DPA Assignment 2

Girish

2023-02-19

#Recitation Problems

#Chapter 4 #4a

Since we are assuming that X is uniformly (evenly) distributed on $[0,1]$ and we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation, we can say that:

$X \in [0.05, 0.95]$ which means the intervals will be $[X - 0.05, X + 0.05]$ and the length will be 0.1 (10%).

$\int_{0.05}^{0.95} 10 \, dx = 10\%$

Therefore, the fraction of the available observations that will be used to make the prediction will be 10%

#4b

Since we assume that (X_1, X_2) are uniformly distributed on $[0,1] * [0,1]$ each with measurements on $p = 2$ features, the fraction of available observations that will be used to make the prediction will be the product of the two observations using the fraction from (a) above:

the length will be $0.1 * 0.1 = 0.01 = 1\%$

Therefore, the fraction of the available observations that will be used to make the prediction will be 1%

#4c

Since we have a set of observations on $p = 100$ features and the observations are again uniformly distributed on each feature where each feature ranges in value from 0 to 1, we can say that:

$0.1^p * 100 = 0.1^{100} * 100$ is the fraction of the available observations that will be used to make the prediction.

#4d

Answers to part (a)-(c): - When $p = 1$, the fraction of the available observations used to make the prediction was 0.1

- When $p = 2$, the fraction of the available observations used to make the prediction was 0.01

- When $p = 100$, the fraction of the available observations used to make the prediction was $0.1^{100} * 100$ which is significantly smaller

$\lim_{x \to \infty} (10\%)^p = 0$

From the above, we can conclude that a drawback of KNN when p is large, there are very few training observations "near" any given test observation.

#4e

- For $p = 1$, the length of each side is $(0.1)^{1/1} = 0.1$
- For $p = 2$, the length of each side is $(0.1)^{1/2} = 0.32$
- For $p = 100$, the length of each side is $(0.1)^{1/100} = 0.98$

Comment:

From the above, we can say that as p increases, the length of each side gets closer to 1.

#6a

$$P(X) \quad = \frac{\exp\left(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2\right)}{1 + \exp\left(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2\right)}$$

$$P(X) \quad = \frac{\exp\left(-6 + (0.05 * 40) + (1 * 3.5)\right)}{1 + \exp\left(-6 + (0.05 * 40) + (1 * 3.5)\right)}$$

$$P(X) \quad = \frac{\exp(-0.5)}{1 + \exp(-0.5)} = 0.38$$

#6b

$$\log\left(\frac{P(X)}{1 - P(X)}\right) \quad = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2$$

$$\log\left(\frac{0.5}{1 - 0.5}\right) \quad = -6 + (0.05 * X_1) + (1 * 3.5)$$

By transposing for X_1, we get:

$$X_1 = \frac{6 - 3.5}{0.05}$$

$X_1 = 50$ hours.

#7

$$P(Y = yes|X = 4) = \frac{\pi_{yes} f_{yes}(x)}{\sum_{l=1}^{K} \pi_l \, f_l(x)}$$

$$P(Y = yes|X = 4) = \frac{\pi_{yes} \exp\left(-1/2\sigma^2\left(x - \mu_{yes}\right)^2\right)}{\sum_{l=1}^{K} \pi_l \exp(-1/2\sigma^2(x - \mu_l)^2)}$$

$$P(Y = yes|X = 4) = \frac{0.8 \times \exp(-0.5)}{0.8 \times \exp(-0.5) + 0.2 \times \exp(-16/72)}$$

$$P(Y = yes|X = 4) = \frac{0.485}{0.645} = 0.75$$

Assuming that X follows a normal distribution, the probability that a company will issue a dividend this year given that its percentage profit was $X = 4$ last year is 75%

#9a

$$Odds = \frac{P(X)}{1 - P(X)}$$

$$0.37 = \frac{P(X)}{1 - P(X)}$$

Transpose and factorize to make $P(X)$ the subject:

$$0.37\left(1 - P(X)\right) = P(X)$$
$$0.37 - 0.37P(X) = P(X)$$
$$0.37 = P(X) + 0.37P(X)$$
$$Factorize P(X):$$
$$0.37 = P(X)(1 + 0.37)$$
$$P(X)(1.37) = 0.37$$
$$P(X) = \frac{0.37}{1.37} = 0.27$$

#9b

$$Odds = \frac{0.16}{1 - 0.16} = 0.19$$

#Chapter 5 #2a

Since the bootstrap sample that is generated has an equal probability, the probability that the first bootstrap observation is the jth observation from the original sample is $\frac{1}{n}$.

Therefore, the probability that the first bootstrap observation is *not* the jth observation from the original sample is $1 - \frac{1}{n}$

#2b

Since bootstrapping has an equal probability of random sampling, the second bootstrap sample does not depend on the first bootstrap sample.

Therefore, the probability that the second bootstrap observation is not the jth observation from the original sample is also $ 1-$

#2c

From the previous question, the probability that a bootstrap observation is not the jth observation from the original sample is also $ 1-$, which means that the probability that the jth observation is not in the bootstrap sample is the product of all bootstrap observations not in the sample which is $\left(1-\frac{1}{n}\right)^n$

#2d

The probability that the jth observation is in the bootstrap sample is $1-(1-1/n)^n$

$$p = 1-(1-1/5)^5 = 0.67$$

#2e

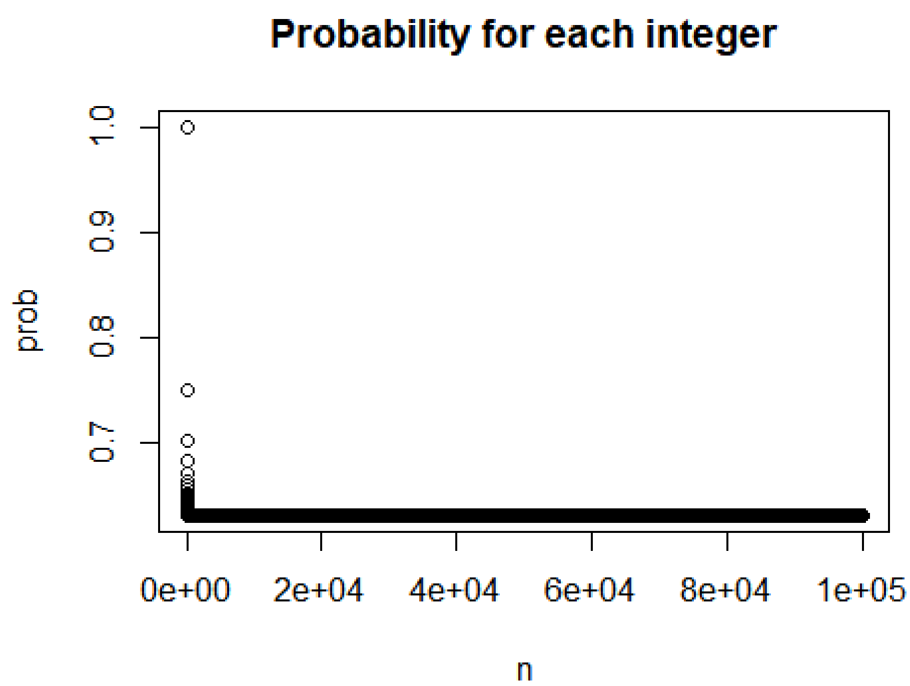
$$p = 1-(1-1/100)^{100} = 0.634$$

#2f

$$p = 1-(1-1/10000)^{10000} = 0.632$$

#2g

```
n <- 1:100000

#The probability that the jth observation is in the bootstrap sample
prob <- 1-(1-1/n)^n

plot(n,prob, main = "Probability for each integer")
```



#From above we can observe that initially, the probability significantly decreases to 0.63 (also observed in 2e and 2f) then remains the same throughout.

#2h

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

## [1] 0.6358
```

From above, we can observe that the probability is very similar to that obtained in 2e and 2f above

#3a

k-fold cross validation involves dividing the data in K subsets of equal size. We then use K-1 folds for training and the first fold is used as the validation set. The MSE of this group is computed as per normal. This process is repeated for K number of folds where the fold used for the validation set changes each time. At the end, we will have an MSE for each k estimates and the final k-fold cross validation MSE is computed by averaging the results.

#3bi

The advantage of k-fold cross-validation compared to the validation set is that it has lower variance on the test error. Secondly, k-fold cross validation has higher accuracy, efficiency and prevents overestimating the test error since at some point, we are using the entire data set when compared to validation set which only uses a subset.

The disadvantage of k-fold cross-validation compared to the validation set is that is computationally more expensive and difficult to implement because k-fold cv has to rerun training k times whereas just using a validation set will only have to train once on that set.

#3bii

The advantage of k-fold cross-validation compared to LOOCV is that since LOOCV has to fit the method n times which is more than k-fold CV k times, k-fold cross validation is computationally less expensive. Secondly, K-fold cross validation gives a more accurate result on test error when compared to LOOCV.

The disadvantage of k-fold cross-validation compred to LOOCV is that when perform bias reduction, we would prefer to use LOOCV since it has a lower bias than k-fold cross validation.

```
#Practicum Problems
#Problem 1
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(corrplot)

## corrplot 0.92 loaded

library(ROCR)

#Load data from UCI repository
abalone_data <- read.csv(file="https://archive.ics.uci.edu/ml/machine-
learning-databases/abalone/abalone.data", col.names= c ("Sex", "Length",
"Diameter", "Height", "Whole weight", "Shucked weight", "Viscera weight",
"Shell weight", "Rings"))

#Remove all observations in the Infant Category
abalone_data <- abalone_data[!abalone_data$Sex == 'I',]

#Using str(abalone_data), we see that Sex data type is char so we change it
to factor
#Turn Sex feature into a factor from char
abalone_data$Sex <- factor(abalone_data$Sex)

#Showing data types, summary and first 6 rows of dataset
str(abalone_data)

## 'data.frame':    2834 obs. of  9 variables:
##  $ Sex           : Factor w/ 2 levels "F","M": 2 1 2 1 1 2 1 1 2 2 ...
```

```
##  $ Length        : num  0.35 0.53 0.44 0.53 0.545 0.475 0.55 0.525 0.43
0.49 ...
##  $ Diameter      : num  0.265 0.42 0.365 0.415 0.425 0.37 0.44 0.38 0.35
0.38 ...
##  $ Height        : num  0.09 0.135 0.125 0.15 0.125 0.125 0.15 0.14 0.11
0.135 ...
##  $ Whole.weight  : num  0.226 0.677 0.516 0.777 0.768 ...
##  $ Shucked.weight: num  0.0995 0.2565 0.2155 0.237 0.294 ...
##  $ Viscera.weight: num  0.0485 0.1415 0.114 0.1415 0.1495 ...
##  $ Shell.weight  : num  0.07 0.21 0.155 0.33 0.26 0.165 0.32 0.21 0.135
0.19 ...
##  $ Rings         : int  7 9 10 20 16 9 19 14 10 11 ...
```

```
summary(abalone_data)
```

```
##  Sex          Length          Diameter          Height        Whole.weight
##  F:1307   Min.   :0.1550   Min.   :0.1100   Min.   :0.0150   Min.
:0.0155
##  M:1527   1st Qu.:0.5150   1st Qu.:0.4000   1st Qu.:0.1350   1st
Qu.:0.7020
##            Median :0.5850   Median :0.4600   Median :0.1550   Median
:1.0032
##            Mean   :0.5696   Mean   :0.4464   Mean   :0.1545   Mean
:1.0170
##            3rd Qu.:0.6350   3rd Qu.:0.5000   3rd Qu.:0.1750   3rd
Qu.:1.2895
##            Max.   :0.8150   Max.   :0.6500   Max.   :1.1300   Max.
:2.8255
##  Shucked.weight   Viscera.weight    Shell.weight        Rings
##  Min.   :0.0065   Min.   :0.0030   Min.   :0.0050   Min.   : 3.0
##  1st Qu.:0.2875   1st Qu.:0.1521   1st Qu.:0.2030   1st Qu.: 9.0
##  Median :0.4315   Median :0.2170   Median :0.2850   Median :10.0
##  Mean   :0.4391   Mean   :0.2226   Mean   :0.2913   Mean   :10.9
##  3rd Qu.:0.5689   3rd Qu.:0.2875   3rd Qu.:0.3650   3rd Qu.:12.0
##  Max.   :1.4880   Max.   :0.7600   Max.   :1.0050   Max.   :29.0
```

```
head(abalone_data)
```

```
##   Sex Length Diameter Height Whole.weight Shucked.weight Viscera.weight
## 1   M  0.350    0.265  0.090       0.2255         0.0995         0.0485
## 2   F  0.530    0.420  0.135       0.6770         0.2565         0.1415
## 3   M  0.440    0.365  0.125       0.5160         0.2155         0.1140
## 6   F  0.530    0.415  0.150       0.7775         0.2370         0.1415
## 7   F  0.545    0.425  0.125       0.7680         0.2940         0.1495
## 8   M  0.475    0.370  0.125       0.5095         0.2165         0.1125
##   Shell.weight Rings
## 1        0.070     7
## 2        0.210     9
## 3        0.155    10
## 6        0.330    20
```

```
## 7        0.260    16
## 8        0.165     9

#Using createDataPartition to perform 80/20 train-test split

datasetPartition <- createDataPartition(abalone_data$Sex, p = 0.8, list =
FALSE, times = 1)

train <- abalone_data[datasetPartition,]
test <- abalone_data[-datasetPartition,]

dim(train)

## [1] 2268    9

dim(test)

## [1] 566   9

#Using glm to fit a logistic regression
set.seed(10)
glm.fits <- glm(Sex ~ Length + Diameter + Height + Whole.weight +
Shucked.weight + Viscera.weight + Shell.weight + Rings, data = train, family
= binomial)

summary(glm.fits)

##
## Call:
## glm(formula = Sex ~ Length + Diameter + Height + Whole.weight +
##     Shucked.weight + Viscera.weight + Shell.weight + Rings, family =
binomial,
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8557  -1.1989   0.8742   1.1190   1.5466
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     2.907446   0.516463   5.630 1.81e-08 ***
## Length         -2.662974   2.275107  -1.170 0.241807
## Diameter       -4.018240   2.709850  -1.483 0.138120
## Height         -2.844661   2.550687  -1.115 0.264742
## Whole.weight   -0.646142   0.869493  -0.743 0.457406
## Shucked.weight  3.730786   1.020016   3.658 0.000255 ***
## Viscera.weight -1.448495   1.453562  -0.997 0.319000
## Shell.weight    1.289343   1.280721   1.007 0.314063
## Rings          -0.002758   0.017825  -0.155 0.877044
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3130.4  on 2267  degrees of freedom
## Residual deviance: 3067.9  on 2259  degrees of freedom
## AIC: 3085.9
##
## Number of Fisher Scoring iterations: 4
```

From above, we can see that the predictors that are relevant, have a lower p-value. Such predictors include Diameter, Shucked.weight, and Viscera.weight where Shucked.weight is the most significant because it has the lowest p-value.

This is also an indicator that since Shucked.weight and Viscera.weight have a low p-value, they are more likely to reject the null hypothesis which means there is a relationship between the predictors Shucked.weight and Viscera.weight with the response Sex.

```
#Using coef to show the coefficients of the fitted model
summary(glm.fits)$coef
```

```
##                    Estimate Std. Error    z value     Pr(>|z|)
## (Intercept)     2.907445565 0.51646272  5.6295361 1.806949e-08
## Length         -2.662974274 2.27510658 -1.1704833 2.418065e-01
## Diameter       -4.018240089 2.70984961 -1.4828277 1.381202e-01
## Height         -2.844661272 2.55068696 -1.1152530 2.647420e-01
## Whole.weight   -0.646141990 0.86949349 -0.7431246 4.574063e-01
## Shucked.weight  3.730786165 1.02001588  3.6575765 2.546112e-04
## Viscera.weight -1.448494913 1.45356179 -0.9965142 3.190004e-01
## Shell.weight    1.289342828 1.28072064  1.0067323 3.140634e-01
## Rings          -0.002757779 0.01782465 -0.1547172 8.770443e-01
```

```
#Confidence Intervals for the predictors
confint(glm.fits)
```

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

```
##                      2.5 %     97.5 %
## (Intercept)     1.91013538 3.93591057
## Length         -7.12891689 1.79541018
## Diameter       -9.34167231 1.28875273
## Height         -7.89115778 2.17930952
## Whole.weight   -2.36949490 1.04996519
## Shucked.weight  1.74768709 5.75444628
## Viscera.weight -4.29925157 1.40544830
## Shell.weight   -1.21682839 3.81536364
## Rings          -0.03772793 0.03219775
```

The confidence intervals for all predictors contain 0 within their range except for Shucked.weight and Viscera.weight. This also means that both of these predictors cannot accept the null hypothesis. Since all other predictors accept the null hypothesis, there is no relationship between those predictors and the response.

```r
#Use predict() function to perform prediction on test data
glm.predict <- predict(glm.fits, test, type = "response")

glm.predict
```

```
##         6        11        13        47        54        63        66 
68 
## 0.4872495 0.6070189 0.5015728 0.5532243 0.6649579 0.6421700 0.5020961 
0.5528305 
##        71        73        88        91        99       107       117 
120 
## 0.6233493 0.5118106 0.5981042 0.4933815 0.6202851 0.5550548 0.5853553 
0.5579101 
##       122       123       135       146       153       158       162 
170 
## 0.5273555 0.6987373 0.6237490 0.6119304 0.4204599 0.4706156 0.5108283 
0.3935343 
##       180       182       192       194       197       200       219 
221 
## 0.4706183 0.5157537 0.4307944 0.6613113 0.4152241 0.5845345 0.6203424 
0.5555506 
##       229       231       253       259       274       275       276 
286 
## 0.5020176 0.4529626 0.5142696 0.5041550 0.4121405 0.5449545 0.4269543 
0.4839375 
##       292       300       301       307       312       324       327 
331 
## 0.4566251 0.6514134 0.4981711 0.3358909 0.4300639 0.7012991 0.5383729 
0.6242632 
##       336       338       340       351       358       365       369 
371 
## 0.4151273 0.5348107 0.6385215 0.4471333 0.4937728 0.4190108 0.4403854 
0.4741861 
##       373       374       384       388       392       406       411 
435 
## 0.3433863 0.3741393 0.5452492 0.5236547 0.5931094 0.6127194 0.5086618 
0.5446624 
##       438       446       451       483       488       491       493 
505 
## 0.6758679 0.4979765 0.4601103 0.5052134 0.5470662 0.4893513 0.4120513 
0.5078271 
##       508       516       517       518       543       544       545 
558 
## 0.4803542 0.6396415 0.7664431 0.7287656 0.5908088 0.6554242 0.7217283 
0.4420975 
##       560       562       568       573       589       593       599 
616 
## 0.5988410 0.5727396 0.7613433 0.5186159 0.6363180 0.3990781 0.4109588 
0.5363271 
##       622       623       626       632       636       655       665
```

```
670
## 0.5940046 0.5229438 0.6097026 0.5755980 0.7108016 0.7571877 0.6325659
0.5604378
##       672       676       685       686       697       699       704
713
## 0.4409062 0.4834459 0.4391199 0.5259745 0.7622888 0.6187519 0.6727509
0.6843394
##       725       734       738       739       742       749       750
751
## 0.5423786 0.5059177 0.5280946 0.5974383 0.4751990 0.3852932 0.4960101
0.5537283
##       754       755       770       774       776       777       778
781
## 0.4640371 0.5425372 0.4221692 0.5172701 0.5007677 0.5336175 0.5572564
0.5290023
##       782       799       807       809       838       840       841
845
## 0.4745803 0.5766906 0.4943764 0.5255799 0.6233831 0.6266119 0.6020404
0.6604659
##       850       853       857       862       864       866       871
872
## 0.5169404 0.4973800 0.5494076 0.5249868 0.5656705 0.4571155 0.5686793
0.5785931
##       877       881       883       931       946       948       954
961
## 0.5408844 0.4889265 0.5191947 0.6263074 0.6425902 0.6617090 0.6204259
0.5919682
##       972       974       980       983       985       992       998
1000
## 0.5560599 0.5244349 0.6460541 0.5227695 0.5971551 0.4523306 0.5690567
0.6292451
##      1007      1008      1014      1018      1041      1044      1048
1049
## 0.5788924 0.5736299 0.5029529 0.5207543 0.5026445 0.6033319 0.6076272
0.4317442
##      1051      1052      1102      1125      1126      1133      1136
1139
## 0.6450328 0.5331123 0.5563305 0.5745281 0.6095734 0.6146867 0.5503273
0.5190852
##      1144      1151      1155      1167      1174      1175      1186
1188
## 0.5432175 0.5258301 0.5641496 0.5780070 0.5613914 0.5192439 0.4308244
0.4690415
##      1205      1280      1282      1325      1334      1340      1343
1346
## 0.5217131 0.5332455 0.5573263 0.5571932 0.6234394 0.5354962 0.6157576
0.4454995
##      1349      1353      1356      1357      1375      1379      1382
1384
## 0.4992824 0.5207479 0.5320013 0.5346183 0.5322462 0.4747996 0.4877069
```

```
0.5205702
##      1388      1389      1401      1404      1406      1407      1410 
1420 
## 0.4966243 0.5550155 0.5274900 0.4401393 0.4349415 0.5551924 0.4841425 
0.5255648 
##      1421      1463      1469      1482      1496      1497      1512 
1513 
## 0.6013672 0.5614210 0.5998144 0.4957013 0.4996805 0.5683544 0.5950181 
0.4365336 
##      1517      1519      1526      1527      1570      1574      1591 
1604 
## 0.4720283 0.4470949 0.5202745 0.6598581 0.5864791 0.5998502 0.5769452 
0.5319380 
##      1611      1637      1642      1643      1644      1647      1655 
1656 
## 0.5215359 0.5227475 0.6015760 0.5747409 0.5463780 0.5395782 0.5132214 
0.6163026 
##      1662      1663      1672      1675      1679      1692      1697 
1700 
## 0.5603584 0.4712555 0.4420800 0.5398851 0.5999264 0.6251811 0.6207035 
0.5336070 
##      1701      1708      1710      1711      1716      1726      1730 
1731 
## 0.5060598 0.5783743 0.5308739 0.5619047 0.6024500 0.5556945 0.5467731 
0.5325573 
##      1742      1744      1745      1756      1758      1780      1781 
1782 
## 0.5470496 0.5371075 0.4670391 0.6158496 0.4151337 0.6669325 0.6494218 
0.6407428 
##      1789      1791      1792      1796      1801      1815      1820 
1822 
## 0.6838422 0.6811466 0.5585798 0.4702861 0.5944585 0.4679132 0.4086649 
0.4446608 
##      1886      1901      1910      1915      1916      1918      1925 
1926 
## 0.5329494 0.5432128 0.4400496 0.5433539 0.4555828 0.4821323 0.5549179 
0.4716907 
##      1938      1941      1946      1955      1957      1959      1965 
1967 
## 0.4900514 0.5249789 0.4874487 0.6458684 0.4588297 0.4381748 0.4278822 
0.4323792 
##      1977      1982      1985      2011      2017      2044      2050 
2051 
## 0.3255402 0.7035298 0.3799807 0.6792656 0.5741102 0.6417920 0.6003461 
0.1082929 
##      2061      2062      2072      2076      2084      2085      2088 
2095 
## 0.5682046 0.5889992 0.5773260 0.5036980 0.5612957 0.4811309 0.4876841 
0.6628364 
##      2097      2103      2108      2109      2112      2118      2120
```

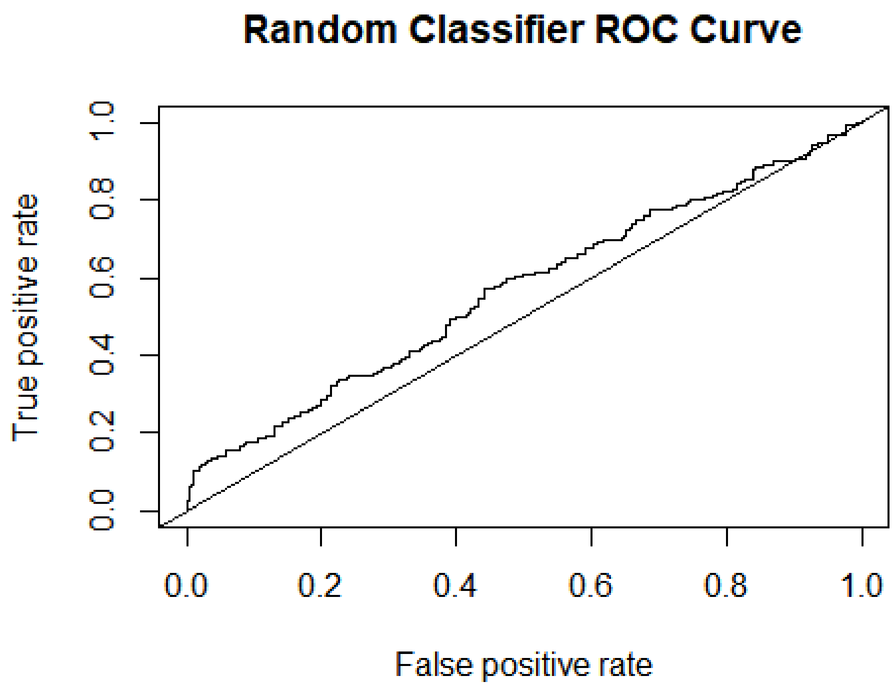
```
2126
## 0.6250690 0.6730563 0.5615127 0.4955295 0.5457553 0.5963316 0.6222580
0.5405464
##      2131      2133      2136      2147      2148      2160      2170
2176
## 0.7396405 0.6179708 0.5488524 0.5573368 0.6451024 0.4570316 0.7704635
0.5322031
##      2179      2190      2191      2199      2200      2203      2206
2214
## 0.4488067 0.4240264 0.4540451 0.3432531 0.4002635 0.5282793 0.7545726
0.6169888
##      2219      2228      2230      2234      2254      2256      2257
2259
## 0.5616713 0.5183780 0.4613496 0.4590995 0.6586107 0.5366357 0.5227688
0.5419317
##      2266      2275      2279      2295      2302      2304      2306
2313
## 0.5182406 0.4325437 0.5380535 0.5629980 0.6044548 0.4795975 0.4724504
0.5158013
##      2314      2322      2326      2331      2333      2334      2338
2344
## 0.5020605 0.5424325 0.5602073 0.4643113 0.3345594 0.3591366 0.5034620
0.3744996
##      2352      2355      2356      2367      2380      2391      2419
2428
## 0.4304851 0.5334792 0.5479114 0.4241526 0.8575085 0.6795505 0.5757675
0.5559770
##      2433      2439      2440      2444      2445      2460      2461
2472
## 0.4459724 0.6890893 0.5642916 0.5356200 0.5399008 0.5669031 0.5785658
0.4998353
##      2475      2477      2481      2486      2492      2499      2500
2524
## 0.4203023 0.5347260 0.6643861 0.5096000 0.4994288 0.4793758 0.5194056
0.5102245
##      2527      2536      2539      2542      2582      2586      2591
2592
## 0.5015849 0.5240544 0.4361483 0.4931991 0.5448986 0.5765133 0.4927361
0.5210213
##      2601      2605      2608      2613      2614      2621      2624
2625
## 0.5413271 0.6587889 0.4655163 0.4808433 0.4635626 0.4937240 0.6409709
0.5906362
##      2679      2680      2683      2689      2691      2694      2698
2706
## 0.6046776 0.4391088 0.6139144 0.4364304 0.4468444 0.4962749 0.4946661
0.5497700
##      2759      2766      2773      2785      2790      2794      2804
2808
## 0.5002412 0.6087380 0.5443489 0.5402699 0.4871536 0.4673248 0.4938648
```

```
0.5475586
##      2826      2840      2841      2859      2908      2916      2921
2937
## 0.6281080 0.4909352 0.5186879 0.5336737 0.5311305 0.4518850 0.5239176
0.6180647
##      2945      2948      2958      2959      2965      2968      2990
3001
## 0.5691234 0.5091000 0.5763302 0.4076192 0.5310430 0.5402315 0.5861816
0.5791728
##      3007      3008      3039      3043      3053      3069      3078
3091
## 0.7741459 0.4369698 0.5161726 0.5001619 0.4705102 0.5933727 0.4738019
0.5416625
##      3096      3119      3121      3125      3130      3131      3152
3154
## 0.5316971 0.5704817 0.6291524 0.6315473 0.4385403 0.5844113 0.5783448
0.5071103
##      3172      3173      3178      3184      3187      3191      3195
3201
## 0.4865458 0.7033675 0.6546516 0.5217788 0.5619528 0.4580828 0.4508053
0.6538957
##      3216      3223      3226      3229      3234      3242      3249
3252
## 0.3598177 0.4731107 0.6111626 0.4061311 0.4644791 0.5084422 0.7400134
0.6081588
##      3258      3260      3261      3281      3282      3290      3295
3300
## 0.5915343 0.5052442 0.4759537 0.4591412 0.4598874 0.5592876 0.5014009
0.4381622
##      3307      3317      3323      3325      3335      3342      3347
3351
## 0.5885089 0.7841377 0.6575973 0.6842450 0.6031103 0.6034091 0.6251817
0.5293031
##      3390      3391      3400      3404      3414      3416      3418
3419
## 0.5640261 0.5512057 0.4605861 0.6567900 0.6063193 0.5956251 0.5062087
0.5632565
##      3421      3422      3442      3454      3455      3457      3461
3465
## 0.5577242 0.5398675 0.6556427 0.5647142 0.5348790 0.4796905 0.4410261
0.4763032
##      3468      3486      3490      3501      3502      3509      3511
3513
## 0.5345307 0.5996396 0.5475095 0.5232536 0.5189782 0.5677800 0.5459010
0.4887729
##      3516      3538      3555      3559      3560      3573      3575
3581
## 0.4812252 0.6519247 0.5364428 0.5722417 0.6590107 0.4578229 0.5096135
0.5798097
##      3583      3595      3608      3612      3614      3616      3617
```

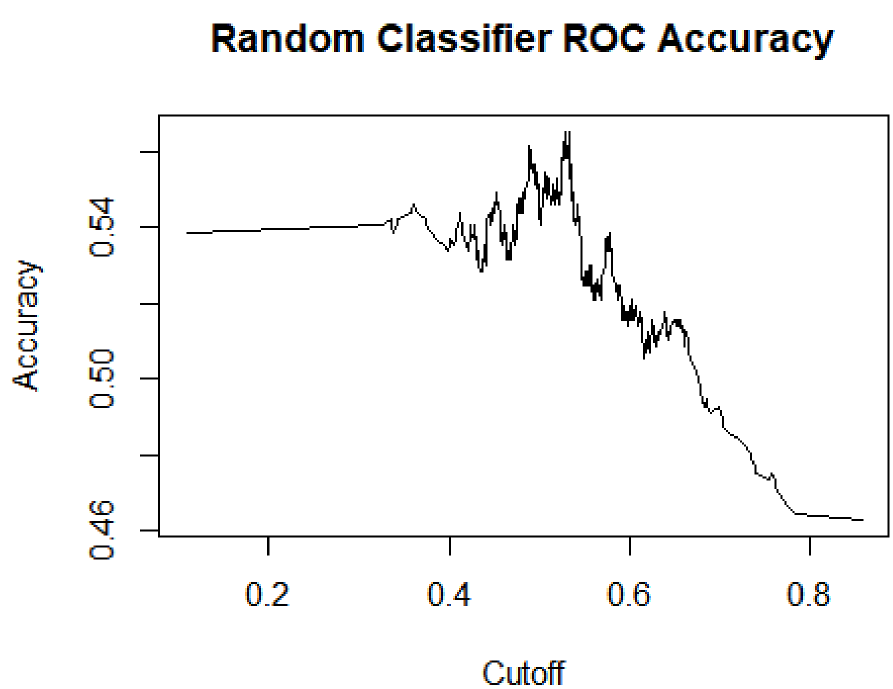
```
3623
## 0.5094299 0.5293335 0.5890786 0.4652554 0.4634752 0.5049268 0.5378872
0.5610395
##      3625      3657      3691      3697      3701      3704      3714
3738
## 0.4066217 0.5591496 0.5324950 0.4754418 0.4281104 0.4008405 0.5740350
0.4412468
##      3740      3741      3760      3761      3766      3773      3781
3786
## 0.5723889 0.4750492 0.5328523 0.5167054 0.5210059 0.4513381 0.5805483
0.5465162
##      3792      3800      3811      3817      3818      3826      3829
3830
## 0.5297408 0.3642742 0.5867721 0.6255879 0.5555631 0.4275602 0.3386474
0.5551538
##      3835      3859      3863      3870      3874      3888      3891
3895
## 0.6338502 0.5269601 0.4235917 0.4491953 0.5176159 0.5592171 0.4614864
0.4884326
##      3901      3920      3923      3930      3939      3941      3943
3949
## 0.5231622 0.7357315 0.6658728 0.4495513 0.4564838 0.5210645 0.4993354
0.4740183
##      3959      3960      3961      3979      3980      3983      3987
4004
## 0.5869448 0.5000828 0.7354823 0.5938803 0.6371492 0.4791729 0.5232760
0.5422744
##      4005      4009      4010      4012      4014      4015      4018
4052
## 0.5250218 0.5766490 0.6155345 0.5211670 0.4770478 0.6280648 0.5524580
0.5820986
##      4053      4058      4061      4064      4075      4089      4092
4118
## 0.4835632 0.5620174 0.4516542 0.5362904 0.5383725 0.6137565 0.5723348
0.4820976
##      4130      4133      4135      4161      4170      4175
## 0.4931545 0.4667228 0.4366093 0.4845308 0.4947791 0.5293155

#Convert Sex Probabilities to "M" if >0.5 else "F" and change it to a factor
from char
sex.prob <- ifelse(glm.predict > 0.5, "M", "F")
sex.prob <- factor(sex.prob)

#Create confusion matrix
confusionMatrix(sex.prob, test$Sex)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   F   M
```

```
##          F  91  87
##          M 170 218
##
##                Accuracy : 0.5459
##                  95% CI : (0.5039, 0.5875)
##     No Information Rate : 0.5389
##     P-Value [Acc > NIR] : 0.3843
##
##                   Kappa : 0.0649
##
##  Mcnemar's Test P-Value : 3.137e-07
##
##             Sensitivity : 0.3487
##             Specificity : 0.7148
##          Pos Pred Value : 0.5112
##          Neg Pred Value : 0.5619
##              Prevalence : 0.4611
##          Detection Rate : 0.1608
##    Detection Prevalence : 0.3145
##       Balanced Accuracy : 0.5317
##
##        'Positive' Class : F
##
```

```
#Random classifier ROC
roc.predict <- prediction(glm.predict, test$Sex)

#Measure performance of Random Classifier on TPR and FPR
roc.perform <- performance(roc.predict, measure = "tpr", x.measure = "fpr")
plot(roc.perform, main="Random Classifier ROC Curve")
#Plot AUC
abline(0,1)
```



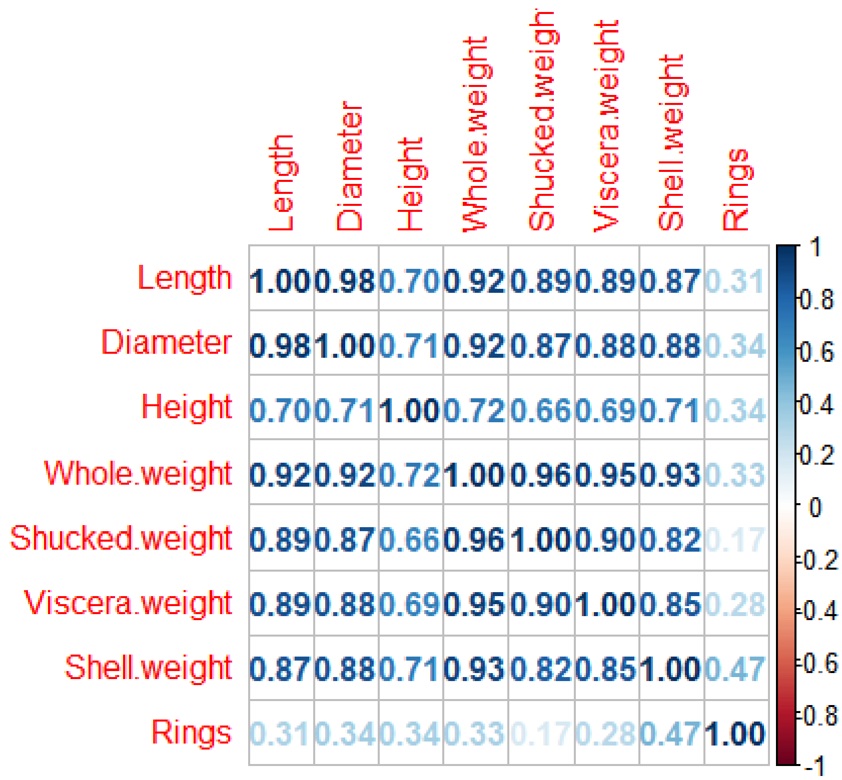

```
#Measure performance of Random Classifier on Accuracy
roc.perform <- performance(roc.predict, measure = "acc")
plot(roc.perform, main="Random Classifier ROC Accuracy")
```



From the above ROC Plots, we can see that our random classifier performs slightly above the random choice resulting in a higher AUC. This means that the model has a higher chance of predicting 'M' as 'M' and 'F' as 'F' (Since it has a higher TPR compared to FPR).

From the second plot above, we can estimate that the accuracy is around 52% at the 50% cutoff point. Our logistic regression model had an accuracy of 53% so we can say the accuracy of our logistic regression model and random classifier ROC are very similar in performance.

```
#Plotting the correlations between the predictors
corrplot(cor(abalone_data[,-1]), method = "number")
```



From the above correlations, we can see that many of the predictors have a high correlation. The only feature that has a weak relationship is Rings.

This means that the performance of the classifier is not great because since we have high correlation, a change in one variable would result in a change in another. This is not good for a model as it can result in fluctuations and instability.

```
#Problem 2
library(e1071)

#Load data from UCI repository
mushroom_data <- read.csv(file="https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data", 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"))

#Replace missing values "?" with NA's
mushroom_data[mushroom_data == "?"]<- NA

#Check how many rows have NA's
sum(is.na(mushroom_data))

## [1] 2480

#Omit all records containing NA's
mushroom_data <- na.omit(mushroom_data)

#Confirm that there aren't any NA's left
sum(is.na(mushroom_data))

## [1] 0
```

From above, we can see that initially, there are 2480 samples that have NA's out of 8123 total samples. This means that a good portion of the sample contains missing values and if we were to replace it with some mean/median, it will affect the results of the model since the data will be biased.

Therefore we drop the records containing NA's which still leaves us with sufficient samples to train/test

```
#Turn Class feature into a factor from char
mushroom_data$Class <- factor(mushroom_data$Class)

#Showing data types, summary and first 6 rows of dataset
str(mushroom_data)

## 'data.frame':    5643 obs. of  23 variables:
##  $ Class                   : Factor w/ 2 levels "e","p": 1 1 2 1 1 1 1 2 1 1 ...
##  $ cap.shape               : chr  "x" "b" "x" "x" ...
##  $ cap.surface             : chr  "s" "s" "y" "s" ...
##  $ cap.color               : chr  "y" "w" "w" "g" ...
##  $ bruises                 : chr  "t" "t" "t" "f" ...
##  $ odor                    : chr  "a" "l" "p" "n" ...
##  $ gill.attachment         : chr  "f" "f" "f" "f" ...
##  $ gill.spacing            : chr  "c" "c" "c" "w" ...
##  $ gill.size               : chr  "b" "b" "n" "b" ...
##  $ gill.color              : chr  "k" "n" "n" "k" ...
##  $ stalk.shape             : chr  "e" "e" "e" "t" ...
##  $ stalk.root              : chr  "c" "c" "e" "e" ...
##  $ stalk.surface.above.ring: chr  "s" "s" "s" "s" ...
##  $ stalk.surface.below.ring: chr  "s" "s" "s" "s" ...
##  $ stalk.color.above.ring  : chr  "w" "w" "w" "w" ...
```

```
##  $ stalk.color.below.ring  : chr  "w" "w" "w" "w" ...
##  $ veil.type               : chr  "p" "p" "p" "p" ...
##  $ veil.color              : chr  "w" "w" "w" "w" ...
##  $ ring.number             : chr  "o" "o" "o" "o" ...
##  $ ring.type               : chr  "p" "p" "p" "e" ...
##  $ spore.print.color       : chr  "n" "n" "k" "n" ...
##  $ population              : chr  "n" "n" "s" "a" ...
##  $ habitat                 : chr  "g" "m" "u" "g" ...
##  - attr(*, "na.action")= 'omit' Named int [1:2480] 3984 4023 4076 4100
4104 4196 4200 4283 4291 4326 ...
##   ..- attr(*, "names")= chr [1:2480] "3984" "4023" "4076" "4100" ...
```

```
summary(mushroom_data)
```

```
##  Class     cap.shape         cap.surface         cap.color        
##  e:3488   Length:5643        Length:5643        Length:5643       
##  p:2155   Class :character   Class :character   Class :character  
##           Mode  :character   Mode  :character   Mode  :character  
##    bruises              odor           gill.attachment    gill.spacing      
##  Length:5643        Length:5643        Length:5643        Length:5643       
##  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:5643        Length:5643        Length:5643        Length:5643       
##  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:5643              Length:5643              Length:5643           
##  Class :character         Class :character         Class :character      
##  Mode  :character         Mode  :character         Mode  :character      
##  stalk.color.below.ring  veil.type          veil.color       
##  Length:5643            Length:5643        Length:5643       
##  Class :character       Class :character   Class :character  
##  Mode  :character       Mode  :character   Mode  :character  
##  ring.number         ring.type         spore.print.color   population       
##  Length:5643        Length:5643        Length:5643        Length:5643       
##  Class :character   Class :character   Class :character   Class :character  
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character  
##    habitat         
##  Length:5643       
##  Class :character  
##  Mode  :character  
```

```
head(mushroom_data)
```

```
##   Class cap.shape cap.surface cap.color bruises odor gill.attachment
## 1     e         x           s         y       t    a               f
## 2     e         b           s         w       t    l               f
## 3     p         x           y         w       t    p               f
## 4     e         x           s         g       f    n               f
## 5     e         x           y         y       t    a               f
```

```
## 6     e         b           s         w       t    a              f
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1            c         b          k           e          c
## 2            c         b          n           e          c
## 3            c         n          n           e          e
## 4            w         b          k           t          e
## 5            c         b          n           e          c
## 6            c         b          g           e          c
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                        s                        s                      w
## 2                        s                        s                      w
## 3                        s                        s                      w
## 4                        s                        s                      w
## 5                        s                        s                      w
## 6                        s                        s                      w
##   stalk.color.below.ring veil.type veil.color ring.number ring.type
## 1                      w         p          w           o         p
## 2                      w         p          w           o         p
## 3                      w         p          w           o         p
## 4                      w         p          w           o         e
## 5                      w         p          w           o         p
## 6                      w         p          w           o         p
##   spore.print.color population habitat
## 1                 n          n       g
## 2                 n          n       m
## 3                 k          s       u
## 4                 n          a       g
## 5                 k          n       g
## 6                 k          n       m
```

```
#Using sample function to perform 80/20 train-test split
sample <- sample(c(TRUE, FALSE), nrow(mushroom_data),replace=TRUE,
prob=c(0.8,0.2))
train <- mushroom_data[sample,]
test <- mushroom_data[!sample,]

dim(train)
```

```
## [1] 4498   23
```

```
dim(test)
```

```
## [1] 1145   23
```

```
#Creating the Naive Bayes classifier
nb.fit <- naiveBayes(Class ~ ., data = train)
nb.fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
```

```
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        e        p
## 0.613606 0.386394
##
## Conditional probabilities:
##    cap.shape
## Y             b           c           f           k           s
x
##   e 0.075000000 0.000000000 0.415579710 0.006159420 0.008333333
0.494927536
##   p 0.017836594 0.002301496 0.451093211 0.007479862 0.000000000
0.521288838
##
##    cap.surface
## Y             f           g           s           y
##   e 0.408333333 0.000000000 0.210869565 0.380797101
##   p 0.345799770 0.002301496 0.255466053 0.396432681
##
##    cap.color
## Y             b           c           e           g           n
p
##   e 0.000000000 0.010144928 0.165217391 0.256521739 0.285869565
0.002898551
##   p 0.059263521 0.005178366 0.005753740 0.369390104 0.062140391
0.037974684
##    cap.color
## Y             w           y
##   e 0.162681159 0.116666667
##   p 0.149597238 0.310701956
##
##    bruises
## Y           f         t
##   e 0.2728261 0.7271739
##   p 0.7100115 0.2899885
##
##    odor
## Y            a          c          f          l          m          n
##   e 0.11739130 0.00000000 0.00000000 0.11521739 0.00000000 0.76739130
##   p 0.00000000 0.08515535 0.73762946 0.00000000 0.01668585 0.04200230
##    odor
## Y            p
##   e 0.00000000
##   p 0.11852704
##
##    gill.attachment
## Y            a          f
```

```
##   e 0.00000000 1.00000000
##   p 0.00863061 0.99136939
##
##    gill.spacing
## Y           c          w
##   e 0.73007246 0.26992754
##   p 0.95109321 0.04890679
##
##    gill.size
## Y           b          n
##   e 0.92898551 0.07101449
##   p 0.78883774 0.21116226
##
##    gill.color
## Y           g          h          k          n          p          r
##   e 0.04420290 0.05724638 0.10072464 0.25036232 0.21086957 0.00000000
##   p 0.23417722 0.23820483 0.02934407 0.05178366 0.30437284 0.01208285
##    gill.color
## Y           u          w          y
##   e 0.12101449 0.21557971 0.00000000
##   p 0.01898734 0.10011507 0.01093211
##
##    stalk.shape
## Y          e         t
##   e 0.2583333 0.7416667
##   p 0.8670886 0.1329114
##
##    stalk.root
## Y           b          c          e          r
##   e 0.54384058 0.15000000 0.25326087 0.05289855
##   p 0.86133487 0.02013809 0.11852704 0.00000000
##
##    stalk.surface.above.ring
## Y            f           k           s           y
##   e 0.122463768 0.000000000 0.872101449 0.005434783
##   p 0.067894131 0.621403913 0.307249712 0.003452244
##
##    stalk.surface.below.ring
## Y           f          k          s          y
##   e 0.11594203 0.00000000 0.82572464 0.05833333
##   p 0.06904488 0.60471807 0.30609896 0.02013809
##
##    stalk.color.above.ring
## Y            b           c           g           n           p
w
##   e 0.000000000 0.000000000 0.158333333 0.005434783 0.163405797
0.672826087
##   p 0.199079402 0.016685846 0.000000000 0.203107020 0.202531646
0.375143843
##    stalk.color.above.ring
```

```
## Y           y
##   e 0.000000000
##   p 0.003452244
##
##    stalk.color.below.ring
## Y           b           c           g           n           p
w
##   e 0.000000000 0.000000000 0.160144928 0.019565217 0.162318841
0.657971014
##   p 0.196777906 0.016685846 0.000000000 0.197353280 0.210586881
0.375143843
##    stalk.color.below.ring
## Y           y
##   e 0.000000000
##   p 0.003452244
##
##    veil.type
## Y   p
##   e 1
##   p 1
##
##    veil.color
## Y           w           y
##   e 1.000000000 0.000000000
##   p 0.996547756 0.003452244
##
##    ring.number
## Y          n          o          t
##   e 0.00000000 0.98586957 0.01413043
##   p 0.01668585 0.94879171 0.03452244
##
##    ring.type
## Y           e           l           n           p
##   e 0.240217391 0.000000000 0.000000000 0.759782609
##   p 0.003452244 0.604718067 0.016685846 0.375143843
##
##    spore.print.color
## Y          h          k          n          r          u          w
##   e 0.00000000 0.46594203 0.49130435 0.00000000 0.01449275 0.02826087
##   p 0.73762946 0.10011507 0.10356732 0.03452244 0.00000000 0.02416571
##
##    population
## Y          a          c          n          s          v          y
##   e 0.11485507 0.00000000 0.07391304 0.21413043 0.30797101 0.28913043
##   p 0.00000000 0.02416571 0.00000000 0.16513234 0.50920598 0.30149597
##
##    habitat
## Y           d           g           l           m           p
u
##   e 0.518115942 0.328623188 0.014130435 0.072826087 0.039130435
```

```
0.027173913
##   p 0.297468354 0.346375144 0.007479862 0.015535098 0.203107020
0.130034522

#Predicting using the Naive Bayes classifier in-training and in-test
predict_train = predict(nb.fit, train)
predict_test = predict(nb.fit, test)

#Calculating the accuracy of the classifiers


cat("Accuracy of the classifier in-training: ",mean(predict_train ==
train$Class) *100,"%")

## Accuracy of the classifier in-training:  95.06447 %

cat("\nAccuracy of the classifier in-test: ",mean(predict_test == test$Class)
*100,"%")

##
## Accuracy of the classifier in-test:  94.84716 %

#Using table function to create a confusion matrix of predicted vs actual
classes

table(predict_test, test$Class)

##
## predict_test   e   p
##            e 727  58
##            p   1 359
```

#The model produced 58 false positives

```
#Question 3

#Load data from UCI repository
yacht_data <- read.table("https://archive.ics.uci.edu/ml/machine-learning-
databases/00243/yacht_hydrodynamics.data", col.names= c ("Longitudinal
position", "Prismatic coefficient", "Length displacement ratio", "Beam
draught ratio", "Length beam ratio", "Froude number", "Residuary
resistance"))


#Showing data types, summary and first 6 rows of dataset
str(yacht_data)

## 'data.frame':    308 obs. of  7 variables:
##  $ Longitudinal.position    : num  -2.3 -2.3 -2.3 -2.3 -2.3 -2.3 -2.3 -2.3
-2.3 -2.3 ...
##  $ Prismatic.coefficient    : num  0.568 0.568 0.568 0.568 0.568 0.568
```

```
0.568 0.568 0.568 0.568 ...
##  $ Length.displacement.ratio: num  4.78 4.78 4.78 4.78 4.78 4.78 4.78 4.78
4.78 4.78 ...
##  $ Beam.draught.ratio       : num  3.99 3.99 3.99 3.99 3.99 3.99 3.99 3.99
3.99 3.99 ...
##  $ Length.beam.ratio        : num  3.17 3.17 3.17 3.17 3.17 3.17 3.17 3.17
3.17 3.17 ...
##  $ Froude.number            : num  0.125 0.15 0.175 0.2 0.225 0.25 0.275
0.3 0.325 0.35 ...
##  $ Residuary.resistance     : num  0.11 0.27 0.47 0.78 1.18 1.82 2.61 3.76
4.99 7.16 ...

summary(yacht_data)

##  Longitudinal.position Prismatic.coefficient Length.displacement.ratio
##  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.ratio 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

head(yacht_data)

##   Longitudinal.position Prismatic.coefficient Length.displacement.ratio
## 1                  -2.3                 0.568                      4.78
## 2                  -2.3                 0.568                      4.78
## 3                  -2.3                 0.568                      4.78
## 4                  -2.3                 0.568                      4.78
## 5                  -2.3                 0.568                      4.78
## 6                  -2.3                 0.568                      4.78
##   Beam.draught.ratio Length.beam.ratio Froude.number Residuary.resistance
## 1               3.99              3.17         0.125                 0.11
## 2               3.99              3.17         0.150                 0.27
## 3               3.99              3.17         0.175                 0.47
## 4               3.99              3.17         0.200                 0.78
## 5               3.99              3.17         0.225                 1.18
## 6               3.99              3.17         0.250                 1.82

#Using createDataPartition to perform 80/20 train-test split
set.seed(10)
datasetPartition <- createDataPartition(yacht_data$Residuary.resistance, p =
0.8, list = FALSE, times = 1)
```

```
train <- yacht_data[datasetPartition,]
test <- yacht_data[-datasetPartition,]

dim(train)

## [1] 248   7

dim(test)

## [1] 60  7

#Using lm to fit model
lm.fits <- lm(Residuary.resistance ~., data = train)

summary(lm.fits)

##
## Call:
## lm(formula = Residuary.resistance ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.533  -7.573  -2.119   5.906  30.617
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 6.0654    32.8452   0.185    0.854
## Longitudinal.position       0.1091     0.3881   0.281    0.779
## Prismatic.coefficient     -47.4781    53.5606  -0.886    0.376
## Length.displacement.ratio  -7.9285    17.3023  -0.458    0.647
## Beam.draught.ratio          3.0198     6.6950   0.451    0.652
## Length.beam.ratio           7.0651    17.3515   0.407    0.684
## Froude.number             121.8248     5.7657  21.129   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.161 on 241 degrees of freedom
## Multiple R-squared:  0.6522, Adjusted R-squared:  0.6436
## F-statistic: 75.33 on 6 and 241 DF,  p-value: < 2.2e-16

#Creating our own function for MSE and RMSE Calculations
MSE <- mean(lm.fits$residuals^2)
RMSE <- sqrt(MSE)

cat("Mean Square Error: ", MSE)

## Mean Square Error:  81.54859

cat(", Root Mean Square Error: ", RMSE)
```

```
## , Root Mean Square Error:  9.030426
```

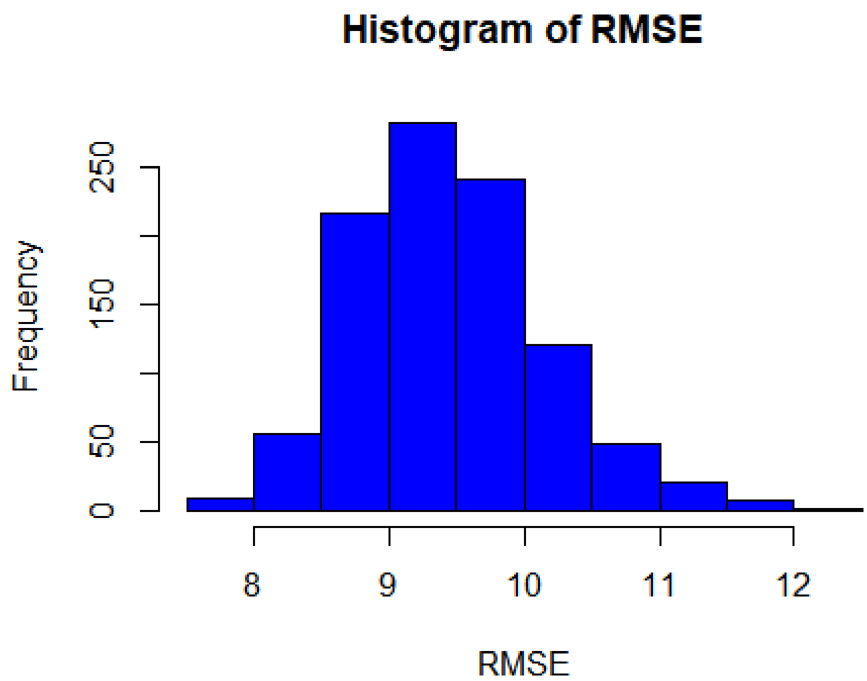
Training MSE is 81.54859, RMSE is 9.030426, and R^2 is 0.6522

```
#Using the trainControl method to perform a bootstrap
fitControl <- trainControl(method="boot", number = 1000)

lm.fits2 <- train(Residuary.resistance~., data = train, method = "lm",
trControl = fitControl)

#Showing results from bootstrap resampling
summary(lm.fits2$resample)

##       RMSE           Rsquared            MAE           Resample
##  Min.   : 7.731   Min.   :0.5277   Min.   :6.162   Length:1000
##  1st Qu.: 8.942   1st Qu.:0.6124   1st Qu.:7.232   Class :character
##  Median : 9.374   Median :0.6334   Median :7.513   Mode  :character
##  Mean   : 9.443   Mean   :0.6316   Mean   :7.541
##  3rd Qu.: 9.851   3rd Qu.:0.6515   3rd Qu.:7.845
##  Max.   :12.093   Max.   :0.7122   Max.   :9.028

mse_boot = mean(lm.fits2$resample$RMSE)^2
rmse_boot = mean(lm.fits2$resample$RMSE)
r2_boot = mean(lm.fits2$resample$Rsquared)

cat("Mean Square Error - Bootstrap: ", mse_boot)

## Mean Square Error - Bootstrap:  89.1627

cat(", Root Mean Square Error - Bootstrap: ", rmse_boot)

## , Root Mean Square Error - Bootstrap:  9.4426

cat(", R^2 - Bootstrap: ", r2_boot)

## , R^2 - Bootstrap:  0.6315964
```

#Bootstrap Model Training MSE is 89.1627, RMSE is 9.4426 and R^2 is 0.6316

#The bootstrap Model has a slightly higher MSE and RMSE compared to the intial model showing a slight decrease in performance

```
#Plotting histogram of the RMSE values using hist
hist(lm.fits2$resample$RMSE, xlab = "RMSE", main = "Histogram of RMSE", col =
"blue")
```




```
#Perform prediction on test set for original and bootstrap models
y_hat_original <- predict(lm.fits,test)
y_hat_bootstrap <- predict(lm.fits2,test)

y_test <- test$Residuary.resistance

#y_hat_bootstrap
#y_hat_original
#Compute testing MSE, RMSE, and R^2 for original and bootstrap models


test_mse_original <- mean((y_test - y_hat_original)^2)
test_rmse_original <- sqrt(test_mse_original)
RSS_original <- sum((y_test - y_hat_original)^2)
TSS_Original <- (sum((y_test - mean(y_test))^2))
test_rsquared_original <- 1-(RSS_original/TSS_Original)

test_mse_bootstrap <- mean((y_test - y_hat_bootstrap)^2)
test_rmse_bootstrap <- sqrt(test_mse_bootstrap)
RSS_bootstrap <- sum((y_test - y_hat_bootstrap)^2)
TSS_bootstrap <- (sum((y_test - mean(y_test))^2))
test_rsquared_boostrap <- 1-(RSS_bootstrap/TSS_bootstrap)

cat("Original Testing MSE : ", test_mse_original)

## Original Testing MSE :  67.05404
```

```r
cat(" Bootstrap Testing MSE : ", test_mse_bootstrap)

##  Bootstrap Testing MSE :  67.05404

cat("Original Testing RMSE : ", test_rmse_original)

## Original Testing RMSE :  8.188653

cat(" Bootstrap Testing RMSE : ", test_rmse_bootstrap)

##  Bootstrap Testing RMSE :  8.188653

cat("Original Testing R^Squared : ", test_rsquared_original)

## Original Testing R^Squared :  0.6757548

cat(" Bootstrap Testing R^Squared : ", test_rsquared_boostrap)

##  Bootstrap Testing R^Squared :  0.6757548
```

From above, we can see that both the original and bootstrap model have identical testing MSE, RMSE, and R^Squared.

```r
#Problem 4

#Load data from UCI repository
German_credit_data <- read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data-numeric', sep= '', header = F )

#Using str(German_credit_data), we see that the last feature data type is int so we change it to factor

#Turn final column into a factor from int
German_credit_data$V25 <- factor(German_credit_data$V25)

#Showing data types, summary and first 6 rows of dataset
str(German_credit_data)

## 'data.frame':    1000 obs. of  25 variables:
##  $ V1 : int  1 2 4 1 1 4 4 2 4 2 ...
##  $ V2 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ V3 : int  4 2 4 2 3 2 2 2 2 4 ...
##  $ V4 : int  12 60 21 79 49 91 28 69 31 52 ...
##  $ V5 : int  5 1 1 1 1 5 3 1 4 1 ...
##  $ V6 : int  5 3 4 4 3 3 5 3 4 1 ...
##  $ V7 : int  3 2 3 3 3 3 3 3 1 4 ...
##  $ V8 : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ V9 : int  1 1 1 2 4 4 2 3 1 3 ...
##  $ V10: int  67 22 49 45 53 35 53 35 61 28 ...
##  $ V11: int  3 3 3 3 3 3 3 3 3 3 ...
##  $ V12: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ V13: int  1 1 2 2 2 2 1 1 1 1 ...
```

```
##  $ V14: int  2 1 1 1 1 2 1 2 1 1 ...
##  $ V15: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ V16: int  0 0 0 0 1 0 0 0 0 1 ...
##  $ V17: int  0 0 0 0 0 0 0 1 0 0 ...
##  $ V18: int  1 1 1 0 1 1 1 1 1 1 ...
##  $ V19: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V20: int  0 0 0 0 0 0 0 1 0 0 ...
##  $ V21: int  1 1 1 0 0 0 1 0 1 1 ...
##  $ V22: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V23: int  0 0 1 0 0 1 0 0 1 0 ...
##  $ V24: int  1 1 0 1 1 0 1 0 0 0 ...
##  $ V25: Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...
```

```
summary(German_credit_data)
```

```
##        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   1:700
##  1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.0   1st Qu.:0.00   2:300
##  Median :1.000   Median :0.000   Median :0.0   Median :1.00
##  Mean   :0.713   Mean   :0.022   Mean   :0.2   Mean   :0.63
##  3rd Qu.:1.000   3rd Qu.:0.000   3rd Qu.:0.0   3rd Qu.:1.00
##  Max.   :1.000   Max.   :1.000   Max.   :1.0   Max.   :1.00

head(German_credit_data)

##   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

#Using createDataPartition to perform 80/20 train-test split

datasetPartition <- createDataPartition(German_credit_data$V25, p = 0.8, list
= FALSE, times = 1)

train <- German_credit_data[datasetPartition,]
test <- German_credit_data[-datasetPartition,]

dim(train)

## [1] 800  25

dim(test)

## [1] 200  25

#Using glm to fit a logistic regression
set.seed(10)
```

```r
glm.fits <- glm(V25 ~ ., data = train, family = binomial)

summary(glm.fits)

##
## Call:
## glm(formula = V25 ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0191  -0.7095  -0.4211   0.8108   2.6275
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.009108   1.254860   2.398 0.016487 *
## V1          -0.542844   0.080082  -6.779 1.21e-11 ***
## V2           0.032318   0.009605   3.365 0.000767 ***
## V3          -0.304940   0.097410  -3.130 0.001745 **
## V4           0.003442   0.004230   0.814 0.415732
## V5          -0.215729   0.066375  -3.250 0.001154 **
## V6          -0.120843   0.084578  -1.429 0.153070
## V7          -0.268982   0.123818  -2.172 0.029826 *
## V8          -0.007400   0.092741  -0.080 0.936402
## V9           0.179724   0.112043   1.604 0.108700
## V10         -0.016742   0.009670  -1.731 0.083388 .
## V11         -0.370143   0.123179  -3.005 0.002656 **
## V12          0.083038   0.185225   0.448 0.653931
## V13          0.219831   0.255602   0.860 0.389762
## V14         -0.269136   0.214763  -1.253 0.210142
## V15         -0.998295   0.625196  -1.597 0.110317
## V16          0.590358   0.214291   2.755 0.005870 **
## V17         -1.000513   0.383962  -2.606 0.009167 **
## V18          0.993309   0.444885   2.233 0.025567 *
## V19          1.114661   0.602358   1.850 0.064242 .
## V20          0.480693   0.404507   1.188 0.234698
## V21         -0.082091   0.350626  -0.234 0.814886
## V22         -0.228084   0.735426  -0.310 0.756456
## V23         -0.078548   0.362342  -0.217 0.828382
## V24          0.022685   0.284986   0.080 0.936555
## ---
## 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: 756.26  on 775  degrees of freedom
## AIC: 806.26
##
## Number of Fisher Scoring iterations: 5
```

```
#Convert V25 fitted values from the model to 2 if >0.5 else 1 and change it
to a factor from int
v25.prob <- ifelse(glm.fits$fitted.values > 0.5,2,1)
v25.prob <- factor(v25.prob)

#Create confusion matrix to use later to find training precision/recall and
F1  results
confusion_matrix <- confusionMatrix(v25.prob, train$V25, mode="everything")
confusion_matrix

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 500 122
##          2  60 118
##
##                Accuracy : 0.7725
##                  95% CI : (0.7418, 0.8011)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 2.700e-06
##
##                   Kappa : 0.4152
##
##  Mcnemar's Test P-Value : 6.137e-06
##
##             Sensitivity : 0.8929
##             Specificity : 0.4917
##          Pos Pred Value : 0.8039
##          Neg Pred Value : 0.6629
##               Precision : 0.8039
##                  Recall : 0.8929
##                      F1 : 0.8460
##              Prevalence : 0.7000
##          Detection Rate : 0.6250
##    Detection Prevalence : 0.7775
##       Balanced Accuracy : 0.6923
##
##        'Positive' Class : 1
##

#Training Precision/Recall and F1 Results
cat("Training Precision: ", confusion_matrix$byClass[5])

## Training Precision:  0.8038585

cat("\nTraining Recall: ", confusion_matrix$byClass[6])

##
## Training Recall:  0.8928571
```

```r
cat("\nTraining F1: ", confusion_matrix$byClass[7])

##
## Training F1:  0.8460237

#Using the trainControl method to perform a cross validation
fitControl <- trainControl(method="cv", number = 10)

glm.fits2 <- train(V25~., data = train, method = "glm", family = "binomial",
trControl = fitControl)

#Convert V25 fitted values from the cv model to 2 if >0.5 else 1 and change
it to a factor from int
v25_cv.prob <- ifelse(glm.fits2$finalModel$fitted.values > 0.5,2,1)
v25_cv.prob <- factor(v25_cv.prob)

#Create confusion matrix to use later to find cross-validated training
precision/recall and F1  results
confusion_matrix_cv <- confusionMatrix(v25_cv.prob, train$V25,
mode="everything")
confusion_matrix_cv

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 500 122
##          2  60 118
##
##                Accuracy : 0.7725
##                  95% CI : (0.7418, 0.8011)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 2.700e-06
##
##                   Kappa : 0.4152
##
##  Mcnemar's Test P-Value : 6.137e-06
##
##             Sensitivity : 0.8929
##             Specificity : 0.4917
##          Pos Pred Value : 0.8039
##          Neg Pred Value : 0.6629
##               Precision : 0.8039
##                  Recall : 0.8929
##                      F1 : 0.8460
##              Prevalence : 0.7000
##          Detection Rate : 0.6250
##    Detection Prevalence : 0.7775
##       Balanced Accuracy : 0.6923
##
```

```
##        'Positive' Class : 1
##

#Training Precision/Recall and F1 Results for cv model
cat("Cross-Validated Training Precision: ", confusion_matrix_cv$byClass[5])

## Cross-Validated Training Precision:  0.8038585

cat("\nCross-Validated Training Recall: ", confusion_matrix_cv$byClass[6])

##
## Cross-Validated Training Recall:  0.8928571

cat("\nCross-Validated Training F1: ", confusion_matrix_cv$byClass[7])

##
## Cross-Validated Training F1:  0.8460237
```

From above, we can see that the cross-validated training precision/recall and F1 values are the exact same as the original fit

```
#Use predict.glm() function to perform prediction on test data using original model
glm.predict <- predict.glm(glm.fits, test, type = "response")

#Convert V25 fitted values from the model to 2 if >0.5 else 1 and change it to a factor from int
v25_test.prob <- ifelse(glm.predict > 0.5,2,1)
v25_test.prob <- factor(v25_test.prob)


#Create confusion matrix to use later to find testing precision/recall and F1 results
confusion_matrix_test <- confusionMatrix(v25_test.prob, test$V25,
mode="everything")
confusion_matrix_test

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 130  34
##          2  10  26
##
##                Accuracy : 0.78
##                  95% CI : (0.7161, 0.8354)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.0071511
##
##                   Kappa : 0.4086
##
```

```
##  Mcnemar's Test P-Value : 0.0005256
##
##             Sensitivity : 0.9286
##             Specificity : 0.4333
##          Pos Pred Value : 0.7927
##          Neg Pred Value : 0.7222
##               Precision : 0.7927
##                  Recall : 0.9286
##                      F1 : 0.8553
##              Prevalence : 0.7000
##          Detection Rate : 0.6500
##    Detection Prevalence : 0.8200
##       Balanced Accuracy : 0.6810
##
##        'Positive' Class : 1
##
```

```
#Testing Precision/Recall and F1 Results on Original model
cat("Testing Precision: ", confusion_matrix_test$byClass[5])
```

```
## Testing Precision:  0.7926829
```

```
cat("\nTesting Recall: ", confusion_matrix_test$byClass[6])
```

```
##
## Testing Recall:  0.9285714
```

```
cat("\nTesting F1: ", confusion_matrix_test$byClass[7])
```

```
##
## Testing F1:  0.8552632
```

```
#Use predict() function to perform prediction on test data using cv model
glm_cv.predict <- predict(glm.fits2, test)

#Create confusion matrix to use later to find testing precision/recall and F1
results
confusion_matrix_test_cv <- confusionMatrix(glm_cv.predict, test$V25,
mode="everything")
confusion_matrix_test_cv
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 130  34
##          2  10  26
##
##                Accuracy : 0.78
##                  95% CI : (0.7161, 0.8354)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.0071511
```

```
##
##                   Kappa : 0.4086
##
##  Mcnemar's Test P-Value : 0.0005256
##
##             Sensitivity : 0.9286
##             Specificity : 0.4333
##          Pos Pred Value : 0.7927
##          Neg Pred Value : 0.7222
##               Precision : 0.7927
##                  Recall : 0.9286
##                      F1 : 0.8553
##              Prevalence : 0.7000
##          Detection Rate : 0.6500
##    Detection Prevalence : 0.8200
##       Balanced Accuracy : 0.6810
##
##        'Positive' Class : 1
##
```

```
#Testing Precision/Recall and F1 Results for cv model
cat("Cross-Validated Testing Precision: ",
confusion_matrix_test_cv$byClass[5])
```

```
## Cross-Validated Testing Precision:  0.7926829
```

```
cat("\nCross-Validated Testing Recall: ",
confusion_matrix_test_cv$byClass[6])
```

```
##
## Cross-Validated Testing Recall:  0.9285714
```

```
cat("\nCross-Validated Testing F1: ", confusion_matrix_test_cv$byClass[7])
```

```
##
## Cross-Validated Testing F1:  0.8552632
```

From above, we can see that the cross-validated testing precision/recall and F1 values are the exact same as the original fit

#4a

Since we are assuming that X is uniformly (evenly) distributed on [0,1] and we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation, we can say that:

$X \in [0.05, 0.95]$ which means the intervals will be $[X - 0.05, X + 0.05]$ and the length will be 0.1 (10%).

$\int_{0.05}^{0.95} 10\, dx = 10\%$

Therefore, the fraction of the available observations that will be used to make the prediction will be 10%