Lab_10T

36-290 – Statistical Research Methodology

Week 10 Tuesday – Fall 2021

Preliminaries

Goal

This is going to be a bit more of an open-ended lab, since there is only really so much one can say about Naive Bayes itself. The goal will be for you to practice learning classifiers for pulsar detection.

Pulsars are neutron stars that spin rapidly (up to many times per second!) and give off "pulses" of electromagnetic radiation (i.e., light). The pulses occur because the physics of the pulsar environment leads to pulsar emission being "beamed"…unlike the Sun, which gives off essentially the exact same amount of light in all directions, a pulsar may give off light in certain preferential directions. (Think of a lighthouse… that will give you the intuitive picture.) If we are lucky enough to sit in a location that the beam passes over every time a pulsar rotates, then we see less light, then more light, then less light, etc.

To back up: a neutron star is a stellar remnant of size roughly 10 miles across. A neutron star is generally formed during a supernova at the end of a massive star's lifetime. (Stars that are eight solar masses or more generally explode and leave behind neutron stars; smaller stars tend to slough off their gas over time and leave behind white dwarfs, which are Earth-sized.) A neutron star is called a neutron star because it is pretty much literally a bag of neutrons (no electrons, no protons, just neutrons) that holds itself up via a mechanism called "degeneracy pressure." Without degeneracy pressure, the neutron star would simply collapse and become a black hole. (And with enough mass, one can induce this sort of collapse–hence the existence of black holes.)

At the end of the day: we scan the skies, we get data, we need to figure out which pulsar candidates are pulsars. A binary classification problem. Onwards…

Data

Today we will work with the pulsar dataset, this time with 1639 "NO"'s and 1639 "YES"'s.

```
rm(list=ls())
file.path = "http://www.stat.cmu.edu/~pfreeman/pulsar.Rdata"
load(url(file.path))
rm(file.path)
set.seed(406)
w.0 = which(response$X9==0)
w.1 = which(response$X9==1)
s = sample(length(w.0),length(w.1))
predictors = predictors[c(w.0[s],w.1),]
response = as.character(response$X9)[c(w.0[s],w.1)]
predictors = scale(predictors)
predictors = data.frame(predictors)
cat("Number of predictor variables: ",ncol(predictors),"\n")
```

```
## Number of predictor variables:  8
```

```
cat("Sample size:                   ",nrow(predictors),"\n")
```

```
## Sample size:                   3278
```

```
response = factor(response,labels=c("NO","YES"))
```

The eight predictors are summary statistics that describe the distribution of brightness measurements of a pulsar candidate (mean, standard deviation, skewness, kurtosis) as well as the distribution of "dispersion measure" readings (also mean, standard deviation, skewness, kurtosis).

Questions

Question 1

Again, this is an open-ended lab. How well can you do trying to identify pulsars? Split the data into training and test subsets, and learn classifiers for naive Bayes, logistic regression, trees, and random forest. (Because we are *not* reducing the dataset size, you will not want to code SVM or KNN. Note that because we are not working with SVM or KNN, we do not standardize the data.) The classes are unbalanced (90% non-pulsars, 10% pulsars), so you should target AUC as your metric by which to compare the results of different classifiers. The bulk of your work will be going back to old labs (and coding Naive Bayes using the notes as a guide), and seeing if bit by bit you can get the AUC for each classifier. Declare a winner: which classifier that you try gives you the best AUC? Overplot all four ROC curves onto a single plot. Take the model with the best AUC value and use Youden's J statistic to determine an optimal class-separation threshold, then use that threshold to make class predictions. Last, construct a confusion matrix and output the misclassification rate.

```
set.seed(100)
fraction=.7
sp = sample(nrow(predictors), round(fraction*nrow(predictors)))
pred.train = predictors[sp ,]
pred.test = predictors[-sp ,]

resp.train = response[sp]
resp.test = response[-sp]

suppressMessages(library(pROC))

#naive Bayes
library(e1071)
nb.out = naiveBayes(resp.train~.,data=pred.train)
nb.pred = predict(nb.out,newdata=pred.test,type="class")

nb.prob = predict(nb.out,newdata=pred.test,type="raw")[,2]

table(nb.pred,resp.test)
```

```
##        resp.test
## nb.pred  NO YES
##     NO  455  64
##     YES  27 437
```

```
mean(nb.pred!=resp.test)
```

```
## [1] 0.09257375
```

```
#MCR:0.09257375


#logistic regression
log.out = glm(resp.train~.,data=pred.train,family=binomial)
log.prob = predict(log.out,newdata=pred.test,type="response")
log.pred = ifelse(log.prob>0.5,"YES","NO")
table(log.pred,resp.test)
```

```
##         resp.test
## log.pred  NO YES
##      NO  475  40
##      YES   7 461
```

```
mean(log.pred!=resp.test)
```

```
## [1] 0.04781282
```

```
#MCR:0.04781282
contrasts(response)
```

```
##     YES
## NO    0
## YES   1
```

```
#No = 0, Yes = 1


#trees

library(rpart)
rpart.out =rpart(resp.train~.,data=pred.train)
class.prob= predict(rpart.out,newdata=pred.test, type="prob")[,2] #prob of class 1
class.pred = ifelse(class.prob>0.5,"YES","NO")
mean(class.pred!=resp.test)
```

```
## [1] 0.08138352
```

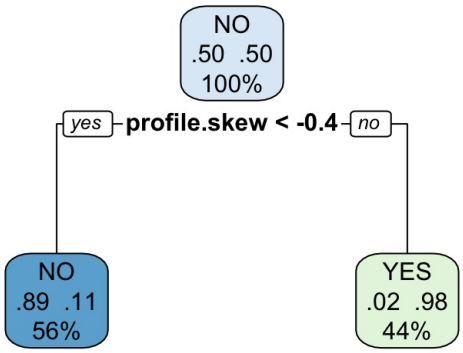
```
#MCR:0.08138352

table(class.pred,resp.test)
```

```
##           resp.test
## class.pred  NO YES
##        NO  471  69
##        YES  11 432
```

```
library(rpart.plot)
rpart.plot(rpart.out,extra=104)


#random forest
suppressMessages(library(tidyverse)) ; suppressMessages(library(randomForest))
```




```
set.seed(100)
rf.out = randomForest(resp.train~.,data=pred.train,importance=TRUE)
resp.prob.rf = predict(rf.out,newdata=pred.test,type="prob")[,2]
resp.pred.rf = ifelse(resp.prob.rf>0.5, "YES","NO")

mean(resp.pred.rf!=resp.test)
```

```
## [1] 0.04781282
```

```
#MCR:0.04781282



roc.glm = roc(resp.test, log.prob)
```

```
## Setting levels: control = NO, case = YES
```

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

```
plot(roc.glm,col="red")

roc.tree = roc(resp.test, class.prob)
```

```
## Setting levels: control = NO, case = YES
## Setting direction: controls < cases
```

```
plot(roc.tree,col="blue", add= TRUE)

roc.rf = roc(resp.test, resp.prob.rf)
```

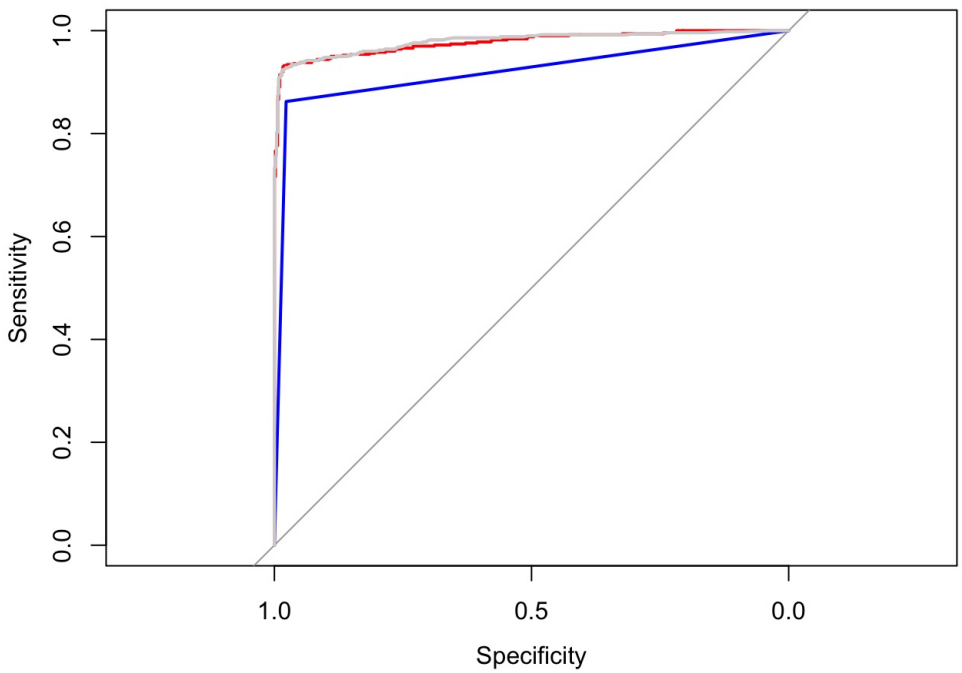
```
## Setting levels: control = NO, case = YES
## Setting direction: controls < cases
```

```
plot(roc.rf, col="pink", add=TRUE)

roc.nb = roc(resp.test, nb.prob)
```

```
## Setting levels: control = NO, case = YES
## Setting direction: controls < cases
```

```
plot(roc.rf, col="lightgrey", add=TRUE)
```




```
cat("AUC for random forest:",roc.rf$auc, "\n")
```

```
## AUC for random forest: 0.9794498
```

```
cat("AUC for naive bayes:",roc.nb$auc, "\n")
```

```
## AUC for naive bayes: 0.9574875
```

```
cat("AUC for tree:",roc.tree$auc, "\n")
```

```
## AUC for tree: 0.9197269
```

```
cat("AUC for logistic regression:",roc.glm$auc, "\n")
```

```
## AUC for logistic regression: 0.9774559
```

```
J = roc.rf$sensitivities + roc.rf$specificities - 1

w = which.max(J)
cat("Optimum threshold for random forest: ",roc.rf$thresholds[w],"\n")
```

```
## Optimum threshold for random forest:  0.543
```

```
#0.543
#data does not have balanced classes but optimal threshold seems like it's close to .5, is this normal?


set.seed(100)
rf.out = randomForest(resp.train~.,data=pred.train,importance=TRUE)
new.resp.prob.rf = predict(rf.out,newdata=pred.test,type="response",cutoff=c(1-0.543,0.543))
table(resp.prob.rf,resp.test)
```

```
##             resp.test
## resp.prob.rf  NO YES
##        0      19   0
##        0.002  24   1
##        0.004  16   0
##        0.006  18   1
##        0.008  18   0
##        0.01   16   0
##        0.012  14   1
##        0.014   9   0
##        0.016   9   0
##        0.018  10   1
##        0.02    9   0
##        0.022   6   0
##        0.024  10   0
##        0.026   8   0
##        0.028   4   0
##        0.03    5   0
##        0.032  11   0
##        0.034   3   0
##        0.036   5   0
##        0.038   6   0
##        0.04    7   0
##        0.042   5   0
##        0.044   5   1
##        0.046   7   0
##        0.048   4   1
##        0.05    7   0
##        0.052   8   0
##        0.054   3   0
##        0.056   4   0
##        0.058   0   1
##        0.06    6   0
##        0.062   7   0
##        0.064   4   0
##        0.066   8   0
##        0.068   1   0
##        0.07    5   0
##        0.072   2   0
##        0.074   2   0
##        0.076   2   0
##        0.078   2   0
##        0.08    5   0
##        0.082   6   1
##        0.084   4   1
##        0.086   4   0
##        0.088   4   0
##        0.09    4   0
##        0.092   3   1
##        0.096   0   1
##        0.098   3   0
##        0.102   1   0
##        0.104   0   1
##        0.106   2   0
##        0.108   1   0
##        0.11    4   0
##        0.112   1   0
##        0.114   4   1
##        0.116   2   1
```

```
##        0.118   2   0
##        0.12    2   1
##        0.124   4   1
##        0.126   1   0
##        0.128   3   2
##        0.13    4   0
##        0.132   1   0
##        0.134   4   0
##        0.136   2   1
##        0.138   2   0
##        0.142   2   0
##        0.144   2   1
##        0.146   1   0
##        0.148   1   0
##        0.15    3   0
##        0.152   2   0
##        0.154   4   0
##        0.156   1   0
##        0.158   1   0
##        0.16    0   1
##        0.164   3   0
##        0.168   1   1
##        0.17    1   0
##        0.172   0   1
##        0.174   1   0
##        0.178   1   0
##        0.18    1   0
##        0.182   1   0
##        0.186   1   0
##        0.188   1   1
##        0.19    3   0
##        0.194   0   1
##        0.202   2   0
##        0.206   2   0
##        0.208   4   0
##        0.212   2   0
##        0.222   1   0
##        0.224   0   1
##        0.23    1   0
##        0.232   1   0
##        0.234   1   0
##        0.236   2   0
##        0.242   1   0
##        0.248   2   0
##        0.25    1   0
##        0.256   1   0
##        0.262   1   0
##        0.264   0   1
##        0.27    1   0
##        0.28    1   0
##        0.282   0   1
##        0.284   2   0
##        0.286   1   0
##        0.29    0   1
##        0.292   1   0
##        0.296   1   0
##        0.298   2   0
##        0.3     1   0
##        0.302   1   0
##        0.306   1   0
##        0.314   1   0
##        0.318   1   0
##        0.324   1   0
##        0.326   1   0
##        0.332   1   0
##        0.334   1   1
##        0.354   1   0
##        0.358   0   1
##        0.366   1   0
##        0.368   1   0
##        0.38    1   0
##        0.386   1   0
##        0.394   0   1
##        0.408   1   0
##        0.414   1   0
##        0.416   1   0
##        0.426   0   1
##        0.43    1   0
##        0.434   1   0
```

```
##        0.438   1   0
##        0.46    1   0
##        0.462   1   0
##        0.464   0   1
##        0.468   0   1
##        0.47    1   0
##        0.476   1   0
##        0.488   1   1
##        0.49    1   0
##        0.492   1   0
##        0.498   1   1
##        0.514   1   1
##        0.52    1   0
##        0.526   0   1
##        0.538   1   0
##        0.548   0   1
##        0.554   0   1
##        0.566   0   1
##        0.572   0   1
##        0.574   1   0
##        0.582   0   1
##        0.594   0   1
##        0.604   1   0
##        0.616   0   1
##        0.624   1   0
##        0.634   0   1
##        0.646   0   1
##        0.652   0   2
##        0.654   0   1
##        0.68    0   1
##        0.682   0   1
##        0.698   0   1
##        0.712   0   1
##        0.73    0   1
##        0.738   0   1
##        0.74    0   1
##        0.742   0   1
##        0.744   0   1
##        0.748   0   1
##        0.754   1   2
##        0.764   0   1
##        0.77    0   1
##        0.778   0   1
##        0.78    0   1
##        0.802   0   1
##        0.808   0   2
##        0.812   0   2
##        0.814   0   1
##        0.816   0   1
##        0.822   0   1
##        0.824   0   1
##        0.826   0   1
##        0.83    0   1
##        0.84    0   1
##        0.85    0   2
##        0.852   0   1
##        0.858   0   2
##        0.862   0   1
##        0.864   0   2
##        0.868   0   1
##        0.87    0   1
##        0.872   0   2
##        0.878   0   1
##        0.88    0   1
##        0.884   0   1
##        0.89    0   1
##        0.892   0   1
##        0.898   0   1
##        0.9     0   1
##        0.904   0   1
##        0.908   1   0
##        0.91    0   2
##        0.912   0   1
##        0.914   0   1
##        0.916   0   2
##        0.918   0   1
##        0.922   0   1
##        0.924   0   2
##        0.926   0   1
```

```
##        0.928   0   1
##        0.93    0   2
##        0.934   0   1
##        0.936   0   1
##        0.938   0   2
##        0.94    0   3
##        0.942   0   1
##        0.944   1   2
##        0.946   0   1
##        0.95    0   1
##        0.952   0   4
##        0.96    0   3
##        0.962   0   1
##        0.964   1   1
##        0.966   0   3
##        0.968   0   3
##        0.97    0   4
##        0.972   0   1
##        0.974   0   2
##        0.976   0   2
##        0.978   0   1
##        0.982   0   3
##        0.984   0   2
##        0.986   0   6
##        0.988   0   5
##        0.99    0   5
##        0.992   0   9
##        0.994   0  11
##        0.996   0  11
##        0.998   0  31
##        1       0 268
```

```
mean(new.resp.prob.rf!=resp.test)
```

```
## [1] 0.04679552
```

```
#MCR:0.04679552
```

which classifier that you try gives you the best AUC: random forest
use Youden's J statistic to determine an optimal class-separation threshold, then use that threshold to make clas
s predictions.

Last, construct a confusion matrix and output the misclassification rate.

```
roc.glm = roc(resp.test, log.pred)
plot(roc.glm, col="blue")

roc.random = roc(test$response, resp.pred.rf)
plot(roc.random,add=TRUE, col="red")

roc.random$auc
roc.glm$auc
```