assignment_1

Salma Elshahawy

10/5/2020

Contents

```r
set.seed(41)
library(skimr)
```

Reading the data

First we need to construct the data and read it into a dataframe format.

```r
df <- data.frame(
  X = as.factor(c(5, 5, 5, 5, 5, 5, 19, 19, 19, 19, 19, 19, 35, 35, 35, 35, 35, 35, 51, 51, 51, 51, 51,
  Y = c("a","b","c","d","e","f","a","b","c","d","e","f","a","b","c","d","e","f","a","b","c","d","e","f"
  label = c("BLUE","BLACK","BLUE","BLACK","BLACK","BLACK","BLUE","BLUE","BLUE","BLUE","BLACK","BLUE","B
)
df
```

```
##     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
## 7  19 a  BLUE
## 8  19 b  BLUE
## 9  19 c  BLUE
## 10 19 d  BLUE
## 11 19 e BLACK
## 12 19 f  BLUE
## 13 35 a BLACK
## 14 35 b BLACK
## 15 35 c  BLUE
## 16 35 d BLACK
## 17 35 e BLACK
## 18 35 f BLACK
## 19 51 a BLACK
## 20 51 b BLACK
## 21 51 c  BLUE
## 22 51 d BLACK
## 23 51 e BLACK
## 24 51 f BLACK
## 25 55 a BLACK
## 26 55 b BLACK
## 27 55 c BLACK
## 28 55 d BLACK
## 29 55 e BLACK
## 30 55 f BLACK
## 31 63 a BLACK
## 32 63 b  BLUE
## 33 63 c  BLUE
## 34 63 d  BLUE
## 35 63 e  BLUE
## 36 63 f  BLUE
```

The data has two two covariates **X** and **Y**, and one target variables **label**. Now let's explore this data using some summary statistics.

Data Exploration

```
str(df)
```

```
## 'data.frame':    36 obs. of  3 variables:
##  $ X    : Factor w/ 6 levels "5","19","35",..: 1 1 1 1 1 1 2 2 2 2 ...
##  $ Y    : Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6 1 2 3 4 ...
##  $ label: Factor w/ 2 levels "BLACK","BLUE": 2 1 2 1 1 1 2 2 2 2 ...
```

Both **X** and **Y** has 6 levels of variability; however, label has only 2 levels, BLACK and BLUE values

```
skim(df)
```

Table 1: Data summary

Name	df
Number of rows	36

Table 1: Data summary

Number of columns	3
Column type frequency:	
factor	3
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
X	0	1	FALSE	6	5: 6, 19: 6, 35: 6, 51: 6
Y	0	1	FALSE	6	a: 6, b: 6, c: 6, d: 6
label	0	1	FALSE	2	BLA: 22, BLU: 14

The data has 36 observations and 3 variables.

```
summary(df)
```

```
##    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
```

Preparing the data for ML model

First, we need to split the data into train and test. Then I created an empty numerical variable for each of the required output metric.

```
library(caret)
library(ModelMetrics)

respCol <- ncol(df)[[1]]
train <- createDataPartition(df[,respCol], p = .70) # training data
obs <- df[-train$Resample1, respCol] # test data


perfALG <- c("LR","NB", "Knn3", "knn5")
perfAUC = numeric()
perfACC = numeric()
perfTPR = numeric()
perfFPR = numeric()
perfTNR = numeric()
perfFNR = numeric()
```

Simple Linear regression Model

```
## Logistic Regression (LR) Model
lrFit <- glm(label~., data = df[train$Resample1, ], family = binomial())
summary(lrFit)
```

```
##
## Call:
## glm(formula = label ~ ., family = binomial(), data = df[train$Resample1,
##     ])
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.17741  -0.00003   0.00000   0.00003   1.17741
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.274e+01  1.881e+04  -0.002    0.998
## X19          4.274e+01  1.881e+04   0.002    0.998
## X35          9.642e-01  2.236e+04   0.000    1.000
## X51          8.598e-02  2.729e+04   0.000    1.000
## X55         -4.146e+01  2.615e+04  -0.002    0.999
## X63          4.274e+01  1.881e+04   0.002    0.998
## Yb           2.186e+01  1.901e+04   0.001    0.999
## Yc           6.340e+01  2.315e+04   0.003    0.998
## Yd           2.098e+01  1.369e+04   0.002    0.999
## Ye           1.583e-15  2.000e+00   0.000    1.000
## Yf           2.109e+01  1.820e+04   0.001    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.6465  on 25  degrees of freedom
## Residual deviance:  5.5452  on 15  degrees of freedom
## AIC: 27.545
##
## Number of Fisher Scoring iterations: 21
```

Fit the data into LR

```
lrProb <- predict(lrFit, newdata = df[-train$Resample1,], type = "response")

lrPred <- rep("BLACK", length(lrProb))
lrPred[lrProb > 0.5] = "BLUE"
lrPred = as.factor(lrPred)
#postResample(lrPred, obs)
perfAUC <- c(perfAUC, auc(actual = obs, predicted = lrPred))
perfACC <- c(perfACC, postResample(lrPred, obs)["Accuracy"])
perfTPR <- c(perfTPR, caret::sensitivity(lrPred, obs))
perfFPR <- c(perfFPR, 1 - caret::specificity(lrPred, obs))
perfTNR <- c(perfTNR, caret::specificity(lrPred, obs))
```

```r
perfFNR <- c(perfFNR, 1 - caret::sensitivity(lrPred, obs))
table(lrPred, obs)
```

```
##        obs
## lrPred  BLACK BLUE
##   BLACK     6    1
##   BLUE      0    3
```

As we can see, we got a confusion matrix with 6 True BLACKS, 3 True BLUES. However, we got 1 False BLUE predicted as BLACK. Over all the performance is not bad and considered a good classifier.

Naive Bayes Model

```r
## Naive Bayes (NB) Model
library(e1071)
nbFit <- naiveBayes(label ~ ., data = df[train$Resample1, ])
print(nbFit)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     BLACK      BLUE
## 0.6153846 0.3846154
##
## Conditional probabilities:
##        X
## Y            5     19     35     51     55     63
##   BLACK 0.2500 0.0625 0.1875 0.2500 0.1875 0.0625
##   BLUE  0.1000 0.4000 0.1000 0.0000 0.0000 0.4000
##
##        Y
## Y            a      b      c      d      e      f
##   BLACK 0.1875 0.1250 0.0625 0.1875 0.3125 0.1250
##   BLUE  0.1000 0.2000 0.3000 0.2000 0.1000 0.1000
```

```r
nbPred <- predict(nbFit, newdata = df[-train$Resample1,], type = "class")
#postResample(nbPred, obs)
perfAUC <- c(perfAUC, auc(actual = obs, predicted = nbPred))
perfACC <- c(perfACC, postResample(nbPred, obs)["Accuracy"])
perfTPR <- c(perfTPR, caret::sensitivity(nbPred, obs))
perfFPR <- c(perfFPR, 1 - caret::specificity(nbPred, obs))
perfTNR <- c(perfTNR, caret::specificity(nbPred, obs))
perfFNR <- c(perfFNR, 1 - caret::sensitivity(nbPred, obs))
table(nbPred, obs)
```

```
##        obs
## nbPred  BLACK BLUE
##   BLACK     6    2
##   BLUE      0    2
```

It surprisingly introduces a higher prediction errors; although NB performs well with text classification and with a small data. It gave 6 Trues on BLACKS, 2 Trues on BLUE. However, it gave 2 Falses on BLUES predicted as BLACK.

KNN ML Model(k=3)

```r
## KNN Model
knnFit3 <- knn3(label ~., data = df[train$Resample1,], k = 3)
knnPred3 <- predict(knnFit3, newdata = df[-train$Resample1,], type = "class")
#postResample(knnPred, obs)
perfAUC <- c(perfAUC, auc(actual = obs, predicted = knnPred3))
perfACC <- c(perfACC, postResample(knnPred3, obs)["Accuracy"])
perfTPR <- c(perfTPR, caret::sensitivity(knnPred3, obs))
perfFPR <- c(perfFPR, 1 - caret::specificity(knnPred3, obs))
perfTNR <- c(perfTNR, caret::specificity(knnPred3, obs))
perfFNR <- c(perfFNR, 1 - caret::sensitivity(knnPred3, obs))
table(knnPred3, obs)
```

```
##         obs
## knnPred3 BLACK BLUE
##    BLACK     6    2
##    BLUE      0    2
```

KNN with K = 3 gave the same performance as Naive Bayes.

KNN ML Model (k=5)

```r
## KNN Model
knnFit5 <- knn3(label ~., data = df[train$Resample1,], k = 5)
knnPred5 <- predict(knnFit5, newdata = df[-train$Resample1,], type = "class")
#postResample(knnPred, obs)
perfAUC <- c(perfAUC, auc(actual = obs, predicted = knnPred5))
perfACC <- c(perfACC, postResample(knnPred5, obs)["Accuracy"])
perfTPR <- c(perfTPR, caret::sensitivity(knnPred5, obs))
perfFPR <- c(perfFPR, 1 - caret::specificity(knnPred5, obs))
perfTNR <- c(perfTNR, caret::specificity(knnPred5, obs))
perfFNR <- c(perfFNR, 1 - caret::sensitivity(knnPred5, obs))
table(knnPred5, obs)
```

```
##         obs
## knnPred5 BLACK BLUE
##    BLACK     6    2
##    BLUE      0    2
```

Again, changing the k parameter didn't chage much in the overall performance to the classifier, where it gave the same k = 3

Generate an accuracy comparing table

```
perf <- data.frame(
  ALGO = perfALG,
  AUC = perfAUC,
  ACCURACY = perfACC,
  TPR = perfTPR,
  FPR = perfFPR,
  TNR = perfTNR,
  FNR = perfFNR
)
```

```
perf
```

```
##   ALGO   AUC ACCURACY TPR  FPR  TNR FNR
## 1   LR 0.875      0.9   1 0.25 0.75   0
## 2   NB 0.750      0.8   1 0.50 0.50   0
## 3 Knn3 0.750      0.8   1 0.50 0.50   0
## 4 knn5 0.750      0.8   1 0.50 0.50   0
```

Summary

It is hard to determine a clear winner among the classifiers, as multiple runs will select different training and testing data and greatly influence the training and performance of each model from run to run. However, since the dataset is pretty small and small datasets require strong assumption (bias). As it is preferable to use Occam's razor first. The less the assumptions are and the hypothesis is, the better is the results. In our case, Linear regression seems to perform well based on the AUC and ACCURACY.