Survival of Breast Cancer Surgery Patients

Srividhya Perumal

8/12/2020

BINARY CLASSIFICATION OF SURVIVAL OF BREAST CANCER SURGERY PATIENTS

INTRODUCTION

The haberman's survival dataset (https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival) from the UCI machine learning repository was used for this project. The haberman dataset was curated and split into two datasets for training and validation purposes. Our model classifies patients as alive or dead using binary classification. We are using the gradient boosting machine (GBM) classification model. The GBM was trained using the training dataset and the accuracy was tested using the validation dataset. The GBM classification model received an accuracy value of 0.82.

Download, install and load packages necessary for our model.

```
# Download, install and load the packages required for constructing our model
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(gbm)) install.packages("gbm", repos = "http://cran.us.r-project.org")
```

```
## Warning: package 'gbm' was built under R version 4.0.2
```

```
library("caret")
library("e1071")
library("gbm")

# set.seed() so our results are reproducible
set.seed(1)
```

Set.seed() to 1 so that our random variables and results would be reproducible.

DATA EXPLORATION

```
# Load and assign data that we downloaded to variable patients
patients <- read.csv("haberman-data.txt", header = FALSE)
# View structure of haberman dataset
str(patients)
```

```
## 'data.frame':    306 obs. of  4 variables:
##  $ V1: int  30 30 30 31 31 33 33 34 34 34 ...
##  $ V2: int  64 62 65 59 65 58 60 59 66 58 ...
##  $ V3: int  1 3 0 2 4 10 0 0 9 30 ...
##  $ V4: int  1 1 1 1 1 1 1 2 2 1 ...
```

By using the structure command to view our dataset we see that there are no column headers. From the UCI machine learning repository we see information about the columns:

1. Age of patient at time of operation (numerical)
2. Patient's year of operation (year - 1900, numerical)
3. Number of positive axillary nodes detected (numerical)
4. Survival status (class attribute)

We will add the column headers to the respective columns below.

```
# Add respective headings to columns
names(patients)<- c('Operation Age', 'Operation Year', '+ve Axillary Nodes', 'Survival')
# Assign 0 as dead and 1 as alive
patients$Survival <- patients$Survival - 1
# View structure of haberman dataset
str(patients)
```

```
## 'data.frame':    306 obs. of  4 variables:
##  $ Operation Age     : int  30 30 30 31 31 33 33 34 34 34 ...
##  $ Operation Year    : int  64 62 65 59 65 58 60 59 66 58 ...
##  $ +ve Axillary Nodes: int  1 3 0 2 4 10 0 0 9 30 ...
##  $ Survival          : num  0 0 0 0 0 0 0 1 1 0 ...
```

The survival column indicates whether a patient survived or died. 1 indicates that the patient died.

Each row in the datasets is a record of one patient. From the Survival column, we see that we can use binary classification for our prediction model.

DATA CLEANING

```
# Check for NAs in the 4 columns
sapply(patients, function(x) sum(is.na(x)))
```

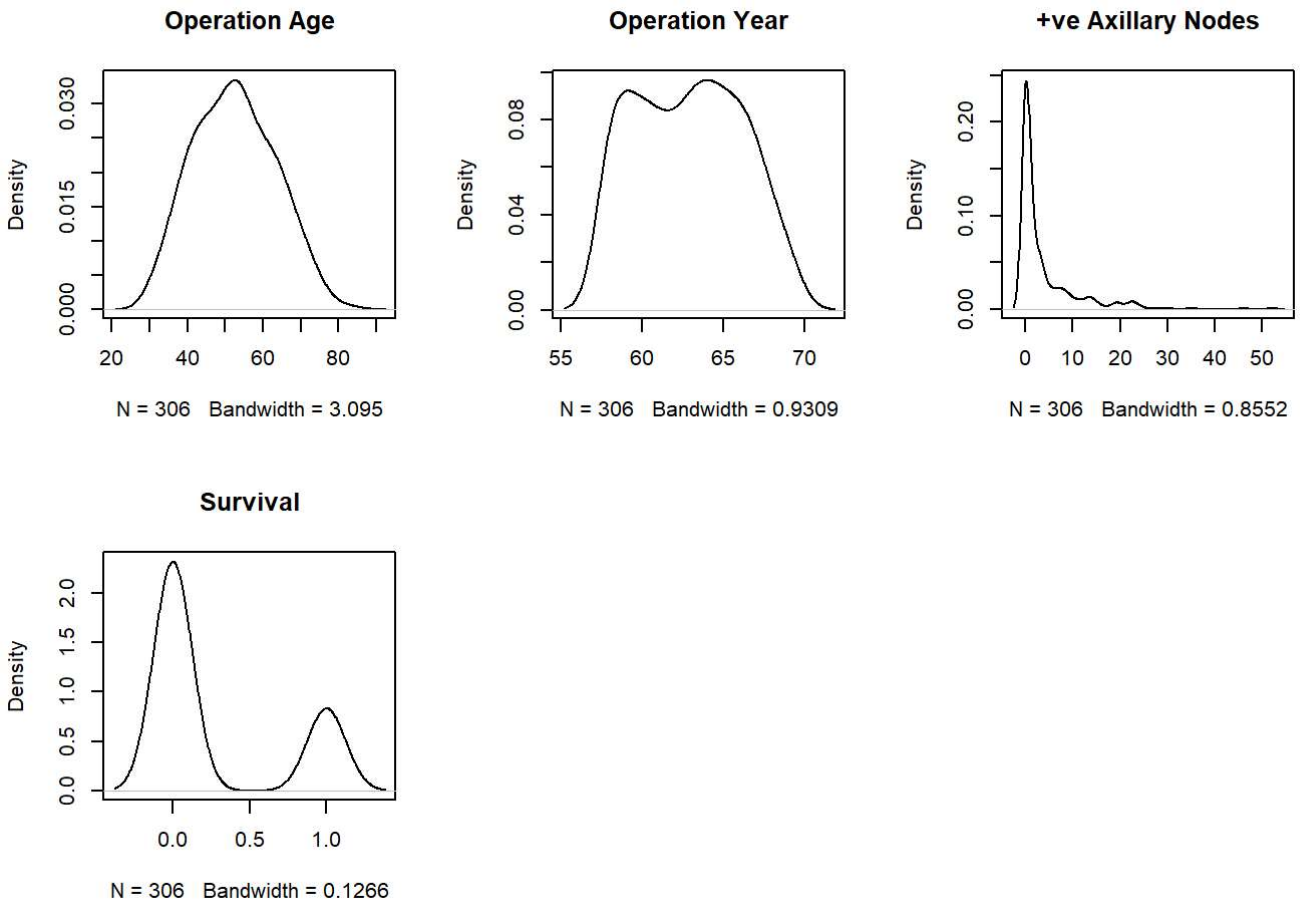
```
##      Operation Age     Operation Year +ve Axillary Nodes           Survival
##                  0                  0                  0                  0
```

There are no NAs and hence the data requires no further cleaning.

DATA VISUALIZATION

To visualize our data we will create density plots by column values.

```
# Create density plots by attribute
par(mfrow=c(2,3))
for(i in c(1:4)) {
  plot(density(patients[,i]), main=names(patients)[i])
}
```



MODELING

```
# Check proportion of dead and alive patients
# Here 1 represents a dead patient
prop.table(table(patients$Survival))
```

```
##
##         0         1
## 0.7352941 0.2647059
```

26.5 % of patients in the dataset have died. This distinction is important because if the proportion was smaller than 15% then it would be considered as a rare event and the data would have to be modeled differently.

```
# Convert values in Survival column to factors
patients$Survival <- ifelse(patients$Survival==1,'yes','no')
patients$Survival <- as.factor(patients$Survival)
```

To use the classification mode of GBM we change the class of the class column to factor.

The haberman dataset is split into training and testing sets to create and evaluate our prediction system.

```
# Create training and testing datasets
test <- createDataPartition(patients$Survival, p = .3, times = 1, list = FALSE)
train_set <- patients[-test,]
test_set <- patients[test,]
```

We control resampling of our data to reduce the time by using the trainControl function of the caret package. This function divides our dataset into a ‘number’ of times to find the best parameters for our model.

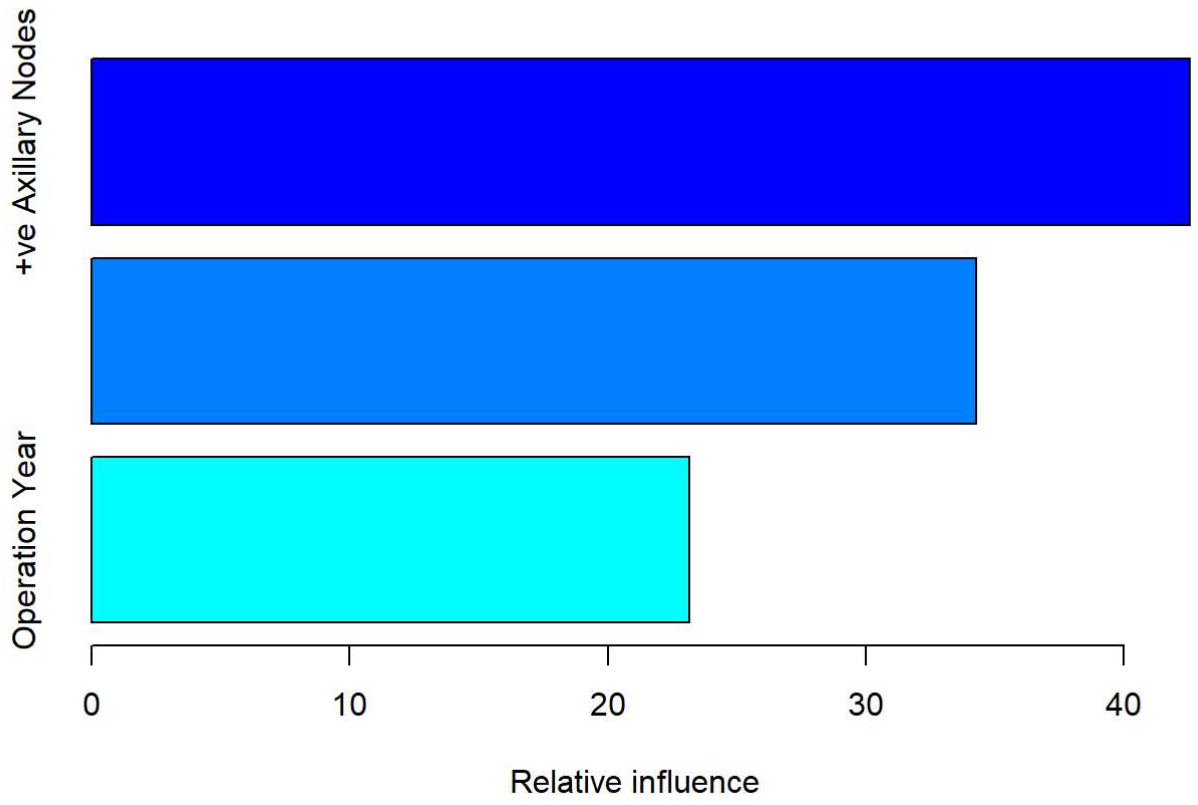
```
# Control resampling of data
control <- trainControl(method='cv', number=2, returnResamp='none', summaryFunction = twoClassSu
mmary, classProbs = TRUE)
# Train model
model <- train(patients[,names(patients)[names(patients) != 'Survival']], patients[,'Survival'],
method='gbm', trControl=control,  metric = "ROC", preProc = c("center", "scale"))
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1377             nan     0.1000    0.0090
##      2        1.1221             nan     0.1000    0.0072
##      3        1.1060             nan     0.1000    0.0038
##      4        1.1066             nan     0.1000   -0.0055
##      5        1.0983             nan     0.1000    0.0023
##      6        1.0816             nan     0.1000   -0.0002
##      7        1.0702             nan     0.1000    0.0021
##      8        1.0631             nan     0.1000   -0.0007
##      9        1.0560             nan     0.1000   -0.0029
##     10        1.0519             nan     0.1000    0.0008
##     20        1.0164             nan     0.1000   -0.0022
##     40        0.9754             nan     0.1000   -0.0008
##     60        0.9419             nan     0.1000   -0.0026
##     80        0.9245             nan     0.1000   -0.0014
##    100        0.9111             nan     0.1000   -0.0021
##    120        0.8952             nan     0.1000   -0.0023
##    140        0.8857             nan     0.1000   -0.0023
##    150        0.8806             nan     0.1000   -0.0012
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1446             nan     0.1000    0.0064
##      2        1.1243             nan     0.1000    0.0058
##      3        1.1075             nan     0.1000    0.0082
##      4        1.0910             nan     0.1000    0.0021
##      5        1.0772             nan     0.1000    0.0028
##      6        1.0592             nan     0.1000    0.0000
##      7        1.0515             nan     0.1000   -0.0045
##      8        1.0404             nan     0.1000    0.0046
##      9        1.0340             nan     0.1000   -0.0004
##     10        1.0265             nan     0.1000    0.0013
##     20        0.9671             nan     0.1000   -0.0055
##     40        0.8903             nan     0.1000   -0.0050
##     60        0.8443             nan     0.1000   -0.0062
##     80        0.8114             nan     0.1000   -0.0023
##    100        0.7827             nan     0.1000   -0.0038
##    120        0.7596             nan     0.1000   -0.0036
##    140        0.7421             nan     0.1000   -0.0069
##    150        0.7281             nan     0.1000   -0.0058
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1284             nan     0.1000    0.0137
##      2        1.1028             nan     0.1000    0.0072
##      3        1.0788             nan     0.1000    0.0075
##      4        1.0581             nan     0.1000    0.0036
##      5        1.0409             nan     0.1000   -0.0031
##      6        1.0259             nan     0.1000    0.0009
##      7        1.0154             nan     0.1000    0.0018
##      8        1.0092             nan     0.1000   -0.0024
##      9        0.9953             nan     0.1000    0.0042
##     10        0.9835             nan     0.1000    0.0022
##     20        0.9212             nan     0.1000   -0.0059
##     40        0.8286             nan     0.1000   -0.0081
```

```
##     60        0.7814             nan     0.1000   -0.0059
##     80        0.7490             nan     0.1000   -0.0062
##    100        0.7165             nan     0.1000   -0.0061
##    120        0.6852             nan     0.1000   -0.0041
##    140        0.6540             nan     0.1000   -0.0040
##    150        0.6394             nan     0.1000   -0.0021
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1280             nan     0.1000    0.0116
##      2        1.1066             nan     0.1000    0.0059
##      3        1.0982             nan     0.1000   -0.0022
##      4        1.0848             nan     0.1000    0.0050
##      5        1.0707             nan     0.1000    0.0001
##      6        1.0612             nan     0.1000    0.0015
##      7        1.0567             nan     0.1000   -0.0002
##      8        1.0486             nan     0.1000    0.0027
##      9        1.0376             nan     0.1000    0.0033
##     10        1.0309             nan     0.1000   -0.0022
##     20        0.9942             nan     0.1000    0.0012
##     40        0.9672             nan     0.1000   -0.0124
##     60        0.9439             nan     0.1000   -0.0053
##     80        0.9266             nan     0.1000   -0.0039
##    100        0.9100             nan     0.1000   -0.0041
##    120        0.9036             nan     0.1000   -0.0091
##    140        0.8938             nan     0.1000   -0.0049
##    150        0.8838             nan     0.1000   -0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1304             nan     0.1000    0.0044
##      2        1.0994             nan     0.1000    0.0102
##      3        1.0780             nan     0.1000    0.0023
##      4        1.0486             nan     0.1000    0.0086
##      5        1.0434             nan     0.1000   -0.0028
##      6        1.0295             nan     0.1000    0.0001
##      7        1.0305             nan     0.1000   -0.0066
##      8        1.0187             nan     0.1000   -0.0052
##      9        1.0104             nan     0.1000    0.0027
##     10        1.0015             nan     0.1000    0.0021
##     20        0.9442             nan     0.1000    0.0006
##     40        0.8737             nan     0.1000   -0.0033
##     60        0.8341             nan     0.1000   -0.0024
##     80        0.8003             nan     0.1000   -0.0034
##    100        0.7760             nan     0.1000   -0.0057
##    120        0.7459             nan     0.1000   -0.0106
##    140        0.7321             nan     0.1000   -0.0076
##    150        0.7183             nan     0.1000   -0.0039
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1264             nan     0.1000    0.0112
##      2        1.0823             nan     0.1000    0.0096
##      3        1.0645             nan     0.1000   -0.0049
##      4        1.0464             nan     0.1000    0.0069
##      5        1.0207             nan     0.1000    0.0077
##      6        1.0050             nan     0.1000   -0.0011
```

```
##      7        0.9861             nan     0.1000    0.0027
##      8        0.9725             nan     0.1000    0.0014
##      9        0.9654             nan     0.1000    0.0007
##     10        0.9532             nan     0.1000   -0.0004
##     20        0.8789             nan     0.1000   -0.0046
##     40        0.8137             nan     0.1000   -0.0098
##     60        0.7542             nan     0.1000   -0.0012
##     80        0.7142             nan     0.1000   -0.0073
##    100        0.6937             nan     0.1000   -0.0093
##    120        0.6677             nan     0.1000   -0.0066
##    140        0.6459             nan     0.1000   -0.0041
##    150        0.6392             nan     0.1000   -0.0035
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1316             nan     0.1000    0.0092
##      2        1.1084             nan     0.1000    0.0068
##      3        1.0959             nan     0.1000    0.0047
##      4        1.0864             nan     0.1000    0.0017
##      5        1.0750             nan     0.1000    0.0055
##      6        1.0634             nan     0.1000    0.0025
##      7        1.0565             nan     0.1000    0.0005
##      8        1.0472             nan     0.1000    0.0048
##      9        1.0416             nan     0.1000   -0.0006
##     10        1.0389             nan     0.1000   -0.0027
##     20        0.9902             nan     0.1000   -0.0023
##     40        0.9343             nan     0.1000   -0.0040
##     50        0.9151             nan     0.1000   -0.0029
```

```
summary(model)
```




```
##                                   var  rel.inf
## +ve Axillary Nodes +ve Axillary Nodes 42.54877
## Operation Age           Operation Age 34.27667
## Operation Year         Operation Year 23.17456
```

```
model
```

```
## Stochastic Gradient Boosting
##
## 306 samples
##   3 predictor
##   2 classes: 'no', 'yes'
##
## Pre-processing: centered (3), scaled (3)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 153, 153
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.7108966  0.9198404  0.2829268
##   1                  100      0.7071668  0.9199194  0.3082317
##   1                  150      0.6941841  0.9155341  0.2594512
##   2                   50      0.7359192  0.9155341  0.2969512
##   2                  100      0.7285285  0.8665455  0.3945122
##   2                  150      0.7156144  0.8755136  0.3948171
##   3                   50      0.7262333  0.8976375  0.3448171
##   3                  100      0.7171809  0.8576169  0.3454268
##   3                  150      0.7136499  0.8620812  0.3454268
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 10.
```

PREDICTIONS AND RESULTS

Since our model is now ready, we make predictions and evaluate them by comparing them to the true values.

```
# Create predictions with model
predictions <- predict(object=model, test_set[,names(patients)[names(patients) != 'Survival']],
 type='raw')
# Compute accuracy
print(postResample(pred=predictions, obs=as.factor(test_set[,'Survival'])))
```

```
##  Accuracy     Kappa
## 0.8172043 0.4752738
```

The accuracy for our classification model is 0.82 > 0.7 and is accurate.

CONCLUSION

A breast cancer surgery patient survival classification system was created to classify surival of patients. The classfication system was validated by comparing its predictions to the true values and the accuracy was determined. The accuracy value of our classfication system is 0.82.