DPA Assignment 2

Girish

2023-02-19

#Recitation Problems

#Chapter 4 #4a

Since we are assuming that is uniformly (evenly) distributed on 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:

which means the intervals will be and the length will be 0.1 (10%).

10%

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

#4b

Since we assume that are uniformly distributed on each with measurements on 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 above:

the length will be %

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 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:

is the fraction of the available observations that will be used to make the prediction.

#4d

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

* When , the fraction of the available observations used to make the prediction was
* When , the fraction of the available observations used to make the prediction was which is significantly smaller

%

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 , the length of each side is
* For , the length of each side is
* For , the length of each side is

Comment:

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

#6a

#6b

By transposing for , we get:

hours.

#7

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

#9a

Transpose and factorize to make the subject:

#9b

#Chapter 5 #2

#3

#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   
## -1.8014 -1.1982 0.8551 1.1172 1.5014   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 2.975812 0.515439 5.773 7.77e-09 \*\*\*  
## Length -0.720035 2.314460 -0.311 0.75572   
## Diameter -6.706939 2.742113 -2.446 0.01445 \*   
## Height -3.371820 2.154948 -1.565 0.11766   
## Whole.weight 0.218772 0.825647 0.265 0.79103   
## Shucked.weight 2.850311 0.995095 2.864 0.00418 \*\*   
## Viscera.weight -2.278756 1.428650 -1.595 0.11070   
## Shell.weight 0.260958 1.264993 0.206 0.83656   
## Rings 0.005944 0.017965 0.331 0.74074   
## ---  
## 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: 3062.7 on 2259 degrees of freedom  
## AIC: 3080.7  
##   
## 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) 2.975812207 0.51543893 5.7733555 7.770824e-09  
## Length -0.720034814 2.31446028 -0.3111027 7.557226e-01  
## Diameter -6.706939371 2.74211261 -2.4459022 1.444902e-02  
## Height -3.371820085 2.15494817 -1.5646873 1.176562e-01  
## Whole.weight 0.218772376 0.82564744 0.2649707 7.910320e-01  
## Shucked.weight 2.850310642 0.99509512 2.8643600 4.178529e-03  
## Viscera.weight -2.278755660 1.42864990 -1.5950413 1.107030e-01  
## Shell.weight 0.260957715 1.26499338 0.2062918 8.365630e-01  
## Rings 0.005944253 0.01796549 0.3308707 7.407422e-01

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

## Waiting for profiling to be done...

## 2.5 % 97.5 %  
## (Intercept) 1.9824599 4.00425056  
## Length -5.2545403 3.82430019  
## Diameter -12.1060285 -1.34828250  
## Height -8.1160022 0.17659568  
## Whole.weight -1.4023576 1.84486444  
## Shucked.weight 0.9046546 4.81259211  
## Viscera.weight -5.0910261 0.51595706  
## Shell.weight -2.2254485 2.74504599  
## Rings -0.0292686 0.04121025

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.

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

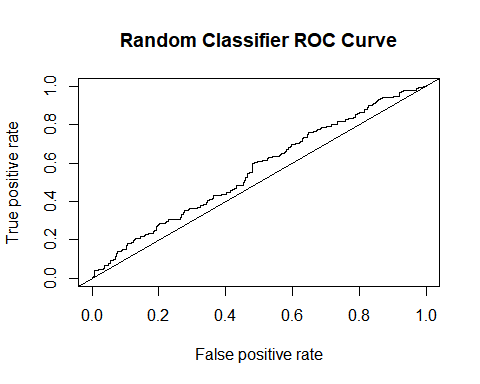
glm.predict

## 3 24 25 31 33 34 41 68   
## 0.5882481 0.4816962 0.5394495 0.4572041 0.5648635 0.3895228 0.5496576 0.5445515   
## 72 83 86 91 92 117 129 132   
## 0.5365388 0.5040548 0.5710730 0.4878161 0.4569002 0.5913110 0.6492209 0.7220109   
## 136 137 140 141 142 143 153 157   
## 0.7587618 0.7365620 0.4675904 0.4694617 0.4175868 0.4594582 0.4131443 0.5214856   
## 164 170 172 192 201 204 208 210   
## 0.4270166 0.3834057 0.5907009 0.4470380 0.4914104 0.6280875 0.4956989 0.5349558   
## 211 221 223 226 230 253 265 271   
## 0.7254975 0.5691070 0.5527188 0.6498703 0.5139334 0.5138012 0.6016012 0.3851910   
## 273 274 284 308 313 318 330 335   
## 0.4385517 0.4320050 0.6001208 0.5125922 0.4235565 0.4849168 0.5099959 0.5309915   
## 339 345 346 355 356 359 370 381   
## 0.4657630 0.5281969 0.4800674 0.4070149 0.5208461 0.5179614 0.4872232 0.5230870   
## 383 406 413 415 420 425 427 430   
## 0.5812244 0.6087651 0.5371159 0.4578903 0.3821217 0.5232897 0.4539499 0.5134773   
## 431 432 439 445 446 450 451 455   
## 0.5248837 0.4880688 0.4911167 0.4193196 0.5123231 0.3801213 0.4598723 0.4617458   
## 461 471 478 489 492 495 501 502   
## 0.4529538 0.5587336 0.3614476 0.5251578 0.4721316 0.3814080 0.3195130 0.4095402   
## 503 505 521 560 562 564 573 575   
## 0.4015178 0.5095973 0.6986113 0.6036872 0.5944785 0.5874699 0.5380117 0.5622314   
## 588 609 617 625 627 629 642 657   
## 0.6108123 0.6982737 0.7350335 0.5290643 0.6697299 0.7105913 0.5125825 0.4195299   
## 663 666 667 669 675 679 683 690   
## 0.6494330 0.5740162 0.5339216 0.5695152 0.4699448 0.7007240 0.5289018 0.5585347   
## 701 708 725 727 737 749 751 752   
## 0.5137480 0.6390144 0.5778868 0.5837456 0.5058758 0.4172681 0.5747985 0.3969729   
## 758 765 766 767 772 782 786 788   
## 0.5786130 0.4855434 0.5233888 0.6871169 0.6648125 0.4701016 0.5112976 0.4845187   
## 838 846 848 858 861 864 881 882   
## 0.6322800 0.5629155 0.5863706 0.5276173 0.5671042 0.5423092 0.4837409 0.6436492   
## 945 947 949 956 961 984 985 989   
## 0.6440439 0.5736536 0.6516271 0.5355264 0.5954404 0.6085674 0.6030917 0.4990509   
## 993 994 996 1006 1009 1011 1026 1034   
## 0.5730990 0.6256743 0.5484479 0.5897647 0.5641313 0.5811525 0.5344270 0.6267960   
## 1035 1038 1041 1046 1051 1052 1109 1112   
## 0.6637275 0.4847748 0.4631213 0.5690219 0.6659051 0.5265418 0.5709335 0.5937840   
## 1120 1121 1123 1124 1136 1137 1142 1149   
## 0.6357305 0.6507565 0.6003690 0.5401801 0.5305690 0.6133902 0.6010585 0.4701215   
## 1150 1151 1159 1161 1169 1176 1180 1188   
## 0.5612185 0.4970265 0.5667711 0.5271228 0.5458286 0.4829674 0.4226253 0.4697908   
## 1201 1204 1208 1280 1282 1322 1326 1327   
## 0.5206640 0.5049895 0.4885094 0.5256752 0.5343010 0.5779149 0.4935691 0.6250419   
## 1330 1335 1336 1339 1344 1362 1369 1371   
## 0.4795769 0.5335516 0.5307903 0.5383626 0.7405960 0.5657228 0.5037284 0.4587793   
## 1377 1386 1403 1409 1414 1415 1419 1425   
## 0.5263675 0.5520148 0.5715306 0.4898438 0.5189030 0.4612959 0.5836837 0.5761212   
## 1449 1464 1469 1472 1477 1480 1500 1501   
## 0.6167402 0.6285917 0.6018882 0.5601137 0.5910858 0.6505047 0.4224522 0.5227024   
## 1509 1511 1515 1516 1517 1519 1595 1624   
## 0.4724805 0.5458638 0.6278720 0.4716032 0.4647029 0.4506659 0.5243106 0.5225684   
## 1636 1640 1651 1660 1668 1679 1689 1701   
## 0.4317117 0.5796273 0.5597923 0.5198639 0.4884230 0.5776750 0.5170347 0.4986385   
## 1704 1709 1712 1719 1722 1723 1726 1728   
## 0.4443348 0.4724803 0.4967961 0.4949846 0.4495989 0.5049830 0.5370279 0.6106939   
## 1735 1739 1743 1745 1747 1754 1761 1763   
## 0.5673151 0.4706124 0.5790456 0.4430368 0.4378230 0.7329311 0.6463384 0.5723610   
## 1767 1769 1774 1778 1783 1788 1789 1802   
## 0.7764922 0.6523523 0.6367757 0.6270603 0.5898682 0.6174722 0.7020385 0.5315546   
## 1804 1806 1811 1861 1883 1890 1899 1908   
## 0.5225138 0.4880970 0.5603537 0.5545564 0.4921644 0.5288601 0.4641301 0.4622420   
## 1913 1918 1922 1930 1931 1932 1941 1944   
## 0.4674721 0.4868576 0.4600574 0.4374394 0.4425480 0.4007444 0.5330951 0.4279960   
## 1947 1950 1953 1955 1956 1959 1962 1966   
## 0.6061485 0.5249272 0.4605022 0.6283891 0.5977370 0.4354471 0.4813546 0.4277145   
## 1967 1970 1976 1983 2023 2024 2026 2030   
## 0.4206168 0.4815606 0.4390756 0.4723950 0.5186872 0.5130414 0.5344487 0.5252627   
## 2044 2060 2069 2074 2079 2081 2086 2095   
## 0.6421391 0.6228489 0.6175097 0.4724762 0.4770957 0.5755997 0.5823887 0.6795732   
## 2098 2104 2109 2118 2123 2129 2130 2139   
## 0.6191835 0.5434105 0.4987413 0.5882241 0.7988125 0.4410854 0.5668089 0.4169033   
## 2143 2149 2158 2164 2177 2190 2200 2209   
## 0.5865278 0.8026972 0.4714874 0.6794165 0.3903037 0.4167845 0.3905699 0.4687106   
## 2212 2217 2222 2240 2268 2270 2272 2273   
## 0.4729091 0.5082453 0.3978132 0.5794222 0.4766285 0.5106798 0.5490262 0.4176897   
## 2276 2279 2280 2296 2297 2307 2310 2311   
## 0.3869878 0.5379652 0.5818365 0.4854526 0.6375644 0.5737883 0.5785323 0.7210596   
## 2319 2327 2333 2334 2339 2352 2355 2356   
## 0.5445662 0.5289888 0.3355855 0.4213856 0.4504512 0.4532704 0.5306103 0.5392177   
## 2366 2376 2379 2384 2401 2403 2409 2413   
## 0.5028137 0.5976863 0.7213363 0.6143900 0.5114992 0.7651196 0.3711010 0.5588167   
## 2423 2436 2439 2440 2448 2452 2470 2472   
## 0.6364356 0.5344319 0.7064819 0.5837508 0.7023736 0.6368622 0.4630792 0.5127422   
## 2476 2491 2492 2527 2532 2540 2580 2584   
## 0.5604015 0.4659476 0.5105583 0.4911955 0.6143580 0.4785323 0.5898931 0.5110811   
## 2586 2596 2602 2604 2606 2616 2618 2621   
## 0.5774470 0.4775726 0.4911406 0.5196752 0.6176120 0.6051777 0.5059835 0.5114412   
## 2645 2666 2680 2685 2689 2694 2703 2705   
## 0.6268921 0.5340481 0.4365513 0.4916265 0.4137326 0.4670518 0.5734370 0.4780119   
## 2755 2766 2770 2772 2781 2783 2785 2793   
## 0.5441679 0.5955830 0.6102827 0.5875898 0.5389863 0.4614342 0.5070080 0.4949929   
## 2803 2805 2809 2826 2832 2837 2843 2852   
## 0.5583114 0.4944996 0.4538105 0.6425713 0.5631386 0.5985160 0.5984881 0.4402621   
## 2861 2863 2890 2893 2905 2913 2923 2953   
## 0.4917635 0.6520468 0.5362376 0.5549516 0.5726348 0.4659642 0.4197040 0.4239156   
## 2959 2966 2973 2981 2982 2983 2984 2986   
## 0.3990383 0.5499155 0.5000198 0.6695343 0.6464156 0.5907082 0.7189227 0.6330536   
## 2988 2991 2993 2995 2998 3001 3003 3004   
## 0.6019445 0.5327203 0.6035045 0.5350451 0.4593920 0.5809211 0.5117288 0.5122771   
## 3005 3008 3034 3043 3054 3057 3062 3065   
## 0.5468123 0.4383070 0.4754477 0.4984650 0.4887594 0.5044013 0.5353097 0.5422050   
## 3066 3067 3069 3073 3081 3091 3102 3116   
## 0.5316035 0.5590234 0.5614198 0.4379684 0.3676091 0.5469666 0.4887733 0.6431187   
## 3121 3123 3124 3127 3131 3134 3135 3137   
## 0.6262584 0.4845143 0.5113565 0.4995282 0.5888246 0.5964547 0.5906921 0.6062427   
## 3138 3140 3147 3149 3152 3158 3162 3164   
## 0.5295723 0.5378709 0.4598373 0.5190518 0.5787107 0.5220114 0.4823551 0.4306389   
## 3169 3181 3184 3186 3188 3195 3199 3202   
## 0.4475411 0.5732390 0.4901819 0.6026046 0.5500054 0.4906697 0.5437163 0.4761424   
## 3203 3210 3214 3221 3222 3223 3227 3238   
## 0.4394946 0.4905265 0.4432111 0.6644413 0.4509522 0.4631898 0.6065972 0.5153064   
## 3248 3249 3257 3261 3263 3265 3285 3299   
## 0.3809291 0.7404645 0.6154322 0.4587491 0.5557927 0.5297207 0.4655794 0.4417760   
## 3300 3304 3310 3313 3319 3322 3325 3343   
## 0.4198761 0.5025696 0.4948839 0.6595698 0.3204123 0.5823249 0.6995879 0.5972918   
## 3351 3354 3360 3361 3364 3375 3394 3398   
## 0.5509809 0.6021994 0.5148346 0.6385132 0.7367348 0.5786091 0.4401489 0.6816288   
## 3401 3404 3413 3458 3463 3467 3486 3492   
## 0.5218877 0.6670378 0.6188490 0.5172763 0.5509759 0.4113548 0.5410749 0.5579732   
## 3505 3507 3510 3512 3513 3515 3518 3566   
## 0.4358309 0.5107619 0.5636349 0.4898957 0.4882903 0.4417683 0.5032848 0.4706050   
## 3570 3579 3580 3583 3591 3595 3598 3610   
## 0.5901346 0.4346015 0.4991872 0.4828160 0.5156170 0.5189007 0.5296409 0.4795014   
## 3612 3622 3624 3628 3659 3686 3690 3692   
## 0.4687075 0.5271402 0.5160271 0.4692551 0.5706479 0.6008714 0.4526569 0.5004955   
## 3697 3704 3735 3743 3759 3766 3772 3779   
## 0.4681792 0.3835287 0.5056709 0.4359453 0.5001617 0.5160898 0.5882169 0.4812123   
## 3795 3796 3799 3808 3809 3816 3819 3820   
## 0.5208712 0.4804974 0.4118447 0.5858498 0.5508009 0.6482418 0.5951192 0.5077813   
## 3822 3829 3844 3847 3848 3853 3862 3868   
## 0.4949996 0.3408611 0.4742620 0.6160725 0.3781463 0.4831165 0.5858568 0.5761080   
## 3872 3875 3882 3883 3888 3891 3901 3904   
## 0.4934753 0.7464401 0.4939006 0.5279107 0.5659952 0.4464124 0.5385359 0.4135352   
## 3918 3925 3935 3939 3981 3983 4007 4008   
## 0.4061556 0.5901502 0.5392558 0.4747058 0.5358285 0.4845482 0.5666536 0.6089447   
## 4009 4012 4034 4045 4050 4054 4056 4057   
## 0.5545772 0.5239072 0.5719627 0.5310833 0.4942870 0.4637584 0.5231526 0.5502692   
## 4079 4092 4094 4097 4100 4130 4139 4141   
## 0.6041857 0.5815889 0.5216834 0.5174304 0.5725112 0.4904486 0.4525099 0.5075944   
## 4144 4145 4146 4162 4168 4176   
## 0.5711530 0.5018296 0.4163783 0.7364711 0.5659206 0.6337948

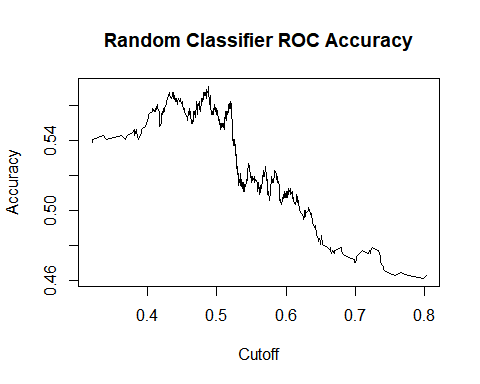
#Convert Sex Probabilities to "M" if >0.5 else "F" and change it to a factor from char  
sex.prob <- ifelse(glm.predict > 0.5, "M", "F")  
sex.prob <- factor(sex.prob)  
  
#Create confusion matrix  
confusionMatrix(sex.prob, test$Sex)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction F M  
## F 108 97  
## M 153 208  
##   
## Accuracy : 0.5583   
## 95% CI : (0.5163, 0.5997)  
## No Information Rate : 0.5389   
## P-Value [Acc > NIR] : 0.1880484   
##   
## Kappa : 0.0973   
##   
## Mcnemar's Test P-Value : 0.0005042   
##   
## Sensitivity : 0.4138   
## Specificity : 0.6820   
## Pos Pred Value : 0.5268   
## Neg Pred Value : 0.5762   
## Prevalence : 0.4611   
## Detection Rate : 0.1908   
## Detection Prevalence : 0.3622   
## Balanced Accuracy : 0.5479   
##   
## '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)



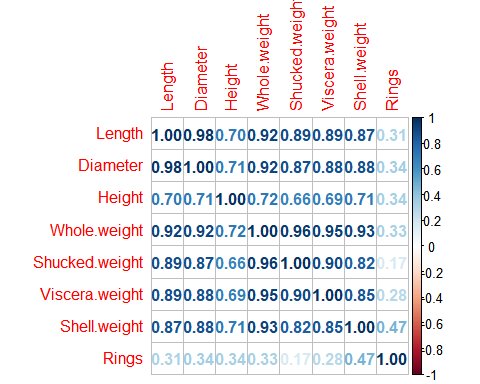
#Measure performance of Random Classifier on Accuracy  
roc.perform <- performance(roc.predict, measure = "acc")  
plot(roc.perform, main="Random Classifier ROC Accuracy")



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 66 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

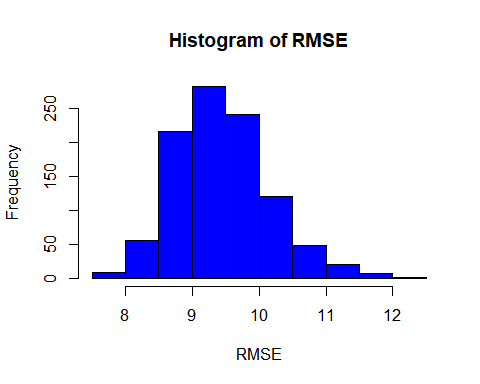
cat(", R^2 - Bootstrap: ", r2\_boot)

## , 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

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.

#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)  
  
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

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 is uniformly (evenly) distributed on 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:

which means the intervals will be and the length will be 0.1 (10%).

10%

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