Code

Classification

Aarya Patil and Austin Girouard

How Do Linear Models for Classification Work?

Classification is a machine learning method in which a model tries to correctly identify the label of a piece of data. Within classification, you have certain algorithms that are used to train the models by providing training data sets. The two specific algorithms that we will discuss here are: Logistic Regression and Naïve Bayes.

Logistic regression predicts the likelihood of a variable. It is usually used when there are 2 choices of classes for the dependent variable, where 1 represents yes and 0 represents no. This algorithm is fairly easy to train, and works really well on large data. However, it can often suffer from over-fitting, which takes place when the number of observations are greater than the number of features.

Naïve Bayes determines whether a piece of data falls into a certain category. While logistic regression is more binary, either one or the other, Naïve Bayes deals more with the actual class identification of data. It uses training data to study features and characteristics of certain data and applies this knowledge to make predictions about other data. This algorithm doesn't require much training data and is very scalable. However, it faces the 'zero-frequency problem', which essentially means that a probabilty of 0 is assigned to any variable that is unavailable in the training data.

Load the Data

Hide

```
df2 <- read.csv ("heart_disease_health_indicators_BRFSS2015.csv", header = TRUE) # sp
ecifies where to load data from
df2 <- df2[,c(1, 2, 3, 5, 19, 20)] # specifies which columns we want

# Make HeartDiseaseorAttack, HighBP, HighChol, and Sex into factors
df2$HeartDiseaseorAttack <- factor(df2$HeartDiseaseorAttack)
df2$HighBP <- factor(df2$HighBP)
df2$HighChol <- factor(df2$HighChol)
df2$Sex <- factor(df2$Sex)
str(df2) # print data frame structure
```

```
'data.frame':   253680 obs. of  6 variables:
 $ HeartDiseaseorAttack: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 2 1 ...
 $ HighBP              : Factor w/ 2 levels "0","1": 2 1 2 2 2 2 2 2 2 1 ...
 $ HighChol            : Factor w/ 2 levels "0","1": 2 1 2 1 2 2 1 2 2 1 ...
 $ BMI                 : num  40 25 28 27 24 25 30 25 30 24 ...
 $ Sex                 : Factor w/ 2 levels "0","1": 1 1 1 1 1 2 1 1 1 2 ...
 $ Age                 : num  9 7 9 11 11 10 9 11 9 8 ...
```

Sample Training and Testing Data

Hide

```
set.seed(1234)
i <- sample(1:nrow(df2), nrow(df2) * 0.8, replace = FALSE) # split data into 80/20 tr
ain/test
train2 <- df2[i, ] # training data
test2 <- df2[-i, ] # testing data
```

Use 5 R Functions for Data Exploration

1. Create a summary of the data

Hide

```
summary(df2) # creates a summary of the data
```

```
 HeartDiseaseorAttack HighBP     HighChol        BMI         Sex             Age        
 0:229787             0:144851   0:146089   Min.   :12.00   0:141974   Min.   : 1.000  
 1: 23893             1:108829   1:107591   1st Qu.:24.00   1:111706   1st Qu.: 6.000  
                                            Median :27.00              Median : 8.000  
                                            Mean   :28.38              Mean   : 8.032  
                                            3rd Qu.:31.00              3rd Qu.:10.000  
                                            Max.   :98.00              Max.   :13.000  
```

2. Display the number of rows in the data set

Hide

```
nrow(df2) # generates the number of rows
```

```
[1] 253680
```

3. Display the number of missing values in each variable

Hide

```
colSums(is.na(df2)) # returns the number of missing values in each variable
```

```
HeartDiseaseorAttack               HighBP             HighChol                  BMI
Sex                   Age
                   0                    0                    0                    0
0                     0
```

4. Output the outliers in the dataset

Hide

```
# Code for finding the outliers of BMI
quartiles <- quantile(df2$BMI, probs = c(.25, .75), na.rm = FALSE)
IQR <- IQR(df2$BMI)

Lower <- quartiles[1] - 1.5 * IQR
Upper <- quartiles[2] + 1.5 * IQR

df2_outliers <- subset(df2, df2$BMI <= Lower | df2$BMI >= Upper) # saves outliers of
BMI into df2_outliers

# Code for finding the outliers of Age
quartiles <- quantile(df2$Age, probs = c(.25, .75), na.rm = FALSE)
IQR <- IQR(df2$Age)

Lower <- quartiles[1] - 1.5 * IQR
Upper <- quartiles[2] + 1.5 * IQR

df2_outliers <- subset(df2, df2$Age <= Lower | df2$Age >= Upper) # saves outliers of
Age into df2_outliers

# Output the outliers & summaries
cat("BMI Outliers:\n")
```

```
BMI Outliers:
```

Hide

```
str(df2_outliers$BMI) # outputs the outliers
```

```
 num(0)
```

Hide

```
summary(df2_outliers$BMI) # outputs a summary
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.


```

Hide

```
cat("Age Outliers:\n")
```

```
Age Outliers:
```

Hide

```
str(df2_outliers$Age) # outputs the outliers
```

```
 num(0)
```

Hide

```
summary(df2_outliers$Age) # outputs a summary
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.


```

5. Find correlations between columns in the data (Age and BMI)

Hide

```
cor.test(df2$BMI, df2$Age, use = "complete") # finds correlations between Age and BMI
```

```


    Pearson's product-moment correlation

data:  df2$BMI and df2$Age
t = -18.455, df = 253678, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.04050326 -0.03273090
sample estimates:
        cor
-0.03661764
```

Create 2 Informative Graphs

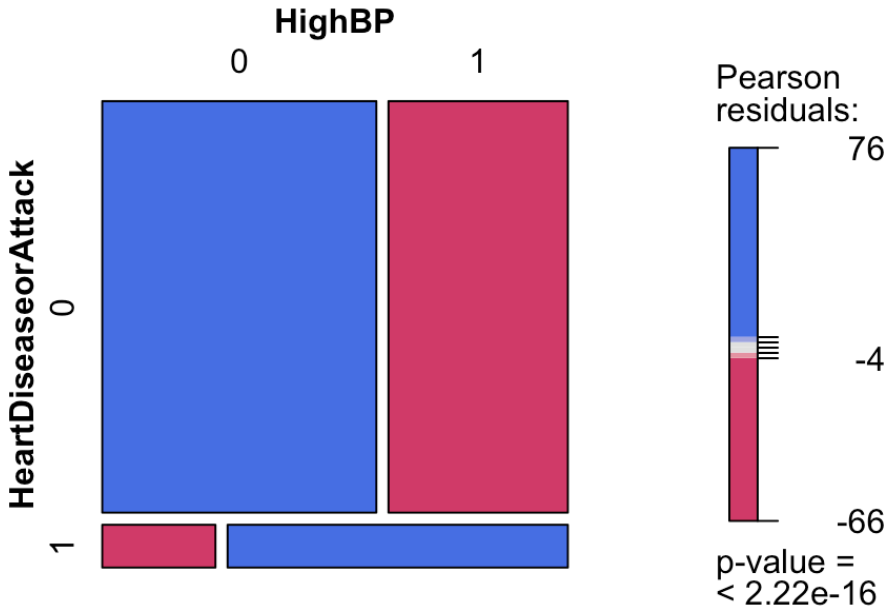
Hide

```
#install.packages('vcd')
library(vcd)
```

```
Loading required package: grid
```

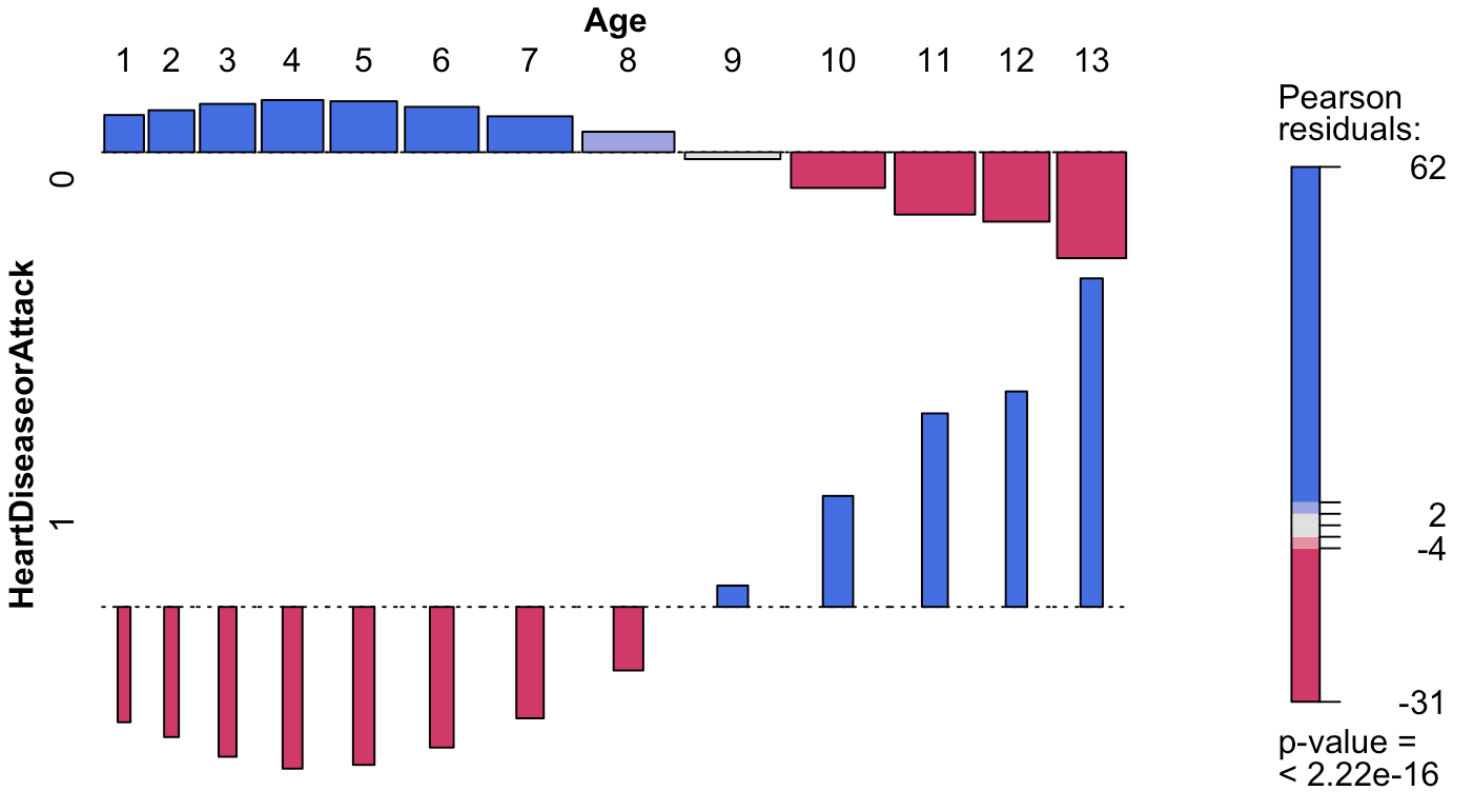
Hide

```
# Graph 1
mosaic(table(df2[, c(1, 2)]), shade = TRUE, legend = TRUE)
```



Hide

```
# Graph 2
assoc(table(df2[, c(1, 6)]), shade = TRUE)
```



Build a Logistic Regression Model

Hide

```
glm1 <- glm(HeartDiseaseorAttack~., data = train2, family = "binomial") # builds logi
stic regression model to predict HeartDiseaseorAttack
summary(glm1) # shows the logistic regression model summary
```

```
Call:
glm(formula = HeartDiseaseorAttack ~ ., family = "binomial",
    data = train2)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5435  -0.4858  -0.2939  -0.1636   3.4452

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.607480   0.056486 -116.97   <2e-16 ***
HighBP1      0.869460   0.018779   46.30   <2e-16 ***
HighChol1    0.767059   0.017451   43.96   <2e-16 ***
BMI          0.021313   0.001209   17.62   <2e-16 ***
Sex1         0.643580   0.016239   39.63   <2e-16 ***
Age          0.270413   0.003673   73.61   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 126719  on 202943  degrees of freedom
Residual deviance: 107111  on 202938  degrees of freedom
AIC: 107123

Number of Fisher Scoring iterations: 6
```

The first stats displayed in the summary are the deviance residuals, which quantify a given point’s contribution to the overall likelihood.

The coefficients section shows us the change in log odds of y for a 1-unit change in x, our predictor.

Finally, the statistics section includes a null deviance and residual deviance score. Null deviance measures a lack of fit of the model when considering only the intercept, while the residual deviance considers the entire model. Here, we want to see a residual deviance that is much lower than the null deviance, which is shown (ND = 126719; RD = 107111).

Build a Naive Bayes Model

Hide

```
library(e1071)
nb1 <- naiveBayes(HeartDiseaseorAttack~., data = train2) # builds naive bayes model t
o predict HeartDiseaseorAttack
nb1
```

```
Naive Bayes Classifier for Discrete Predictors

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

A-priori probabilities:
Y
         0          1
0.90577696 0.09422304

Conditional probabilities:
   HighBP
Y           0         1
  0 0.6047046 0.3952954
  1 0.2489279 0.7510721

   HighChol
Y           0         1
  0 0.6048297 0.3951703
  1 0.2997071 0.7002929

   BMI
Y       [,1]     [,2]
  0 28.26578 6.577879
  1 29.48081 6.793538

   Sex
Y           0         1
  0 0.5741206 0.4258794
  1 0.4280933 0.5719067

   Age
Y        [,1]     [,2]
  0  7.815534 3.046907
  1 10.123366 2.221625
```

A-priori tells us the probability of HeartDiseaseorAttack being a 0 or a 1 prior to executing the Naive Bayes algorithm. We can see that approximately 90.6% of data points record HeartDiseaseorAttack as negative and 9.4% recording positive.

Second, we see the conditional probabilities. This shows the probability of each combination of true/false values between the predictor and each other variable OR the mean and standard deviation for continuous data (non-factors). For example, we can see that the probability of a data point recording true for HeartDiseaseorAttack AND true for HighBP is 75.1%. We can also see that the mean age of a data point with HeartDiseaseorAttack marked true is 10.12 years.

Logistic Regression vs Naive Bayes Results

Logistic Regression Statistics

Hide

```
# TP = 2,"1"
# TN = 1, "0"
# FN = 2, "0"
# FP = 1, "1"
# Calculate logistic regression for data set
probs <- predict(glm1, newdata = test2, type = "response")
pred <- ifelse(probs > 0.5, 1, 0)
t1 <- table(pred, test2$HeartDiseaseorAttack) # store table results into t1

# Print out the confusion matrix
library(caret)
```

```
Loading required package: ggplot2
Learn more about the underlying theory at https://ggplot2-book.org/
Loading required package: lattice
```

Hide

```
confusionMatrix(as.factor(pred), reference = test2$HeartDiseaseorAttack)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 45959  4770
         1     6     1

               Accuracy : 0.9059
                 95% CI : (0.9033, 0.9084)
    No Information Rate : 0.906
    P-Value [Acc > NIR] : 0.5341

                  Kappa : 1e-04

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9998695
            Specificity : 0.0002096
         Pos Pred Value : 0.9059709
         Neg Pred Value : 0.1428571
             Prevalence : 0.9059642
         Detection Rate : 0.9058459
   Detection Prevalence : 0.9998620
      Balanced Accuracy : 0.5000395

       'Positive' Class : 0
```

Hide

```
print(t1)
```

```
pred     0     1
   0 45959  4770
   1     6     1
```

Hide

```
# Calculate and print out accuracy, sensitivity, and specificity
acc <- mean(pred == test2$HeartDiseaseorAttack) # calculate accuracy
sens <- t1[2,"1"] / (t1[2,"1"] + t1[2,"0"]) # calculate sensitivity
spec <- t1[1,"0"] / (t1[1,"0"] + t1[1,"1"]) # calculate specificity

print(paste("accuracy = ", acc))
```

```
[1] "accuracy =  0.905865657521287"
```

Hide

```
print(paste("sensitivity = ", sens))
```

```
[1] "sensitivity =  0.142857142857143"
```

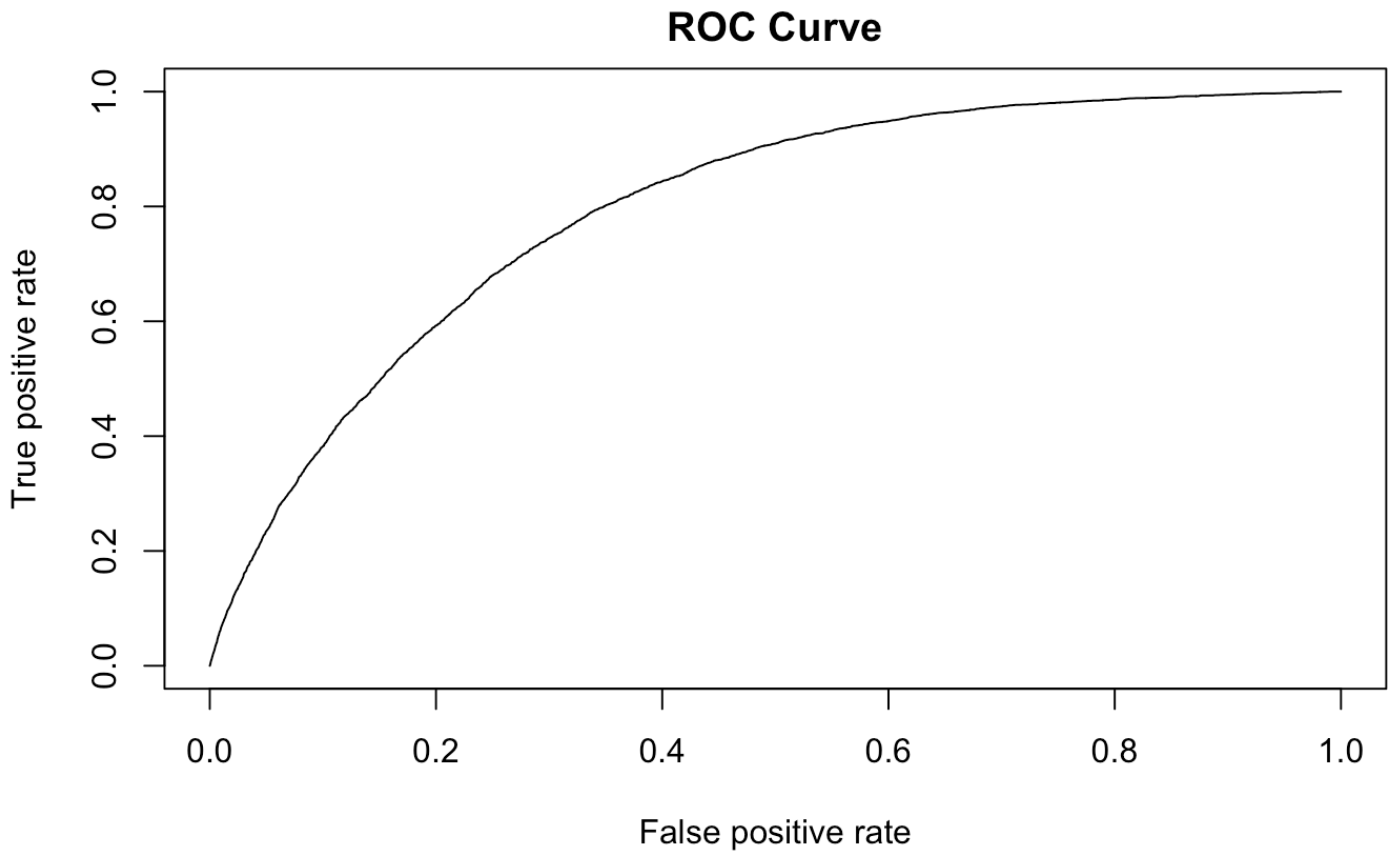
Hide

```
print(paste("specificity = ", spec))
```

```
[1] "specificity =  0.905970943641704"
```

Hide

```
# Plot the ROC curve
library(ROCR)
p <- predict(glm1, newdata = test2, type = "response")
pr <- prediction(p, test2$HeartDiseaseorAttack)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf, main = "ROC Curve")
```



Hide

```
# Calculate AUC
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
print(paste("AUC = ", auc)) # display AUC
```

```
[1] "AUC =  0.793240662754458"
```

Naive Bayes Statistics

Hide

```
# Calculate Naive Bayes for data set
p1 <- predict(nb1, newdata=test2, type = "class")
t2 <- table(p1, test2$HeartDiseaseorAttack)


# Print the confusion matrix
library(caret)
confusionMatrix(as.factor(p1), reference = test2$HeartDiseaseorAttack)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 44852  4160
         1  1113   611

               Accuracy : 0.8961
                 95% CI : (0.8934, 0.8987)
    No Information Rate : 0.906
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1455

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9758
            Specificity : 0.1281
         Pos Pred Value : 0.9151
         Neg Pred Value : 0.3544
             Prevalence : 0.9060
         Detection Rate : 0.8840
   Detection Prevalence : 0.9660
      Balanced Accuracy : 0.5519

       'Positive' Class : 0
```

Hide

```
print(t2)
```

```
p1      0     1
  0 44852  4160
  1  1113   611
```

Hide

```
# Calculate and print accuracy, sensitivity, and specificity
acc <- mean(p1 == test2$HeartDiseaseorAttack) # calculate accuracy
sens <- t2[2,"1"] / (t2[2,"1"] + t2[2,"0"]) # calculate sensitivity
spec <- t2[1,"0"] / (t2[1,"0"] + t2[1,"1"]) # calculate specificity

print(paste("accuracy = ", acc))
```

```
[1] "accuracy =  0.896069851781772"
```

Hide

```
print(paste("sensitivity = ", sens))
```

```
[1] "sensitivity =  0.354408352668213"
```

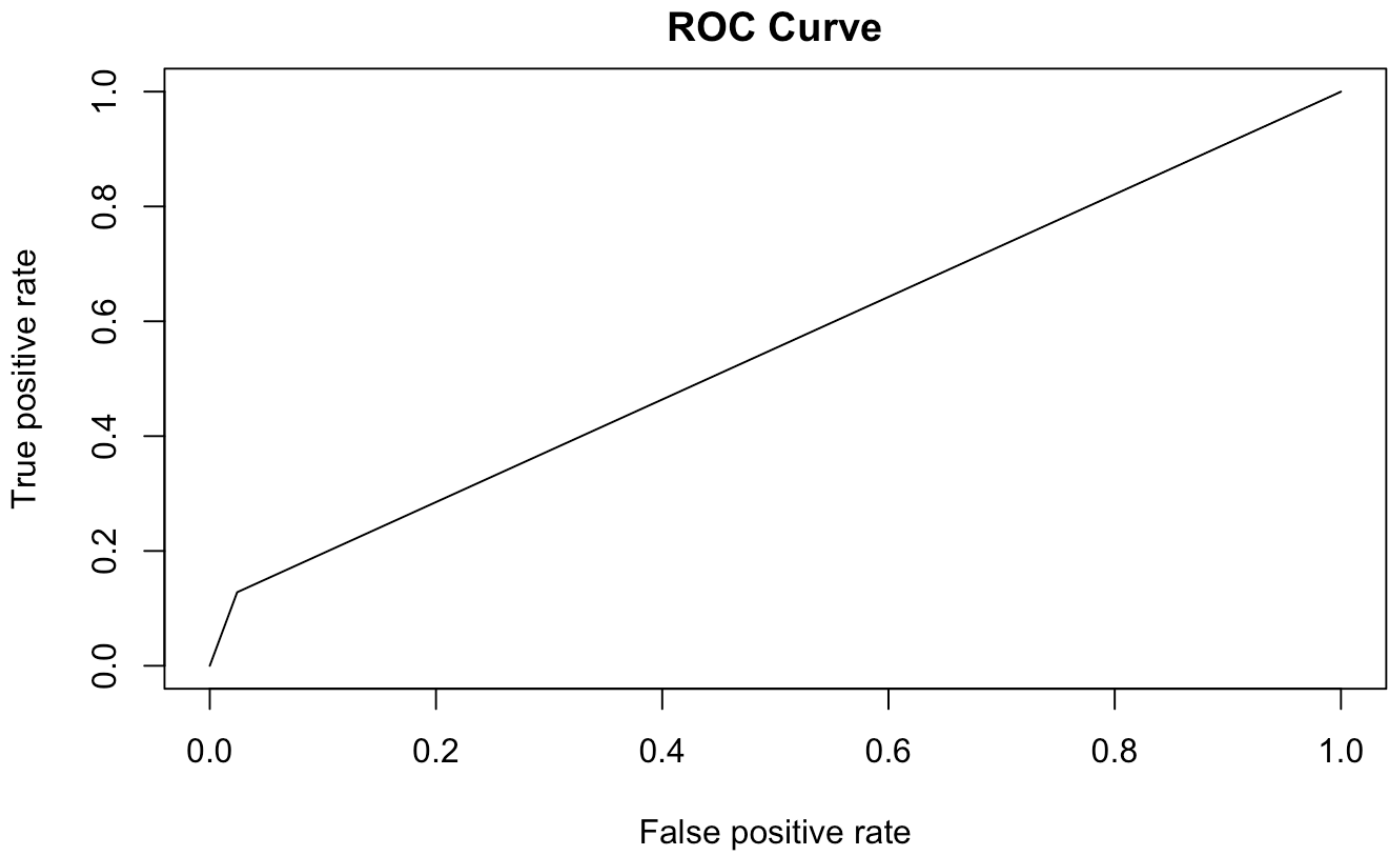
Hide

```
print(paste("specificity = ", spec))
```

```
[1] "specificity =  0.91512282706276"
```

Hide

```
# Plot the ROC curve
library(ROCR)
p1 <- predict(nb1, newdata = test2, type = "class")
pr2 <- prediction(as.numeric(p1), as.numeric(test2$HeartDiseaseorAttack))
prf <- performance(pr2, measure = "tpr", x.measure = "fpr")
plot(prf, main = "ROC Curve")
```



Hide

```
# Calculate AUC
auc2 <- performance(pr2, measure = "auc")
auc2 <- auc2@y.values[[1]]
print(paste("AUC = ", auc2)) # display AUC
```

```
[1] "AUC =  0.551925659584016"
```

Result Comparison and Analysis

Looking at the tables for Naive Bayes and Logistic Regression, we can see that the main difference is in the number of false negatives and true positives. Logistic Regression produced 6 results as FN and 1 as TP in these two sections out of about 50000 data points. Naive Bayes produced 1113 FNs and 611 FPs.

Accuracy: we can see that the accuracy for both models is relatively the same. This is because while the number of TPs recorded for Naive Bayes increased, the number of True Negatives went down proportionally. Both models predict values with a high accuracy.

Sensitivity: we see a .2 variation in sensitivity between the two models. This change is the result of the proportionally large variation between TPs measured between the two models. Because the Naive Bayes measured more TPs overall, it's sensitivity is higher. These measurements show that both models do not perform well when predicting True Positives.

Specificity: the specificity calculation between the two models is relatively similar due to the minimal difference in measured TNs and FPs. This shows that both models perform similarly well in measuring true negatives.

AUC (Logistic Regression): Ideally, we would like to see a curve that spikes up very quickly and plateaus off quickly. Here, we can see that our curve doesn't rise as steeply as we would hope to see in the beginning, but the shape is still very curved and our AUC is .79. As mentioned before, this doesn't necessarily mean our predictor is good, it just means that it performs well when distinguishing between classes in our data.

AUC (Naïve Bayes): Here we can see that the ROC curve for Naive Bayes is much worse than that of the logistic Regression. It is nearly linear across the plot with an AUC of .55. This doesn't mean our predictor model is poor, but it does mean that our model has difficulty distinguishing between different classes in our data.

MCC (Logistic Regression): The MCC for Logistic Regression takes into account the differences in class distribution to calculate a more accurate 'accuracy' score. We can see that the accuracy score does not differ much in comparison to our original calculation.

MCC (Naïve Bayes): Similarly, the MCC accuracy does not differ much from the original accuracy score.

Strengths and Weaknesses of Naïve Bayes and Logistic Regression

Naïve Bayes is very strong algorithm to use when working with smaller data. The training time is often low as it doesn't require a lot of training data to estimate the test data. However, this algorithm faces the 'zero-frequency problem'. This means that if variable is not available in the training data set, the algorithm assigns a zero probability to the category in the test data set. Therefore, the estimations made by the Naïve Bayes algorithm can sometimes be inaccurate.

Logistic Regression on the other hand, is an algorithm that can handle larger data very well and is one of the easiest machine learning algorithms to understand. It makes no assumptions about distributions of classes which makes the results much more accurate, and even with a complicated linear problem and not a whole lot of data it still produces useful predictions. However, overfitting is a very real issue when it comes to Logistic Regression. This basically means that when a data set with lots of features and little training data is provided, the algorithm overstates the accuracy of the prediction on those statistics. That is a case when you should avoid using this algorithm.

Benefits, Drawbacks, and Indicators of Each Classification Metric Used

Accuracy

Accuracy is essentially the ratio of correct/accurate prediction over total number of predictions. It shows how often the algorithm predicted correctly. This metric is usually very useful when the target class within the data we are provided is well balanced. Imbalanced data can often lead to incorrect accuracy metrics, such as 99% accuracy, which can make someone feel that their model is performing well when it's not.

Sensitivity and Specificity

Sensitivity, or True Positive Rate, is the measure of how many positive instances a model can correctly identify. This is a very important metric as it is essential to making accurate predictions. Specificity, or False Positive Rate, is the measure of how many true negatives a model can correctly identify. This metric is also very important for making making accurate predictions. Sensitivity and specificity are actually inversely proportional to each other. One of the drawbacks of sensitivity is that is more likely to be affected by imbalances in data, so it can alter the performance of a model with inaccurate statistics.

Kappa

Kappa is a metric that is used in machine learning to measure the agreement between two evaluators or raters. It is actually a very useful metric to have when dealing with imbalanced data as it takes any imbalance in class distribution into account when calculating its values. However, one of the drawbacks of this metric is that the full range of values (-1, 1) is not reachable. Even though Kappa takes imbalance into account when calculating the statistics, an imbalanced data distribution will make higher Kappa values very hard to reach. You would require a near perfect distribution of data to get the higher Kappa value, which is not very common. This metric can also be written off as "not helpful" since it doesn't give much information about prediction accuracy, which is essential to all machine learning models.

ROC Curves and AUC

ROC, or Receiver Operator Characteristic, is a probability curve that plots the TPR (True Positive Rate) against the FPR (False Positive Rate). The AUC, or Area Under the Curve, represents the capability of distinguishing between classes. The value can range from 0 to 1, with 1 being the ideal case since a higher AUC value is better. While ROC AUC is a very reliable metric, it does not reflect the underlying probability values predicted by the classifier. The reliability of the metric can also falter when it comes to smaller sample sizes.

MCC

MCC, or Matthew's Correlation Coefficient, is a single value used to summarize the confusion matrix. This is a very valuable metric as it gives as overall "rating" for the model using all of the prediction and analysis statistics. The MCC value will only be high (close to 1) if the model is performing well in all of the considered areas. This is very widely considered to be the best metric for classification. The only time that the MCC is not preferred is when low precision and recall are unknown or unable to be quantified, but other than that it is the perfect metric to accurately understand the performance of a model.