracy	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Prevalence	Detection Rate	Detection Prevalence	F1	Elapsed time
Random Forest with 160	0.8978675[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.08000000000000001

The table speaks for the performance of the model. Here ar some of the observations:

1)Although the accuracies were the same, NB (2nd row) took a little more than LR (1st row). However, the AUC of NB was lesser than LR.
2)With almost the same accuracies, each of the Cross Validations took much less time than standalone LR and NB.
3)Bootstrapping took way more time, with almost same level of accuracy. 4)Random Forest seems to have performed better than others.

As far as I know, Pareto's principle states, "80% of consequences come from 20% of the causes". I am not sure how I would even apply this principle here.

Occam's Razor states that "Entities are not to multiplied without necessity" (Reference, History Of Western Philosophy, page 462, chapter XIV, "Franciscan Schoolmen"). The general interpretation of this has been, to reduce the number of assumtions to a minimum. In order to explain something, if it is sufficient to make three assumptions, then it's not necessary to postulate a fourth one.

Yes, I didn't make any unnecessary assumptions here.

All in all, Random Forest seems to be the most performant, or did I make any mistake?

The Engineering

In order to accomplish this project, since time is short, I selected two models. Furthermore, in doing part B, I am required to do Random Forest. So, Logistic Regression and Naive Bayes are not bad choices.

In this exercise, I realized that there were repeatitiion of tasks. So, I wrote functions instead of flat out scripting approach. That way, I not only resued the same code over and over, but also built my own library of useful functions for later use. I saved time for future.

It was quite an arduous task. Had it been Python, I wouldn't have that many surprises. R doesn't return multiple values (not uncommon in languages). However, several objects can be put into a list to return out of a function. But, the return values were often not what I expected. So, many experiments had to be done and I had to constantly troubleshoot my way through a forest of quagmires. I enjoyed that.

Marker: 622_p