# DATA 622: Homework 01

Shovan Biswas

10/15/2020

## Load input data dataset.csv

```
mydata <- read.csv('./dataset.csv', head = TRUE, sep = ',', stringsAsFactors = TRUE)
head(mydata)
```

```
##   X Y label
## 1 5 a  BLUE
## 2 5 b BLACK
## 3 5 c  BLUE
## 4 5 d BLACK
## 5 5 e BLACK
## 6 5 f BLACK
```

Converting variable X to factor.

```
mydata$X <- factor(mydata$X)
```

## Data exploration.

```
print(paste0("Number of observations: ", dim(mydata)[1], "   Number of columns: ", dim(mydata)[2]))
```

```
## [1] "Number of observations: 36   Number of columns: 3"
```

Ratio of BLACK to BLUE in response variable label. The data is somewhat imbalanced.

```
table(mydata$label)
```

```
##
## BLACK  BLUE
##    22    14
```

```
summary(mydata)
```

```
##    X      Y       label
##   5 :6   a:6   BLACK:22
##   19:6   b:6   BLUE :14
##   35:6   c:6
##   51:6   d:6
##   55:6   e:6
##   63:6   f:6
```

```r
xtabs(~label + X, data = mydata)
```

```
##        X
## label   5 19 35 51 55 63
##   BLACK 4  1  5  5  6  1
##   BLUE  2  5  1  1  0  5
```

```r
xtabs(~label + Y, data = mydata)
```

```
##        Y
## label   a b c d e f
##   BLACK 4 4 1 4 5 4
##   BLUE  2 2 5 2 1 2
```

# Logistic Regression on entire dataset.

## Executing Logistic Regression model i.e. glm() function.

```r
lr_glm_model <- glm(label~., data = mydata, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
lr_glm_model
```

```
##
## Call:  glm(formula = label ~ ., family = "binomial", data = mydata)
##
## Coefficients:
## (Intercept)          X19          X35          X51          X55
##  -1.151e+00    3.141e+00   -1.925e+01   -1.925e+01   -5.769e+01
##         X63           Yb           Yc           Yd           Ye
##   3.141e+00    4.103e-15    3.971e+01    1.726e-15   -2.068e+00
##          Yf
##   2.001e-15
##
## Degrees of Freedom: 35 Total (i.e. Null);  25 Residual
## Null Deviance:       48.11
## Residual Deviance: 13.38     AIC: 35.38
```

```r
summary(lr_glm_model)
```

```
##
## Call:
## glm(formula = label ~ ., family = "binomial", data = mydata)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.05849  -0.00005   0.00000   0.12658   1.68851
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.151e+00  1.792e+00  -0.642   0.5209
## X19          3.141e+00  1.757e+00   1.787   0.0739 .
## X35         -1.925e+01  7.142e+03  -0.003   0.9978
## X51         -1.925e+01  7.142e+03  -0.003   0.9978
## X55         -5.769e+01  1.232e+04  -0.005   0.9963
## X63          3.141e+00  1.757e+00   1.787   0.0739 .
## Yb           4.103e-15  2.253e+00   0.000   1.0000
## Yc           3.971e+01  8.768e+03   0.005   0.9964
## Yd           1.726e-15  2.253e+00   0.000   1.0000
## Ye          -2.068e+00  2.183e+00  -0.947   0.3436
## Yf           2.001e-15  2.253e+00   0.000   1.0000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 48.114  on 35  degrees of freedom
## Residual deviance: 13.385  on 25  degrees of freedom
## AIC: 35.385
##
## Number of Fisher Scoring iterations: 20
```

**Now let us compute the performance of the classifier. We know that random choice would have scored 0.5. If our model is any good it must perform better than 0.5.**

**Prediction and generation of Confusion Matrix.**

```r
lr_prediction <- predict(lr_glm_model, newdata = mydata[,1:2], type = 'response')
lr_prediction_labels <- ifelse(lr_prediction > 0.5, "BLUE", "BLACK")
table(lr_prediction_labels)
```

```
## lr_prediction_labels
## BLACK  BLUE
##    23    13
```

```r
table(mydata$label)
```

```
##
## BLACK  BLUE
##    22    14
```

### Computation of confusion matrix, with table() function.

```r
lr_confusion_matrix <- table(mydata$label, lr_prediction_labels)

# Gathering parts of confusion matrix, for later use:
lr_TP = lr_confusion_matrix[1, 1]
lr_FP = lr_confusion_matrix[1, 2]
lr_FN = lr_confusion_matrix[2, 1]
lr_TN = lr_confusion_matrix[2, 2]

lr_confusion_matrix
```

```
##        lr_prediction_labels
##         BLACK BLUE
##   BLACK    21    1
##   BLUE      2   12
```

### Computation of Accuracy parameter from confusion matrix.

```r
lr_accuracy <- sum(diag(lr_confusion_matrix)) / sum(lr_confusion_matrix) * 100
lr_accuracy
```

```
## [1] 91.66667
```

### Verifying computation of confusion matrix, with caret::confusionMatrix function.

```r
caret::confusionMatrix(table(mydata$label, lr_prediction_labels))
```

```
## Confusion Matrix and Statistics
##
##        lr_prediction_labels
##         BLACK BLUE
##   BLACK    21    1
##   BLUE      2   12
##
##                Accuracy : 0.9167
##                  95% CI : (0.7753, 0.9825)
##     No Information Rate : 0.6389
```

```
##     P-Value [Acc > NIR] : 0.0001496
##
##                   Kappa : 0.8224
##
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.9130
##             Specificity : 0.9231
##          Pos Pred Value : 0.9545
##          Neg Pred Value : 0.8571
##              Prevalence : 0.6389
##          Detection Rate : 0.5833
##    Detection Prevalence : 0.6111
##       Balanced Accuracy : 0.9181
##
##        'Positive' Class : BLACK
##
```

So, with both methods (i.e. with table() function and caret::confusionMatrix() function) Accuracy is 0.9167 = 91.67%, which is greater than 70%.

**A model with a accuracy of 70% or higher is perofrmant, and therefore is not underfitting. So, we conclude that our model is capable of learning.**

# =============== Having computed for the entire dataset, now we'll proceed to split the data and do necessary computations.

# Split Data.

```
set.seed(43)
mydata_train_index <- sample(1:nrow(mydata), 0.30 * nrow(mydata), replace = F)
mydata_train <- mydata[-mydata_train_index, ]
mydata_test <- mydata[mydata_train_index, ]
```

Ratio of BLACK to BLUE in response variable label in train data. The data is imbalanced.

```
table(mydata_train$label)
```

```
##
## BLACK  BLUE
##    14    12
```

Ratio of BLACK to BLUE in response variable label in test data. The data is almost balanced.

```
table(mydata_test$label)
```

```
##
## BLACK  BLUE
##     8     2
```

## Logistic Regression on training dataset.

```
lr_glm_model_train <- glm(label~., data = mydata_train, family = 'binomial')
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
lr_glm_model_train
```

```
##
## Call:  glm(formula = label ~ ., family = "binomial", data = mydata_train)
##
## Coefficients:
## (Intercept)          X19          X35          X51          X55
##      24.486       48.494       -1.345      -48.997      -50.676
##         X63           Yb           Yc           Yd           Ye
##      96.458      -48.063        1.302      -48.063      -96.767
##          Yf
##     -48.846
##
## Degrees of Freedom: 25 Total (i.e. Null);  15 Residual
## Null Deviance:       35.89
## Residual Deviance: 6.019e-10     AIC: 22
```

```
summary(lr_glm_model_train)
```

```
##
## Call:
## glm(formula = label ~ ., family = "binomial", data = mydata_train)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -9.662e-06  -3.504e-06  -2.110e-08   2.671e-06   8.130e-06
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    24.486 119185.310       0        1
## X19            48.494 137823.692       0        1
## X35            -1.345 146745.399       0        1
## X51           -48.997 174511.434       0        1
## X55           -50.676 184375.278       0        1
## X63            96.458 196216.153       0        1
## Yb            -48.063 199390.646       0        1
## Yc              1.302 175851.278       0        1
## Yd            -48.063 199390.646       0        1
## Ye            -96.767 196978.959       0        1
## Yf            -48.846 156460.438       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 3.5890e+01  on 25  degrees of freedom
## Residual deviance: 6.0191e-10  on 15  degrees of freedom
## AIC: 22
##
## Number of Fisher Scoring iterations: 25
```

```
lr_prediction_train <- predict(lr_glm_model_train, newdata = mydata_train[,1:2], type = 'response')
lr_prediction_labels_train <- ifelse(lr_prediction_train > 0.5, "BLUE", "BLACK")
table(lr_prediction_labels_train)
```

```
## lr_prediction_labels_train
## BLACK  BLUE
##    14    12
```

```
table(mydata_train$label)
```

```
##
## BLACK  BLUE
##    14    12
```

### Computation of confusion matrix for train data, with table() function.

```
lr_confusion_matrix_train <- table(mydata_train$label, lr_prediction_labels_train)

# Gathering parts of confusion matrix, for later use:
lr_TP_train = lr_confusion_matrix_train[1, 1]
lr_FP_train = lr_confusion_matrix_train[1, 2]
lr_FN_train = lr_confusion_matrix_train[2, 1]
lr_TN_train = lr_confusion_matrix_train[2, 2]

lr_confusion_matrix_train
```

```
##        lr_prediction_labels_train
##         BLACK BLUE
##   BLACK    14    0
##   BLUE      0   12
```

### Computation of Accuracy parameter from confusion matrix for train data.

```
lr_accuracy_train <- sum(diag(lr_confusion_matrix_train)) / sum(lr_confusion_matrix_train) * 100
lr_accuracy_train
```

```
## [1] 100
```

### Verifying computation of confusion matrix for train data, with caret::confusionMatrix function.

```
caret::confusionMatrix(table(mydata_train$label, lr_prediction_labels_train))
```

```
## Confusion Matrix and Statistics
##
##        lr_prediction_labels_train
##         BLACK BLUE
##   BLACK    14    0
##   BLUE      0   12
##
##                Accuracy : 1
##                  95% CI : (0.8677, 1)
##     No Information Rate : 0.5385
##     P-Value [Acc > NIR] : 1.023e-07
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5385
##          Detection Rate : 0.5385
##    Detection Prevalence : 0.5385
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : BLACK
##
```

**So, with both methods Accuracy is 100%, the model is able to learn.**

Now, I'll compile together all the required information for train data i.e. AUC, ACCURACY, TPR, FPR, TNR, FNR.

We have already computed Accuracy in lr_accuracy_train.

### Computation of AUC for train data.

```
lr_AUC_train <- prediction(lr_prediction_train, mydata_train$label)
lr_AUC_train <- performance(lr_AUC_train, 'auc')
lr_AUC_train <- lr_AUC_train@y.values[[1]]
```

### Computation of TPR, FPR, TNR, FNR.

```
lr_TPR_train <- lr_TP_train / (lr_TP_train + lr_FN_train) * 100
lr_FPR_train <- lr_FP_train / (lr_FP_train + lr_TN_train) * 100
lr_FNR_train <- 100 - lr_TPR_train
lr_TNR_train <- 100 - lr_FPR_train
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
lr_AUC_ACCURACY_TPR_FPR_TNR_FNR_train <- c('LR_train', lr_AUC_train, round(lr_accuracy_train, 2), round
lr_AUC_ACCURACY_TPR_FPR_TNR_FNR_train
```

```
## [1] "LR_train" "1"        "100"      "100"      "0"        "100"
## [7] "0"
```

## =============== Having computed for the train dataset, now we'll compute for the test data and do necessary computations.

Ratio of BLACK to BLUE in response variable label in test data. The data is imbalanced.

```
table(mydata_test$label)
```

```
##
## BLACK  BLUE
##     8     2
```

## Logistic Regression on testing dataset.

```
lr_prediction_test <- predict(lr_glm_model_train, newdata = mydata_test[,1:2], type = 'response')
lr_prediction_labels_test <- ifelse(lr_prediction_test > 0.5, "BLUE", "BLACK")
table(lr_prediction_labels_test)
```

```
## lr_prediction_labels_test
## BLACK  BLUE
##     7     3
```

```
table(mydata_test$label)
```

```
##
## BLACK  BLUE
##     8     2
```

### Computation of confusion matrix for test data, with table() function.

```r
lr_confusion_matrix_test <- table(mydata_test$label, lr_prediction_labels_test)

# Gathering parts of confusion matrix, for later use:
lr_TP_test = lr_confusion_matrix_test[1, 1]
lr_FP_test = lr_confusion_matrix_test[1, 2]
lr_FN_test = lr_confusion_matrix_test[2, 1]
lr_TN_test = lr_confusion_matrix_test[2, 2]

lr_confusion_matrix_test
```

```
##        lr_prediction_labels_test
##         BLACK BLUE
##   BLACK     6    2
##   BLUE      1    1
```

### Computation of Accuracy parameter from confusion matrix for test data.

```r
lr_accuracy_test <- sum(diag(lr_confusion_matrix_test)) / sum(lr_confusion_matrix_test) * 100
lr_accuracy_test
```

```
## [1] 70
```

### Verifying computation of confusion matrix for test data, with caret::confusionMatrix function.

```r
caret::confusionMatrix(table(mydata_test$label, lr_prediction_labels_test))
```

```
## Confusion Matrix and Statistics
##
##        lr_prediction_labels_test
##         BLACK BLUE
##   BLACK     6    2
##   BLUE      1    1
##
##                Accuracy : 0.7
##                  95% CI : (0.3475, 0.9333)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.6496
##
##                   Kappa : 0.2105
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.8571
##             Specificity : 0.3333
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.5000
```

```
##               Prevalence : 0.7000
##           Detection Rate : 0.6000
##     Detection Prevalence : 0.8000
##        Balanced Accuracy : 0.5952
##
##         'Positive' Class : BLACK
##
```

So, with both methods Accuracy is 0.92 = 92%

Now, I'll compile together all the required information for test data i.e. AUC, ACCURACY, TPR, FPR, TNR, FNR.

We have already computed Accuracy in lr_accuracy_test.

### Computation of AUC for test data.

```
lr_AUC_test <- performance(prediction(lr_prediction_test, mydata_test$label), 'auc')
lr_AUC_test <- lr_AUC_test@y.values[[1]]
```

### Computation of TPR, FPR, TNR, FNR.

```
lr_TPR_test <- lr_TP_test / (lr_TP_test + lr_FN_test) * 100
lr_FPR_test <- lr_FP_test / (lr_FP_test + lr_TN_test) * 100
lr_FNR_test <- 100 - lr_TPR_test
lr_TNR_test <- 100 - lr_FPR_test
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
lr_AUC_ACCURACY_TPR_FPR_TNR_FNR_test <- c('LR_test', lr_AUC_test, round(lr_accuracy_test, 2), round(lr_
lr_AUC_ACCURACY_TPR_FPR_TNR_FNR_test
```

```
## [1] "LR_test" "0.84375" "70"      "85.71"   "66.67"   "33.33"   "14.29"
```

# =============== Having computed for the LR algorithm, now we'll compute for the Naive Bayes.

## Naive Bayes on entire dataset.

### Executing naiveBayes().

```r
nb_naiveBayes_model <- naiveBayes(label~., data = mydata)
nb_naiveBayes_model
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     BLACK      BLUE
## 0.6111111 0.3888889
##
## Conditional probabilities:
##        X
## Y                5         19         35         51         55         63
##   BLACK 0.18181818 0.04545455 0.22727273 0.22727273 0.27272727 0.04545455
##   BLUE  0.14285714 0.35714286 0.07142857 0.07142857 0.00000000 0.35714286
##
##        Y
## Y                a          b          c          d          e          f
##   BLACK 0.18181818 0.18181818 0.04545455 0.18181818 0.22727273 0.18181818
##   BLUE  0.14285714 0.14285714 0.35714286 0.14285714 0.07142857 0.14285714
```

```r
summary(nb_naiveBayes_model)
```

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

### Prediction and generation of Confusion Matrix.

```r
nb_prediction <- predict(nb_naiveBayes_model, mydata)
nb_confusion_matrix <- table(nb_prediction, mydata$label)
nb_confusion_matrix
```

```
##
## nb_prediction BLACK BLUE
##         BLACK    20    1
##         BLUE      2   13
```

```r
nb_accuracy <- sum(diag(nb_confusion_matrix)) / sum(nb_confusion_matrix)
nb_accuracy
```

```
## [1] 0.9166667
```

The Accuracy agrees with what we computed with LR.

**Since this is greater than 70%, this again affirms the point that the model is not underfitting, and therefore capable of learning.**

# Naive Bayes on training dataset.

```
nb_naiveBayes_model_train <- naiveBayes(label~., data = mydata_train)
nb_naiveBayes_model_train
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     BLACK      BLUE
## 0.5384615 0.4615385
##
## Conditional probabilities:
##        X
## Y                5         19         35         51         55         63
##   BLACK 0.07142857 0.07142857 0.28571429 0.28571429 0.28571429 0.00000000
##   BLUE  0.16666667 0.33333333 0.08333333 0.00000000 0.00000000 0.41666667
##
##        Y
## Y                a          b          c          d          e          f
##   BLACK 0.14285714 0.14285714 0.07142857 0.14285714 0.28571429 0.21428571
##   BLUE  0.08333333 0.16666667 0.33333333 0.16666667 0.08333333 0.16666667
```

```
summary(nb_naiveBayes_model_train)
```

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

## Computation of confusion matrix for train data, with table() function.

```
nb_prediction_train <- predict(nb_naiveBayes_model_train, mydata_train)
nb_confusion_matrix_train <- table(nb_prediction_train, mydata_train$label)

# Gathering parts of confusion matrix, for later use:
nb_TP_train = nb_confusion_matrix_train[1, 1]
```

```
nb_FP_train = nb_confusion_matrix_train[1, 2]
nb_FN_train = nb_confusion_matrix_train[2, 1]
nb_TN_train = nb_confusion_matrix_train[2, 2]

nb_confusion_matrix_train
```

```
##
## nb_prediction_train BLACK BLUE
##               BLACK    12    0
##               BLUE      2   12
```

### Computation of Accuracy parameter from confusion matrix.

```
nb_accuracy_train <- sum(diag(nb_confusion_matrix_train)) / sum(nb_confusion_matrix_train)
nb_accuracy_train
```

```
## [1] 0.9230769
```

### Computation of AUC for train data.

```
nb_AUC_train <- roc(mydata_train$label, as.numeric(nb_prediction_train))
nb_AUC_train <- nb_AUC_train$auc
nb_AUC_train
```

```
## Area under the curve: 0.9286
```

### Computation of TPR, FPR, TNR, FNR.

```
nb_TPR_train <- nb_TP_train / (nb_TP_train + nb_FN_train) * 100
nb_FPR_train <- nb_FP_train / (nb_FP_train + nb_TN_train) * 100
nb_FNR_train <- 100 - nb_TPR_train
nb_TNR_train <- 100 - nb_FPR_train
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
nb_AUC_ACCURACY_TPR_FPR_TNR_FNR_train <- c('NB_train', nb_AUC_train, round(nb_accuracy_train, 2), round
nb_AUC_ACCURACY_TPR_FPR_TNR_FNR_train
```

```
## [1] "NB_train"          "0.928571428571429" "0.92"
## [4] "85.71"             "0"                 "100"
## [7] "14.29"
```

### =============== Having computed for the train dataset, now we'll compute for the test data and do necessary computations.

### Naive Bayes on test dataset.

```
nb_prediction_test <- predict(nb_naiveBayes_model_train, mydata_test)
```

#### Computation of confusion matrix for test data, with table() function.

```
nb_confusion_matrix_test <- table(nb_prediction_test, mydata_test$label)

# Gathering parts of confusion matrix, for later use:
nb_TP_test = nb_confusion_matrix_test[1, 1]
nb_FP_test = nb_confusion_matrix_test[1, 2]
nb_FN_test = nb_confusion_matrix_test[2, 1]
nb_TN_test = nb_confusion_matrix_test[2, 2]

nb_confusion_matrix_test
```

```
##
## nb_prediction_test BLACK BLUE
##              BLACK     5    1
##              BLUE      3    1
```

#### Computation of Accuracy parameter from confusion matrix.

```
nb_accuracy_test <- sum(diag(nb_confusion_matrix_test)) / sum(nb_confusion_matrix_test)
nb_accuracy_test
```

```
## [1] 0.6
```

#### Computation of AUC for test data.

```
library(pROC)
nb_AUC_test <- roc(mydata_test$label, as.numeric(nb_prediction_test))
nb_AUC_test <- nb_AUC_test$auc
nb_AUC_test
```

```
## Area under the curve: 0.5625
```

#### Computation of TPR, FPR, TNR, FNR.

```
nb_TPR_test <- nb_TP_test / (nb_TP_test + nb_FN_test) * 100
nb_FPR_test <- nb_FP_test / (nb_FP_test + nb_TN_test) * 100
nb_FNR_test <- 100 - nb_TPR_test
nb_TNR_test <- 100 - nb_FPR_test
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
nb_AUC_ACCURACY_TPR_FPR_TNR_FNR_test <- c('NB_test', nb_AUC_test, round(nb_accuracy_test, 2), round(nb_
nb_AUC_ACCURACY_TPR_FPR_TNR_FNR_test
```

```
## [1] "NB_test" "0.5625"  "0.6"     "62.5"    "50"      "50"      "37.5"
```

# =============== Having computed for the NB algorithm, now we'll compute for the KNN.

# KNN on entire training dataset.

Since we ran LR and NB models on the enitre dataset, to determine whether the model can learn, we are skipping that step in KNN.

### Executing KNN().

Based on the requirement, first we'll do KNN on training data, with k = 3, and then k = 5.

```
knn_fit_train <- knn3(label~., data = mydata_train, k = 3)   # KNN with k = 3.
KNN_train <- predict(knn_fit_train, newdata = mydata_train, type = "class")
```

### Creating the confusion matrix.

```
knn_confusion_matrix_train <- table(KNN_train, mydata_train$label)

# Gathering parts of confusion matrix, for later use:
knn_TP_train = knn_confusion_matrix_train[1, 1]
knn_FP_train = knn_confusion_matrix_train[1, 2]
knn_FN_train = knn_confusion_matrix_train[2, 1]
knn_TN_train = knn_confusion_matrix_train[2, 2]

knn_confusion_matrix_train
```

```
##
## KNN_train BLACK BLUE
##     BLACK    12    2
##     BLUE      2   10
```

### Computing accuracy.

```
knn_accuracy_train <- sum(diag(knn_confusion_matrix_train)) / sum(knn_confusion_matrix_train)
knn_accuracy_train
```

```
## [1] 0.8461538
```

### Computation of AUC for train data.

```
knn_AUC_train <- roc(mydata_train$label, as.numeric(KNN_train))
knn_AUC_train <- knn_AUC_train$auc
knn_AUC_train
```

```
## Area under the curve: 0.8452
```

### Computation of TPR, FPR, TNR, FNR.

```
knn_TPR_train <- knn_TP_train / (knn_TP_train + knn_FN_train) * 100
knn_FPR_train <- knn_FP_train / (knn_FP_train + knn_TN_train) * 100
knn_FNR_train <- 100 - knn_TPR_train
knn_TNR_train <- 100 - knn_FPR_train
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_train3 <- c('KNN_train3', knn_AUC_train, round(knn_accuracy_train, 2),
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_train3
```

```
## [1] "KNN_train3"        "0.845238095238095" "0.85"
## [4] "85.71"             "16.67"             "83.33"
## [7] "14.29"
```

## =============== Having computed for the train dataset, now we'll compute for the test data and do necessary computations.

## KNN on entire testing dataset.

Since we ran LR and NB models on the enitre dataset, to determine whether the model can learn, we are skipping that step in KNN.

### Executing KNN().

Based on the requirement, we'll do KNN on testing data, with k = 3.

```
knn_fit_test <- predict(knn_fit_train, newdata = mydata_test, type = "class")
```

### Creating the confusion matrix.

```
knn_confusion_matrix_test <- table(knn_fit_test, mydata_test$label)

# Gathering parts of confusion matrix, for later use:
knn_TP_test = knn_confusion_matrix_test[1, 1]
knn_FP_test = knn_confusion_matrix_test[1, 2]
knn_FN_test = knn_confusion_matrix_test[2, 1]
knn_TN_test = knn_confusion_matrix_test[2, 2]

knn_confusion_matrix_test
```

```
##
## knn_fit_test BLACK BLUE
##        BLACK     5    0
##        BLUE      3    2
```

### Computing accuracy.

```
knn_accuracy_test <- sum(diag(knn_confusion_matrix_test)) / sum(knn_confusion_matrix_test)
knn_accuracy_test
```

```
## [1] 0.7
```

### Computation of AUC for test data.

```
knn_AUC_test <- roc(mydata_test$label, as.numeric(knn_fit_test))
knn_AUC_test <- knn_AUC_test$auc
knn_AUC_test
```

```
## Area under the curve: 0.8125
```

### Computation of TPR, FPR, TNR, FNR.

```
knn_TPR_test <- knn_TP_test / (knn_TP_test + knn_FN_test) * 100
knn_FPR_test <- knn_FP_test / (knn_FP_test + knn_TN_test) * 100
knn_FNR_test <- 100 - knn_TPR_test
knn_TNR_test <- 100 - knn_FPR_test
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_test3 <- c('KNN_test3', knn_AUC_test, round(knn_accuracy_test, 2), rou
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_test3
```

```
## [1] "KNN_test3" "0.8125"    "0.7"       "62.5"      "0"         "100"
## [7] "37.5"
```

## =============== Having computed for the KNN = 3, now we'll compute for the KNN = 5.

### Executing KNN().

Based on the requirement, first we'll do KNN on training data, with k = 5.

```
knn_fit_train5 <- knn3(label~., data = mydata_train, k = 5)    # KNN with k = 5.
KNN_train5 <- predict(knn_fit_train5, newdata = mydata_train, type = "class")
```

### Creating the confusion matrix.

```
knn_confusion_matrix_train5 <- table(KNN_train5, mydata_train$label)

# Gathering parts of confusion matrix, for later use:
knn_TP_train5 = knn_confusion_matrix_train5[1, 1]
knn_FP_train5 = knn_confusion_matrix_train5[1, 2]
knn_FN_train5 = knn_confusion_matrix_train5[2, 1]
knn_TN_train5 = knn_confusion_matrix_train5[2, 2]

knn_confusion_matrix_train5
```

```
##
## KNN_train5 BLACK BLUE
##      BLACK    12    1
##      BLUE      2   11
```

### Computing accuracy.

```
knn_accuracy_train5 <- sum(diag(knn_confusion_matrix_train5)) / sum(knn_confusion_matrix_train5)
knn_accuracy_train5
```

```
## [1] 0.8846154
```

### Computation of AUC for train data.

```
knn_AUC_train5 <- roc(mydata_train$label, as.numeric(KNN_train5))
knn_AUC_train5 <- knn_AUC_train5$auc
knn_AUC_train5
```

```
## Area under the curve: 0.8869
```

### Computation of TPR, FPR, TNR, FNR.

```
knn_TPR_train5 <- knn_TP_train5 / (knn_TP_train5 + knn_FN_train5) * 100
knn_FPR_train5 <- knn_FP_train5 / (knn_FP_train5 + knn_TN_train5) * 100
knn_FNR_train5 <- 100 - knn_TPR_train5
knn_TNR_train5 <- 100 - knn_FPR_train5
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_train5 <- c('KNN_train5', knn_AUC_train5, round(knn_accuracy_train5, 2)
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_train5
```

```
## [1] "KNN_train5"        "0.886904761904762" "0.88"
## [4] "85.71"             "8.33"              "91.67"
## [7] "14.29"
```

## =============== Having computed for the train dataset, now we'll compute for the test data and do necessary computations.

## KNN on entire testing dataset.

Since we ran LR and NB models on the enitre dataset, to determine whether the model can learn, we are skipping that step in KNN.

### Executing KNN().

Based on the requirement, we'll do KNN on testing data, with k = 5.

```
KNN_test5 <- predict(knn_fit_train5, newdata = mydata_test, type = "class")
```

```
caret::confusionMatrix(KNN_test5, mydata_test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BLACK BLUE
##      BLACK     5    0
##      BLUE      3    2
##
##                Accuracy : 0.7
##                  95% CI : (0.3475, 0.9333)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 0.8791
##
##                   Kappa : 0.4
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.6250
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.4000
##              Prevalence : 0.8000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8125
##
##        'Positive' Class : BLACK
##
```

### Creating the confusion matrix.

```r
knn_confusion_matrix_test5 <- table(KNN_test5, mydata_test$label)

# Gathering parts of confusion matrix, for later use:
knn_TP_test5 = knn_confusion_matrix_test5[1, 1]
knn_FP_test5 = knn_confusion_matrix_test5[1, 2]
knn_FN_test5 = knn_confusion_matrix_test5[2, 1]
knn_TN_test5 = knn_confusion_matrix_test5[2, 2]

knn_confusion_matrix_test5
```

```
##
## KNN_test5 BLACK BLUE
##     BLACK     5    0
##     BLUE      3    2
```

### Computing accuracy.

```r
knn_accuracy_test5 <- sum(diag(knn_confusion_matrix_test5)) / sum(knn_confusion_matrix_test5)
knn_accuracy_test5
```

```
## [1] 0.7
```

### Computation of AUC for test data.

```
knn_AUC_test5 <- roc(mydata_test$label, as.numeric(KNN_test5))
knn_AUC_test5 <- knn_AUC_test5$auc
knn_AUC_test5
```

```
## Area under the curve: 0.8125
```

### Computation of TPR, FPR, TNR, FNR.

```
knn_TPR_test5 <- knn_TP_test5 / (knn_TP_test5 + knn_FN_test5) * 100
knn_FPR_test5 <- knn_FP_test5 / (knn_FP_test5 + knn_TN_test5) * 100
knn_FNR_test5 <- 100 - knn_TPR_test5
knn_TNR_test5 <- 100 - knn_FPR_test5
```

### Creating a vector (Algo, AUC, ACCURACY, TPR, FPR, TNR, FNR), for future use.

```
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_test5 <- c('KNN_test5', knn_AUC_test5, round(knn_accuracy_test5, 2), r
knn_AUC_ACCURACY_TPR_FPR_TNR_FNR_test5
```

```
## [1] "KNN_test5" "0.8125"    "0.7"       "62.5"      "0"         "100"
## [7] "37.5"
```

==========================================

## Final answers to the Homework 1 requirements.

1) The two required tables are shown at the bottom. The first table is the learnability table (based on training data) and the second table is the generalizability table (based test data). The former shows the model's ability to learn and the latter shows its ability to generalize.

2) "1) Help your client understand that you have selected an appropriate model that has the capacity to learn".
   I built the models for entire dataset with Logistic Regression (LR), Naive Bayes (NB). In both cases the readings of Accuracy parameter were greater than 70%. So, the models are not underfitting. So, based on class lectures and notes, I conclude that the models are capable of learning.

3) "2) Help your client understand that when deployed your model is capable of generalizing".
   All the models LR, NB and KNN performed very well on the training dataset, keeping an Accuracy of 100% in the first, 92% in the sedcond and 92% and 88% in two KNNs. However, none of the three models LR, NB, KNN (with k == 3 and k == 5) were able to generalize.

   The Accuracy of LR deteriorated from 100% to 70%, NB deteriorated from 92% to 60%, KNN (k == 3) deteriorated from 85% to 70% and KNN (k == 5) deteriorated from 88% to 70%. The relative deterioration was least in KNN (k == 3), which was (85 - 70) / 85 * 100 = 17.64%.

So, overall, I would think that KNN (k == 3) was least bad. So, if I have to, then I would recommend this to my client.

However, a scanty data of 36 records and 2 independent column is not a realistic situation. So, I would request my client for a bigger dataset, including a good metadata.

Furthermore, I have seen KNN (k == 3) to perform erratically, yielding close, but different values (confusion matrix, accuracy, AUC etc) on different runs, the data and seed value for datasplit remaining constant.

```
table_1 <- data.frame(matrix(ncol = 6, nrow = 0))
table_1 <- rbind(table_1, lr_AUC_ACCURACY_TPR_FPR_TNR_FNR_train, nb_AUC_ACCURACY_TPR_FPR_TNR_FNR_train,
colnames(table_1) <- c("ALGO", "AUC","ACCURACY", "TPR", "FPR", "TNR", "FNR")

print("Learning table:")
```

```
## [1] "Learning table:"
```

```
table_1
```

```
##         ALGO                 AUC ACCURACY   TPR   FPR   TNR   FNR
## 1   LR_train                   1      100   100     0   100     0
## 2   NB_train 0.928571428571429     0.92 85.71     0   100 14.29
## 3 KNN_train3 0.845238095238095     0.85 85.71 16.67 83.33 14.29
## 4 KNN_train5 0.886904761904762     0.88 85.71  8.33 91.67 14.29
```

```
table_2 <- data.frame(matrix(ncol = 6, nrow = 0))
table_2 <- rbind(table_2, lr_AUC_ACCURACY_TPR_FPR_TNR_FNR_test, nb_AUC_ACCURACY_TPR_FPR_TNR_FNR_test, kn
colnames(table_2) <- c("ALGO", "AUC","ACCURACY", "TPR", "FPR", "TNR", "FNR")

print("Generalizing table:")
```

```
## [1] "Generalizing table:"
```

```
table_2
```

```
##        ALGO     AUC ACCURACY   TPR   FPR   TNR   FNR
## 1   LR_test 0.84375       70 85.71 66.67 33.33 14.29
## 2   NB_test  0.5625      0.6  62.5    50    50  37.5
## 3 KNN_test3  0.8125      0.7  62.5     0   100  37.5
## 4 KNN_test5  0.8125      0.7  62.5     0   100  37.5
```