DATA622_HW1

Mia Chen

10/15/2020

```
library(e1071)
library(caret)
```

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

```
## Warning: package 'lattice' was built under R version 3.6.2
```

```
## Loading required package: ggplot2
```

```
library(class)
```

Load input data dataset_hw1.csv

```
path <- "file:///Users/bobo/Documents/622/HW1/15001956_p1_dataset_hw1.csv"
data <-read.csv(path, header=TRUE, sep=',', stringsAsFactors=FALSE)
```

Review the data, quick and easy EDA

```
head(data)
```

```
##   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
```

```
tail(data)
```

```
##     X Y label
## 31 63 a BLACK
## 32 63 b  BLUE
## 33 63 c  BLUE
## 34 63 d  BLUE
## 35 63 e  BLUE
## 36 63 f  BLUE
```

Shape of this dataset

There are 36 observations and 3 variables in this dataset.

```
dim(data)
```

```
## [1] 36  3
```

Manipulating column names

Let us change the name of the first column

```
names(data)[[1]]<-'X'

names(data)
```

```
## [1] "X"     "Y"     "label"
```

Are the covariates correlated?

Since the variables are categorical, we can't review the correlation amongst independent variables

Distribution of Target Variable

Let's assume our customers want us to predict label given the rest. We want to make sure the dataset have a balanced number of each class. There are 22 BLACK and 14 BLUE in the dataset, so we have a good balance.

```
classLabels<-table(data$label) # this is the actual class distribution both test/train partitions must
print(classLabels)
```

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

```
names(classLabels)
```

```
## [1] "BLACK" "BLUE"
```

Binary vs Multi-class Classification

There are only two outcomes in the target variable, thus we have a binary classification.

```
length(names(classLabels))
```

```
## [1] 2
```

```
ifelse(length(names(classLabels))==2,"binary classification", "multi-class classification")
```

```
## [1] "binary classification"
```

First Classification run

In order to prepare for a logistic regression, we are going to replace BLACK with 0, and BLUE with 1.

```
data$label[data$label == "BLACK"] <- 0
data$label[data$label == "BLUE"] <- 1
data$label <- as.integer(data$label)
```

```
options(scipen=999)

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

```
##
## Call:  glm(formula = label ~ ., family = "binomial", data = data)
##
## Coefficients:
##            (Intercept)                       X                      Yb
## -0.3664897106446274866  -0.0087345905773133573  -0.0000000000000005899
##                      Yc                      Yd                      Ye
##  2.3185312063557401707  -0.0000000000000005103  -0.9218473391890408264
##                      Yf
## -0.0000000000000002729
##
## Degrees of Freedom: 35 Total (i.e. Null);  29 Residual
## Null Deviance:       48.11
## Residual Deviance: 41.14     AIC: 55.14
```

```
summary(glm_model)
```

```
##
## Call:
## glm(formula = label ~ ., family = "binomial", data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8321  -0.8572  -0.6662   0.8365   1.9931
##
## Coefficients:
##                            Estimate             Std. Error z value Pr(>|z|)
## (Intercept) -0.3664897106446274866  1.1006141266524169797  -0.333   0.7391
## X           -0.0087345905773133573  0.0183410392923599591  -0.476   0.6339
## Yb          -0.0000000000000005899  1.2292282378523728958   0.000   1.0000
## Yc           2.3185312063557401707  1.4017970122604217487   1.654   0.0981 .
## Yd          -0.0000000000000005103  1.2292282378523733399   0.000   1.0000
## Ye          -0.9218473391890408264  1.4005965197126557520  -0.658   0.5104
```

```
## Yf          -0.0000000000000002729  1.2292282378523735620   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: 41.139  on 29  degrees of freedom
## AIC: 55.139
##
## Number of Fisher Scoring iterations: 4
```

We will reject any variable with a p-value greater than 0.05. In this case, looks like none of the coefficients are significant here.

```
glm_probs <- predict(glm_model, type="response")
glm_probs[1:5]
```

```
##         1         2         3         4         5
## 0.3988731 0.3988731 0.8708358 0.3988731 0.2088271
```

```
glm_pred <- ifelse(glm_probs < 0.5, 0, 1)

table(glm_pred, data$label)
```

```
##
## glm_pred  0  1
##        0 21  9
##        1  1  5
```

Now let us compute the performance of the classifier. If our model is any good it must perform better than 0.5.

```
mean(glm_pred == data$label)
```

```
## [1] 0.7222222
```

From the first run, we saw a 72% accuracy. But this is from data the model has already seen.

Data Partition Repeatability/Reproducibility

Let us split our data into "random" test/train disjoint partitions. s

```
set.seed(33)
tstidx<-sample(1:nrow(data),0.30*nrow(data),replace=F)
trdata<-data[-tstidx,]
tsdata<-data[tstidx,]
```

Now let us run the model.

```r
glm.trmodel<-glm(label ~ X+Y, data=trdata,family='binomial')
summary(glm.trmodel)
```

```
##
## Call:
## glm(formula = label ~ X + Y, family = "binomial", data = trdata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7538  -0.7921  -0.6637   0.8624   1.8469
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.56338    1.46635  -1.066    0.286
## X            0.01165    0.02170   0.537    0.591
## Yb          -0.32698    1.61672  -0.202    0.840
## Yc           2.21845    1.64613   1.348    0.178
## Yd           1.16719    1.85224   0.630    0.529
## Ye          -0.35263    1.61757  -0.218    0.827
## Yf           0.41758    1.45212   0.288    0.774
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 33.542  on 25  degrees of freedom
## Residual deviance: 29.121  on 19  degrees of freedom
## AIC: 43.121
##
## Number of Fisher Scoring iterations: 4
```

```r
predtr<-predict(glm.trmodel,trdata[,1:2],type='response')
head(predtr)
```

```
##         1         2         3         4         6        11
## 0.1816638 0.1379887 0.6711381 0.4163146 0.2520828 0.1551700
```

```r
predtrclass<-ifelse(predtr<0.5, 0, 1)
table(trdata[[3]])
```

```
##
##  0  1
## 17  9
```

```r
table(predtrclass)
```

```
## predtrclass
##  0  1
## 21  5
```

```r
length(predtrclass)==length(trdata[[3]])
```

```
## [1] TRUE
```

```
(trcfm<-confusionMatrix(table(trdata[[3]],predtrclass)))
```

```
## Confusion Matrix and Statistics
##
##    predtrclass
##      0  1
##   0 16  1
##   1  5  4
##
##                Accuracy : 0.7692
##                  95% CI : (0.5635, 0.9103)
##     No Information Rate : 0.8077
##     P-Value [Acc > NIR] : 0.7792
##
##                   Kappa : 0.4307
##
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.7619
##             Specificity : 0.8000
##          Pos Pred Value : 0.9412
##          Neg Pred Value : 0.4444
##              Prevalence : 0.8077
##          Detection Rate : 0.6154
##    Detection Prevalence : 0.6538
##       Balanced Accuracy : 0.7810
##
##        'Positive' Class : 0
##
```

Accuracy on training data is 0.7692 indicating model is capable of learning. Now let us predict the class for never seen before data. That is our 'held out' test dataset. This is what matters to the business.

```
predts<-predict(glm.trmodel,tsdata[,1:2],type='response')
predtsclass<-ifelse(predts<0.5, 0, 1)
table(predtsclass)
```

```
## predtsclass
## 0 1
## 5 5
```

```
table(tsdata[[3]])
```

```
##
## 0 1
## 5 5
```

```
tscfm<-confusionMatrix(table(tsdata[[3]],predtsclass))
tscfm
```

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

```
##
##    predtsclass
##     0 1
##   0 2 3
##   1 3 2
##
##                Accuracy : 0.4
##                  95% CI : (0.1216, 0.7376)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.8281
##
##                   Kappa : -0.2
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.4
##             Specificity : 0.4
##          Pos Pred Value : 0.4
##          Neg Pred Value : 0.4
##              Prevalence : 0.5
##          Detection Rate : 0.2
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 0.4
##
##        'Positive' Class : 0
##
```

Accuracy on the test set (held out data or never seen before data) is 0.4, and has fallen down from accuracy obtained during training phase 0.76. The drop is about 47% drop in performance and therefore we conclude Model is over-fitting.

Now let us visualize our performance using ROC plots. We need ROCR or pROC package and there are other packages. I use pROC.

```
## [1] 5.1 4.1 4.1 2.1
```

```
## Loading required package: pROC
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls > cases
```

Now let us compute AUC and plot Receiver Operating Curve (ROC) using ROCR package.

```
getMetrics<-function(actual_class,predicted_response)
{
X=list()
if ( require(ROCR) ) {
auc_1=prediction(predicted_response,actual_class)
prf=performance(auc_1, measure="tpr",x.measure="fpr")
slot_fp=slot(auc_1,"fp")
slot_tp=slot(auc_1,"tp")

fpr=unlist(slot_fp)/unlist(slot(auc_1,"n.neg"))
tpr=unlist(slot_tp)/unlist(slot(auc_1,"n.pos"))

auc<-performance(auc_1,"auc")
AUC<-auc@y.values[[1]]
X=list(fpr=fpr,tpr=tpr,auc=AUC)
}
X
}
```

time to test our utility function...

```
L<-getMetrics(tsdata[[3]],predts)
```

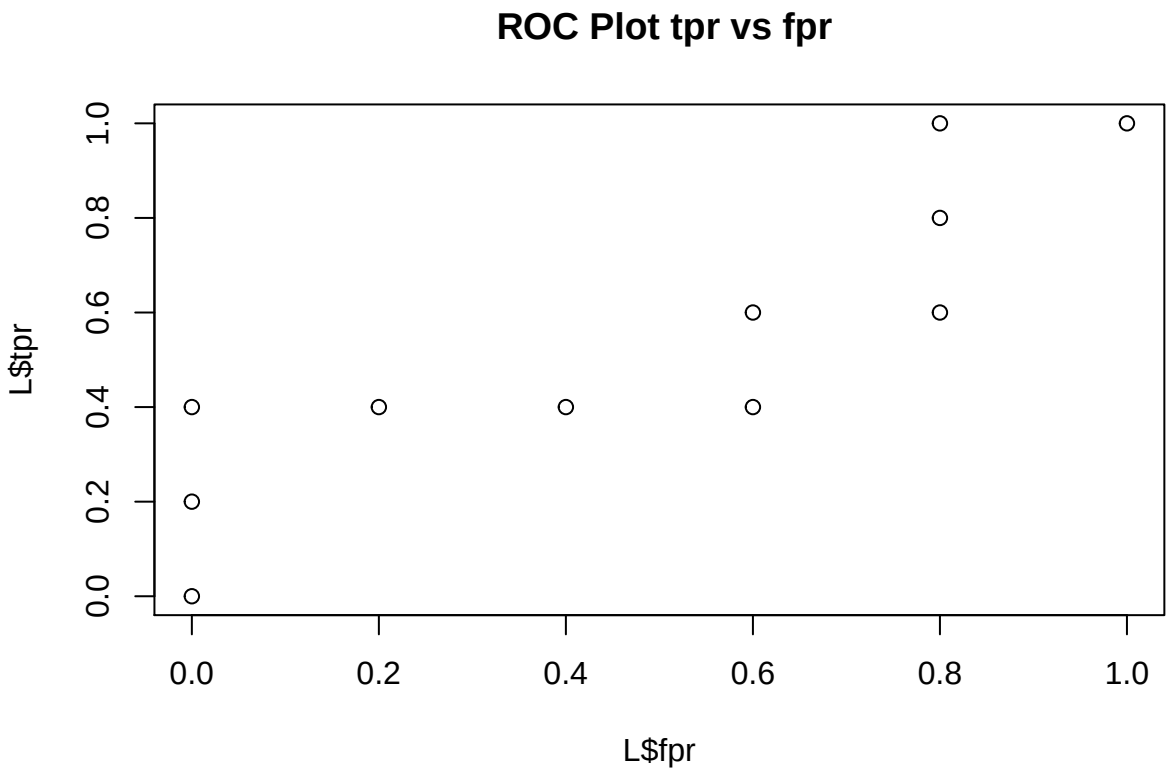
```
## Loading required package: ROCR

## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```
plot(L$fpr,L$tpr,main=" ROC Plot tpr vs fpr")
```




```
print(paste("AUC=",L$auc,sep=''))
```

```
## [1] "AUC=0.56"
```

We would prefer 90 or above. . . yet we only have AUC=0.56 and accuracy=0.4, indicating this is not a good model.

Naive Bayes

Looking at the training and test set distribution, they are varied significantly. Therefore, the model cannot generalize.

```
table(trdata$label)
```

```
##
##  0  1
## 17  9
```

```
table(tsdata$label)
```

```
##
## 0 1
## 5 5
```

Train NB model using the training set

```
nbtr.model<-naiveBayes(label~.,data=trdata)
```

Performance over the training set

```
nbtr.trpred<-predict(nbtr.model,trdata[,-c(3)],type='raw')
nbtr.trclass<-unlist(apply(round(nbtr.trpred),1,which.max))-1
nbtr.trtbl<-table(trdata[[3]], nbtr.trclass)
tr.cfm<-confusionMatrix(nbtr.trtbl)
tr.cfm
```

```
## Confusion Matrix and Statistics
##
##    nbtr.trclass
##      0  1
##   0 15  2
##   1  5  4
##
##                Accuracy : 0.7308
##                  95% CI : (0.5221, 0.8843)
##     No Information Rate : 0.7692
##     P-Value [Acc > NIR] : 0.7641
##
##                   Kappa : 0.3546
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.7500
##             Specificity : 0.6667
##          Pos Pred Value : 0.8824
##          Neg Pred Value : 0.4444
##              Prevalence : 0.7692
##          Detection Rate : 0.5769
##    Detection Prevalence : 0.6538
##       Balanced Accuracy : 0.7083
##
##        'Positive' Class : 0
##
```

Performance over held out data, the test set

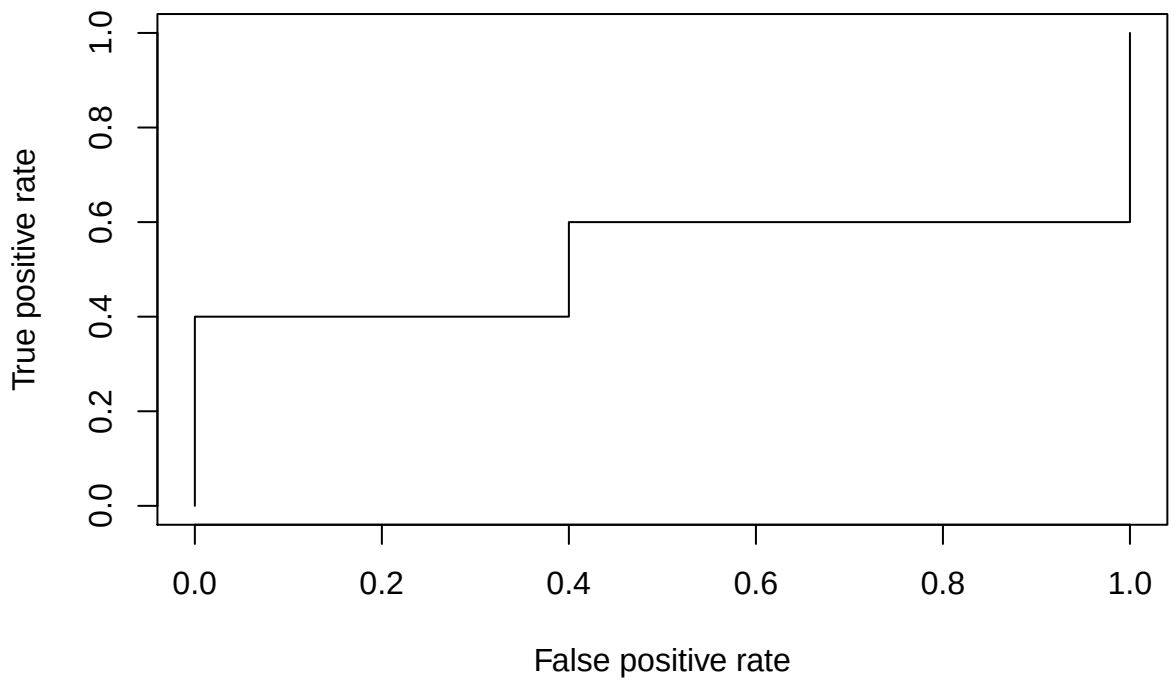
```
nbtr.tspred<-predict(nbtr.model,tsdata[,-c(3)],type='raw')
roc.nbtr.tspred<-nbtr.tspred[,2]
nbtr.tsclass<-unlist(apply(round(nbtr.tspred),1,which.max))-1
nbtr.tstbl<-table(tsdata[[3]], nbtr.tsclass)
tst.cfm<-confusionMatrix(nbtr.tstbl)
tst.cfm
```

```
## Confusion Matrix and Statistics
##
##    nbtr.tsclass
##     0 1
##   0 5 0
```

```
##   1 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 : 0
##
```

Let us plot the ROCR curve and determine AUC for the e1071 standard implementation.

```
nbtr.pred <- prediction(nbtr.tspred[,2], tsdata[[3]])
perf_nb <- performance(nbtr.pred, measure='tpr', x.measure='fpr')
plot(perf_nb)
```




```
auc <- performance(nbtr.pred, 'auc')
AUC <- auc@y.values[[1]]
AUC
```

```
## [1] 0.52
```

We obtained AUC=0.52 and accuracy=0.7 from the Naive Bayes model.

kNN

```
library(dplyr)
```

```
##
## 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
```

```
library(gmodels)
```

```
##
## Attaching package: 'gmodels'

## The following object is masked from 'package:pROC':
##
##     ci
```

```
library(psych)
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
# Read data
path <- "file:///Users/bobo/Documents/622/HW1/15001956_p1_dataset_hw1.csv"
data2 <-read.csv(path, header=TRUE, sep=',', stringsAsFactors=FALSE)

# Make a copy of dataset
data_class <- data2

# Change target to factor in preparation for knn classification
data_class$label <- as.factor(data_class$label)
```

```r
# Isolate target varialb from the dataset
label_outcome <- data_class %>% select(label)
data_class <- data_class %>% select(-label)

str(data_class)
```

```
## 'data.frame':    36 obs. of  2 variables:
##  $ X: int  5 5 5 5 5 5 19 19 19 19 ...
##  $ Y: chr  "a" "b" "c" "d" ...
```

We see that Y is a categorical variable that have more than two levels, so we need a dummy code.

```r
Y <- as.data.frame(dummy.code(data_class$Y))
head(Y)
```

```
##   a b c d e f
## 1 1 0 0 0 0 0
## 2 0 1 0 0 0 0
## 3 0 0 1 0 0 0
## 4 0 0 0 1 0 0
## 5 0 0 0 0 1 0
## 6 0 0 0 0 0 1
```

Combine new dummy variables with original dataset

```r
data_class <- cbind(data_class, Y)
```

Remove original variable Y that had been dummy coded

```r
data_class <- data_class %>% select(-Y)

head(data_class)
```

```
##   X a b c d e f
## 1 5 1 0 0 0 0 0
## 2 5 0 1 0 0 0 0
## 3 5 0 0 1 0 0 0
## 4 5 0 0 0 1 0 0
## 5 5 0 0 0 0 1 0
## 6 5 0 0 0 0 0 1
```

Split and partition data into train and test sets

```r
set.seed(44)
sample_size <- floor(0.70*nrow(data_class))
train_ind <- sample(seq_len(nrow(data_class)), size = sample_size)
pred_train <- data_class[train_ind,]
pred_test <- data_class[-train_ind,]
```

Split outcome variable into training and test sets using the same partition as above

```
outcome_train <- label_outcome[train_ind, ]
outcome_test <- label_outcome[-train_ind, ]
```

Use caret package. Run k-NN classification.

```
knn_model <- train(pred_train, outcome_train, method = "knn", preProcess = c("center", "scale"))
```

```
## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: d

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: d

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: d

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: e

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: e

## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: e
```

```
knn_model
```

```
## k-Nearest Neighbors
##
## 25 samples
##  7 predictor
##  2 classes: 'BLACK', 'BLUE'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 25, 25, 25, 25, 25, 25, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.4627229   0.01519995
##   7  0.4035815  -0.04952520
##   9  0.3686176  -0.12254381
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

When k=5, we obtain the optimal model with best accuracy.

Next we predict values using the knn model and compare to actual values with a confusion matrix.

```r
knn_pred <- predict(knn_model, newdata = pred_test)

confusionMatrix(knn_pred, outcome_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BLACK BLUE
##      BLACK     5    0
##      BLUE      4    2
##
##                Accuracy : 0.6364
##                  95% CI : (0.3079, 0.8907)
##     No Information Rate : 0.8182
##     P-Value [Acc > NIR] : 0.9653
##
##                   Kappa : 0.3125
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.5556
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.3333
##              Prevalence : 0.8182
##          Detection Rate : 0.4545
##    Detection Prevalence : 0.4545
##       Balanced Accuracy : 0.7778
##
##        'Positive' Class : BLACK
##
```

Computing ROC and AUC is somewhat non-trivial as kNN do not compute probabilities and results in unreliable ROC plots.

```r
myControl <- trainControl(
  method = "cv", # cross validation
  number = 10, # 10-fold cross validation
  summaryFunction = twoClassSummary,
  classProbs = TRUE,
  verboseIter = FALSE
)

knn_control <- train(label ~.,
                     data2,
                     method = "knn",
                     trControl = myControl
                     )
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
knn_control
```

```
## k-Nearest Neighbors
##
## 36 samples
##  2 predictor
##  2 classes: 'BLACK', 'BLUE'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 31, 33, 33, 33, 33, 33, ...
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   5  0.8625000  0.9166667  0.75
##   7  0.8583333  0.7833333  0.75
##   9  0.8083333  0.7833333  0.55
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

Although we obtain highest ROC with k=7, we would still use k=5 with a slightly lower ROC score but a higher accuracy as above.

```
Algorithm <- c("LR", "NB", "kNN")
AUC <- c(0.56, 0.52, 0.75)
Accuracy <- c(0.4, 0.7, 0.64)
TPR <- c(0.4, 0.625, 0.67)
FPR <- c(1-0.4, 1-0.625, 1-0.67)
TNR <- c(0.4, 1, 0.5)
FNR <- c(1-0.4, 1-1, 1-0.5)
df <- data.frame(Algorithm, AUC, Accuracy, TPR, FPR, TNR, FNR)
df
```

```
##   Algorithm  AUC Accuracy   TPR   FPR TNR FNR
## 1        LR 0.56     0.40 0.400 0.600 0.4 0.6
## 2        NB 0.52     0.70 0.625 0.375 1.0 0.0
## 3       kNN 0.75     0.64 0.670 0.330 0.5 0.5
```

From the table, we will say that NB is performing better than LR and kNN since it has the highest accuracy and TNR (specificity). However, we won't be able to generalize NB. In which case, kNN might be a better choice.