assignment_11.1_MunjewarSheetal

Sheetal M

2023-02-25

Install and Load required packages :

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE)
knitr::opts_chunk$set(fig.width = 12, fig.height = 10)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 70), tidy = TRUE)

# Package names
# packages <- c("ggplot2","dplyr","tidyr","magrittr","tidyverse","purrr")
packages <- c("broom","dplyr","RWeka","class","ggplot2","caret","formatR")

# Install packages not yet installed
installed_packages <- packages %in% rownames(installed.packages())
if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages])
}

# Packages loading
invisible(lapply(packages, library, character.only = TRUE))
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: lattice
```

Problem statement : Predict one year life expectancy of lung cancer patients post surgery.

Set the working directory to the root of your DSC 520 directory

setwd("E:\Data_Science_DSC510\DSC520-Statistics\dsc520")

```r
## Set the working directory to the root of your DSC 520 directory
setwd("E:\\Data_Science_DSC510\\DSC520-Statistics\\dsc520")

## Load data from data/binary-classifier-data.csv
bc_data <- read.csv("data/binary-classifier-data.csv")
str(bc_data)
```

```
## 'data.frame':    1498 obs. of  3 variables:
##  $ label: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ x    : num  70.9 75 73.8 66.4 69.1 ...
##  $ y    : num  83.2 87.9 92.2 81.1 84.5 ...
```

```r
# nrow(pat_data)
```

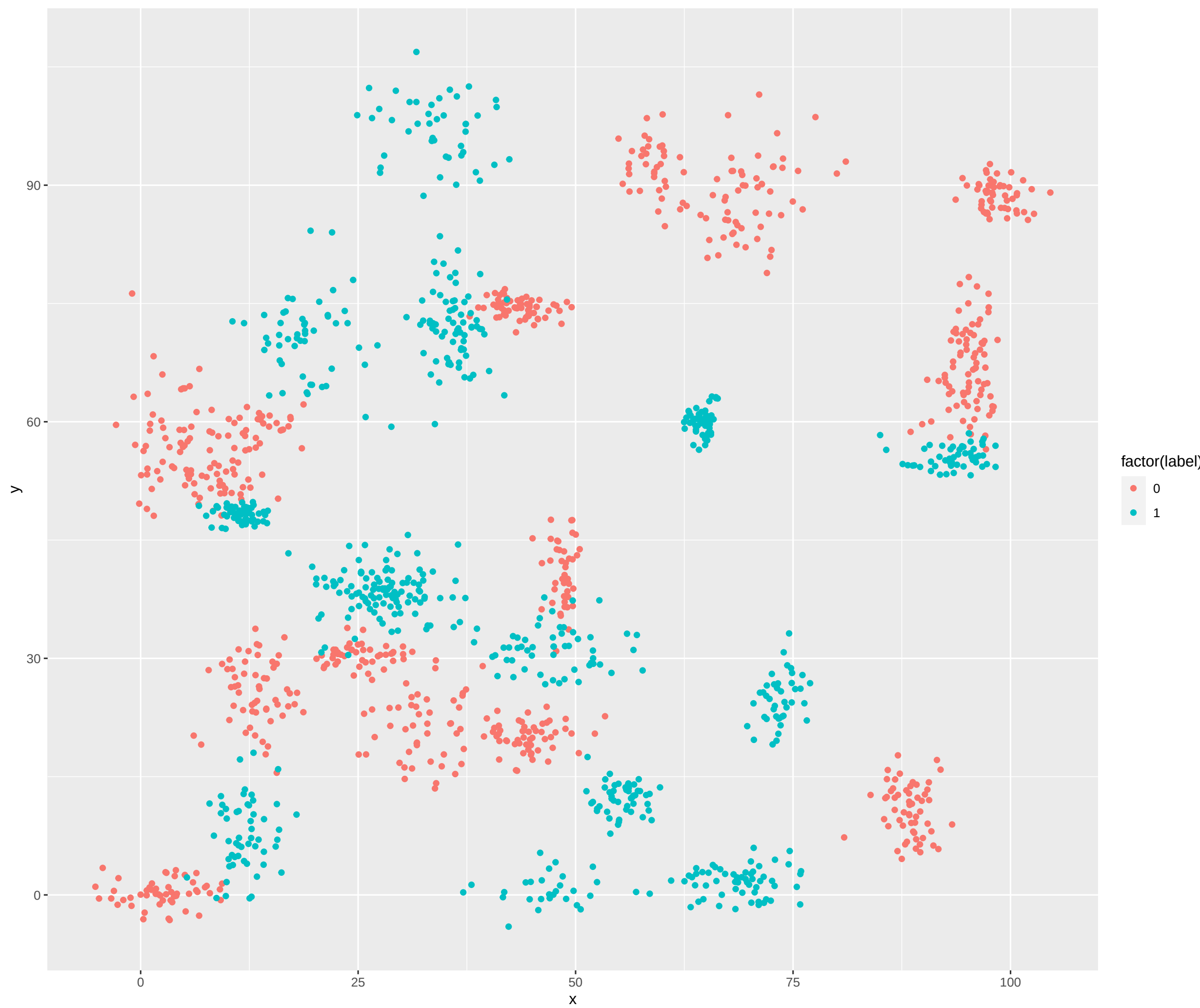
Convert label column data type into factor

```r
bc_data$label <- as.factor(bc_data$label)
str(bc_data)
```

```
## 'data.frame':    1498 obs. of  3 variables:
##  $ label: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ x    : num  70.9 75 73.8 66.4 69.1 ...
##  $ y    : num  83.2 87.9 92.2 81.1 84.5 ...
```

Visualize data

```r
ggplot(data = bc_data, aes(x, y, color = factor(label))) + geom_point()
```



Generalized Linear Model

```
bc_mod01 <- glm(label ~ ., data = bc_data, family = "binomial")
```

Model Summary

```
summary(bc_mod01)
```

```
##
## Call:
## glm(formula = label ~ ., family = "binomial", data = bc_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3728  -1.1697  -0.9575   1.1646   1.3989
##
```

```
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.424809   0.117224   3.624  0.00029 ***
## x           -0.002571   0.001823  -1.411  0.15836
## y           -0.007956   0.001869  -4.257 2.07e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2075.8  on 1497  degrees of freedom
## Residual deviance: 2052.1  on 1495  degrees of freedom
## AIC: 2058.1
##
## Number of Fisher Scoring iterations: 4
```

Variables with significance

1. y is Most Significant

Dataframe with new predicted column predict_Risk

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE)
knitr::opts_chunk$set(fig.width = 12, fig.height = 10)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 70), tidy = TRUE)

# mod_plus <- augment(pat_mod01, type type.predict='response')
# class(mod_plus)
bc_mod01_predict <- augment(bc_mod01, type.predict = "response") %>%
    mutate(predict_Risk = round(.fitted))

# Name additional columns and check class.  class(mod_plus)
# names(mod_plus)
# https://cyberactive.bellevue.edu/ultra/courses/_514803_1/cl/outline
# alternate options using predict function() - predict(bc_mod01, type
# = 'response')
```

Confusion matrix to calculate accurracy

```
bc_mod01_predict %>%
    select(label, predict_Risk) %>%
    table()
```

```
##      predict_Risk
## label   0   1
##     0 429 338
##     1 286 445
```

```
# Alternate option : predict <- predict(logit, data_test, type =
# 'response') table_mat <- table(data_test$income, predict > 0.5)
```

Accuracy of the Model

accuracy = correctly predicted / total Predicted * 100

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE)
knitr::opts_chunk$set(fig.width = 12, fig.height = 10)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 70), tidy = TRUE)


# Evaluating accuracy of Regression Models -
# https://www.youtube.com/watch?v=03FrK8d2QVQ Accuracy using
# confusion matric :
# https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7

accuracy <- (429 + 445)/(429 + 338 + 286 + 445)
accuracy <- accuracy * 100
print(paste(round(accuracy), "%"))
```

```
## [1] "58 %"
```

KNN Nearest neighbors model on actual data set.

```
near_mod <- knn(train = bc_data, test = bc_data, cl = bc_data$label, k = 5)
summary(near_mod)
```

```
##   0   1
## 771 727
```

```
# Calculate the proportion of correct classification for k = 5
acc_mod1 <- 100 * sum(bc_data$label == near_mod)/NROW(bc_data$label)
# acc_mod1

# Calculate accuracy.
table(near_mod, bc_data$label)
```

```
##
## near_mod   0   1
##        0 756  15
##        1  11 716
```

```
confusionMatrix(table(near_mod, bc_data$label))
```

```
## Confusion Matrix and Statistics
##
```

```
##
## near_mod   0   1
##        0 756  15
##        1  11 716
##
##                Accuracy : 0.9826
##                  95% CI : (0.9747, 0.9886)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9653
##
##  Mcnemar's Test P-Value : 0.5563
##
##             Sensitivity : 0.9857
##             Specificity : 0.9795
##          Pos Pred Value : 0.9805
##          Neg Pred Value : 0.9849
##              Prevalence : 0.5120
##          Detection Rate : 0.5047
##    Detection Prevalence : 0.5147
##       Balanced Accuracy : 0.9826
##
##        'Positive' Class : 0
##
```

Get actual false and true labels from dataset

```r
paste("True values count:", length(bc_data$label[bc_data$label == 1]))
```

```
## [1] "True values count: 731"
```

```r
paste("False values count:", length(bc_data$label[bc_data$label == 0]))
```

```
## [1] "False values count: 767"
```

```r
table(bc_data$label)
```

```
##
##   0   1
## 767 731
```

```r
# count_unique <- rapply(bc_data, function(x) length(unique(x)))
# count_unique n_distinct(bc_data$label) bc_data %>%
# group_by(bc_data$label) %>% summarize(count_distinct =
# n_distinct(x)) length(unique(bc_data$label))
```

If we use same data for training and testing the accuracy of the nearest neighbour algorithm is 99.73%

Creating random 80% records for training set and 20% test set

```
ran <- sample(1:nrow(bc_data), 0.8 * nrow(bc_data))
train_data <- bc_data[ran, ]
test_data <- bc_data[-ran, ]
test_data
```

```
##      label           x           y
## 5        0 69.07399466 84.53738605
## 15       0 70.70260446 86.51170828
## 24       0 66.26135349 90.76869718
## 35       0 68.49424528 82.44224160
## 42       0 72.37433853 80.93594197
## 43       0 70.79669662 90.86586943
## 46       0 67.52768826 98.87718070
## 50       0 68.69495698 89.33047854
## 57       0 49.05763896 38.49490904
## 62       0 48.40125766 42.33842638
## 67       0 49.09991050 40.07400160
## 73       0 49.64304961 45.91976022
## 88       0 46.09650929 36.21655624
## 89       0 49.22916143 39.46245715
## 96       0 48.83413052 41.56444321
## 97       0 49.62158797 47.53151307
## 98       0 49.51866513 47.50066326
## 106      0 40.73934421 76.31974656
## 108      0 41.50492569 74.40347216
## 117      0 38.81479800 74.48936759
## 123      0 45.53569945 74.56817305
## 127      0 45.23098122 72.24428396
## 129      0 41.10078814 75.48428201
## 143      0 41.29851767 76.06712165
## 147      0 41.78333686 75.13428004
## 148      0 46.88954708 74.09267686
## 157      0 41.76045012 74.14467130
## 161      0 96.65315399 64.07243515
## 162      0 91.73866637 65.17427668
## 165      0 98.05995692 61.90060000
## 166      0 93.29748020 63.86145848
## 168      0 94.04884260 74.09198065
## 172      0 95.08437281 61.99645529
## 173      0 93.80638935 62.02356894
## 175      0 96.20467275 61.62066547
## 184      0 90.88731223 60.03321961
## 185      0 92.89140665 61.51275912
## 191      0 92.94050645 64.47822890
## 198      0 17.21583287 60.59492993
## 211      0 18.72656269 62.19570237
## 212      0 10.06634646 58.24107519
## 214      0 12.22254537 61.86700122
## 217      0 16.10027448 58.89830844
## 218      0 18.53279336 56.63622682
## 220      0 16.99892211 59.41882178
```

```
## 222       0 16.23924965  58.95575102
## 225       0 57.94094074  96.27850571
## 234       0 56.09877176  92.11809090
## 237       0 62.02365107  86.93962712
## 240       0 61.99968498  93.54841379
## 249       0 59.91063750  88.29815950
## 250       0 56.16402065  91.40711495
## 251       0 59.78274871  92.68129416
## 255       0 59.04329093  91.70828497
## 265       0 47.37082510  18.65221668
## 267       0 43.53921368  19.21951427
## 269       0 46.19907652  21.70881897
## 270       0 46.67619507  23.85622174
## 271       0 50.36837350  17.98462671
## 273       0 44.52045009  23.12917483
## 276       0 44.76884548  19.70530032
## 277       0 44.91986574  17.94112767
## 282       0 43.02296335  19.12716594
## 285       0 47.68761418  20.60782734
## 287       0 44.33352230  19.06727653
## 301       0 41.24423961  19.51689514
## 302       0 53.38618429  22.65660802
## 309       0 47.16314215  20.00655965
## 319       0  8.47001310  54.74680953
## 325       0  5.41315794  57.52632016
## 332       0 15.80034817  50.24626519
## 333       0  1.51235170  48.07708322
## 338       0  5.65833721  57.90455752
## 346       0  4.87056544  56.81682753
## 349       0  5.10927056  51.94005431
## 351       0 13.23402268  56.72119601
## 354       0  8.06346011  51.55280970
## 359       0 10.57975853  55.07927312
## 378       0 91.54113813  17.10736325
## 379       0 89.92923135   7.21404147
## 383       0 89.62541829   5.41251891
## 384       0 85.60741272  12.27407061
## 386       0 87.49729969   4.56805940
## 387       0 89.80260490  11.93087229
## 389       0 86.49674763  13.51062424
## 392       0 87.74017400  10.47434900
## 395       0 89.19583549  13.98633967
## 398       0 90.90199217   8.06430947
## 405       0 88.02435731  13.26384475
## 407       0 88.33010848  12.86628493
## 408       0 90.61125033  14.28065667
## 409       0 86.69410275  12.32419526
## 415       0 91.70778582   5.81445961
## 419       0 91.13453661   6.26027555
## 425       0 88.35228055  11.82856044
## 426       0 89.15473296   7.05867396
## 427       0 91.95867083  15.88338021
## 428       0 28.18654434  30.45362210
## 430       0 22.41461761  31.09312418
```

```
## 431       0 15.87470619  30.38069984
## 437       0 24.50570128  27.82143918
## 439       0 15.64557866  29.30598975
## 448       0 30.22668016  29.90242873
## 453       0 22.66853285  30.74228219
## 455       0 22.48931596  30.65772126
## 456       0 33.93771118  29.73940110
## 462       0 23.73767102  31.37838682
## 469       0 20.21827594  29.94228948
## 472       0 21.41601549  29.28951309
## 478       0 23.03827988  30.46224181
## 488       0  2.55070714  -0.69867277
## 493       0  3.21951100  -3.01778398
## 502       0  1.02125821   0.94381330
## 505       0  3.63154720  -0.91895992
## 507       0 -3.07239409   0.50857131
## 511       0 -1.16405737  -0.35276467
## 521       0  0.45735269  -2.24268442
## 523       0  2.92727325   0.08262864
## 525       0 -4.36036923   3.42470093
## 528       0  9.19787040  -0.65762944
## 546       0 96.49704426  72.97658376
## 549       0 95.70067908  66.61221520
## 553       0 96.99518741  70.25635965
## 555       0 94.49028626  71.11950923
## 556       0 95.14770616  75.01668607
## 559       0 97.17904084  56.50926060
## 567       0 94.22574245  68.42965139
## 569       0 95.60288884  72.36184949
## 581       0 31.14696384  25.08131541
## 584       0 35.59084299  21.71793270
## 588       0 26.59972263  27.27039954
## 597       0 32.90131716  24.79506286
## 600       0 26.60778396  23.51591342
## 602       0 32.96125313  20.47580828
## 613       0 31.30881281  21.47887490
## 615       0 29.76314913  16.74529648
## 617       0 39.32171190  29.01503806
## 621       0 37.14716981  18.50940082
## 623       0 33.97431259  14.16459394
## 628       0 17.70576225  24.18011227
## 630       0 10.39580774  26.34712306
## 632       0 16.32128104  22.72744618
## 638       0 10.20066212  22.17754895
## 640       0 12.79587994  24.10850757
## 643       0 11.20068627  26.57738411
## 653       0 10.56039882  28.65929968
## 661       0 13.38954071  24.61168378
## 669       0 18.70815222  23.17206609
## 677       0 16.83712841  26.06166208
## 680       0  5.06863594  64.23869032
## 681       0  1.39988803  60.91771538
## 693       0  0.33579701  56.29955201
## 698       0  1.05634706  58.87173294
```

```
## 711       0  5.55303869  52.79678989
## 712       0  0.05664209  53.22522797
## 719       0 97.15483392  89.27348562
## 735       0 97.17793501  86.41514506
## 736       0 97.74561782  89.95400416
## 745       0 96.66502955  87.51776759
## 756       0 96.19806684  89.46055431
## 759       0 96.41364346  89.98631184
## 763       0 93.69239067  88.16739658
## 764       0 97.75175780  87.94686896
## 771       1 15.93545995  69.67978739
## 774       1 18.59967568  73.03604627
## 779       1 25.86052384  60.59903216
## 785       1 19.18852149  63.52369978
## 791       1 22.45279376  72.48513234
## 792       1 20.86259977  64.41186627
## 800       1 14.20083843  73.54088374
## 801       1 19.53375500  64.70159286
## 811       1 18.85407971  70.22527230
## 814       1 21.32619508  64.52262286
## 818       1 24.42891713  77.98285137
## 824       1 50.27384132  32.45098277
## 827       1 44.97185160  30.34563137
## 830       1 51.59551870  29.12021653
## 832       1 45.06228464  31.41524349
## 833       1 41.02634554  27.75110747
## 843       1 42.35260295  31.37471811
## 850       1 51.99298808  30.98932149
## 852       1 48.20528563  33.94875184
## 863       1 42.09331884  29.79278089
## 874       1 36.69897050  34.59910543
## 879       1 28.90251404  39.56579367
## 880       1 23.39439121  41.17560290
## 896       1 28.33624866  41.44302288
## 901       1 26.36353607  36.24966052
## 902       1 29.18288517  38.41622001
## 914       1 20.17610713  39.38207340
## 918       1 28.81566170  37.98407307
## 919       1 29.32031713  37.49383903
## 922       1 28.22355208  42.46212898
## 935       1 96.02717669  54.85634157
## 937       1 94.60247498  54.32922695
## 943       1 96.83491230  55.71979201
## 949       1 93.98160699  55.86160074
## 962       1 88.19376438  54.49047543
## 964       1 93.47269367  53.50778987
## 966       1 96.75114292  54.30669304
## 970       1 97.27697059  54.63290690
## 979       1 98.27507781  56.95674991
## 982       1 93.25403284  54.57572978
## 985       1 93.05207646  54.84388426
## 987       1 27.81166955  34.42186806
## 1000      1 21.34140809  39.07418594
## 1004      1 24.62652851  32.47112864
```

```
## 1005      1 22.96825562  39.92196948
## 1006      1 25.86928800  39.00743550
## 1020      1 21.12916922  40.19710657
## 1038      1 23.84480021  35.15613797
## 1042      1 64.90073720  57.03805909
## 1043      1 65.03021092  60.80807148
## 1054      1 62.99340913  61.37024545
## 1065      1 63.54432967  57.04007187
## 1072      1 65.13604097  57.63907115
## 1076      1 63.54003603  60.32573067
## 1082      1 65.24813914  58.58057184
## 1083      1 64.91567158  59.49662587
## 1086      1 64.18635007  60.31120892
## 1092      1 63.78249411  59.51852145
## 1093      1 65.10064134  59.69382884
## 1095      1 63.87793339  61.73995284
## 1100      1 76.58258201  22.11953409
## 1104      1 70.45215238  24.29115922
## 1107      1 73.26717685  26.82285281
## 1108      1 74.85107292  26.87011598
## 1115      1 73.06887539  22.63192833
## 1121      1 73.89894884  22.71364468
## 1130      1 74.34054736  29.11183003
## 1156      1 47.03188634  -0.41801570
## 1157      1 37.08732167   0.32195510
## 1158      1 41.79892920   0.35604284
## 1163      1 46.11487372   1.84970293
## 1171      1 71.04023155  -0.88122250
## 1184      1 71.61854986   2.30215669
## 1189      1 66.18768785   1.76078027
## 1196      1 70.58868184   0.32421449
## 1201      1 65.28640433   2.81367871
## 1214      1 72.88016493   1.13945683
## 1217      1 72.59230537   1.77562393
## 1218      1 75.42816434   1.01919773
## 1219      1 71.10504130   3.64848244
## 1220      1 70.39854516   2.95064659
## 1226      1 51.81772421  11.62618521
## 1227      1 56.11841504  13.57884193
## 1235      1 58.74552758   9.47876721
## 1239      1 51.24912728  13.14713360
## 1241      1 56.36985805  10.53622060
## 1242      1 58.51007626  12.80975318
## 1247      1 55.00846114   9.52886586
## 1257      1 56.37220422  12.25005777
## 1258      1 54.13972814  12.49695982
## 1259      1 56.85062099  12.64176828
## 1271      1 40.05909477  66.43116632
## 1282      1 32.36363886  75.37630537
## 1293      1 33.91824652  72.34319699
## 1295      1 42.12163739  75.49732005
## 1297      1 33.99504696  78.84271864
## 1298      1 34.43728672  76.05441413
## 1299      1 36.09060119  75.37898538
```

```
## 1301     1 35.46057652  67.54107831
## 1306     1 36.48567577  71.57324633
## 1307     1 34.95312978  71.39246069
## 1312     1 36.17819360  78.87509494
## 1316     1 38.64079780  72.09318597
## 1319     1 37.85814285  65.49042686
## 1321     1 36.22328820  77.61583982
## 1323     1 39.03286775  78.72882916
## 1327     1 36.40245523  71.61744123
## 1330     1 35.08658406  75.20296842
## 1331     1 36.21435294  71.14785637
## 1342     1 35.62992454  67.22166058
## 1357     1 11.39534575  47.45663215
## 1361     1  9.92336961  48.01134509
## 1367     1 11.37337338  48.73124436
## 1377     1 14.14065577  48.58071691
## 1383     1 12.90352657  49.81468125
## 1389     1 12.08934524  46.99940463
## 1393     1 11.93485836  48.48102882
## 1395     1  7.53779270  48.07708005
## 1396     1  9.61536527  48.21330302
## 1398     1  9.34583149  46.52648745
## 1404     1 10.11139532   4.53952041
## 1408     1 10.49869981   5.05621008
## 1417     1  9.89957384   9.67960943
## 1418     1 12.98179465  10.21373757
## 1425     1 13.55622919   7.01146303
## 1427     1 17.92171886  10.18767884
## 1438     1 12.39806662   6.47687229
## 1452     1  7.91965352  11.58614027
## 1453     1  9.88185363   1.63145590
## 1461     1 33.43578095 100.18689320
## 1464     1 27.43005766  99.66445614
## 1469     1 35.54688221 102.10574242
## 1482     1 32.51855795  88.65248384
## 1487     1 35.40184709  93.48607792
## 1489     1 28.88575919  98.25351827
## 1496     1 38.99738680  90.57554585
## 1498     1 33.47046269  95.61268248
```

```
nrow(train_data)
```

```
## [1] 1198
```

```
nrow(test_data)
```

```
## [1] 300
```

Train model on trainig data set

```r
knn_mod <- knn(train = train_data, test = test_data, cl = train_data$label,
    k = 5)
summary(knn_mod)
```

```
##   0   1
## 161 139
```

```r
# Calculate the proportion of correct classification for k = 5
acc_mod <- 100 * sum(test_data$label == knn_mod)/NROW(test_data$label)
acc_mod
```

```
## [1] 97.33333
```

```r
# Calculate accuracy.
table(knn_mod, test_data$label)
```

```
##
## knn_mod   0   1
##       0 158   3
##       1   5 134
```

```r
confusionMatrix(table(knn_mod, test_data$label))
```

```
## Confusion Matrix and Statistics
##
##
## knn_mod   0   1
##       0 158   3
##       1   5 134
##
##                Accuracy : 0.9733
##                  95% CI : (0.9481, 0.9884)
##     No Information Rate : 0.5433
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9463
##
##  Mcnemar's Test P-Value : 0.7237
##
##             Sensitivity : 0.9693
##             Specificity : 0.9781
##          Pos Pred Value : 0.9814
##          Neg Pred Value : 0.9640
##              Prevalence : 0.5433
##          Detection Rate : 0.5267
##    Detection Prevalence : 0.5367
##       Balanced Accuracy : 0.9737
##
##        'Positive' Class : 0
##
```

NROW(test_data) NROW(test_data$label)
```

### Get actual false and true labels from test dataset

```
paste("True values count:", length(test_data$label[test_data$label == 1]))
```

```
## [1] "True values count: 137"
```

```
paste("False values count:", length(test_data$label[test_data$label ==
    0]))
```

```
## [1] "False values count: 163"
```

If we use same 80% dataset to train and 20% dataset to test the accuracy of the nearest neighbour algorithm is 99.33%

Accuracy Comparison

- Logistic Regression: 58%
- Knn with 100% training and test data: 98.26%
- Knn with 80% training and 20% test data: 97:00%

Reason for Accuracy difference

Based on derived values, KNN algorithm accuracy is more in comparison with logistic regression. Obvious reason for difference in accuracy will be KNN is more efficient for distinguishable closed clustered, as showed in scatter plot above.
```