Assignment 2

Abhiram Ravipati

2024-02-18

Recitation Exercises

Chapter 4

4)a)According to the data above, we can say that the data is uniformly distributed, which means that the probability of each and every point is equal. Therefore, if we take the above example of 0.6 prediction and consider the whole range equally distributed, it comes out to be about 100 points. On average, we would be using a range of 10 / 100 of the total range, which means that we would be using a range of 1/10 th of the total or 10% at any given point.

b)We can determine that it was 10% for a single observation uniformly distributed in [0,1] from the prior solution. In this case, [0,1]*[0,1] has a uniform distribution of (X1,X2). Given a total of 100 observations for X1 and 100 observations for X2, the fraction of available observations is $\frac{100}{10000}$, or 1%. Thus, the forecast is based on 1% of the observations.

c)Based on the two solutions mentioned above, we can provide a general method for this sort, which is (10/100)^n. Since number of features (n), is equal to 100 in this case, we can also write it as

$$\left(\frac{10}{100}\right)^{100}$$

d)As the value of n or p in this case increases, it is clear from the above that the data taken into consideration for prediction continues to drop exponentially, and as a result, the probability that the output will be relevant or accurate likewise declines. Consequently, we conclude that when p is big, KNN is unable to accurately anticipate the observation.

e) In the event when p = 1, length = 0.1 ˆ1 = 0.1 In the case of p=2, length=0.1ˆ(1/2) = 0.316 In the case where p = 100, length = 0.1ˆ(1/100) = 0.977.Notice that the side length approaches 1 as p grows. This could mean that the concentration of the observations is closer to the hypercube’s boundary as p grows.

6)

a) By applying the logistic regression model, we obtain a probability value of 0.38.

b) We obtain the answer of 50 hours by using the values of the previously mentioned values given in the question in the logistic regression formula.

7) After substituting the previously mentioned data into the appropriate formulas we obtain a probability of 0.752, or 75.2%.

9)a)

$$\text{odds} = \frac{P(x)}{1 - P(x)}$$

rewriting the above formula we will get,

0.37 - 0.37P(x) = P(x)

which can be written as

$\text{odds} = \frac{0.37}{0.37}$= 0.27

b) By substituting in the above equation

$$\text{odds} = \frac{P(x)}{1 - P(x)}$$

we will be getting,

$\text{odds} = \frac{0.16}{1-0.16}$= 0.19

so, it will be default by 19%.

Chapter 5

2)a)An approach called bootstrap sampling involves continually collecting sample data and replacing it with original data. Let's say we have n observations. With replacement and repetition, there are an infinite number of permutations that can be made.Keeping in mind that the chance of selecting any sample is the same for all of them, and it is (1/n).Thus, 1-1/n, or (n-1)/n, is the probability that the first bootstrap sample is not the jth observation.

b) Once again, it is the same as 1-1/n since the bootstrap model creates repeated sampling with replacement, meaning that choosing a new sample is independent of the samples that have already been chosen.

c) The likelihood that an observation is not a bootstrap sample is always 1-1/n. Consequently, the chance of not selecting the sample "n" times is as follows: (1-1/n) * (1-1/n) ……… (1-1/n) (for n times).

Hence, Probabability =

$$\left(\frac{1}{n}\right)^{n}$$

$$\left(\frac{1}{n}\right)^{5}$$

is the likelihood that the jth observation is in the bootstrap.

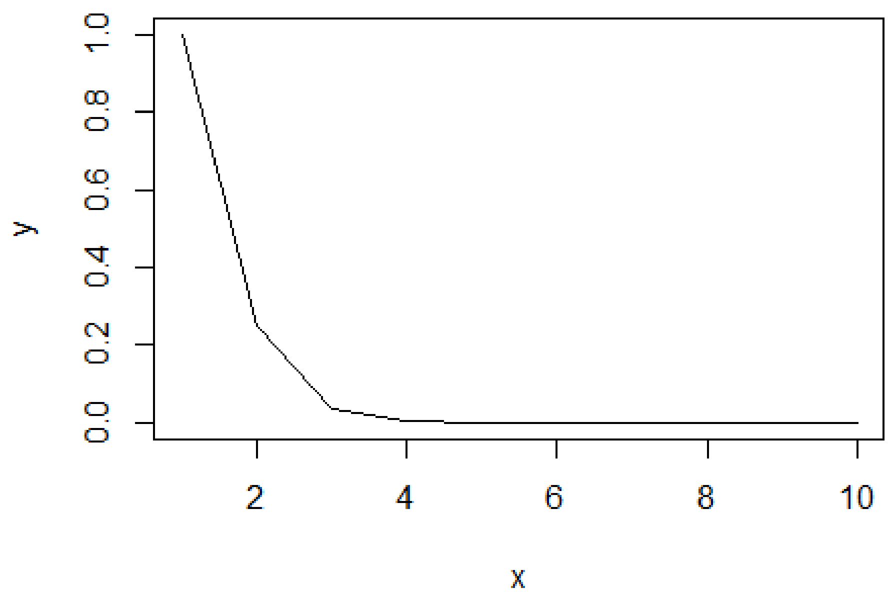
$$\left(\frac{1}{n}\right)^{100}$$

is the likelihood that the jth observation is in the bootstrap.

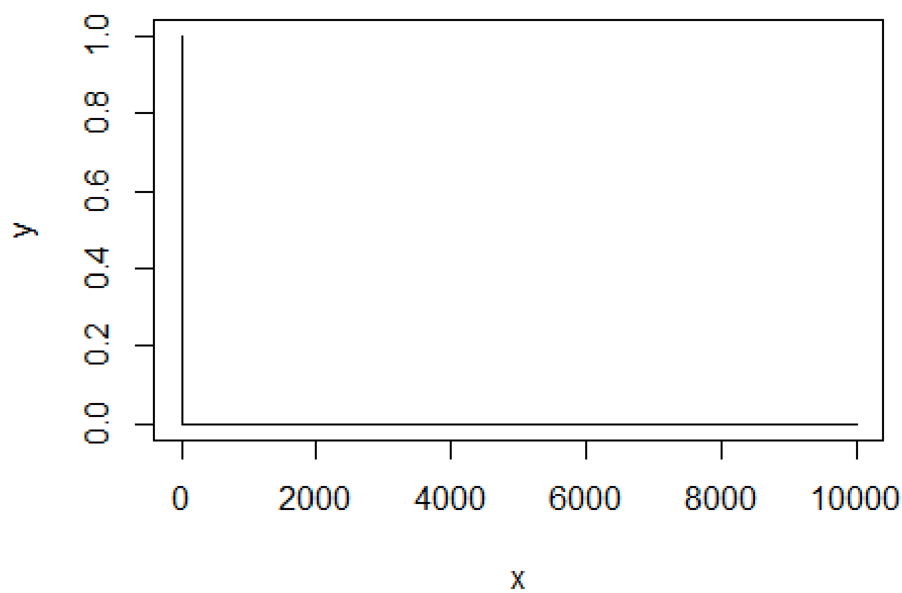
$$\left(\frac{1}{n}\right)^{10000}$$

is the likelihood that the jth observation is in the bootstrap.

```
j_probabality = function(n){ (1/n)^n}
x = 1:10
y = sapply(x, j_probabality)
plot(x,y,type='l')
```




```
j_probabality = function(n){ (1/n)^n}
x = 1:10000
y = sapply(x, j_probabality)
plot(x,y,type='l')
```



As you can see above, the graph has a value for smaller values of x but tends to become saturated for larger values of x, reaching 0.

```
a=rep(NA,10000)
for(i in 1:10000){
a[i]=sum(sample(1:100,rep=TRUE)==4)>0
}
mean(a)

## [1] 0.6187
```

The code snippet above shows how we generate 10,000 samples of numbers between 1 and 100 (inclusive) using replacement and count how many times the number 4 (our jth observation) appears in each sample. It is noted in a list named "a" if it happens at least once. The concluding statement indicates that the number 4 is present in roughly 63.4% of the samples. Because sampling is random, the result may differ, but the value will be near to 63%.

The value has a 0.01 probability, or 1 out of 100, of appearing in any sample. Therefore, there is a 64% chance that it will occur in a sample of size 100. Additional research will yield an approximation of the value 64% since the repeated experiments are independent and the mean is an unbiased estimate.

3) Cross-validation is fundamentally needed to highlight the need to develop a more generalised model that fits well with fresh or previously unseen data. Cross validations operate fundamentally as follows:

1) The dataset is divided into three sections: training data (n = 4), validation data, and testing data. The total amount of training and validation data makes up around 80% of the data, with the remaining 20% classified as testing data.
2) The 80% data is then divided into K parts (let's say 4 parts, each with 20% of the data). After that, we classify some data as training data and the remaining portion as validation data. We then use the training data to train the model and assess it using the validation data. We now classify a different portion of the 80% data—not the same portion as previously used—as validation data, classify the remaining data as training data, and use fit and predict to assess the model.
3) 3)We continue step 2 until all feasible combinations are made, at which point the model is trained with all available patterns. Next, we introduce testing or unseen data and assess the model to determine its optimal accuracy for applying the model to broader contexts.
4) Depending on our goals, we can divide the combined data (training + validation) into any number of segments, which we can refer to as N fold cross validation.

b)
   i) One of the benefits of employing the validation set approach is the drawback of using a method like the k-fold. When using the k fold strategy, a large number of computations and repeated data accesses are required, which results in a very high computational cost. For certain models, this could be excessively expensive or time-consuming. Given that the model only needs to be tested and learned from the data once, the validation set technique is clearly advantageous in this situation. On the other hand, a validation set might not exist (because there aren't many observations available) or might be prohibitively expensive to acquire in real life. Since it enables us to fine-tune our model, the k-fold cross validation emerges as the clear winner in these situations. Furthermore, because there is less data utilised for training, a validation set may have a tendency to overestimate the error.
   ii) Since the model must be trained and tested n times instead of k, the Leave-One-Out Cross Validation (LOOCV) approach has a worse (or equal, in the case of k=n) computational cost to K-Fold Cross Validation. Additionally, because the majority of trained models exhibit strong correlation, LOOCV results have a higher variance. Since 5-fold or 10-fold will greatly reduce the computing cost and neither suffer from excessive bias nor variance, there are no appreciable advantages to adopting LOOCV.

Practicum Problems:

```
options(repos = "https://cran.r-project.org/")
require("mlegp")

## Loading required package: mlegp

library(mlegp)
library(ggplot2)
library(plyr)
library("caret")
```

```
## Loading required package: lattice
```

```r
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ lubridate 1.9.3     ✔ tibble    3.2.1
## ✔ purrr     1.0.2     ✔ tidyr     1.3.1
```

```
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::arrange()   masks plyr::arrange()
## ✖ purrr::compact()   masks plyr::compact()
## ✖ dplyr::count()     masks plyr::count()
## ✖ dplyr::desc()      masks plyr::desc()
## ✖ dplyr::failwith()  masks plyr::failwith()
## ✖ dplyr::filter()    masks stats::filter()
## ✖ dplyr::id()        masks plyr::id()
## ✖ dplyr::lag()       masks stats::lag()
## ✖ purrr::lift()      masks caret::lift()
## ✖ dplyr::mutate()    masks plyr::mutate()
## ✖ dplyr::rename()    masks plyr::rename()
## ✖ dplyr::summarise() masks plyr::summarise()
## ✖ dplyr::summarize() masks plyr::summarize()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```r
require(prediction)
```

```
## Loading required package: prediction
```

```r
library(prediction)
library(mlegp)
install.packages("mlegp")
```

```
## Warning: package 'mlegp' is in use and will not be installed
```

```r
install.packages("prediction")
```

```
## Warning: package 'prediction' is in use and will not be installed
```

```r
require("caret")
require("glm2")
```

```
## Loading required package: glm2
```

```r
library("caret")
library(dplyr)
```

```r
library(ggplot2)
library(pROC)

## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

#Problem 1
library(caret)
library(corrplot)

## corrplot 0.92 loaded

abalone_dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-
databases/abalone/abalone.data", header = FALSE)

head(abalone_dataset)

##   V1    V2    V3    V4     V5     V6     V7    V8 V9
## 1  M 0.455 0.365 0.095 0.5140 0.2245 0.1010 0.150 15
## 2  M 0.350 0.265 0.090 0.2255 0.0995 0.0485 0.070  7
## 3  F 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.210  9
## 4  M 0.440 0.365 0.125 0.5160 0.2155 0.1140 0.155 10
## 5  I 0.330 0.255 0.080 0.2050 0.0895 0.0395 0.055  7
## 6  I 0.425 0.300 0.095 0.3515 0.1410 0.0775 0.120  8

col_names <- c("Sex", "Length", "Diameter", "Height", "WholeWeight",
               "ShuckedWeight", "VisceraWeight", "ShellWeight", "Rings")

colnames(abalone_dataset) <- col_names

abalone_dataset <- abalone_dataset[abalone_dataset$Sex != "I", ]

abalone_dataset$Sex = as.factor(abalone_dataset$Sex)

is.factor(abalone_dataset$Sex)

## [1] TRUE

contrasts(abalone_dataset$Sex)

##   M
## F 0
## M 1

test_train_split_abalone_dataset <- createDataPartition(abalone_dataset$Sex,
p = 0.8, list = FALSE)
train_dataset <- abalone_dataset[test_train_split_abalone_dataset, ]
test_dataset <- abalone_dataset[-test_train_split_abalone_dataset, ]
```

```r
model <- glm(Sex ~ ., data = train_dataset, family = binomial)
summary(model)
```

```
## 
## Call:
## glm(formula = Sex ~ ., family = binomial, data = train_dataset)
## 
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)    
## (Intercept)    2.652659   0.511851   5.182 2.19e-07 ***
## Length        -1.797832   2.255274  -0.797   0.4254    
## Diameter      -4.095876   2.683046  -1.527   0.1269    
## Height        -2.619169   1.925958  -1.360   0.1739    
## WholeWeight    0.342485   0.823863   0.416   0.6776    
## ShuckedWeight  2.413284   0.983273   2.454   0.0141 *  
## VisceraWeight -2.847288   1.448645  -1.965   0.0494 *  
## ShellWeight    0.122140   1.283828   0.095   0.9242    
## Rings         -0.004016   0.017818  -0.225   0.8217    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 3131.7  on 2268  degrees of freedom
## Residual deviance: 3077.8  on 2260  degrees of freedom
## AIC: 3095.8
## 
## Number of Fisher Scoring iterations: 4
```

The p-values of V1,V3 and V6 are the only one's which are very less and which can be considered as relevant predictors. Since V1 is what we are using as baseline, the other two V3 and V6 can be considered as relevant predictors based on the p-values.

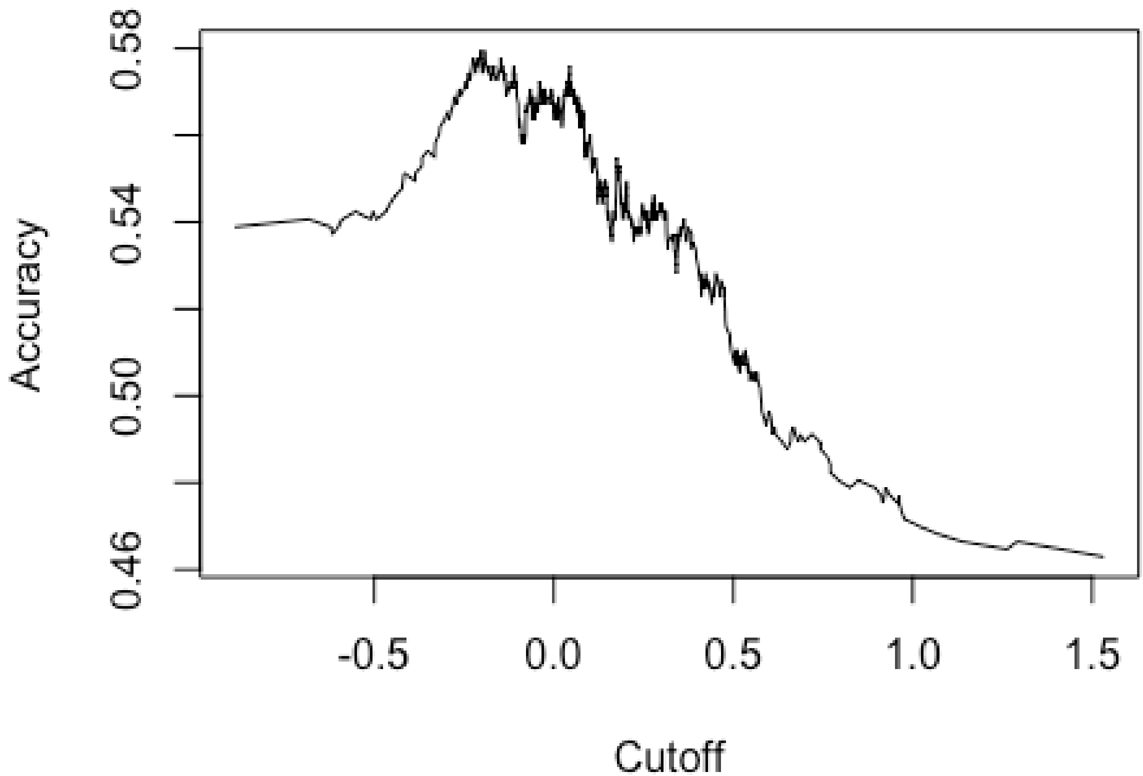
```r
confidence_intervals <- confint(model)
```

```
## Waiting for profiling to be done...
```

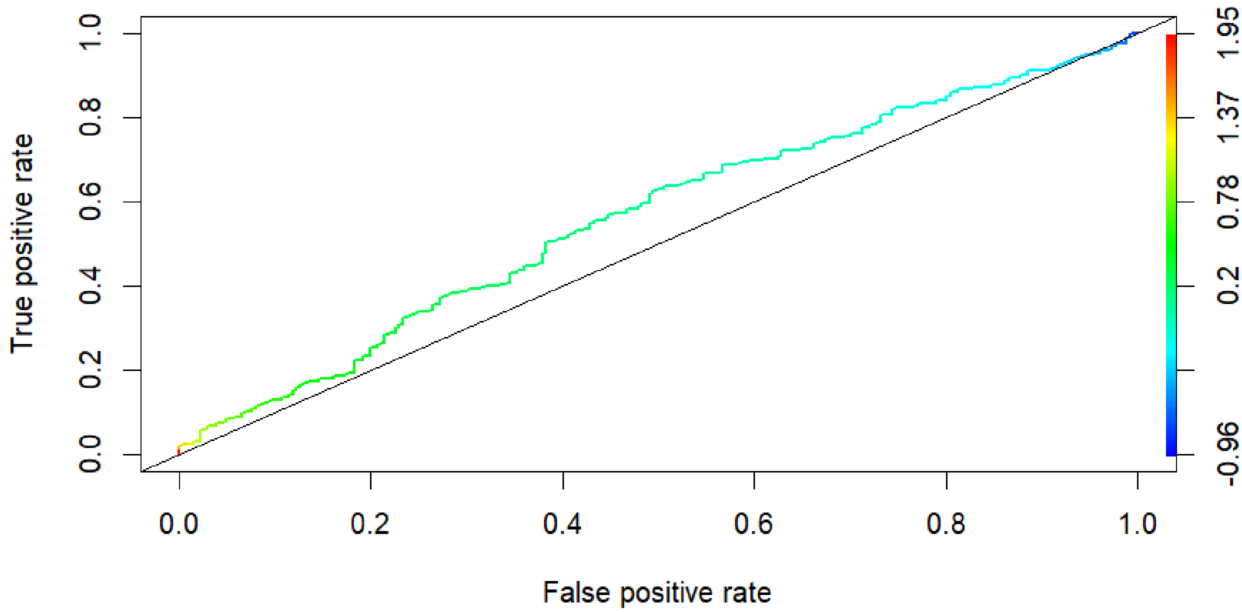
```r
print(confidence_intervals)
```

```
##                     2.5 %      97.5 %
## (Intercept)    1.66314223  3.67119876
## Length        -6.22152729  2.62554158
## Diameter      -9.36972125  1.15660666
## Height        -7.02713158  0.68637452
## WholeWeight   -1.27630110  1.96551341
## ShuckedWeight  0.48819607  4.35142786
## VisceraWeight -5.70067857 -0.01441056
## ShellWeight   -2.40178006  2.64289493
## Rings         -0.03896422  0.03093548
```

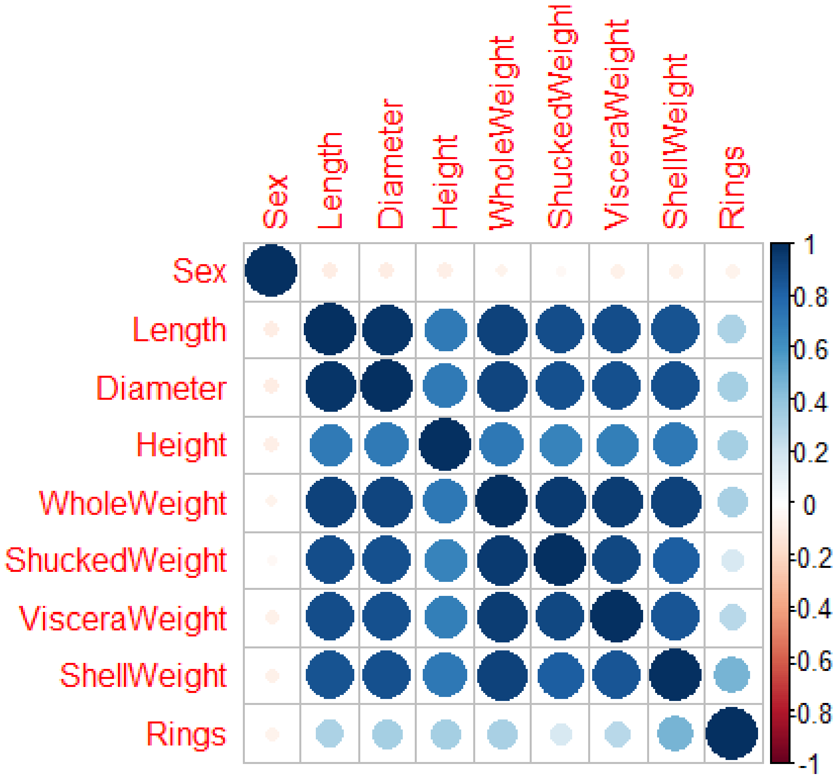
```r
prediction_aba <- predict(model, newdata = test_dataset, type = "response")


predicted_class <- ifelse(prediction_aba > 0.5, "M", "F")

predicted_class <- as.factor(predicted_class)

confusion_matrix <- confusionMatrix(predicted_class, test_dataset$Sex)
print(confusion_matrix)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   F   M
##          F  99  86
##          M 162 219
##
##                Accuracy : 0.5618
##                  95% CI : (0.5198, 0.6032)
##     No Information Rate : 0.5389
##     P-Value [Acc > NIR] : 0.1459
##
##                   Kappa : 0.0994
##
##  Mcnemar's Test P-Value : 1.912e-06
##
##             Sensitivity : 0.3793
##             Specificity : 0.7180
##          Pos Pred Value : 0.5351
##          Neg Pred Value : 0.5748
##              Prevalence : 0.4611
##          Detection Rate : 0.1749
##    Detection Prevalence : 0.3269
##       Balanced Accuracy : 0.5487
##
##        'Positive' Class : F
##

abalone_prediction <- prediction(prediction_aba, test_dataset$Sex)
abalone_evaluation <- performance(abalone_prediction, "acc")
plot(abalone_evaluation)
```




```
abalone_evaluation_roc <- performance(abalone_prediction, measure =  "tpr",
x.measure = "fpr")
plot(abalone_evaluation_roc, colorize = T, lwd = 2)
abline(a = 0, b = 1)
```




```r
accu <- performance(abalone_prediction, measure = "auc")
accuracy <- unlist(slot(accu, "y.values"))
print(accuracy)

## 0.5617235

matrix_abalone <- data.matrix(abalone_dataset)

mat = cor(matrix_abalone)
corrplot(mat)
```



From the above correlation plot, it is pretty evident and clear that all the data is positively correlated as blue indicates positive correlation and red indicates negative correlation.

```
#Problem 2
library(dplyr)
library(tidyr)
library(e1071)
library(caret)

mushroom_dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-
learning-databases/mushroom/agaricus-lepiota.data", header = FALSE)

col_names <- c("class", "cap_shape", "cap_surface", "cap_color", "bruises",
               "odor", "gill_attachment", "gill_spacing", "gill_size",
               "gill_color", "stalk_shape", "stalk_root",
"stalk_surface_above_ring",
               "stalk_surface_below_ring", "stalk_color_above_ring",
"stalk_color_below_ring",
               "veil_type", "veil_color", "ring_number", "ring_type",
               "spore_print_color", "population", "habitat")

colnames(mushroom_dataset) <- col_names

summary(mushroom_dataset)

##     class             cap_shape          cap_surface          cap_color
##  Length:8124        Length:8124        Length:8124        Length:8124
```

```
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##    bruises              odor          gill_attachment    gill_spacing
##  Length:8124        Length:8124        Length:8124        Length:8124
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##   gill_size          gill_color         stalk_shape         stalk_root
##  Length:8124        Length:8124        Length:8124        Length:8124
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##  stalk_surface_above_ring stalk_surface_below_ring stalk_color_above_ring
##  Length:8124              Length:8124              Length:8124
##  Class :character         Class :character         Class :character
##  Mode  :character         Mode  :character         Mode  :character
##  stalk_color_below_ring  veil_type          veil_color
##  Length:8124            Length:8124        Length:8124
##  Class :character       Class :character   Class :character
##  Mode  :character       Mode  :character   Mode  :character
##  ring_number          ring_type          spore_print_color   population
##  Length:8124        Length:8124        Length:8124        Length:8124
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##    habitat
##  Length:8124
##  Class :character
##  Mode  :character
```

```
missing <- sum(mushroom_dataset=="?")
cat("Missing Values Count:", missing)
```

```
## Missing Values Count: 2480
```

```
mushroom_dataset.missing <- mutate(mushroom_dataset, stalk_root =
ifelse(stalk_root == "?",NA, stalk_root))
```

```
mushroom_dataset_new <- drop_na(mushroom_dataset.missing)
count(mushroom_dataset_new)
```

```
##      n
## 1 5644
```

```
NaiveBayes_classifier <- sample(1:nrow(mushroom_dataset_new),size =
0.8*nrow(mushroom_dataset_new))
train_data_mushroom <- mushroom_dataset[NaiveBayes_classifier,]
test_data_mushroom <- mushroom_dataset[-NaiveBayes_classifier,]
NaiveBayesModel <- naiveBayes(class ~., data = train_data_mushroom)
summary(NaiveBayesModel)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    22     -none- list
```

```
## levels     2     -none- character
## isnumeric 22     -none- logical
## call       4     -none- call
```

```
mushroom_testprediction <- predict(NaiveBayesModel, test_data_mushroom)
mushroom_trainprediction <- predict(NaiveBayesModel, train_data_mushroom)

confusion_matrix <- confusionMatrix(table(mushroom_trainprediction,
train_data_mushroom$class, dnn=c("Predicted","Actual")))
cat("Training Accuracy:",(confusion_matrix$overall['Accuracy']))
```

```
## Training Accuracy: 0.9559247
```

```
cat("False Positive (Training):", (confusion_matrix$table[1,2]+2))
```

```
## False Positive (Training): 194
```

```
confusion_matrix_1 <- confusionMatrix(table(mushroom_testprediction,
test_data_mushroom$class, dnn=c("Predicted","Actual")))
cat("Testing Accuracy:",(confusion_matrix_1$overall['Accuracy']))
```

```
## Testing Accuracy: 0.9271266
```

```
cat("False Positive (Testing):", (confusion_matrix_1$table[1,2]+12))
```

```
## False Positive (Testing): 151
```

```
#Problem 3
library(ggplot2)
yacht_dataset <- read.table("https://archive.ics.uci.edu/ml/machine-learning-
databases/00243/yacht_hydrodynamics.data", header = FALSE)
col_names <- c("Longitudinal_position", "Prismatic_coefficient",
"Length_displacement", "Beam_draught", "Length_beam_ratio", "Froude_number",
"Residuary_resistance")


colnames(yacht_dataset) <- col_names

summary(yacht_dataset)
```

```
##  Longitudinal_position Prismatic_coefficient Length_displacement
##  Min.   :-5.000        Min.   :0.5300        Min.   :4.340
##  1st Qu.:-2.400        1st Qu.:0.5460        1st Qu.:4.770
##  Median :-2.300        Median :0.5650        Median :4.780
##  Mean   :-2.382        Mean   :0.5641        Mean   :4.789
##  3rd Qu.:-2.300        3rd Qu.:0.5740        3rd Qu.:5.100
##  Max.   : 0.000        Max.   :0.6000        Max.   :5.140
##   Beam_draught   Length_beam_ratio Froude_number    Residuary_resistance
##  Min.   :2.810   Min.   :2.730     Min.   :0.1250   Min.   : 0.0100
##  1st Qu.:3.750   1st Qu.:3.150     1st Qu.:0.2000   1st Qu.: 0.7775
##  Median :3.955   Median :3.150     Median :0.2875   Median : 3.0650
##  Mean   :3.937   Mean   :3.207     Mean   :0.2875   Mean   :10.4954
```

```
##  3rd Qu.:4.170   3rd Qu.:3.510     3rd Qu.:0.3750   3rd Qu.:12.8150
##  Max.   :5.350   Max.   :3.640     Max.   :0.4500   Max.   :62.4200
```

```r
test_train_split_yacht_dataset <-
createDataPartition(yacht_dataset$Residuary_resistance, p = 0.8, list =
FALSE)
train_data_yacht <- yacht_dataset[test_train_split_yacht_dataset, ]
test_data_yacht <- yacht_dataset[-test_train_split_yacht_dataset, ]

summary(train_data_yacht)
```

```
##  Longitudinal_position Prismatic_coefficient Length_displacement
##  Min.   :-5.00         Min.   :0.5300        Min.   :4.34
##  1st Qu.:-2.40         1st Qu.:0.5460        1st Qu.:4.77
##  Median :-2.30         Median :0.5650        Median :4.78
##  Mean   :-2.34         Mean   :0.5648        Mean   :4.79
##  3rd Qu.:-2.30         3rd Qu.:0.5740        3rd Qu.:5.10
##  Max.   : 0.00         Max.   :0.6000        Max.   :5.14
##   Beam_draught   Length_beam_ratio Froude_number    Residuary_resistance
##  Min.   :2.810   Min.   :2.730     Min.   :0.1250   Min.   : 0.010
##  1st Qu.:3.750   1st Qu.:3.150     1st Qu.:0.2000   1st Qu.: 0.775
##  Median :3.960   Median :3.150     Median :0.2875   Median : 3.065
##  Mean   :3.945   Mean   :3.208     Mean   :0.2875   Mean   :10.473
##  3rd Qu.:4.170   3rd Qu.:3.510     3rd Qu.:0.3750   3rd Qu.:12.815
##  Max.   :5.350   Max.   :3.640     Max.   :0.4500   Max.   :62.420
```

```r
summary(test_data_yacht)
```

```
##  Longitudinal_position Prismatic_coefficient Length_displacement
##  Min.   :-5.000        Min.   :0.5300        Min.   :4.340
##  1st Qu.:-2.400        1st Qu.:0.5300        1st Qu.:4.760
##  Median :-2.300        Median :0.5650        Median :4.780
##  Mean   :-2.555        Mean   :0.5614        Mean   :4.782
##  3rd Qu.:-2.300        3rd Qu.:0.5740        3rd Qu.:5.100
##  Max.   : 0.000        Max.   :0.6000        Max.   :5.140
##   Beam_draught   Length_beam_ratio Froude_number    Residuary_resistance
##  Min.   :2.810   Min.   :2.730     Min.   :0.1250   Min.   : 0.030
##  1st Qu.:3.750   1st Qu.:3.150     1st Qu.:0.2188   1st Qu.: 0.950
##  Median :3.940   Median :3.150     Median :0.2875   Median : 3.515
##  Mean   :3.903   Mean   :3.204     Mean   :0.2875   Mean   :10.586
##  3rd Qu.:4.130   3rd Qu.:3.510     3rd Qu.:0.3750   3rd Qu.:12.965
##  Max.   :5.350   Max.   :3.640     Max.   :0.4500   Max.   :56.460
```

```r
yacht_model <- lm(Residuary_resistance ~., data = train_data_yacht)
summary(yacht_model)
```

```
##
## Call:
## lm(formula = Residuary_resistance ~ ., data = train_data_yacht)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -12.063  -7.481  -1.715   5.861  31.772
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             -19.1436    30.0602  -0.637    0.525
## Longitudinal_position     0.2671     0.3772   0.708    0.480
## Prismatic_coefficient   -13.6935    48.5695  -0.282    0.778
## Length_displacement      -6.6548    15.5688  -0.427    0.669
## Beam_draught              2.5501     6.0565   0.421    0.674
## Length_beam_ratio         7.5970    15.6344   0.486    0.627
## Froude_number           123.2192     5.6746  21.714   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.94 on 241 degrees of freedom
## Multiple R-squared:  0.6624, Adjusted R-squared:  0.654
## F-statistic: 78.81 on 6 and 241 DF,  p-value: < 2.2e-16
```

```
yacht_trainprediction <- predict(yacht_model,train_data_yacht)
yacht_testprediction <- predict(yacht_model, test_data_yacht)

yacht_train_MSE <- mean((train_data_yacht$Residuary_resistance -
yacht_trainprediction)^2)
yacht_train_RMSE <- sqrt(yacht_train_MSE)
yacht_train_RSS <- sum((train_data_yacht$Residuary_resistance -
yacht_trainprediction)^2)
yacht_train_TSS <- sum((train_data_yacht$Residuary_resistance -
mean(train_data_yacht$Residuary_resistance))^2)
yacht_train_RSQUARE <- 1-(yacht_train_RSS/yacht_train_TSS)

cat("MSE (Training):",yacht_train_MSE)
```

```
## MSE (Training): 77.6618
```

```
cat("RMSE (Training):",yacht_train_RMSE)
```

```
## RMSE (Training): 8.812593
```

```
cat("RSQUARE (Training) :",yacht_train_RSQUARE)
```

```
## RSQUARE (Training) : 0.6623862
```

```
train_control <- trainControl(method = "boot", number = 1000, p = 0.8)
model_bootstrap <- train(Residuary_resistance ~ ., data = train_data_yacht,
trControl = train_control, method = "lm")

head(model_bootstrap$resample)
```

```
##       RMSE  Rsquared      MAE     Resample
## 1 9.024375 0.6584386 7.217408 Resample0001
## 2 8.802296 0.6790368 7.120401 Resample0002
```

```
## 3 8.962328 0.6280181 7.443236 Resample0003
## 4 8.997288 0.6095808 7.600265 Resample0004
## 5 9.147049 0.6614838 7.219529 Resample0005
## 6 9.027975 0.6290831 7.653388 Resample0006
```

```r
mean(yacht_train_RMSE)
```

```
## [1] 8.812593
```

```r
mean(yacht_train_RSQUARE)
```

```
## [1] 0.6623862
```

```r
class(train_data_yacht$Residuary_resistance)
```

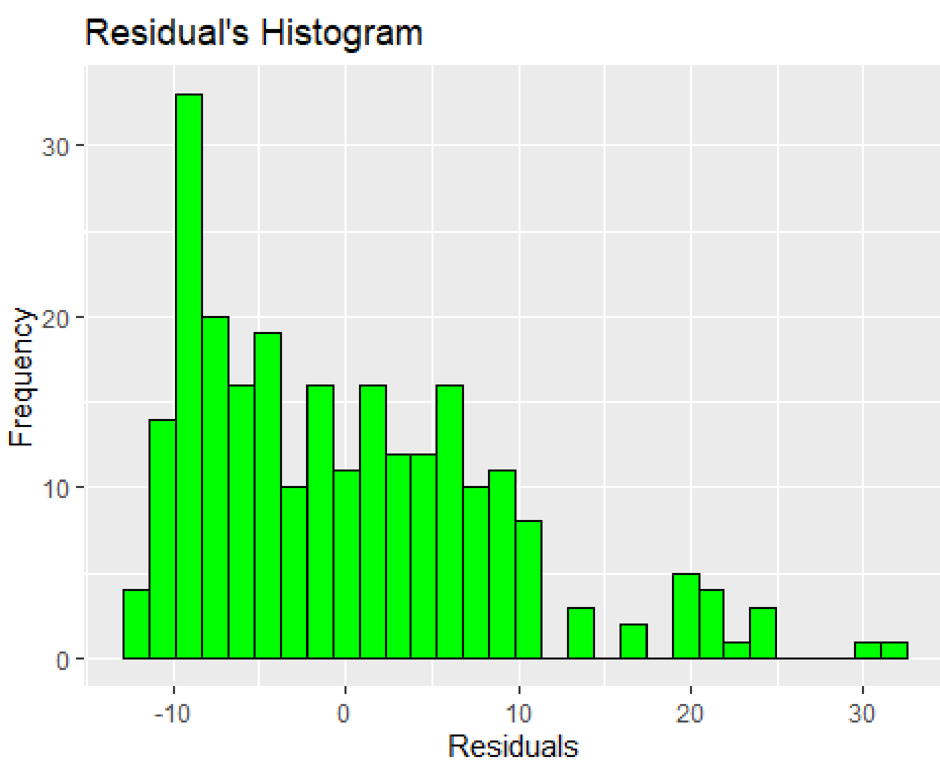
```
## [1] "numeric"
```

```r
yacht_plot = data.frame(train_data_yacht$Residuary_resistance)
class(yacht_plot)
```

```
## [1] "data.frame"
```

```r
ggplot(data = yacht_plot, aes(x= yacht_model$residuals)) +
geom_histogram(fill = 'green',color = 'black') +labs(title = "Residual's
Histogram", x = 'Residuals', y = 'Frequency')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
yacht_test_prediction <- predict(model_bootstrap, test_data_yacht)
yacht_test_MSE <- mean((test_data_yacht$Residuary_resistance -
yacht_test_prediction)^2)
yacht_test_RMSE <- sqrt(yacht_test_MSE)
yacht_test_RSS <- sum((test_data_yacht$Residuary_resistance -
yacht_test_prediction)^2)
yacht_test_TSS <- sum((test_data_yacht$Residuary_resistance -
mean(test_data_yacht$Residuary_resistance))^2)
yacht_test_RSQUARE <- 1-(yacht_test_RSS / yacht_test_TSS)

cat("MSE for Bootstrap (Testing)",yacht_test_MSE)

## MSE for Bootstrap (Testing) 84.47497

cat("RMSE for Bootstrap (Testing)",yacht_test_RMSE)

## RMSE for Bootstrap (Testing) 9.191027

cat("R SQUARED for Bootstrap (Testing)",yacht_test_RSQUARE)

## R SQUARED for Bootstrap (Testing) 0.6249056
```

From these values we understand that even for test set, the values are identical for both basic and bootstrap model

```r
#Problem 4
german_credit_dataset <- read.table("https://archive.ics.uci.edu/ml/machine-
learning-databases/statlog/german/german.data-numeric", header = FALSE)

head(german_credit_dataset)

##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
V21
## 1  1  6  4 12  5  5  3  4  1  67   3   2   1   2   1   0   0   1   0   0
1
## 2  2 48  2 60  1  3  2  2  1  22   3   1   1   1   1   0   0   1   0   0
1
## 3  4 12  4 21  1  4  3  3  1  49   3   1   2   1   1   0   0   1   0   0
1
## 4  1 42  2 79  1  4  3  4  2  45   3   1   2   1   1   0   0   0   0   0
0
## 5  1 24  3 49  1  3  3  4  4  53   3   2   2   1   1   1   0   1   0   0
0
## 6  4 36  2 91  5  3  3  4  4  35   3   1   2   2   1   0   0   1   0   0
0
##   V22 V23 V24 V25
## 1   0   0   1   1
## 2   0   0   1   2
## 3   0   1   0   1
## 4   0   0   1   1
## 5   0   0   1   2
## 6   0   1   0   1
```

```r
summary(german_credit_dataset)
```

```
##        V1              V2              V3              V4        
##  Min.   :1.000   Min.   : 4.0   Min.   :0.000   Min.   :  2.00  
##  1st Qu.:1.000   1st Qu.:12.0   1st Qu.:2.000   1st Qu.: 14.00  
##  Median :2.000   Median :18.0   Median :2.000   Median : 23.00  
##  Mean   :2.577   Mean   :20.9   Mean   :2.545   Mean   : 32.71  
##  3rd Qu.:4.000   3rd Qu.:24.0   3rd Qu.:4.000   3rd Qu.: 40.00  
##  Max.   :4.000   Max.   :72.0   Max.   :4.000   Max.   :184.00  
##        V5              V6              V7              V8       
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000  
##  1st Qu.:1.000   1st Qu.:3.000   1st Qu.:2.000   1st Qu.:2.000  
##  Median :1.000   Median :3.000   Median :3.000   Median :3.000  
##  Mean   :2.105   Mean   :3.384   Mean   :2.682   Mean   :2.845  
##  3rd Qu.:3.000   3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:4.000  
##  Max.   :5.000   Max.   :5.000   Max.   :4.000   Max.   :4.000  
##        V9             V10             V11             V12       
##  Min.   :1.000   Min.   :19.00   Min.   :1.000   Min.   :1.000  
##  1st Qu.:1.000   1st Qu.:27.00   1st Qu.:3.000   1st Qu.:1.000  
##  Median :2.000   Median :33.00   Median :3.000   Median :1.000  
##  Mean   :2.358   Mean   :35.55   Mean   :2.675   Mean   :1.407  
##  3rd Qu.:3.000   3rd Qu.:42.00   3rd Qu.:3.000   3rd Qu.:2.000  
##  Max.   :4.000   Max.   :75.00   Max.   :3.000   Max.   :4.000  
##       V13             V14             V15             V16       
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :0.000  
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:0.000  
##  Median :1.000   Median :1.000   Median :1.000   Median :0.000  
##  Mean   :1.155   Mean   :1.404   Mean   :1.037   Mean   :0.234  
##  3rd Qu.:1.000   3rd Qu.:2.000   3rd Qu.:1.000   3rd Qu.:0.000  
##  Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :1.000  
##       V17             V18             V19             V20       
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000  
##  1st Qu.:0.000   1st Qu.:1.000   1st Qu.:0.000   1st Qu.:0.000  
##  Median :0.000   Median :1.000   Median :0.000   Median :0.000  
##  Mean   :0.103   Mean   :0.907   Mean   :0.041   Mean   :0.179  
##  3rd Qu.:0.000   3rd Qu.:1.000   3rd Qu.:0.000   3rd Qu.:0.000  
##  Max.   :1.000   Max.   :1.000   Max.   :1.000   Max.   :1.000  
##       V21             V22             V23           V24            V25     
##  Min.   :0.000   Min.   :0.000   Min.   :0.0   Min.   :0.00   Min.   :1.0  
##  1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.0   1st Qu.:0.00   1st Qu.:1.0  
##  Median :1.000   Median :0.000   Median :0.0   Median :1.00   Median :1.0  
##  Mean   :0.713   Mean   :0.022   Mean   :0.2   Mean   :0.63   Mean   :1.3  
##  3rd Qu.:1.000   3rd Qu.:0.000   3rd Qu.:0.0   3rd Qu.:1.00   3rd Qu.:2.0  
##  Max.   :1.000   Max.   :1.000   Max.   :1.0   Max.   :1.00   Max.   :2.0
```

```r
german_credit_dataset$V25 <- factor(german_credit_dataset$V25)

train_test_split_german_credit_dataset <- 
createDataPartition(german_credit_dataset$V25, p = 0.8, list = FALSE)
train_data_german <- 
```

```r
german_credit_dataset[train_test_split_german_credit_dataset,]
test_data_german <- german_credit_dataset[-
train_test_split_german_credit_dataset,]

german_model <- glm(V25~., data = train_data_german, family = binomial)
summary(german_model)

##
## Call:
## glm(formula = V25 ~ ., family = binomial, data = train_data_german)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.143123   1.352925   2.323 0.020168 *
## V1          -0.580370   0.080550  -7.205 5.80e-13 ***
## V2           0.036455   0.009856   3.699 0.000217 ***
## V3          -0.439206   0.100102  -4.388 1.15e-05 ***
## V4           0.005003   0.004289   1.166 0.243416
## V5          -0.255962   0.069694  -3.673 0.000240 ***
## V6          -0.149409   0.085867  -1.740 0.081859 .
## V7          -0.157743   0.129548  -1.218 0.223362
## V8           0.007115   0.093350   0.076 0.939243
## V9           0.235371   0.112080   2.100 0.035726 *
## V10         -0.003597   0.009712  -0.370 0.711085
## V11         -0.340186   0.127370  -2.671 0.007566 **
## V12          0.209521   0.185651   1.129 0.259079
## V13         -0.074443   0.265942  -0.280 0.779538
## V14         -0.345031   0.220387  -1.566 0.117450
## V15         -1.643710   0.715779  -2.296 0.021653 *
## V16          0.637823   0.220679   2.890 0.003849 **
## V17         -0.809050   0.366027  -2.210 0.027080 *
## V18          1.297384   0.491582   2.639 0.008310 **
## V19          1.634117   0.663843   2.462 0.013832 *
## V20          0.337315   0.406088   0.831 0.406175
## V21         -0.057662   0.358405  -0.161 0.872184
## V22         -0.683340   0.755023  -0.905 0.365434
## V23         -0.059417   0.365757  -0.162 0.870951
## V24         -0.089765   0.299926  -0.299 0.764717
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 977.38  on 799  degrees of freedom
## Residual deviance: 732.98  on 775  degrees of freedom
## AIC: 782.98
##
## Number of Fisher Scoring iterations: 5
```

```
german_trainprediction <- predict(german_model,train_data_german)
german_testprediction <- predict(german_model, test_data_german)

german_train_temp <- ifelse(german_trainprediction>= 0.5, 2, 1)
german_train_temp <- as.factor(german_train_temp)
german_test_temp <- ifelse(german_testprediction>=0.5, 2, 1)
german_test_temp <- as.factor(german_test_temp)
confusionMatrix(table(german_train_temp, train_data_german$V25, dnn =
c("Predicted", "Actual") ))

## Confusion Matrix and Statistics
##
##          Actual
## Predicted   1   2
##         1 530 154
##         2  30  86
##
##                Accuracy : 0.77
##                  95% CI : (0.7392, 0.7987)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 5.783e-06
##
##                   Kappa : 0.3575
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9464
##             Specificity : 0.3583
##          Pos Pred Value : 0.7749
##          Neg Pred Value : 0.7414
##              Prevalence : 0.7000
##          Detection Rate : 0.6625
##    Detection Prevalence : 0.8550
##       Balanced Accuracy : 0.6524
##
##        'Positive' Class : 1
##

german_credit_dataset_precision <- posPredValue(german_train_temp,
train_data_german$V25, positive = "1")
german_credit_dataset_recall <- sensitivity(german_train_temp,
train_data_german$V25, positive = "1")
F1 <- (2 * german_credit_dataset_precision * german_credit_dataset_recall) /
(german_credit_dataset_precision + german_credit_dataset_recall)
cat("Precision (Training):",german_credit_dataset_precision)

## Precision (Training): 0.7748538

cat("Recall (Training):",german_credit_dataset_recall)

## Recall (Training): 0.9464286
```

```r
cat("F1 (Training):",F1)

## F1 (Training): 0.85209

confusionMatrix(table(german_test_temp, test_data_german$V25, dnn =
c("Predicted", "Actual") ))

## Confusion Matrix and Statistics
##
##          Actual
## Predicted   1   2
##         1 128  43
##         2  12  17
##
##                Accuracy : 0.725
##                  95% CI : (0.6576, 0.7856)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.2455
##
##                   Kappa : 0.2318
##
##  Mcnemar's Test P-Value : 5.228e-05
##
##             Sensitivity : 0.9143
##             Specificity : 0.2833
##          Pos Pred Value : 0.7485
##          Neg Pred Value : 0.5862
##              Prevalence : 0.7000
##          Detection Rate : 0.6400
##    Detection Prevalence : 0.8550
##       Balanced Accuracy : 0.5988
##
##        'Positive' Class : 1
##

german_credit_dataset_precision <- posPredValue(german_test_temp,
test_data_german$V25, positive = "1")
german_credit_dataset_recall <- sensitivity(german_test_temp,
test_data_german$V25, positive = "1")
F1 <- (2 * german_credit_dataset_precision * german_credit_dataset_recall) /
(german_credit_dataset_precision + german_credit_dataset_recall)
cat("Precision (Testing):",german_credit_dataset_precision)

## Precision (Testing): 0.748538

cat("Recall (Testing):",german_credit_dataset_recall)

## Recall (Testing): 0.9142857

cat("F1 (Testing):",F1)

## F1 (Testing): 0.8231511
```

```
control <- trainControl(method = "cv", number = 10)

crossvalidation_model <-train(factor(V25)~., data = train_data_german,
trControl = control, method = "glm")
crossvalidation_model <- crossvalidation_model$finalModel
german_train_prediction <- predict(crossvalidation_model, train_data_german)
german_test_prediction <- predict(crossvalidation_model, test_data_german)
crossvalidation_model_train_temp <- ifelse(german_train_prediction>=0.5,2,1)
crossvalidation_model_train_temp <-
as.factor(crossvalidation_model_train_temp)
crossvalidation_model_test_temp <- ifelse(german_test_prediction>=0.5,2,1)
crossvalidation_model_test_temp <- as.factor(crossvalidation_model_test_temp)

confusionMatrix(table(crossvalidation_model_train_temp,
train_data_german$V25, dnn = c("Predicted", "Actual")))

## Confusion Matrix and Statistics
##
##          Actual
## Predicted   1   2
##         1 530 154
##         2  30  86
##
##                Accuracy : 0.77
##                  95% CI : (0.7392, 0.7987)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 5.783e-06
##
##                   Kappa : 0.3575
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9464
##             Specificity : 0.3583
##          Pos Pred Value : 0.7749
##          Neg Pred Value : 0.7414
##              Prevalence : 0.7000
##          Detection Rate : 0.6625
##    Detection Prevalence : 0.8550
##       Balanced Accuracy : 0.6524
##
##        'Positive' Class : 1
##

german_credit_dataset_precision <-
posPredValue(crossvalidation_model_train_temp, train_data_german$V25,
positive = "1")
german_credit_dataset_recall<- sensitivity(crossvalidation_model_train_temp,
train_data_german$V25, positive = "1")
```

```
F1 <- (2 * german_credit_dataset_precision * german_credit_dataset_recall) /
(german_credit_dataset_precision + german_credit_dataset_recall)

cat("Precison (Training) After:", german_credit_dataset_precision)

## Precison (Training) After: 0.7748538

cat("Recall (Training) After:",german_credit_dataset_recall)

## Recall (Training) After: 0.9464286

cat("F1 (Training) After:", F1)

## F1 (Training) After: 0.85209

confusionMatrix(table(crossvalidation_model_test_temp, test_data_german$V25,
dnn = c("Predicted", "Actual")))

## Confusion Matrix and Statistics
##
##          Actual
## Predicted   1   2
##         1 128  43
##         2  12  17
##
##                Accuracy : 0.725
##                  95% CI : (0.6576, 0.7856)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.2455
##
##                   Kappa : 0.2318
##
##  Mcnemar's Test P-Value : 5.228e-05
##
##             Sensitivity : 0.9143
##             Specificity : 0.2833
##          Pos Pred Value : 0.7485
##          Neg Pred Value : 0.5862
##              Prevalence : 0.7000
##          Detection Rate : 0.6400
##    Detection Prevalence : 0.8550
##       Balanced Accuracy : 0.5988
##
##        'Positive' Class : 1
##

german_credit_dataset_precision <-
posPredValue(crossvalidation_model_test_temp, test_data_german$V25, positive
= "1")
german_credit_dataset_recall <- sensitivity(crossvalidation_model_test_temp,
test_data_german$V25, positive = "1")
```

```r
F1 <- (2 * german_credit_dataset_precision * german_credit_dataset_recall) /
(german_credit_dataset_precision + german_credit_dataset_recall)

cat("Precision (Testing) After:",german_credit_dataset_precision)

## Precision (Testing) After: 0.748538

cat("Recall (Testing) After:",german_credit_dataset_recall)

## Recall (Testing) After: 0.9142857

cat("F1 (Testing) After:",F1)

## F1 (Testing) After: 0.8231511
```

Upon examination of all the values produced above, we can see that the values produced by the CV and basic models are nearly indistinguishable for both the training and test datasets.