Naive Bayes

Melissa Paniagua

10/8/2020

Assigment 3 | Module 5

The purpose of this assignment is to use Naive Bayes for classification.

The Flight Delays data set has data about flight delays and on-time from the Washington, DC area into the New York City area during January 2004. The variables in the dataset are the following:

- CRS_DEP_TIME : It indicates the actual departure time. The range of time is from 6:00 AM and 10:00 PM.
- CARRIER : Airline code. This dataset has eight airlines, which are CO(Continental), DH (Atlantic Coast), DL(Delta), MQ(American Eagle), OH(Comair), RU(Continental Express), UA(United), and US(USAirways)
- DEP_TIME : It indicates the intended departure time.
- DEST : Destination airport.
- DISTANCE : Distance in miles.
- FL_DATE : Flight date.
- FL_NUM : Flight number,
- ORIGIN : Originating the airport using the airport code. Our dataset has three airport codes: DCA (Reagan National), IAD (Dulles), and BWI (Baltimore–Washington International).
- Weather : Weather of that day, and it shows if 1 if there was a weather-related delay, and 0 otherwise.
- DAY_WEEK : Day of the week, which 1 = Monday, 2 = Tuesday,. . . , 7 = Sunday.
- DAY_OF_MONTH : Day of the month.
- TAIL_NUM : Plane's tail number.
- FLIGHT STATUS : It indicates if the flight was delayed or on-time.

The data were obtained from the Bureau of Transportation Statistics (available on the web at www.transtats.bts.gov).

```
#Load the libraries needed
library(readr)
library(caret)
library(ggplot2)
library(magrittr)
```

```r
library(dplyr)
library(e1071)
library("gmodels")
library(pROC)
library(naivebayes)

# Load the file
FlightDelays <- read_csv("FlightDelays.csv")

#Show the first 6 rows
head(FlightDelays)
```

```
# A tibble: 6 x 13
  CRS_DEP_TIME CARRIER DEP_TIME DEST  DISTANCE FL_DATE FL_NUM ORIGIN Weather
         <dbl> <chr>      <dbl> <chr>    <dbl> <chr>    <dbl> <chr>    <dbl>
1         1455 OH          1455 JFK        184 01/01/~   5935 BWI          0
2         1640 DH          1640 JFK        213 01/01/~   6155 DCA          0
3         1245 DH          1245 LGA        229 01/01/~   7208 IAD          0
4         1715 DH          1709 LGA        229 01/01/~   7215 IAD          0
5         1039 DH          1035 LGA        229 01/01/~   7792 IAD          0
6          840 DH           839 JFK        228 01/01/~   7800 IAD          0
# ... with 4 more variables: DAY_WEEK <dbl>, DAY_OF_MONTH <dbl>,
#   TAIL_NUM <chr>, `Flight Status` <chr>
```

Data Exploration and Data Preparation

```r
# Get structure of the dataframe
str(FlightDelays)
```

```
Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame':    2201 obs. of  13 variables:
 $ CRS_DEP_TIME : num  1455 1640 1245 1715 1039 ...
 $ CARRIER      : chr  "OH" "DH" "DH" "DH" ...
 $ DEP_TIME     : num  1455 1640 1245 1709 1035 ...
 $ DEST         : chr  "JFK" "JFK" "LGA" "LGA" ...
 $ DISTANCE     : num  184 213 229 229 229 228 228 228 228 228 ...
 $ FL_DATE      : chr  "01/01/2004" "01/01/2004" "01/01/2004" "01/01/2004" ...
 $ FL_NUM       : num  5935 6155 7208 7215 7792 ...
 $ ORIGIN       : chr  "BWI" "DCA" "IAD" "IAD" ...
 $ Weather      : num  0 0 0 0 0 0 0 0 0 0 ...
 $ DAY_WEEK     : num  4 4 4 4 4 4 4 4 4 4 ...
 $ DAY_OF_MONTH : num  1 1 1 1 1 1 1 1 1 1 ...
 $ TAIL_NUM     : chr  "N940CA" "N405FJ" "N695BR" "N662BR" ...
 $ Flight Status: chr  "ontime" "ontime" "ontime" "ontime" ...
 - attr(*, "spec")=
  .. cols(
  ..   CRS_DEP_TIME = col_double(),
  ..   CARRIER = col_character(),
  ..   DEP_TIME = col_double(),
  ..   DEST = col_character(),
  ..   DISTANCE = col_double(),
  ..   FL_DATE = col_character(),
```

```
  ..   FL_NUM = col_double(),
  ..   ORIGIN = col_character(),
  ..   Weather = col_double(),
  ..   DAY_WEEK = col_double(),
  ..   DAY_OF_MONTH = col_double(),
  ..   TAIL_NUM = col_character(),
  ..   `Flight Status` = col_character()
  .. )
```

```
#Getting the 5 variables needed, plus the outcome column (flight status)
mydf <- FlightDelays[, c(10, 1, 8, 4, 2, 13)]

#Show the first 6 rows
head(mydf)
```

```
# A tibble: 6 x 6
  DAY_WEEK CRS_DEP_TIME ORIGIN DEST  CARRIER `Flight Status`
     <dbl>        <dbl> <chr>  <chr> <chr>   <chr>
1        4         1455 BWI    JFK   OH      ontime
2        4         1640 DCA    JFK   DH      ontime
3        4         1245 IAD    LGA   DH      ontime
4        4         1715 IAD    LGA   DH      ontime
5        4         1039 IAD    LGA   DH      ontime
6        4          840 IAD    JFK   DH      ontime
```

Create slots for the schedule departure time column in groups of 30 minutes. It will generate 32 groups from 6:00 am to 10:00 pm.

```
#Create slots for the schedule departure time column in groups of 30 minutes.

# To copy columns needed
y <- mydf$CRS_DEP_TIME

# To set parameters
lowerbound <- 600
upperbound <- 2200
groupname <- matrix(NA, nrow = length(y), ncol = 1)
i = 1

# Create a while loop function to assigned the time frames in groups
while (any(is.na(groupname)))
  {
  position <- y >= lowerbound & y < (lowerbound+30)
  lowerbound <- lowerbound+30
  groupname[position]=i
  i= i+1
  if (i %% 2 != 0 & i>1) {
    lowerbound <- lowerbound + 40
  }
}

# To see the new groups
sort(unique(groupname))
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32
```

```r
#Add groupname column to the dataframe
mydf$groupname <- groupname

#To see the dataframe's datatype
str(mydf)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   2201 obs. of  7 variables:
 $ DAY_WEEK     : num  4 4 4 4 4 4 4 4 4 4 ...
 $ CRS_DEP_TIME : num  1455 1640 1245 1715 1039 ...
 $ ORIGIN       : chr  "BWI" "DCA" "IAD" "IAD" ...
 $ DEST         : chr  "JFK" "JFK" "LGA" "LGA" ...
 $ CARRIER      : chr  "OH" "DH" "DH" "DH" ...
 $ Flight Status: chr  "ontime" "ontime" "ontime" "ontime" ...
 $ groupname    : num [1:2201, 1] 18 22 14 23 10 6 14 22 23 31 ...
```

Now, we have to transfrom the Week, Time, and Group Name variables into a categorical variable. We are going to use the function as.factors to accomplish it.

```r
# Select the numeric variables
mycol_num <- mydf[, c(1, 2, 7)]

# Transform those variables into factors
mycol_factor <- as.data.frame(lapply(mycol_num, as.factor))
```

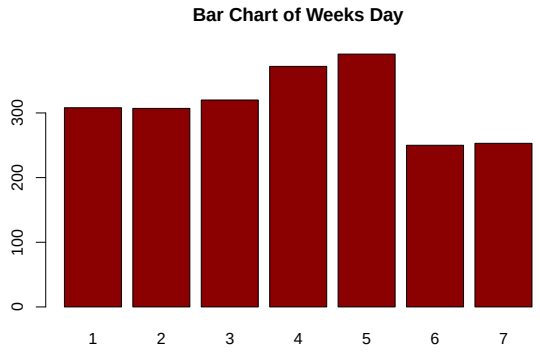
Once the variables are into categorical, we have to add them in the dataframe.

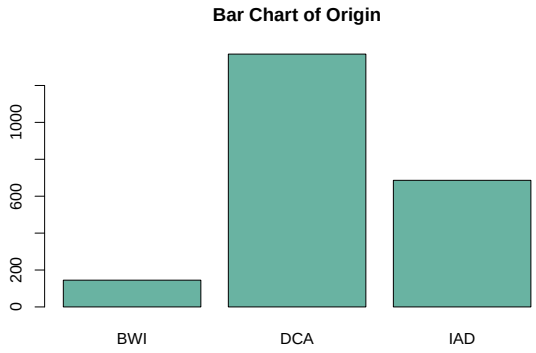
```r
#Merge the columns transformed to the main dataframe and reorganize the dataframe columns
mydf$DAY_WEEK_cat <- mycol_factor$DAY_WEEK

mydf$CRS_DEP_TIMEcat <- mycol_factor$CRS_DEP_TIME

mydf$GROUP_NAMEcat <- mycol_factor$groupname

#Dropping the extra columns. Notice that we are also dropping CRS_DEP_TIME (in numerical
# and categorical) because we already have our slots needed to make the prediction.
mydf <- mydf[, -c(1, 2, 7, 9)]

# Change the Flight Status valuable in a binary.
mydf$`Flight Status`[mydf$`Flight Status` == "delayed" ] <- 1
mydf$`Flight Status`[mydf$`Flight Status` == "ontime" ] <- 0

# To prove the all the columns are in categorical
str(mydf)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   2201 obs. of  6 variables:
 $ ORIGIN       : chr  "BWI" "DCA" "IAD" "IAD" ...
 $ DEST         : chr  "JFK" "JFK" "LGA" "LGA" ...
 $ CARRIER      : chr  "OH" "DH" "DH" "DH" ...
 $ Flight Status: chr  "0" "0" "0" "0" ...
 $ DAY_WEEK_cat : Factor w/ 7 levels "1","2","3","4",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ GROUP_NAMEcat: Factor w/ 32 levels "1","2","3","4",..: 18 22 14 23 10 6 14 22 23 31 ...
```

In the data exploration and preparation, it is essential to get the frecuency of the data to use the most uniform variable to split the data balanced.

```
# PLot the variables to see their frecuency

barplot(table(mydf$DAY_WEEK_cat),
        main= "Bar Chart of Weeks Day",
        col= c("darkred"))

barplot(table(mydf$ORIGIN),
        main= "Bar Chart of Origin",
        col= c("#69b3a2"))

barplot(table(mydf$DEST),
        main= "Bar Chart of Destionation",
        col= c("darkblue"))

barplot(table(mydf$CARRIER),
        main= "Bar Chart of Airline Company",
        col= c("purple"))

barplot(table(mydf$`Flight Status`),
        main= "Bar Chart of Flight Status",
        col= c("darkgreen"))

barplot(table(mydf$GROUP_NAMEcat),
        main= "Bar Chart of Times Groups",
        col= c("turquoise"))
```





These plots allow us to see the frecuency of the data points. Based on that, the DAY_WEEK variable has a more uniform distribution.

Data Split

As mentioned before, the DAY_WEEK variable has a uniform distribution. So, we are going to use this variable to evently split the data, which will be 60% into training set and 40% of validation set.

```
# To get the same random variables
set.seed(123)

#Divide data into training and validation using DAY_WEEK_cat
datasplit <-createDataPartition(mydf$DAY_WEEK_cat, p=0.6, list=FALSE)

#Now, let's to create a dataframe with 60% for training sets and 40% validation sets.
train <-mydf [datasplit, ]
valid  <-mydf [-datasplit, ]

#Now, let's do summary function to get some descritive statistics and see how
# well it has been distributed.
summary(train$DAY_WEEK_cat)
```

```
  1   2   3   4   5   6   7
185 185 192 224 235 150 152
```

```
summary(valid$DAY_WEEK_cat)
```

```
  1   2   3   4   5   6   7
123 122 128 148 156 100 101
```

As we can prove, our training and validation sets are stable and well balanced.

Naive Bayes Model

Use the Naive Bayes model to predict whether the flight is delayed or not.

```
# Build a naïve Bayes classifier. It is a common way to specify the model.
nb_model <- naiveBayes(`Flight Status`~ ORIGIN+DEST+CARRIER+DAY_WEEK_cat+GROUP_NAMEcat,
                       data = train)

#To show the model
nb_model
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1
0.8095238 0.1904762

Conditional probabilities:
   ORIGIN
Y          BWI        DCA        IAD
  0 0.06069094 0.64519141 0.29411765
  1 0.09126984 0.53174603 0.37698413

   DEST
Y         EWR       JFK       LGA
  0 0.2969188 0.1699346 0.5331466
  1 0.3373016 0.2142857 0.4484127

   CARRIER
Y           CO         DH         DL         MQ         OH         RU
  0 0.03641457 0.23436041 0.18767507 0.12324930 0.01493931 0.19327731
  1 0.05952381 0.31746032 0.11904762 0.18650794 0.01190476 0.18650794
   CARRIER
Y           UA         US
  0 0.01493931 0.19514472
  1 0.01587302 0.10317460

   DAY_WEEK_cat
Y            1          2          3          4          5          6
```

```
  0 0.13165266 0.14005602 0.15032680 0.17553688 0.17460317 0.12698413
  1 0.17460317 0.13888889 0.12301587 0.14285714 0.19047619 0.05555556
   DAY_WEEK_cat
Y            7
  0 0.10084034
  1 0.17460317

   GROUP_NAMEcat
Y             1           2           3           4           5           6
  0 0.011204482 0.052287582 0.040149393 0.020541550 0.016806723 0.058823529
  1 0.007936508 0.035714286 0.035714286 0.015873016 0.015873016 0.043650794
   GROUP_NAMEcat
Y             7           8           9          10          11          12
  0 0.043884220 0.016806723 0.013071895 0.038281979 0.025210084 0.009337068
  1 0.023809524 0.000000000 0.000000000 0.019841270 0.019841270 0.000000000
   GROUP_NAMEcat
Y            13          14          15          16          17          18
  0 0.009337068 0.047619048 0.060690943 0.026143791 0.023342670 0.070961718
  1 0.000000000 0.063492063 0.031746032 0.003968254 0.007936508 0.150793651
   GROUP_NAMEcat
Y            19          20          21          22          23          24
  0 0.034547152 0.022408964 0.035480859 0.056956116 0.086834734 0.018674136
  1 0.047619048 0.019841270 0.039682540 0.035714286 0.107142857 0.023809524
   GROUP_NAMEcat
Y            25          26          27          28          29          30
  0 0.014005602 0.023342670 0.033613445 0.010270775 0.010270775 0.017740430
  1 0.003968254 0.031746032 0.103174603 0.007936508 0.011904762 0.007936508
   GROUP_NAMEcat
Y            31          32
  0 0.051353875 0.000000000
  1 0.083333333 0.000000000
```

Now, use the Naive Bayes model on the validation set to predict whether the flight is delayed or not.

```
# Predict the probability default status of valid dataset.
#It is essential for the ROC curve analysis.
Predicted_Test_Prob <- predict(nb_model, valid, type = "raw")

#show the first few values
head(Predicted_Test_Prob)
```

```
             0         1
[1,] 0.6192312 0.3807688
[2,] 0.7284173 0.2715827
[3,] 0.7627166 0.2372834
[4,] 0.6414421 0.3585579
[5,] 0.7884862 0.2115138
[6,] 0.8990663 0.1009337
```

Here, we are going to perfrom a counts table and proportion table outlining how many and what proportion of flights were delayed and on-time at each of the three airports. It is important to compare the real values with our prediction.

```
# Count Table
table( valid$ORIGIN, valid$`Flight Status`)
```

```
       0   1
  BWI  43  14
  DCA 458  87
  IAD 201  75
```

```
# Proportion Table
prop.table(table( valid$ORIGIN, valid$`Flight Status`))
```

```
              0          1
  BWI 0.04897494 0.01594533
  DCA 0.52164009 0.09908884
  IAD 0.22892938 0.08542141
```

Remember that 0 = on time, and 1 = delayed. In total, we have 702 (79.95%) of flights on time, and 176 (20.05%) flights delayed in our validation set.

Confusion Matrix for the validation data

Before running the confusion matrix, we have to fix the "Predicted_Test_Prob" variable length.

Note: To fix the variable length, we could also use the following code: "Predicted_Test_Class <- predict(nb_model, valid, type ="class")." Then, perfom the confusion matrix.

```
# Copy the dataframe
Pred_temp = Predicted_Test_Prob

# Find the position as TRUE and FALSE for those greater than 50%
x <- (Pred_temp>=0.50)[,1]

# Assign 0 on position TRUE and 1 on position FALSE
Pred_temp[x,1] = 0
Pred_temp[not(x),1] = 1

#Create a results on a vector
result <- Pred_temp[,1]

# Perform the confusion matrix of the classifier
conf_matrix <- CrossTable( x= valid$`Flight Status`, y= result, prop.chisq = FALSE)
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
```

```
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  878


                     | result
valid$`Flight Status` |         0 |         1 | Row Total |
----------------------|-----------|-----------|-----------|
                    0 |       673 |        29 |       702 |
                      |     0.959 |     0.041 |     0.800 |
                      |     0.808 |     0.644 |           |
                      |     0.767 |     0.033 |           |
----------------------|-----------|-----------|-----------|
                    1 |       160 |        16 |       176 |
                      |     0.909 |     0.091 |     0.200 |
                      |     0.192 |     0.356 |           |
                      |     0.182 |     0.018 |           |
----------------------|-----------|-----------|-----------|
         Column Total |       833 |        45 |       878 |
                      |     0.949 |     0.051 |           |
----------------------|-----------|-----------|-----------|
```

The confusion matrix help us to see the true positives and true negatives, but also those data points that has been misclassified.

In our problem, the model is misclassifying 189 flights.

Here, I decided to calcutale the accuracy, recall, precision, specificity for the model

```
#Calcutale the accuracy
accuracy <- (conf_matrix$t[2,2] + conf_matrix$t[1,1])/ sum(conf_matrix$t)
print(accuracy)
```

```
[1] 0.784738
```

```
#Calcutale the recall or sensitivity
recall <- conf_matrix$t[2,2]/ (conf_matrix$t[2,2] + conf_matrix$t[2,1])
print(recall)
```

```
[1] 0.09090909
```

Remember that specificity shows the true positive rate, in our case the delayed flights that have been correctly classified as delayed.

As we can prove in the confusion matrix, the model is only classifying 19 flights as delayed, where in reality our validation set has 176 delayed flights. For this reason, the sensitivity value is too low.

```r
#Calcutale the precision
precision <- conf_matrix$t[2,2]/ (conf_matrix$t[2,2] + conf_matrix$t[1,2])
print(precision)
```

```
[1] 0.3555556
```

```r
#Calcutale the specificity
specificity <- conf_matrix$t[1,1]/ (conf_matrix$t[1,1] + conf_matrix$t[1,2])
print(specificity)
```

```
[1] 0.9586895
```

Remember that the specificity shows the true negative rate. In our example, it will be related to the on-time flights that were correctly classified.

Our specificity is performing very well because out of 702 on-time flights the model is classifying correctly 673.

ROC for the validation data

Here, we will use the second column of the predicted probabilities, which has the probability associate to 'Delayed.'

```r
#ROC
roc(valid$`Flight Status`, Predicted_Test_Prob[,2])
```

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

```
Setting direction: controls < cases
```

```

Call:
roc.default(response = valid$`Flight Status`, predictor = Predicted_Test_Prob[,     2])

Data: Predicted_Test_Prob[, 2] in 702 controls (valid$`Flight Status` 0) < 176 cases (valid$`Flight Sta
Area under the curve: 0.6722
```

The area under the curve (AUC) is the value that helps us to identify how well the model is classifying, in other words, the ability of the model to distinguish between classes and grouping them in the correctly. AUC closer to 1 is better.

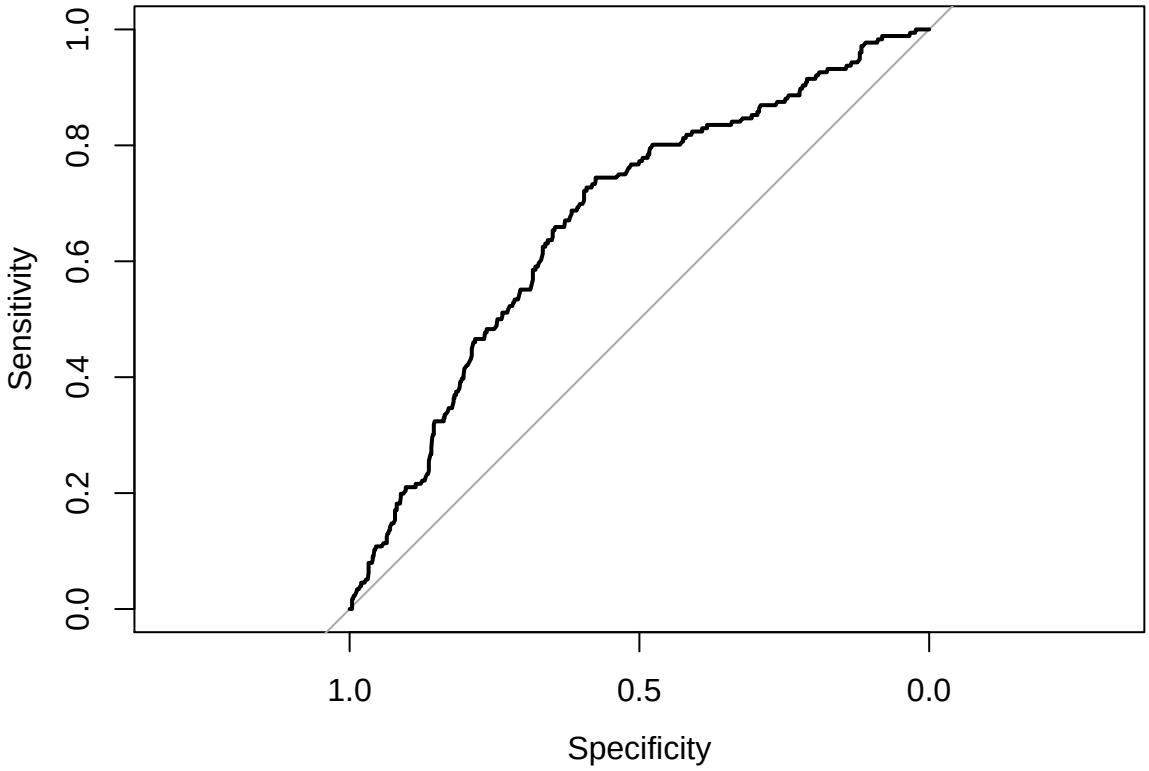
Our model is an AUC of 67.22%.

Now, we are going to plot the ROC to have a better understanding of it.

```r
#Plot the ROC
plot.roc(valid$`Flight Status`,Predicted_Test_Prob[,2])
```

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

```
Setting direction: controls < cases
```



The ROC curve shows the performance of a classification model. As we can see, our model is not performing well.

To enhance its performance, we should add more data to the dataset, and we might also think about using different predictors to increase performance.